Java-Zertifizierung

Prüfungsvorbereitung für den
Sun Certified Java Programmer (SCJP):
Java 2 Platform 1.4 und Platform 5.0

Inhalt

I. Vorwort ... 1

II. Einleitung ... 3

A. Über dieses Buch ... 3
1. Über den Inhalt ... 3
2. Wie arbeite ich mit diesem Buch? ... 3
3. Wie und wo melde ich mich für die Prüfung an? ... 3
4. Wie sieht die Prüfung aus? ... 4
5. Wo kann ich Informationen zu Prüfungsfragen finden? ... 4
6. Wo kann ich zusätzliche Informationen über die genauen Themen der Prüfung finden? ... 4

B. Wie arbeite ich mit Java und Eclipse? ... 5
1. Was ist das Java Development Kit (JDK) und Eclipse? ... 5
2. Installation des Java Development Kits ... 5
3. Installation von Eclipse ... 8

III. Grundlagen der Programmierung in Java ... 9

A. Das erste Java-Programm in Eclipse ... 9

B. Die main()-Methode ... 11
1. Begriffsbestimmung ... 11
2. Fehlermöglichkeiten in der main()-Methoden ... 11
3. Übungen ... 13
4. Lösungen ... 15

C. Namen in Java: Bezeichner ... 16
1. Begriffserläuterung ... 16
2. Übungen ... 17
3. Lösungen ... 19

D. Schlüsselwörter ... 20
1. Begriffserläuterung ... 20
2. Übungen ... 21
3. Lösungen ... 22

E. Zahlen und Buchstaben in Java: Primitive Datentypen und Variablendeklaration ... 23
1. Primitiver Datentyp ... 23
2. Variablendeklarationen und Initialisierungen von Variablen ... 24
3. Literale ... 24
4. Übungen ... 28
5. Lösungen ... 33

F. Zuweisungen und Typumwandlung ... 34
1. Zuweisungen ... 34
2. Implizite und explizite Typanpassung ... 34
3. Übungen: ... 36
4. Lösungen ... 38

IV. Grundlagen der Objektorientierung ... 39

A. Klassen und Objekte ... 39
1. Allgemeines ... 39
2. Erstellung einer Klasse ... 40
3. Bestandteile von Klassen ... 40
4. Instanziierung von Objekten ... 43
5. Geltungsbereich von Variablen ... 44
6. Packages ... 45
7. Übungen ... 47
8. Lösungen ... 48

B. Vererbung ... 49
1. Begriffserläuterung ... 49
2. Is-a-Relationship und Has-a-Relationship ... 50
3. Übungen ... 51
4. Lösungen ... 51

C. Interfaces und abstrakte Klassen ... 52
1. Interfaces (Schnittstellen) ... 52
2. Abstrakte Klassen ... 54

3.Übungen ... 55
4.Lösungen ... 58
D. Modifier ... 59
1.Allgemeines ... 59
2.Zusammenfassende Aufstellung der Modifier und ihrer Verwendung ... 60
3.Übungen ... 61
4.Lösungen ... 63
E. Überladen und Überschreiben von Methoden ... 64
1.Überladen von Methoden ... 64
2.Übungen ... 66
3.Lösungen ... 67
4.Überschreiben von Methoden ... 68
5.Übungen ... 71
6.Lösungen ... 74
F. Objektreferenz und Konstruktoraufruf mit this und super ... 75
1.Objektreferenz mit this und super ... 75
2.Konstruktoraufruf mit this und super ... 77
3.Übungen ... 84
4.Lösungen ... 87
G. Subtyping und Casting von Objekten ... 88
1.Subtyping ... 88
2.Casten von Objekten ... 90
3.Operator: instanceof ... 92
4.Übungen ... 94
5.Lösungen ... 101

V. Operatoren ... 103

A. Arithmetische Operatoren ... 103
1.Die Operatoren für Subtraktion und Addition ... 103
2.Die Operatoren für Multiplikation und Division ... 103
3.Der Operator Modulo % ... 104
4.Bearbeitungsreihenfolge der Operatoren ... 106
5.Zusammengesetzte Zuweisungsoperatoren: *=, /=, %=, +=, - = ... 107
6.Übungen ... 109
7.Lösungen ... 114
B. Die Boolean Logischen Operatoren ... 115
1.Der Logische Nicht-Operator ... 115
2.Der Logische Und-Operator ... 115
3.Der Logische Oder-Operator ... 116
4.Logischer XOR-Operator ... 116
5.Bearbeitungsreihenfolge bei den Logischen Operatoren ... 117
6.Übungen ... 118
7.Lösungen ... 122
8.Short-Circuit-Auswertung ... 123
9.Übungen ... 124
10.Lösungen ... 126
C. Die Operatoren Inkrement ++ und Dekrement -- ... 127
1.Übungen ... 129
2.Lösungen ... 133
D. Relationale Operatoren ... 134
E. Integer Bitweise Operatoren ... 134
1.Exkurs: Zahlensysteme ... 134
2.Übungen ... 138
3.Lösungen ... 141
4.Der bitweise Not-Operator ~ ... 144
5.Der bitweise And-Operator ... 144
6.Der bitweise Und-Operator ... 145
7.Der bitweise XOR-Operator ... 145
8.Übungen ... 146
9.Lösungen ... 149
F. Der Shift-Operator (Verschiebeoperator) ... 151
1.Der Links-Shift: << ... 151
2.Der Rechts-Shift-Operator: >> ... 153

3.Vorzeichenloser Rechts-Shift: >>> ... 155
4.Übungen ... 156
5.Lösungen ... 161

VI. Kontrollstrukturen ... 163

A. Selektionen ... 163
1.Die if-Struktur ... 163
2.Die if-else-Struktur ... 164
3.Die else-if-Struktur ... 164
4.Übungen ... 167
5.Lösungen ... 173
6.Die switch-Struktur ... 174
7.Übungen ... 176
8.Lösungen ... 179
B. Iterationen ... 180
1.Die while-Schleife ... 180
2.Die do-while-Schleife ... 181
3.Die for-Schleife ... 182
4.Die Sprungbefehle break und continue und markierte Anweisungen ... 184
5.Übungen ... 186
6.Lösungen ... 193

VII. Arrays ... 195

1.Eindimensionale Arrays ... 195
2.Mehrdimensionale Arrays ... 198
3.Übungen ... 201
4.Lösungen ... 206

VIII. Wichtige Standardklassen (Fundamental Classes) ... 209

A. Die Klasse Object ... 209
B. Die Wrapper-Klassen (Hüllklassen) ... 209
1.Umwandeln von primitiven Datentypen in Objekte der Wrapper-Klassen ... 209
2.Besonderheiten bei der Wrapper-Klasse Boolean ... 212
3.Umwandeln eines Strings in einen Wrapper mit Hilfe von valueOf(String s) ... 213
4.Umwandeln eines Wrappers in einen primitiven Datentyp: xxxValue() ... 214
5.Umwandeln eines Strings in einen primitiven Datentyp: parseXXX(String s) ... 215
6.Übungen ... 216
7.Lösungen ... 226
C. Die Klasse Math ... 228
1.Die Minimum- und die Maximum-Methoden ... 228
2.Die abs-Methode ... 230
3.Auf- und Abrunden ... 230
4.Die Methode random() ... 231
5.Die Methode sqrt(double d) ... 232
6.Die Methoden sin(double d), cos(double d), tan(double d) ... 232
7.Übungen ... 233
8.Lösungen ... 243
D. Die Klasse String ... 246
1.Die Methode concat() ... 247
2.Die Methode String replace(char oldChar(), char newChar()) ... 247
3.Die Methode int indexOf() und lastIndexOf() ... 248
4.Die Methode String trim() ... 249
5.Die Methode substring() ... 249
6.Die Methode charAt() ... 250
7.Umwandeln von primitiven Datentypen und Objekten in einen String ... 250
8.Übungen ... 251
9.Lösungen ... 258
E. Die Klasse StringBuffer ... 260
1.Konstruktoren der Klasse StringBuffer ... 260
2.Länge und Kapazität eines StringBuffers ... 260
3.Umwandeln eines StringBuffers in einen String ... 261
4.Die Methode append() ... 261
5.Die Methode insert() ... 262

6.Die Methode delete() ... 263
7.Die Methode reverse() ... 263
8.Übungen ... 264
9.Lösungen ... 268
F. Stringkonkatenation ... 270
1.Verketten von Objekten der String-Klasse mit Objekten der String-Klasse ... 270
2.Verketten von Objekten der String-Klasse mit primitiven Datentypen ... 270
3.Verketten von Objekten der String-Klasse mit Objekten der StringBuffer-Klasse ... 271
4.Übungen ... 273
5.Lösungen ... 276
G. Vergleich von Objekten und primitiven Datentypen ... 277
1.Vergleich von Strings ... 277
2.Vergleich von Objekten der Klasse StringBuffer ... 278
3.Vergleich von Objekten der Klasse String und StringBuffer ... 279
4.Vergleich von Objekten der Wrapper-Klassen ... 279
5.Vergleich von primitiven Datentypen ... 281
6.Vergleich von primitiven Datentypen mit Objekten der Wrapper-Klasse ... 281
7.Übungen ... 282
8.Lösungen ... 290

IX. Exceptions ... 293

1.Begriffserläuterung ... 293
2.Checked und Unchecked Exceptions ... 294
3.Exception Handling ... 298
4.Der throw-Ausdruck ... 301
5.Die throws-Klausel ... 303
6.Übungen ... 304
7.Lösungen ... 315

X. Assertions ... 319

1.Begriffserläuterung ... 319
2.Kompilieren von Klassen mit Assertions ... 321
3.Ein- und Ausschalten von Assertions für Nicht-Systemdateien ... 322
4.Ein- und Ausschalten von Assertions für Systemdateien ... 322
5.Regeln für die Verwendung von Assertions ... 323
6.Übungen ... 325
7.Lösungen ... 333

XI. Threads ... 337

1.Begriffserläuterung ... 337
2.Erstellung eines Threads ... 337
3.Der Main-Thread und die Daemon-Threads ... 340
4.Synchronisation ... 341
5.Die unterschiedlichen Thread-Stadien und die dazugehörigen Methoden ... 344
6.Übungen ... 349
7.Lösungen ... 356

XII. Garbage Collection ... 359

1.Begriffserläuterung ... 359
2.Freigabe von Objekten zur Garbage Collection ... 359
3.Object Finalization ... 365
4.Die Methode void gc() ... 366
5.Übungen ... 368
6.Lösungen ... 373

XIII. Das Collections-Framework ... 375

A. Begriffsbestimmung ... 375
1.Übersicht Interface Collection ... 375
2.Übersicht Interface Map ... 376
B. Das Interface Collection ... 376
1.Das Interface Set ... 376
2.Das Interface List ... 379
C. Das Interface Map ... 381

1.Hashtable ... 381
2.HashMap und LinkedHashMap ... 383
3.TreeMap ... 383
D. Übersicht nach der internen Ordnung der Collections ... 383
E. Das Comparator- und Comparable- Interface ... 383
1.Das Comparable Interface ... 384
2.Das Comparator Interface ... 386
F. Übungen ... 388
G. Lösungen ... 400
H. Die hashCode()-Methode ... 403
1.Begriffsbestimmung ... 403
2.Übungen ... 405
3.Lösungen ... 407

XIV. Innere Klassen ... 409

A. Statische und nicht-statische innere Klassen und Interfaces ... 409
1.Begriffserläuterung ... 409
2.Instanziierung von inneren Klassen ... 409
3.Zugelassene Modifier für innere Klassen ... 413
4.Zugriff auf Bestandteile der äußeren Klassen durch die innere Klasse ... 413
5.Zugriff auf Variablen der inneren und äußeren Klasse mit this ... 414
6.Zugelassene Deklarationen innerhalb von inneren Klassen ... 416
7.Zusammenfassende Aufstellung: Innere Klassen ... 416
8.Übungen ... 417
9.Lösungen ... 427
B. Lokale Klassen ... 430
1.Begriffserläuterung ... 430
2.Instanziierung im statischen und nicht-statischen Kontext ... 430
3.Zugelassene Modifier für lokale Klassen ... 430
4.Zugriff auf Bestandteile der äußeren Klasse durch die lokale Klasse ... 430
5.Zugriff auf Bestandteile des Blocks ... 432
6.Zugelassene Deklarationen innerhalb von lokale Klassen ... 432
7.Zusammenfassende Aufstellung: Lokale Klassen ... 433
8.Übungen ... 434
9.Lösungen ... 439
C. Anonyme Klassen ... 441
1.Begriffserläuterung ... 441
2.Erstellen von anonymen Klassen ... 441
3.Zugriff auf Bestandteile der äußeren Klasse durch die anonyme Klasse ... 447
4.Übungen ... 448
5.Lösungen ... 453

XV. Neuerungen in Java 2 Plattform 5.0 ... 455

A. Kontrollstrukturen ... 455
1.Die erweiterte for-Schleife ... 455
2.Übungen ... 458
3.Lösungen ... 459
B. Autoboxing ... 460
1.Begriffsdefinition ... 460
2.Autoboxing und Operatoren ... 460
3.Probleme und Besonderheiten ... 461
4.Übungen ... 470
5.Lösungen ... 476
C. Methoden und Konstruktoren mit variabler Argumentanzahl (Varargs) ... 478
1.Begriffsbestimmung ... 478
2.Auslesen einer variablen Argumentliste ... 479
3.Autoboxing und Varargs ... 480
4.Übungen ... 481
5.Lösungen ... 483
D. Generics im Collections-Framework ... 484
1.Generics bei Listen ... 484
2.Generics bei Sets und Maps ... 486
3.Der Einsatz von Wildcards ... 487

4.Generische Interfaces, Klassen und Methoden 491
5.Das Collections-Framework und Arrays 492
6.Das Interface Queue 495
7.Übungen 496
8.Lösungen 507
E. Enums 510
1.Begriffsbestimmung 510
2.Auslesen von Werten eines Enums 511
3.Methoden, Variablen und Konstruktoren in einem Enum 512
4.Enums und die switch-Struktur 515
5.Die Methode valueOf() 516
6.Der statische Import 517
7.Übungen 519
8.Lösungen 525
F. Bearbeiten von Text 526
1.Die Klasse StringBuilder 526
2.Die Klasse Scanner und die Verwendung von regulären Ausdrücken 526
3.Die Klassen Pattern und Matcher 530
4.Die Methode split() der Klasse String 533
5.Übungen 534
6.Lösungen 538
G. Formatieren von Text, Zahlen und Datum 539
1.Die Klasse Locale 539
2.Formatieren von Text und Zahlen 540
3.Das Datumsformat 545
4.Zahlen- und Währungsformate 548
5.Übungen 551
6.Lösungen 559
H. Ein- und Ausgabe über Datenströme: java.io 561
1.Datenströme 561
2.Die Klasse File 561
3.Ein- und Ausgabe über Zeichenströme: 562
4.Ein- und Ausgabe über Byteströme 566
5.Übungen 571
6.Lösungen 579

Index 581

I. Vorwort

Zu Beginn der 90er-Jahre begann man bei Sun Microsystems unter Leitung von James Gosling eine neue Programmiersprache zu entwickeln, die am Anfang den Namen Oak (Eiche) trug. 1995 wurde sie, aufgrund namensrechtlicher Probleme, in Java umbenannt und am 23. Mai 1995 auf der SunWorld in San Francisco vorgestellt. Einem Siegeszug von Java stand nichts mehr im Wege. Java wurde nach dem amerikanischen Slangbegriff für Kaffee und dem T-Shirt-Aufdruck „It's a jungle out there, so drink your Java" benannt. Java ist eine objektorientierte Programmiersprache, die heute für viele Internetanwendungen und für Handy-Software verwendet wird.

Die Bedeutung von Java hat über die Jahre immer mehr zugenommen, so wird es für Programmierer immer wichtiger, Java zu beherrschen. So stellt sich die Frage: Wie können Sie sich das entsprechende Wissen aneignen oder sogar Ihr Wissen erweitern und unter Beweis stellen? Ein Weg dorthin wäre: Machen Sie die Prüfung für den Sun Certified Java Programmer (SCJP)! Die bestandene Zertifizierung könnte ein Meilenstein in Ihrer beruflichen Karriere sein. In diesem Buch können Sie anhand von Fragen gezielt Java-Programmiergrundlagen erwerben und die Prüfung anschließend problemlos bestehen. Dieses Buch soll es neben Programmierprofis auch Anfängern ermöglichen, sich Kenntnisse anzueignen. So sind die Beispiele einfach gehalten, und Kompliziertes und Komplexes wird leicht verständlich erklärt.

Danksagung

Meinen besoderen Dank gilt Andreas Buchman, der mich durch alle Phasen der Bucherstellung mit seiner Kritik, seinen Anregungen und seinen Ideen begleitet hat.

II. Einleitung

A. Über dieses Buch

1. Über den Inhalt

Dieses Buch besteht aus zwei Teilen, der erste Teil bereitet Sie auf die SCJP-Prüfung für Java 2 Platform 1.4 vor, und der zweite ergänzt die Themen, die zusätzlich für die Java 2 Platform-5.0-Zertifizierung notwendig sind. Die Prüfung, die Sie ablegen werden, bezieht sich immer auf die entsprechende Java-Version und muß nur einmal abgelegt und nie erneuert werden. Es besteht aber die Möglichkeit, sich zusätzlich für eine neuere Java-Version zertifizieren zu lassen.

Für jedes prüfungsrelevante Thema erfolgt eine Einführung mit allen für die Zertifizierung wichtigen Details. Nach dieser Beschreibung gibt es Übungen, die vom Schwierigkeitsgrad her leichter sind als die tatsächlichen Zertifizierungsfragen, da an dieser Stelle, pro Aufgabe nur ein Detail variiert und trainiert werden soll. Diese Vorgehensweise soll Anfänger vor einer Überforderung bewahren und Fortgeschrittene trotzdem nicht langweilen. Im Anschluß gibt es Lösungen, die teilweise kommentiert sind. Die Lösungen sind nur teilweise kommentiert, da ich bereits in den vorangegangenen Ausführungen auf alle notwendigen Einzelheiten eingegangen bin.

2. Wie arbeite ich mit diesem Buch?

Die einzelnen Kapitel bauen aufeinander auf und somit ist es vor allem für den Anfänger sinnvoll, dieses Buch Kapitel für Kapitel durchzuarbeiten. So laufen Sie nicht Gefahr, überfordert zu werden. Die Übungen dienen dazu, das Wissen des jeweiligen Themas zu vertiefen und zu festigen. Sie sollten zuerst ein Kapitel lesen, anschließend die Aufgaben machen und dann Ihre Ergebnisse mit den Lösungen vergleichen. Ergeben sich in diesem Zusammenhang noch Fragen, rekapitulieren Sie den noch nicht verstandenen Stoff. Von Zeit zu Zeit sollten Sie einige Seiten zurückblättern und das eine oder andere Thema wiederholen, da doch beim ersten Mal nicht jedes Detail vollkommen beherrscht wird.

3. Wie und wo melde ich mich für die Prüfung an?

Die Anmeldung für die Prüfung erfolgt über die Prometric (www.prometric.com), einer weltweiten Zentralstelle für Zertifizierungen. Da Sie den Test nur unter Beaufsichtigung ablegen dürfen, brauchen Sie ein Schulungsunternehmen, das für die Prometric Prüfungen abnehmen darf. Solche Bildungseinrichtungen gibt es in allen größeren deutschen Städten. Bevor Sie sich zertifizieren lassen können, müssen Sie bei der Prometric einen Gutschein beantragen, der 12 Monate gültig ist. Die entsprechenden Kosten belaufen sich auf 150 US $.

4. Wie sieht die Prüfung aus?

Für die Prüfung 1.4 (CX-310-035) müssen 61 Fragen in 120 Minuten beantwortet werden, die vom Schwierigkeitsgrad her höher sind als in meinem Buch, da sie pro Aufgabe nicht jedesmal nur ein Detail abfragen. So kann es vorkommen, daß in einer Aufgabe mehrere Themen miteinander kombiniert werden. Wann haben Sie die Prüfung bestanden? Es muß eine Frage mehr als die Hälfte richtig sein. Dies entspricht 32 korrekten Antworten. Bei der Zertifizierung für die Platform 5.0 (CX-310-055) sind es 72 Fragen in 175 Minuten und 43 müssen zutreffend beantwortet werden. Die alte Prüfung besteht ausschließlich aus Muliple-Choice-Fragen, wohingegen die neue, zusätzlich Drag-and-Drop-Aufgaben enthält.

5. Wo kann ich Informationen zu Prüfungsfragen finden?

Fangen wir mit einer Informationsquelle an, die umsonst ist: Auf der Seite von Dan Chisholm (www.danchisholm.net) finden Sie eine Vielzahl an Prüfungsaufgabe mit kommentierten Lösungen, die viel schwerer sind, als die Tatsächlichen. Sie stellen eine gute zusätzliche Prüfungsvorbereitung dar, wobei man sich nicht frustrieren lassen sollte, da die tatsächlichen Zertifizierungsfragen viel leichter ist.

Dann gibt es noch eine amerikanische Lernsoftware von Whizlab (www.whizlab.com), die die Prüfungssituation simuliert. Die Fragen mit kommentierten Lösungen sind ein optimales Training, und man kann sich auf Prüfungsfragen mit ähnlichem Schwierigkeitsgrad einstellen.

Zum Schluß möchte ich Ihnen noch ein Forum in englischer Sprache vorstellen, das mehrere Rubriken zum Thema Java-Zertifizierung besitzt: www.javaranch.com.

6. Wo kann ich zusätzliche Informationen über die genauen Themen der Prüfung finden?

Die verlässlichste Quelle ist Sun MicroSystems selbst. Auf der Sun-Website (www.sun.com/training/certification) finden Sie die entsprechenden Prüfungsthemen detailliert aufgelistet. Dieses Buch deckt alle prüfungsrelevanten Themen inhaltlich ab.

B. Wie arbeite ich mit Java und Eclipse?

1. Was ist das Java Development Kit (JDK) und Eclipse?

a) Java Development Kit (JDK)

Sie wollen Java-Programme ausführen? Dann müssen Sie die Java Virtual Machine (JVM) auf Ihrem Computer installieren. Die Java Virtual Machine ist Bestandteil der Java Runtime Environment (JRE), zu deutsch auch Java-Laufzeitumgebung genannt. Die JRE enthält außerdem die Java-Api (Application and Programming Interface), die alle wichtigen Klassen, die Sie zum Programmieren benötigen, bereitstellt. Die Java-Api ist eine Bibliothek, die alle wichtigen virtuellen Bücher (Packages) mit Informationen über die Programmiersprache Java enthält. Die Java Virtual Machine wird auch Interpreter genannt, sie „interpretiert", den für Sie gänzlich unverständlichen Bytecode und übersetzt ihn in eine Maschinensprache und macht es so möglich, daß ein Programm problemlos laufen kann.

Was brauchen Sie aber, um aus Ihrem Code ein lauffähiges Programm machen zu können? Sie benötigen einen Kompiler, der den Quelltext (die .java-Dateien) in einen Bytecode (die .class-Dateien) übersetzt. Sie brauchen also einen Dolmetscher, der Ihre Zeilen in eine Sprache überträgt, die die Java Virtual Machine versteht. Diese Aufgaben erfüllt das Java Development Kit (JDK), das auch den Namen Java 2 Software Development Kit trägt (J2SDK). Das Java Development Kit (JDK) enthält nicht nur einen Compiler, sondern auch die Java Runtime Environment (JRE).

b) Eclipse

Eclipse ist eine Entwicklungsumgebung, die eine große Verbreitung findet und eine Hilfe beim Programmieren darstellt. Eclipse unterstützt Sie, indem es Tipp- und Syntaxfehler anzeigt und Ihnen Programmiervorschläge macht und erleichtert Ihnen somit das Programmieren.

2. Installation des Java Development Kits

(1) Schritt

Laden Sie das Java 2 Software Development Kit von folgender Seite herunter http://java.sun.com/j2se/1.4.2/download.html. Sollten Sie die neue Prüfung zu dem Tiger Release machen wollen, können Sie auch das JDK 1.5 herunterladen. Achten Sie darauf, daß Sie das J2SDK herunterladen und nicht das JRE. Das JRE (Java Runtime Environment) erlaubt nur das Ausführen von bereits übersetzten Java-Klassen und nicht das Kompilieren (Übersetzen), wohingegen das JDK (Java Development Kit) Ihnen beide Funktionen zur Verfügung stellt.

(2) Schritt

Doppelklicken Sie auf die entsprechende Datei mit der Endung .exe. Java schlägt Ihnen als Verzeichnisort und -name das Verzeichnis c:\j2sdk1.4.2_05 vor,

übernehmen Sie diesen Vorschlag. Sie sehen im Explorer das entstandene Java-Verzeichnis:

(3) Schritt

Sie müssen Ihrem Windows-System zeigen, wo sich dieses neu installierte Programm und alle dazugehörigen wichtigen Teil-Programme befinden. Diesen Schritt übernimmt gewöhnlicherweise bei anderen Programmen der Installationsvorgang, nicht so bei dem SDK von Java. Sie müssen selbst Veränderungen bei der Path-Variablen und der Umgebungsvariablen vornehmen. Je nach Betriebssystem funktioniert dies anders. An dieser Stelle sagen Sie dem Betriebssystem mithilfe der Umgebungsvariable, wo sich das Java-Verzeichnis (C:\j2sdk1.4.2_05) befindet. Es ist notwendig dieses Verzeichnis in JAVA_HOME umzubenennen, da Java sich intern immer auf diesen Aliasname bezieht. JAVA_HOME ist der Name für das Verzeichnis, in dem sich alle Java-Dateien befinden. Des weiteren muß die Path-Variable angepasst werden.

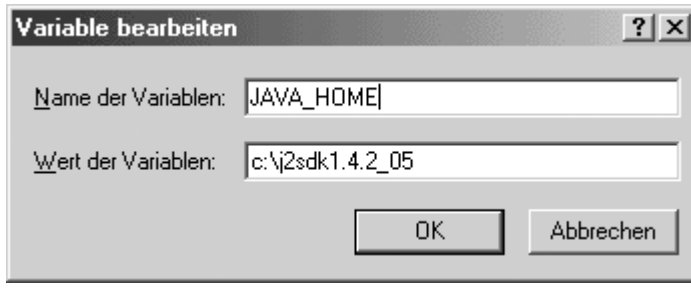
a) Unter Windows 95/98

Unter den alten Betriebssystemen müssen Sie die Systemdatei autoexec.bat verändern, die sich direkt unter C:\ befindet. Bevor Sie dies tun, empfehle ich Ihnen allerdings von dieser Datei eine Sicherungskopie zu erstellen. Dieser Datei müssen Sie die Umgebungsvariable SET JAVA_HOME=c:\j2sdk1.4.2_05 hinzufügen. Des weiteren sollte die Path-Variable so aussehen: SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\j2sdk1.4.2_05\bin. Der letzte Eintrag C:\j2sdk1.4.2_05\bin wird an den Schluß gesetzt, der aber zuerst mit einem Semikolon ergänzt wird.

b) Unter Windows Me

Hier funktioniert das Ganze komfortabler, aber leider ganz anders als unter Windows XP. Gehen Sie zu Start, und wählen Sie Start/Programme/Zubehör/Systemprogramme/Systeminformationen, wenn Sie dort angelangt sind, wählen Sie im Menüpunkt unter Extras den Punkt Systemkonfigurationsprogramm und die Registerkarte Umgebung aus. Klicken Sie auf den Button neu und erstellen die

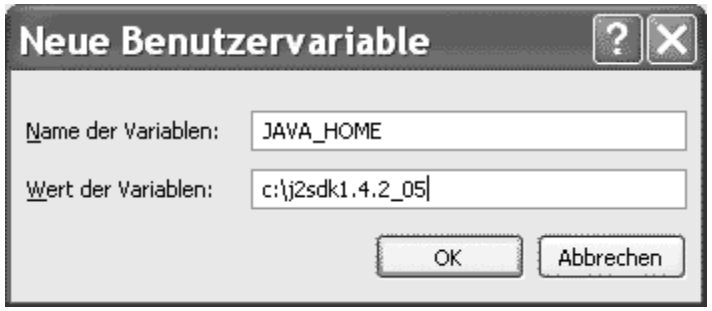
Umgebungsvariable wie folgt:



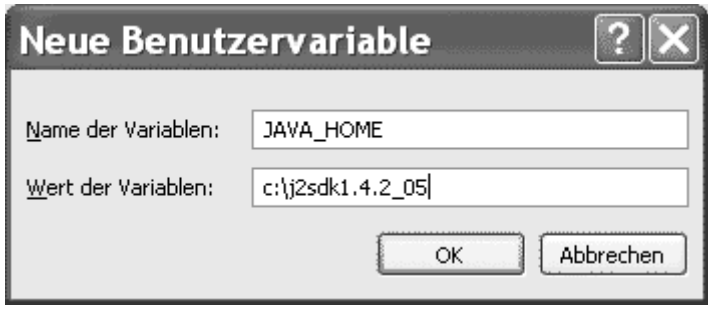
Die Path-Variable müssen Sie verändern, indem Sie die Path-Variable markieren und auf den Button Bearbeiten klicken. Anschließend fügen Sie an den Schluß ein Semikolon und folgenden Eintrag c:\j2sdk1.4.2_05\bin ein:

c) Unter Windows NT, 2000 und XP

Unter Windows NT und Windows 2000 wird wie folgt vorgegangen: Start/ Einstellungen/Systemsteuerung/System. Und unter XP geringfügig anders: Start/ Systemsteuerung/System. Es wird die Umgebungsvariablen neu anlegt und der PATH ergänzt. Unter Windows NT müssen Sie die Umgebungsregisterkarte, unter Windows 2000 die Registerkarte erweiterte Einstellungen und unter XP die Registerkarte Erweitert auswählen. Anschließend wird der Button Umgebungsvariable angeklickt, und die Benutzervariable, durch anklicken auf den Button neu, wie folgt erstellt:



Die Path-Variable verändern Sie, indem Sie sie markieren und auf den Button Bearbeiten klicken. An den Schluß wird ein Semikolon und folgender Eintrag c:\j2sdk1.4.2_05\bin angefügt. Die PATH_Variable befindet sich unter den Systemvariablen.



3. Installation von Eclipse

(1) Schritt

Laden Sie bitte von folgender Seite www.eclipse.org Eclipse 3.0 (für Java 1.4), Eclipse 3.1 (für Java 1.5) oder höher herunter.

(2) Schritt

Entzippen Sie die gezippte Datei in das Laufwerk c:\. Dort wird ein Verzeichnis erstellt, das eclipse heißt.

(3) Schritt

Nur für Windows XP/2000/NT: Kopieren Sie nun das Verzeichnis jre, das sich im Ordner C:\j2sdk1.4.2_05 befindet, in den Ordner c:\eclipse. Der Ordner jre enthält die Java Runtime Environment, die das Ausführen von bereits übersetzten Java-Klassen erlaubt.

III. Grundlagen der Programmierung in Java

A. Das erste Java-Programm in Eclipse

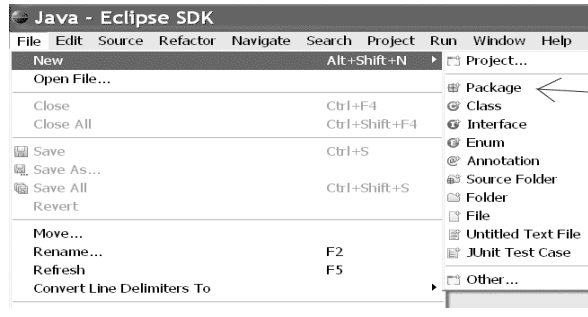
Lassen Sie uns mit unserem ersten Java-Programm beginnen: Ein Beispiel, das in keinem Programmierhandbuch fehlen darf. Wir veranlassen mit dem Befehl System.out.print(„Hello World“);, daß auf dem Bildschirm, der Satz „Hello World“ ausgegeben wird. Gestartet wird dieser Befehl mit der Methode „public static void main (String[ ] args)“, die als Anfangspunkt des Programms gilt. Dieser Befehl läßt sich vergleichen mit der Play-Taste auf der Fernbedienung Ihres Videorekorders.

```
public class Java_Applications {
        //Startpunkt des Programms
        public static void main(String[ ] args) {
            //Methode zur Ausgabe auf dem Bildschirm
            System.out.print(”Hello World“);
        }
    }
```

Ausgabe auf der Konsole:

Hello World

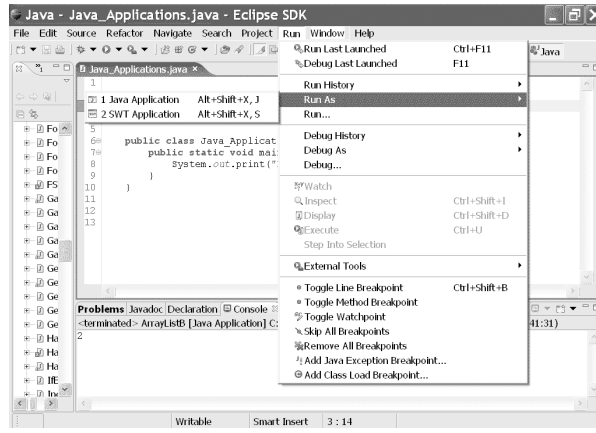
Wie wird nun unser erstes Programm umgesetzt? Zuerst müssen Sie das hierfür benötigte Programm öffnen: Eclipse (vgl. Kapitel Wie arbeite ich mit Java und Eclipse?). Was ist als nächstes notwendig? Bevor Sie ein Programm erstellen können, brauchen Sie ein sogenanntes Package, das vergleichbar ist mit einem Ordner in Windows. In einem Package werden mehrere Programme zusammengefaßt. Ein Package erstellen Sie, indem Sie im Menü File ➔ New ➔ Package anklicken und einen Namen vergeben.



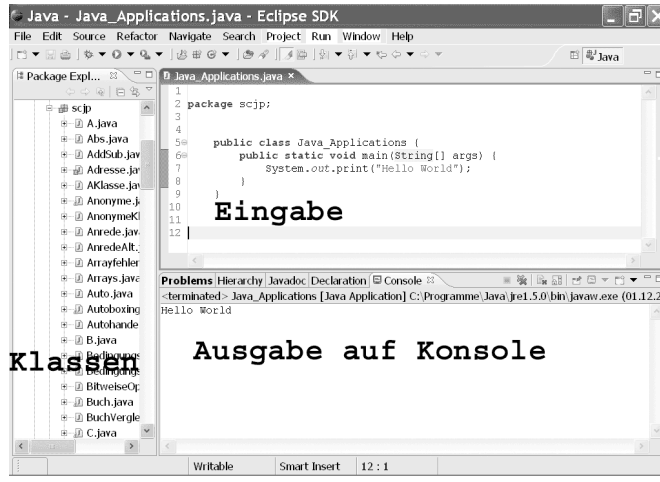
Im nächsten Schritt erstellen Sie eine Klasse (File ➔ New ➔ Class), vergeben den Namen Java_Applications und tippen obenstehende Programmierzeilen ein. Der Begriff Klasse steht in Java für ein Java-Programm. Diese Erläuterung soll

uns an dieser Stelle genügen, im Kapitel Klassen und Objekte wird dieser Fachausdruck näher erläutert.

Zum Schluß wird das Programm kompiliert, d. h. übersetzt. Kompilieren können Sie das Programm, indem Sie im Menü Run ➔ Run as ➔ Java Application anklicken.



In untenstehender Abbildung sehen Sie das Ergebnis unseres Programms: Es wird der Satz „Hello World“ auf der Konsole ausgegeben.



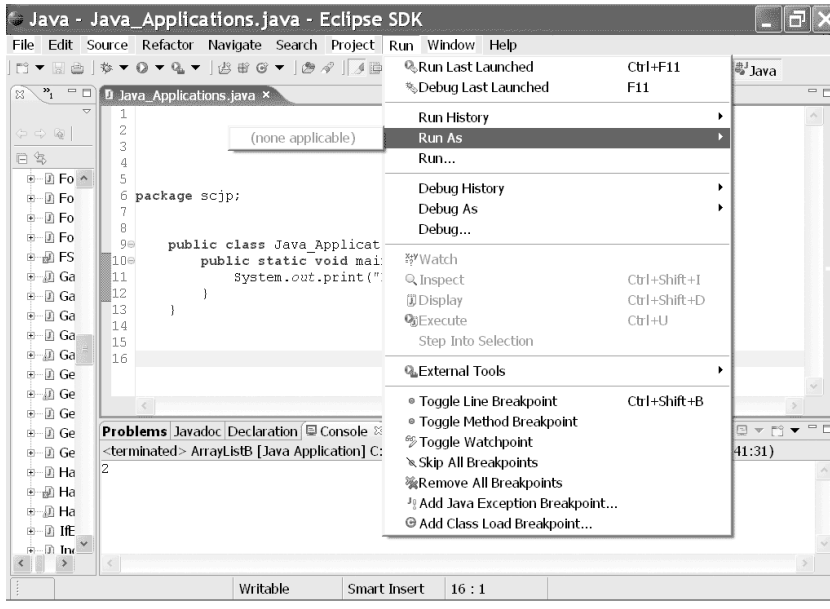
B. Die main()-Methode

1. Begriffsbestimmung

Wird ein Programm kompiliert, sucht der Kompiler als erstes nach der Methode „public static void main (String[ ] args)“. Sie gilt als Startpunkt des Programms, fehlt sie, kann der Quelltext nicht in Java übersetzt werden, und es wird nichts auf dem Bildschirm ausgegeben.

2. Fehlermöglichkeiten in der main()-Methoden

Wie Sie im weiteren immer wieder sehen werden, hängt oft der Erfolg beim Programmieren von Feinheiten ab. Läßt man etwas weg, wie z. B. eine Klammer oder einen Semikolon, kommt es zu einem Kompilierfehler. Diese Fehler bemerkt Eclipse sofort und markiert sie rot. Andere Fehler, Laufzeitfehler, werden erst bemerkt, nachdem das Programm gestartet worden ist, solange das Programm läuft. Wird allerdings in der main()-Methode etwas falsch geschrieben, schlägt schon der Versuch fehl, das Programm zu starten. Werden z. B. die eckigen Klammern nach String vergessen, ist ein Kompilieren nicht möglich:



In den Antworten der Prüfung wird nicht davon ausgegangen, daß Sie mit dem komfortablen Hilfsmittel Eclipse arbeiten, sondern nur mit dem Java-Kompiler,

somit wird in den Lösungen davon gesprochen, daß der Versuch, das Programm auf der Kommandozeile zu starten, fehlschlägt.

Hier einige Beispiele, welche Variationsmöglichkeiten der main()-Methode erlaubt sind und welche nicht.

Die main()-Methode kann nur wie folgt variiert werden:

- public static void main(String[ ] args) {}
- public static void main(String [ ] args) {}
- public static void main(String args[ ]) {}

Bei folgenden Variationsmöglichkeiten, wird der Versuch, das Programm auf der Kommandozeile zu starten, fehlschlagen:

- public void main(String[ ] args) {} (static fehlt)
- public static void main(String args) {} ([ ] fehlt)
- private static void main(String[ ] args) {} (es darf kein privat stehen)
- protected static void main(String[ ] args) {} (es darf kein protected stehen)

3. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A { public static void main(String[ ] args) {} }      Zeile 1
public class B { public static void main(String [ ] args) {} }     Zeile 2
public class C { public static void main(String args[ ]) {} }      Zeile 3
```

a) Kompilierfehler in Zeile 1.
b) Kompilierfehler in Zeile 2.
c) Kompilierfehler in Zeile 3.
d) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 1 fehlschlagen.
e) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 2 fehlschlagen.
f) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 3 fehlschlagen.
g) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A { public void main(String[ ] args) {} }             Zeile 1
public class B { public void main(String [ ] args) {} }            Zeile 2
public class C { public void main(String args[ ]) {} }             Zeile 3
```

a) Kompilierfehler in Zeile 1.
b) Kompilierfehler in Zeile 2.
c) Kompilierfehler in Zeile 3.
d) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 1 fehlschlagen.
e) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 2 fehlschlagen.
f) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 3 fehlschlagen.
g) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A { public static void main(String[ ] args) {} }      Zeile 1
public class B { public static void main(String args) {} }         Zeile 2
public class C { public static void main(String args[ ]) {} }      Zeile 3
```

a) Kompilierfehler in Zeile 1.

b) Kompilierfehler in Zeile 2.
c) Kompilierfehler in Zeile 3.
d) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 1 fehlschlagen.
e) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 2 fehlschlagen.
f) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 3 fehlschlagen.
g) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A { public static void main(String[ ] args) {} }        Zeile 1
public class B { public static void main(String[ ] args) {} }        Zeile 2
public class C { protected static void main(String[ ] args) {} }     Zeile 3
```

a) Kompilierfehler in Zeile 1.
b) Kompilierfehler in Zeile 2.
c) Kompilierfehler in Zeile 3.
d) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 1 fehlschlagen.
e) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 2 fehlschlagen.
f) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 3 fehlschlagen.
g) Keine dieser Möglichkeiten.

(5) Frage

Was wird passieren, wenn Sie versuchen untenstehende Programme kompilieren und laufen zu lassen?

```
public class A { public static void main(String[ ] args) {} }        Zeile 1
public class B { private static void main(String[ ] args) {} }       Zeile 2
public class C { public static void main(String[ ] args) {} }        Zeile 3
```

a) Kompilierfehler in Zeile 1.
b) Kompilierfehler in Zeile 2.
c) Kompilierfehler in Zeile 3.
d) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 1 fehlschlagen.
e) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 2 fehlschlagen.
f) Der Versuch, das Programm auf der Kommandozeile zu starten, wird bei Zeile 3 fehlschlagen.
g) Keine dieser Möglichkeiten.

4. Lösungen

(1) Frage

g

(2) Frage

d, e, f

(3) Frage

e

(4) Frage

f

(5) Frage

e

C. Namen in Java: Bezeichner

1. Begriffserläuterung

Wie wir bereits gesehen haben, gibt es auch in Java für alles einen Namen. Namen in Programmen heißen Bezeichner. Ein Bezeichner besteht aus mehreren Zeichen, wie Buchstaben oder Ziffern. Bei der Namensgebung in Java müssen folgende Regeln eingehalten werden:

Bezeichner dürfen folgende Zeichen enthalten:

- Buchstaben
- Ziffern
- Unterstrich (_)
- Währungssymbole (wie z. B. $ und €)

Sollten Bezeichner folgende Zeichen enthalten, führt dies zu einem Kompilierfehler:

- erstes Zeichen darf keine Ziffer sein
- **alle** Sonderzeichen (wie z. B. §,%,& und #)
- Bindestriche (-)

a) Beispiele

Beispiele für erlaubte Bezeichner:

- baum_stumpf
- $bild
- haus1
- baum$_12

Beispiele für nicht erlaubte Bezeichner:

- 1baum
- müller@t-online.de
- baum-stumpf
- 34haus
- %baum
- computer#bild

2. Übungen

(1) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Bezeichner{
        String a5;                  Zeile 1
        String 5a;                  Zeile 2
        String €buch;               Zeile 3
        String text;                Zeile 4
        String @reminder;           Zeile 5
        String honigmund1;          Zeile 6
        String text-art;            Zeile 7
        String &computer;           Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8

(2) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Bezeichner{
        String auto;                Zeile 1
        String honig;               Zeile 2
        String question1;           Zeile 3
        String _text;               Zeile 4
        String 2question;           Zeile 5
        String question_1;          Zeile 6
        String text_art;            Zeile 7
        String question-2;          Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8

(3) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Bezeichner{
        String grüßen;                  Zeile 1
        String number;                  Zeile 2
        String first_number;            Zeile 3
        String first-number;            Zeile 4
        String number%;                 Zeile 5
        String number1;                 Zeile 6
        String 1number;                 Zeile 7
        String number-1;                Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Bezeichner{
        String plan5;                   Zeile 1
        String 5plan;                   Zeile 2
        String $plan;                   Zeile 3
        String plan;                    Zeile 4
        String märz;                    Zeile 5
        String total+;                  Zeile 6
        String total;                   Zeile 7
        String total_plan;              Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Bezeichner{
        String hallo;                   Zeile 1
        String a2c;                     Zeile 2
        String hot;                     Zeile 3
```

```
        String hot57;                    Zeile 4
        String version-4;                Zeile 5
        String version1;                 Zeile 6
        String version_9;                Zeile 7
        String _version;                 Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8

3. Lösungen

(1) Frage

b, e, g, h

(2) Frage

e, h

(3) Frage

a, d, e, g, h

(4) Frage

b, f

(5) Frage

e

D. Schlüsselwörter

1. Begriffserläuterung

Folgende Schlüsselwörter dürfen nicht als Namen / Bezeichner verwendet werden, da sie bereits eine vordefinierte Bedeutung in Java haben! So ist es nicht erlaubt, Ihr Programm break oder boolean zu nennen. Diese Wörter sind keysensitive, d. h. es wird zwischen Groß- und Kleinbuchstaben unterschieden. So sind die Begriffe break (kleiner Anfangsbuchstabe) und Break (großer Anfangsbuchstabe) nicht identisch. Das Wort break, das ein Schlüsselwort ist, darf nicht als Bezeichner benutzt werden, aber Break. Also immer auf Groß- und Kleinschreibung achten, da alle Schlüsselwörter nur Kleinbuchstaben enthalten!

Liste der Schlüsselwörter:

					abstract
assert	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

Reservierte Literale:

null, true, false

Reservierte Schlüsselwörter, die im Moment nicht benutzt werden:

const, goto

Neues Schlüsselwort ab Version 1.5:

enum

2. Übungen

(1) Frage

Welcher der untenstehenden Begriffe gehören zu den Java-Schlüsselwörter?

a) instanceof
b) Instanceof
c) better
d) value
e) switch
f) goto
g) cast
h) Keine dieser Möglichkeiten.

(2) Frage

Welcher der untenstehenden Begriffe gehören zu den Java-Schlüsselwörter?

a) array
b) decimal
c) application
d) While
e) Long
f) string
g) print
h) Keine dieser Möglichkeiten.

(3) Frage

Welcher der untenstehenden Begriffe gehören zu den Java-Schlüsselwörter?

a) while
b) map
c) do
d) exception
e) argument
f) collection
g) set
h) Keine dieser Möglichkeiten.

(4) Frage

Welcher der untenstehenden Begriffe gehören zu den Java-Schlüsselwörter?

a) this
b) try
c) nested
d) loop
e) constructor
f) trim

g) class
h) Keine dieser Möglichkeiten.

(5) Frage

Welcher der untenstehenden Begriffe gehören zu den Java-Schlüsselwörter?

a) args
b) public
c) system
d) single
e) double
f) new
g) old
h) Keine dieser Möglichkeiten.

3. Lösungen

(1) Frage

a, e, f

(2) Frage

h

(3) Frage

a, c

(4) Frage

a, b, g

(5) Frage

b, e, f

E. Zahlen und Buchstaben in Java: Primitive Datentypen und Variablendeklaration

1. Primitiver Datentyp

Nehmen wir an, Sie wollen Zigaretten am Automaten ziehen, und Sie haben nur einen 5 Euro-Schein. Keine Ein-Euro-Münzen, keine Zigaretten. Ebenso geht es Ihnen am Kaugummiautomat, kein 10 Cent Stück, kein Kaugummi. In Java heißen die verschiedenen Automaten, primitive Datentypen (vgl. Kapitel Überschreiben von Methoden). Ein Datentyp legt von vorneherein fest, wie groß eine Zahl sein darf. So darf eine Zahl vom primitiven Datentyp byte nicht größer als +127 sein. Wollen Sie in den primitiven Datentyp byte eine Zahl +128 oder 1.267 (Kommazahl in amerikanischem Zahlenformat) eingeben, so kommt es zu einer Fehlermeldung. Aber, so denken Sie, 1.267 ist doch viel kleiner als 127! Kein Zweifel, Sie haben Recht. In Java jedoch, sieht der Datentyp byte keinen Platz für eine Zahl mit Nachkommastellen vor, somit ist die Zahl 1.267 dennoch viel zu groß. Bitte beachten Sie außerdem, daß Fließkommazahlen in amerikanischen Zahlenformat eingegeben werden müssen, mit Punkt statt Komma als Trennzeichen.

Hier eine Zusammenstellung der wichtigsten Datentypen:

Daten-typen	Bytes	Bits	Reichweite	Minimaler und Maximaler Wert (MIN_VALUE, MAX_VALUE)	
boolean			true oder false		false
char	2	16	16-Bit Unicode Zeichen, nur ein Zeichen, nur positive Zeichen (Ziffern, Buchstaben und Sonderzeichen)	(0x0000...0xffff) 0 ... 65535 (\u0000 ... \uffff)	‚\u0000‘
byte	1	8	-2^7 bis $2^7 - 1$	-128...127	0
short	2	16	-2^{15} bis $2^{15} - 1$	-32768...32767	0
int	4	32	-2^{31} bis $2^{31} - 1$	-2147483648...2147483647	0
long	8	64	-2^{63} bis $2^{63} - 1$	-9223372036854775808... 9223372036854775807	0L
float	4	32	1.40239846E-45f 3.40282347E+38f		0.0F
double	8	64	4.94065645841246544E-324 1.79769131486231570E+308		0.0D

2. Variablendeklarationen und Initialisierungen von Variablen

Wir wollen sicherstellen, daß Ihr Kind nicht mehr als 127 Kaugummis im Monat kauft und ißt. Die vorgesehene Menge Kaugummis darf also die Zahl 127 nicht übersteigen. So wäre der hierfür ideale Datentyp, der Datentyp byte. Die Menge wäre der Name unserer Variablen (vgl. Kapitel Bestandteile von Klassen). Variablen sind veränderliche Größen, und sie beinhalten Werte eines bestimmten primitiven Datentyps. Wird der Variablen einen Namen zugewiesen, wie z. B. byte menge; wird von einer **Variablendeklaration** gesprochen. Variablennamen werden klein geschrieben, und Variablendeklarationen werden mit einem Semikolon abgeschlossen. Die Variablendeklaration byte a, b, c; ist identisch mit byte a; byte b; byte c;. Wird der Variablen noch ein Wert zugewiesen, spricht man von einer **Initialisierung** (byte menge = 30). Sie legen also mit byte menge = 30 fest, daß Ihr Kind normalerweise 30 Kaugummis essen darf. Wird einer Variablen kein Wert zugewiesen, hat sie trotzdem einen Wert, den sogenannten Defaultwert. In der Fachliteratur wird auch der Begriff Standardwert als Synonym verwendet, und der Defaultwert beim Datentyp byte ist die Zahl 0. Wird also keine Menge festgelegt, darf Ihr Kind gewöhnlich keinen Kaugummi essen.

3. Literale

Der Wert, der der Variablen während der Initialisierung zugewiesen wird, ist ein sogenannter Literal. In unserem Beispiel byte menge = 30, ist byte der Datentyp, menge der Name der Variablen und 30 ist der dazugehörige Wert, also der byte-Literal.

a) Char-Literale

Der primitive Datentyp char darf nur ein Zeichen enthalten, das eine positive Ziffer, ein Buchstabe und ein Sonderzeichen sein kann. Char darf ein Unicode-Zeichen zum Inhalt haben, wobei z. B. \u0031 die Zahl 1 bedeutet. Der Unicode ist ein international gültiger Code, der es möglich macht, daß zusätzlich zu unseren Buchstaben (A - Z) auch arabische und chinesische Zeichen ausgegeben werden können. Char-Literale werden mit einfachen Anführungszeichen dargestellt, Ziffern ohne. Fehlen die einfachen Anführungszeichen, oder beinhaltet char eine negative Zahl, kommt es zu einem Kompilierfehler. Char kann als Hexadezimalzahl und Oktalzahl ohne Anführungszeichen dargestellt werden: char a = 0xffff; oder char a = 062;. Vergleichen Sie hierzu Kapitel Exkurs: Zahlensysteme im Kapitel Integer Bitweise Operatoren.

Mit char-Literalen besteht die Möglichkeit, alle 65.536 Zeichen des Unicodes darzustellen, so gibt z. B. das Unicode-Zeichen u\0041 den Großbuchstaben A wieder. In untenstehender Abbildung wird gezeigt, daß eine Zahl ohne Hochkommata zu der Ausgabe des Zeichens an dieser Position führt und nicht zu einer Ausgabe dieser Zahl. Jede Zahl entspricht einem Unicode-Zeichen. Es gibt also mehrere Möglichkeiten, zusätzlich zu char = ‚A‘; den Großbuchstaben A auf der Konsole auszugeben.

```
public class character {
        public static void main(String[ ] args) {
                char a = 65;
                char b =‘A‘;
                char c = ‘\u0041’;

                System.out.print(“Dies ist der Inhalt der Variable a: “);
                System.out.println(a);
                System.out.print(“Dies ist der Inhalt der Variable b: “);
                System.out.println(b);
                System.out.print(“Dies ist der Inhalt der Variable c: “);
                System.out.println(c);
        }
}
```

Ausgabe auf der Konsole:

```
Dies ist der Inhalt der Variable a: A
Dies ist der Inhalt der Variable b: A
Dies ist der Inhalt der Variable c: A
```

Erlaubte Darstellungsweise von char:

- char a = 'a'; (Buchstaben bei char immer mit einfachen Hochkommata)
- char a = '\u0031'; (Unicode-Zeichen)
- char a = 0xffff;(als Hexadezimalzahl)
- char a = 061; (als Oktalzahl)
- char a = 65535; (Zahlen bis 65535, da es 65536 Unicode-Zeichen gibt)

Darstellungsweisen, die zu Kompilierfehlern führen:

- char a = -1; (negative Zahlen sind nicht erlaubt)
- char a = a; (Hochkommata fehlen)
- char a = \u0031; (Hochkommata fehlen)
- char a = 0xfffff;(maximaler Wert 0xffff)
- char a = 65536 (maximaler Wert 65535)
- char a = '65'; (zwei Ziffern in Hochkommata)
- char a = 'ab'; (zwei Buchstaben)

b) Integer-Literale

Die Ganzzahlen int, long, byte und short werden unter dem Oberbegriff Integer-Literale zusammengefaßt. Sie können sowohl als Hexadezimalzahl als auch als Oktalzahl dargestellt werden, und Zahlen stehen nie in Hochkommata. Integer Literale sind automatisch immer vom Datentyp int. Sollen sie long sein muß ein L dahinter stehen (long = 10L). Die primitiven Datentypen short oder byte müssen nicht speziell gekennzeichnet werden.

Erlaubte Darstellungsweisen von Integer Literalen:

- byte a = -128;
- byte a = 127;
- short a = -32768;
- short a = 32767;

Darstellungsweisen, die zu Kompilierfehlern führen:

- byte a = 128 (zu groß, maximaler positiver Wert 127)
- byte a = -129 (zu klein, maximaler negativer Wert -128)
- short a = 32768 (zu groß, maximaler positiver Wert 32767)

c) Fließkommazahlen

Alle Kommazahlen werden unter dem Oberbegriff Fließkommazahlen zusammengefaßt, wobei zwischen double und float unterschieden wird. Als Trennzeichen steht ein Punkt, kein Komma, da Java sich der amerikanischen Schreibweise bedient. Kommazahlen sind grundsätzlich vom primitiven Datentyp double: Steht z. B. float a = 0.0; so ist a trotzdem vom primitiven Datentyp double und nicht float. Wollen Sie einen Wert von Typ float haben, muß f explizit dastehen: float a = 0.0F;.

d) Boolean-Literale

Boolean-Literale geben darüber Auskunft, ob etwas zutrifft oder nicht, wobei sie nur zwei Werte, true oder false, annehmen können. Mit einem boolean-Ausdruck sind Sie in der Lage festzustellen, ob heute Montag ist oder nicht. Es gibt nur zwei Möglichkeiten boolean-Literale einer Variablen zuzuweisen, ohne daß es zu einem Kompilierfehler kommt:

- boolean b = true;
- boolean b = false;

e) String-Literale

Ein String-Literal besteht aus mehreren Zeichen oder sogar Wörtern, sprich Text. Mit Text kann man nicht rechnen, so kann man einen String-Literal nicht mit einem anderen String-Literalen multiplizieren. String-Literale werden in Hochkommata gesetzt, wobei es eine Ausnahme gibt, die nicht in Anführungszeichen gesetzt werden muß, und zwar der Defaultwert (String a = null;).

Beispiele, für die korrekte Anwendung:

- String a = null;
- String a = “Ich bin ein String!“

Folgende Beispiele führen zu einem Kompilierfehler:

- String a =’Ich bin ein String!’ (einfache Hochkommata)
- String a = Ich bin ein String! (ohne Anführungszeichen)
- String a = “null“; (der Defaultwert muß ohne Hochkommata stehen)

4. Übungen

(1) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        char    a = a;                      Zeile 1
        char    b = ‘a‘;                    Zeile 2
        char    c = ‘abc’;                  Zeile 3
        char    d = ‘\u0031’;               Zeile 4
        char    e = \u0031;                 Zeile 5
        char    f = 0xff;                   Zeile 6
        char    g = 1;                      Zeile 7
        char    h = -1;                     Zeile 8}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(2) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        char    a = ‘a’;                    Zeile 1
        char    b = abc;                    Zeile 2
        char    c = 2;                      Zeile 3
        char    d = ‘b’;                    Zeile 4
        char    e = -3;                     Zeile 5
        char    f = 0xfffff;                Zeile 6
        char    g = 0xffff;                 Zeile 7
        char    h = ‘e’;                    Zeile 8}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(3) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        byte    a = -129;           Zeile 1
        byte    b = -128;           Zeile 2
        int     c = 2;              Zeile 3
        int     d = 32768;          Zeile 4
        short   e = -32768;         Zeile 5
        short   f = 32768;          Zeile 6
        short   g = -32769;         Zeile 7
        int     h = -32769;         Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        byte    a = -12563;         Zeile 1
        byte    b = -128;           Zeile 2
        int     c = 36589;          Zeile 3
        int     d = 32;             Zeile 4
        short   e = -3276825;       Zeile 5
        short   f = 327;            Zeile 6
        char    g = -327;           Zeile 7
        int     h = -3276925;       Zeile 8}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        String  a = -125;           Zeile 1
```

```
        String    b = “hallo, hier bin ich“;        Zeile 2
        char      c = “hallo, hier bin ich“;        Zeile 3
        String    d = null;                         Zeile 4
        boolean   e = 825;                          Zeile 5
        boolean   f = true;                         Zeile 6
        boolean   g = false;                        Zeile 7
        boolean   h = “true“;                       Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(6) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        String    a = ‘Ich bin ein String’;         Zeile 1
        String    b = “Ich bin ein String“;         Zeile 2
        char      c = ‘Ich bin ein String’;         Zeile 3
        String    d = null;                         Zeile 4
        boolean   e = TRUE;                         Zeile 5
        boolean   f = “false“;                      Zeile 6
        boolean   g = false;                        Zeile 7
        boolean   h = 0;                            Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(7) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        long      a = 35;                           Zeile 1
        String    b = “Ja“;                         Zeile 2
        char      c = ‘I’;                          Zeile 3
```

```
        String    d = null;          Zeile 4
        boolean   e = true;          Zeile 5
        byte      f = 127;           Zeile 6
        boolean   g = false;         Zeile 7
        int       h = 0;             Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(8) <u>Frage</u>

Welche Reichweite besitzt der primitive Datentyp byte?

a) -2^7 bis $2^7 -1$
b) -2^8 bis $2^8 -1$
c) -2^9 bis $2^9 -1$
d) -2^5 bis $2^5 -1$
e) -2^{20} bis $2^{20} -1$
f) -2^{31} bis $2^{31} -1$
g) -2^{63} bis $2^{63} -1$
h) -2^{64} bis $2^{64} -1$
i) Keine dieser Möglichkeiten.

(9) <u>Frage</u>

Welche Reichweite besitzt der primitive Datentyp long?

a) -2^6 bis $2^6 -1$
b) -2^5 bis $2^5 -1$
c) -2^9 bis $2^9 -1$
d) -2^{51} bis $2^{51} -1$
e) -2^{15} bis $2^{15} -1$
f) -2^{31} bis $2^{31} -1$
g) -2^{65} bis $2^{65} -1$
h) -2^{64} bis $2^{64} -1$
i) Keine dieser Möglichkeiten.

(10) <u>Frage</u>

Welche Reichweite besitzt der primitive Datentyp int?

a) -2^7 bis $2^7 -1$
b) -2^8 bis $2^8 -1$

c) -2^9 bis 2^9 -1
d) -2^5 bis 2^5 -1
e) -2^{20} bis 2^{20} -1
f) -2^{31} bis 2^{31} -1
g) -2^{63} bis 2^{63} -1
h) -2^{64} bis 2^{64} -1
i) Keine dieser Möglichkeiten.

(11) Frage

Welches ist der größte positive Wert, den der primitive Datentyp byte einnehmen kann, ohne daß es zu einem Kompilierfehler kommt?

a) 562
b) 32767
c) 2147483647
d) 128
e) 127
f) -128
g) Keine dieser Möglichkeiten.

(12) Frage

Welches ist der größte positive Wert, den der primitive Datentyp int einnehmen kann, ohne daß es zu einem Kompilierfehler kommt?

a) 32768
b) 32767
c) 2147483648
d) 128
e) 127
f) 129
g) Keine dieser Möglichkeiten.

(13) Frage

Welches ist der größte positive Wert, den der primitive Datentyp short einnehmen kann, ohne daß es zu einem Kompilierfehler kommt?

a) 32768
b) 32767
c) 2147483647
d) 2147483648
e) 127
f) 128
g) Keine dieser Möglichkeiten.

5. Lösungen

(1) Frage
a, c, e, h

(2) Frage
b, e, f

(3) Frage
a, f, g

(4) Frage
a, e, g

(5) Frage
a, c, e, h

(6) Frage
a, c, e, f, h

(7) Frage
i

(8) Frage
a

(9) Frage
i

(10) Frage
f

(11) Frage
e

(12) Frage
g

(13) Frage
b

F. Zuweisungen und Typumwandlung

1. Zuweisungen

Bei einer Zuweisung wird einer Variablen ein Wert oder ein neuer Wert zugewiesen.

Der Variablen i wird der Wert 5 zugewiesen:
int i = 5;

Der Variablen j wird der Wert 10 zugewiesen:
int j =10;

Durch erneute Zuweisung wir der Wert der Variablen i geändert, er ist nun 10.
i = j;

In der letzten Zeile wird der Variablen i, die bisher den Wert 5 hatte, ein neuer Wert zugewiesen, nämlich 10.

2. Implizite und explizite Typanpassung

Primitive Datentypen können in andere primitive Datentypen umgewandelt werden (vgl. Zahlen und Buchstaben in Java: Primitive Datentypen und Variablendeklaration). Nehmen wir einmal an, ich habe einer Variablen den Datentyp byte, die eine maximale positive Zahl von 127 darstellen kann, zugewiesen. Ich habe aber z. B. 1000 Autos, die ich verkaufen will. So kann ich diese Zahl nicht mehr mit byte darstellen, sondern ich brauche den nächstgrößeren Datentyp int. So besteht die Möglichkeit, ohne Datenverlust und Fehlermeldungen einem int ein byte zuzuweisen. Diesen Vorgang nennt man implizite Typanpassung. Eine kleine Schachtel kann man immer in eine größere Schachtel legen, aber nicht umgekehrt.

```
public class Typanpassung {
        public static void main(String[ ] args) {
                byte b = 1;
                int i = 1;

                i = b;
        }
}
```

Habe ich einen 5-Euro-Schein, und ich will mir Zigaretten aus dem Automaten ziehen, muß ich vorher Geld wechseln gehen. Genauso verhält es sich, wenn ich eine Zahl vom Datentyp int habe, aber eine byte-Zahl brauche. Dies geht nur mit einer expliziten Typanpassung. Ich wandle also eine größere in eine kleinere Zahl um. In der Fachliteratur wird auch der Ausdruck Cast synonym verwendet. Bei einem Cast kann es passieren, daß die Zahl gekappt wird, damit sie von der

Größe her in das kleinere Format paßt, so werden z. B. die Nachkommastellen abgeschnitten, wenn ein double (Fließkommazahl) in ein long (Ganzzahl) überführt wird. Fehlt in diesem Fall das Casting, kommt es zu einem Kompilierfehler.

```
public class Typanpassung {
        public static void main(String[ ] args) {
                byte b = 1;
                int i = 1;

                b = (byte)i;
        }
}
```

Der primitive Datentyp int i wird auf diese Weise in den primitiven Datentyp byte, der kleiner ist, umgewandelt. Immer wenn ein primitiver Datentyp in einen kleineren Datentyp abgeändert werden soll, braucht man eine explizite Typanpassung, sprich einen Cast.

<u>Folgende Variationen der Typumwandlung sind möglich:</u>

```
byte ➔ short ➔ int ➔ long ➔ float ➔ double
                ↗
          char

---------------------------------------------------------------▸
        implizite Typanpassung

◂ ---------------------------------------------------------------
        explizite Typanpassung, benötigt Casting
```

3. Übungen:

(1) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        long        a = 35;         Zeile 1
        int         b = 35;         Zeile 2
        byte        c = 1;          Zeile 3
        byte        d = 25;         Zeile 4
        int         e = a;          Zeile 5
        byte        f = a;          Zeile 6
        long        g = b;          Zeile 7
        int         h = g;          Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(2) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        short       a = 7;          Zeile 1
        int         b = 980;        Zeile 2
        int         c = 1;          Zeile 3
        byte        d = 298;        Zeile 4
        int         e = a;          Zeile 5
        byte        f = (byte) a;   Zeile 6
        long        g = b;          Zeile 7
        int         h = (int) g;    Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(3) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        double   a = 5.0d;          Zeile 1
        float    b = (float) a;     Zeile 2
        long     c = 1;             Zeile 3
        long     d = 5;             Zeile 4
        int      e = a;             Zeile 5
        byte     f = a;             Zeile 6
        long     g = b;             Zeile 7
        int      h = (int) c;       Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        long     a = 8989;          Zeile 1
        long     b = 9853;          Zeile 2
        long     c = 34;            Zeile 3
        byte     d = 12385;         Zeile 4
        long     e = a;             Zeile 5
        byte     f = a;             Zeile 6
        short    g = b;             Zeile 7
        int      h = 0;             Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
public class Variablen {
        double   a = 8763.0;                    Zeile 1
        float    b =  a;                        Zeile 2
        double   c = 45.0;                      Zeile 3
        long     d = 5;                         Zeile 4
        int      e = a;                         Zeile 5
        int      f = a;                         Zeile 6
        long     g = b;                         Zeile 7
        int      h = b;                         Zeile 8
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Zeile 8
i) Keine dieser Möglichkeiten.

4. Lösungen

(1) Frage

e, Erfordert expliziten Cast, da long größer als int ist.
f, Erfordert expliziten Cast, da long größer als byte ist.
h, Erfordert expliziten Cast, da long größer als int ist.

(2) Frage

d

(3) Frage

e, Erfordert expliziten Cast, da double größer als int.
f, Erfordert expliziten Cast, da double größer als byte.
g, Erfordert expliziten Cast, da float größer als long.

(4) Frage

d,
f, Erfordert expliziten Cast, da long größer als byte ist.
g, Erfordert expliziten Cast, da long größer als short ist.

(5) Frage

b, Erfordert expliziten Cast, da double größer als float ist.
e, Erfordert expliziten Cast, da double größer als int ist.
f, Erfordert expliziten Cast, da double größer als int ist.
g, Erfordert expliziten Cast, da float größer als long ist.
h, Erfordert expliziten Cast, da float größer als int ist.

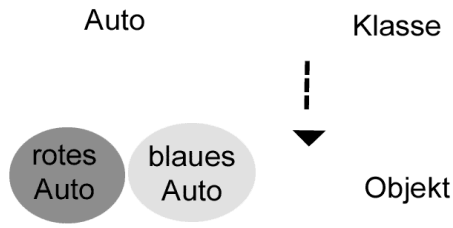
IV. Grundlagen der Objektorientierung

A. Klassen und Objekte

1. Allgemeines

Objektorientiertes Programmieren orientiert sich an den Dingen des Lebens, wie z. B. an Autos, Pizzen, Stühle, Pfirsichen, Büchern und Prüfungen. Diese Dinge werden in Java gruppiert und jeweils in einer Klasse zusammengefaßt. Eine Klasse ist ein Sammelbegriff für gleichartige Objekte. So ist ein Auto ein Sammelbegriff für rote, grüne oder blaue Autos, oder Pizza ist ein Sammelbegriff für Pizza Meeresfrüchte oder Pizza Hawaii. Das blaue Auto oder die Pizza Hawaii sind Objekte, und Autos und Pizzen sind Klassen. Das Wort Instanz wird häufig als Synonym für den Fachbegriff Objekt verwendet. Man kann sich eine Klasse auch als Prototyp für ein Auto vorstellen, für das viel Entwicklungsarbeit geleistet werden muß und das dann als Modell für eine Vielzahl von Objekten benutzt wird. So wird , beispielsweise bei einem neuen Automodell, das Design, der Motor und die Größe festgelegt und jedes einzelne Auto wird in seinen Eigenschaften, wie Farbe oder Innenausstattung, abgeändert.

In die Planung einer Klasse muß ähnlich viel Arbeit gesteckt werden wie in die Entwicklung eines Autos. Bis zur Serienreife eines Autos wird Markforschung betrieben, werden Materialen getestet und neu entwickelt, wird an der Formgestaltung der Karosserie gearbeitet, und es wird neues Know-how berücksichtigt. Sind all diese Anstrengungen zu einem erfolgreichen Ende gebracht worden, wird die Massenproduktion des Autos in Angriff genommen. Softwareentwickler erfüllen vergleichbare Aufgaben wie Ingenieure und Designer, sie planen und konzipieren Klassen. Dieser Prozess wird Softwaredesign oder auch Softwareentwurf genannt. Sind die Klassen erstellt, kann bei jeder entsprechenden Gelegenheit ein gleichartiges Objekt abgeleitet werden. Jedes Objekt ist somit eine leicht veränderte Kopie des Originals, der Klasse. Objektorientiertes Programmieren ermöglicht somit einen hohen Grad an Wiederverwertbarkeit von Klassen. Programmcode, den man erarbeitet hat, kann man in leicht veränderter Art und Weise immer wieder benützen. Es ist wie mit dem Motor eines Autos, ist ein Motor entwickelt, wird beim nächsten Mal nicht wieder ein komplett neuer Motor erstellt, sondern nur noch der bereits Existierende angepaßt und verbessert. So muß nicht jeder das Rad neu erfinden.



2. Erstellung einer Klasse

Nachfolgend wollen wir eine Klasse erstellen, die den Namen Auto trägt. Vor dem Namen der Klasse muß das Schlüsselwort class stehen, das festlegt, daß es sich um eine Klasse handelt (vgl. Kapitel Schlüsselwörter). Klassen sind in Java in der Regel public, d. h. öffentlich, dies soll sicherstellen, daß eine Klasse von überall her sichtbar ist. Dies entspricht dem Grundgedanken einer Klasse, da eine Klasse als Modell für Objekte dienen soll. Und ich kann keine Objekte auf Basis von geheimen Plänen erstellen.

```
public class Auto{ }
```

3. Bestandteile von Klassen

a) Variablen

Variablen sind veränderbare Größen, wie z. B. die Eigenschaft farbe unseres Autos. Unser Auto hat im Moment nur die Eigenschaft farbe, somit enthält unsere erste Klasse auch nur die Variable farbe, die vom Typ String ist und private. Private bedeutet, daß diese Variable nur innerhalb der Klasse Auto sichtbar ist, d. h. von außerhalb der Klasse kann diese Variable weder gesehen noch verändert werden. Hiermit möchte ich verhindern, daß Unbefugte Zugriff auf die Eigenschaften meiner Klasse Auto haben. Private steht somit im Gegensatz zu public. String bedeutet, daß unsere Eigenschaft farbe des Objektes Auto, Text ist und keine Zahl (vgl. Kapitel Die Klasse String). Zahl in diesem Sinne bedeutet, ich kann mit dieser Zahl Berechnungen wie Multiplikation und Addition durchführen, wohingegen ich dies mit einem String nicht kann.

```
public class Auto {
    private String farbe;
}
```

b) Methoden

Mit Methoden können Aktionen durchgeführt oder Eigenschaften verändert werden. Mit den sogenannten Getter- und Setter-Methoden können wir die Eigenschaft farbe unseres Autos modifizieren. Nachstehend unsere Klasse, der Getter- und Setter-Methoden hinzugefügt worden sind, wobei die Getter-Methode, auch Akzessor genannt wird, und die Setter-Methode, Mutator. Mit der Setter-Methode (setFarbe(String farbe)) kann man die Variable farbe verändern und mit der Getter-Methode (getFarbe()) die Variable farbe abfragen. Das Vorgehen, Variablen möglichst privat zu deklarieren und sie dann anschließend mit Getter- und Setter-Methoden zugänglich zu machen, nennt man **Kapselung**. In unserem Beispiel können Sie im Moment noch nicht jede Einzelheit, wie z. B. void, verstehen, da diese Details erst auf den folgenden Seiten erklärt werden, und es im Moment nur um das Verstehen des Prinzips geht.

```
public class Auto {

    private String farbe;

    //Diese Methode zeigt Ihnen die Farbe an.
    public String getFarbe() {
        return farbe;
    }

    //Mit dieser Methode können Sie die Farbe verändern.
    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }
}
```

(1) Rückgabetypen und Parameter

Soeben haben wir gesehen, daß es mit der Methode public String getFarbe() möglich ist, die Farbe eines Objektes auszulesen. Die Methode gibt eine Variable vom Rückgabetyp String zurück. Die Methode public void setFarbe(String farbe) hat den Rückgabetyp void (leer), da sie nichts zurückgibt. Nehmen Sie bitte zur Kenntnis, daß auf jeden Fall void stehen muß, der Methodennamen darf nicht alleine stehen.

Die Methode public void setFarbe(String farbe) gibt zwar nichts zurück, aber dafür ändert sie die Eigenschaft String farbe, indem ihr die Variable farbe vom Typ String zur Verfügung gestellt wird. Wird einer Methode eine Variable übergeben, heißt dieser Wert Übergabewert, wobei der Begriff Parameter in der Fachliteratur synonym verwendet wird.

```
public class Auto {
    private String farbe;
                                    Rückgabetyp
    public String getFarbe() {
        return farbe;
    }                               Parameter

    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }
}
```

(2) Zugriffsbezeichner (Modifier)

Es gibt neben private (Sichtbarkeit nur innerhalb der Klasse) noch andere Zugriffsbezeichner für Klassen, Variablen und Methoden, und zwar public, protected und default. Der Zugriffsbezeichner default steht nicht explizit da, es steht statt dessen nichts.

Zugriffsbezeichner	**Zugriff auf Klassen, Methoden und Eigenschaften**
public	Zugriff von überall her möglich.
protected	Zugriff von Klassen im gleichen Package und nur durch Subklassen der Klasse in anderen Packages.
default oder package local (es steht nichts da)	Nur Zugriff von Klassen und Subklassen, innerhalb des gleichen Packages.
private	Nur Zugriff innerhalb der eigenen Klasse.

c) Konstruktoren

Um von Klassen, Objekte erzeugen zu können, brauchen wir Konstruktoren. Ein Konstruktor ist vergleichbar mit einem Autokonstrukteur, der ein Auto entwirft, das dann in vielfacher Ausfertigung produziert werden kann. Der Name des Konstruktors ist per Definition identisch mit dem Klassennamen, und er wird ohne Rückgabetyp deklariert, d. h. erstellt. Der Konstruktor darf auch nicht vom Rückgabetyp void sein. Konstruktoren können die Zugriffsbezeichner public, protected, package local und private haben. Eine Klasse kann mehrere Konstruktoren haben, die sich u. a. durch die übergebenen Parameter unterscheiden.

Beispiele für Konstruktoren:

1. Auto () { System.out.println(“Ich bin ein Konstruktor!“); }
2. Auto(String farbe) { System.out.println(“Ich bin ein Konstruktor, dem die Variable vom Typ String übergeben wird!“); }

Ist kein Konstruktor explizit definiert, wird automatisch ein impliziter Standard-

Konstruktor (default constructor) erstellt, der nicht explizit dasteht, sprich er ist unsichtbar. Dieser Konstruktor ist leer und enthält keine Parameter. Für unsere Klasse Auto wäre das der Standardkonstruktor Auto( ){ } (vgl. Kapitel Objektreferenz und Konstruktoraufruf mit this und super).

d) Konstanten

Konstanten sind Werte, die immer gleich sind. So ist z. B. die Zahl Pi eine bereits vordefinierte konstante Zahl. Will man eine Konstante definieren, muß der Zugriffsbezeichner final davor stehen.

4. Instanziierung von Objekten

Nehmen wir an, Sie wollen eine neue Instanz der Klasse Auto erstellen, so nennt sich dieser Vorgang Instanziierung. Oder anders ausgedrückt, es wird von einer Klasse Auto ein neues Objekt, sprich eine Kopie, mit dem Namen gruenesAuto unter Verwendung des Konstruktors, angefertigt. Der Prozeß der Instanziierung umfaßt zwei Schritte:

1. Ihrem neuen Objekt wird ein Name gegeben, und es wird festgelegt, von welcher Klasse das Objekt abgeleitet wird. Dieser Vorgang nennt sich Deklaration einer Referenzvariablen vom Typ Auto (vgl. Subtyping und Casting von Objekten), wobei der Name der Variablen, gruenesAuto, kleingeschrieben wird.

   ```
   Auto gruenesAuto;
   ```

2. Erstellung eines Objektes unter Verwendung des New-Operators.

   ```
   gruenesAuto = new Auto();
   ```

Man kann auch beide Schritte zu einem verbinden:

```
Auto gruenesAuto = new Auto();
```

Hier unsere erste Instanz der Klasse Auto in der Klasse Autohandel:

```
public class Autohandel {
    public static void main(String[ ] args) {
        //Neue Instanz der Klasse Auto
        Auto gruenesAuto = new Auto();
    }
}
```

Der Name der Variablen wird auch Objektreferenz genannt (vgl. Kapitel Garbage Collection).

5. Geltungsbereich von Variablen

a) Klassen- und Instanzvariablen

Instanzvariablen kennen wir bereits. Eine Instanzvariable war unsere Variable farbe, die sich für jedes Objekt Auto anpassen läßt. Im Gegensatz hierzu gibt es Klassenvariablen, die als static definiert werden, und die für jedes Objekt der Klasse Auto gleich sind. So könnte z. B. in unserer Klasse Auto eine Variable static String raeder = „4“; definiert werden.

```
public class Auto {
        //Instanzvariable
        private String farbe;
        //Klassenvariable
        static String raeder = “4”;

        public String getFarbe() {
                return farbe;
        }

        public void setFarbe(String farbe) {
                this.farbe = farbe;
        }
}
```

Die Eigenschaft farbe ist bei jeder Instanz der Klasse Auto anders, so hat jedes Auto eine andere Farbe, es gibt ein grünes Auto oder ein rotes Auto, aber jedes Auto hat vier Räder. Eine Klassenvariable bezieht sich auf die ganz Klasse und ist für die ganze Klasse gleich. Die Klassenvariable gibt es nur ein einziges Mal, auch wenn es mehrere Objekte gibt, so teilen sich alle Objekte einer Klasse die gleiche Klassenvariable, wohingegen jedes Objekt eine eigene Instanzvariable besitzt.

(1) Zugriff auf Instanz- und Klassenvariablen

Um auf statische Variablen oder Methoden zugreifen zu können, muß nicht ein Objekt instanziiert werden, es genügt mit Auto.raeder darauf zuzugreifen. Dies nennt man auch statischer Zugriff. Will man auf nicht statische Variablen oder Methoden zugreifen, muß man zuerst ein Objekt erstellen, auf dem man dann die entsprechenden Methoden ausführt. Auf Variablen, die private sind, kann man nur mit den entsprechenden Getter- und Setter-Methoden zugreifen.

```
public class Autohandel {
    public static void main(String[ ] args) {
        String farbe = “grün”;

        /*Zugreifen auf statische Variable (Klassenvariable),
        ohne Instanziierung eines Objektes Auto*/
        System.out.println(Auto.raeder);
```

```
            /*Zugreifen auf eine Instanzvariable nur
            mit Hilfe von vorheriger Instanziierung
            des Objektes Auto möglich*/
            Auto gruenesAuto = new Auto();
            gruenesAuto.setFarbe(farbe);
            System.out.print("Die Farbe des Auto ist: ");
            System.out.println(gruenesAuto.getFarbe());
        }
}
```

Ausgabe auf der Konsole:

```
4
Die Farbe des Autos ist: grün
```

Bitte beachten Sie, daß das Verwenden von statischen Variablen und Methoden der Objektorientierung widerspricht und deshalb nur sparsam eingesetzt werden sollte.

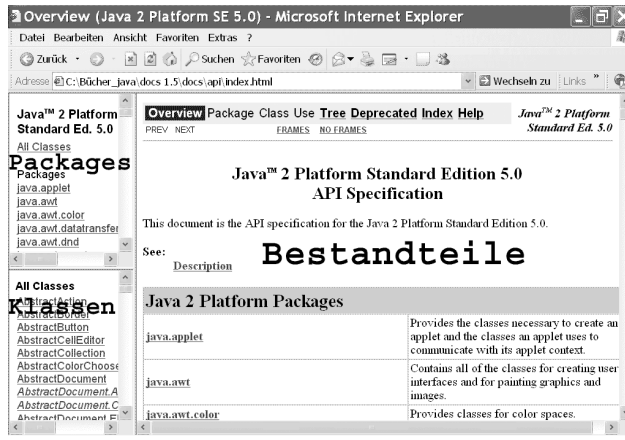
b) Lokale Variablen

Lokale Variablen sind Variablen, die weder für eine Klasse noch für ein Objekt gelten, sondern nur innerhalb einer Methode, eines Blocks oder einer for-Schleife. Lokale Variablen sind nur in dem Bereich bekannt, in dem sie definiert wurden. Von außerhalb können Sie auf diese Variablen nicht zugreifen (vgl. Kapitel „Die for-Schleife"), sprich der Geltungsbereich (scope) bezieht sich nur auf einen kleinen bestimmten Bereich.

6. Packages

Mehrere Klassen werden zu einem Package zusammengefaßt, da in Java Ordnung herrscht. So wird nicht alles durcheinander in einen Schrank geworfen, sondern alles wird wohlgeordnet in Schubladen verstaut. Diese Schubladen heißen Packages. Die Schachtel für unser Prüfungsaufgaben heißt z. B. scjp. So steht in einer Klasse ganz oben, zu welchem Package sie gehört. Und darunter stehen die importierten Packages, die bereits von anderen Programmierern erstellt worden sind und Ihnen zur Verfügung stehen. Eine Beschreibung dieser Packages befindet sich in den Javadocs (Api Documentation), die Sie unter http://java.sun.com/j2se/1.5.0/docs/index.html herunterladen können (vgl. Kapitel Wie arbeite ich mit Java und Eclipse?).

Es ist sinnvoll, von Zeit zu Zeit in dieser Dokumentation etwas nachzuschauen, um ausführliche Hintergrundinformationen über Klassen und Methoden zu erhalten. Um zur Startseite der Dokumentation zu gelangen, öffnen Sie bitte die index.html, nachdem Sie die Dokumentation heruntergeladen und entzippt haben, und klicken auf den Link Java 2 Platform API Specification. Die erste Seite sieht von der Aufteilung so aus, wie Sie es in untenstehender Abbildung sehen können, links befinden sich die Packages und Klassen und rechts können Sie den jeweils entsprechenden Inhalt sehen.



Das Package scjp ist ein Package, das ich erstellt habe und in das ich alle Klassen dieses Buches abgelegt habe. Wie Sie eigene Packages anlegen, können Sie im Kapitel Das erste Java-Programm in Eclipse nachlesen. So könnte z. B. in der Klasse Auto unseres Buches ganz oben, noch vor der Klassendefinition, Folgendes stehen: Das Package, in dem Ihre Klasse sich befindet und gleich danach alle importierten Klassen und Packages, die Sie zusätzlich in Ihrer Klasse brauchen.

```
//Package, in dem sich die Klasse Auto befindet
package scjp;

//importiertes Package, das Sie für Ihren Code brauchen
import java.util.*;

public class Auto { ...

                    }
```

Wollen Sie, wenn Sie sich in einer anderen Klasse eines anderen Packages befinden, die Klasse Auto importieren, gibt es zwei Möglichkeiten dies zu tun:

import scjp.*; oder import scjp.Auto;

Mit der ersten Variante importieren Sie alle Klassen des Packages scjp, mit der zweiten nur die Klasse Auto.

7. Übungen

(1) Frage

Welche dieser Aussagen über Konstruktoren trifft zu?

a) Gibt es in einer Klasse keinen Konstruktor, so wird automatisch einer durch den Kompiler erstellt.
b) Ein Konstruktor hat immer den Rückgabewert void.
c) Ein Konstruktor hat immer den Zugriffsbezeichner private.
d) Es darf nur einen Konstruktor geben.
e) Ein Konstruktor darf keinen Parameter haben.
f) Ein Konstruktor hat immer den Rückgabewert String.
g) Ein Konstruktor hat immer den Zugriffsbezeichner protected.

(2) Frage

Welche dieser Aussagen über Zugriffsbezeichner trifft zu?

a) Auf Methoden, die public sind, kann nur von ihrer eigenen Klasse zugegriffen werden.
b) Auf Methoden, die private sind, kann nur von ihrer eigenen Klasse zugegriffen werden.
c) Auf Methoden, die public sind, kann von überall zugegriffen werden.
d) Auf Methoden, die protected sind, kann nur von ihrer eigenen Klasse zugegriffen werden.
e) Auf Methoden, die package local sind, kann nur von ihrer eigenen Klasse zugegriffen werden.
f) Die Zugriffsbezeichner protected und package local bedeuten das gleiche.

(3) Frage

Welche der untenstehenden Codefragmente beinhaltet eine korrekte Implementierung eines Konstruktors?

a) public class Car{ Car void() { }; }
b) public class Car{ Car () { }; }
c) public class Car{ car () { }; }
d) public class Car{ Car (String anzahl) { }; }
e) public class Car{ Car String() { }; }
f) public class Car{ int Car () { }; }

(4) Frage

Welche der untenstehenden Aussagen treffen zu?

a) Der Packagename, in der sich die Klasse befindet, steht in einer Klasse ganz oben.
b) Der Packagename, in der sich die Klasse befindet, steht in einer Klasse ganz unten.
c) Die Packages, die in eine Klasse importiert werden, stehen ganz oben in der

Klasse.

d) Die Packages, die in eine Klasse importiert werden stehen gleich hinter dem Namen des Package.

e) Der Name des Package steht noch vor der Klassendefinition.

8. Lösungen

(1) Frage

a

(2) Frage

b, c

(3) Frage

b, d

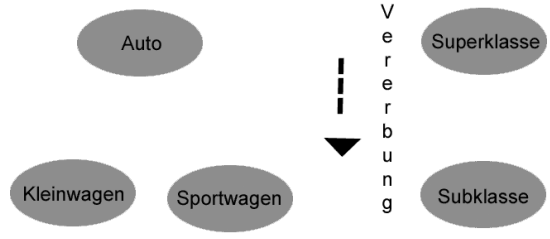
(4) Frage

a, d, e

B. Vererbung

1. Begriffserläuterung

Im Objektorientierten Programmieren gibt es ein Prinzip, das sich Vererbung nennt. Dieser Begriff wurde in Analogie zu der Vererbung in der Welt der Tiere und der Menschen gewählt. Ein Kind erbt von seinem Vater die Nase, mathematisches Grundverständnis oder musikalisches Empfinden. Wird an ein Kind die Musikalität von seinem Vater weitergegeben, kann es richtig singen oder leichter ein Musikinstrument erlernen. Vererbungsbeziehungen werden in Java als Relation zwischen Superklasse (Vaterklasse) und Subklasse (Kindklasse) abgebildet.



Erinnern wir uns an unsere Klasse Auto, die die Eigenschaft farbe hatte, die verändert werden konnte und wie folgt aussah:

```
public class Auto {
        private String farbe;

        public String getFarbe() {
                return farbe;
        }

        public void setFarbe(String farbe) {
                this.farbe = farbe;
        }
}
```

Unsere erste Subklasse soll eine Kindklasse zu unserer Klasse Auto sein. Diese Vater-Kind-Beziehung heißt is-a-Relationship und wird in der Klassendefinition mit dem Schlüsselwort extends gekennzeichnet. Es gilt zu beachten, daß eine Klasse per Definition nur eine extends-Beziehung haben kann.

```
public class Sportwagen extends Auto{

}
```

Wie Sie sehen können, ist unsere Subklasse Sportwagen noch ohne Inhalt. Im Moment stehen dieser Klasse nur Eigenschaften und Methoden zur Verfügung, die sie von der Superklasse Auto erbt. Das bedeutet, daß die Klasse Sportwagen von der Superklasse Auto die Eigenschaft farbe und die dazugehörigen Getter- und Settermethoden übernimmt. In untenstehendem Beispiel wird in der Klasse Autohandel sowohl eine Instanz der Klasse Auto als auch der Klasse Sportwagen erstellt, und wie Sie sehen können, kann man mit den Methoden der Klasse Auto, die Farbe sowohl des Autos als auch des Sportwagens aus- und wieder einlesen. Sie müssen also nur einmal diese Methoden definieren, und haben Sie dann in allen Subklassen zur Verfügung.

```
package scjp;

public class Autohandel {
	public static void main(String[ ] args) {
		String farbe = “grün”;
		String farbeSport = ”rot“;

		Auto gruenesAuto = new Auto();
		gruenesAuto.setFarbe(farbe);
		System.out.print(“Die Farbe des Auto ist grün: “);
		System.out.println(gruenesAuto.getFarbe());

		Sportwagen ferrari = new Sportwagen();
		ferrari.setFarbe(farbeSport);
		System.out.print(“Die Farbe des Sportwagens ist rot: “);
		System.out.println(ferrari.getFarbe());
	}
}
```

Ausgabe auf der Konsole:

```
Die Farbe des Autos ist grün: grün
Die Farbe des Sportwagens ist rot: rot
```

Unsere Klasse Sportwagen kann also die Methoden setFarbe(String farbe) und getFarbe() verwenden, da sie eine Kindklasse unserer Klasse Auto ist. In unserem Beispiel erbt die Klasse Sportwagen die Methoden der Klasse Auto.

2. Is-a-Relationship und Has-a-Relationship

Wie wir obenstehend gelernt haben verstehen wir Vater-Kind-Beziehungen als is-a-Relationship, wohingegen has-a-Relationship eine Aggregation, ein Zusammenbau, darstellt. So besitzt z. B. ein Auto Türen, Räder, Farbe und Sitze und man kann sagen, daß ein Auto einen Sitz hat („has-a“). Analog hierzu kann die Klasse Auto auch diese Bestandteile aufweisen.

3. Übungen

(1) Frage

Welche der untenstehenden Bemerkungen trifft zu?

a) Die Beziehung zwischen einer Superklasse und einer Subklasse nennt sich has-a-Relationship.
b) Dies ist eine korrekte Formulierung für eine Vererbungsbeziehung zwischen Klassen : public class Kaktus implements Pflanze { }.
c) Eine Superklasse erbt Methoden von der Subklasse.
d) Die Beziehung zwischen einer Superklasse und einer Subklasse nennt sich is-a-Relationship.
e) Eine Subklasse erbt Methoden von der Superklasse.
f) Dies ist eine korrekte Formulierung für eine Vererbungsbeziehung zwischen Klassen : public class Kaktus extends Pflanze { }.
g) In Java gibt es keine Vererbungsbeziehungen.
h) Besteht z. B. die Klasse Tisch aus Tischbeinen und einer Tischplatte, so handelt es sich um eine has-a-Relationship.

4. Lösungen

(1) Frage

d, e, f, h

C. Interfaces und abstrakte Klassen

1. Interfaces (Schnittstellen)

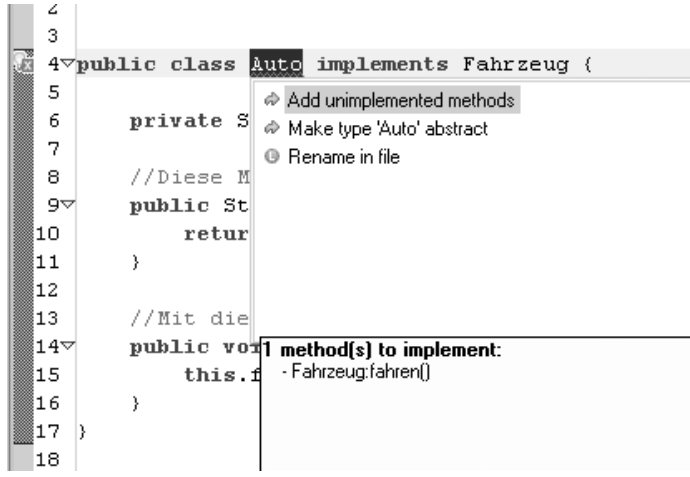
Bleiben wir bei unserem Beispiel Auto. Sie haben also eine Klasse Auto ausgearbeitet, und Sie stellen fest, Sie benötigen weitere Klassen, wie z. B. die Klasse Motorrad, Wohnwagen und Lastwagen. Da Sie nicht viel Zeit haben, keine Arbeit doppelt machen und außerdem auch nichts vergessen wollen, überlegen Sie sich, wie es sinnvoll wäre vorzugehen. Der erste Schritt wäre, diese Klassen unter einem Thema zusammenzufassen. Eine mögliche Kategorie könnte der Begriff Fahrzeug sein. Als nächstes machen Sie sich Gedanken über Gemeinsamkeiten, die alle Fahrzeuge haben. Sie sollten alle fahren, bremsen und blinken können. Will ich sichergehen, daß alle Klassen dies können, erstelle ich einen Vorschriftenkatalog. In diesem Fall ist es nicht sinnvoll eine Superklasse zu konzipieren, da die Methode fahren() in jeder Klasse wieder neu mit Inhalt gefüllt werden muß. Sie wollen nur kein Risiko eingehen, daß Ihr Auto für immer stehen bleiben muß, sprich jedes Fahrzeug muß eine Methode fahren() enthalten. Diese Funktion erfüllt ein Interface. Ein Interface stellt Regeln für Klassen zur Verfügung und ordnet sie nach Kategorien.

Wir erstellen im folgenden ein Interface Fahrzeug, das eine Methode fahren( ) vorschreibt. Diese Methode gilt als abstract, obwohl es nicht explizit dasteht. Abstract bedeutet, daß die Methode keinerlei Inhalt besitzt, sie ist leer. Leer besagt, daß die Methode erst zu einem späteren Zeitpunkt mit Programmcode gefüllt wird. Interfaces sind per Definition public und abstract. Da dies als selbstverständlich erachtet wird, werden diese zwei Schlüsselwörter in der Regel weggelassen.

```
package scjp;

public interface Fahrzeug {
    //Methoden in Interfaces sind implizit abstract und public.
    void fahren();
}
```

Will ich jetzt meiner Klasse Auto sagen, sie soll das Interface Fahrzeug implementieren, um zu garantieren, daß meine Klasse Auto auch eine Methode fahren( ) hat, muß ich meine Klasse um das Schlüsselwort implements und den Namen des Interfaces ergänzen. Sobald ich dies getan habe, bringt mir Eclipse sofort eine Fehlermeldung und einen Hinweis darüber, was ich tun soll (nachdem Sie am linken Rand auf das rote X geklickt haben).



Ich bekomme also die Empfehlung, eine Methode fahren() in meine Klasse Auto einzufügen. Es wird gewährleistet, daß mein Auto auf jeden Fall fahren kann, sie erinnern uns daran, nichts zu vergessen. Das Interface stellt zusätzlich sicher, daß die Methode fahren() in allen Klassen gleich heißt, um Konfusionen in bezug auf Methodennamen zu verhindern. Würde die Methode fahren in jeder Klasse unterschiedlich genannt werden, wie z. B. in der Klasse Auto autofahren(), in der Klasse Motorrad motorrad_fahren() oder in der Klasse Lastwagen fahrenLastwagen(), wäre die Gefahr groß, diese Namen zu vergessen und durcheinanderzubringen.

Werden nicht alle Methoden aus dem Interface in einer Klasse implementiert, muß diese Klasse wiederum als abstract deklariert werden. Des weiteren kann ein Interface ein anderes Interface mit dem Schlüsselwort extends ergänzen. Ein Interface und eine Klasse können durch mehrere Interfaces erweitert werden. Die Methoden innerhalb eines Interfaces sind implizit abstract und public, was normalerweise weggelassen wird bzw. weggelassen werden kann, und es darf dort nur abstract und public stehen und nichts anderes. Interfaces können zusätzlich Konstanten enthalten, die als implizit public, static und final gelten. Es besteht aber die Möglichkeit, daß keiner, einer oder zwei dieser Begriffe dastehen, ohne daß es zu einem Kompilierfehler führt. Es darf dort aber nichts anderes stehen.

Untenstehend unser Interface Fahrzeug mit einer Methode und einer Konstante.

```
package scjp;

public interface Fahrzeug {
    /*dies sind Konstanten, die implizit public,
    static und final sind*/
    double d = 3.14;
```

```
    //Methoden sind implizit abstract und public
    void fahren();
}
```

2. Abstrakte Klassen

Abstrakte Klassen haben eine ähnliche Funktion wie Interfaces. Der Unterschied zum Interface liegt allerdings darin, daß abstrakte Klassen nicht nur leere Methoden umfassen können, sondern zusätzlich auch ausformulierte, sprich sie können Inhalt besitzen. Abstrakte Klassen können nicht mit dem Schlüsselwort new instanziiert werden, obwohl sie einen Konstruktor haben, wir können nur auf die statischen Methoden einer abstrakten Klasse zugreifen. Der Konstruktor der abstrakten Klasse besteht also nur aus Implementierungsvorschriften. Hier ein Beispiel eines statischen Zugriffs auf die Methode getInstance() der abstrakten Klasse Calendar, die Ihnen verschiedene Kalenderfunktionen zur Verfügung stellt (vgl. Kapitel Datumsformat).

```
Calendar cal = Calender.getInstance();
```

Wir können mit abstrakten Klassen, wie auch mit Interfaces nur sicherstellen, daß in Klassen gewisse Methoden vorhanden sind. In einer Subklasse der abstrakten Klasse müssen die abstrakten Methoden im Gegensatz zu den ausformulierten implementiert werden. Des weiteren müssen Klassen als abstract gekennzeichnet werden, wenn sie eine als abstract deklarierte Methode enthalten. Abstrakte Klassen, die ein Interface implementieren, müssen nicht alle Methoden des Interfaces implementieren.

3. Übungen

(1) Frage

Welche der untenstehenden Begriffe können in Interfaces vor Methoden stehen?

a) protected
b) private
c) final
d) abstract
e) public
f) package local
g) static
h) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Aussagen sind korrekt?

a) Ein Interface darf keine Konstanten enthalten.
b) Ein Interface darf Methoden mit Methodenkörper enthalten.
c) Methoden in Interfaces müssen leer sein.
d) Eine abstrakte Klasse darf Methoden mit Methodenkörper enthalten.
e) Es heißt Interface A implements Interface B.
f) Methoden in Interfaces sind implizit private.
g) Enthält eine Klasse eine abstrakte Methode, muß die Klasse nicht abstrakt sein.
h) Methoden in Interfaces sind nie static.

(3) Frage

Welche der untenstehenden Aussagen sind korrekt?

a) Ein Interface darf eine extends-Beziehung zu mehreren Interfaces haben.
b) Eine abstrakte Klasse darf Methoden mit Methodenkörper enthalten.
c) Methoden in abstrakten Klassen müssen leer sein.
d) Eine Klasse darf Methoden ohne Methodenkörper enthalten.
e) Es heißt interface A extends interface B.
f) Methoden in Interfaces sind implizit public und abstract.
g) Enthält eine Klasse eine abstrakte Methode, muß die Klasse abstrakt sein.
h) Es heißt class a extends interface B.

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
        public byte            a = 1;                    Zeile 1
        public static final byte b = 1;                  Zeile 2
        byte c = 1;                                      Zeile 3
        long d = 5;                                      Zeile 4
        static int e = 1;                                Zeile 5
```

```
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
        protected short a = 2;          Zeile 1
        private byte b = 2;             Zeile 2
        short c = 2;                    Zeile 3
        long d = 2;                     Zeile 4
        int e = 2;                      Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(6) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
        public short        a = 25;     Zeile 1
        public final byte b = 25;       Zeile 2
        static int c = 25;              Zeile 3
        long d = 25;                    Zeile 4
        final int e = 21;               Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(7) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
```

```
        public void a1 ( );                    Zeile 1
        private void a2 ( );                   Zeile 2
        protected void a3 ( );                 Zeile 3
        abstract void a4 ( );                  Zeile 4
        final void a5 ( );                     Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(8) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
        public void a1 ( );                    Zeile 1
        void a2 ( );                           Zeile 2
        int a3 ( );                            Zeile 3
        abstract a4 ( );                       Zeile 4
        abstract public a5 ( );                Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(9) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
interface a {
        private void a1 ( );                   Zeile 1
        void a2 ( );                           Zeile 2
        int a3 ( );                            Zeile 3
        protected a4 ( );                      Zeile 4
        public a5 ( );                         Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

4. Lösungen

(1) Frage
d, e

(2) Frage
c, d, e, h

(3) Frage
a, b, e, f, g

(4) Frage
f

(5) Frage
a, b

(6) Frage
f

(7) Frage
b, c, e

(8) Frage
f

(9) Frage
a, d

D. Modifier

1. Allgemeines

Wie wir bereits in den vorangegangenen Kapiteln gelernt haben, können Klassen, Methoden und Variablen public, private, protected oder package local sein. Das Schlüsselwort private hat die Aufgabe vor unbefugtem Zugriff von außen zu schützen, wohingegen public gewährleisten soll, daß eine Klasse, als öffentlich zugängliches Modell für Objekte gilt (vgl. Kapitel Bestandteile von Klassen). Abstrakte Methoden sind leere Methoden, die erst später mit Inhalt gefüllt werden sollen, und konstante Werte, wie z. B. der Wert PI, müssen als final deklariert werden. Begriffe, die die Eigenschaften von Klassen, Methoden und Variablen näher definieren, nennt man Modifier. Nun wollen wir einige weitere Modifier und ihre Verwendungsmöglichkeiten kennenlernen.

a) native

Der Ausdruck native kann nur Methoden näher bestimmen. Native Methoden bestehen nicht aus Javacode, sondern aus Code einer anderen Programmiersprache, wie z. B. C oder C++.

b) transient

Transiente Variablen stehen in direkter Verbindung mit dem Thema Speicherung von Objekten (vgl. Kapitel Serialisierung von Objekten). Variablen, die als transient beschrieben werden, beinhalten Werte, die nicht serialisiert, sprich gespeichert werden sollen. Und nur Variablen können als transient deklariert werden.

c) synchronized

Die Verwendung des Modifiers synchronized steht in direkter Verbindung mit der Themenstellung Threads (vgl. Kapitel Threads). Eine Methode, die durch den Modifier synchronized näher bestimmt wird, soll bei der Verwendung von Threads sicherstellen, daß nur ein Thread Zugriff auf eine Methode hat, und nicht zwei Threads gleichzeitig. Der Modifier synchronized bezieht sich nur auf Methoden.

d) strictfp

Strictfp stellt bei Methoden, Interfaces und Klassen sicher, daß bei Nachkommaberechnungen immer das gleich Ergebnis herauskommt, sprich es soll zu keinen Rundungsfehlern kommen.

e) final

Den Begriff final haben wir bereits kennengelernt als Modifier für Werte, die nicht verändert werden dürfen, sprich Konstanten. Der Modifier final kann auch bei Klassen verwendet werden. Finale Klassen dürfen keine Subklassen haben, in denen die Methoden der Klasse überschrieben werden könnten. Dies stellt einen

unüberbrückbaren Gegensatz zum Modifier abstract dar, da eine finale Klasse nicht mehr verändert werden darf, eine abstrakte Klasse, aber noch verändert werden muß (vgl. Kapitel Interfaces und abstrakte Klassen)! Diese beiden Modifier schließen sich gegenseitig aus, sie widersprechen sich. Alle Methoden in einer final class sind implizit final und können somit auch nicht überschrieben werden.

2. Zusammenfassende Aufstellung der Modifier und ihrer Verwendung

Hier eine zusammenfassende Aufstellung der Modifier und ihrer Verwendung. Für einen Anfänger stellt die Verinnerlichung der Regeln für den Einsatz der verschiedenen Modifier eine große Herausforderung dar. Aber keine Sorge in den folgenden Kapiteln wird immer wieder Bezug auf dieses Kapitel genommen und die Thematik an anderer Stelle vertieft.

Konstruktoren	Interfaces	Variablen	Methoden	Klassen
• private • protected • public • package local (default)	• per Definition abstract und public • strictfp	• transient • final • private • protected • public • package local (default) • static	• private • protected • public • static • abstract • final • native • synchronized • strictfp	• strictfp • public • package local (default) • final • abstract

a) Verwendung der Modifier in Interfaces

Erlaubte Modifier in Interfaces	
für Methoden	**für Konstanten**
• abstract • public	• public • static • final

3. Übungen

(1) Frage

Welche der untenstehenden Modifier für Methoden können in Interfaces verwendet werden?

a) protected
b) private
c) final
d) abstract
e) public
f) package local
g) static
h) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Modifier für Konstanten können in Interfaces verwendet werden?

a) protected
b) private
c) transient
d) abstract
e) strictfp
f) package local
g) native
h) Keine dieser Möglichkeiten.

(3) Frage

Welche der untenstehenden Modifier für Konstanten in Interfaces können verwendet werden?

a) transient
b) private
c) final
d) abstract
e) public
f) package local
g) native
h) Keine dieser Möglichkeiten.

(4) Frage

Welche der untenstehenden Modifier können für Konstruktoren verwendet werden?

a) protected
b) private

c) final
d) abstract
e) public
f) transient
g) native
h) Keine dieser Möglichkeiten.

(5) Frage

Welche der untenstehenden Modifier können für Variablen verwendet werden?

a) native
b) private
c) synchronized
d) abstract
e) public
f) protected
g) strictfp
h) Keine dieser Möglichkeiten.

(6) Frage

Welche der untenstehenden Bemerkungen trifft nicht zu?

a) Eine abstrakte Klasse kann instanziiert werden.
b) Die Modifier abstrakt und final schließen sich gegenseitig aus.
c) Von einer final class darf es Subklassen geben.
d) Alle Methoden in einer final class sind implizit final.
e) Nativer Code enthält auf jeden Fall Javacode.
f) Es gibt synchronized Methoden und Klassen.

(7) Frage

Welche der untenstehenden Deklarationen für Klassen und Interfaces sind möglich?

a) abstract final class a { }
b) final public class a { }
c) private interface a { }
d) abstract class a { }
e) abstract public interface a { }
f) protected interface a { }
g) interface a{ }

(8) Frage

Welche der untenstehenden Deklarationen für Methoden und Variablen sind nicht möglich?

a) abstract final void m ();
b) public void m();

c) abstract int a = 34;
d) native int a = 34;
e) native void m();
f) private int a = 34;

4. Lösungen

(1) Frage

d, e

(2) Frage

h

(3) Frage

c, e

(4) Frage

a, b, e

(5) Frage

b, e, f

(6) Frage

a, c, e, f

(7) Frage

b, d, e, g

(8) Frage

a, Abstrakt und final ist ein Widerspruch.
c, Abstrakte Variablen gibt es nicht, nur abstrakte Methoden und Klassen.
d, Native Variablen gibt es nicht, sondern nur native Methoden.

E. Überladen und Überschreiben von Methoden

1. Überladen von Methoden

Nehmen wir an, Sie wollen feststellen, welche von zwei Zahlen die größere ist. Dies wollen Sie nicht nur für ganze Zahlen wissen, sondern auch für Zahlen mit Dezimalstellen. In Programmiersprachen werden Zahlen, je nach Größe, in Kategorien, Datentypen, eingeteilt (vgl. Kapitel Zahlen und Buchstaben in Java: primitive Datentypen und Variablendeklarationen). So gibt es unterschiedlich große Schachteln für Zahlen, für kleine Zahlen gibt es kleine Schachteln und für große Zahlen gibt es große Schachteln. Nehmen Sie aus einer großen Schachtel eine Zahl, so paßt diese nicht in eine kleine Schachtel. Die Zahl eine Million ist zu groß für eine Schachtel, in die man nur Zahlen bis 100 legen darf. Zahlen mit Nachkommastellen passen nicht in eine Schachtel, die nur für ganzzahlige Werte vorgesehen sind. Es ist nicht ausreichend Platz für sie da. Wenn Sie versuchen eine Kommazahl in diese kleinere Schachtel zu zwängen, führt dies zu einem Kompilierfehler. Das gleiche Prinzip gilt für Zigarettenautomaten, es können nur Münzen eingeworfen werden und keine Geldscheine. Also keine Münzen, keine Zigaretten. Diese Schachteln haben unterschiedliche Namen, die unter dem Sammelbegriff Datentyp zusammengefaßt werden. Ganzzahlige Werte können vom Typ long oder int sein und Zahlen mit Dezimalstellen vom Typ double oder float.

Nehmen wir an, Sie wollen das Maximum von 4 oder 5 und von 4.524 oder 5.5478 herausfinden. Für diesen Zweck würde es nicht ausreichen, wenn es nur eine einzige Maximum-Methode gäbe. Unterschiedliche Datentypen, verlangen nach unterschiedlichen max()-Methoden, sogenannten überladenen Methoden. Im untenstehenden Schaubild sehen sie einen Ausschnitt aus der Klasse java. lang.Math und die dazugehörigen vier überladenen max()-Methoden. Diese Methoden ermitteln den größten Wert von zwei übergebenen Zahlen. Überladene Methoden haben identische Namen und befinden sich in der gleichen Klasse, sie unterscheiden sich aber durch den Datentyp der übergebenen Parameterliste und den Rückgabewert.

static double Rückgabetyp	max(double a, double b) Parameter Returns the greater of two double values.
static float	max(float a, float b) Returns the greater of two float values.
static int	max(int a, int b) Returns the greater of two int values.
static long	max(long a, long b) Returns the greater of two long values.

Diese überladenen Methoden sind dazu da, daß für Zahlen mit unterschiedlichem Datentyp ein maximaler Wert berechnet werden kann. Es wird für ganzzahlige Werte und für Fließkommazahlen der maximale Wert herausgefunden. Fließkommazahlen müssen in amerikanischen Zahlenformat eingegeben werden, mit Punkt statt Komma als Trennzeichen.

```
public class maxWert {
    public static void main(String[ ] args) {
        //Maximum von zwei Ganzzahlen (int-Werten)
        System.out.println(Math.max(4, 5));
        //Maximum von zwei Dezimalzahlen (double-Werten)
        System.out.println(Math.max(4.524, 5.5478));
    }
}
```

Ausgabe auf der Konsole:

```
5
5.5478
```

In Java ist es möglich, innerhalb der gleichen Klasse Methoden zu überladen, indem mehrere Methoden mit dem gleichen Namen deklariert, ihnen aber unterschiedliche Parameterlisten zugewiesen werden. Beim Überprüfen der überladenen Methoden achtet der Kompiler auf die Anzahl, den Datentyp und die Reihenfolge der Parameter, aber nicht auf die Namen der Parameter oder auf den Rückgabetyp der Methode. Überladenen Methoden können sowohl privater als auch öffentlicher in bezug auf ihre Sichtbarkeit werden.

Beispiel für überladene Methoden, die zu Kompilierfehlern führen:

```
void m1(long a, int b);
long m1(long x, int y);
```

Diese zwei Methoden unterscheiden sich zwar durch den Rückgabewert (void und long), aber nicht durch die Anzahl und den Datentyp der Parameter, die beides Mal long und int sind. Hierbei spielen unterschiedliche Namen für die Parameter keine Rolle. Es würde genügen die Parameter zu vertauschen, um einen Kompilierfehler zu verhindern.

```
long m1(int a, long b);
```

Weitere Möglichkeiten für überladene Methoden, die auch nicht zu Kompilierfehler führen, ist das Ändern der Anzahl der Parameter:

```
long m1(int a);
long m1();
```

Im ersten Fall haben Sie statt zwei Parameter nur noch einen und im zweiten Fall gar keinen mehr.

2. Übungen

(1) Frage

Bei welcher der überladenen Methoden kommt es zu einem Kompilierfehler?

```
public class {
        public long methodA(long a, int b) { }          Zeile 1
        public long methodA(long a, long b) { }         Zeile 2
        public long methodA(long a) { }                 Zeile 3
        public long methodA(int a) { }                  Zeile 4
        public long methodA( ) { }                      Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(2) Frage

Bei welcher der überladenen Methoden kommt es zu einem Kompilierfehler?

```
public class {
        public long methodA(long a, long b) { }         Zeile 1
        public int methodA(long a, long b) { }          Zeile 2
        public long methodA(long a, int b) { }          Zeile 3
        public long methodA(int a, long b) { }          Zeile 4
        public long methodA( ) { }                      Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(3) Frage

Bei welcher der überladenen Methoden kommt es zu einem Kompilierfehler?

```
public class {
        public int methodA(long a, int b) { }           Zeile 1
        public int methodA(long x, int y) { }           Zeile 2
        public int methodA(int b, long a) { }           Zeile 3
        public long methodA(int a) { }                  Zeile 4
        public long methodA( ) { }                      Zeile 5
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3

d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(4) Frage

Welche der überladenen Methoden können an der gekennzeichneten Stelle eingefügt werden, ohne daß es zu einem Kompilierfehler kommt.

```
public class {
        public double methodA(double a) { }
        Einfügen einer weiteren Methode}
```

a) public long methodA(double a) { }
b) public double methodA(long a) { }
c) public double methodA(double a, double b) { }
d) public double methodA(float a) { }
e) Keine dieser Möglichkeiten.

(5) Frage

Welche der überladenen Methoden können an der gekennzeichneten Stelle eingefügt werden, ohne daß es zu einem Kompilierfehler kommt.

```
public class {
        public double methodA(double a, double b) { }
        Einfügen einer weiteren Methode}
```

a) public long methodA(double x, double y) { }
b) public double methodA(long a) { }
c) public int methodA(double a, double b) { }
d) public double methodA(double a) { }
e) Keine dieser Möglichkeiten.

3. Lösungen

(1) Frage

f

(2) Frage

b

(3) Frage

b

(4) Frage

b, c, d

(5) Frage

b, d

4. Überschreiben von Methoden

Wir haben im Kapitel Vererbung gelernt, daß Methoden von der Superklasse an die Subklasse vererbt werden, genauso wie Musikalität vom Vater zum Sohn weitergegeben wird. So hatte unser Sportwagen auch Zugriff auf die Getter- und Setter-Methoden der Vaterklasse Auto. Wir konnten die Farbe des Sportwagens verändern, obwohl in der Klasse Sportwagen keine entsprechende Methode vorhanden war. Nun sind aber Autos und Sportwagen nicht immer gleich. Genauso wie der Vater eine musische Begabung zwar an seinen Sohn weitervererbt, diese aber bei seinem Sohn lange nicht so ausgeprägt ist. So könnte die Klasse Auto, die Methode fahren() und die Klasse Sportwagen die gleichnamige, aber abgeänderte, Methode besitzen. Die Methode fahren() der Klasse Sportwagen würde im Gegensatz zur Methode fahren() der Klasse Auto schnelles statt langsames Fahren beinhalten. Die Methode fahren() von der Kindklasse Sportwagen wäre geringfügig zu verändern. Diesen Vorgang nennt man in einer Objektorientierten Programmiersprache Überschreiben der Methode fahren() der Klasse Auto in der Klasse Sportwagen.

Nehmen wir nun an, es gäbe noch eine weitere Subklasse der Klasse Auto, die Klasse Golf. Nun wird die Methode fahren() in der Klasse Golf nicht überschrieben, da es für einen Golf vollkommen ausreichend ist, langsam zu fahren. Nur in der Subklasse Sportwagen wird die fahren()-Methode mit neuem Inhalt gefüllt, weil ein Sportwagen schneller fahren kann als ein normales Auto. Für die Klasse Golf ist im Gegensatz zur Klasse Sportwagen die ererbte Methode fahren() vollkommen ausreichend. Sie denken jetzt sicher, daß es ein großer Nachteil ist, gleichnamige Methoden zu haben, das Gegenteil ist der Fall. Es ist viel einfacher, sich Methoden mit gleichem Namen zu merken. Je weniger unterschiedliche Namen in Benutzung sind, desto größer ist die Wahrscheinlichkeit, daß man sie nicht vergißt.

Wie können wir dies nun in Programmiercode umsetzen? Lassen Sie uns hierfür ein neues Beispiel mit zwei neuen Klassen erstellen: die Klasse Vater mit der Methode void laufen() und eine leere Klasse Sohn.

```
public class Vater {
        void laufen() {
            System.out.println("Ich bin der Vater!");
        }
}
```

Die Klasse Sohn weist eine Vererbungsbeziehung extends mit der Klasse Vater auf.

```
public class Sohn extends Vater{

}
```

Im nächsten Schritt entwerfen wir eine Klasse Familie und instanziieren jeweils ein Objekt Vater und ein Objekt Sohn. Führen wir für beide Objekte die Methode laufen() durch, stellen wir fest, daß wir die gleiche Ausgabe erhalten, nämlich „Ich bin der Vater!“.

```
public class Familie {
	public static void main(String[ ] args) {
		Vater neuerVater = new Vater();
		Sohn neuerSohn = new Sohn();

		neuerVater.laufen();
		neuerSohn.laufen();
	}
}
```

Ausgabe auf der Konsole:

```
Ich bin der Vater!
Ich bin der Vater!
```

Der Sohn erbt also die Methode laufen() vom Vater und sagt nun, wie der Vater während er läuft, daß er der Vater sei. Hätte der Sohn keine Vererbungsbeziehung, wäre die Methode laufen in der Klasse unbekannt, und es würde zur Laufzeit eine Exception geworfen werden (vgl. Kapitel Exceptions).

Wollen wir nun, daß der Sohn über eine eigene Methode laufen() verfügt, und er etwas anderes sagt als der Vater, müssen wir eine neue Methode laufen() in der Klasse Sohn erstellen. Diesen Vorgang nennt man Überschreiben der Methode laufen():

```
public class Sohn extends Vater{
	void laufen(){
		System.out.println(“Ich bin der Sohn!”);
	}
}
```

Nun sieht unser Ergebnis der Klasse Familie ganz anders aus. Es erscheinen auf der Konsole zwei unterschiedliche Sätze „Ich bin der Vater!“ und „Ich bin der Sohn!“.

```
public class Familie {
	public static void main(String[ ] args) {
		Vater neuerVater = new Vater();
		Sohn neuerSohn = new Sohn();

		neuerVater.laufen();
		neuerSohn.laufen();
```

```
    }
}
```

Ausgabe auf der Konsole:

```
Ich bin der Vater!
Ich bin der Sohn!
```

Die Methode laufen() in der Klasse Sohn überschreibt die gleichnamige Methode der Klasse Vater. Die Methode laufen() in der Klasse Vater wird verdeckt (hidden). Die überschreibende Methode muß den gleichen Namen, die gleichen Parameter und den gleichen Rückgabetyp wie die Originalmethode haben. Wird der Rückgabewert verändert, ohne daß sich der Parameter oder der Name ändert, kommt es zu einem Kompilierfehler. Die Methode, die überschreibt, darf die Zugriffsrechte der ursprünglichen Methode nicht einschränken, sondern sie müssen mindestens gleich bleiben oder erweitert werden, d. h., ich kann gleich öffentlich bleiben oder ich muß öffentlicher werden, andernfalls kommt es zu einem Kompilierfehler. So darf eine protected Methode mit einer public oder einer protected Methode überschrieben werden, aber nicht mit einer Methode, die private ist. Methoden, die static, final oder private sind, dürfen nicht überschrieben werden.

5. Übungen

(1) Frage

Welche der untenstehenden Bemerkungen treffen zu?

a) Private Methoden können überschrieben werden.
b) Final Methoden können überschrieben werden.
c) Static Methoden können überschrieben werden.
d) Keine Methode einer Superklasse kann in der Subklasse überschrieben werden.
e) Die Begriffe Überschreiben und Überladen können synonym verwendet werden.
f) Ich muß beim Überladen von Methoden öffentlicher werden.
g) Ich muß Methoden aus der Superklasse in der Subklasse implementieren.
h) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Bemerkungen treffen zu?

a) Eine Subklasse erbt Methoden aus der Superklasse.
b) Final Methoden dürfen nicht überschrieben werden.
c) Methoden aus der Superklasse müssen in der Subklasse implementiert werden.
d) Methoden einer Superklasse können in der Subklasse überschrieben werden.
e) Die neue Methode muß den gleichen Namen, die gleichen Parameter und den gleichen Rückgabetyp haben.
f) Ich darf beim Überladen von Methoden nicht öffentlicher werden.
g) Ich darf eine protected Methode mit einer public Methode überschreiben.
h) Keine dieser Möglichkeiten.

(3) Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter(){
void f() { System.out.println(”M“) }}
public class Tochter extends Mutter() {}
public class Familie{
Mutter m = new Mutter();
Tochter t = new Tochter();
m.f();
t.f();
}
```

a) MM
b) M
c) Nichts
d) Es wird eine Exception geworfen.

(4) Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter(){
void f() { System.out.println(”M“) }
}
public class Tochter () {
}
public class Familie{
Mutter m = new Mutter();
Tochter t = new Tochter();
m.f();
t.f();
}
```

a) MM
b) M
c) Nichts
d) Es wird eine Exception geworfen.

(5) Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter(){
void f() { System.out.println(”M“) }}
public class Tochter extends Mutter() {
void f() { System.out.println(”T“) }}
public class Familie{
Mutter m = new Mutter();
Tochter t = new Tochter();
m.f();
t.f();
}
```

a) MT
b) TM
c) M
d) T
e) Nichts
f) Es wird eine Exception geworfen.

(6) Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter(){
void f() { System.out.println(”M“) }
}
public class Tochter () {
void f() { System.out.println(”T“) }
}
public class Familie{
```

```
Mutter m = new Mutter();
Tochter t = new Tochter();
m.f();
t.f();
}
```

a) MT
b) TM
c) M
d) T
e) Nichts
f) Es wird eine Exception geworfen.

(7) Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, darf in der Klasse Tochter stehen, ohne daß es zu einem Kompilierfehler kommt ?

```
public class Mutter(){
void f() { System.out.println("M") }
}
public class Tochter extends Mutter() {
Welche Methode darf hier stehen?}
```

a) private void f( ){ }
b) public void f( ){ }
c) protected void f( ){ }
d) int f( ) {}
e) Keine dieser Möglichkeiten.

(8) Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, darf in der Klasse Tochter stehen, ohne daß es zu einem Kompilierfehler kommt ?

```
public class Mutter(){
public void f() { System.out.println(„M") }
}
public class Tochter extends Mutter() {
Welche Methode darf hier stehen?
}
```

a) private void f( ){ }
b) public void f( ){ }
c) protected void f( ){ }
d) int f( ) {}
e) Keine dieser Möglichkeiten.

(9) Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, darf in der Klasse Tochter stehen, ohne daß es zu einem Kompilierfehler kommt ?

```
public class Mutter(){
private void f() { System.out.println(”M“) }
}
public class Tochter extends Mutter() {
Welche Methode darf hier stehen?
}
```

a) final void f( ){ }
b) static void f( ){ }
c) void f( ){ }
d) string f( ) {}
e) Keine dieser Möglichkeiten.

6. Lösungen

(1) Frage

h

(2) Frage

a, d, g

(3) Frage

a

(4) Frage

d, Es kommt zu einer Exception, da eine extends-Beziehung zwischen den zwei Klassen fehlt.

(5) Frage

a

(6) Frage

f, Es kommt zu einer Exception, da eine extends-Beziehung zwischen den zwei Klassen fehlt.

(7) Frage

b, c

(8) Frage

b

(9) Frage

e

F. Objektreferenz und Konstruktoraufruf mit this und super

1. Objektreferenz mit this und super

a) Objektreferenz mit super

Erinnern wir uns an die Klasse Auto und die Klasse Sportwagen, beide hatten eine Methode fahren(). Fahren in der Klasse Auto bedeutete langsam fahren und in der Klasse Sportwagen schnell fahren. Für einen Sportwagen war es unmöglich langsamer zu fahren. Diesen Mangel wollen wir nun beheben. Der Sportwagen soll auch gemütlich fahren können (vgl. Kapitel Überladen und überschreiben von Methoden). Oder unser Sohn soll auch sagen können „Ich bin der Vater!“ und nicht nur „Ich bin der Sohn!“. Wir haben also zum Ziel, auf die Methode laufen() der Klasse Vater zuzugreifen, ohne die Methode laufen() in der Klasse Sohn ändern zu müssen. Dies geht nur mit der Objektreferenz super. Super wird in der Subklasse verwendet, um Methoden der Superklasse aufzurufen. Wir ergänzen unsere Klasse Sohn wie folgt:

```
public class Sohn extends Vater{
	void laufen(){
		System.out.println("Ich bin der Sohn!");
		super.laufen();
	}
}
```

Nun sieht die Ausgabe in der Klasse Familie wie folgt aus:

```
public class Familie {
	public static void main(String[ ] args) {
		Sohn neuerSohn = new Sohn();

		neuerSohn.laufen();
	}
}
```

Ausgabe auf der Konsole:

```
Ich bin der Sohn!
Ich bin der Vater!
```

Die Methode laufen() in der Klasse Sohn, ruft mit dem Befehl super.laufen() zusätzlich die Methode laufen() der Klasse Vater auf.

b) Objektreferenz mit this

Rufen wir uns unsere Klasse Auto ins Gedächtnis, dieses Auto war entweder rot oder rosa. Die Eigenschaft farbe nannte sich Instanzvariable und sie war für jedes Objekt anders.

Instanzvariable

```
private String farbe;
```

Die Eigenschaft Farbe hatten wir verändert mit der Methode setFarbe(String farbe), der wir den Parameter String farbe übergeben haben.

Parameter

```
public void setFarbe(String farbe) {
    this.farbe = farbe;
}
```

Um diesen Parameter von der Instanzvariablen zu unterscheiden, gibt es das Wort this (vgl. Kapitel Variablen). So bezieht sich this.farbe auf die Instanzvariable und farbe bezieht sich auf den übergebenen Parameter. Es wird also der Instanzvariablen der Parameterwert farbe zugewiesen. Beachten Sie bitte, daß eine Objektreferenz mit this in statischen Methoden nicht möglich ist.

```
public class Auto {

    private String farbe;
    static byte raeder = 4;

    //Diese Methode zeigt Ihnen die Farbe an
    public String getFarbe() {
        return farbe;
    }

    //Mit dieser Methode können Sie die Farbe verändern
    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }
}
```

2. Konstruktoraufruf mit this und super

a) Konstruktoraufruf mit super()

Der Konstruktor war der Bauplan, der es uns möglich machte, Kopien von einer Klasse anzufertigen (vgl. Kapitel Klassen und Objekte). Bis jetzt blieb uns die Arbeit erspart, der Klasse einen Konstruktor hinzufügen zu müssen, da diese Arbeit für uns Java übernommen hat. Es wird der Klasse automatisch ein impliziter Standardkonstruktor hinzugefügt, wobei implizit in diesem Zusammenhang unsichtbar bedeutet. Nehmen wir unsere Klasse Vater, die mit einem sichtbar gemachten Standardkonstruktor (Vater(){}), wie folgt aussehen würde:

```
public class Vater{
        Vater(){}
}
```

In jeder Subklasse steht ebenfalls ein impliziter Standardkonstruktor, der implizit den Aufruf super() enthält. Mit super() wird der Standardkonstruktor der Vaterklasse aufgerufen. Hier ein Beispiel des Standardkonstruktors mit super() in unserer Klasse Sohn:

```
public class Sohn extends Vater{
    Sohn(){
        super();
    }
}
```

Weder in der Vater- noch in der Sohnklasse steht der Standardkonstruktor normalerweise explizit da, so sieht die Subklasse normalerweise wie unten aus:

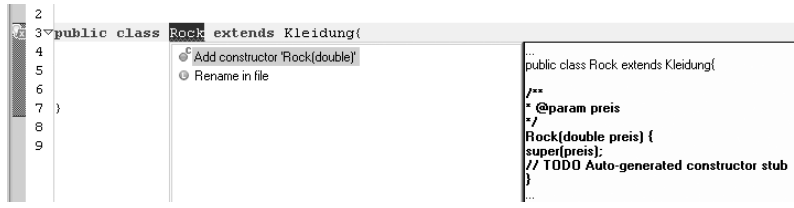
```
public class Sohn extends Vater{

}
```

Im folgenden ist unser Ziel, die Handhabung der Konstruktoren näher zu betrachten. Dies wollen wir anhand einer neuen Klasse Kleidung tun. Diese Klasse enthält einen Konstruktor, dem ein Parameter Preis übergeben wird. Fügen wir der Klasse einen Konstruktor Kleidung (double preis) hinzu, hat dies zur Folge, daß es keinen Standardkonstruktor mehr gibt. Dadurch kommt es an mehreren Stellen zu Kompilierfehlern. Will man Fehler vermeiden, sollte man in diesem Fall immer, zusätzlich einen Standardkonstruktor erstellen. Sollten Sie die folgenden Ausführungen verwirren, ist dies nur verständlich, da es schwierig ist, sich vorzustellen, wie ein unsichtbarer Konstruktor gelöscht werden kann.

```
public class Kleidung {
	Kleidung(double preis){
		System.out.println("Ich koste " + preis);
	}
}
```

Wird der eliminierte Standardkonstruktor nicht ersetzt, erhalten wir in der Subklasse Rock sofort eine Fehlermeldung (klicken Sie bitte links auf das rote Kreuz). Eclipse gibt Ihnen einen Hinweis, wie Sie diesen Fehler beheben können, Sie sollen in der Subklasse einen Konstruktor definieren. Dieser Vorschlag basiert auf dem Konstruktor, Rock (double preis), den wir erstellt haben und nicht auf dem Standardkonstruktor, da dieser nicht mehr existiert. Als wir den Konstruktor mit dem Parameter preis hinzugefügt haben, wurde gleichzeitig der Standardkonstruktor der Klasse Kleidung gelöscht.



Folgen wir den Anweisungen von Eclipse, erstellt er uns folgenden Konstruktor, der einen Konstruktoraufruf mit super(preis) enthält. Der Aufruf super() muß im Konstruktor immer ganz oben stehen, und er muß die gleichen Parameter, hier den Preis, enthalten.

```
public class Rock extends Kleidung{
	Rock(double preis) {
		super(preis);
	}
}
```

Würden wir nun in der Klasse Rock einen Standardkonstruktor hinzufügen, würde dies zu einem Kompilierfehler führen, da der Konstruktor Rock (){ } ganz oben implizit den Konstruktoraufruf super() enthält. Super() ruft den Standardkonstruktor der Klasse Kleidung auf, den gibt es aber nicht mehr, so landet der Aufruf im Nichts.

```
 2
 3 public class Rock extends Kleidung{
 4
 5                    Implicit super constructor Kleidung() is undefined. Must explicitly invoke another constructor
 6     Rock(){
 7
 8     }
 9
10     Rock(double preis) {
11         super(preis);
12     }
13 }
```

Es gibt jetzt zwei Möglichkeiten dieses Problem zu lösen, entweder wir schreiben in den Konstruktor Rock () {} super(preis) und rufen so den vorhandenen Konstruktor auf, oder wir ergänzen in der Klasse Kleidung einen Standardkonstruktor. Wobei letztere Lösung immer vorzuziehen ist, und dies wollen wir im folgenden auch tun:

```
public class Kleidung {
        Kleidung(){
                System.out.println("Ich bin ein Kleidungsstück");
        }

        Kleidung(double preis){
                System.out.println("Ich koste " + preis);
        }
}
```

Unter diesen Umständen kommt es zu keiner Fehlermeldung mehr. Wir haben den Konstruktor, den wir gelöscht haben, wiederhergestellt.

Wird jetzt ein Objekt der Klasse Rock instanziiert, wird automatisch der Konstruktor Kleidung(){ System.out.println(„Ich bin ein Kleidungsstück"); } der Vaterklasse aufgerufen. Es kommt also zur Ausgabe von „Ich bin ein Kleidungsstück", da im Konstruktor Rock ( ){ } ganz oben implizit super() steht, es wird also zuerst der Standardkonstruktor der Superklasse aufgerufen und ausgeführt und dann erst der Konstruktor der Subklasse. Der Konstruktor wird also implizit oder explizit aufgerufen und nicht vererbt.

```
public class Rock extends Kleidung{
        //Im Standardkonstuktor steht implizit super();
        Rock(){

        }

        Rock(double preis) {
                super(preis);
}
```

```
        public static void main(String[ ] args) {
                Rock rock = new Rock();
        }
}
```

Ausgabe auf der Konsole:

Ich bin ein Kleidungsstück!

Erstellen wir eine Instanz der Klasse Rock mit dem Konstruktor Rock(double Preis){ ... } erhalten wir untenstehende Ausgabe, da dieser Konstruktor den Konstruktor Kleidung(double preis) { ... } mit dem Aufruf super(preis) aus der Vaterklasse aufruft:

```
public class Rock extends Kleidung{
        Rock(){
        }

        Rock(double preis) {
                super(preis);
        }

        public static void main(String[ ] args) {
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

Ich koste 2.2

Ändern wir die Klasse Rock, indem wir in den Konstruktor Rock(preis){ ... } die Ausgabe eines Satzes hinzufügen, erhalten wir untenstehende Ausgabe. Richten Sie Ihr Augenmerk darauf, daß der Superaufruf immer ganz oben in dem Konstruktor stehen muß, ansonsten kommt es zu einem Kompilierfehler.

```
public class Rock extends Kleidung{
        Rock(){
        }

        Rock(double preis) {
                super(preis);
                System.out.println("Ich bin ein Rock");
        }

        public static void main(String[ ] args) {
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

Ich koste 2.2
Ich bin ein Rock

Im Standardkonstruktor der Kindklasse steht immer implizit super(). Wie Sie in untenstehendem Beispiel sehen können, hat dies genau den gleichen Effekt, als ob ich super () hinschreibe. Der Konstruktoraufruf super() muß immer ganz oben im Konstruktor stehen. Er darf sich an keiner anderen Stelle befinden, auch nicht in der main()-Methode, da dies zu einem Kompilierfehler führen würde.

```
public class Rock extends Kleidung{
        Rock(){
                super();
        }

        Rock(double preis) {
                super(preis);
                System.out.println("Ich bin ein Rock");
        }

        public static void main(String[ ] args) {
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

Ich koste 2.2
Ich bin ein Rock

Steht im Standardkonstruktor der Klasse Rock allerdings super(3.2), wird der Konstruktor der Vaterklasse mit dem Parameter Preis aufgerufen. Der Konstruktoraufruf mit dem Parameter preis, führt dazu, daß der Konstruktoraufruf super() nicht mehr existiert.

```
public class Rock extends Kleidung{
        Rock(){
                super(3.2);
        }

        Rock(double preis) {
                super(preis);
                System.out.println("Ich bin ein Rock");
        }

        public static void main(String[ ] args) {
                Rock rock = new Rock();
```

```
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

```
Ich koste 3.2
Ich koste 2.2
Ich bin ein Rock
```

Sieht der Standardkonstruktor allerdings wie in untenstehendem Beispiel aus, wird weiterhin als erstes super() ausgeführt. Der Aufruf super() wird in diesem Fall nicht überschrieben, sondern steht weiterhin implizit dort.

```
public class Rock extends Kleidung{
        Rock(){
                System.out.println("Ich bin ein neuer Rock");
        }

        Rock(double preis) {
                super(preis);
                System.out.println("Ich bin ein Rock");
        }

        public static void main(String[ ] args) {
                Rock rock = new Rock();
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

```
Ich bin ein Kleidungsstück
Ich bin ein neuer Rock
Ich koste 2.2
Ich bin ein Rock
```

b) Konstruktoraufruf mit this()

Beabsichtigen Sie einen Konstruktor der Klasse, in der Sie sich befinden und nicht einen Konstruktor der Superklasse aufzurufen, müssen Sie den Konstruktoraufruf this() verwenden. Der Konstruktoraufruf mit this() hat zur Folge, daß er den impliziten Konstruktoraufruf super() ersetzt. Somit wird in untenstehendem Beispiel nur der Konstruktoraufruf mit this(3.2) durchgeführt.

```
public class Rock extends Kleidung{
        Rock(){
                this(3.2);
        }
```

```
        Rock(double preis) {
                super(preis);
                System.out.println("Ich bin ein Rock");
        }

        public static void main(String[ ] args) {
                Rock rock = new Rock();
                Rock preisRock = new Rock(2.2);
        }
}
```

Ausgabe auf der Konsole:

```
Ich koste 3.2
Ich bin ein Rock
Ich koste 2.2
Ich bin ein Rock
```

3. Übungen

(1) Frage

Welche der untenstehenden Aussagen treffen nicht zu?

a) Konstruktoren werden von der Superklasse an die Subklasse vererbt.
b) Der Aufruf super() muß im Konstruktor immer an erster Stelle stehen.
c) Existieren nur die impliziten Standardkonstruktoren, so wird automatisch im Standardkonstruktor der Subklasse der Konstruktor der Superklasse mit super() aufgerufen.
d) Es existiert immer ein Standardkonstruktor.
e) Mit this() wird der Standardkonstruktor der eigenen Klasse aufgerufen.
f) Mit super.laufen() wird die Methode laufen() der Superklasse aufgerufen.
g) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Aussagen trifft zu?

a) Ein fehlender Standardkonstruktor kann nie zu Kompilierfehlern führen.
b) Der Aufruf super() muß im Konstruktor immer an letzter Stelle stehen.
c) Existieren nur die impliziten Standardkonstruktoren, so wird automatisch im Standardkonstruktor der Subklasse der Konstruktor der Superklasse mit this() aufgerufen.
d) Es existiert immer ein Standardkonstruktor.
e) Mit this() wird der Standardkonstruktor der Superklasse aufgerufen.
f) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
        Vater(){
                System.out.println("Vater");
        }
}

class Tochter extends Vater{
        Tochter(){
                System.out.println("Tochter");
}

public static void main(String[ ] args){
Tochter tochter = new Tochter();
}
}
```

a) Tochter
b) Vater Tochter
c) Tochter Vater

d) Vater
e) Keine Ausgabe
f) Kompilierfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
        Vater(){
                System.out.println("Vater");
        }
}

class Tochter {
        Tochter(){
                System.out.println("Tochter");
}

public static void main(String[ ] args){
Tochter tochter = new Tochter();
}
}
```

a) Tochter
b) Vater Tochter
c) Tochter Vater
d) Vater
e) Keine Ausgabe
f) Kompilierfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
        Vater(){
                System.out.println("Vater");
        }
}
class Tochter extends Vater{
        Tochter(){
                System.out.println("Tochter");
}

public static void main(String[ ] args){
Tochter tochter = new Tochter();
super();
}
}a)     Tochter
b) Vater Tochter
```

c) Tochter Vater
d) Vater
e) Keine Ausgabe
f) Kompilierfehler

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
        Vater(){
                System.out.println("Vater");
        }
}

class Tochter extends Vater{
        Tochter(){
super()
                System.out.println("Tochter");
}

public static void main(String[ ] args){
Tochter tochter = new Tochter();
}
}
```

a) Tochter
b) Vater Vater Tochter
c) Vater Tochter Vater
d) Vater Tochter
e) Keine Ausgabe
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
        Vater(){
                System.out.println("Vater");
        }
}

class Tochter extends Vater{
Tochter(){
this("Tochter");
System.out.println("Tochter");
}

Tochter(String wort){
```

```
System.out.println("Ich bin " + wort);
}
public static void main(String[ ] args){
Tochter tochter = new Tochter();
}
}
```

a) Ich bin Tochter
b) Vater Vater Ich bin Tochter
c) Vater Ich bin Tochter Vater
d) Vater Ich bin Tochter Tochter
e) Keine Ausgabe
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

4. Lösungen

(1) Frage

a, d, Es geht in der Frage darum, was nicht zutrifft.

(2) Frage

f

(3) Frage

b

(4) Frage

a, Es fehlt die extends-Beziehung, also gibt es auch keine Verwandtschaftsverhältnis.

(5) Frage

f, Der Aufruf super() muß im Konstruktor ganz oben stehen, sonst kommt es zu einem Kompilierfehler. Außerdem darf er nicht in der main()-Methode stehen.

(6) Frage

d, Der explizite und implizite Aufruf von super() hat den gleichen Effekt.

(7) Frage

g

G. Subtyping und Casting von Objekten

1. Subtyping

Nehmen wir einmal an, Sie wollen sich ein Möbelstück kaufen und Sie wissen am Anfang nicht genau, ob es ein Tisch oder ein Stuhl sein soll. So können Sie zu Beginn erst einmal allgemein festlegen, daß Sie ein Möbelstück haben wollen. Sie müssen sich dann erst später entscheiden, ob Sie sich lieber einen Stuhl oder einen Tisch anschaffen. Nun beginnen wir Schritt für Schritt, dies in Code umzusetzen. Im Moment haben wir untenstehende Superklasse Moebel und die Subklasse Tisch:

```
public class Moebel {
    public void stehen(){
        System.out.println("Ich bin ein Möbelstück");
    }
}

public class Tisch extends Moebel{
    public void stehen() {
        System.out.println("Ich bin ein Tisch!");
    }
}
```

Normalerweise instanziiert man ein Objekt der Klasse Tisch folgendermaßen: Tisch neuer Tisch = new Tisch();, da Sie aber noch nicht sicher sind, ob Sie einen Tisch oder Stuhl benötigen, geht dies mit Moebel neuMoebel = new Tisch();. Sie sagen also, daß Sie ein Möbelstück brauchen, das im Moment ein Tisch ist. Da ein Tisch ein Moebel ist, kann die Referenzvariable neuMoebel, die vom Typ her Moebel ist, ein Objekt vom Typ Tisch speichern. Diese Vorgehensweise wird Subtyping genannt.

Ein Tisch ist ein Möbelstück, deshalb ist diese Zuweisung möglich. Andersherum geht es nicht, da ein Möbelstück nicht notwendigerweise ein Tisch ist. Versuchen wir dies zu tun, wird die entsprechende Stelle in Eclipse als falsch gekennzeichnet.

```
 5 public class Tisch extends Moebel{
 6
 7     public static void main(String[] args) {
 8
 9         Moebel neuMoebel = new Tisch();
10         //Ein Möbelstück ist kein Tisch!!
11         Tisch neuTisch = new Moebel();
12     }
13
14     public  void stehen() {
15         System.out.println("Ich bin ein Tisch!");
16     }
17 }
```

Lassen Sie uns nun erforschen, was passiert, wenn Sie auf dem Objekt neuMoebel die Methode stehen() aufrufen. Es wird die Methode stehen() des Objektes Tisch, der Sohnklasse, ausgeführt. Zuerst wird immer nach der Methode stehen() in der Vaterklasse gesucht, wenn allerdings in der Sohnklasse eine überschriebene Methode existiert, wird diese genommen, andernfalls wird die Methode der Vaterklasse ausgeführt. Existiert allerdings in der Vaterklasse diese Methode nicht, kommt es zu einem Kompilierfehler.

```
public class Tisch extends Moebel{
	public static void main(String[ ] args) {
		Moebel neuMoebel = new Tisch();
		neuMoebel.stehen();
	}

	public  void stehen() {
		System.out.println("Ich bin ein Tisch!");
	}
}
```

Ausgabe auf der Konsole:

```
Ich bin ein Tisch!
```

Nehmen wir nun an, daß wir eine zusätzliche Subklasse der Klasse Moebel, und zwar die Klasse Stuhl haben.

```
public class Stuhl, extends Moebel{
		public static void main(String[ ] args){
		}

		private void stehen() {
			System.out.println("Ich bin ein Stuhl!");
		}
}
```

Nun können wir nachträglich der Referenzvariablen neuMoebel ein Stuhlobjekt zuweisen. Dies ist möglich, da sowohl ein Objekt der Klasse Tisch als auch der Klasse Stuhl ein Möbelstück ist. Sie erhalten nun als Ausgabe „Ich bin ein Stuhl!“.

```
public class Tisch extends Moebel{
		public static void main(String[ ] args) {
			Moebel neuMoebel = new Tisch();
			neuMoebel = new Stuhl();
			neuMoebel.stehen();
		}

		public  void stehen() {
```

```
                System.out.println("Ich bin ein Tisch!");
        }
}
```

Ausgabe auf der Konsole:

Ich bin ein Stuhl!

2. Casten von Objekten

Manchmal besteht die Notwendigkeit, ein Objekt in einen anderen Typ umzuwandeln, diesen Vorgang nennt man explizite Typanpassung oder Casting. Dies ist zwar kein guter Programmierstil, aber manchmal unumgänglich.

Lassen Sie uns mit folgendem Cast beginnen, der zwar überflüssig ist, aber Ihnen einen ersten Einblick in ein komplexes Thema gibt. Diese Vorgehensweise ist identisch mit Moebel neuMoebel = new Tisch(); aus dem Kapitel Subtyping. Man kann einem Moebel einen Tisch oder einen Stuhl zuweisen, da ein Moebel sowohl ein Tisch als auch ein Stuhl sein kann.

```
public class Moebel {
        public void stehen(){
                System.out.println("Ich bin ein Möbelstück");
        }

        public static void main(String[ ] args){
                Moebel moebel = new Moebel();
                Tisch tisch = new Tisch();
                Stuhl stuhl = new Stuhl();
                Moebel mo = (Moebel)tisch;
                mo.stehen();
        }
}
```

Ausgabe auf der Konsole:

Ich bin ein Tisch!

Wohingegen weder ein Tisch in einen Stuhl, noch ein Stuhl in einen Tisch verwandelt werden kann. Diesbezügliche Zuweisungen und Castings führen unweigerlich zu einem Kompilierfehler, da ein Stuhl kein Tisch ist, und ein Tisch kein Stuhl.

```
public class Moebel {
        public void stehen(){
                System.out.println("Ich bin ein Möbelstück");
        }

        public static void main(String[ ] args){
```

```
                Moebel moebel = new Moebel();
                Tisch tisch = new Tisch();
                Stuhl stuhl = new Stuhl();
                stuhl = tisch; //Kompilierfehler
                stuhl = (Stuhl)tisch; //Kompilierfehler
        }
}
```

Will man ein Objekt vom Typ Moebel in einen Tisch oder Stuhl umwandeln, schlägt dieser Versuch fehl, da die Klasse Moebel sowohl einen Tisch als auch einen Stuhl beinhalten kann und nicht nur das eine oder das andere. Dies bemerkt der Kompiler allerdings nicht zur Kompilierzeit, Eclipse zeigt keinen Fehler an, es kommt erst zur Laufzeit zu einer ClassCastException.

```
 5 public class Tisch extends Moebel{
 6
 7     public static void main(String[] args) {
 8
 9         Moebel moebel = new Moebel();
10         Tisch tisch = new Tisch();
11         Stuhl stuhl = new Stuhl();
12
13         tisch = (Tisch)moebel;
14         stuhl = (Stuhl)moebel;
15     }
16 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Tisch [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (
java.lang.ClassCastException
        at scjp.Tisch.main(Tisch.java:13)
Exception in thread "main"
```

Diese ClassCastException kann ich vermeiden, indem ich der Referenzvariablen moebel sage, daß sie ein Objekt vom Typ Tisch beinhalten kann. Sie fügen in Zeile 13 moebel = tisch ein. So kann ich nachträglich ein Möbelstück in einen Tisch umwandeln. Es kommt nun weder zu einem Kompilier- noch zu einem Laufzeitfehler.

```
public class Tisch extends Moebel{
        public static void main(String[ ] args) {
                Moebel moebel = new Moebel();
                Tisch tisch = new Tisch();
                Stuhl stuhl = new Stuhl();
                moebel = tisch;
                tisch = (Tisch)moebel;
        }
```

}
Ändere ich allerdings stattdessen den Code, indem ich der Referenzvariable moebel ein Objekt vom Typ Stuhl zuweise, kommt es zu einem Laufzeitfehler. Diese Tatsache wird vom Kompiler nicht entdeckt, da gemäß Deklaration moebel vom Typ Moebel ist und so theoretisch auch ein Objekt vom Typ Tisch enthalten kann, aber nun beinhaltet es ein Objekt vom Typ Stuhl. Und einen Stuhl kann ich nicht in einen Tisch umwandeln. Diese Tatsache wird erst zur Laufzeit entdeckt (ClassCastException).

```
7     public static void main(String[] args) {
8
9         Moebel moebel = new Moebel();
10        Tisch tisch = new Tisch();
11        Stuhl stuhl = new Stuhl();
12        //der Referenzvariable moebel wird ein Objekt Stuhl zugewiesen
13        moebel = stuhl;
14        /*und wie wir wissen kann ich einen Stuhl nicht in einen
15        Tisch umwandeln, da aber theoretisch ein moebel ein Tisch
16        sein könnte, merkt er dies nicht zur Kompilierzeit, sondern
17        erst zur Laufzeit*/
18        tisch = (Tisch)moebel;
19    }
20 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Tisch [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (02.05.2005 10:29:41)
java.lang.ClassCastException
        at scjp.Tisch.main(Tisch.java:13)
Exception in thread "main"
```

3. Operator: instanceof

Will man herausfinden, ob ein neues Auto tatsächlich ein Objekt der Klasse Auto ist, kann man dies mit instanceof. Für Instanzen der gleichen Klasse und der Subklassen gibt instanceof den boolean-Wert true zurück (vgl. Kapitel Literale). Für Objekte der Vaterklasse wird ein false zurückgeben. Der Wert true bedeutet wahr und false nicht wahr. So ist das Objekt auto kein Sportwagen (false), aber das Objekt Sportwagen ist ein Auto (true).

```
public class Sportwagen extends Auto{
        public static void main(String[ ] args) {
                Auto auto = new Auto();
                Sportwagen sportwagen = new Sportwagen();

                boolean a = sportwagen instanceof Auto;
                boolean b = auto instanceof Sportwagen;
                boolean c = auto instanceof Auto;

                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
        }
}
```

Ausgabe auf der Konsole:

true
false
true

Gibt es kein Verwandtschaftsverhältnis zwischen den Objekten, kommt es zu einem Kompilierfehler. So besteht z. B. keinerlei Verbindung zwischen Tischen und Autos.

```
3  public class Sportwagen extends Auto{
4
5      public static void main(String[] args) {
6
7          Auto auto = new Auto();
8          Tisch tisch = new Tisch();
9
10         boolean a = tisch instanceof Auto;
11         boolean b = auto instanceof Auto;
12
13         System.out.println(a);
14         System.out.println(b);
15
16     }
17 }
```

Wird allerdings ein Objekt mit einem Interface verglichen, mit dem es keinerlei Verwandtschaftsverhältnis gibt, wird false zurückgegeben, und es kommt weder zu einem Kompilier- noch zu einem Laufzeitfehler. In untenstehendem Beispiel soll herausgefunden werden, ob ein Sportwagen eine Collection ist. Da dies nicht der Fall ist, wird false auf der Konsole ausgegeben (vgl. Kapitel Collections und Maps).

```
import java.util.Collection;

public class Sportwagen extends Auto{
        public static void main(String[ ] args) {
                Sportwagen sportwagen = new Sportwagen();

                boolean a = sportwagen instanceof Collection;

                System.out.println(a);
        }
}
```

Ausgabe auf der Konsole:

false

4. Übungen

(1) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A { }
class B extends A{ }
class C extends B{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                a = b;                  Zeile 4
                b = (B) a;              Zeile 5
                b = c;                  Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(2) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B {}
class C {
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                a = b;                  Zeile 4
                b = (B) a;              Zeile 5
                b = c;                  Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(3) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B extends A{}
class C extends B{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                a = b;                  Zeile 4
                b = a;                  Zeile 5
                b = c;                  Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                a = b;                  Zeile 4
                a = c;                  Zeile 5
                b = c;                  Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                a = b;                  Zeile 4
                b = a;                  Zeile 5
                a = c;                  Zeile 6
        }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(6) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new C();          Zeile 3
                b = (b)a;               Zeile 4
                a = c;                  Zeile 5
                b = a;                  Zeile 6
                }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(7) Frage

In welcher Zeile tritt ein Kompilierfehler auf?

```
class A {}
class B extends A{}
```

```
class C extends A{
        public static void main(String[ ] args){
                A a = new A();          Zeile 1
                B b = new B();          Zeile 2
                C c = new B();          Zeile 3
                a = b;                  Zeile 4
                b = c;                  Zeile 5
                b = a;                  Zeile 6
        }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten

(8) Frage

In welcher Zeile tritt welcher Fehler auf?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                a = b;                  Zeile 1
                c = (C)a;               Zeile 2
        }
}
```

a) Es tritt in Zeile 1 ein Kompilierfehler auf.
b) Es tritt in Zeile 1 ein Laufzeitfehler auf.
c) Es tritt in Zeile 2 ein Kompilierfehler auf.
d) Es tritt in Zeile 2 ein Laufzeitfehler auf.
e) Keine dieser Möglichkeiten

(9) Frage

In welcher Zeile tritt welcher Fehler auf?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                a = b;                  Zeile 1
```

```
                c = a;                          Zeile 2
        }
}
```

a) Es tritt in Zeile 1 ein Kompilierfehler auf.
b) Es tritt in Zeile 1 ein Laufzeitfehler auf.
c) Es tritt in Zeile 2 ein Kompilierfehler auf.
d) Es tritt in Zeile 2 ein Laufzeitfehler auf.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;        Zeile 1
                boolean b2 = a instanceof C;        Zeile 2
                boolean b3 = c instanceof A;        Zeile 3
                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, false, true
d) false, true, false
e) false, true, true
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;   Zeile 1
                boolean b2 = a instanceof C;   Zeile 2
```

```
                boolean b3 = c instanceof B;    Zeile 3

                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, true, false
d) true, false, false
e) false, true, true
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;    Zeile 1
                boolean b2 = a instanceof C;    Zeile 2
                boolean b3 = b instanceof A;    Zeile 3

                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, true, false
d) true, false, false
e) false, false, true
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;            Zeile 1
                boolean b2 = a instanceof C;            Zeile 2
                boolean b3 = c instanceof Object;       Zeile 3

                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, false, true
d) false, true, false
e) true, false, false
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

(14) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;   Zeile 1
                boolean b2 = c instanceof C;   Zeile 2
                boolean b3 = b instanceof A;   Zeile 3

                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, false, true
d) false, true, false

e) false, true, true
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class A {}
class B extends A{}
class C extends A{
        public static void main(String[ ] args){
                A a = new A();
                B b = new B();
                C c = new C();
                boolean b1 = a instanceof B;          Zeile 1
                boolean b2 = a instanceof Tisch;      Zeile 2
                boolean b3 = c instanceof A;          Zeile 3

                System.out.println(b1);
                System.out.println(b2);
                System.out.println(b3);
        }
}
```

a) true, true, false
b) true, false, true
c) false, false, true
d) true, false, false
e) false, true, true
f) Es tritt in Zeile 1 ein Kompilierfehler auf.
g) Es tritt in Zeile 2 ein Kompilierfehler auf.
h) Es tritt in Zeile 3 ein Kompilierfehler auf.

5. Lösungen

(1) Frage

g

(2) Frage

d, e, f, Die extends-Beziehungen fehlen.

(3) Frage

e

(4) Frage
f

(5) Frage
e

(6) Frage
d, f

(7) Frage
c, e, f

(8) Frage
d

(9) Frage
c

(10) Frage
c

(11) Frage
h

(12) Frage
e

(13) Frage
c, Alle Objekte sind per Definition Subklassen von java.lang.Object.

(14) Frage
e

(15) Frage
g

V. Operatoren

Wenn Sie zwei Zahlen miteinander addieren wollen, benötigen Sie das Pluszeichen (+). Oder bei der Multiplikation brauchen Sie das Malzeichen (*). In Programmiersprachen werden diese Zeichen unter dem Oberbegriff Operatoren zusammengefaßt. Will man in einer Programmiersprache Berechnungen durchführen, benötigt man Operatoren.

A. Arithmetische Operatoren

Unter Verwendung von Arithmetischen Operatoren können Sie in Java Zahlen subtrahieren, addieren, multiplizieren und dividieren, wobei Punkt vor Strich gilt.

1. Die Operatoren für Subtraktion und Addition

In untenstehden Beispiel wollen wir die Zahlen 2 und 5 zusammenzählen und die Zahl 2 von 5 abziehen.

```
public class AddSub {
	public static void main(String[ ] args) {
		int i = 2;
		int b = 5;
		System.out.println("Ergebnis der Subtraktion: " + (b - i));
		System.out.println("Ergebnis der Addition: " + (i + b));
	}
}
```

Ausgabe auf der Konsole:

```
Ergebnis der Subtraktion: 3
Ergebnis der Addition: 7
```

2. Die Operatoren für Multiplikation und Division

Der Operator * multipliziert die zwei Zahlen, 12 und 13, miteinander.

```
public class Multiplikation {
	public static void main(String[ ] args) {
		int i = 12;
		int i1 = 13;
		System.out.println(i*i1);
	}
}
```

Ausgabe auf der Konsole:

```
156
```

Der Divisionsoperator dividiert zwei Zahlen miteinander. Hierbei ist bei einer Division mit Ganzzahlen (short, byte, int und long) zu beachten, daß nach der Division die Nachkommastellen abgeschnitten werden. Dies ist auch der Fall, wenn der Rückgabetyp ein double ist. Wenn Sie ein Ergebnis haben wollen, das die Nachkommastellen nicht kappt, müssen Sie zwei Kommazahlen miteinander dividieren, sprich die Zahlen müssen in Java als Fließkommazahlen dargestellt werden. Dies wäre entweder eine Zahl vom Datentyp double (10.0 oder 10.0d) oder float (10.f). Werden unterschiedliche Datentypen verwendet, findet eine implizite, sprich automatische, Typanpassung statt, es wird die kürzere Zahl in die längere umgewandelt. So wird bei double d = 10.0/3, die Zahl 3, die vom primitiven Datentyp int ist, an den größeren Datentyp double der Zahl 10.0 angepaßt. Anschließend wird eine Zahl durch die andere geteilt.

```
public class Division {
	public static void main(String[ ] args) {
		int a = 10/3;
		double b = 10/3;
		double c = 10.0/3.0;
		double d = 10.0/3;

		System.out.println("Ganzzahldivision " + a);
		System.out.println("Ganzzahldivision " + b);
		System.out.println("Division von Fließkommazahlen " + c);
		System.out.println("Division von Fließkommazahlen " + d);
	}
}
```

Ausgabe auf der Konsole:

```
Ganzzahldivision 3
Ganzzahldivision 3.0
Division von Fließkommazahlen 3.3333333333333335
Division von Fließkommazahlen 3.3333333333333335
```

3. Der Operator Modulo %

Im Gegensatz zur Division werden mit dem Operator Modulo weder die Nachkommastellen abgeschnitten noch berechnet, sondern es wird der Restwert ermittelt. Bei der Moduloberechnung von 10 % 3 wird festgestellt, daß die Zahl 3 dreimal in 10 passt (3 * 3 = 9), also beträgt der Restwert 1. Die Moduloberechnung 16 % 5 ergibt ebenfalls den Restwert 1, da 5 dreimal in 15 passt (5 * 3 = 15) und 16 – 15 die Zahl 1 zum Ergebnis hat.

```
public class Division {
	public static void main(String[ ] args) {
		int a = 10/3;
		int b = 10%3;
		int c = 16/5;
```

```
                int d = 16%5;

                System.out.println(”Division: ” + a);
                System.out.println(”Modulo: ” + b);
                System.out.println(“Division: “ + c);
                System.out.println(“Modulo: “ + d);
        }
}
```

Ausgabe auf der Konsole:

```
Division: 3
Modulo: 1
Division: 3
Modulo: 1
```

Bei der Moduloberechnung kommt es nur zu einem negativen Ergebnis, wenn die Zahl vor dem Modulozeichen negativ ist.

```
public class Division {
        public static void main(String[ ] args) {
                System.out.println(-5 % 2);
                System.out.println(5 % -2);
                System.out.println(-5 % -2);
        }
}
```

Ausgabe auf der Konsole:

```
-1
1
-1
```

4. Bearbeitungsreihenfolge der Operatoren

Grundsätzlich gilt bei Berechnungen in Java, wie in der Mathematik, die Regel Punkt vor Strich. Des weiteren gibt es zusätzlich eine verbindliche Reihenfolge, in der Operatoren abgearbeitet werden müssen. Die Prioritäten werden in untenstehender Tabelle zusammengefaßt. Haben zwei Operatoren allerdings die gleiche Priorität, werden sie in der Reihenfolge von links nach rechts abgearbeitet. Die Bearbeitungsreihenfolge kann man, wie in der Mathematik, mit Klammern beeinflussen, was man zur besseren Lesbarkeit auch tun sollte.

Operatoren	Priorität	Typ	Beschreibung
`++, --`	1	arithmetisch	Inkrement und Dekrement
`+, -`	1	arithmetisch	unäres Plus und Minus
`~`	1	integral	bitweises Komplement
`!`	1	boolean	logisches Komplement
(Typ)	1	jedes	Cast
`*, /, %`	2	arithmetisch	Multiplikation, Division, Rest
`+, -`	3	arithmetisch	Addition und Subtraktion
`+`	3	String	String-Konkatenation
`<<`	4	integral	Shift links
`>>`	4	integral	Shift rechts m. Vorzeichenerweiterung
`>>>`	4	integral	Shift rechts o. Vorzeichenerweiterung
`<, <=, >, >=`	5	arithmetisch	numerische Vergleiche
`instanceof`	5	Objekt	Typvergleich
`==, !=`	6	primitiv	Gleich-/Ungleichheit von Werten
`==, !=`	6	Objekt	Gleich-/Ungleichheit von Referenzen
`&`	7	integral	bitweises Und
`&`	7	boolean	logisches Und
`^`	8	integral	bitweises Xor
`^`	8	boolean	logisches Xor
`|`	9	integral	bitweises Oder
`|`	9	boolean	logisches Oder
`&&`	10	boolean	logisches konditionales Und, Kurzschluss
`||`	11	boolean	logisches konditionales Oder, Kurzschluss
`?:`	12	alles	Bedingungsoperator
`=`	13	jede	Zuweisung
`*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=`	14	jede	Zuweisung mit Operation

Für die Zertifizierung ist es nützlich folgendes zu verinnerlichen: Der Grundsatz Punkt vor Strich ist verbindlich. Die Operatoren Modulo % und Multiplikation * haben die gleiche Priorität, was zur Folge hat, daß die Operatoren von links nach rechts berücksichtigt werden.

In untenstehender erster Berechnung wird zuerst 10 * 5 = 50 durchgeführt, dann 50 % 2 = 0 und anschließend wird 0 zu 4 addiert. Somit ist die Zahl 4 das Ergebnis. Beim zweiten Beispiel wird zuerst 2 % 3 = 2 und danach 2 * 4 = 8 berechnet. Zum Schluß wird zum Ergebnis der Multiplikation die 1 addiert (8 + 1), was 9 ergibt.

```
public class PunktVorStrich {
	public static void main(String[ ] args) {
		System.out.println(10 * 5 % 2 + 4);
		System.out.println(1 + 2 % 3 * 4);
	}
}
```

Ausgabe auf der Konsole:

```
4
9
```

5. Zusammengesetzte Zuweisungsoperatoren: *=, /=, %=, +=, - =

Zusammengesetzte Operatoren wie z. B. int i += 5 sind die abgekürzte Version von i = i + 5. Diese Berechnung addiert jeweils zu i die Zahl 5. So erhalten Sie als Ergebnis in untenstehendem Code, die Zahlen 10 und 15.

```
public class Operatoren {
	public static void main(String[ ] args) {
		int i = 5;
		i += 5;
		System.out.println(i);
		i = i + 5;
		System.out.println(i);
	}
}
```

Ausgabe auf der Konsole:

```
10
15
```

Probleme und Unterschiede tauchen allerdings auf, wenn der Rückgabetyp der Variablen kleiner als int ist, wie z. B. byte. Beim zusammengesetzten Zuweisungsoperator (i += 5) wird die rechte Seite automatisch in den kleineren

Datentyp umgewandelt, sprich es findet ein impliziter Cast statt. Bei der einfachen Zuweisung ist dies nicht der Fall. Hier wäre ein expliziter Cast notwendig, deswegen kommt es bei i = i + 5 zu einem Kompilierfehler.

```
 4
 5 public class Operatoren {
 6     public static void main(String[] args) {
 7         byte i = 5;
 8         i += 5;
 9         i = i + 5;
10     }
11 }
```

Wenn Sie einen Kompilierfehler in der Klasse Operatoren verhindern wollen, müssen Sie den Code, wie folgt abändern:

```
public class Operatoren {
        public static void main(String[ ] args) {
                byte i = 5;
                i += 5;
                i = (byte) (i + 5);
        }
}
```

6. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class P {
	public static void main(String[ ] args) {
		System.out.println(1 * 2 % 3 + 4);
		System.out.println(1 + 2 + 3 * 4);
	}
}
```

a) 6 15
b) 6 24
c) 7 15
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class P {
	public static void main(String[ ] args) {
		System.out.println(5 * 2 % 3 + 4);
		System.out.println(5 % 2 + 3 * 4);
	}
}
```

a) 19 13
b) 5 16
c) 5 13
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class P {
	public static void main(String[ ] args) {
		System.out.println(15 * 5 % 3 + 6);
		System.out.println(15 % 5 + 3 * 6);
	}
}
```

a) 6 18
b) 36 18
c) 6 30

d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
    public static void main(String[ ] args) {
        System.out.println(-8 % 3);
        System.out.println(8  * -3);
        System.out.println(-8 % -3);
    }
}
```

a) -2 2 -2
b) -2 -2 -2
c) -2 -24 -2
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
    public static void main(String[ ] args) {
        System.out.println(-8 % 3);
        System.out.println(8  % -3);
        System.out.println(-8 % -3);
    }
}
```

a) -2 2 -2
b) -2 -2 -2
c) -2 -24 -2
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
    public static void main(String[ ] args) {
        System.out.println(-8 % 3);
        System.out.println(8  / -3);
        System.out.println(-8 % -3);
    }
}
```

a) -2 2 -2
b) -2 -2 -2
c) -2 -24 -2
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
    public static void main(String[ ] args) {
        System.out.println(-9.0 / 2.0);
        System.out.println(10.0 / -2.0);
        System.out.println(-7.0 / -4.0);
    }
}
```

a) -4.5 -5.0 1.75
b) -4.0 -5.0 1.0
c) -4 -5 1
d) -5 -5 2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
    public static void main(String[ ] args) {
        System.out.println(-9 / 2);
        System.out.println(10 / -2);
        System.out.println(-7 / -4);
    }
}
```

a) -4.5 -5.0 1.75
b) -4.0 -5.0 1.0
c) -4 -5 1
d) -5 -5 2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class O {
	public static void main(String[ ] args) {
		short i = 5;
		i += 2;
		System.out.println(i);
	}
}
```

a) 5
b) 7
c) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class O {
	public static void main(String[ ] args) {
		short i = 5;
		i = i + 2;
		System.out.println(i);
	}
}
```

a) 5
b) 7
c) Kompilierfehler

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class O {
	public static void main(String[ ] args) {
		int i = 5;
		i = i + 2;
		System.out.println(i);
	}
}
```

a) 5
b) 7
c) Kompilierfehler

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
	public static void main(String[ ] args) {
```

```
            System.out.println(-7 % 2);
            System.out.println(7 % -2);
            System.out.println(7 % -4);
      }
}
```

a) -1 1 3
b) 1 -1 -3
c) -3 -3 -1
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
      public static void main(String[ ] args) {
            System.out.println(-7 / 2);
            System.out.println(7 / -2);
            System.out.println(7 / -4);
      }
}
```

a) -1 1 3
b) 1 -1 -3
c) -3 -3 -1
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class D {
      public static void main(String[ ] args) {
            System.out.println(-7.0 / 2.0);
            System.out.println(7.0 / -2.0);
            System.out.println(7.0 / -4.0);
      }
}
```

a) -3.5 -3.5 -1.75
b) -3.5 3.5 1.75
c) -3 -3 -1
d) -3.0 -3.0 -1.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

7. Lösungen

(1) Frage
a

(2) Frage
c

(3) Frage
a

(4) Frage
c

(5) Frage
a

(6) Frage
b

(7) Frage
a

(8) Frage
c

(9) Frage
b

(10) Frage
c

(11) Frage
b

(12) Frage
a

(13) Frage
c

(14) Frage
a

B. Die Boolean Logischen Operatoren

Es gibt vier verschiedene Boolean Logische Operatoren der Nicht-Operator, das Logische Und, das Logische Oder und das Logische XOR.

1. Der Logische Nicht-Operator

Der logische Nicht-Operator gibt den gegenteiligen Wert des ursprünglichen Wertes aus. Wie sie in untenstehendem Beispiel sehen können, wird bei !true (nicht richtig = falsch) als Ergebnis false ausgegeben.

```
public class LogischeOperatoren {
	public static void main(String[ ] args) {
		boolean a = true;
		System.out.println(!a);
	}
}
```

Ausgabe auf der Konsole:

```
false
```

2. Der Logische Und-Operator

Werden zwei boolean-Werte mit dem Logische Und-Operator in Beziehung gesetzt, erhält man nur true, wenn beide Operanten true sind.

```
public class LogischeOperatoren {
	public static void main(String[ ] args) {
		boolean a = true;
		boolean b = true;
		boolean c = false;
		boolean d = false;
		System.out.println(a & b);
		System.out.println(b & c);
		System.out.println(c & d);
	}
}
```

Ausgabe auf der Konsole:

```
true
false
false
```

3. Der Logische Oder-Operator

Wenn mindestens einer von zwei Operanten true ist, gibt der Logische Oder-Operator als Resultat true auf der Konsole aus.

```
public class LogischeOperatoren {
	public static void main(String[ ] args) {
		boolean a = true;
		boolean b = true;
		boolean c = false;
		boolean d = false;
		System.out.println(a | b);
		System.out.println(b | c);
		System.out.println(c | d);
	}
}
```

Ausgabe auf der Konsole:

```
true
true
false
```

4. Logischer XOR-Operator

Der Logische XOR-Operator hat als Ergebnis nur true, wenn einer von beiden Operatoren true ist. Es darf also nur einer true sein, sind beide true oder beide false wird false zurückgegeben.

```
public class LogischeOperatoren {
	public static void main(String[ ] args) {
		boolean a = true;
		boolean b = true;
		boolean c = false;
		boolean d = false;
		System.out.println(a ^ b);
		System.out.println(b ^ c);
		System.out.println(c ^ d);
	}
}
```

Ausgabe auf der Konsole:

```
false
true
false
```

5. Bearbeitungsreihenfolge bei den Logischen Operatoren

Es gilt wie bei allen Operatoren, sie werden zuerst nach Prioritäten abgearbeitet und anschließend bei Operatoren mit gleicher Priorität von links nach rechts (siehe Bearbeitungsreihenfolge der Operatoren). Es wird zuerst &, dann ^ und dann | durchgeführt. In untenstehendem Beispiel wird zuerst b & c berechnet, dies gibt false zurück, dann wird false ^ d bearbeitet und man erhält true als Resultat und zum Schluß wird a | true ausgewertet, was true als Ergebnis hat. Somit wird der Wahrheitswert true auf der Konsole ausgegeben.

```
public class LogischeOperatoren {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                boolean d = true;
                System.out.println(a | b & c ^ d);
        }
}
```

Ausgabe auf der Konsole:

```
true
```

6. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
	public static void main(String[ ] args) {
		boolean a = true;
		boolean b = false;
		boolean c = false;
		boolean d = false;
		System.out.println(a | b);
		System.out.println(b ^ c);
		System.out.println(c & d);
	}}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
	public static void main(String[ ] args) {
		boolean a = true;
		boolean b = false;
		boolean c = false;
		boolean d = false;
		System.out.println(a | b);
		System.out.println(b | c);
		System.out.println(c | d);
	}}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = true;
                boolean b = false;
                boolean c = false;
                boolean d = false;
                System.out.println(a ^ b);
                System.out.println(b ^ c);
                System.out.println(c ^ d);
        }
}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = true;
                boolean b = false;
                boolean c = false;
                boolean d = false;
                System.out.println(a & b);
                System.out.println(b & c);
                System.out.println(c & d);
        }
}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                boolean d = true;
                System.out.println(a & b);
                System.out.println(b & c);
                System.out.println(c & d);
        }
}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                boolean d = true;
                System.out.println(a ^ b);
                System.out.println(b ^ c);
                System.out.println(c ^ d);
        }}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
```

```
                boolean c = true;
                boolean d = true;
                System.out.println(a | b);
                System.out.println(b | c);
                System.out.println(c | d);
        }
}
```

a) false false false
b) true false false
c) true true false
d) true true true
e) false true false
f) false false true
g) false true true

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                System.out.println(a | b & c ^ a);
        }
}
```

a) true
b) false

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                System.out.println(a & b & c ^ a);
        }
}
```

a) true
b) false

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class L {
        public static void main(String[ ] args) {
                boolean a = false;
                boolean b = false;
                boolean c = true;
                System.out.println(a | b | c ^ a);
        }
}
```

a) true
b) false

7. Lösungen

(1) Frage
b

(2) Frage
b

(3) Frage
b

(4) Frage
a

(5) Frage
f

(6) Frage
e

(7) Frage
g

(8) Frage
b

(9) Frage
b

(10) Frage
a

8. Short-Circuit-Auswertung

Es gibt zwei weitere logische Operatoren && und ||. Bei && kommt nur true als Ergebnis heraus, wenn beide Operanten true sind. Bei der Auswertung mit || genügt es, wenn einer von zwei Operanten true ist, damit das Resultat true ist. Soweit entspricht es dem Verhalten von & und |, nun gibt es aber einen großen Unterschied, der sich Short-Circuit nennt. Short-circuit bedeutet, daß zwar die Operatoren von links nach rechts abgearbeitet werden, die rechte Seite jedoch unter Umständen gar nicht mehr in Betracht gezogen wird. Steht aufgrund der linken Seite bereits ein definitives Ergebnis fest, das durch die rechte Seite nicht mehr beeinflußt werden kann, wird die Auswertung abgebrochen. Dies steht im Gegensatz zu den anderen Logischen Operatoren, bei denen in jedem Fall beide Seiten berücksichtigt werden müssen.

Hier ein Beispiel, wo der Unterschied zum Tragen kommt. Für den nicht geübten Programmierer ist es zu empfehlen, sich vorher die Kapitel Kontrollstrukturen und Shift-Operatoren anzusehen. Der Ausdruck auf der rechten Seite ++b wird erst ausgeführt, wenn ++a < 2 auf der linken Seite false wird, also nachdem die for-Schleife zweimal durchlaufen wurde.

```
public class ShortCircuit {
	public static void main(String[ ] args) {
		int a = 0;
		int b = 0;
			for (int i = 0; i<5; i++){
				if(++a <2 || ++b <2){
					System.out.println(a + „ „ + b);
				}
			}
	}
}
```

Ausgabe auf der Konsole:

```
1 0
2 1
```

Wohingegen, wenn || durch | ersetzt wird, sofort beide Seiten ausgewertet werden, somit wird die for-Schleife nur ein einziges Mal ausgeführt.

```
public class ShortCircuit {
	public static void main(String[ ] args) {
		int a = 0;
		int b = 0;
			for (int i = 0; i<5; i++){
				if(++a <2 | ++b <2){
					System.out.println(a + „ „ + b);
				}
			}
	}
}
```

Ausgabe auf der Konsole:

```
1 1
```

9. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		int a = 1;
		int b = 1;
			for (int i = 0; i<5; i++){
				if(++a <3 || ++b <3){
					System.out.println(a + „ „ + b);
				}
			}
	}
}
```

a) 2 2
b) 2 1 3 2

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		int a = 1;
		int b = 1;
			for (int i = 0; i<5; i++){
				if(++a <3 | ++b <3){
					System.out.println(a + „ „ + b);
				}
			}
	}
}
```

a) 2 2
b) 2 1 3 2

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		int a = 1;
		int b = 1;
			for (int i = 0; i<5; i++){
				if(++a <3 && ++b <3){
					System.out.println(a + „ „ + b);
```

```
                    }
                }
        }
}
```

a) 2 2
b) 2 1 3 2

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0;
                int b = 0;
                        for (int i = 0; i<5; i++){
                                if(++a <2 || ++b <3){
                                        System.out.println(a + „ „ + b);
                                }
                        }
        }
}
```

a) 1 0 2 1 3 2
b) 1 1 2 2
c) 1 1

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0;
                int b = 0;
                        for (int i = 0; i<5; i++){
                                if(++a <2 | ++b <3){
                                        System.out.println(a + „ „ + b);
                                }
                        }
        }
}
```

a) 1 0 2 1 3 2
b) 1 1 2 2
c) 1 1

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0;
                int b = 0;
                        for (int i = 0; i<5; i++){
                                if(++a <2 && ++b <3){
                                        System.out.println(a + „ „ + b);
                                }
                        }
        }
}
```

a) 1 0 2 1 3 2
b) 1 1 2 2
c) 1 1

10. Lösungen

(1) Frage
b

(2) Frage
a

(3) Frage
a

(4) Frage
a

(5) Frage
b

(6) Frage
c

C. Die Operatoren Inkrement ++ und Dekrement --

Bei der Inkrement-Funktion ++i wird i jeweils um 1 erhöht und bedeutet somit das gleiche wie i= i + 1, wohingegen die Dekrement-Funktion --i, i jeweils um 1 reduziert. Es gibt nun zwei Möglichkeiten: Man kann das ++ vor i schreiben oder nach i, also entweder ++i oder i++. Bei ++i wird i vor der Addition, um 1 erhöht, wohingegen bei i++, i erst danach um 1 erhöht wird. Nun ein praktisches Beispiel, in untenstehendem Fall wird k mit dem Wert 10 zu i addiert (10 + 11), somit ist das Ergebnis 21. Die Variable i ist also bereits vor der Berechnung, von 10 auf 11 erhöht worden. Nachdem die beiden Zahlen zusammengezählt worden sind, wird der Wert von k um 1 erhöht und k hat jetzt den Wert 11.

```
public class IncrDecre {
        public static void main(String[ ] args){
                int i = 10;
                int k = 10;
                int l = ++i + k++;
                //i wird hochgezählt vor der Berechnung
                System.out.println(i);
                //k wird hochgezählt nach der Berechnung
                System.out.println(k);
                //es wird i = 11 und k = 10 zusammengezählt
                System.out.println(l);
        }
}
```

Ausgabe auf der Konsole:

```
11
11
21
```

In untenstehender Klasse IncreDecre wird vor der Berechnung zuerst ++i von 12 auf 13 gesetzt, k++ wird nicht verändert und bleibt bei dem Wert 12 und i++ behält den Wert 13. Diese Zahlen werden addiert und das Ergebnis ist 38 (13 + 12 + 13 = 38). Nachdem diese Werte zusammengezählt wurden, wird i von 13 auf 14 und k von 12 auf 13 hochgezählt. So haben die Variablen zum Schluß bei der Ausgabe folgende Werte: i = 14, k = 13 und l = 38.

```
public class IncrDecre {
        public static void main(String[ ] args){
                int i = 12;
                int k = 12;
                //38 = 13 + 12 + 13
                int l = ++i + k++ + i++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);
```

```
        }
}
```

Ausgabe auf der Konsole:

```
14
13
38
```

1. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class {
        public static void main(String[ ] args){
                int i = 1;
                int k = 1;
                int l = i++ + k++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);}}
```

a) 2 2 4
b) 1 1 2
c) 2 2 2
d) 1 2 3
e) 2 2 3
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class {
        public static void main(String[ ] args){
                int i = 1;
                int k = 1;
                int l = i++ + ++k;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);}
}
```

a) 2 2 4
b) 1 1 2
c) 2 2 2
d) 1 2 3
e) 2 2 3
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class {
```

```
public static void main(String[ ] args){
                int i = 1;
                int k = 1;
                int l = ++i + k++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);
}}
```

a) 2 2 4
b) 1 1 2
c) 2 2 2
d) 1 2 3
e) 2 2 3
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class {
        public static void main(String[ ] args){
                int i = 1;
                int k = 1;
                int l = ++i + ++k;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);}}
```

a) 2 2 4
b) 1 1 2
c) 2 2 2
d) 1 2 3
e) 2 2 3
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
                int i = 5;
                int k = 3;
                int l = ++i + ++k + i++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);
}}
```

a) 7 4 16
b) 6 3 14
c) 6 3 15
d) 7 4 15
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
                int i = 5;
                int k = 3;
                int l = ++i + k++ + i++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);
        }
}
```

a) 7 4 16
b) 6 3 14
c) 6 3 15
d) 7 4 15
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
                int i = 5;
                int k = 3;
                int l = i++ + ++k + i++;
                System.out.println(i);
                System.out.println(k);
                System.out.println(l);
        }
}
```

a) 7 4 16
b) 6 3 14
c) 6 3 15
d) 7 4 15
e) Kompilierfehler

f) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
            int k = 6;
            int i = 6;
            int l = ++i + k++;
            System.out.println(i);
            System.out.println(k);}
}
```

a) 6 6
b) 6 7
c) 7 6
d) 7 7
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
            int k = 6;
            int i = 6;
            int l = i++ + k++;
            System.out.println(i);
            System.out.println(k); }}
```

a) 6 6
b) 6 7
c) 7 6
d) 7 7
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class I {
        public static void main(String[ ] args){
            int k = 6;
            int i = 6;
            int l = ++i + k++;
```

```
            System.out.println(i);
            System.out.println(l);
        }
}
```

a) 6 12
b) 6 13
c) 7 13
d) 7 14
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

2. Lösungen

(1) Frage
c

(2) Frage
e

(3) Frage
e

(4) Frage
a

(5) Frage
a

(6) Frage
d

(7) Frage
d

(8) Frage
d

(9) Frage
d

(10) Frage
c

D. Relationale Operatoren

Es gibt die relationalen Operatoren, die Ihnen aus der Mathematik bekannt sind: < (kleiner), <= (kleiner gleich), > (größer) oder >= (größer gleich). So können Sie z. B. feststellen, wer älter, jünger oder gleich alt ist, Sie oder Ihr Vater.

E. Integer Bitweise Operatoren

1. Exkurs: Zahlensysteme

a) Binäres Zahlensystem

Jeder Computer versteht eigentlich nur ein und aus bzw. 0 und 1. So ist die Aufgabe einer Programmiersprache, Code in eine Sprache zu übersetzen, die nur aus Nullen und Einsen besteht. Manchmal ist es auch notwendig direkt mit den Zahlen 0 und 1, dem sogenannten binären Zahlensystem, zu arbeiten. Dann muß man Zahlen aus dem „normalen“ dezimalen System in das binäre Zahlensystem, sprich in Zahlen zur Basis 2, umwandeln.

(1) Umrechnungsbeispiele von binär nach dezimal

Von binär 0001_2 nach dezimal 1:

Die binäre Zahl ganz rechts wird mit 2^0 malgenommen, also $1 * 2^0$, die nächste Zahl mit 2^1 ($0 * 2^1$), die nächste mit 2^2 ($0 * 2^2$) und die nächste mit 2^3 ($0 * 2^3$). Das Ergebnis dieser Multiplikationen wird addiert (0 + 0 + 0 + 1 = 1) und somit erhalten Sie als Summe die dezimale Zahl 1. In den Beispielen weiter unten wird analog vorgegangen.

$0001 = 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 0 + 0 + 0 + 1 = 1$

0	0	0	1	
$0 * 2^3$	$+ 0 * 2^2$	$+ 0 * 2^1$	$+ 1 * 2^0$	
0	+ 0	+ 0	+ 1	= 1

Von binär 0010_2 nach dezimal 2:

0	0	1	0	
$0 * 2^3$	$+ 0 * 2^2$	$+ 1 * 2^1$	$+ 0 * 2^0$	
0	+ 0	+ 2	+ 0	= 2

Von binär 0100_2 nach dezimal 4:

0	1	0	0	
$0 * 2^3$	$+ 1 * 2^2$	$+ 0 * 2^1$	$+ 0 * 2^0$	
0	+ 4	+ 0	+ 0	= 4

Von binär 1000_2 nach dezimal 8:

1	0	0	0	
$1 * 2^3$	$+ 0 * 2^2$	$+ 0 * 2^1$	$+ 0 * 2^0$	
8	+ 0	+ 0	+ 0	= 8

(2) Zusammenfassende Umrechnungstabelle von binär nach dezimal

Untenstehende Aufstellung hat als Überschrift die Ergebnisse der Hochzahlen 2^0 bis 2^6, diese sollen es Ihnen erleichtern, Zahlen vom dezimalen in das binäre System und umgekehrt umzurechnen. Die dezimale Zahl 32 läßt sich durch 1 * 32 + 0 * 16 + 0 * 8 + 0 * 4 + 0 * 2 + 0 *1 darstellen, somit ist die dazugehörige binäre Zahl 0010 0000 (binäre Zahlen werden immer in Vierer-Blöcken wiedergegeben und werden von links mit Nullen aufgefüllt). 33 läßt sich durch 1 * 32 + 0 * 16 + 0 * 8 + 0 * 4 + 0 * 2 + 1 * 1 errechnen und beträgt 0010 0001. Bei den Zahlen 20, 66 und 97 wird entsprechend vorgegangen.

$2^6 = 64$	**$2^5 = 32$**	**$2^4 = 16$**	**$2^3 = 8$**	**$2^2 = 4$**	**$2^1 = 2$**	**$2^0 = 1$**		
0	1	0	0	0	0	0	1 * 32	= 32
0	1	0	0	0	0	1	32 + 1	= 33
0	0	1	0	1	0	0	16 + 4	= 20
1	0	0	0	0	1	0	64 + 2	= 66
1	1	0	0	0	1	0	64 + 32 + 2	= 97

b) Oktales Zahlensystem

Oktale Zahlen, d. h. Zahlen zur Basis 8, können auch in dezimale Zahlen umgewandelt werden. Wobei die Zahlen auf zwei Arten dargestellt werden können: 0132 oder 132_8. Das oktale Zahlensystem umfaßt lediglich die Ziffern 0, 1, 2, 3, 4, 5, 6, 7, sprich es gibt die Zahlen 8 und 9 nicht.

(1) Umrechnungsbeispiel von oktal nach dezimal

Von oktal 132_8 nach dezimal 90:

1	3	2	
$+ 1 * 8^2$	$+ 3 * 8^1$	$+ 2 * 8^0$	
+ 64	+ 24	+ 2	= 90

(2) Zusammenfassende Umrechnungstabelle von oktal nach dezimal

$8^3 = 512$	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$		
		1	2	8 + 2 * 1	= 10
	2	0	0	2 * 64	= 128
1	0	1	0	512 + 8	= 520
	3	1	1	3 * 64 + 8 + 1	= 200

c) Hexadezimales Zahlensystem

Nun wollen wir noch Zahlen in das Hexadezimale Zahlensystem umrechnen: Hexadezimale Zahlen, d. h. Zahlen zur Basis 16, werden wie folgt geschrieben: 0x12a oder $12a_{16.}$ Die Zahlen gehen von 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e und f, wobei a für 10, b für 11, c für 12, d für 13, e für 14 und f für 15 steht.

(1) Umrechnungsbeispiel von hexadezimal nach dezimal

Von hexadezimal $12a_{16}$ nach dezimal 1:

1	2	a	
$+ 1 * 16^2$	$+ 2 * 16^1$	$+ a * 16^0$	
+ 256	+ 32	+ 10	= 298

(2) Zusammenfassende Umrechnungstabelle von hexadezimal nach dezimal

$16^3 = 1024$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$		
	1	2	a	256 + 32 + 10	= 298
1	0	0	1	1024 + 1	= 1025
	1	1	b	256 + 16 + 11	= 283
1	0	0	f	1024 + 15	= 1039

d) Aufstellungen Zahlensysteme

Hier eine Art „Umrechnungstabelle“ zwischen den verschiedenen Zahlensystemen:

Dezimale Zahlen	Binäre Zahlen	Oktale Zahlen	Hexadezimale Zahlen
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10

Da eine Zahl vom Datentyp int, 32 mögliche Bits hat, hier noch einige ausgewählte Umrechnungsbeispiele mit 32 Bit. Hierbei ist zu beachten, daß nur die linken Einsen und nicht die Nullen der binären Zahlen auf dem Bildschirm ausgegeben werden. Die linken Nullen stehen hier nur, um das Ergebnis besser veranschaulichen zu können.

Dezimale Zahlen	Binäre Zahlen	Hexadezimale Zahlen
0	0000 0000 0000 0000 0000 0000 0000 0000	0x0
1	0000 0000 0000 0000 0000 0000 0000 0001	0x1
2	0000 0000 0000 0000 0000 0000 0000 0010	0x2
15	0000 0000 0000 0000 0000 0000 0000 1111	0xf
31	0000 0000 0000 0000 0000 0000 0001 1111	0x1f
-1	1111 1111 1111 1111 1111 1111 1111 1111	0xffffffff
-2	1111 1111 1111 1111 1111 1111 1111 1110	0xfffffffe
-3	1111 1111 1111 1111 1111 1111 1111 1101	0xfffffffd
-4	1111 1111 1111 1111 1111 1111 1111 1100	0xfffffffc
-9	1111 1111 1111 1111 1111 1111 1111 0111	0xfffffff7

2. Übungen

(1) Frage

Rechnen Sie bitte 101_2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 5
b) 65
c) 257
d) Keine dieser Möglichkeiten.

(2) Frage

Rechnen Sie bitte 101_8 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 5
b) 65
c) 257
d) Keine dieser Möglichkeiten.

(3) Frage

Rechnen Sie bitte 101_{16} in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 5
b) 65
c) 257
d) Keine dieser Möglichkeiten.

(4) Frage

Rechnen Sie bitte 0x4e in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 31
b) 30
c) 40
d) Keine dieser Möglichkeiten.

(5) Frage

Rechnen Sie bitte 0x3f in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 64
b) 62
c) 63
d) Keine dieser Möglichkeiten.

(6) Frage

Rechnen Sie bitte 013 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 12
b) 13
c) 11
d) Keine dieser Möglichkeiten.

(7) Frage

Rechnen Sie bitte 0x44 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 36
b) 68
c) 69
d) Keine dieser Möglichkeiten.

(8) Frage

Rechnen Sie bitte 044 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 36
b) 68
c) 69
d) Keine dieser Möglichkeiten.

(9) Frage

Rechnen Sie bitte 1011_2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 11
b) 10
c) 12
d) Keine dieser Möglichkeiten.

(10) Frage

Rechnen Sie bitte 11001_2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 24
b) 23
c) 25
d) Keine dieser Möglichkeiten.

(11) Frage

Rechnen Sie bitte 100000_2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 31
b) 30
c) 32
d) Keine dieser Möglichkeiten.

(12) Frage

Rechnen Sie bitte 111_2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 6
b) 7
c) 8
d) Keine dieser Möglichkeiten.

(13) Frage

Rechnen Sie bitte 02 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 2
b) 3
c) 4
d) Keine dieser Möglichkeiten.

(14) Frage

Rechnen Sie bitte 0x2 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 2
b) 3
c) 4
d) Keine dieser Möglichkeiten.

(15) Frage

Rechnen Sie bitte 0x22 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 32
b) 33
c) 34
d) Keine dieser Möglichkeiten.

(16) Frage

Rechnen Sie bitte 0x202 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 130
b) 514
c) 258

d) Keine dieser Möglichkeiten.

(17) Frage

Rechnen Sie bitte 0202 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 130
b) 514
c) 258
d) Keine dieser Möglichkeiten.

(18) Frage

Rechnen Sie bitte 1111 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 15
b) 14
c) 16
d) Keine dieser Möglichkeiten.

(19) Frage

Rechnen Sie bitte 11111 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 30
b) 31
c) 32
d) Keine dieser Möglichkeiten.

(20) Frage

Rechnen Sie bitte 111000 in eine dezimale Zahl um. Welches Ergebnis erhalten Sie?

a) 58
b) 56
c) 57
d) Keine dieser Möglichkeiten.

3. Lösungen

(1) Frage

a

(2) Frage

b

(3) Frage
c

(4) Frage
d

(5) Frage
c

(6) Frage
c

(7) Frage
b

(8) Frage
a

(9) Frage
a

(10) Frage
c

(11) Frage
c

(12) Frage
b

(13) Frage
a

(14) Frage
a

(15) Frage
c

(16) Frage
b

(17) Frage
a

(18) Frage
a

(19) Frage
b

(20) Frage
b

4. Der bitweise Not-Operator ~

Der Not-Operator ~ wandelt bei binärer Schreibweise 0 in 1 und 1 in 0 um. Untenstehend wird die Zahl 6 mittels einer bitweisen Operation, mit dem Not-Operator ~, in –7 umgewandelt. Hierbei gilt zu beachten, daß man sich bei der Zahl 6, dargestellt als 110, noch die Nullen dazu denken muß. Die Zahl ausführlich dargestellt, müßte, da es sich um einen int-Wert handelt, eigentlich wie folgt aussehen: 0000 0000 0000 0000 0000 0000 0000 0110. Die Funktion toBinaryString(i) wandelt für Sie die Zahl 6 in eine binäre Zahl um.

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 6;
		System.out.println(Integer.toBinaryString(i));
		System.out.println(Integer.toBinaryString(~i));
		System.out.println(i);
		System.out.println(~i);
	}
}
```

Ausgabe auf der Konsole:

```
110
11111111111111111111111111111001
6
-7
```

Nun denken Sie sicherlich, das dies furchtbar kompliziert ist. Es gibt aber eine sehr einfache Methode, viel schneller zum Ergebnis zu kommen. Für den Not-Operator gilt die Zuweisung ~x == -x -1, die immer zutrifft und das „kippen“ der Bits für eine Berechnung des Ergebnisses entbehrlich macht. Wenden Sie diese Formel an, so erhalten Sie das Ergebnis –7 (~x == -6 –1).

5. Der bitweise And-Operator

Bei dem bitweisen And-Operator werden zwei binäre Zahlen miteinander verglichen. Die binäre Zahl 1 erhalten Sie nur als Resultat, wenn bei beiden Zahlen an dieser Stelle eine 1 steht, andernfalls wird 0 zurückgegeben.

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(Integer.toBinaryString(i));
		System.out.println(Integer.toBinaryString(b));
		System.out.println(Integer.toBinaryString(i & b));
	}
}
```

Ausgabe auf der Konsole:

```
1101
1100
1100
```

6. Der bitweise Und-Operator

Im Gegensatz zum Oder-Operator muß mindestens eine von beiden Zahlen an dieser Stelle eine 1 haben, damit als Ergebnis eine 1 herauskommt.

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(Integer.toBinaryString(i));
		System.out.println(Integer.toBinaryString(b));
		System.out.println(Integer.toBinaryString(i | b));
	}
}
```

Ausgabe auf der Konsole:

```
1101
1100
1101
```

7. Der bitweise XOR-Operator

Der bitweise XOR-Operator hat als Ergebnis 1, wenn nur einer von zwei bits eine 1 ist, andernfalls wird null zurückgegeben.

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(Integer.toBinaryString(i));
		System.out.println(Integer.toBinaryString(b));
		System.out.println(Integer.toBinaryString(i ^ b));
	}
}
```

Ausgabe auf der Konsole:

```
1101
1100
1
```

8. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(i^b);
	}
}
```

a) 1
b) 13
c) 12
d) Keine dieser Möglichkeiten

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(i | b);
	}
}
```

a) 1
b) 13
c) 12
d) Keine dieser Möglichkeiten

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
	public static void main(String[ ] args) {
		int i = 13;
		int b = 12;
		System.out.println(i & b);
	}
}
```

a) 1
b) 13

c) 12
d) Keine dieser Möglichkeiten

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
	public static void main(String[ ] args) {
		int i = 20;
		int b = 37;
		System.out.println(i & b);
	}
}
```

a) 4
b) 20
c) 5
d) 37
e) Keine dieser Möglichkeiten

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 20;
		int b = 37;
		System.out.println(~i);
		System.out.println(~b);
	}
}
```

a) –21 –38
b) –20 – 37
c) 21 38
d) 20 38
e) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class BitweiseOperatoren {
	public static void main(String[ ] args) {
		int i = 2;
		int b = 3;
		System.out.println(~i);
		System.out.println(~b);
```

```
        }
}
```

a) –2 –3
b) –3 – 4
c) 3 4
d) 2 3
e) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet??

```
public class B {
        public static void main(String[ ] args) {
                int i = 2;
                int b = 3;
                System.out.println(i | b);
        }
}
```

a) 4
b) 2
c) 3
d) 5
e) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
        public static void main(String[ ] args) {
                int i = 2;
                int b = 3;
                System.out.println(i ^ b);
        }
}
```

a) 4
b) 2
c) 3
d) 5
e) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
        public static void main(String[ ] args) {
                int i = 8;
                int b = 9;
                System.out.println(i ^ b);
        }
}
```

a) 4
b) 2
c) 3
d) 1
e) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
        public static void main(String[ ] args) {
                int i = 8;
                int b = 9;
                System.out.println(i | b);
        }
}
```

a) 4
b) 2
c) 3
d) 1
e) Keine dieser Möglichkeiten.

9. Lösungen

(1) Frage

a

(2) Frage

b

(3) Frage

c

(4) Frage

a

(5) Frage
a

(6) Frage
b

(7) Frage
c

(8) Frage
e

(9) Frage
d

(10) Frage
e

F. Der Shift-Operator (Verschiebeoperator)

1. Der Links-Shift: <<

Mit diesem Operator wird ein Bit um eine gewisse Zahl nach links verschoben und von rechts sowohl bei positiven als auch bei negativen Zahlen mit Nullen aufgefüllt. In unserem untenstehenden Beispiel wird das Bit 1 immer um eins weiter nach links verschoben. Intern wird die dezimale Zahl eins in eine binäre Zahl umgewandelt und es wird wieder eine dezimale Zahl zurückgegeben. Unten erscheint allerdings eine binäre Zahl, da die dezimale Zahl wieder in eine Binäre mit Hilfe der Methode Integer.toBinaryString(int i) geändert wird.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = 1;
                int b = a << 2;
                int c = a << 3;
                int d = a << 4;
                System.out.println(Integer.toBinaryString(a));
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toBinaryString(c));
                System.out.println(Integer.toBinaryString(d));
                }
}
```

Ausgabe auf der Konsole:

```
1
100
1000
10000
```

Bei einem Shift um 32 bleibt das Ergebnis unverändert, da es genau 32 Bits bei einer Integerzahl gibt, sprich es wird um 32 geshiftet, wobei wieder die Ursprungsposition erreicht wird.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = 20;
                int b = a << 32;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                }
}
```

Ausgabe auf der Konsole:

```
20
10100
20
10100
```

Bei einem Shift um 36 wird nach 32 wieder die Ursprungsposition erreicht und anschließend wird die binäre Zahl nochmals um 4 verschoben.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = 20;
                int b = a << 36;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                }
}
```

Ausgabe auf der Konsole:

```
20
10100
320
101000000
```

Es besteht auch die Möglichkeit, hexadezimale Zahlen mit dem Links-Shift-Operator zu verändern und das Ergebnis wieder mit Hilfe der Methode Integer.toHexString(int i) als hexadezimale Zahl auszugeben. Wollen Sie allerdings eine dezimale Zahl als Ergebnis haben, können Sie darauf verzichten, diese Methode anzuwenden. Um sich Rechenarbeit sparen zu können, noch ein paar Bemerkungen: Es werden immer 4 Zeichen aus dem binären System zu einer hexadezimalen Ziffer zusammengefasst. So werden 4 Einsen (1111) im Hexadezimalsystem als f dargestellt, vier Nullen (0000) als O und 1100 als c (siehe Aufstellung Zahlensysteme im Kapitel integer Bitweise Operatoren).

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = 0xffffffff;
                int b = a << 2;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(Integer.toHexString(a));
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toHexString(b));
                }
}
```

Ausgabe auf der Konsole:

```
-1
11111111111111111111111111111111
ffffffff
-4
11111111111111111111111111111100
fffffffc
```

2. Der Rechts-Shift-Operator: >>

Der Rechts-Shift-Operator füllt eine positive binäre Zahl von links mit Nullen auf. Eine Eins wird also von links nach rechts verschoben, wobei sie das Vorzeichen behält.

```
public class ShiftOperator {
	public static void main(String[ ] args) {
		int a = 20;
		int b = a >> 2;
		System.out.println(a);
		System.out.println(Integer.toBinaryString(a));
		System.out.println(Integer.toHexString(a));
		System.out.println(b);
		System.out.println(Integer.toBinaryString(b));
		System.out.println(Integer.toHexString(b));
		}
}
```

Ausgabe auf der Konsole:

```
20
10100
14
5
101
5
```

Der Rechts-Shift-Operator füllt eine negative binäre Zahl von links mit Einsen auf, so wird eine eins von links nach rechts verschoben, und sie behält dabei das Vorzeichen.

```
public class ShiftOperator {
	public static void main(String[ ] args) {
		int a = -20;
		int b = a >> 2;
		System.out.println(a);
		System.out.println(Integer.toBinaryString(a));
		System.out.println(Integer.toHexString(a));
```

```
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toHexString(b));
                }
}
```

Ausgabe auf der Konsole:

```
-20
11111111111111111111111111101100
ffffffec
-5
11111111111111111111111111111011
fffffffb
```

Wird allerdings die Zahl –1 um eine beliebige Zahl nach rechts „geshiftet“, kommt immer –1 als Ergebnis heraus, da von links mit Einsen aufgefüllt wird.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = -1;
                int b = a >> 2;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(Integer.toHexString(a));
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toHexString(b));
                }
}
```

Ausgabe auf der Konsole:

```
-1
11111111111111111111111111111111
ffffffff
-1
11111111111111111111111111111111
ffffffff
```

Genauso verhält es sich mit der Zahl 0, da diese von links immer mit 0 aufgefüllt wird, wobei nur die 0 auf der Konsole ausgegeben wird.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = 0;
                int b = a >> 2;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(Integer.toHexString(a));
```

```
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toHexString(b));
                }
}
```

Ausgabe auf der Konsole:

```
0
0
0
0
0
0
```

3. Vorzeichenloser Rechts-Shift: >>>

Hierbei wird sowohl bei positiven als auch bei negativen Zahlen von links mit Nullen aufgefüllt und die binären Zahlen werden nach rechts verschoben. Dies führt nur bei negativen Zahlen zu anderen Ergebnissen als beim Rechts-Shift. Die Nullen, die von links hinzugefügt wurden, sind bei der Ausgabe auf der Konsole unsichtbar.

```
public class ShiftOperator {
        public static void main(String[ ] args) {
                int a = -20;
                int b = a >>> 2;
                System.out.println(a);
                System.out.println(Integer.toBinaryString(a));
                System.out.println(Integer.toHexString(a));
                System.out.println(b);
                System.out.println(Integer.toBinaryString(b));
                System.out.println(Integer.toHexString(b));
                }
}
```

Ausgabe auf der Konsole:

```
-20
11111111111111111111111111101100
ffffffec
1073741819
11111111111111111111111111011 //Von links wurde mit Nullen aufgefüllt!
3fffffb
```

4. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0xfffffff4;
                int b = a >> 6;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xfffffff4 0xffffffff
b) 0xfffffff4 0xfffffffe
c) 0xfffffff4 0xfffffd00
d) 0xfffffff4 0xfffffffc
e) 0xfffffff4 0x3fffffff
f) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0xfffffff4;
                int b = a >>> 6;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xfffffff4 0xffffffff
b) 0xfffffff4 0xfffffffe
c) 0xfffffff4 0xfffffd00
d) 0xfffffff4 0xfffffffc
e) 0xfffffff4 0x3fffffff
f) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0xfffffff4;
                int b = a << 6;
```

```
                    System.out.println(Integer.toHexString(a));
                    System.out.println(Integer.toHexString(b));
                    }
}
```

a) 0xfffffff4 0xffffffff
b) 0xfffffff4 0xfffffffe
c) 0xfffffff4 0xfffffd00
d) 0xfffffff4 0xfffffffc
e) 0xfffffff4 0x3fffffff
f) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                    int a = 0xffffffff;
                    int b = a >> 2;
                    System.out.println(Integer.toHexString(a));
                    System.out.println(Integer.toHexString(b));
                    }
}
```

a) 0xffffffff 0xffffffff
b) 0xffffffff 0xfffffffe
c) 0xffffffff 0xfffffd00
d) 0xffffffff 0xfffffffc
e) 0xffffffff 0x3fffffff
f) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                    int a = 0x00000000;
                    int b = a >> 6;
                    System.out.println(Integer.toHexString(a));
                    System.out.println(Integer.toHexString(b));
                    }
}
```

a) 0 0
b) 0x00000000 0x00000000
c) 0x00000000 0x00000020
d) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String[ ] args) {
                int a = 0x00000000;
                int b = a >>> 6;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0 0
b) 0x00000000 0x00000000
c) 0x00000000 0x00000020
d) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 0xffffffff;
                int b = a >>> 2;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xffffffff 0xffffffff
b) 0xffffffff 0x3fffffff
c) 0xffffffff 0xfffffff3
d) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 0xffffffff;
                int b = a >> 32;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xffffffff 0xffffffff

b) 0xffffffff 0x3fffffff
c) 0xffffffff 0xfffffff3
d) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 0xffffffff;
                int b = a >>> 32;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xffffffff 0xffffffff
b) 0xffffffff 0x3fffffff
c) 0xffffffff 0xfffffff3
d) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 0x1f;
                int b = a >>> 33;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0x1f 0x1f
b) 0x1f 0xf
c) 0x1f 0x3e
d) Keine dieser Möglichkeiten.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 0x1f;
                int b = a << 33;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
```

```
                    }
}
```

a) 0x1f 0x1f
b) 0x1f 0xf
c) 0x1f 0x3e
d) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 23;
                int b = a << 33;
                System.out.println(a);
                System.out.println(b);
                }
}
```

a) 23 47
b) 23 11
c) 23 46
d) 23 12
e) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = 23;
                int b = a >> 33;
                System.out.println(a);
                System.out.println(b);
                }
}
```

a) 23 47
b) 23 11
c) 23 46
d) 23 12
e) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
        public static void main(String [ ] args) {
                int a = -23;
                int b = a >>> 33;
                System.out.println(Integer.toHexString(a));
                System.out.println(Integer.toHexString(b));
                }
}
```

a) 0xffffffe9 7ffffff4
b) 0xffffffd9 7ffffff5
c) 0xffffffe9 7ffffff5
d) Keine dieser Möglichkeiten.

5. Lösungen

(1) Frage
a

(2) Frage
e

(3) Frage
c

(4) Frage
a

(5) Frage
a

(6) Frage
a

(7) Frage
b

(8) Frage
a

(9) Frage
a

(10) Frage
b

(11) Frage
c

(12) Frage
c

(13) Frage
b

(14) Frage
a

VI. Kontrollstrukturen

A. Selektionen

1. Die if-Struktur

Mit der if-Struktur ist es möglich, zwischen mindestens 2 Möglichkeiten zu wählen. Wenn in untenstehendem Beispiel die Anrede Frau ist, soll der Satz „Ich bin eine Frau!“ auf der Konsole erscheinen. Da die Bedingung zutrifft und der boolean-Ausdruck Anrede == „Frau“ den Wert true zurück gibt, wird der Satz „Ich bin eine Frau!“ ausgegeben. Die Anweisung der if-Struktur wird nur ausgeführt, wenn die dazugehörige Bedingung wahr ist. Die geschweiften Klammern { } der if-Struktur können weggelassen werden.

```
public class IfElse {
	public static void main(String[ ] args) {
		String Anrede = “Frau”;

		//Wenn der Inhalt der Variable identisch Frau ist
		if (Anrede == “Frau“){
			//Soll folgender Satz ausgegeben werden:
			System.out.println(“Ich bin eine Frau!“);
		}
	}
}
```

Ausgabe auf der Konsole:

Ich bin eine Frau!

Ist allerdings der Inhalt der Variable das Wort Mann, so wird in unserem Beispiel gar nichts ausgegeben. Der boolean-Wert gibt nun false zurück, da Mann nicht identisch mit Frau ist, so wird der Befehl, den Satz „Ich bin eine Frau!“ zu schreiben, nicht ausgeführt. Das Programm überspringt nun diese Anweisung.

```
public class IfElse {
	public static void main(String[ ] args) {
		String Anrede = “Mann”;

		//Wenn der Inhalt der Variable nicht identsich Frau ist,
		if (Anrede == “Frau“){
			//wird dieser Satz nicht auf der Konsole ausgegeben:
			System.out.println(“Ich bin eine Frau!“);
		}
	}
}
```

Ausgabe auf der Konsole:

Es wird nichts ausgegeben!

2. Die if-else-Struktur

Will ich aber auch die Alternative haben, den Satz „Ich bin ein Mann!“ auszugeben, brauche ich zusätzlich einen sogenannten else-Zweig. Nun erscheint für den Fall, daß die Variable Anrede den Inhalt Mann hat, der Satz „Ich bin ein Mann!“ auf der Konsole.

```
public class IfElse {
	public static void main(String[ ] args) {
		String Anrede = “Mann”;

		if (Anrede == “Frau”){
			System.out.println(“Ich bin eine Frau!”);
		}
		else{
			System.out.println(“Ich bin ein Mann!”);
		}
	}
}
```

Ausgabe auf der Konsole:

Ich bin ein Mann!

3. Die else-if-Struktur

Wollen wir einen zusätzlichen Satz für ein Kind erstellen, brauchen wir eine else-if-Struktur.

```
public class IfElse {
	public static void main(String[ ] args) {
		String Anrede = ”Kind“;

		if (Anrede == “Frau”){
			System.out.println(“Ich bin eine Frau!”);
		}
		else if (Anrede == “Mann”){
			System.out.println(“Ich bin ein Mann!”);
		}
		else if (Anrede == ”Kind“){
			System.out.println(“Ich bin ein Kind!”);
		}
	}
}
```

Ausgabe auf der Konsole:

Ich bin ein Kind!

In der Klammer nach dem if steht immer ein boolean-Ausdruck. Wir erinnern uns, daß dieser nur 2 Zustände annehmen kann, und zwar true oder false (vgl. Kapitel Boolean-Literale). Der if-Zweig wird aber nur ausgeführt, wenn der boolean-Ausdruck true ist, bei false wird die Anweisung übersprungen. So ist von untenstehendem Beispiel das Ergebnis auf der Konsole das Wort zwei.

```
public class IfElse {
	public static void main(String[ ] args) {
		boolean b = false;

		if (b){
			System.out.println(”eins“);
		}
		else if (b = true){
			System.out.println(“zwei”);
		}
	}
}
```

Ausgabe auf der Konsole:

zwei

Für den Fall, daß es zwei boolean-Ausdrücke gibt, die wie in untenstehendem Beispiel, true zurückgeben, wird nur die erste if-Anweisung mit true ausgeführt, danach springt das Programm weiter. Der dritte boolean-Ausdruck gibt false zurück, da b im zweiten Boolean-Ausdruck auf true gesetzt wurde und !b (nicht true) false ist.

```
public class IfElse {
	public static void main(String[ ] args) {
		boolean b;

		if (b = true){
			System.out.println(“eins”);
		}
		else if (b = true){
			System.out.println(“zwei”);
		}
		else if (!b){
			System.out.println(“drei”);
		}
	}
}
```

Ausgabe auf der Konsole:

eins

Steht in der Klammer nach dem if oder else if ein Vergleich und keine Zuweisung, wird in untenstehendem Beispiel zwei auf der Konsole ausgegeben.

```
public class IfElse {
	public static void main(String[ ] args) {
		boolean b = false;

		if (b == true){
			System.out.println("eins");
		}
		else if (b == false){
			System.out.println("zwei");
		}
	}
}
```

Ausgabe auf der Konsole:

zwei

Eine Zuweisung zu einem Begriff wie Frau (Anrede = "Frau") führt allerdings zu einem Kompilierfehler, da dies im Gegensatz zu b = true kein boolean-Ausdruck ist, und in der Klammer nach if muß immer ein boolean-Ausdruck stehen.

```
public class IfElse {
	public static void main(String[ ] args) {
		String Anrede = "Frau";

		if (Anrede = "Frau"){
			System.out.println("Ich bin eine Frau!");
		}
	}
}
```

Das gleiche gilt für den Fall, daß der boolean-Ausdruck nach dem if oder else durch if ("a") ersetzt wird, es kommt zu einem Kompilierfehler.

4. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 ;
        if (b1 = false){
            System.out.println("Hund");
        }
        else if(b1 = true){
            System.out.println("Katze");
        }
        else if(!b1){
            System.out.println("Maus");
        }
    }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 ;
        if (b1 = true){
            System.out.println("Hund");
        }
        else if(b1 = true){
            System.out.println("Katze");
        }
        else if(!b1){
            System.out.println("Maus");
        }
    }
}
```

a) Hund
b) Katze

c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 = true;
        if (b1 = false){
            System.out.println("Hund");
        }
        else if(b1 = false){
            System.out.println("Katze");
        }
        else if(!b1){
            System.out.println("Maus");
        }
    }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 = true;
        if (b1 = false){
            System.out.println("Hund");
        }
        else if(!b1){
            System.out.println("Katze");
        }
        else if(!b1){
            System.out.println("Maus");
        }
```

```
    }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 = true;
        if (b1 = false){
            System.out.println("Hund");
        }
        else if(b1 = true){
            System.out.println("Katze");}
        else if(!b1){
            System.out.println("Maus");
        }
    }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 = true;
        if (b1 == false){
            System.out.println("Hund");
        }
        else if(b1 == false){
```

```
                System.out.println("Katze");
            }
            else if(!b1){
                System.out.println("Maus");}
        }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b1 = true;
        if (b1 == true){
            System.out.println("Hund");
        }
        else if(b1 == false){
            System.out.println("Katze");
        }
        else if(!b1){
            System.out.println("Maus");
        }
    }
}
```

a) Hund
b) Katze
c) Maus
d) Katze Maus
e) Maus Katze
f) Hund Katze Maus
g) Maus Katze Hund
h) Keine dieser Möglichkeiten

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
```

```
        boolean b1 = true;
        if (b1 == false){
            System.out.println("Hund");
        }
        else if(!b1){
            System.out.println("Maus");
        } }
}
```

a) Hund
b) Maus
c) Maus Hund
d) Hund Maus
e) Keine dieser Möglichkeiten

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = "A";
        if (s == "A"){
            System.out.println("A");
        }
        else if(s == "B"){
            System.out.println("B");  }}}
```

a) A
b) B
c) A B
d) B A
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = "B";
        if (s == "A"){
            System.out.println("A");
        }
        else if(s == "B"){
            System.out.println("B");
        }}}
```

a) A

b) B
c) A B
d) B A
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “A”;
        if (s = “A”){
            System.out.println(”A“);
        }
        else if(s == “B”){
            System.out.println(”B“);
        }
    }
}
```

a) A
b) B
c) A B
d) B A
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “a”;
        if (s == “A”){
            System.out.println(”A“);
        }
        else if(s == “B”){
            System.out.println(”B“);
        }
    }
}
```

a) A
b) B
c) A B

d) B A
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten

5. Lösungen

(1) Frage
b

(2) Frage
a

(3) Frage
c

(4) Frage
b

(5) Frage
b

(6) Frage
h, Achtung! Hier steht b1 == false, dies ist ein Vergleich und keine Zuweisung!

(7) Frage
a, Achtung! Hier steht b1 == true, dies ist ein Vergleich und keine Zuweisung!

(8) Frage
e, Achtung! Hier steht b1 == false, dies ist ein Vergleich und keine Zuweisung!

(9) Frage
a

(10) Frage
b

(11) Frage
e, In der Klammer nach dem If muß eine Bedingung stehen, dort steht aber eine Zuweisung s = „A“. Um den Kompilierfehler zu vermeiden, müsste dort ein Vergleich stehen: s == „A“!

(12) Frage
g, Java unterscheidet zwischen Groß- und Kleinbuchstaben.

6. Die switch-Struktur

Wie wir gesehen haben, ist die if-else-Struktur für mehrere Auswahlmöglichkeiten unübersichtlich, deswegen ist es sinnvoll in diesen Fällen auf die switch-Struktur zurückzugreifen. In der untenstehenden Klasse SwitchStruktur wird die Zahl 1 mit den verschiedenen Fällen, auch Label oder case genannt, verglichen und es erscheint „Ich bin Fall 1“ auf der Konsole. Für alle Labels, für die weder Fall 1 noch Fall 2 zutrifft, tritt automatisch der Standardfall ein, der auch Default-Fall genannt wird. In der Klammer nach der switch-Anweisung dürfen nur folgende Datentypen stehen: char, byte, short und int. Enthält die Klammer einen anderen Datentyp, kommt es zu einem Kompilierfehler.

```
public class SwitchStruktur {
        public static void main(String[ ] args) {
        //x darf aus folgenden Datentypen bestehen: char, byte, short oder int
                int x = 1;
                switch (x){
                        case 1: System.out.println("Ich bin Fall 1"); break;
                        case 2: System.out.println("Ich bin Fall 2"); break;
                        default: System.out.println("Ich bin der Standardfall");
                }
        }
}
```

Ausgabe auf der Konsole:

```
Ich bin Fall 1
```

Vergißt man die break-Anweisungen am Ende jedes Falles, werden alle anderen Fälle auch abgearbeitet. Hinter dem Standardfall steht kein break, da dies der letzte Fall ist.

```
public class SwitchStruktur {
        public static void main(String[ ] args) {
                int x = 1;
                switch (x){
                        case 1: System.out.println("Ich bin Fall 1");
                        case 2: System.out.println("Ich bin Fall 2");
                        default: System.out.println("Ich bin der Standardfall");
                }
        }
}
```

Ausgabe auf der Konsole:

```
Ich bin Fall 1
Ich bin Fall 2
Ich bin der Standardfall
```

Die Reihenfolge der verschiedenen Labels spielt keine Rolle, case 2 kann vor case 1 stehen und auch der Standardfall muß nicht der Letzte sein. Befindet sich der default-Fall nicht am Ende, sollte dahinter ein break stehen, da sonst, wie man in untenstehendem Beispiel sehen kann, auch der nächste Fall mit abgearbeitet wird.

```
public class SwitchStruktur {
        public static void main(String[ ] args) {
                int x = 0;
                switch (x){
                        default: System.out.println("Ich bin der Standardfall");
                        case 1: System.out.println("Ich bin Fall 1"); break;
                        case 2: System.out.println("Ich bin Fall 2"); break;
                }
        }
}
```

Ausgabe auf der Konsole:

```
Ich bin der Standardfall
Ich bin Fall 1
```

7. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		short a = 1;

		switch (a){
			case 1: System.out.println(" 1 "); break;
			case 2: System.out.println(" 2 "); break;
			default: System.out.println(" Standard ");
		}
	}
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		short a = 1;

		switch (a){
			case 1: System.out.println(" 1 ");
			case 2: System.out.println(" 2 ");
			default: System.out.println(" Standard ");
		}
	}
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String a = 1;

        switch (a){
            case 1: System.out.println(" 1 "); break;
            case 2: System.out.println(" 2 "); break;
            default: System.out.println(" Standard ");
        }
    }
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        short a = 5;

        switch (a){
            case 1: System.out.println(" 1 "); break;
            case 2: System.out.println(" 2 "); break;
            default: System.out.println(" Standard ");
        }
    }
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		short a = 1;

		switch (a){
			default: System.out.println(" Standard ");
			case 1: System.out.println(" 1 ");
			case 2: System.out.println(" 2 ");
		}
	}
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		short a = 1;

		switch (a){
			default: System.out.println(" Standard ");
			case 2: System.out.println(" 2 ");
			case 1: System.out.println(" 1 ");
		}
	}
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        boolean b;

        switch (b){
            default: System.out.println(" Standard ");
            case 2: System.out.println(" 2 ");
            case 1: System.out.println(" 1 ");
        }
    }
}
```

a) 1 2 Standard
b) 1 Standard 2
c) Standard 1 2
d) Standard
e) 1
f) 2
g) Kompilierfehler
h) Keine dieser Möglichkeiten.

8. Lösungen

(1) Frage

e

(2) Frage

a

(3) Frage

g

(4) Frage

d

(5) Frage

h

(6) Frage

e

(7) Frage

g

B. Iterationen

1. Die while-Schleife

Will man z. B. eine Adressliste aller Kunden aus einer Datenbank auslesen, verwendet man eine sogenannte while-Schleife. Solange also nicht der letzte Kunde erreicht worden ist, soll jede weitere Adresse ausgedruckt werden.

Wir fangen mit einer einfachen while-Schleife an, die die Zahlen 0 bis 4 ausgeben soll. Am Anfang benötigen wir einen Einstiegspunkt, eine deklarierte und initialisierte Variable, hier in unserem Fall ist dies int i = 0. So ist der erste Wert, der ausgegeben wird, die Zahl 0. Dann braucht man eine Abbruchbedingung, die vor Durchlauf der Schleife überprüft wird. Diese besteht aus einem boolean-Ausdruck i<5, der zur Folge hat, daß die Schleife solange durchlaufen wird, bis der Ausdruck nicht mehr true ist. In unserem Fall bedeutet dies, die Schleife wird 5 Mal abgearbeitet und als letztes erscheint die Zahl 4. 4 ist die letzte Zahl, für die die Bedingung kleiner 5 wahr ist. Der Operator i++ sorgt innerhalb der Schleife dafür, daß die Variable bei jedem Durchlauf um 1 erhöht wird. Die Variable i wird auch als Schleifenzähler bezeichnet.

```
5  public class WhileSchleife {
6
7      public static void main(String[] args) {
8
9          //Einstiegspunkt
10         int i = 0;
11         //Bedingung für den Ausstiegspunkt
12         while (i<5){
13             //Ausgabe
14             System.out.println("Dies ist der Durchlauf Nr. " + i);
15             //Erhöhen von i um 1 bei jedem Durchlauf
16             i++;
17         }
18     }
19 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> WhileSchleife [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (10.05.2005 11:41:37)
Dies ist der Durchlauf Nr. 0
Dies ist der Durchlauf Nr. 1
Dies ist der Durchlauf Nr. 2
Dies ist der Durchlauf Nr. 3
Dies ist der Durchlauf Nr. 4
```

Sollte die Abbruchbedingung der while-Schleife gleich zu false führen, so wird die Schleife kein einziges Mal durchlaufen und es wird nichts ausgegeben. Dies ist in unserem Beispiel für i<0 der Fall.

```
public class WhileSchleife {

        public static void main(String[ ] args) {
```

```
                int i = 0;
                while (i<0){
                        System.out.println("Dies ist der Durchlauf Nr. " + i);
                        i++;
                }
        }
}
```

Ausgabe auf der Konsole:

Steht in der Klammer nach dem while keine Bedingung, sondern etwas anderes, wie z. B. while (i), kommt es zu einem Kompilierfehler.

2. Die do-while-Schleife

Im Gegensatz zur while-Schleife wird die do-while-Schleife auf jeden Fall einmal durchlaufen, da die Abbruchbedingung nicht am Anfang der Schleife steht, sondern am Ende. Untenstehendes Beispiel führt sowohl für die do-while-Schleife als auch für die while-Schleife zum gleichen Ergebnis.

```
public class DoWhileSchleife {
        public static void main(String[ ] args) {
                //Einstiegspunkt
                int i = 0;
                do {
                        //Ausgabe
                        System.out.println("Dies ist der Durchlauf Nr. " + i);
                        //Erhöhen von i um 1 bei jedem Durchlauf
                        i++;
                //Bedingung für den Ausstiegspunkt
                }while (i < 5);
        }
}
```

Ausgabe auf der Konsole:

```
Dies ist der Durchlauf Nr. 0
Dies ist der Durchlauf Nr. 1
Dies ist der Durchlauf Nr. 2
Dies ist der Durchlauf Nr. 3
Dies ist der Durchlauf Nr. 4
```

Die do-while-Schleife führt bei der Bedingung i < 0 zu einem anderen Resultat wie die while-Schleife. Die do-while-Schleife wird mindestens einmal abgearbeitet, da sie erst nach dem ersten Durchlauf auf die Abbruchbedingung stößt.

```
public class DoWhileSchleife {
	public static void main(String[ ] args) {
		int i = 0;
		do {
			System.out.println("Dies ist der Durchlauf Nr. " + i);
			i++;
		}while (i < 0);
	}
}
```

Ausgabe auf der Konsole:

```
Dies ist der Durchlauf Nr. 0
```

3. Die for-Schleife

Die for-Schleife funktioniert vom Prinzip her wie eine while-Schleife, nur die Syntax ist geringfügig anders. Der Schleifenzähler i ist eine lokale Variable, sprich der Gültigkeitsbereich der Variablen erstreckt sich nur auf die for-Schleife. Die Variable i ist außerhalb der for-Schleife unbekannt.

```
public class ForSchleife {
	public static void main(String[ ] args) {
		//Einstiegsbedingung; Abbruchbedingung; Erhöhung um 1
		for(int i = 0; i < 5; i++){
			System.out.println("Dies ist der Durchlauf Nr." + i);
		}
	}
}
```

Ausgabe auf der Konsole:

```
Dies ist der Durchlauf Nr. 0
Dies ist der Durchlauf Nr. 1
Dies ist der Durchlauf Nr. 2
Dies ist der Durchlauf Nr. 3
Dies ist der Durchlauf Nr. 4
```

Des weiteren gibt es noch mehrere erlaubte Variationen für die Erstellung einer for-Schleife, so kann die Variable außerhalb der for-Schleife deklariert und innerhalb der Klammer der for-Schleife initialisiert werden.

```
public class ForSchleife {
	public static void main(String[ ] args) {
		//Deklaration des Schleifenzählers i
		int i;
		//Initialisierung des Schleifenzählers i
		for(i = 0; i < 5; i++){
```

```
                        System.out.println("Dies ist der Durchlauf Nr." + i);
                }
        }
}
```

Der Schleifenzähler i kann auch außerhalb deklariert und initialisiert werden, wobei darauf geachtet werden muß, daß in der Klammer nach for weiterhin der erste Strichpunkt stehen bleibt.

```
public class ForSchleife {
        public static void main(String[ ] args) {
                int i = 0,

                for(; i < 5; i++){
                        System.out.println("Dies ist der Durchlauf Nr." + i);
                }
        }
}
```

Der Operator i++ darf auch innerhalb der for-Schleife geschrieben werden, aber der Strichpunkt muß in der Klammer nach for erhalten bleiben, sonst kommt es zu einem Kompilierfehler.

```
public class ForSchleife {
        public static void main(String[ ] args) {
                for(int i = 0; i < 5;){
                        i++
                }
        }
}
```

Die Abbruchbedingung kann weggelassen werden und durch eine break-Anweisung innerhalb der for-Schleife ersetzt werden. Ohne break wäre es eine Endlosschleife, ebenso wie for (; ;).

```
public class ForSchleife {
        public static void main(String[ ] args) {
                for(int i = 0; ; i++){
                        break;
                }
        }
}
```

Folgende weitere Varianten der for-Schleife führen zu Kompilierfehlern:

1. Doppelte Variablendeklaration: int i; for(int i = 0; i<5; i++){ }

2. Doppelte Variablendeklaration und –intialisierung: int i = 0; for(int i = 0; i<5; i++){ }
3. Bedingung wird durch eine Zuweisung ersetzt, so ist i = 5 eine Zuweisung und i< 5 ist eine Bedingung: for(int i = 0; i=5; i++){ }

a) Die for-Schleife mit mehreren Schleifenvariablen

Es besteht die Möglichkeit, eine for-Schleife mit mehreren Schleifenvariablen zu definieren.

```
public class ForSchleife {
	public static void main(String[ ] args) {
		for(int i = 0, j=5; i < j; i++, j--){
			System.out.println("Dies ist der Durchlauf von i: " + i);
			System.out.println("Dies ist der Durchlauf von j: " + j);
		}
	}
}
```

Ausgabe auf der Konsole:

```
Dies ist der Durchlauf von i: 0
Dies ist der Durchlauf von j: 5
Dies ist der Durchlauf von i: 1
Dies ist der Durchlauf von j: 4
Dies ist der Durchlauf von i: 2
Dies ist der Durchlauf von j: 3
```

4. Die Sprungbefehle break und continue und markierte Anweisungen

Wir wollen nun innerhalb eines Programms Punkte markieren, zu denen man mit Hilfe von break und continue springen kann. Es ist so ähnlich wie bei Monopoly, wenn Sie eine Karte ziehen, die Ihnen z. B. mitteilt, daß Sie direkt in das Gefängnis gehen und erst einmal eine Pause einlegen sollen, bevor es weitergeht. In Java können Sie auch, den Ablauf der for-Schleife und switch-Struktur verändern, indem Sie das Programm veranlassen, zu einer Stelle weiter oben zurückzugehen. Die Anweisung break oben sorgt dafür, daß zu der markierten „Sprungmarke“ oben gesprungen und anschließend die Schleife abgebrochen wird. Wohingegen die Anweisung continue oben bewirkt, daß die for-Schleife fortgesetzt wird.

```
public class ForSchleife {
	public static void main(String[ ] args) {
		oben:
		for(int i = 0; i < 5; i++){
			switch (i){
			case 1: System.out.println("Ich bin Fall 1"); continue oben;
```

```
                    case 2: System.out.println("Ich bin Fall 2"); break oben;
                    default: System.out.println("Standard");
                    }
                }
        }
}
```

Ausgabe auf der Konsole:

```
Standard
Ich bin Fall 1
Ich bin Fall 2
```

5. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
	public static void main(String[ ] args) {
		for (int j = 1; j < 2; j++){
			switch (j){
				case 1: System.out.println("1");
				case 2: System.out.println("2");
				default: System.out.println("Standard");
			}
		}
	}
}
```

a) 1
b) 2
c) Standard
d) 12Standard
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
	public static void main(String[ ] args) {
		for (int j = 1; j = 2; j++){
			switch (j){
				case 1: System.out.println("1"); break;
				case 2: System.out.println("2");break;
				default: System.out.println("Standard");
			}
		}
	}
}
```

a) 1
b) 2
c) Standard
d) 12Standard
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class SwitchStruktur {
	public static void main(String[ ] args) {
		for (long j = 1; j < 2; j++){
			switch (j){
				case 1: System.out.println("1"); break;
				case 2: System.out.println("2"); break;
				default: System.out.println("Standard");
			}
		}
	}
}
```

a) 1
b) 2
c) Standard
d) 12Standard
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
	public static void main(String[ ] args) {
		for (int j = 1; j < 2; j++){
			switch (j){
				case 1: System.out.println("1"); break;
				 default: System.out.println("Standard");
				case 2: System.out.println("2");break;

			}
		}
	}
}
```

a) 1
b) 2
c) Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
	public static void main(String[ ] args) {
```

```
            int j = 1;
            for ( ; j < 2; j++){
                switch (j){
                    case 1: System.out.println("1"); break;
                    default: System.out.println("Standard");
                    case 2: System.out.println("2");break;

                }
            }
        }
}
```

a) 1
b) 2
c) Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
        int j = 1;
        for (j < 2; j++){
            switch (j){
                case 1: System.out.println("1"); break;
                default: System.out.println("Standard");
                case 2: System.out.println("2");break;

            }
        }
    }
}
```

a) 1
b) 2
c) Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
        int j;
```

```
            for (j = 1; j < 2; j++){
                switch (j){
                    case 1: System.out.println("1"); break;
default: System.out.println("Standard");
                    case 2: System.out.println("2");break;

                }
            }
        }
}
```

a) 1
b) 2
c) Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
    außen:
        for (int j = 1; j < 5; j++){
            switch (j){
                case 1: System.out.println("1"); break;
                default: System.out.println("Standard");
                case 2: System.out.println("2");break außen;

            }
        }
    }
}
```

a) 12
b) 21
c) 12Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
    außen:
        for (int j = 1; j < 5; j++){
```

```
                switch (j){
                    case 1: System.out.println("1"); break;
                    default: System.out.println("Standard");
                    case 2: System.out.println("2");break;

                }
            }
        }
}
```

a) 12
b) 21
c) 12Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class SwitchStruktur {

    public static void main(String[ ] args) {
    außen:
        for (int j = 1; j < 5; j++){
            switch (j){
                case 1: System.out.println("1"); break;
default: System.out.println("Standard");
                case 2: System.out.println("2");continue außen;

            }
        }
    }
}
```

a) 12
b) 21
c) 12Standard
d) 1Standard2
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(11) Frage

Welche der untenstehenden for-Schleifen führt zu einem Kompilierfehler.

a) int i; for(int i = 0; i<100; i) { }
b) int i; for(i= 1; i<100; i++) { }
c) int i = 0; for(; i<100; i++) { }
d) int i; for(i= 1; i<100;) { i++ }

e) for(int i = 0; ; i++) { }
f) for(; ;) { }
g) Keine dieser Möglichkeiten.

(12) Frage

Welche der untenstehenden for-Schleifen führt zu einem Kompilierfehler.

a) for(int i = 0; i<=100; i++) { }
b) int i; for(i= 1; i<100; i++) { }
c) int i = 1; for(; i<100; i++) { }
d) for(int i= 1; i<100;) { i++ }
e) for(int i = 0; ; i++) { break; }
f) for(; ;) { break; }
g) Keine dieser Möglichkeiten.

(13) Frage

Welche der untenstehenden for-Schleifen führt zu einem Kompilierfehler.

a) for(int i = 0; i=100; i++) { }
b) for(int i= 1; i<100; i++) { }
c) int i = 0; for(i<100; i++) { }
d) int i; for(i= 1; i<100) { i++ }
e) for(int i = 0; ; i++) { }
f) for(; ;) { }
g) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class WhileSchleife {
    public static void main(String[ ] args) {
        int i = 0;
        while (i<2){
            System.out.println(i);
            i++;
        }
    }
}
```

a) 01
b) 012
c) Kompilierfehler.
d) Keine dieser Möglichkeiten.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class WhileSchleife {
    public static void main(String[ ] args) {
        int i = 0;
        while (i){
            System.out.println(i);
            i++;
        }
    }
}
```

a) 01
b) 012
c) Kompilierfehler.
d) Keine dieser Möglichkeiten.

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class WhileSchleife {
    public static void main(String[ ] args) {
        int i = 0;
        while (i<0){
            System.out.println(i);
            i++;
        }
    }
}
```

a) 01
b) 012
c) Kompilierfehler.
d) Keine dieser Möglichkeiten.

(17) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class DoWhileSchleife {
    public static void main(String[ ] args) {
        int i = 1;
        do {
            System.out.println(i);
            i++;
        }while (i < 2);
    }
}
```

a) 1

b) 01
c) 012
d) Kompilierfehler.
e) Keine dieser Möglichkeiten.

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class DoWhileSchleife {
	public static void main(String[ ] args) {
		int i = 1;
		do {
			System.out.println(i);
			i++;
		}while (i < 1);
	}
}
```

a) 1
b) 01
c) 012
d) Kompilierfehler.
e) Keine dieser Möglichkeiten.

6. Lösungen

(1) Frage

d

(2) Frage

e, Es müsste j < 2 lauten, j = 2 ist eine Zuweisung und keine Bedingung, deshalb kommt es zu einem Kompilierfehler.

(3) Frage

e, Die Variable j darf nicht vom Datentyp long sein, sondern nur char, byte, short und int.

(4) Frage

a, Die Fälle müssen nicht in einer gewissen Reihenfolge stehen.

(5) Frage

a, Die Variable j darf außerhalb der for-Schleife deklariert und initialisiert werden.

(6) Frage

e, Die Variable j darf zwar außerhalb der for-Schleife deklariert und initialisiert

werden, dann muß aber innerhalb der Klammer vor der Bedingung ein Strichpunkt stehen. Es müsste wie folgt aussehen: for (; j < 2; j++).

(7) Frage

e, Die Variable j darf außerhalb der for-Schleife deklariert und innerhalb initialisiert werden, aber es muß trotzdem in der Klammer der Strichpunkt stehen.

(8) Frage

a

(9) Frage

f

(10) Frage

f

(11) Frage

a, Doppelte Variablendeklaration von i führt zu einem Kompilierfehler.

(12) Frage

g

(13) Frage

a, Es müsste an zweiter Stelle in der Klammer ein Bedingung stehen, es steht aber eine Zuweisung (i=100)
c, Strichpunkt fehlt.
d, Strichpunkt fehlt.

(14) Frage

a

(15) Frage

c, In der Klammer nach while steht (i), dies ist keine Bedingung / Boolean-Ausdruck, deswegen kommt es zu einem Kompilierfehler.

(16) Frage

d, Die Bedingung 1<0 ist bereits zu Beginn false, deswegen wird die while-Schleife übersprungen.

(17) Frage

a

(18) Frage

a

VII. Arrays

1. Eindimensionale Arrays

Planen Sie Milch, Brot und Marmelade zu kaufen und brauchen eine Einkaufsliste? Oder benötigen Sie eine Aufzählung aller Schulnoten? So ist in Java ein Array die erste Wahl. Ein Array macht es möglich mehrere Werte gleichzeitig zu speichern, wie z. B. die Zahlen 1, 2, 3, 4, 5 und 6, die die Zensuren sehr gut bis ungenügend darstellen sollen. Die Werte sind indexiert und es besteht ein sogenannter random access, sprich auf jedes Element kann gleich schnell zugegriffen werden. In Java sind Arrays Objekte und können Werte von unterschiedlichen primitiven Datentypen oder Objekte enthalten.

In untenstehendem Beispiel können Sie erkennen, wie ein int-Array deklariert und mit Standardwerten initialisiert wird (vgl. Initialisierung und Deklaration von Variablen). Hierbei wird auch gleich die Größe des Arrays bestimmt. Das Array i und b besteht aus 6 Elementen, was Ihnen bestimmt seltsam vorkommt, ist es aber nicht, da die Durchnumerierung der Indexpositionen bei 0 und nicht bei 1 beginnt.

```
public class Arrays {
        public static void main(String[ ] args) {
        //Deklaration eines Arrays
        int[ ] i;
        int a[ ];
        //Konstruktion und Initialisierung mit Standardwerten
        i = new int[5];
        //Deklaration und Initialisierung mit Standardwerten in einem Schritt
        int[ ] b = new int[5];
        }
}
```

Vergessen Sie beim Konstruieren die Größe des Arrays, kommt es zu einem Kompilierfehler, wobei allerdings ein Array der Länge 0 erlaubt ist.

```
5 public class Arrays {
6     public static void main(String[] args) {
7         int b[] = new int[];
8     }
9 }
```

Schreiben Sie die Größe des Arrays in die Klammer vor der Zuweisung führt dies ebenfalls zu einem Kompilierfehler. Die Größe des Arrays muß immer nach der Zuweisung angegeben werden.

```
5 public class Arrays {
6     public static void main(String[] args) {
7         int b[5] = new int[5];
8     }
9 }
```

Sie können aber auch ein Array in einem Schritt mit expliziten Werten initialisieren und anschließend mit i [0] auf das erste Element, mit i [1] auf das zweite Element und mit i [2] zugreifen. Bitte beachten Sie, daß sich das erste Element immer an der Position 0 befindet.

Element	1	2	3
Position	0	1	2

```
public class Arrays {
        public static void main(String[ ] args) {
                int[ ] i = {1, 2, 3};
                System.out.println("Dies ist das erste Element des Arrays: " + i[0]);
                System.out.println("Dies ist das zweite Element des Arrays: " + i[1]);
                System.out.println("Dies ist das dritte Element des Arrays: " + i[2]);
        }
}
```

Ausgabe auf der Konsole:

```
Dies ist das erste Element des Arrays: 1
Dies ist das zweite Element des Arrays: 2
Dies ist das dritte Element des Arrays: 3
```

Werden dem Array keine expliziten Werte zugewiesen, besitzt das Array den Standardwert des jeweiligen Datentyps des Arrays. Bei dem primitiven Datentyp int ist dies der Wert 0 und beim Wrapper Integer ist dies null (vgl. Kapitel über die Wrapper-Klassen und über die primitiven Datentypen).

```
public class Arrays {
        public static void main(String[ ] args) {
                int[ ] i = new int [1];
                Integer b[ ] = new Integer[1];
                System.out.println("Dies ist das erste Element des Arrays i: " + i[0]);
                System.out.println("Dies ist das erste Element des Arrays b: " + b[0]);
        }
}
```

Ausgabe auf der Konsole:

Dies ist das erste Element des Arrays i: 0
Dies ist das erste Element des Arrays b: null

Weist ein Array mehrere Elemente auf, können Sie die Elemente mithilfe der for-Schleife und der Methode a.length auslesen. Die Methode a.length gibt die Länge der Elemente einer Ebene des Arrays wieder. Hier ein Beispiel:

```
public class Arrays {
    public static void main(String[ ] args) {
        int[ ] a = {1, 2, 3};
        for (int i=0; i<a.length; i++){
            System.out.println("An der Indexposition " + i + " befindet sich der Wert: " + a[i]);

        }
    }
}
```

Ausgabe auf der Konsole:

An der Indexposition 0 befindet sich der Wert: 1
An der Indexposition 1 befindet sich der Wert: 2
An der Indexposition 2 befindet sich der Wert: 3

Wollen Sie auf die Indexposition 2 zugreifen, die bei int[ ] a = new int[2]; nicht vorhanden ist, kommt es zur Laufzeit zu einer ArrayIndexOutOfBoundsException (siehe Kapitel Exceptions), die eine RuntimeException ist. Dies erkennt der Kompiler erst zur Laufzeit, da dies kein Syntaxfehler, sondern ein Logikfehler ist.

```
4  public class Arrayfehler {
5
6      public static void main(String[] args) {
7          int[] a = new int[2];
8          a[1]= 2;
9          a[2]= 3;
10     }
11 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Arrayfehler [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (1
java.lang.ArrayIndexOutOfBoundsException: 2
        at scjp.Arrayfehler.main(Arrayfehler.java:10)
Exception in thread "main"
```

2. Mehrdimensionale Arrays

Unter Zuhilfenahme eines zweidimensionalen Arrays ist es denkbar ein Matrix abzubilden. Diese könnte wie folgt aussehen:

Indexposition	**0**	**1**
0	1	2
1	3	4

Der erste Schritt ein zweidimensionales Array zu erstellen, ist die Deklaration. Es existieren drei Wege ein Array fehlerfrei zu deklarieren:

```
public class Arrays {
        public static void main(String[ ] args) {
                int [ ][ ] a;
                int [ ]b[ ];
                int c[ ][ ];
        }
}
```

Ebenso gibt es drei Arten, ein zweidimensionales Array zu deklarieren und mit Standardwerten zu initialisieren:

```
public class Arrays {
        public static void main(String[ ] args) {
                int [ ][ ] a = new int[2][2];
                int [ ]b[ ] = new int[2][2];
                int c[ ][ ] = new int[2][2];
        }
}
```

Lenken Sie Ihr Augenmerk auf die rechte Seite, wo mindestens in der ersten Klammer ein Wert stehen muß.

```
public class Arrays {
        public static void main(String[ ] args) {
                int [ ][ ] a = new int[2][ ];
                int [ ]b[ ] = new int[2][ ];
                int c[ ][ ] = new int[2][ ];
        }
}
```

Vergessen Sie auf der rechten Seite in der eckigen Klammer einen Wert einzusetzen, kommt es zu einem Kompilierfehler.

```
5 public class Arrays {
6     public static void main(String[] args) {
7         int [][] a = new int[][];
8         int [] b[] = new int[][];
9         int c[][] = new int[][];
10    }
11 }
```

Zusätzlich zu unserem zweidimensionalen Array, wollen wir jetzt ein dreidimensionales Array deklarieren und mit Standardwerten initialisieren. Wie Sie sehen können, dürfen Sie auf der rechten Seite nie die rechteckige Klammer in der Mitte ohne Wert lassen, da dies zu Problemen führt. Die anderen zwei untenstehende Varianten sind erlaubt.

```
5 public class Arrays {
6     public static void main(String[] args) {
7         int [][][] a = new int[2][][];
8         int [][][] b = new int[2][2][];
9         int [][][] c = new int[2][][2];
10    }
11 }
```

Nun zurück zu unseren zweidimensionalen Arrays, die man in einem Schritt deklarieren und mit expliziten Werten initialisieren kann. Als Basis nehmen wir die obenstehende Matrix. In untenstehendem Beispiel befindet sich die Zahl 1 an der Indexposition a[0][0], die Zahl 2 befindet sich an der Indexposition a[0][1], die Zahl 3 an der Indexposition a[1][0] und die Zahl 4 an der Indexposition a[1][1]. Um einen fehlerfreien Code zu erhalten, muß zwischen den Zahlen und den Indexpositionen ein Komma stehen und kein Semikolon. Und die geschweiften Klammern dürfen nicht durch normale oder eckige ersetzt werden.

```
public class Arrays {
        public static void main(String[ ] args) {
                int [ ][ ] a = {
                        {1, 2},
                        {3, 4}
                };
        }
}
```

Die Werte des zweidimensionalen Arrays können mit einer verschachtelten for-Schleife wieder ausgelesen werden.

```
public class Arrays {
	public static void main(String[ ] args) {
		int [ ][ ] a = {
			{1, 2},
			{3, 4}};

		for(int i = 0; i<a.length; i++){
			for(int j = 0; j<a.length; j++){
				System.out.println("Dies ist der Wert an der
				Indexposition (" + i + ", " + j + "): " + a[i][j]);
			}
		}
	}
}
```

Ausgabe auf der Konsole:

```
Dies ist der Wert an der Indexposition (0, 0): 1
Dies ist der Wert an der Indexposition (0, 1): 2
Dies ist der Wert an der Indexposition (1, 0): 3
Dies ist der Wert an der Indexposition (1, 1): 4
```

Wie Sie sehen können, wird zweimal die Methode a.length benötigt, da a.length nicht alle Array-Elemente zählt, sondern nur die Elemente einer Ebene. In unserem Fall, wie Sie unten sehen können, kommt a.length in der ersten Ebene zu dem Ergebnis 2.

```
public class Arrays {
	public static void main(String[ ] args) {
		int [ ][ ]  a = {
			{1, 2},
			{3, 4}};
		System.out.println(a.length);
	}
}
```

Ausgabe auf der Konsole:

```
2
```

3. Übungen

(1) Frage

Sie haben das Array int [ ] a = new int[2], wie viele Elemente besitzt es?

a) 1
b) 2
c) 3
d) 4
e) 5

(2) Frage

Sie haben das Array int [ ] a = new int[2], wie kann man die Anzahl der Elemente einer Ebene feststellen?

a) a.length()
b) a.length
c) a.getLength
d) a[ ].length
e) a.capacity
f) a.size

(3) Frage

Welche Array-Deklarationen und Array-Initialisierung mit Standardwerten führen nicht zu Kompilierfehlern?

a) int [2] a = new int[2];
b) int [ ] a = new int[ ];
c) int [ ][ ] a = new int [ ][ ]
d) int [ ][ ] a = new int [2][ ]
e) int [ ][ ] a = new int [2][3]
f) int [2][3] a = new int [2][3]

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		int [ ][ ] a = {
				{5, 4},
				{3, 2}};
		for (int i = 0; i<a.length; i++){
			for (int j = 0; j<a.length; j++){
				System.out.println(a[i][j]);
			}
		}
	}
}
```

a) 4, 3, 2
b) 5, 4, 3, 2
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
	public static void main(String[ ] args) {
		int [2][2] a = {
				{5, 4},
				{3, 2}};
		for (int i = 0; i<a.length; i++){
			for (int j = 0; j<a.length; j++){
				System.out.println(a[i][j]);
			}
		}
	}
}
```

a) 4, 3, 2
b) 5, 4, 3, 2
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(6) Frage

Welche Array-Deklarationen und Array-Initialisierung mit Standardwerten führen nicht zu Kompilierfehlern?

a) double [5] a = new double[5];
b) double a [ ]= new double [ ];
c) double [ ] a [ ] = new double [ ][ ]
d) double [ ][ ][ ] a = new double [2][ ][ ]
e) double [2][3][ ] a = new double [2][3][ ]
f) double [ ][ ][ ] a = new double [2][ ][3]

(7) Frage

Welche der untenstehenden Möglichkeiten deklariert und initialisiert ein Array mit fünf Werten, ohne einen Kompilierfehler zu verursachen?

a) int a[ ] = new int[5];
b) int a[5] = new int[5];
c) int a() = new int(5);
d) int a[ ] = {2, 3, 4, 5};

e) int a[ ] = {2, 3, 4, 5, 6};
f) int a[5] = {2, 3, 4, 5, 6};
g) int a [5];

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {

        double [ ][ ] a = {
                {4.0, 8.0},
                {7.0, 10.0}};
        System.out.println(a[1][1]);
    }
}
```

a) 4.0
b) 10.0
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double [ ][ ] a = {
                {4.0, 8.0},
                {7.0, 10.0}};
        System.out.println(a[2][2]);
    }
}
```

a) 4.0
b) 10.0
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [ ][ ] a = {
```

```
                {6, 9},
                {3, 4},
                {2, 7, 5}};
        System.out.println(a[2][2]+ " " + a[1][1]);
    }
}
```

a) 4, 6
b) 5, 4
c) 10
d) 9
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [ ][ ] a = {
                {6, 9},
                {3, 4},
                {2, 7, 5}};
        System.out.println(a[2][2]+ a[1][1]);
    }
}
```

a) 4, 6
b) 5, 4
c) 10
d) 9
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [2][2] a = {
                {6, 9},
                {3, 4},
                {2, 7, 5}};
        System.out.println(a[2][2]+ a[1][1]);

    }
}
```

a) 4, 6
b) 5, 4
c) 10
d) 9
e) Kompilierfehler
f) Laufzeitfehler
g) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [ ][ ] a = {
                {15, 14, 12},
                {53, 46, 91},
                {96, 72, 12},
                {45, 34, 23}};
        System.out.println(a[3][2]+ ” ” + a[2][2]);
    }
}
```

a) 23 12
b) 72 91
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [ ][ ] a = {
                {15, 14, 12},
                {53, 46, 91},
                {96, 72, 12},
                {45, 34, 23}};
        System.out.println(a[3][3]+ ” ” + a[2][2]);
    }
}
```

a) 23 12
b) 72 91
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int [ ][ ] a = {
                {15, 14, 12},
                {53, 46, 91},
                {96, 72, 12},
                {45, 34, 23}};
        System.out.println(a[2][2]+ " " + a[1][2]);
    }
}
```

a) 91 12
b) 12 91
c) Kompilierfehler
d) Laufzeitfehler
e) Keine dieser Möglichkeiten.

4. Lösungen

(1) Frage

b

(2) Frage

b

(3) Frage

b, d, e

(4) Frage

b

(5) Frage

c, Beim Deklarieren und Intitalisieren, darf auf der linken Seite in den eckigen Klammern nichts stehen. Es müsste statt int [2][2] a, int [ ][ ] a stehen.

(6) Frage

d

(7) Frage

a, e

(8) Frage

b

(9) Frage

d, Es kommt zur Laufzeit zu einer ArrayIndexOutOfBoundsException, da das Array keine Position a [2][2] besitzt, sondern maximal eine Position a [1][1], da die Indexposition bei 0 anfängt zu zählen und nicht bei 1.

(10) Frage

b

(11) Frage

d, Hier werden die zwei Ergebnisse 4 und 5 addiert!

(12) Frage

e, Beim Deklarieren und Intitalisieren, darf auf der linken Seite in den eckigen Klammern nichts stehen. Es müsste statt int [2][2] a, int [ ][ ] a stehen.

(13) Frage

a

(14) Frage

d, Es kommt zur Laufzeit zu einer ArrayIndexOutOfBoundsException, da das Array keine Position a [3][3] besitzt, sondern maximal eine Position a [3][2], da die Indexposition bei 0 anfängt zu zählen und nicht bei 1.

(15) Frage

b

VIII. Wichtige Standardklassen (Fundamental Classes)

A. Die Klasse Object

Die Klasse Object aus dem Package java.lang ist die Mutter aller Klassen. Alle Objekte in Java sind automatisch Subklassen der Klasse Object. Wir haben also bereits die ganze Zeit Objekte erstellt, die automatisch eine Kindklasse der Klasse java.lang.Object waren (vgl. Kapitel Vererbung). Aber was ist mit Klassen, die bereits eine extends-Beziehung haben, und jede Klasse jeweils nur eine extends-Beziehung haben kann? Diese wiederum haben eine Superklasse, die implizit eine Subklasse der Klasse java.lang.Object ist. So schließt sich der Kreis wieder. Alle Objekte, die keine explizite extends-Beziehung besitzen, haben eine implizite extends-Beziehung zu der Klasse Object.

Wichtige Methoden dieser Klasse sind die Methoden int hashCode() und boolean equals(Object obj), die weiter unten näher erläutert werden.

B. Die Wrapper-Klassen (Hüllklassen)

Das englische Verb wrap bedeutet im deutschen einhüllen und einpacken. Die Wrapper-Klassen nehmen also eine Zahl und packen sie in eine Schachtel. Diese Verpackung eröffnet einer Zahl eine größere Anzahl an Möglichkeiten. Bisher haben wir für Zahlen und Text nur primitiven Datentypen kennengelernt und primitive Datentypen sind keine Objekte. Nun wollen wir aber die Vorteile von Objekten nutzen, indem wir primitive Datentypen in Objekte umwandeln. Dies macht man mit Hilfe von Wrapper-Klassen, die auch Hüllklassen genannt werden. Es wird ein primitive Zahl „gewrappt“, sprich eingehüllt.

Wrapper-Klassen sind die folgenden: Void, Boolean, Character, Byte, Short, Integer Long, Float und Double, wobei Byte, Short, Integer, Long, Float und Double direkte Subklassen der Klasse Number verkörpern. Die Klassen Void, Boolean und Character sind keine Subklassen der Klasse java.lang.Number. Alle Wrapper-Klassen sind final und serialisierbar. Die Eigenschaft final hat zur Folge, daß alle Objekte von Wrapper-Klassen unveränderbar sind.

1. Umwandeln von primitiven Datentypen in Objekte der Wrapper-Klassen

Lassen Sie uns in unserem ersten Beispiel mit der Wrapperklasse Integer beginnen. Wollen Sie den primitiven Datentyp int in ein Objekt überführen, geschieht dies mit einem Konstruktor der Klasse Integer. Alle Wrapperklassen, außer den Wrapper-Klassen Character und Void, haben einen Konstruktor mit ihrem eigenen primitiven Datentyp (nicht für ihren Wrappertyp) und einem String. Im Gegensatz hierzu hat die Klasse Character nur einen Konstruktor mit ihrem eigenen primitiven Typ und keinen Konstruktor mit einem String.

```
public class Wrappers {
        public static void main(String[ ] args) {
                int i = 1;
                String s = “1“;
                Integer wrapperi = new Integer(i);
                Integer wrappers = new Integer(s);
                System.out.println(wrapperi);
                System.out.println(wrappers);
        }
}
```

Ausgabe auf der Konsole:

```
1
1
```

Beinhaltet String s allerdings einen Wert, der sich nicht in einen int-Wert umwandeln läßt, kommt es zu einer NumberFormatException (vgl. Kapitel Exceptions). Dies wird erst zur Laufzeit bemerkt, da es sich nicht um einen Syntax-, sondern um einen Logikfehler, handelt. Ein Syntaxfehler würde bereits zur Kompilierzeit bemerkt und rot unterkringelt werden. Das gleiche gilt für den Fall, daß der String direkt in den Konstruktor eingegeben wird (new Integer(“gut“)):

```
5  public class Wrappers {
6
7      public static void main(String[] args) {
8          int i = 1;
9          String s = "gut";
10         Integer wrapperi = new Integer(i);
11         Integer wrappers = new Integer(s);
12         System.out.println(wrapperi);
13         System.out.println(wrappers);
14
15     }
16 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Wrappers [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (30.05.2005 13:08:36)
java.lang.NumberFormatException: For input string: "gut"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.<init>(Unknown Source)
        at scjp.Wrappers.main(Wrappers.java:11)
Exception in thread "main"
```

Werden als Parameter andere primitive Datentypen übergeben, werden diese automatisch in ein int überführt, sofern sie vom Typ her kleiner sind, also vom Typ byte, short oder char. Die gleichen Regeln gelten für andere Darstellungsweisen von primitiven Datentypen, wie z. B. die Hexadezimal-Schreibweise.

```
public class Wrappers {
        public static void main(String[ ] args) {
                int i = 1;
                char c = ‘2‘;
                short s = 3;
                byte b = 5;
                Integer wrapperi = new Integer(i);
                Integer wrapperc = new Integer(c);
                Integer wrappers = new Integer(s);
                Integer wrapperb = new Integer(b);
                System.out.println(wrapperi);
                System.out.println(wrapperc);
                System.out.println(wrappers);
                System.out.println(wrapperb);
        }
}
```

Ausgabe auf der Konsole:

```
1
50
3
5
```

Wird allerdings versucht eine Variable von einem größeren primitiven Datentyp als int in den Konstruktor einzufügen, kommt es zu einem Kompilierfehler. Dies ist z. B. in untenstehendem Beispiel der Fall: Es wird dem Konstruktor ein Parameter vom Typ long übergeben. Dies funktioniert bereits zur Kompilierzeit nicht und zur Laufzeit wird ein Error geworfen (siehe Kapitel Exceptions).

```
5  public class Wrappers {
6      public static void main(String[] args) {
7          int i = 1;
8          char c = '2';
9          short s = 3;
10         long l = 4;
11         Integer wrapperi = new Integer(i);
12         Integer wrapperc = new Integer(s);
13         Integer wrappers = new Integer(s);
14         Integer wrapperl = new Integer(l);
15         System.out.println(wrapperi);
16         System.out.println(wrapperc);
17         System.out.println(wrappers);
18         System.out.println(wrapperl);
19     }
20 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Wrappers [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.e
java.lang.Error: Unresolved compilation problem:
        The constructor Integer(long) is undefined

        at scjp.Wrappers.main(Wrappers.java:14)
Exception in thread "main"
```

In untenstehendem Fall kommt es auch zu einem Kompilierfehler und zur Laufzeit zu einem Error, da der Parameter, die Zahl 1, standardmäßig vom Typ int und somit vom Typ her zu groß ist. Die Zahl 1 wird nicht automatisch in den primitiven Datentyp byte überführt, wenn Sie sie als Parameter dem Konstruktor der Byte-Klasse übergeben. Würde dort allerdings ein String stehen (Byte b = new Byte("1");), würde dies zu keinem Fehler führen, da der String intern automatisch umgewandelt wird. Das gleiche gilt für den Konstruktor des Wrappers Short.

```
4
5 public class Wrappers {
6     public static void main(String[] args) {
7         Byte b = new Byte(1);
8     }
9 }
10
```

Ist der primitive Datentyp zu groß für den Konstruktor der Wrapperklasse, kann dieses Problem durch einen Cast gelöst werden (vgl. Kapitel Implizite und explizite Typanpassung). Wird die Zahl 1 vorher zu einem primitiven byte gecastet, läuft das Programm fehlerfrei.

```
public class Wrappers {
        public static void main(String[ ] args) {
                Byte b = new Byte((byte)1);
        }
}
```

2. Besonderheiten bei der Wrapper-Klasse Boolean

In dem Kapitel Literale haben wir gelernt, daß der primitive Datentyp boolean nur die Werte true oder false annehmen kann und diese beiden Begriffe komplett aus Kleinbuchstaben bestehen müssen. Bei der Umwandlung eines primitiven boolean-Ausdrucks in ein Objekt, können in den Konstruktor des Wrappers diese beiden Werte eingefügt werden. Es gibt allerdings in der Klasse Boolean auch einen Konstruktor mit einem Parameter vom Typ String, in den auch true oder false als String eingegeben werden kann. Hierbei muß nicht auf Groß- oder Kleinschreibung geachtet werden. Als Parameter kann auch der Defaultwert der Stringobjekte, nämlich null oder irgendein Text verwendet werden, wobei dann false als Ergebnis zurückgegeben wird.

```
public class WrapperBoolean {
        public static void main(String[ ] args) {
                Boolean b1 = new Boolean(false);
                Boolean b2 = new Boolean(true);
                Boolean b3 = new Boolean("FALSE");
```

```
                    Boolean b4 = new Boolean(”Hallo!“);
                    Boolean b5 = new Boolean(null);
                    System.out.println(b1);
                    System.out.println(b2);
                    System.out.println(b3);
                    System.out.println(b4);
                    System.out.println(b5);
          }
}
```

Ausgabe auf der Konsole:

```
false
true
false
false
false
```

Zu Kompilierfehlern kommt es allerdings in den Fällen, in denen true oder false groß geschrieben und nicht als String eingegeben wird. Außerdem darf kein Wrapperobjekt ein Parameter sein.

```
public class WrapperBoolean {
    public static void main(String[ ] args) {
        Boolean b1 = new Boolean(false);
        Boolean b2 = new Boolean(TRUE); //Kompilierfehler und zur Laufzeit Error
        Boolean b3 = new Boolean(FALSE); //Kompilierfehler und zur Laufzeit Error
        Boolean b4 = new Boolean(b1); //Kompilierfehler und zur Laufzeit Error
    }
}
```

3. Umwandeln eines Strings in einen Wrapper mit Hilfe von valueOf(String s)

Aus einem String kann man ein Wrapperobjekt machen, indem man die Methode valueOf(String s) verwendet. Diese Methode gibt es in fast allen Wrapperklassen, außer in der Klasse Character. Verwendet man die Methode valueOf() auf einem Character-Objekt, kommt es, wie Sie unten sehen können, zu einem Kompilierfehler.

```
4  public class ValueOf {
5
6      public static void main(String[] args) {
7          String s = "1";
8          Integer i = Integer.valueOf(s);
9          Byte b = Byte.valueOf(s);
10         Short sh = Short.valueOf(s);
11         Long l = Long.valueOf(s);
12         Float f = Float.valueOf(s);
13         Double d = Double.valueOf(s);
14         Boolean bo = Boolean.valueOf(s);
15         Character c = Character.valueOf(s);
```

Zu Problemen kommt es auch, wenn der Parameter nicht vom Typ String oder leer ist. Sollte der Rückgabetyp der Methode valueOf(String) ein primitiver Wert sein, führt dies ebenfalls zu Komplikationen.

```
4  public class ValueOf {
5
6      public static void main(String[] args) {
7          String s = "1";
8          /*Kompilierfehler, da der Parameter vom
9           Typ String sein muß*/
10         Integer i = Integer.valueOf(1);
11         Byte b = Byte.valueOf("1");
12
13         /*Kompilierfehler, da der Rückgabetyp ein
14         Wrapper und keine primitiver Typ ist*/
15         short sh = Short.valueOf(s);
16     }
17 }
```

Würde dort allerdings String = “gut“ stehen, würde es nicht zu einem Kompilierfehler kommen, sondern zu einem Laufzeitfehler und einer NumberFormatException, da der Kompiler erst zur Laufzeit bemerkt, daß sich der String nicht in einen passenden primitiven Datentyp überführen läßt (vgl. Kapitel Exceptions).

4. Umwandeln eines Wrappers in einen primitiven Datentyp: xxxValue()

In der Klasse java.lang.Number gibt es folgende Methoden, die es Ihnen ermöglichen einen Wrapper in einen primitiven Datentyp umzuwandeln: byteValue(), doubleValue(), floatValue(), intValue(), longValue() und shortValue(). Diese Methoden werden an alle Subklassen der Klasse Number, wie Byte, Short, Integer, Long, Float und Double vererbt, somit stehen sie in all diesen Klassen zur Verfügung. Besondere Aufmerksamkeit verdient die Tatsache, daß in diesen Klassen weder eine Methode charValue() noch booleanValue() existiert, obwohl man es vermuten könnte. In der Klasse Character ist nur die Methode charValue() und in der Klasse Boolean nur die Methode booleanValue() vorhanden. Wollen Sie allerdings charValue() auf i einem Integer (Wrapper) ausführen, eine Methode, die es in der Klasse Integer nicht gibt, kommt es zu einem Kompilierfehler. Des weiteren kommt es zu Problemen, wenn ein Parameter zu der Methode (i.charValue(2)) hinzugefügt wird.

```
public class XxxValue {
        public static void main(String[ ] args){
                Integer i = new Integer(2);
                byte b = i.byteValue();
```

```
                    short s = i.shortValue();
                    double d = i.doubleValue();
                    float f = i.floatValue();
                    long l = i.longValue();
                    char c = i.charValue(); //Kompilierfehler
                    boolean bo = i.booleanValue(); //Kompilierfehler
          }
}
```

5. Umwandeln eines Strings in einen primitiven Datentyp: parseXXX(String s)

Ein Stringobjekt kann man mit der Methode parseInt(String s) der Klasse Integer in einen primitiven Datentyp überführen. Diese Methode wandelt ein Stringobjekt in ein int um. In den Klassen Byte, Double, Float, Long und Short sind entsprechende Methoden vorhanden, wohingegen in den Klassen Boolean und Char keine solche Methode implementiert ist. Es gibt in den Klassen Boolean oder Char keine Methoden parseBoolean(String) oder parseChar(String), obwohl man dies vermuten könnte. Somit hat die Anwendung dieser Methoden einen Kompilierfehler zur Folge.

```
public class parseXXX {
          public static void main(String[ ] args){
                    String s = ”1“;
                    int i = Integer.parseInt(s);
                    byte b = Byte.parseByte(s);
                    long l = Long.parseLong(s);
                    double d = Double.parseDouble(s);
                    float f = Float.parseFloat(s);
                    short sh = Short.parseShort(s);
          }
}
```

Es kommt aber auch zu einem Kompilierfehler, wenn der Parameter fehlt (z. B. Integer.parseInt()) oder kein String als Parameter übergeben wird.

6. Übungen

(1) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
    public static void main(String[ ] args) {
        Boolean b1 = new Boolean(false);        Zeile 1
        Boolean b2 = new Boolean(true);         Zeile 2
        Boolean b3 = new Boolean(FALSE);        Zeile 3
        Boolean b4 = new Boolean(TRUE);         Zeile 4
        Boolean b5 = new Boolean(b2);           Zeile 5
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
        System.out.println(b5);
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(2) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
    public static void main(String[ ] args) {
        Boolean b1 = new Boolean(false);        Zeile 1
        Boolean b2 = new Boolean(true);         Zeile 2
        Boolean b3 = new Boolean(null);         Zeile 3
        Boolean b4 = new Boolean(”Ich“);        Zeile 4
        Boolean b5 = new Boolean(”True“);       Zeile 5
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
        System.out.println(b5);
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(3) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
    public static void main(String[ ] args) {
        Boolean b1 = new Boolean(false);          Zeile 1
        Boolean b2 = new Boolean(true);           Zeile 2
        Boolean b3 = new Boolean(”FALSE“);        Zeile 3
        Boolean b4 = new Boolean(”TRuE“);         Zeile 4
        Boolean b5 = new Boolean(”fALSE“);        Zeile 5
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
        System.out.println(b5);
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(4) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
    public static void main(String[ ] args) {
        Boolean b1 = new Boolean(null);           Zeile 1
        Boolean b2 = new Boolean(true);           Zeile 2
        Boolean b3 = new Boolean(Ich);            Zeile 3
        Boolean b4 = new Boolean(TRUE);           Zeile 4
        Boolean b5 = new Boolean(false);          Zeile 5
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
        System.out.println(b5);}}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(5) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
        public static void main(String[ ] args) {
        int i = 1;
        char c = ‘2‘;
        short s = 3;
        long l = 4;
        byte b = 5;
        Integer wi = new Integer(i);                    Zeile 1
        Integer wc = new Integer(s);                    Zeile 2
        Integer ws = new Integer(s);                    Zeile 3
        Integer wl = new Integer(l);                    Zeile 4
        Byte wb = new Byte(i);                          Zeile 5
        Short sh = new Short(s);                        Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(6) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
        public static void main(String[ ] args) {
        String s = ”3“;
        Double wd = new Double(s);                      Zeile 1
        Float wf = new Float(s);                        Zeile 2
        Integer ws = new Integer(s);                    Zeile 3
        Character wc = new Character(s);                Zeile 4
        Byte wb = new Byte(s);                          Zeile 5
        Short sh = new Short(s);                        Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(7) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
        public static void main(String[ ] args) {
        short s = 3;
        char c = '2';
        Double wd = new Double(s);                    Zeile 1
        Float wf = new Float(s);                      Zeile 2
        Integer ws = new Integer(s);                  Zeile 3
        Character wc = new Character(c);              Zeile 4
        Byte wb = new Byte(s);                        Zeile 5
        Short sh = new Short(s);                      Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(8) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
        public static void main(String[ ] args) {
        short s = 3;
        char c = '2';
        Double wd = new Double(s);                    Zeile 1
        Float wf = new Float(s);                      Zeile 2
        Integer ws = new Integer(s);                  Zeile 3
        Character wc = new Character(c);              Zeile 4
        Byte wb = new Byte((byte)s);                  Zeile 5
        Short sh = new Short(s);                      Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(9) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class Wrappers {
        public static void main(String[ ] args) {
        Double wd = new Double(1);                    Zeile 1
```

```
        Float wf = new Float(1);                    Zeile 2
        Integer ws = new Integer(1);                Zeile 3
        Character wc = new Character(1);            Zeile 4
        Byte wb = new Byte(1);                      Zeile 5
        Short sh = new Short(1);                    Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(10) <u>Frage</u>

In welcher der untenstehenden Zeilen kommt es zu einer NumberFormatException?

```
public class Wrappers {
        public static void main(String[ ] args) {
        String s = ”gut“;
        char c = ‘2‘;
        byte b = 2;
        Double wd = new Double(b);                  Zeile 1
        Float wf = new Float(s);                    Zeile 2
        Integer ws = new Integer(b);                Zeile 3
        Character wc = new Character(c);            Zeile 4
        Byte wb = new Byte(b);                      Zeile 5
        Short sh = new Short(b);                    Zeile 6
}
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(11) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”1“;
        Integer i = Integer.valueOf(s);
        Byte b = Byte.valueOf(”1“);
```

```
            Short sh = Short.valueOf(s);
            Float f = Float.valueOf(s);
        }
}
```

a) Kompilierfehler
b) Laufzeitfehler (NumberFormatException)
c) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”1“;
        Integer i = Integer.valueOf(s);
        Byte b = Byte.valueOf(1);
        Short sh = Short.valueOf(s);
        Float f = Float.valueOf(s);
    }
}
```

a) Kompilierfehler
b) Laufzeitfehler (NumberFormatException)
c) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”eins“;
        Integer i = Integer.valueOf(s);
        Byte b = Byte.valueOf(”1“);
        Short sh = Short.valueOf(s);
        Float f = Float.valueOf(s);
    }
}
```

a) Kompilierfehler
b) Laufzeitfehler (NumberFormatException)
c) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
```

```
    public static void main(String[ ] args) {
        String s = ”1“;
        Integer i = Integer.valueOf(s);
        byte b = Byte.valueOf(”1“);
        Short sh = Short.valueOf(s);
        Float f = Float.valueOf(s);
    }
}
```

a) Kompilierfehler
b) Laufzeitfehler (NumberFormatException)
c) Keine dieser Möglichkeiten.

(15) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        Integer i = new Integer(2);
        byte b = i.byteValue();                    Zeile 1
        short s = i.shortValue();                  Zeile 2
        double d = i.doubleValue();                Zeile 3
        float f = i.floatValue();                  Zeile 4
        long l = i.longValue();                    Zeile 5
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(16) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        Integer i = new Integer(2);
        byte b = i.byteValue();                    Zeile 1
        short s = i.shortValue();                  Zeile 2
        char c = i.charValue();                    Zeile 3
        float f = i.floatValue();                  Zeile 4
        long l = i.longValue();                    Zeile 5
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3

d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(17) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        Integer i = new Integer(2);
        byte b = i.byteValue();                    Zeile 1
        short s = i.shortValue();                  Zeile 2
        boolean bo = i.booleanValue();             Zeile 3
        float f = i.floatValue();                  Zeile 4
        long l = i.longValue();                    Zeile 5
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Keine dieser Möglichkeiten.

(18) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        String s = ”1“;
        int i = Integer.parseInt(s);               Zeile 1
        byte b = Byte.parseByte(s);                Zeile 2
        long l = Long.parseLong(1);                Zeile 3
        double d = Double.parseDouble(s);          Zeile 4
        float f = Float.parseFloat(s);             Zeile 5
        short sh = Short.parseShort(s);            Zeile 6
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(19) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        String s = ”1“;
        int i = Integer.parseInt(s);                    Zeile 1
        byte b = Byte.parseByte(s);                     Zeile 2
        long l = Long.parseLong(s);                     Zeile 3
        double d = Double.parseDouble(s);               Zeile 4
        float f = Float.parseFloat(s);                  Zeile 5
        short sh = Short.parseShort(s);                 Zeile 6
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(20) Frage

In welcher der untenstehenden Zeilen kommt es zu einem Kompilierfehler?

```
public class A {
    public static void main(String[ ] args){
        String s = ”1“;
        int i = Integer.parseInt(s);                    Zeile 1
        byte b = Byte.parseByte(s);                     Zeile 2
        long l = Long.parseLong(s);                     Zeile 3
        double d = Double.parseDouble(s);               Zeile 4
        float f = Float.parseFloat(s);                  Zeile 5
        boolean bo = Boolean.parseBoolean(s);           Zeile 6
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(21) Frage

Welche der untenstehenden Aussagen trifft zu?

a) Alle Wrapperklassen sind final und serialisierbar.
b) Die Klasse Boolean ist eine direkte Subklasse von java.lang.Number.
c) Primitive Datentypen sind Objekte.

d) Die Klasse Character hat einen Konstruktor mit ihrem eigenen primitiven Datentyp und einem String.
e) Alle Wrapperklassen sind Subklassen von java.lang.Number.
f) Der Methode valueOf wird keinen Parameter übergeben.

(22) Frage

Welche der untenstehenden Klassen sind Subklassen von java.lang.Number?

a) Object
b) Character
c) Boolean
d) Void
e) Byte
f) String
g) Keine dieser Möglichkeiten.

(23) Frage

In welcher Klasse gibt es die charValue()-Methode?

a) Boolean
b) Byte
c) Integer
d) String
e) Character
f) Long
g) Short
h) Double

(24) Frage

In welcher Klasse gibt es die byteValue()-Methode?

a) Boolean
b) Byte
c) Integer
d) String
e) Character
f) Long
g) Short
h) Double

(25) Frage

In welcher Klasse gibt es die booleanValue()-Methode?

a) Boolean
b) Byte
c) Integer
d) String

e) Character
f) Long
g) Short
h) Double

7. Lösungen

(1) Frage

c, d, e

(2) Frage

f

(3) Frage

f

(4) Frage

c, d

(5) Frage

d, e

(6) Frage

d, Der Wrapper Character hat keinen Konstruktor mit einem String, sondern nur einen mit einem primitiven char.

(7) Frage

e, Der primitive Datentyp short wird nicht automatisch in den primitiven Datentyp byte umgewandelt, dies würde einen expliziten Cast erfordern.

(8) Frage

g

(9) Frage

d, e, f, Die Zahl 1 ist vom Typ int und wird nicht automatisch in den entsprechenden primitiven Datentyp umgewandelt, dies würde einen expliziten Cast erfordern.

(10) Frage

b, String s = “gut“ kann man nicht in einen Zahlenwert umwandeln, deswegen kommt es zu einer NumberFormatException.

(11) Frage

c

(12) Frage

a, In die Funktion valueOf(s) kann nur ein String als Parameter eingefügt werden und kein anderer Datentyp wie z. B. int (Byte.valueOf(1);).

(13) Frage

b, String s = “eins“ kann man nicht in einen Zahlenwert umwandeln, deswegen kommt es zu einer NumberFormatException.

(14) Frage

a, Auf der linken Seite muß immer der WrapperTyp stehen, es darf nicht der primitive Typ stehen (byte b = Byte.valueOf(“1“);), da die Funktion valueOf(String s) einen Wrapper zurückgibt und keinen primitiven Datentyp.

(15) Frage

f

(16) Frage

c, Es gibt keine Methode charValue() in der Klasse Integer.

(17) Frage

c, Es gibt keine Methode booleanValue() in der Klasse Integer.

(18) Frage

c, In die Methode parseInt(String s) muß ein String als Parameter übergeben werden, es darf kein anderer Datentyp drin stehen, wie z. B. int (long l = Long. parseLong(1);).

(19) Frage

g

(20) Frage

f, In der Klasse Boolean gibt es keine solche Methode.

(21) Frage

a

(22) Frage

e

(23) Frage

e

(24) Frage

b, c, f, g, h

(25) Frage

a

C. Die Klasse Math

Die Klasse Math definiert einige wichtige statische Methoden: Rundungsmethoden und eine Methode zur Wurzelberechnung, um nur einige zu nennen. So erhalten Sie z. B. Antwort auf die Frage: Wie rundet man die Kommazahl 0,97 zu der Ganzzahl 1 auf?

1. Die Minimum- und die Maximum-Methoden

Welche Zahl ist kleiner 4 oder 5? Eine Lösung für dieses Problem liefert die min()-Methode der Klasse java.lang.Math. Die Minimum-Methode gibt den kleinsten von zwei Werten zurück, wohingegen die Maximum-Methode, den größten von zwei Werten zurückgibt.

Es gibt folgende überladene Minimum-Methoden (vgl. Kapitel Überladen von Methoden):

static int min(int a, int b)
static long min(long a, long b)
static float min(float a, float b)
static double min(double a, double b)

Und es gibt folgende überladene Maximum-Methoden:

static int max(int a, int b)
static long max(long a, long b)
static float max(float a, float b)
static double max(double a, double b)

Ist die Ziffer 2 oder 4 größer? In der untenstehenden Klasse MathClass wird das Ergebnis zu dieser und anderen Fragen berechnet. Wobei ganze Zahlen immer als int angesehen werden, es sei denn, es steht ein l dahinter, dann ist es eine Zahl vom Typ long. Fließkommazahlen sind automatisch vom Typ double, außer sie werden explizit mit einem f für float gekennzeichnet (vgl. Kapitel zum Thema primitiver Datentyp und Typumwandlung).

```
public class MathClass {
	public static void main(String[ ] args) {
		System.out.println(Math.min(2,4));
		System.out.println(Math.min(2l,4l));
		System.out.println(Math.min(1.1f, 1.2f));
		System.out.println(Math.min(1.1d, 1.2d));
		System.out.println(Math.max(2,4));
		System.out.println(Math.max(2l,4l));
		System.out.println(Math.max(1.1f, 1.2f));
		System.out.println(Math.max(1.1d, 1.2d));
	}
}
```

Ausgabe auf der Konsole:

2
2
1.1
1.1
4
4
1.2
1.2

Wird ein Parameter vom Typ short, byte oder char eingegeben, wird dieser Parameter automatisch in den Typ int umgewandelt. Bei Parameter unterschiedlichen Typs wird der kleinere Typ in den größeren Typ überführt. Sind also die Parameter vom Typ int (1) und float (1.2f), erhält man als Rückgabetyp float(1.0), sprich das Ergebnis auf der Konsole wird mit Nachkommastellen ausgegeben.

```
public class MathClass {
        public static void main(String[ ] args) {
                //linker Parameter int, rechter Parameter float, Rückgabetyp
                //float (Ergebnis 1.0)
                System.out.println(Math.min(1, 1.2f));
                //linker Parameter int, rechter Parameter double,
                //Rückgabetyp double (Ergebnis 1.0)
                System.out.println(Math.min(1, 1.2d));

        }
}
```

Ausgabe auf der Konsole:

1.0
1.0

Sie sehen unten: Die min()-Methode, in die zwei Parameter vom primitiven Datentyp byte, short oder char eingegeben werden, hat den Rückgabetyp int, da die Parameter automatisch in den primitiven Datentyp int umgewandelt werden. So gibt die min()-Methode den primitiven Datentyp int zurück, auch wenn die übergebenen Parameter vom Typ byte sind.

```
5  public class MathClass {
6      public static void main(String[] args) {
7          int i = Math.min((byte)2,(byte)4);
8          byte b = Math.min((byte)2,(byte)4);
9      }
10 }
```

Werden mehr als zwei Parameter übergeben, kommt es auch zu einem Kompilierfehler.

2. Die abs-Methode

Die Funktion abs() liefert den absoluten Wert einer Zahl: Sie geben eine negative Zahl ein und erhalten eine Positive.

In der API (Javadocs) finden wir folgende überladenen abs()-Methoden:

```
static int abs(int i)
static long abs(long l)
static float abs(float f)
static double abs(double d)
```

```
public class MathClass {
	public static void main(String[ ] args) {
		System.out.println(Math.abs(-4));
		System.out.println(Math.abs(-4l));
		System.out.println(Math.abs(-0.6f));
		System.out.println(Math.abs(-0.6d));
	}
}
```

Ausgabe auf der Konsole:

```
4
4
0,6
0.6
```

3. Auf- und Abrunden

Nehmen wir z. B. die Aufgabenstellungen: Runden Sie 1,5 auf! Oder: Runden Sie 1,5 ab! So geben uns mehrere Rundungsmethoden Lösungen: Die Methoden static double ceil(double d) und static double floor(double d), wobei erstere immer aufrundet und letztere immer abrundet. Die Methoden ceil und floor haben immer den Rückgabetyp double, sprich eine Fließkommazahl bleibt eine Fließkommazahl.

Die Methode round mit den überladenen Versionen static int round(float f) und static long round(double d) addiert zuerst 0.5 zu der ursprünglichen Zahl und führt dann die Funktion floor durch. Die Methode round hat für den Parameter float den Rückgabetyp int und für den Parameter double den Rückgabetyp long. Hier wird jeweils aus einer Fließkommazahl eine Ganzzahl gemacht.

```
public class MathClass {
	public static void main(String[ ] args) {
		int m1	= Math.round(2.0f);
```

```
                long m2     = Math.round(2.0d);
                double m3 = Math.ceil(2.0f);
                double m4 = Math.floor(2.0d);

                System.out.println(m1);
                System.out.println(m2);
                System.out.println(m3);
                System.out.println(m4);
        }
}
```

Ausgabe auf der Konsole:

```
2
2
2.0
2.0
```

Bei den Rundungsmethoden kommt es allerdings in den Fällen zu Problemen, in denen der Rückgabetyp von der Größe her kleiner ist als der Rückgabetyp der Methode. So ist der Rückgabetyp für die Methode round(double d) long, somit ist der Rückgabetyp int zu klein und führt zu Komplikationen. Der Rückgabetyp von round(float f) ist int, es kann aber als Rückgabetyp ohne weiteres long stehen, da dieser Datentyp größer ist und eine automatische Anpassung erfolgt. Der Rückgabetyp von ceil(double d) ist double, so führt der Rückgabetyp float zu einem Kompilierfehler.

```
 5 public class MathClass {
 6     public static void main(String[] args) {
 7         long l = Math.round(2.2d);
 8         int i = Math.round(2.2d);
 9         long l1 = Math.round(2.2f);
10         double d = Math.ceil(2.2d);
11         float f = Math.ceil(2.2d);
12     }
13 }
```

4. Die Methode random()

Die Methode random() gibt eine Zufallszahl vom Typ double zurück, die größer oder gleich 0.0 oder kleiner 1.0 ist. Wird die random()-Methode mehrmals hintereinander durchgeführt, erhält man jedes Mal eine andere Zahl als Ergebnis.

```
public class MathClass {
        public static void main(String[ ] args) {
                System.out.println(Math.random());
        }
}
```

Ausgabe auf der Konsole:

0.5155740397844537

5. Die Methode sqrt(double d)

Was ist die Wurzel von 4? Die Methode static double sqrt(double d) berechnet Ihnen das Ergebnis! Es ist die Zahl 2. Wird dieser Methode eine negative Zahl oder NaN (Not a Number) übergeben, kommt es zu keiner Fehlermeldung, sondern NaN ist das Resultat .

```
public class MathClass {
	public static void main(String[ ] args) {
		System.out.println(Math.sqrt(4));
		System.out.println(Math.sqrt(5));
	}
}
```

Ausgabe auf der Konsole:

2
2.23606797749979

6. Die Methoden sin(double d), cos(double d), tan(double d)

Es gibt noch folgende Methoden, die den Cosinus, Sinus und den Tangens berechnen.

static double sin(double d)
static double cos(double d)
static double tan(double d)

```
public class MathClass {
	public static void main(String[ ] args) {
		System.out.println(Math.sin(0.0));
		System.out.println(Math.cos(0.0));
		System.out.println(Math.tan(0.0));
	}
}
```

Ausgabe auf der Konsole:

0.0
1.0
0.0

7. Übungen

(1) Frage

Welches ist der Rückgabetyp der Methode max(int a, int b)?

a) short
b) byte
c) int
d) long
e) float
f) double

(2) Frage

Welches ist der Rückgabetyp der Methode ceil(double d)?

a) short
b) byte
c) int
d) long
e) float
f) double

(3) Frage

Welches ist der Rückgabetyp der Methode round(double d)?

a) short
b) byte
c) int
d) long
e) float
f) double

(4) Frage

Welches ist der Rückgabetyp der Methode round(float f)?

a) short
b) byte
c) int
d) long
e) float
f) double

(5) Frage

Welches ist der Rückgabetyp der Methode random()?

a) short
b) byte

c) int
d) long
e) float
f) double

(6) Frage

Welches ist der Rückgabetyp der Methode abs(int a)?

a) short
b) byte
c) int
d) long
e) float
f) double

(7) Frage

Die Methode random() gibt Zahlen aus welchem Zahlenbereich zurück?

a) Zahlen vom Typ double, die größer 0.0 oder kleiner 1.0 sind.
b) Zahlen vom Typ int, die größer oder gleich 0.0 oder kleiner 1.0 sind.
c) Zahlen vom Typ double, die größer oder gleich 0.0 oder kleiner 1.0 sind.
d) Zahlen vom Typ float, die größer oder gleich 0.0 oder kleiner 1.0 sind.
e) Zahlen vom Typ double, die größer oder gleich 0.0 oder kleiner gleich 1.0 sind.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(2.2d);
        int b = Math.round(2.2f);
        double c = Math.floor(2.2d);
        System.out.println(a + b + c);
    }
}
```

a) 5.0
b) 6.0
c) 7.0
d) 8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(-2.2d);
        int b = Math.round(-2.2f);
        double c = Math.floor(-2.2d);
        System.out.println(a + b + c);
    }
}
```

a) -5.0
b) -6.0
c) -7.0
d) -8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(-2.2d);
        int b = Math.round(-2.2f);
        double c = Math.ceil(-2.2d);
        System.out.println(a + b + c);
    }
}
```

a) -5.0
b) -6.0
c) -7.0
d) -8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        int a = Math.round(-2.2d);
        int b = Math.round(-2.2f);
        double c = Math.ceil(-2.2d);
        System.out.println(a + b + c);
    }
}
```

a) -5.0

b) -6.0
c) -7.0
d) -8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(2.8d);
        int b = Math.round(2.8f);
        double c = Math.floor(2.8d);
        System.out.println(a + b + c);
    }
}
```

a) 5.0
b) 6.0
c) 7.0
d) 8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(2.8d);
        int b = Math.round(2.8f);
        double c = Math.ceil(2.8d);
        System.out.println(a + b + c);
    }
}
```

a) 5.0
b) 6.0
c) 7.0
d) 8.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(5.6d);
        int b = Math.round(-5.6f);
        double c = Math.ceil(5.8d);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 5 -6 6.0
b) 5 -5 6.0
c) 6 -6 6.0
d) 6 -5 6.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(5.6d);
        int b = Math.round(-5.6f);
        float c = Math.ceil(5.8d);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 5 -6 6.0
b) 5 -5 6.0
c) 6 -6 6.0
d) 6 -5 6.0
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        long a = Math.round(-5.6d);
        int b = Math.round(-5.6f);
        double c = Math.ceil(-5.8d);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) -5 -6 -6.0

b) -5 -5 -6.0
c) -6 -6 -6.0
d) -6 -5 -6.0
e) Kompilierfehler
g) Keine dieser Möglichkeiten.

(17) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.abs(-6.7d);
        float b = Math.abs(-6.6f);
        int c = Math.abs(-5);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6 6 5
b) 6.7 6.6 5
c) -6.7 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        float a = Math.abs(-6.7d);
        float b = Math.abs(-6.6f);
        int c = Math.abs(-5);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6 6 5
b) 6.7 6.6 5
c) -6.7 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
```

```
    public static void main(String[ ] args) {
        double a = Math.abs(6.7d);
        float b = Math.abs(6.6f);
        int c = Math.abs(5);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6 6 5
b) 6.7 6.6 5
c) -6.7 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.round(-3.6d);
        float b = Math.abs(-6.6f);
        double c = Math.sqrt(4);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) -3 -6.6 2.0
b) -4 -6.6 2.0
c) -4 6.6 2.0
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(21) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.round(-3.6d);
        float b = Math.abs(-6.6f);
        double c = Math.sqrt(-4);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) -3 -6.6 2.0
b) -4 -6.6 2.0
c) -4 6.6 2.0
d) Kompilierfehler

e) Keine dieser Möglichkeiten.

(22) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.round(-3.6d);
        float b = Math.abs(-6.6f);
        float c = Math.sqrt(4);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) -3 -6.6 2.0
b) -4 -6.6 2.0
c) -4 6.6 2.0
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(23) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6f);
        int c = Math.min(-4, -5);
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6.7 -6.6 -4
b) -3.6 5.6 -5
c) -3.6 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(24) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6f);
        byte c = Math.min((byte)-4,(byte) -5);
        System.out.println(a + " " + b + " " + c);
```

```
    }
}
```

a) 6.7 -6.6 -4
b) -3.6 5.6 -5
c) -3.6 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(25) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6f);
        int c = Math.min((byte)-4,(byte) -5);
        System.out.println(a + " " + b + " " + c);
    }}
```

a) 6.7 -6.6 -4
b) -3.6 5.6 -5
c) -3.6 -6.6 -5
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(26) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6f);
        int c = Math.min('s','g');
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6.7 -6.6 103
b) -3.6 5.6 103
c) -3.6 -6.6 103
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(27) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6f);
        char c = Math.min('s','g');
        System.out.println(a + " " + b + " " + c);}}
```

a) 6.7 -6.6 103
b) -3.6 5.6 103
c) -3.6 -6.6 103
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(28) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        double a = Math.min(-3.6d, 6.7d);
        float b = Math.max(-6.6f, 5.6d);
        int c = Math.min('s','g');
        System.out.println(a + " " + b + " " + c);
    }
}
```

a) 6.7 -6.6 103
b) -3.6 5.6 103
c) -3.6 -6.6 103
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(29) Frage

Welche der untenstehenden Funktionen aus java.lang.Math ist keine statische Methode?

a) min
b) sin
c) sqrt
d) round
e) ceil
f) floor
g) Keine dieser Möglichkeiten.

(30) Frage

Welche der untenstehenden Funkionen aus java.lang.Math hat keinen Rückgabetyp double?

a) min
b) sin
c) sqrt
d) round
e) ceil
f) floor
g) Keine dieser Möglichkeiten.

(31) Frage

Welche der untenstehenden Funkionen aus java.lang.Math hat keinen Rückgabetyp int?

a) min
b) abs
c) sqrt
d) round
e) random
f) sin
g) Keine dieser Möglichkeiten.

(32) Frage

Welche der untenstehenden Aussagen trifft zu?

a) Die Methode java.lang.Math.round addiert 0.5 und führt dann die Methode floor durch.
b) Die Methode java.lang.Math.min besitzt keinen Rückgabetyp int.
c) Die Methode java.lang.Math.sqrt hat den Rückgabetyp int.
d) Die Methode java.lang.Math.max besitzt keinen Rückgabetyp double.
e) Die Methode java.lang.Math.sqrt gibt für einen negativen Parameter einen Kompilierfehler zurück.

8. Lösungen

(1) Frage

c

(2) Frage

f

(3) Frage

d

(4) Frage

c

(5) Frage
f

(6) Frage
c

(7) Frage
c

(8) Frage
b

(9) Frage
c

(10) Frage
b

(11) Frage
e, Der Rückgabetyp für int a = Math.round(-2.2d)ist falsch, er müsste long sein.

(12) Frage
d

(13) Frage
f

(14) Frage
c

(15) Frage
e, Der Rückgabetyp für float c = Math.ceil(5.8d); ist falsch, er müsste double sein.

(16) Frage
g

(17) Frage
b

(18) Frage
d, Der Rückgabetyp für float a = Math.abs(-6.7d); ist falsch, er müsste double sein.

(19) Frage
b

(20) Frage

c

(21) Frage

e, Das Ergebnis von double c = Math.sqrt(-4); ist NaN.

(22) Frage

d, Der Rückgabetyp für float c = Math.sqrt(4); ist falsch, er müsste double sein.

(23) Frage

b

(24) Frage

d, Es gibt keine Rückgabetyp byte für die Funktion min, deshalb kommt es zu einem Kompilierfehler. Der entsprechende Rückgabetyp müsste int lauten.

(25) Frage

b

(26) Frage

b

(27) Frage

d, Es gibt keine Rückgabetyp char für die Funktion min, deshalb kommt es zu einem Kompilierfehler. Der entsprechende Rückgabetyp müsste int lauten.

(28) Frage

d, Achtung einer der Parameter ist eine double (float b = Math.max(-6.6f, 5.6d);) deshalb muß auch der Rückgabtyp ein double sein und nicht float. Deshalb kommt es zu einem Kompilierfehler.

(29) Frage

g

(30) Frage

d

(31) Frage

c, e, f

(32) Frage

a

D. Die Klasse String

Wollen Sie ganze Worte oder sogar Sätze speichern, brauchen Sie einen String. Ein String, sprich eine Aneinanderreihung von Zeichen oder auch eine Zeichenkette, kann auf zweierlei Arten erzeugt werden.

Erstens mit dem new-Operator, der ein neues Objekt erstellt:

String s = new String(“Ich bin eine Zeichenkette!“)

Und zweitens als Zuweisung zu einem String-Literal. Bekanntlich ist der Wert, der der Variablen während der Initialisierung zugewiesen wird, ein sogenannter Literal (vgl. Kapitel Literale).

String s = “Ich bin eine Zeichenkette!“

Alle String-Objekte werden einem sogenannten String-Pool hinzugefügt, wobei mit new immer ein neues String-Objekt mit einer neuen Adresse erzeugt wird. Im Gegensatz hierzu wird bei einem String-Literal, der String-Pool nach einem exakt gleichen String durchsucht. Ist ein solcher vorhanden, wird diesem String eine Referenz auf den bereits vorhanden String zugewiesen. Sprich sie besitzen anschließend beide eine Referenz auf das gleiche Objekt, sie haben die gleiche Adresse. Dieser Umstand spielt beim Vergleich von String-Objekten, der weiter unten näher erläutert wird, eine Rolle.

Hier nun eine weitere Eigenschaft von String-Objekten: String-Objekte sind im Gegensatz zu StringBuffer-Objekten unveränderbar. So ändern z. B. die Methoden toUpperCase() oder toLowerCase(), die den String in Groß- oder Kleinbuchstaben umwandeln sollen, nicht den vorhandenen String, sondern erzeugen eine neuen String.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”ich“;
                s.toUpperCase();
                System.out.println(s);
        }
}
```

Ausgabe auf der Konsole:

```
ich
```

Der neu entstandene String muß der Variablen s erneut zugewiesen werden, damit ICH in Großbuchstaben ausgegeben werden kann.

```
public class Texte {
        public static void main(String[ ] args) {
```

```
                String s = ”ich“;
                s = s.toUpperCase();
                System.out.println(s);
        }
}
```

Ausgabe auf der Konsole:

ICH

1. Die Methode concat()

Mit Hilfe der Methode concat(String str) können Sie, an einen String einen anderen String anhängen. Dieser so entstandene String muß anschließend auch wieder einer Variablen zugewiesen werden.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”ich“;
                s = s.concat(“ bin ein angehängter String!”);
                System.out.println(s);
        }
}
```

Ausgabe auf der Konsole:

ich bin ein angehängter String!

2. Die Methode String replace(char oldChar(), char newChar())

Mit dieser Methode können einzelne Zeichen des Strings ersetzt werden. In untenstehendem Beispiel wird das i durch ein a ersetzt.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”ich“;
                s = s.replace(‘i‘, ‚a‘);
                System.out.println(s);
        }
}
```

Ausgabe auf der Konsole:

ach

3. Die Methode int indexOf() und lastIndexOf()

Diese Methoden durchsuchen eine Zeichenkette nach einem bestimmten Zeichen und geben dann die Position dieses Zeichens zurück. So befindet sich in unserem Beispiel der erste Buchstabe M an der Stelle 0, das a an der Stelle 1, das r an der Stelle 2 und das i an der Stelle 3 usw.

M	a	r	i	o	,		M	a	r	i	o
0	1	2	3	4	5	6	7	8	9	10	11

Die Methode int indexOf(String str) gibt die Position eines Zeichens oder einer Zeichenkette wieder, wohingegen die Methode int indexOf(String str, int fromIndex) erst ab einer bestimmten Stelle anfängt zu suchen. In untenstehendem Beispiel fängt sie erst an der Position 4 an zu suchen, an der Stelle, an der das erste o steht. So wird das zweite i, an der Indexposition 10, als Ergebnis ermittelt.

```
public class Texte {
	public static void main(String[ ] args) {
		String s = ”Mario, Mario“;
		int a = s.indexOf(”i“);
		int b = s.indexOf(”i“, 4);
		System.out.println(a);
		System.out.println(b);
	}
}
```

Ausgabe auf der Konsole:

```
3
10
```

Wie wir unten sehen, funktioniert die Methode int lastIndexOf(String str) ähnlich: Sie beginnt aber hinten zu suchen und zählt die gefundene Position von vorne durch. Die Methode int lastIndexOf(String str, int fromIndex) fängt zusätzlich erst an der Position fromIndex an zu suchen. Die Stelle fromIndex wird aber wieder von vorne durchgezählt.

```
public class Texte {
	public static void main(String[ ] args) {
		String s = ”Mario, Mario“;
		int a = s.lastIndexOf(”i“);
		int b = s.lastIndexOf(”i“, 4);
		System.out.println(a);
		System.out.println(b);
	}
}
```

Ausgabe auf der Konsole:

10
3

Ist die Suche nicht erfolgreich wird -1 zurückgegeben.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”Mario, Mario“;
                int a = s.indexOf(”p“);
                int b = s.indexOf(”s“, 4);
                System.out.println(a);
                System.out.println(b);
        }
}
```

Ausgabe auf der Konsole:

-1
-1

4. Die Methode String trim()

Die Methode String trim() löscht die Leerzeichen am Anfang und Ende der Zeichenkette.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = “    Mario, Mario    “;
                String a = s.trim();
                System.out.println(a + a);
        }
}
```

Ausgabe auf der Konsole:

Mario, MarioMario, Mario

5. Die Methode substring()

Die Methoden String substring(int startIndex) und String substring(int startIndex, int endIndex) geben eine Zeichenkette zurück. Bei der ersten muß die Startposition eingegeben werden und bei der zweiten die Start- und Endposition.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”Mario, Mario“;
                String a = s.substring(3);
                String b = s.substring(3, 5);
                System.out.println(a);
                System.out.println(b);
        }
}
```

Ausgabe auf der Konsole:

```
io, Mario
io
```

6. Die Methode charAt()

Mit der Methode char charAt(int index) wird das Zeichen ermittelt, das sich an einer bestimmten Indexposition befindet. Bitte beachten Sie, daß der Rückgabetyp dieser Methode char ist.

```
public class Texte {
        public static void main(String[ ] args) {
                String s = ”Mario, Mario“;
                char a = s.charAt(3);
                System.out.println(a);
        }
}
```

Ausgabe auf der Konsole:

```
i
```

7. Umwandeln von primitiven Datentypen und Objekten in einen String

Sie können primitive Datentypen und Objekte in ein String-Objekt umwandeln. Bitte beachten Sie, daß die übergebenen Parameter nicht die Werte byte oder short annehmen können.

```
static String valueOf(Object obj)
static String valueOf(char[ ] character)
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(float f)
static String valueOf(double d)
```

8. Übungen

(1) Frage

Welche der untenstehenden Methoden sind Methoden der Klasse java.lang. String?

a) random
b) concat
c) delete
d) substring
e) insert
f) sqrt
g) append

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Fenster“;
        char b = s.charAt(3);
        System.out.println(b);
    }
}
```

a) ster
b) s
c) t
d) ter
e) n
f) nster
g) Kompilierfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Fenster“;
        String b = s.charAt(3);
        System.out.println(b);
    }
}
```

a) ster
b) s
c) t

d) ter
e) n
f) nster
g) Kompilierfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Fenster“;
        s.concat(“ zum Hof”);
        System.out.println(s); }
}
```

a) Fenster
b) Fenster zum Hof
c) Fensterzum Hof
d) Kompilierfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Fenster“;
        s = s.concat(“ zum Hof”);
        System.out.println(s);
    }
}
```

a) Fenster
b) Fenster zum Hof
c) Fensterzum Hof
d) Kompilierfehler

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”    Honig    ”;
        s.trim();
        s.concat(“ zum Frühstück”);
        System.out.println(s);
    }}
```

a) Honig zum Frühstück
b) Honig
c) Honig
d) Kompilierfehler

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”    Honig    ”;
        s = s.trim();
        s.concat(“ zum Frühstück”);
        System.out.println(s);
    }
}
```

a) Honig zum Frühstück
b) Honig
c) Honig
d) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”    Honig    ”;
        s = s.trim();
        s = s.concat(“ zum Frühstück”);
        System.out.println(s);
    }
}
```

a) Honig zum Frühstück
b) Honig
c) Honig
d) Kompilierfehler

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”maus“;
        s = s.toUpperCase();
        s = s.replace(‘u‘, ‘i‘);
```

```
            System.out.println(s);
        }
}
```

a) maus
b) MAUS
c) MAIS
d) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”maus“;
        s = s.toUpperCase();
        s = s.replace(‘U‘, ‘I‘);
        System.out.println(s); }}
```

a) maus
b) MAUS
c) MAIS
d) Kompilierfehler

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.indexOf(”a“);
        System.out.println(s);
    }
}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
```

```
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.indexOf(”a“);
        System.out.println(i);
    }
}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.lastIndexOf(”a“);
        System.out.println(i);
    }
}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.lastIndexOf(”a“, 3);
        System.out.println(i);
    }}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.lastIndexOf(”a“, 10);
        System.out.println(i);
    }
}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Das Haus ist neu!”;
        int i = s.indexOf(”a“, 10);
        System.out.println(i);
    }
}
```

a) Das Haus ist neu!
b) 1
c) as Haus ist neu!
d) 5
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(17) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = “Die Sonne scheint!”;
        String a = s.substring(4);
        String b = s.substring(4, 7);
        System.out.println(a);
        System.out.println(b);}}
```

a) Die Sonne scheint! Die Sonne scheint!
b) Sonne scheint! Son
c) S S n
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Honigmund“;
        String a = s.substring(7);
        String b = s.substring(4, 7);
        System.out.println(a);
        System.out.println(b);
    }
}
```

a) n g n
b) nd gmu
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Ledersofa“;
        String a = s.substring(7);
        char b = s.charAt(7);
        System.out.println(a);
        System.out.println(b);
    }
}
```

a) fa a
b) of o
c) a fa
d) o of
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class A {
    public static void main(String[ ] args) {
        String s = ”Ledersofa“;
        String a = s.substring(6);
        char b = s.charAt(6);
        System.out.println(a);
        System.out.println(b);
    }
}
```

a) ofa s
b) sofa s
c) o ofa
d) ofa o
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

9. Lösungen

(1) Frage

b,d

(2) Frage

b

(3) Frage

g, Der Rückgabetyp von String b = s.charAt(3); ist falsch, er müsste char sein.

(4) Frage

a, Strings sind unveränderbar, es ist ein neuer String entstanden, dieser ist nicht erneut einer Variablen zugewiesen worden, deswegen kommt es nur zur Ausgabe von Fenster.

(5) Frage

b

(6) Frage

b, Strings sind unveränderbar, es ist ein neuer String entstanden, dieser ist nicht erneut einer Variablen zugewiesen worden, deswegen kommt es nur zur Ausgabe von “ Honig “.

(7) Frage

c, Strings sind unveränderbar, es ist ein neuer String entstanden, bei dem die Leerzeichen gelöscht worden sind und dieses Ergebnis ist erneut der Variablen s

zugewiesen worden, somit kommt es zur Ausgabe von Honig.

(8) Frage

a,

(9) Frage

b, Da der String in Großbuchstaben umgewandelt worden ist, wird das kleine u nicht mehr gefunden, da es nun ein großes U ist. Somit wird es auch nicht ersetzt.

(10) Frage

c

(11) Frage

a, Achtung die Variable s soll auf dem Bildschirm ausgegeben werden und nicht die Variable i.

(12) Frage

b

(13) Frage

d

(14) Frage

b

(15) Frage

d

(16) Frage

f, Keine dieser Möglichkeiten, da erst ab der 10 Position gesucht wird, und a nicht gefunden werden kann. Das Ergebnis ist -1.

(17) Frage

b

(18) Frage

b

(19) Frage

f

(20) Frage

d

E. Die Klasse StringBuffer

Im Gegensatz zu der Klasse String, die nicht veränderbare Strings erstellt, können Objekte der Klasse StringBuffer verändert werden, ohne daß dabei eine neues Objekt ensteht.

1. Konstruktoren der Klasse StringBuffer

Als erstes lernen wir, wie wir ein Objekt der Klasse StringBuffer instanziieren können. Es gibt drei Konstruktoren: Der Konstruktor StringBuffer(String s) erstellt ein Objekt von der Länge des Strings, der übergeben wird, plus einer zusätzlichen Kapazität von 16. Der Konstruktor StringBuffer(int length) legt einen leeren StringBuffer von der Kapazität der Länge des Parameters int an. Diese Kapazität ist in unserem untenstehenden Beispiel 2. Der Konstruktor StringBuffer() erzeugt ein Objekt der Klasse StringBuffer mit der Kapazität 16.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                int i = 2;
                String s = ”ich“;

                //Stringbuffer mit Kapazität 16 + Länge des Wortes
                StringBuffer a = new StringBuffer(s);
                //Stringbuffer mit Kapazität 2
                StringBuffer b = new StringBuffer(i);
                //Stringbuffer mit Kapazität 16
                StringBuffer c = new StringBuffer();

                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
        }
}
```

Ausgabe auf der Konsole:

```
ich
```

Wir haben also gesehen: Unter Kapazität wird die potentielle Länge, sprich das Fassungsvermögen, verstanden und nicht die tatsächliche Länge.

2. Länge und Kapazität eines StringBuffers

Wollen Sie die Kapazität eines StringBuffers feststellen, müssen Sie die Methode int capacity benutzen. Die Länge können Sie mit der Methode length() herausfinden. Der untenstehende StringBuffer hat somit nur die Länge 3, aber eine Kapazität von 3 plus 16, also 19.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
```

```
                StringBuffer str = new StringBuffer(”ich“);
                int a = str.length();
                int b = str.capacity();

                System.out.println(a);
                System.out.println(b);
        }
}
```

Ausgabe auf der Konsole:

```
3
19
```

Noch eine Methode: setLength() mit der Sie die Länge des StringBuffers verändern können. Wenden Sie diese Methode an, kann dies zur Folge haben, daß der bereits existierende Text abgeschnitten wird. Nehmen wir untenstehendes Beispiel: Im Gegensatz zu String-Objekten läßt sich der ursprüngliche StringBuffer verändern. Es entsteht kein neuer StringBuffer, der einer Variablen zugewiesen werden muß. StringBuffers sind im Unterschied zu Strings veränderbar.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                StringBuffer str = new StringBuffer(”ich“);
                str.setLength(2);
                System.out.println(str);
        }
}
```

Ausgabe auf der Konsole:

```
ic
```

3. Umwandeln eines StringBuffers in einen String

Die Methode toString() wandelt einen StringBuffer in einen String um.

4. Die Methode append()

Mit dieser Methode kann man an das Ende eines StringBuffers ein Zeichen oder eine Zeichenkette anfügen. Stellen wir wieder die überladenen Versionen der append()-Methode zusammen:

```
StringBuffer append(String str)
StringBuffer append(char[ ] str)
StringBuffer append(char[ ] str, int offset, int len)
StringBuffer append(char c)
```

Die untenstehenden überladenen Methoden append() wandeln zuerst den primitiven Wert des Parameters mit Hilfe von String.valueOf() in einen String um und dann erst in einen StringBuffer.

StringBuffer append(boolean b)
StringBuffer append(int i)
StringBuffer append(long l)
StringBuffer append(float f)
StringBuffer append(double d)

Was lernen wir aus untenstehendem Beispiel? Wir lernen: Strings und StringBuffer verhalten sich unterschiedlich. Ein StringBuffer läßt sich verändern, ein String nicht. Es entsteht kein neuer StringBuffer, der einer Variablen zugewiesen werden muß, damit er auf dem Bildschirm ausgegeben werden kann.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                StringBuffer str = new StringBuffer("ich");
                str.append(" bin ein Genie!");
                System.out.println(str);
        }
}
```

Ausgabe auf der Konsole:

ich bin ein Genie!

5. Die Methode insert()

Mit dieser Methode können Sie an einer bestimmten Indexposition, z. B. ein Zeichen, einfügen. Es gibt folgende überladene Versionen der insert()-Methode:

StringBuffer insert(int offset, Object obj)
StringBuffer insert(int offset, String str)
StringBuffer insert(int offset, char[ ] str)
StringBuffer insert(int offset, char c)
StringBuffer insert(int offset, boolean b)
StringBuffer insert(int offset, int i)
StringBuffer insert(int offset, float f)
StringBuffer insert(int offset, double d)

An der Stelle 1 (es wird bei 0 angefangen zu zählen) wird ein neues Wort eingefügt.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                StringBuffer str = new StringBuffer(”ich“);
                str.insert(1, “ bin ein Genie!”);
                System.out.println(str);
        }
}
```

Ausgabe auf der Konsole:

i bin ein Genie!ch

6. Die Methode delete()

Sie können entweder ein Zeichen löschen, mit StringBuffer deleteCharAt(int index), oder mehrere Zeichen mit StringBuffer delete(int start, int end).

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                StringBuffer str = new StringBuffer(”Meinung“);
                str.deleteCharAt(1);
                System.out.println(str);
                str.delete(0, 2);
                System.out.println(str);
        }
}
```

Ausgabe auf der Konsole:

Minung
nung

7. Die Methode reverse()

Die Methode reverse() dreht die Buchstabenreihenfolge des StringBuffers um.

```
public class VeraenderbarerText {
        public static void main(String[ ] args) {
                StringBuffer str = new StringBuffer(“Meinung”);
                str.reverse();
                System.out.println(str);
                }
}
```

Ausgabe auf der Konsole:

gnunieM

8. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer("Die Sonne scheint!");
        str.setLength(3);
        System.out.println(str);
    }
}
```

a) Die Sonne scheint!
b) Die
c) 3
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer("Die Sonne scheint!");
        str.getLength();
        System.out.println(str);
    }
}
```

a) Die Sonne scheint!
b) Die
c) 18
d) 34
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer("Die Sonne scheint!");
        str.length();
        System.out.println(str);
    }
}
```

a) Die Sonne scheint!
b) Die
c) 18
d) 34
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer(”heute“);
        str.append(“ morgen”);
        System.out.println(str);
    }
}
```

a) heute
b) heute morgen
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String str = new String(”heute“);
        str.append(“ morgen”);
        System.out.println(str);
    }
}
```

a) heute
b) heute morgen
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String str = new String(”heute“);
```

```
            str.concat(“ morgen”);
            System.out.println(str);
        }
}
```

a) heute
b) heute morgen
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
        public static void main(String[ ] args) {
            String str = new String(”heute“);
            str = str.concat(“ morgen”);
            System.out.println(str);
        }
}
```

a) heute
b) heute morgen
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
        public static void main(String[ ] args) {
            String s = new String(”A“);
            s.concat(”B“);
            StringBuffer str = new StringBuffer(”A“);
            str.append(”B“);
            System.out.println(s + str);
        }
}
```

a) ABAB
b) AA
c) AAB
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer(”Haus“);
        str.insert(1, “B”);
        System.out.println(str);
    }
}
```

a) BHaus
b) HBaus
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer(”Haus“);
        str.delete(1, 2);
        System.out.println(str);
    }
}
```

a) Hs
b) Hus
c) us
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer str = new StringBuffer(”Haus“);
        str.reverse();
        System.out.println(str);
    }
}
```

a) Haus
b) suaH
c) aus
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(12) Frage

Welche der untenstehenden Methoden sind Methoden der Klasse java.lang. StringBuffer?

a) append
b) concat
c) toUpperCase
d) substring
e) insert
f) delete

(13) Frage

Welche der untenstehenden Methoden sind Methoden der Klasse java.lang. String?

a) toLowerCase
b) concat
c) append
d) trim
e) delete
f) insert

9. Lösungen

(1) Frage

b

(2) Frage

e, Es kommt zu einem Kompilierfehler, da die zugehörige Methode int length() heißt und nicht getLength().

(3) Frage

c, 18 ist die Länge und 34 die maximale Kapazität.

(4) Frage

b

(5) Frage

c, In der Klasse String gibt es keine Methode append.

(6) Frage

a, Strings sind im Gegensatz zu StringBuffers unveränderbar, es ist ein neuer String entstanden, dieser ist nicht erneut einer Variablen zugewiesen worden, deswegen

kommt es nur zur Ausgabe von heute.

(7) Frage
b

(8) Frage
c, Strings sind im Gegensatz zu StringBuffers unveränderbar, es ist ein neuer String entstanden, dieser ist nicht erneut einer Variablen zugewiesen worden, deswegen kommt es nur zur Ausgabe von AAB.

(9) Frage
b

(10) Frage
b

(11) Frage
b

(12) Frage
a, e, f

(13) Frage
a,b,d

F. Stringkonkatenation

1. Verketten von Objekten der String-Klasse mit Objekten der String-Klasse

Mit dem Operator + kann man an das Ende eines Strings ein anderes String-Objekt anfügen. Wie Sie in untenstehendem Beispiel sehen können, wird der String s an den String st angehängt.

```
public class Stringkonkatenation {
        public static void main(String[ ] args) {
                String st = new String(”ich“);
                String s = new String(” bin“);
                System.out.println(st +  s);
        }
}
```

Ausgabe auf der Konsole:

```
ich bin
```

Das gleiche gilt für Zahlen. In der Klasse java.lang.String ist der binäre Operator + überladen, das bedeutet, daß zwei Zahlen, die von Typ String sind, nicht miteinander addiert werden. Stattdessen werden die Zahlen wie Text behandelt und es wird, wie Sie unten sehen können, Text an Text gehängt. So erhalten Sie das Ergebnis 55 und nicht 10, da die zwei Zahlen nicht zusammengezählt werden.

```
public class Stringkonkatenation {
        public static void main(String[ ] args) {
                String s = new String(”5“);
                System.out.println(s +  s);
        }
}
```

Ausgabe auf der Konsole:

```
55
```

2. Verketten von Objekten der String-Klasse mit primitiven Datentypen

Was passiert, wenn Sie mit dem Operator + an einen String einen primitiven Datentypen anhängen? Es werden alle Operanden implizit in einen String umgewandelt. So haben Sie in untenstehendem Beispiel als Ergebnis den Satz „Ich habe 55“ und nicht „Ich habe 10“. Die zwei Zahlen 5, die vom Typ int sind, werden also implizit in einen String umgewandelt.

```
public class Stringkonkatenation {
        public static void main(String[ ] args) {
                String s = new String(”Ich habe ”);
                System.out.println(s +  5 + 5);
        }
}
```

Ausgabe auf der Konsole:

Ich habe 55

Stehen die beiden Zahlen vom Typ int am Anfang, sieht das Ergebnis anders aus, nun werden beide Zahlen miteinander addiert, da sie in diesem Fall nicht automatisch in einen String abgeändert werden.

```
public class Stringkonkatenation {
        public static void main(String[ ] args) {
                String s = new String(”Ich habe ”);
                System.out.println(5 + 5 + s);
        }
}
```

Ausgabe auf der Konsole:

10Ich habe

3. Verketten von Objekten der String-Klasse mit Objekten der StringBuffer-Klasse

Will man zwei Stringbuffer mit dem Operator + verketten, kommt es zu einem Kompilierfehler, da der Operator + in der Klasse StringBuffer nicht definiert ist.

```
5  public class Stringkonkatenation {
6      public static void main(String[] args) {
7          StringBuffer st = new StringBuffer("ja");
8          System.out.println(st + st);
9      }
10 }
```

Zusätzlich ist einiges zu beachten, wenn Objekte der Klasse StringBuffer durch den + Operator mit Objekten der Klasse String kombiniert werden. Steht das String-Objekt am Anfang, so werden die Objekte der StringBuffer-Klasse implizit in ein

String-Objekt umgewandelt.

```
public class Stringkonkatenation {
        public static void main(String[ ] args) {
                StringBuffer sb = new StringBuffer(”25“);
                String s = new String(“Ja: ”);

                System.out.println(s + sb + sb);
        }
}
```

Ausgabe auf der Konsole:

Ja: 2525

Die umgekehrte Reihenfolge führt allerdings wieder zu einem Kompilierfehler.

```
5  public class Stringkonkatenation {
6      public static void main(String[] args) {
7          StringBuffer sb = new StringBuffer("25");
8          String s = new String("Ja: ");
9          System.out.println(sb + sb + s);
10     }
11 }
```

4. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		String st = new String(”1“);
		String s = new String(”Die Summe ist: ”);
		System.out.println(s + st + st);
	}
}
```

a) Die Summe ist: 11
b) die Summer ist: 2
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		StringBuffer st = new StringBuffer(”1“);
		StringBuffer s = new StringBuffer(”Die Summe ist:  ”);
		System.out.println(s + st + st);
	}
}
```

a) Die Summe ist: 11
b) die Summer ist: 2
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
	public static void main(String[ ] args) {
		StringBuffer st = new StringBuffer(”1“);
		String s = new String (”Die Summe ist: ”);
		System.out.println(s + st + st);
	}
}
```

a) Die Summe ist: 11
b) die Summer ist: 2

c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        StringBuffer st = new StringBuffer(”1“);
        String s = new String (”Die Summe ist: ”);
        System.out.println(st + st + s);
    }
}
```

a) Die Summe ist: 11
b) die Summer ist: 2
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        String s = new String(” Die Summe ist: ”);
        System.out.println(1+ 1 + s);
    }
}
```

a) 11 Die Summe ist:
b) 2 die Summer ist:
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        String st = new String(”30“);
        String s = new String(”20“);
        System.out.println(st + s);
    }
}
```

a) 50
b) 3020

c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        String st = new String(”30“);
        String s = new String(”20“);
        System.out.println(st + s + 20);
    }
}
```

a) 70
b) 302020
c) Kompilierfehler
d) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        String st = new String(”30“);
        String s = new String(”20“);
        System.out.println(30 + 20 + st + s);
    }
}
```

a) 503020
b) 100
c) 30203020
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class S {
    public static void main(String[ ] args) {
        String st = new String(”30“);
        String s = new String(”20“);
        System.out.println(st + s+ 30 + 20);
    }
}
```

a) 302050
b) 100
c) 30203020
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

5. Lösungen

(1) Frage
a

(2) Frage
c

(3) Frage
a

(4) Frage
c

(5) Frage
b

(6) Frage
b

(7) Frage
b

(8) Frage
a

(9) Frage
c

G. Vergleich von Objekten und primitiven Datentypen

1. Vergleich von Strings

Wir haben bereits im Kapitel Die Klasse String gesehen: Es werden alle String-Objekte in einem sogenannten String-Pool erstellt, wobei mit new immer ein neues String-Objekt erzeugt wird, das immer eine neue Adresse erhält. Im Gegensatz hierzu wird bei der Erzeugung mit Hilfe eines String-Literals, der String-Pool nach einem exakt gleichen String durchsucht. Ist ein solcher vorhanden, wird diesem String eine Referenz auf den bereits vorhanden String zugewiesen, sprich sie haben die gleiche Adresse. Dies steht im Gegensatz zu String-Objekten, die trotz identischem Inhalt immer unterschiedliche Referenzen haben. Diese Tatsache hat Einfluß auf den Vergleich von String-Objekten und von String-Literalen.

a) Vergleich von String-Objekten

Der Vergleichsoperator == überprüft, ob die Referenzen der beiden Objekte übereinstimmen. Er vergleicht also nicht den Inhalt, sondern str und str1, und diese beiden Adressen sind definitiv nicht identisch. Will man allerdings den Inhalt der beiden Objekte gegenüberstellen, braucht man die Methode boolean equals(Object anObject) der Klasse java.lang.String. Die Methode equals erbt die Klasse String von der Klasse java.lang.Object und überschreibt sie. So gibt der Vergleich in untenstehendem Beispiel mit str == str1 false und mit str.equals(str1) true zurück.

```
public class Vergleich {
        public static void main(String[ ] args) {
                String str = new String(”ich“);
                String str1 = new String (”ich“);
                System.out.println(str == str1);
                System.out.println(str.equals(str1));
        }
}
```

Ausgabe auf der Konsole:

```
false
true
```

b) Vergleich von String-Literalen

Bei String-Literalen weicht das Ergebnis ab, da String-Literale im String-Pool die gleichen Referenzen zugewiesen bekommen, so führen beide Vergleiche zu der Ausgabe von true (vgl. Kapitel String-Literale und Kapitel Die Klasse String).

```
public class Vergleich {
        public static void main(String[ ] args) {
```

```
                    String str = “ich”;
                    String str1 = “ich”;
                    System.out.println(str == str1);
                    System.out.println(str.equals(str1));
          }
}
```

Ausgabe auf der Konsole:

```
true
true
```

c) Vergleich von String-Literalen und String-Objekten

Überprüfen Sie String-Literale mit String-Objekten auf Gleichheit, so erhalten Sie die gleichen Ergebnisse wie oben bei der Gegenüberstellung mit == und equals() von String-Objekten.

```
public class Vergleich {
          public static void main(String[ ] args) {
                    String str = “ich”;
                    String str1 = new String (”ich“);
                    System.out.println(str == str1);
                    System.out.println(str.equals(str1));
          }
}
```

Ausgabe auf der Konsole:

```
false
true
```

2. Vergleich von Objekten der Klasse StringBuffer

In der Klasse StringBuffer wird die Methode boolean equals(Object anObject) auch von der Klasse java.lang.Object geerbt, aber nicht überschrieben. Dies hat zur Folge, daß nicht der Inhalt der Objekte auf Identität untersucht wird, deswegen kommt es, wie Sie in untenstehendem Beispiel erkennen können, bei beiden Arten des Vergleichs zu der Ausgabe von false. Man kann also nicht den Inhalt von StringBuffer-Objekten mit der Methode equals auf Gleichheit überprüfen, da dies nicht den gewünschten Effekt hat.

```
public class Vergleich {
          public static void main(String[ ] args) {
                    StringBuffer str = new StringBuffer(”ich“);
                    StringBuffer str1 = new StringBuffer (”ich“);
                    System.out.println(str == str1);
                    System.out.println(str.equals(str1));
```

```
        }
}
```

Ausgabe auf der Konsole:

```
false
false
```

3. Vergleich von Objekten der Klasse String und StringBuffer

Nehmen wir an, Sie setzen Objekte der Klasse String und StringBuffer mit der Methode equals() miteinander in Beziehung, so wird false zurückgegeben. Untersuchen Sie String-Literale und Objekte der Klasse StringBuffer auf Übereinstimmung, erhalten Sie ebenfalls false als Ergebnis.

```
public class Vergleich {
        public static void main(String[ ] args) {
                String s = new String(”also“);
                StringBuffer sb = new StringBuffer (”also“);
                boolean b = s.equals(sb);
                System.out.println(b);
        }
}
```

Ausgabe auf der Konsole:

```
false
```

Zu Problemen kommt es allerdings, wenn Objekte der Klasse String oder String-Literale mit Objekten der Klasse StringBuffer mit == verglichen werden.

```
5  public class Vergleich {
6      public static void main(String[] args) {
7          String s = new String("also");
8          StringBuffer sb = new StringBuffer ("also");
9          boolean b = s == sb;
10         System.out.println(b);
11     }
12 }
```

4. Vergleich von Objekten der Wrapper-Klassen

In allen Wrapper-Klassen gibt es eine equals-Methode, die von der Klasse java.lang.Object geerbt und überschrieben wird, so wird es möglich, den Inhalt der Objekte auf Gleichheit zu überprüfen. Wohingegen mit dem Vergleich mit == bei

den Wrapper-Objekten nur die Adresse, also die Referenz, in Beziehung gesetzt wird.

```
public class Vergleich {
	public static void main(String[ ] args) {
		Boolean b1 = new Boolean(true);
		Boolean b2 = new Boolean(true);
		System.out.println(b1 == b2);
		System.out.println(b1.equals(b2));
	}
}
```

Ausgabe auf der Konsole:

```
false
true
```

Werden unterschiedliche Wrapper-Objekte gegenübergestellt, gibt die Methode equals() false zurück.

```
public class Vergleich {
	public static void main(String[ ] args) {
		Integer i = new Integer(”7“);
		Long l = new Long(”7“);
		System.out.println(i.equals(l));
	}
}
```

Ausgabe auf der Konsole:

```
false
```

Bei einem Vergleich mit == kommt es bei unterschiedlichen Wrappern zu einem Kompilierfehler.

```
5  public class Vergleich {
6      public static void main(String[] args) {
7          Integer i = new Integer(7);
8          Long l = new Long(7);
9          System.out.println(i == l);
10     }
11 }
```

5. Vergleich von primitiven Datentypen

Primitive Typen kann ich nur mit dem Operator == auf Entsprechung untersuchen.

```
public class Vergleich {
        public static void main(String[ ] args) {
                int i = 7;
                long l = 7;
                System.out.println(i == l);
        }
}
```

Ausgabe auf der Konsole:

true

Ein Vergleich mit der Methode equals() ist nicht möglich, da primitive Datentypen keine Methoden besitzen. Somit haben sie auch keine equals()-Methode.

```
5  public class Vergleich {
6      public static void main(String[] args) {
7          int i = 7;
8          long l = 7;
9          boolean b = i.equals(l);
10         System.out.println(b);
11     }
12 }
```

6. Vergleich von primitiven Datentypen mit Objekten der Wrapper-Klasse

Eine weitere wichtige Feststellung zum Schluß: Ich kann einen primitiven Datentyp mit einem Objekt der Wrapper-Klasse weder mit dem Vergleichsoperator noch mit der Methode equals() auf Identität überprüfen. In beiden Fällen kommt es zu einem Kompilierfehler.

```
5  public class Vergleich {
6      public static void main(String[] args) {
7          int i = 7;
8          Integer i1 = new Integer(7);
9          boolean b = i.equals(i1);
10         boolean b1 = i == i1;
11         System.out.println(b);
12     }
13 }
```

7. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        long l = 2;
        Long l1 = new Long(2);
        boolean b = l1.equals(l);
        boolean b1 = l1 == l;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        Long l = new Long(2);
        Long l1 = new Long(2);
        boolean b = l1.equals(l);
        boolean b1 = l1 == l;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
```

```
        String s = new String(”neu“);
        String s1 = new String(”neu“);
        boolean b = s.equals(s1);
        boolean b1 = s == s1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = ”neu“;
        String s1 = new String(”neu“);
        boolean b = s.equals(s1);
        boolean b1 = s == s1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = ”neu“;
        String s1 = ”neu“;
        boolean b = s.equals(s1);
        boolean b1 = s == s1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer sb = new StringBuffer(”neu“);
        StringBuffer sb1 = new StringBuffer(”neu“);
        boolean b = sb.equals(sb1);
        boolean b1 = sb == sb1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        Integer i = new Integer(1);
        Integer i1 = new Integer(1);
        boolean b = i.equals(i1);
        boolean b1 = i == i1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class B {
    public static void main(String[ ] args) {
        int i = 1;
        Integer i1 = new Integer(1);
        boolean b = i.equals(i1);
        boolean b1 = i == i1;
        System.out.println(b);
        System.out.println(b1);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        int i = 1;
        long l =  1;
        boolean b = i.equals(l);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        int i = 1;
        long l =  1;
        boolean b = i == l;
        System.out.println(b);
    }
}
```

a) true
b) false

c) Kompilierfehler

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        Integer i = new Integer(1);
        Long l = new Long (1);
        boolean b = i == l;
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        Integer i = new Integer(1);
        Long l = new Long (1);
        boolean b = i.equals(l);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = new String(”neu“);
        StringBuffer sb = new StringBuffer (”neu“);
        boolean b = s == sb;
        System.out.println(b);
    }
}
```

a) true

b) false
c) Kompilierfehler

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = new String(”neu“);
        StringBuffer sb = new StringBuffer (”neu“);
        boolean b = s.equals(sb);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = new String(”neu“);
        String sb = new String (”neu“);
        boolean b = s.equals(sb);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer s = new StringBuffer(”neu“);
        StringBuffer sb = new StringBuffer (”neu“);
        boolean b = s.equals(sb);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(17) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = ”neu“;
        String sb = “neu”;
        boolean b = s.equals(sb);
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = ”neu“;
        String sb = “neu”;
        boolean b = s == sb;
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        StringBuffer s = new StringBuffer(”neu“);
        StringBuffer sb = new StringBuffer (”neu“);
        boolean b = s == sb;
        System.out.println(b);
    }
}
```

a) true
b) false
c) Kompilierfehler

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class B {
    public static void main(String[ ] args) {
        String s = new String (”neu“);
        String sb = new String (”neu“);
        boolean b = s == sb;
        System.out.println(b);
    }}
```

a) true
b) false
c) Kompilierfehler

(21) Frage

In welchen Klassen wird die equals-Methode nicht überschrieben.

a) String
b) Double
c) Boolean
d) Integer
e) Long
f) Short
g) Keine dieser Möglichkeiten.

(22) Frage

In welchen Klassen wird die equals-Methode nicht überschrieben.

a) StringBuffer
b) Double
c) Boolean
d) Integer
e) Long
f) Short
g) Keine dieser Möglichkeiten.

8. Lösungen

(1) Frage

e, Primitive Datentypen können weder mit der equals-Methode noch mit dem Vergleichsoperator == mit Objekten der Wrapper-Klassen verglichen werden.

(2) Frage

b

(3) Frage

b

(4) Frage

b

(5) Frage

a

(6) Frage

d

(7) Frage

b

(8) Frage

e, Primitive Datentypen können weder mit der equals-Methode noch mit dem Vergleichsoperator == mit Objekten der Wrapper-Klassen verglichen werden.

(9) Frage

c, Primitive Datentypen besitzen keine Methoden.

(10) Frage

a

(11) Frage

c, Unterschiedliche Objekte von Wrapper-Klassen können nicht mit dem Operator == verglichen werden.

(12) Frage

b

(13) Frage

c, Objekte der Klasse String und der Klasse String-Buffer können nicht mit dem Operator == verglichen werden.

(14) Frage
b

(15) Frage
a

(16) Frage
b

(17) Frage
a

(18) Frage
a

(19) Frage
b

(20) Frage
b

(21) Frage
g

(22) Frage
a

IX. Exceptions

1. Begriffserläuterung

Immer wieder tauchen in Programmen Fehler auf, Fehler, die entweder durch externe Quellen entstehen oder durch den Programmierer selbst. Mögliche Pannen können bei Datenbankabfragen auftauchen oder wenn dem Programmierer ein Denkfehler unterläuft. Dann werden sogenannte Exceptions geworfen. Exceptions versuchen Schwierigkeiten abzufangen und zu lokalisieren, indem sie Fehlermeldungen auf dem Bildschirm ausgeben. Sie stehen im Gegensatz zu den bereits kennengelernten Kompilierfehlern, die auf Unrichtigkeiten bei der Syntax beruhen. Vergessen Sie also ein Komma oder eine Klammer, wird keine Exception geworfen, sondern es kommt zu einem Kompilierfehler.

Hier können Sie einen Ausschnitt aus der API (Javadocs) sehen: Die Klasse Exception und die Klasse Error sind Subklassen der Klasse Throwable.

java.lang

Class Throwable

```
java.lang.Object
  └ java.lang.Throwable
```

All Implemented Interfaces:

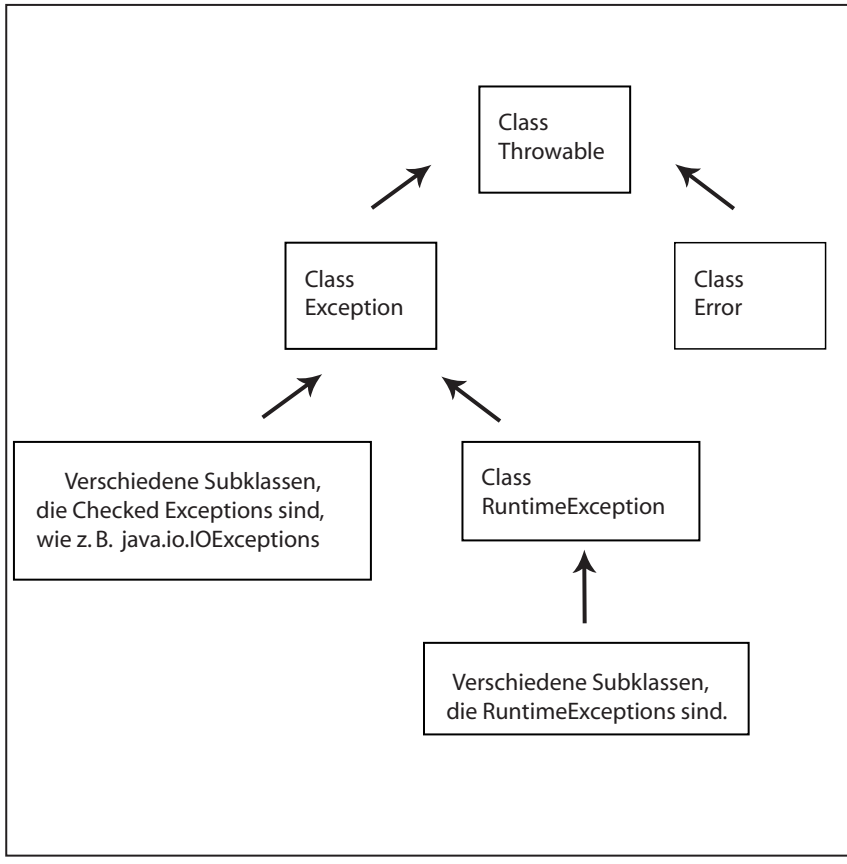
Serializable

Direct Known Subclasses:

Error, Exception

2. Checked und Unchecked Exceptions

Exceptions werden unterteilt in Checked und Unchecked Exceptions: Checked Exceptions sind Probleme, die in der Regel außerhalb des Einflußbereichs des Programmierers liegen, wohingegen die Unchecked Exceptions, die RuntimeExceptions, auf klassischen Programmierfehlern beruhen. Checked bzw. explizite Exceptions, sind die Exceptions, die abgefangen werden müssen, da es Situationen gibt, die absehbar sind und für die man von Anfang an geeignete Maßnahmen planen muß. Im Gegensatz hierzu liegt für unchecked bzw. implizite Exceptions nur eine Empfehlung zum Abfangen vor, da man seine eigenen Fehler schon im Vorfeld vermeiden bzw. suchen und eliminieren sollte. In untenstehender Darstellung können Sie die Verwandtschaftsverhältnisse der verschiedenen Exceptions sehen:

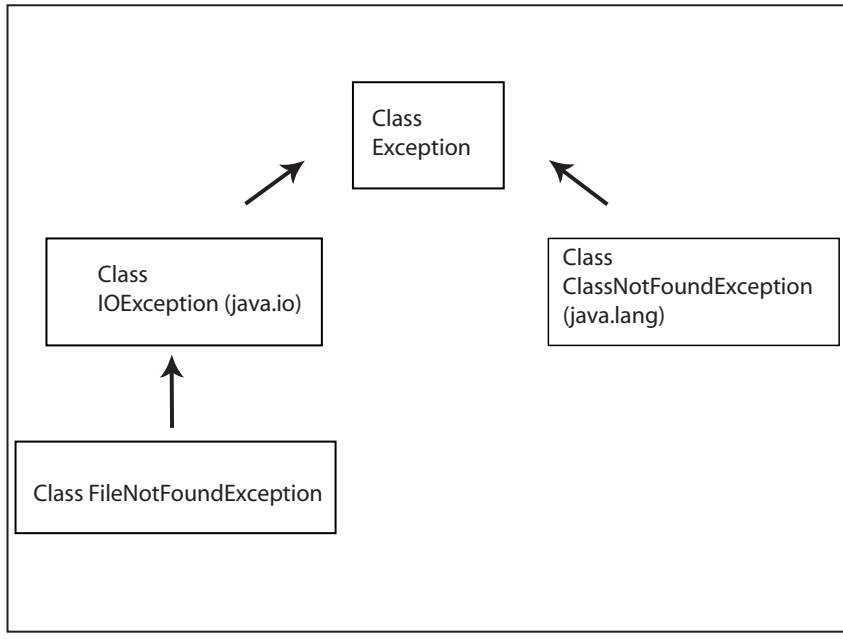


a) Checked Exceptions

Wir sehen also in obenstehender Abbildung: Alle Checked Exceptions sind Subklassen der Klasse Exception. In untenstehender Tabelle stelle ich die wichtigsten Checked Exceptions zusammen:

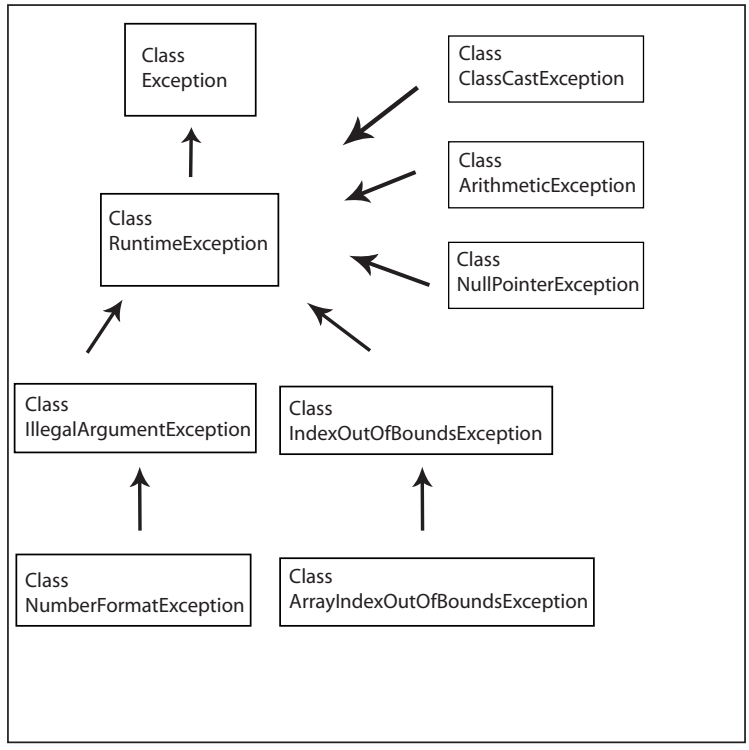
Exception	Beschreibung
ClassNotFoundException	Sie wird geworfen, wenn eine bestimmte Klasse nicht vorhanden ist oder nicht gefunden wird.
IOException	Sie gehört dem Package java.io und ist eine direkte Subklasse der Klasse Exception.
FileNotFoundException	Sie tritt auf, wenn ein Textdokument nicht vorhanden ist oder nicht gefunden werden kann. Sie ist eine direkte Subklasse der Klasse IOException.

Sicherlich denken Sie jetzt: Warum kann es ein Programmierer nicht kontrollieren, ob es eine Datei gibt oder nicht? Ganz einfach: In der Regel erstellen andere Personen zu einem anderen Zeitpunkt diese Dateien!



b) RuntimeExceptions

Wir erinnern uns, RuntimeExceptions beruhen auf Denkfehlern des Programmierers. In den folgenden Abschnitten werde ich einige RuntimeExceptions (Unchecked Exceptions) näher erläutern. Hier als erstes eine Abbildung über die Verwandschaftsverhältnisse der RuntimeExceptions:



(1) ClassCastException

Wir haben bereits eine RuntimeException kennengelernt, und zwar die ClassCastException (vgl. Kapitel Subtyping und Casting von Objekten). Man kann ein Objekt der Vaterklasse nicht in ein Objekt der Kindklasse umwandeln, auch mit einem Cast nicht. Dies merkt der Kompiler nicht zur Kompilierzeit, sondern erst zur Laufzeit und es wird zur Laufzeit eine ClassCastException geworfen. Bei einem Kompilierfehler würde Eclipse dies bereits bei der Eingabe anzeigen, indem es den Code rot unterkringelt, was hier nicht der Fall ist.

```
5  public class Tisch extends Moebel{
6
7      public static void main(String[] args) {
8
9          Moebel moebel = new Moebel();
10         Tisch tisch = new Tisch();
11         Stuhl stuhl = new Stuhl();
12
13         tisch = (Tisch)moebel;
14         stuhl = (Stuhl)moebel;
15     }
16 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Tisch [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (
java.lang.ClassCastException
        at scjp.Tisch.main(Tisch.java:13)
Exception in thread "main"
```

(2) ArithmeticException

Begehen Sie den Fehler in Java eine Zahl durch 0 dividieren zu wollen, kommt es zu einem Problem, da man eine Zahl nicht durch 0 teilen kann und es wird eine ArithmeticException geworfen.

```
5  public class Rechenfehler {
6
7      public static void main(String[] args) {
8          int i = 0;
9          System.out.println(5/i);
10     }
11 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Rechenfehler [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (1
java.lang.ArithmeticException: / by zero
        at scjp.Rechenfehler.main(Rechenfehler.java:9)
Exception in thread "main"
```

(3) ArrayIndexOutOfBoundsException

Diese Exception wird geworfen, wenn Sie auf ein Arrayelement zugreifen wollen, das es nicht gibt. In untenstehendem Beispiel haben wir ein Array mit 2 Elementen erstellt. Nun wollen wir dem dritten Element, das nicht existiert, einen Wert zuweisen, was zu einer ArrayIndexOutOfBoundsException führt (vgl. Kapitel Arrays). Diese ist eine direkte Subklasse, der IndexOutOfBoundsException, die wiederum eine

direkte Subklasse der Klasse RuntimeException ist. Erinnern wir uns, wir können auf das erste Element eines Array mit a[0] und nicht mit a[1] zugreifen. So ist das zweite Element a[1] und nicht a[2], sprich es gibt kein Element a[2].

```
4  public class Arrayfehler {
5
6      public static void main(String[] args) {
7          int[] a = new int[2];
8          a[1]= 2;
9          a[2]= 3;
10     }
11 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Arrayfehler [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (1
java.lang.ArrayIndexOutOfBoundsException: 2
        at scjp.Arrayfehler.main(Arrayfehler.java:10)
Exception in thread "main"
```

3. Exception Handling

a) try- und catch-Blöcke

Nun bleibt die Frage: Wie kann ich einen Programmabbruch bei einer Exception verhindern? Die Antwort ist einfach: Wir brauchen einen try-catch-Block. Lassen Sie mich dies anhand der ArithmeticException zeigen: Wir erinnern uns, daß diese Exception geworfen wurde, als wir eine Zahl durch 0 dividieren wollten, was ja nicht geht. Wir schreiben jetzt unsere Berechnung in einen try-catch-Block. Wir „versuchen" (try) also unsere Berechnung durchzuführen und „fangen" (catch) den Fehler wieder auf.

```
public class Rechenfehler {
        public static void main(String[ ] args) {
                //Versuch eine Berechnung durchzuführen
                try{
                int i = 0
                System.out.println(5/i);
                //Auffangen des Fehlers durch Ausgabe einer Fehlermeldung
                }catch(ArithmeticException rechenFehler){
                        System.out.println("Dies ist eine ArithmeticException");
                }
        }
}
```

Ausgabe auf der Konsole:

Dies ist eine ArithmeticException

Es besteht die Möglichkeit mehrere catch-Blöcke hintereinander einzufügen, wobei allerdings auf die Klassenhierarchie geachtet werden muß. Es macht keinen Sinn ganz oben eine Exception abzufangen, da diese alle Exceptions abfängt. So würde bereits im ersten catch-Block die ArithmeticExceptions abgefangen werden. Somit wären Sie nicht in der Lage, zwischen den verschiedenen Exceptions zu unterscheiden. Dies wollen wir nicht. Deshalb müssen wir die Klassenhierarchie von unten nach oben abarbeiten. So erhalten wir die Möglichkeit, die entsprechende Exception abzufangen und in unserem untenstehenden Beispiel wird die korrekte Meldung ausgeben, daß es sich um eine ArithmeticException handelt. Sollte allerdings in den catch-Blöcken nicht die Exception abgefangen werden, die im try-Block geworfen wird, wird zur Laufzeit wieder eine Exception geworfen.

```
public class Rechenfehler {
        public static void main(String[ ] args) {
                try{
                int i = 0;
                System.out.println(5/i);
                }catch(ArithmeticException rechenFehler){
                        System.out.println(”Dies ist eine ArithmeticException“);
                }catch (Exception fehlerException){
                        System.out.println(“Dies ist eine Exception”);
                }
        }
}
```

Ausgabe auf der Konsole:

Dies ist eine ArithmeticException

b) Der finally-Block

Zusätzlich zu dem try- und catch-Blöcken kann noch ein finally-Block stehen, der auf jeden Fall ausgeführt werden muß. Klassischer Weise handelt es sich hierbei um Programmcode, der reservierte Ressourcen freigibt, indem er geöffnete Dateien oder Programme schließt. Finally-Blöcke haben nicht die Aufgabe Exceptions abzufangen. Es ist zu beachten, daß ein finally-Block nicht von einem catch-Block gefolgt werden darf, da es andernfalls zu einem Kompilierfehler kommt.

```
public class Rechenfehler {
        public static void main(String[ ] args) {
                try{
                int i = 0;
                System.out.println(5/i);
                }catch(ArithmeticException rechenFehler){
                        System.out.println(”Dies ist eine ArithmeticException“);
                }finally {
                        System.out.println(“Ich werde auf jeden Fall ausgeführt”);
                }
        }
}
```

Ausgabe auf der Konsole:

```
Dies ist eine ArithmeticException
Ich werde auf jeden Fall ausgeführt
```

Es darf zwar ein finally-Block mit einem try-Block alleine dastehen, sprich es kommt zu keinem Kompilierfehler, aber zur Laufzeit wird trotzdem eine Exception geworfen und das Programm wird abgebrochen, da der finally-Block keine Exception auffangen kann und dennoch in jedem Fall ausgeführt wird. Es wird also sowohl der try- und finally-Block abgearbeitet als auch zur Laufzeit eine Exception geworfen. In unserem Beispiel handelt es sich um eine explizite Exception, die wir im nächsten Abschnitt näher betrachten wollen (siehe Kapitel Der Throw-Ausdruck).

```
public class TryFinally {
        public static void main(String[ ] args) {
                try{
                        System.out.println(”try“);
                        throw new RuntimeException();
                }finally{
                        System.out.println(”finally“);
                }
        }
}
```

Ausgabe auf der Konsole:

```
try
finally
Exception in thread „main“ java.lang.RuntimeException
        at scjp.TryFinally.main(TryFinally.java:10)
```

Steht ein try- mit einem finally-Block alleine da, wird das Programm nicht abgebrochen, wenn, wie in untenstehendem Beispiel, im try-Block keine Exception geworfen wird. Es wird sowohl der try-Block als auch der finally-Block problemlos abgearbeitet.

```
public class TryFinally {
        public static void main(String[ ] args) {
                try{
                        System.out.println(”try“);
                }finally{
                        System.out.println(”finally“);
                }
        }
}
```

Ausgabe auf der Konsole:

```
try
finally
```

4. Der throw-Ausdruck

Die bisher kennengelernten Exceptions waren implizit, sprich sie wurden erst während der Laufzeit entdeckt. Nun kann man aber auch absichtlich Exceptions werfen, explizite Exceptions. Dies macht man mit dem throw-Ausdruck (Achtung: nicht throws mit s!). Sollte allerdings in den catch-Blöcken nicht die explizite Exception abgefangen werden, die im try-Block geworfen wird, kommt es zu einem Kompilierfehler, da dies der Kompiler sofort bemerkt.

```
public class ExpliziteException {
        public static void main(String[ ] args) {
                try{
                   throw new ArithmeticException();
                }catch (ArithmeticException rechenfehler){
                   System.out.println("Ich bin ein expliziter Rechenfehler!");
                }
        }
}
```

Ausgabe auf der Konsole:

Ich bin ein expliziter Rechenfehler!

Hinter dem Ausdruck throw new ArithmeticException(); muß direkt ein catch-Block stehen. Wird ein weiterer try-Block oder sonstiger Code eingefügt, führt dies zu einem Kompilierfehler.

```
4  public class ExpliziteException {
5
6      public static void main(String[] args) {
7
8          try{
9              throw new ArithmeticException();
10             try{
11                 throw new Exception();
12
13             }catch(System.out.println("1");)
14         }
15         catch (ArithmeticException rechenfehler){
16             System.out.println("Ich bin ein expliziter Rechenfehler!");
17         }
18     }
19 }
```

Befindet sich unterhalb des try-, catch und finally-Blocks noch Code, wird dieser nach Auffangen der Exception ausgeführt.

```
public class ExpliziteException {
```

```
        public static void main(String[ ] args) {
                try{
                        throw new ArithmeticException();
                }catch (ArithmeticException rechenfehler){
                        System.out.println("A ");
                }catch (RuntimeException fehler){
                        System.out.println("B ");
                }finally{
                        System.out.println("C ");
                }
                System.out.println("D ");
        }
}
```

Ausgabe auf der Konsole:

A
C
D

Wird allerdings in dem ersten catch-Block erneut eine Exception geworfen, kann diese nicht mehr aufgefangen werden. So wird zwar die erste Exception aufgefangen, aber die zweite nicht mehr, und es tritt ein Laufzeitfehler auf. Der finally-Block wird noch ausgeführt, aber dann wird das Programm abgebrochen, so ist die Ausgabe nur A und C. Also bitte beachten Sie, daß eine Exception nur in einem try-Block geworfen werden kann.

```
4  public class ExpliziteException {
5      public static void main(String[] args) {
6          try{
7              throw new ArithmeticException();
8          }
9          catch (ArithmeticException rechenfehler){
10             System.out.println("A ");
11             throw new RuntimeException();
12         }
13         catch (RuntimeException fehler){
14             System.out.println("B ");
15         }
16         finally{
17             System.out.println("C ");
18         }
19         System.out.println("D ");
20     }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> ExpliziteException [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (17.05.2005 15
A java.lang.RuntimeException
        at scjp.ExpliziteException.main(ExpliziteException.java:11)

C
Exception in thread "main"
```

5. Die throws-Klausel

Will man eine sogenannte Checked Exception werfen, muß diese oder eine Exception der Vaterklasse im Methodenkopf stehen oder es kommt zu einem Kompilierfehler. Dies trifft nicht auf die main() zu, dort muß die Checked Exception nicht im Methodenkopf stehen, aber sie darf dort stehen. Im untenstehenden Beispiel wird eine FileNotFoundException geworfen, die sich im java.io-Paket befindet und nicht im java.lang-Paket, deshalb muß diese Klasse zusätzlich importiert werden. Um Kompilierfehler zu vermeiden, gilt es zweierlei zu beachten: Im Methodenkopf muß throws mit s stehen und nicht throw, und in der Methode selbst muß throw ohne s stehen und nicht throws.

```
import java.io.FileNotFoundException;

public class ThrowsKlausel {
	public static void main(String[ ] args) {
		try{
			ThrowsKlausel neueAusnahme = new ThrowsKlausel();
			neueAusnahme.ausnahme();
		}catch(Exception fehler){
			System.out.println("Ich fange!");
		}
	}

	public void ausnahme()throws FileNotFoundException{
		throw new FileNotFoundException();
	}
}
```

Ausgabe auf der Konsole:

```
Ich fange!
```

6. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
	public static void main(String[ ] args) {
		try{
			throw new ArithmeticException();
		}
		catch (ArithmeticException rechenfehler){
			System.out.println(”Rechenfehler“);
		}
		catch (Exception fehler){
			System.out.println(“Fehler”);
		}
		finally{
			System.out.println(”finally“);
		}
		System.out.println(“Ende”);
	}
}
```

a) Rechenfehler Fehler finally
b) Rechenfehler Fehler finally Ende
c) Rechenfehler finally Ende
d) Fehler finally Ende
e) finally Ende
f) Kompilierfehler
g) Laufzeitfehler

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
	public static void main(String[ ] args) {
		try{
			throw new ClassCastException();
		}
		catch (ArithmeticException rechenfehler){
			System.out.println(”Rechenfehler“);
		}
		catch (Exception fehler){
			System.out.println(“Fehler”);
		}
		finally{
			System.out.println(”finally“);
		}
		System.out.println(“Ende”);
	}}
```

a) Rechenfehler Fehler finally
b) Rechenfehler Fehler finally Ende
c) Rechenfehler finally Ende
d) Fehler finally Ende
e) finally Ende
f) Kompilierfehler
g) Laufzeitfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            throw new ArithmeticException();
        }
        catch (ArithmeticException rechenfehler){
            System.out.println(”Rechenfehler ”);
        }
        finally{
            System.out.println(”finally“);
        }
        catch (Exception fehler){
            System.out.println(“Fehler”);
        }
        System.out.println(“Ende”);
    }
}
```

a) Rechenfehler Fehler finally
b) Rechenfehler Fehler finally Ende
c) Rechenfehler finally Ende
d) Fehler finally Ende
e) finally Ende
f) Kompilierfehler
g) Laufzeitfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            throw new ClassCastException();
        }
        catch (ArithmeticException rechenfehler){
            System.out.println(”Rechenfehler ”);
        }
```

```
            catch (InterruptedException fehler){
                System.out.println("Fehler");
            }
            finally{
                System.out.println("finally");
            }
            System.out.println("Ende");
        }
}
```

a) Rechenfehler Fehler finally
b) Rechenfehler Fehler finally Ende
c) Rechenfehler finally Ende
d) Fehler finally Ende
e) finally Ende
f) Kompilierfehler
g) Laufzeitfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            throw new ClassCastException();
        }
        catch (ClassCastException fehler){
            System.out.println("Fehler");
            throw new ArithmeticException();
        }
        catch (ArithmeticException rechenfehler){
            System.out.println("Rechenfehler");
        }
        finally{
            System.out.println("finally");
        }
        System.out.println("Ende");
    }
}
```

a) Rechenfehler Fehler finally
b) Rechenfehler Fehler finally Ende
c) Rechenfehler finally Ende
d) Fehler finally
e) finally Ende
f) Kompilierfehler
g) Laufzeitfehler

(6) Frage

In unstenstehendem Code wird eine Exception abgefangen, um welche handelt

es sich?

```
public class Fehler {
	public static void main(String[ ] args) {
		try
		{int[ ] a = new int[5];
		a[5]= 2;
		}catch (Exception fehler){
			System.out.println("Fehler");
		}
	}
}
```

a) Error
b) RuntimeException
c) InterruptedException
d) ClassCastException
e) NullPointerException
f) ArrayIndexOutOfBoundsException

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
	public static void main(String[ ] args) {
		try{
			System.out.println("try");
		}finally{
			System.out.println("finally");
		}
	}
}
```

a) try finally
b) finally try
c) finally
d) Laufzeitfehler
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
	public static void main(String[ ] args) {
		try{
			System.out.println("try");
		}finally{
			System.out.println("finally");
```

```
            }
            System.out.println("Ende");
        }
}
```

a) try finally Ende
b) finally try Ende
c) finally Ende
d) Laufzeitfehler
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            System.out.println("try");
            throw new RuntimeException();
        }finally{
            System.out.println("finally");
        }
    }
}
```

a) try finally
b) finally try
c) finally
d) Laufzeitfehler
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            throw new ClassCastException();
            System.out.println("werfen");
        }
        catch (ArithmeticException rechenfehler){
            System.out.println("fangen");
        }
        catch (ClassCastException fehler){
            System.out.println("nochmals fangen");
        }
        finally{
```

```
                System.out.println("Restarbeit");
            }
            System.out.println("Ende");
        }
}
```

a) werfen nochmals fangen Restarbeit Ende
b) nochmals fangen Restarbeit Ende
c) fangen nochmals fangen Restarbeit Ende
d) Kompilierfehler
e) Laufzeitfehler
f) Keine dieser Möglichkeiten.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            throw new ClassCastException();
        }
        catch (ArithmeticException rechenfehler){
            System.out.println("fangen");
        }
        catch (ClassCastException fehler){
            System.out.println("nochmals fangen");
        }
        finally{
            System.out.println("Restarbeit");
        }
        System.out.println("Ende");
    }
}
```

a) werfen nochmals fangen Restarbeit Ende
b) nochmals fangen Restarbeit Ende
c) fangen nochmals fangen Restarbeit Ende
d) Kompilierfehler
e) Laufzeitfehler
f) Keine dieser Möglichkeiten.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.FileNotFoundException;

public class Fehler {
    public static void main(String[ ] args) throws FileNotFoundException {
        try{
```

```
            throw new FileNotFoundException();
        }
        catch (FileNotFoundException dokumentfehler){
            System.out.println("fangen");
        }
        catch (ClassCastException fehler){
            System.out.println("nochmals fangen");
        }
        finally{
            System.out.println("Restarbeit");
        }
        System.out.println("Ende");
    }
}
```

a) werfen nochmals fangen Restarbeit Ende
b) nochmals fangen Restarbeit Ende
c) fangen Restarbeit Ende
d) Kompilierfehler
e) Laufzeitfehler
f) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.FileNotFoundException;

public class Fehler {
    public static void main(String[ ] args) throws FileNotFoundException {
        try{
            throw new FileNotFoundException();
        }
        catch (ArithmeticException dokumentfehler){
            System.out.println("fangen");
        }
        catch (ClassCastException fehler){
            System.out.println("nochmals fangen");
        }
        finally{
            System.out.println("Restarbeit");
        }
        System.out.println("Ende");
    }
}
```

a) werfen nochmals fangen Restarbeit Ende
b) nochmals fangen Restarbeit Ende
c) fangen Restarbeit Ende
d) Kompilierfehler
e) Laufzeitfehler
f) Keine dieser Möglichkeiten.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.FileNotFoundException;

public class Fehler {
	public static void main(String[ ] args) throw FileNotFoundException {
		try{
			throw new FileNotFoundException();
		}
		catch (FileNotFoundException dokumentfehler){
			System.out.println("fangen");
		}
		catch (ClassCastException fehler){
			System.out.println("nochmals fangen");
		}
		finally{
			System.out.println("Restarbeit");
		}
		System.out.println("Ende");
	}
}
```

a) werfen nochmals fangen Restarbeit Ende
b) nochmals fangen Restarbeit Ende
c) fangen Restarbeit Ende
d) Kompilierfehler
e) Laufzeitfehler
f) Keine dieser Möglichkeiten.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
	public static void main(String[ ] args) {
		try{
			Fehler fehler = new Fehler ();
			fehler.ausnahme();
		}catch(FileNotFoundException fehler){
			System.out.println("Ich fange!");
		}
	}

	public void ausnahme() throws FileNotFoundException {
		throw new FileNotFoundException();
	}
}
```

a) Ich fange!
b) Kompilierfehler

c) Laufzeitfehler
d) Keine dieser Möglichkeiten

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {

    public static void main(String[ ] args) {
        try{
            Fehler fehler = new Fehler ();
            fehler.ausnahme();
        }catch(FileNotFoundException fehler){
            System.out.println("Ich fange!");
        }
    }

    public void ausnahme(){
        throw new FileNotFoundException();
    }
}
```

a) Ich fange!
b) Kompilierfehler
c) Laufzeitfehler
d) Keine dieser Möglichkeiten

(17) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        try{
            Fehler fehler = new Fehler ();
            fehler.ausnahme();
        }catch(ClassCastException fehler){
            System.out.println("Ich fange!");
        }
    }

    public void ausnahme() throws FileNotFoundException {
        throw new FileNotFoundException();
    }
}
```

a) Ich fange!

b) Kompilierfehler
c) Laufzeitfehler
d) Keine dieser Möglichkeiten

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        FileNotFoundException fehler = new FileNotFoundException();
        boolean a = fehler instanceof Throwable;
        boolean b = fehler instanceof Exception;
        System.out.println(a);
        System.out.println(b);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
    public static void main(String[ ] args) {
        FileNotFoundException fehler = new FileNotFoundException();
        boolean a = fehler instanceof Throwable;
        boolean b = fehler instanceof Error;
        System.out.println(a);
        System.out.println(b);}}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Fehler {
```

```
    public static void main(String[ ] args) {
        FileNotFoundException fehler = new FileNotFoundException();
        Throwable throwable = new Throwable();
        boolean a = throwable instanceof FileNotFoundException;
        boolean b = fehler instanceof Throwable;
        System.out.println(a);
        System.out.println(b);
    }
}
```

a) true true
b) true false
c) false true
d) false false
e) Kompilierfehler
f) Keine dieser Möglichkeiten.

(21) Frage

Welche Klasse, der untenstehenden Klassen, ist die direkte Superklasse von FileNotFoundException?

a) Exception
b) Error
c) Throwable
d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException

(22) Frage

Welche Klasse, der untenstehenden Klassen, ist die direkte Superklasse von Exception?

a) Exception
b) Error
c) Throwable
d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException

(23) Frage

Welche Klasse, der untenstehenden Klassen, ist die direkte Superklasse von NumberFormatException?

a) Exception
b) Error
c) Throwable

d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException

(24) Frage

Welche Klasse, der untenstehenden Klassen, ist die direkte Superklasse von ClassCastException?

a) Exception
b) Error
c) Throwable
d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException

(25) Frage

Welche Klasse, der untenstehenden Klassen, ist die direkte Superklasse von NullPointerException?

a) Exception
b) Error
c) Throwable
d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException

7. Lösungen

(1) Frage

c

(2) Frage

d, Exception ist die Superklasse der ClassCastException, also wird diese vom zweiten catch-Block aufgefangen.

(3) Frage

f, Nach einem finally-Block darf kein catch-Block mehr stehen.

(4) Frage

f, Die ClassCastException wird nicht aufgefangen, deshalb kommt es zu einem Kompilierfehler.

(5) Frage

d, g, In einem catch-Block darf keine Exception geworfen werden. Die erste Exception wird aufgefangen, die zweite nicht und dann wird auf jeden Fall noch der finally-Block ausgeführt.

(6) Frage

f

(7) Frage

a

(8) Frage

a

(9) Frage

a, d, Sowohl der try- als auch der finally-Block ausgeführt, es existiert allerdings kein catch-Block, der die geworfene RuntimeException auffangen würde.

(10) Frage

d, Nach der geworfenen Exception muß sofort ein catch-Block stehen. Es darf kein anderer Code dazwischen stehen.

(11) Frage

b

(12) Frage

c

(13) Frage

d, Die Exception, die geworfen wird, wird nicht aufgefangen.

(14) Frage

d, Im Methodenkopf steht throw statt throws.

(15) Frage

a

(16) Frage

b, Im Methodenkopf fehlt die throws-Klausel.

(17) Frage

b, Die FileNotFoundException, die geworfen wird, wird nicht aufgefangen.

(18) Frage

a

(19) Frage

d, Die Klasse Error ist zwar eine Subklasse von Throwable, aber keine Superklasse von Exception, deswegen kommt es zu einem Kompilierfehler.

(20) Frage

c

(21) Frage

d

(22) Frage

c

(23) Frage

g

(24) Frage

e

(25) Frage

e

X. Assertions

1. Begriffserläuterung

Jetzt kommt die nächste Frage: Wie verhindern wir Programmierfehler? Vorgesehen hierfür sind Assertions. Assertions sind ein Werkzeug zum Debuggen von Programmcode. Assertions dienen dazu, sicherzustellen, daß Methoden einer Klasse korrekt und ohne Fehler aufgerufen werden. Wie geschieht dies? Es werden Bedingungen als boolean-Ausdruck formuliert, die erfüllt werden müssen oder zum Abbruch des Programms führen. Assertions werden nur aktiviert solange ein Programm entwickelt und erprobt wird, im Einsatz vor Ort beim Kunden werden sie deaktiviert. Sie ersetzen das bisher übliche Vorgehen mit System.out.print()-Befehlen, möglichen Bugs auf die Spur zu kommen. System.out.print()-Befehle mußte man nach der Debugging-Phase wieder löschen. Dies ist jetzt nicht mehr notwendig, da die Ausgabebefehle durch assert-Statements ersetzt und je nach Bedarf an- oder ausgeschaltet werden.

Beginnen wir wieder mit einem Beispiel: Es soll ein negativer Preis verhindert werden. Aber: Die Bedingung trifft nicht zu, so wird ein AssertionError geworfen. Es wird allerdings trotzdem der Satz „Ich bin ein Stuhl!“ ausgegeben. Wäre der Preis eine positive Zahl, würde das Programm normal weiter laufen. Die Klasse AssertionError ist eine direkte Subklasse von Error, die wiederum eine Subklasse von Throwable ist, genauso wie die Klasse Exception (vgl. Kapitel Exceptions).

```
 4 public class Stuhl{
 5
 6     double preis;
 7
 8     public static void main(String[] args){
 9         Stuhl stuhl = new Stuhl();
10         stuhl.stehen(-8.0);
11     }
12
13     private void stehen(double preis) {
14         System.out.println("Ich bin ein Stuhl!");
15         assert preis > 0.0;
16     }
17 }
```

```
Problems | Javadoc | Declaration | Console
<terminated> Stuhl [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\ja
java.lang.AssertionError
Ich bin ein Stuhl!
        at scjp.Stuhl.stehen(Stuhl.java:15)
        at scjp.Stuhl.main(Stuhl.java:10)
Exception in thread "main"
```

Sind die Assertions nicht aktiviert, läuft das Programm, wie Sie unten sehen können, normal weiter. Nun ist es dem Programm egal, ob der Preis positiv oder negativ ist, da der assert-Ausdruck ignoriert wird. Weiter unten lernen wir, wie Assertions

an- und ausgeschaltet werden können.

```
public class Stuhl{
        double preis;

        public static void main(String[ ] args){
                Stuhl stuhl = new Stuhl();
                stuhl.stehen(-8.0);
        }

        private void stehen(double preis) {
                System.out.println("Ich bin ein Stuhl!");
                assert preis > 0.0;
        }
}
```

Ausgabe auf der Konsole:

Ich bin ein Stuhl!

Es gibt zwei Arten Assertions: Die erste Form besteht aus einem assert-Statement und einer Bedingung. Die Bedingung ist ein primitiver boolean-Ausdruck, wie z. B. assert preis > 0;. Dem assert-Statement darf nichts anderes als ein boolean-Ausdruck folgen, sprich, achten Sie insbesondere darauf, daß dort keine Zuweisung steht, da dies sonst, wie Sie in untenstehender Abbildung sehen können, zu einem Kompilierfehler führt. Dies ist ein Syntaxfehler, der bereits zur Kompilierzeit bemerkt wird und zur Laufzeit einen Error wirft. Hierbei spielt keine Rolle, ob die Assertions aktiviert oder deaktiviert sind.

```
 4 public class Stuhl{
 5
 6     double preis;
 7
 8     public static void main(String[] args){
 9         Stuhl stuhl = new Stuhl();
10         stuhl.stehen(8.0);
11     }
12
13     private void stehen(double preis) {
14         System.out.println("Ich bin ein Stuhl!");
15         assert preis = 0.0;
16     }
17 }
```

Problems | Javadoc | Declaration | Console | Error Log

```
<terminated> Stuhl [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (25.05.2005 15:1
java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from double to boolean

        at scjp.Stuhl.stehen(Stuhl.java:15)
        at scjp.Stuhl.main(Stuhl.java:10)
Exception in thread "main"
```

Die zweite Form des assert-Statements enthält zusätzlich einen Doppelpunkt und nach dem Doppelpunkt, z. B. eine Fehlermeldung für den Fall, daß der assert-Ausdruck false wird. So könnte dies wie folgt aussehen: assert preis > 0: „Der Preis muß einen positiven Wert haben.“;. Nach dem Doppelpunkt darf folgendes stehen: eine primitive Variable, eine Methode, ein Objekt und ein Stringobjekt. Aber folgendes nicht: eine Methode, die nichts zurückgibt (void), return und break. Wie Sie in untenstehendem Beispiel sehen können, kommt es andernfalls zu einem Kompilierfehler und es wird zur Laufzeit ein Error geworfen, da es sich um einen Syntaxfehler handelt und nicht um eine bloße Zuwiderhandlung einer unverbindlichen Regel (siehe unten). Hierbei spielt es keine Rolle, ob die Assertions aktiviert oder deaktiviert sind.

```
4  public class Stuhl{
5      double preis;
6      public static void main(String[] args){
7          Stuhl stuhl = new Stuhl();
8          stuhl.stehen(8.0);
9      }
10     public void schreiben(){
11         System.out.println("b");
12     }
13     private void stehen(double preis) {
14         System.out.println("Ich bin ein Stuhl!");
15         assert preis > 0.0: schreiben();
16     }
17 }
```

```
Problems | Javadoc | Declaration | Console
<terminated> Stuhl [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\
java.lang.Error: Unresolved compilation problem:
        Expression must return a value

        at scjp.Stuhl.stehen(Stuhl.java:17)
        at scjp.Stuhl.main(Stuhl.java:10)
Exception in thread "main"
```

2. Kompilieren von Klassen mit Assertions

Assertions wurden mit dem J2SE 1.4 eingeführt, sprich vorher gab es keine Assertions. Wird eine Klasse mit javac -source 1.3 Vater.java, also einer alten Javaversion, kompiliert, kennt der Kompiler keine Assertions und es wird die Fehlermeldung ausgegeben, assert sei ein Schlüsselwort. Wird die Klasse allerdings mit javac -source 1.4 kompiliert, werden die Assertions als Assertions erkannt und bearbeitet.

3. Ein- und Ausschalten von Assertions für Nicht-Systemdateien

Der Befehl der Kommandozeile -ea oder die Langform -enableassertions aktiviert die Assertions für alle Nicht-Systemdateien und der Befehl -da oder die Langform -disableassertions deaktiviert sie wieder. Es ist denkbar die Aktivierung oder Deaktivierung auf ein Package oder eine Klasse zu beschränken, dann steht dort nach einem Doppelpunkt der Packagename oder die entsprechende Klasse(-ea: buch.scjp oder -ea:Vater, -da:buch.scjp oder -da:Vater). Wollen Sie sich auf das Package, in dem Sie gerade arbeiten, ohne explizite Namensnennung beziehen, müssen sie nach dem Doppelpunkt drei Punkte einfügen (-ea:... oder -da:...).

a) Exkurs: Javakommandozeile in Eclipse

Es besteht auch in Eclipse die Möglichkeit, Befehle in die Kommandozeile einzugeben und dies geht wie folgt: Wählen Sie im Menü, den Menüpunkt Run aus und dann wieder Run. So gelangen Sie in untenstehendes Fenster, wo Sie im Register Arguments in dem Feld VM arguments den Befehl –ea eingeben können.

4. Ein- und Ausschalten von Assertions für Systemdateien.

Der Befehl die Assertions für die Systemdateien zur Laufzeit zu aktivieren, lautet: -enablesystemassertions und in Kurzform -esa. Man kann Sie wieder deaktivieren

mit folgendem Befehl: -disablesystemassertions und der dazugehörigen Kurzform -das.

5. Regeln für die Verwendung von Assertions

Es ist sicherzustellen, daß Assertions keinerlei unerwünschte Nebeneffekte haben und den Programmablauf nicht verändern. Deshalb gilt es einige Grundregeln für die Verwendung von Assertions zu beachten, die im weiteren näher ausgeführt werden. Eine Zuwiderhandlung dieser Regeln führt weder zu einem Kompilierfehler noch zu einem AssertionError, da dies keine Syntaxregeln sind, sondern Richtlinien für gutes und erwünschtes Programmieren.

a) Empfehlung Assertions nicht in Methoden zu verwenden, die public sind.

In Methoden, die public sind, dürfen keine Assertions stehen, da auf diese Methoden von überall her zugegriffen werden kann. Somit ist jede Kontrolle über eine eventuelle Veränderung von außen unmöglich. Wird diese Anweisung ignoriert, kommt es zu keinem Kompilierfehler, da dies nur eine Richtlinie ist. Es kommt allerdings trotzdem zu einem AssertionError, wenn der Preis negativ ist, da dies ja mit dem Befehl assert sichergestellt werden soll.

```
public String stehen(double preis) {
                    System.out.println("Ich bin ein Stuhl!");
                    assert preis > 0.0d;
          }
```

b) Empfehlung Assertions nicht für Werte zu verwenden, die in der Kommandozeile eingegeben werden.

Da man hier wie bei Methoden, die öffentlich sind, keinerlei Kontrolle darüber hat, was eingegeben wird.

c) Empfehlung der Verwendung von Assertions in Methoden, die private sind.

Normalerweise ist ein kontrollierter Zugriff von außen bei privaten Methoden gewährleistet.

```
private String stehen(double preis) {
                    System.out.println("Ich bin ein Stuhl!");
                    assert preis > 0.0d;
          }
```

d) Empfehlung der Verwendung von Assertions, sogar in Methoden die public sind, in Fällen, in denen davon ausgegangen werden kann, daß sie niemals eintreten.

Ein Beispiel hierfür ist der default-Fall einer switch-Struktur, da gemeinhin davon ausgegangen wird, daß dieser Fall nicht eintritt.

```
switch (j){
        case 1: System.out.println(”1“); break;
        case 2: System.out.println(”2“); break;
        default: assert x > 0;
}
```

e) Unerwünschte Nebeneffekte sollen vermieden werden

Das assert-Statement soll den weiteren Programmablauf nicht verändern, deshalb soll nach dem Doppelpunkt keine Zuweisung stehen. In untenstehendem Programm könnte preis++ zu einer Erhöhung des Preises führen, falls der Preis negativ ist und die Assertions aktiviert sind. Dies wäre ein nicht erwünschter Nebeneffekt.

```
private String stehen(double preis) {
                System.out.println(“Ich bin ein Stuhl!“);
                assert preis > 0.0d: preis++;
        }
```

6. Übungen

(1) Frage

Welche Klasse der untenstehenden Klassen ist eine direkte Superklasse von AssertionError?

a) Exception
b) Error
c) Throwable
d) IOException
e) RuntimeException
f) IndexOutOfBoundsException
g) IllegalArgumentException
h) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Nicht-Systemdateien zu aktivieren?

a) -ea
b) -da
c) -esa
d) -dsa
e) -eda
f) -dea
g) Keine dieser Möglichkeiten.

(3) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Systemdateien zu aktivieren?

a) -ea
b) -da
c) -esa
d) -dsa
e) -eda
f) -dea
g) Keine dieser Möglichkeiten.

(4) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Nicht-Systemdateien zu aktivieren?

a) -enableassertion
b) -disableassertions
c) -enablesystemassertions
d) -disablesystemassertions

e) -enablesystemassertion
f) -esa
g) Keine dieser Möglichkeiten.

(5) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Systemdateien zu aktivieren?

a) -enableassertion
b) -disableassertions
c) -enablesystemassertions
d) -disablesystemassertions
e) -enablesystemassertion
f) -esa
g) Keine dieser Möglichkeiten.

(6) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Nicht-Systemdateien zu aktivieren?

a) -enableassertions
b) -dsa
c) -enablesystemassertions
d) -ea
e) -enablesystemassertion
f) -esa
g) Keine dieser Möglichkeiten.

(7) Frage

Welche der untenstehenden Befehle dienen dazu, Assertions für Systemdateien zu aktivieren?

a) -enableassertions
b) -dsa
c) -enablesystemassertions
d) -ea
e) -enablesystemassertion
f) -esa
g) Keine dieser Möglichkeiten.

(8) Frage

Was passiert, wenn der boolean-Ausdruck einer Assertion false wird?

a) Es wird eine Exception geworfen.
b) Es wird ein AssertionError geworfen.
c) Es wird eine AssertionException geworfen.
d) Es wird ein Error geworfen.

e) Keine dieser Möglichkeiten.

(9) Frage

Welche der untenstehenden assert-Statements sind unerwünscht, da sie schlechten Programmierstil darstellen?

a) assert x < 1: “Ich bin nicht erwünscht”;
b) assert x < 1;
c) assert x < 1: x;
d) assert x >20: “Ich bin zu klein”;
e) Keine dieser Möglichkeiten.

(10) Frage

Welche der untenstehenden assert-Statements sind unerwünscht, da sie schlechten Programmierstil darstellen?

a) assert x >= 20: “falsch”;
b) assert x <= 20;
c) assert x <= 20: x +1;
d) assert x > 4: “falsch”;
e) Keine dieser Möglichkeiten.

(11) Frage

Welche der untenstehenden assert-Statements führt zu einem Kompilierfehler?

a) assert x < 0: “Fehler”;
b) assert x == 1;
c) assert x == 1: x;
d) assert x >20: “Fehler”;
e) assert i == 0: i++;
f) Keine dieser Möglichkeiten.

(12) Frage

Welche der untenstehenden assert-Statements führt zu einem Kompilierfehler?

a) assert x < 0: “Fehler”;
b) assert x == 1;
c) assert x = 1: x;
d) assert x > 20: “Fehler”;
e) assert i == 0: i++;
f) Keine dieser Möglichkeiten.

(13) Frage

Welche der untenstehenden assert-Statements führt zu einem Kompilierfehler?

a) assert x <= 0: “Fehler”;

b) assert i == 8;
c) assert i++: i;
d) assert x > 20: “Fehler”;
e) assert i == 0: i++;
f) Keine dieser Möglichkeiten.

(14) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(8.0);
        }
        public void mb(){
                System.out.println(“Ich bin ein Fehler”);
        }
        private void mc(int x) {
                System.out.println(“Ich bin Methode mc”);
                assert preis == 0.0;
        }
}
```

a) Ich bin ein Fehler
b) Ich bin Methode mc
c) Kompilierfehler
d) AssertionError
e) Keine dieser Möglichkeiten

(15) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(0.0);
        }
        public void mb(){
                System.out.println(“Ich bin ein Fehler”);
        }
        private void mc(int x) {
                System.out.println(“Ich bin Methode mc”);
                assert preis == 0.0;
        }
}
```

a) Ich bin ein Fehler

b) Ich bin Methode mc
c) Kompilierfehler
d) AssertionError
e) Keine dieser Möglichkeiten

(16) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse nicht aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(8.0);
        }
        public void mb(){
                System.out.println("Ich bin ein Fehler");
        }
        private void mc(int x) {
                System.out.println("Ich bin Methode mc");
                assert preis == 0.0;
        }
}
```

a) Ich bin ein Fehler
b) Ich bin Methode mc
c) Kompilierfehler
d) AssertionError
e) Keine dieser Möglichkeiten

(17) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(0.0);
        }
        public void mb(){
                System.out.println("Ich bin ein Fehler");
        }
        private void mc(int x) {
                System.out.println("Ich bin Methode mc");
                assert preis = 0.0;
        }
}
```

a) Ich bin ein Fehler
b) Ich bin Methode mc

c) Kompilierfehler
d) AssertionError
e) Keine dieser Möglichkeiten

(18) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(0.0);
        }
        public void mb(){
                System.out.println("Ich bin ein Fehler");
        }
        private void mc(int x) {
                System.out.println("Ich bin Methode mc");
                assert preis < 0.0;
        }
}
```

a) Ich bin ein Fehler
b) Ich bin Methode mc
c) Kompilierfehler
d) AssertionError
e) Keine dieser Möglichkeiten

(19) Frage

Was wird bei untenstehendem Code, für den Fall, daß die Assertions für diese Klasse aktiviert sind, auf der Konsole ausgegeben?

```
public class A{
        int x;
        public static void main(String[ ] args){
                A a = new a();
                a.mc(0.0);
        }
        public void mb(){
                System.out.println("Ich bin ein Fehler");
        }
        private void mc(int x) {
                System.out.println("Ich bin Methode mc");
                assert preis <= 0.0: mb();
        }
}
```

a) Ich bin ein Fehler
b) Ich bin Methode mc
c) Kompilierfehler

d) AssertionError
e) Keine dieser Möglichkeiten

(20) Frage

Welche der untenstehenden assert-Statements führt zu einem Kompilierfehler?

a) assert x < 0: break;
b) assert i == 8;
c) assert i == 1: i;
d) assert i == 1: “Fehler”;
e) assert i == 1: i++;
f) Keine dieser Möglichkeiten.

(21) Frage

Welche der untenstehenden assert-Statements führt zu einem Kompilierfehler?

a) assert i >= 8;
b) assert i < 1: i;
c) assert i > 1: “Fehler”;
d) assert i > 1: i++;
e) assert x < 0: return;
f) Keine dieser Möglichkeiten.

(22) Frage

Welche der untenstehenden Aussagen trifft zu?

a) Assertions sind Standardmäßig aktiviert.
b) Assertions lassen sich für einzelne Klassen aktivieren.
c) Assertions wurden mit J2SE 1.3 eingeführt.
d) Assertions wurden mit J2SE 1.4 eingeführt.
e) Assertions sollen Nebeneffekte haben.
f) Assertions sollen in Methoden, die public sind, verwendet werden.
g) Keine dieser Möglichkeiten.

(23) Frage

Welche der untenstehenden Aussagen trifft zu?

a) Es wird empfohlen, Assertions nicht in Methoden zu verwenden, die public sind.
b) Es wird empfohlen, Assertions nicht in Methoden zu verwenden, die private sind.
c) Es wird empfohlen, unerwünschte Nebeneffekte bei der Verwendung von Assertions zu verhindern.
d) Es wird empfohlen, Assertions sogar in Methoden, die public sind, zu verwenden, in Fällen, in denen davon ausgegangen werrden kann, daß sie niemals eintreten, sprich im default-Fall einer switch-Struktur.
e) Es wird empfohlen, Assertions in Methoden zu verwenden, die öffentlich sind.

f) Keine dieser Möglichkeiten.

(24) Frage

Welche der untenstehenden Aussagen treffen, für den Fall, daß Assertions aktiviert sind, zu?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
        for (int j = 1; j < 5; j++){
            switch (j){
                case 1: System.out.println("1"); break;
                case 2: System.out.println("2"); break;
                case 3: System.out.println("3"); break;
                case 4: System.out.println("4"); break;
                default: assert false;
            }
        }
    }
}
```

a) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4.
b) Es kommt zu einem Kompilierfehler.
c) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4 und AssertionError.
d) Assertion sollen nicht in einer Methode verwendet werden, die public ist.
e) Keine dieser Möglichkeiten

(25) Frage

Welche der untenstehenden Aussagen treffen, für den Fall, daß Assertions aktiviert sind, zu?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
        for (int j = 1; j < 6; j++){
            switch (j){
                case 1: System.out.println("1"); break;
                case 2: System.out.println("2"); break;
                case 3: System.out.println("3"); break;
                case 4: System.out.println("4"); break;
                default: assert false;
            }
        }
    }
}
```

a) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4.
b) Es kommt zu einem Kompilierfehler.
c) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4 und AssertionError.
d) Assertion sollen nicht in einer Methode verwendet werden, die public ist.
e) Keine dieser Möglichkeiten

(26) Frage

Welche der untenstehenden Aussagen treffen, für den Fall, daß Assertions nicht aktiviert sind, zu?

```
public class SwitchStruktur {
    public static void main(String[ ] args) {
        for (int j = 1; j < 6; j++){
            switch (j){
                case 1: System.out.println("1"); break;
                case 2: System.out.println("2"); break;
                case 3: System.out.println("3"); break;
                case 4: System.out.println("4"); break;
                default: assert false;
            }
        }
    }
}
```

a) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4.
b) Es kommt zu einem Kompilierfehler.
c) Es wird auf der Konsole folgendes ausgegeben: 1 2 3 4 und AssertionError.
d) Assertion sollen nicht in einer Methode verwendet werden, die public ist.
e) Keine dieser Möglichkeiten

7. Lösungen

(1) Frage

b

(2) Frage

a

(3) Frage

c

(4) Frage

g

(5) Frage

c, f

(6) Frage

a, d

(7) Frage

c, f

(8) Frage

b, d

(9) Frage

e

(10) Frage

c, Nach dem Doppelpunkt darf keine Zuweisung stehen.

(11) Frage

f

(12) Frage

c, Nach dem assert-Statement muß ein primitiver boolean-Ausdruck stehen, es darf keine Zuweisung stehen.

(13) Frage

c, Nach dem assert-Statement muß ein primitiver boolean-Ausdruck stehen, es darf keine Zuweisung stehen.

(14) Frage

b, d

(15) Frage

b

(16) Frage

b, Sind die Assertions nicht aktiviert, wird das assert-Statement ignoriert.

(17) Frage

c, Nach dem assert-Statement muß ein primitiver boolean-Ausdruck stehen, es darf keine Zuweisung stehen.

(18) Frage

b, d

(19) Frage

c, Nach dem Doppelpunkt im assert-Statement darf kein Methodenaufruf für eine Methode ohne Rückgabewert (void) stehen, da dies zu einem Kompilierfehler

führt.

(20) Frage

a, Nach dem Doppelpunkt im assert-Statement darf kein break stehen, da dies zu einem Kompilierfehler führt.

(21) Frage

e, Nach dem Doppelpunkt im assert-Statement darf kein return stehen, da dies zu einem Kompilierfehler führt.

(22) Frage

b, d

(23) Frage

a, c, d

(24) Frage

a, Empfehlung der Verwendung von Assertions, sogar in Methoden die public sind, in Fällen, in denen davon ausgegangen werden kann, daß sie niemals eintreten. Dies ist im Fall des Default-Falles einer switch-Strukur der Fall und es kommt weder zu einem AssertionError noch zu einem Kompilierfehler.

(25) Frage

c, Für j=6 tritt der default-Fall ein, deswegen wird zusätzlich bei aktivierten Assertions ein AssertionError geworfen.

(26) Frage

a, Für j=6 tritt der default-Fall ein, bei nicht aktivierten Assertions hat dies keinerlei Ausgabe zur Folge.

XI. Threads

1. Begriffserläuterung

Threads zu deutsch Fäden, sind quasi parallel ablaufend Aktivitäten innerhalb eines Programms. So können Sie sich beispielsweise Ihr Word-Programm vorstellen, in dem Sie gleichzeitig drucken und ihren Text formatieren können. Dem User wird hierbei vorgegaukelt, daß diese Prozesse gleichzeitig ablaufen. Tatsächlich aber wechseln sie sich in der Bearbeitung nach dem Zufallsprinzip ab. Nehmen wir einmal an, Sie haben drei Threads in Gang gebracht. So könnten die Threads wie folgt abgearbeitet werden:

thread a
thread a
thread b
thread c
thread a
thread a

Wie Sie sehen können, ist keinerlei Systematik erkennbar. Die Threads werden nach einem chaotischen Prinzip abwechselnd aufgerufen. Diese Reihenfolge kann theoretisch beeinflußt werden, indem wir höhere Prioritäten setzen, de facto ist dies allerdings nicht möglich. Die Standardpriorität liegt bei 5 (Thread.NORM_PRIORITY), die minimale bei 1 (Thread_MIN_PRIORITY) und die höchste bei 10 (Thread.MAX_PRIORITY).

Threads sind nicht nur zufallsgesteuert, sondern auch plattformabhängig. Von Computer zu Computer und von Betriebssystem zu Betriebssystem können so unterschiedliche Ergebnisse die Folge sein.

2. Erstellung eines Threads

In Java ist ein Thread ein Objekt der Threadklasse und kann auf zwei Arten formuliert werden: Entweder durch Implementieren des Runnable Interface (implements Runnable) oder durch Erstellen einer Subklasse der Thread-Klasse (extends Thread). Vergleichen Sie hierzu das Kapitel Vererbung und das Kapitel Interfaces und abstrakte Klassen. Hier zum besseren Verständnis ein Auszug der Javadocs (Api):

java.lang

Class Thread

```
java.lang.Object
  └ java.lang.Thread
```

All Implemented Interfaces:
Runnable

---

public class **Thread**
extends Object
implements Runnable

a) Erstellen eines Threads durch Implementierung des Runnable Interface

Wenn ein Thread mit dem Runnable Interface erstellt wird, muß die leere Methode run() implementiert werden, da das Interface Runnable wie folgt aussieht:

```
public interface Runnable {
        void run()
}
```

Vergessen Sie die Methode run() zu implementieren, kommt es zu einem Kompilierfehler. Es wird wie folgt vorgegangen:

```
//1. Erstellen einer Klasse, die die Methode run() implementiert
public class Run implements Runnable{
                public static void main(String[ ] args) {
                //2. Erstellen eines Objekts dieser Klasse
                Run a = new Run();
                /*3. Erstellen eines Thread-Objektes,
                dem das Objekt der Klasse Run übergeben wird*/
                Thread b = new Thread(a);
                //4.Starten des Threads
                b.start();
        }

        public void run() {
                System.out.println("Dies ist ein Thread");
        }
}
```

Ausgabe auf der Konsole:

```
Dies ist ein Thread
```

Die Methode start() startet den Thread. Bei b.start() wird zuerst die start()-Methode ausgeführt und dann automatisch die run()-Methode. Somit gibt die Klasse Run auf der Konsole den Satz „Dies ist ein Thread“ aus. Hierbei ist zu beachten, daß ein Thread nur einmal mit der Methode start() zum Laufen gebracht werden kann, ein zweiter Versuch führt zu einem Fehler (java.lang.IllegalThreadStateException). Würde nun b.start() durch b.run(); ersetzt werden, käme es zur gleichen Ausgabe. Es würde allerdings kein Thread gestartet, sondern nur die run()-Methode ausgeführt werden.

Was passiert jedoch, wenn Sie vergessen das Run-Objekt an den Thread zu übergeben? Dies führt lediglich zu keiner Ausgabe und zu keinem Kompilier- oder Laufzeitfehler.

In diesem Zusammenhang ist es wichtig zu wissen, welche Thread-Konstruktoren es gibt. Hier ein Auszug aus der Api (Javadocs):

Constructor Summary
Thread() Allocates a new Thread object.
Thread(Runnable target) Allocates a new Thread object.
Thread(Runnable target, String name) Allocates a new Thread object.
Thread(String name) Allocates a new Thread object.
Thread(ThreadGroup group, Runnable target) Allocates a new Thread object.
Thread(ThreadGroup group, Runnable target, String name) Allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group.
Thread(ThreadGroup group, Runnable target, String name, long stackSize) Allocates a new Thread object so that it has target as its run object, has the specified name as its name, belongs to the thread group referred to by group, and has the specified *stack size*.
Thread(ThreadGroup group, String name) Allocates a new Thread object.

b) Erstellen eines Threads durch Erweiterung mit extends Thread

Erzeugen Sie einen Thread als Subklasse der Klasse Thread, die das Interface Runnable implementiert, wird ähnlich vorgegangen. Im Gegensatz zu der bereits erläuterten Alternative einen Thread mithilfe des Runnable Interface zu erstellen, muß hierbei, die Methode run( ) nicht implementiert werden, da sie bereits in der Klasse Thread existiert und somit an die Subklasse vererbt wird bzw. auch überschrieben werden darf.

```
/*1. Erstellen einer Klasse, die Thread erweitert (extends Thread) und
die Methode run() überschreibt!*/
public class Faden extends Thread{
        public static void main(String[ ] args) {
                //2. Erstellen eines Objektes Faden
                Faden faden = new Faden();
                //3. Starten des Threads
                faden.start();
        }

        public void run() {
                System.out.println("Dies ist ein Faden!");
        }
}
```

Ausgabe auf der Konsole:

Dies ist ein Faden!

3. Der Main-Thread und die Daemon-Threads

Der sogenannte Main-Thread wird automatisch mit jedem Objekt gestartet, und er hat die Aufgabe die main()-Methode zum Laufen zu bringen. Sogenannte DaemonThreads sind Hilfs-Threads für alle „normalen" Threads, sie leben nur, solange andere Threads laufen. In untenstehendem Beispiel wird der Name des Main-Threads mithilfe der Methode Thread.currentThread().getName() ausgegeben, nämlich main. Auf den Main-Thread kann zugegriffen werden, da der Thread meinThread nicht gestartet wird, sondern nur die run()-Methode aufgerufen wird. So existiert nur der Main-Thread und dessen Name erscheint auf der Konsole.

```
public class MeinThread extends Thread{
        //Konstruktor Thread (String name) wird nicht geerbt, sondern muß mit
        //super(s)aufgerufen werden.
        MeinThread(String s){
                super(s);
        }

        public void run(){
                //Gibt den Namen des gerade laufenden Threads wieder
                System.out.print(Thread.currentThread().getName());
        }

        public static void main(String[ ] args) {
                MeinThread meinThread = new MeinThread("eins");
                //Der Thread wird nicht gestartet, es wird nur die run()-Methode
                //aufgerufen
                meinThread.run();
        }
}
```

Ausgabe auf der Konsole:

```
main
```

Wird nun aber der Thread meinThread mit der start()-Methode ins Leben gerufen, wird der Name des Thread meinThread ausgegeben: eins.

```
public class MeinThread extends Thread{
        MeinThread(String s){
                super(s);
        }

        public void run(){
                System.out.print(Thread.currentThread().getName());
        }

        public static void main(String[ ] args) {
                MeinThread meinThread = new MeinThread("eins");
                //Der Thread wird gestartet!
```

```
                meinThread.start();
        }
}
```

Ausgabe auf der Konsole:

eins

4. Synchronisation

Es gibt manche Situationen, in denen es wünschenswert ist, daß nur ein Thread auf eine gemeinsame Ressource zugreifen kann. So wäre es z. B. fatal, wenn man von Ihrem Bankkonto gleichzeitig an mehreren Stellen, am Bankautomat und am Schalter, Geld abheben könnte. Da auf diese Art und Weise der Kreditrahmen ohne Probleme überzogen werden könnte. Es ist sinnvoll, wenn diese Vorgänge hintereinander stattfinden. Synchronisierte Methoden und Blöcke stellen dies sicher. Sind Daten synchronisiert und ist der Zugriff auf die Daten sicher, nennt man diesen Zustand threadsicher. Es können nur Blöcke und Methoden synchronisiert werden und nicht Konstruktoren oder Klassen. Versucht man es trotzdem, kommt es zu einem Kompilierfehler.

Hier ein Beispiel für Threads, die nicht synchronisiert sind. Wie Sie sehen können, laufen die Threads nicht hintereinander ab, sondern in einer nicht vorhersehbaren Reihenfolge. Der zweite Thread startet irgendwann, während der erste Thread noch läuft. Die Ausgabe ist zufalls- und plattformabhängig und ist somit bei jedem Durchlauf eine andere.

```
public class SynchronizedThread extends Thread{
        public static void main(String[ ] args) {
                SynchronizedThread synchronizedThread1 = new
                SynchronizedThread();
                synchronizedThread1.start();
                SynchronizedThread synchronizedThread2 = new
                SynchronizedThread();
                synchronizedThread2.start();
        }

        public void run() {
                        for (int i=0; i<200; i++ ){
                                System.out.println(i);
                        }
        }
}
```

Ausgabe auf der Konsole (Ausschnitt):

```
120
121
122
123
124
```

```
0
125
1
2
3
```

a) Synchronisation von Blöcken

Um zu garantieren, daß Threads hintereinander ausgeführt werden, wird ein Block innerhalb der Klasse SynchronizedThread mit Hilfe eines LOCKs synchronisiert (synchronized (LOCK) { }). Das Lock private static Object LOCK = new Object();, gewährleistet, daß jeweils nur ein Objekt auf diesen Block zugreifen kann. Statische Bestandteile einer Klasse gibt es nur ein einziges Mal pro Klasse, sprich jedes Objekt besitzt kein eigenes LOCK, sondern teilt sich das gleiche LOCK mit allen anderen Objekten der gleichen Klasse. Es gibt also für alle Objekte dieser Klasse nur ein einziges statisches LOCK (vgl. Kapitel Klassen- und Instanzvariablen). Nun werden die Threads hintereinander ausgeführt und es gibt kein Zahlenchaos mehr. Jedes Objekt hat ein Lock, was auf deutsch übersetzt Schloß heißt, und jeweils nur ein Objekt kann im Besitz des Schlüssels dieses Schlosses sein, da es nur einen Schlüssel gibt.

```
public class SynchronizedThread extends Thread{
        private static Object LOCK = new Object();
        public static void main(String[ ] args) {
                SynchronizedThread synchronizedThread1 = new
                SynchronizedThread();
                synchronizedThread1.start();
                SynchronizedThread synchronizedThread2 = new
                SynchronizedThread();
                synchronizedThread2.start();
        }
        public void run() {
                synchronized (LOCK) {
                        for (int i=0; i<200; i++ ){
                                System.out.println(i);
                        }
                }
        }
}
```

Ausgabe auf der Konsole (Ausschnitt):

```
197
198
199
0
1
2
```

Hier noch ein Beispiel eines gelockten Blocks. Jetzt werden zwei Threads, die mithilfe des Runnable Interfaces erstellt wurden, gestartet.

```
public class Synchronized implements Runnable{
	private static Object LOCK = new Object();

	public void run( ) {
		synchronized (LOCK) {
			for (int i=0; i<200; i++){
				System.out.println(i);
			}
		}
	}

	public static void main(String[ ] args) {
		Synchronized synchronized1 = new Synchronized();
		Thread a = new Thread(synchronized1);
		Thread b = new Thread(synchronized1);
		a.start();
		b.start();
	}
}
```

Ausgabe auf der Konsole (Ausschnitt):

```
197
198
199
0
1
2
```

Würde es genügen synchronized (this){ } statt synchronized (LOCK){ } zu schreiben? Nein! Der Ausdruck this bezieht sich auf das jeweilige Objekt und jedes Objekt besitzt eine eigene Methode run( ). Sie haben zwei Thread-Objekte so kann jedes Objekt auf seine eigene Methode run( ) zugreifen und es kommt wieder zu einem Zahlenchaos. Es reicht auch nicht aus, die Methode run( ) mit public synchronized void run() zu synchronisieren, da jeder Thread über seine eigene Methode run() verfügt.

```
public class SynchronizedThread extends Thread{
	public static void main(String[ ] args) {
		SynchronizedThread synchronizedThread1 = new
		SynchronizedThread();
		synchronizedThread1.start();
		SynchronizedThread synchronizedThread2 = new
		SynchronizedThread();
		synchronizedThread2.start();
	}
	public void run() {
		synchronized (this) {
			for (int i=0; i<200; i++ ){
				System.out.println(i);
			}
		}
```

```
        }
}
```

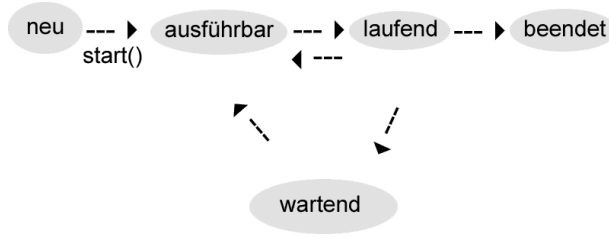
Ausgabe auf der Konsole (Ausschnitt):

```
123
124
0
125
1
2
```

Noch ein paar weitere Bemerkungen zur Synchronisation: Sie können auch andere Objekte zum Synchronisieren benutzen. Nehmen Sie z. B. ein statisches String-Objekt, dann hat dies den gleichen Effekt wie ein statischer Lock, da es das statische String-Objekt pro Klasse auch nur einmal gibt. Bei String-Literalen verhält es sich gleich, da String-Literale mit gleichem Inhalt, auch nur ein einziges Mal im String-Pool vorkommen. Nur bei nicht statischen String-Objekten ist es anders, jedes Objekt besitzt ein separates String-Objekt (vgl. Kapitel zum Thema „Die Klasse String“).

5. Die unterschiedlichen Thread-Stadien und die dazugehörigen Methoden

In untenstehendem Schaubild können Sie die verschiedenen Stadien eines Threads sehen. Mit der Methode start() wird ein neuer Thread in einen ausführbaren Zustand gebracht, d. h. der Thread ist bereit zu laufen, läuft aber noch nicht. Die Java Virtual Machine regelt den Übergang von ausführbar zu laufend, wobei der genaue Zeitpunkt plattformabhängig ist. Ein Thread kann während er läuft in eine wartende Phase überführt werden. Ist er allerdings einmal beendet, kann er nicht wieder zum Leben erweckt werden, er ist tot. Befindet sich ein Thread im wartenden Stadium, wartet er darauf, daß ihm der Schlüssel des Locks (zu deutsch: Schloß), den es nur einmal gibt, übergeben wird.



In der folgenden Aufstellung werden alle wichtige Methoden die Threads betreffend in eine Systematik gebracht. Es ist für das Zertifikat notwendig zu wissen, welcher

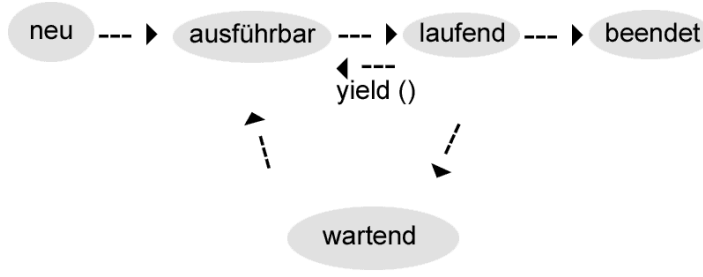
Klasse oder welchem Interface sie angehören:

Klasse java.lang.Object	***Klasse java.lang.Thread***	***Interface Runnable***
wait()	start()	run()
notify()	yield() (statische Methode)	
notifyAll()	sleep() (statische Methode)	
	join()	

Zunächst einige allgemein für die Zertifizierung relevante Eigenschaften dieser Methoden: Den Methoden wait() und join() kann man einen Zeitparameter übergeben, wohingegen man der Methode sleep(long) einen Zeitparameter übergeben muß. Zusätzlich gibt es Methoden von den abgeraten wird, sie in Zukunft zu benutzen, die es aber noch gibt, sie sind deprecated (resume, stop, suspend). Die Methoden wait(), notify() und notifyAll() dürfen nur in synchronisierten Blöcken verwendet werden und die Methoden sleep, join und wait müssen sich in einem try/catch-Block befinden und werfen eine InterruptedException.

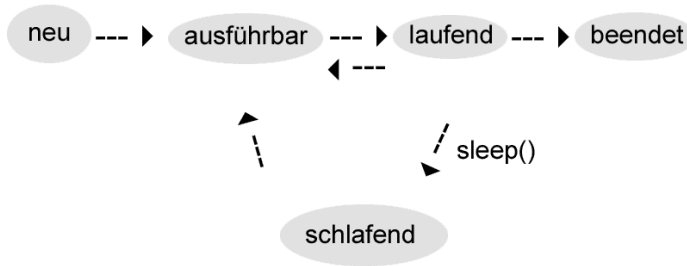
a) <u>Die Methode yield()</u>

Yield() bringt den Thread vom laufenden Zustand in den Ausführbaren und ermöglicht es somit, anderen Threads in den Laufenden gelangen zu können. Dies ist aber nur der Fall, wenn es andere Threads gibt, die auf die Ausführung warten. Ist kein Thread im ausführbaren Stadium, läuft der Thread trotz yield() weiter! Außerdem kann nicht vorausgesagt werden, ob der Thread tatsächlich in den ausführbaren Zustand gelangt, da die Methode yield() den Thread beeinflußt oder auch nicht. Die tatsächliche Reaktion ist also nicht sicher.



b) <u>Die Methode sleep()</u>

Die Methode sleep(long) schickt den Thread zum Schlafen und zwar für mindestens die angegebene Zeit. Sie muß in einem try/catch-Block stehen, da der Thread, wenn er unabsichtlich aufgeweckt wird, eine InterruptedException wirft, die aufgefangen werden muß. Fehlt der try/catch-Block, kommt es ebensfalls zu einer Exception.



In untenstehendem Beispiel können Sie sehen, wie zwischen der Ausgabe von „Hallo“ und „eins“ mindestens 1 Sekunde (1000 Millisekunden) vergeht! Als Parameter muß der Methode sleep(long) eine Zeitangabe übergeben werden, um so einen Kompilierfehler zu verhindern.

```
public class MeinFaden extends Thread{
        public void run(){
                try{
                        System.out.println(”Hallo“);
                        Thread.sleep(1000);
                        System.out.println(Thread.currentThread( ).getName( ));
                }
                catch(InterruptedException e){ }
        }

        public static void main(String[ ] args) {
                Thread meinFaden1 = new Thread(new MeinFaden(),“eins“);
                meinFaden1.start();
        }
}
```

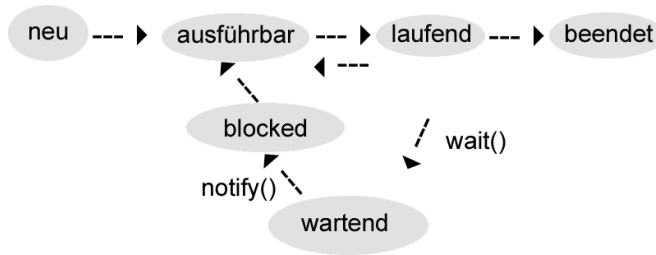
Ausgabe auf der Konsole:

Hallo
(Wartezeit von mindestens 1000 Millisekunden)
eins

c) Die Methoden wait() und notify() und notifyAll()

Diese Methoden beziehen sich auf Threads, die auf das gleiche synchronisierte Objekt zugreifen. Der Thread wartet, wenn auf ihm die Methode wait() (zu deutsch: warten) aufgerufen wird, bis die Methode notify() (zu deutsch: benachrichtigen) ihn wieder benachrichtigt. Handelt es sich um mehrere Threads, kann notifyAll() alle Threads benachrichtigen. Während die Methode wait() läuft, gibt der Thread vorübergehend den Schlüssel für das Lock ab, wobei gleichzeitig verhindert wird, daß dieser Thread von außen unterbrochen wird, bis die notify()-Methode ausgeführt wird. Damit ein Thread, die notify()-Methode auf einem Objekt aufrufen kann, muß es den Schlüssel des Locks von diesem Objekt besitzen, sprich es

muß in einem synchronized(this)-Block stehen. Nach der Beendigung der Methode notify() kommt der Thread u. U. zuerst in den Zustand blocked, in dem er auf den Schlüssel warten muß, bevor er dann wieder in den Ausführbaren kommt.



Untenstehende Threads zeigen anhand eines Beispiels die Funktionen der Methoden wait() und notify(). Die Methode wait() stellt sicher, daß die for-Schleife von der Klasse ThreadNotify bis zum Ende durchlaufen werden kann. Ist die for-Schleife beendet, kommt notify() zum Zuge und die Klasse ThreadWait läuft weiter. Wir erhalten also als Ausgabe für den letzten Eintrag die Zahl 99. Die Methode notify() muß in einem synchronized-Block stehen, der den Schlüssel zu dem Lock des Objektes besitzen muß, das die wait()-Methode aufgerufen hat. So muß dort in untenstehendem Fall synchronized (this) stehen (vgl. Kapitel zum Thema this). Ist dies nicht der Fall, wird eine java.lang.IllegalMonitorStateException geworfen. Das Thread-Objekt, das sich jeweils im ausführbaren Zustand befindet, verfügt über den Schlüssel des LOCKs. Beide Threads wechseln sich also mit dem Besitz ab.

```
class ThreadWait {
        public static void main(String[ ] args) {
                ThreadNotify n = new ThreadNotify();
                n.start();

                //wait() muß in einem synchronized und in einem try/catch-Block
                //stehen
                synchronized (n) {
                        try {
                                //wait() veranlaßt das Programm zu warten
                                n.wait();
                        } catch (InterruptedException e) {
                        }
                }
                System.out.println("Der letzte Eintrag ist: " + n.last);
        }
}

class ThreadNotify extends Thread {
        int last;

        public void run() {
                //notify() muß in einem synchronized Block stehen
```

```
                synchronized (this) {
                        for (int i = 0; i < 100; i++) {
                                last = i;
                        }
                //notify() benachrichtigt wait(), daß das Warten ein Ende hat
                        notify();
                }
        }
}
```

Ausgabe auf der Konsole:

Wir warten...
Der letzte Eintrag ist: 99

Was würde passieren, wenn die wait()-Methode fehlen würde? Das Programm würde nach dem Starten des Threads sofort, nach einmaligem Durchlaufen der for-Schleife, weiterlaufen und die Funktion System.out.println(„Der letzte Eintrag ist: „ + b.last); würde ausgeführt werden, wobei dann auf der Konsole der Satz „Der letzte Eintrag ist 0“ erscheinen würde, wie Sie in untenstehendem Beispiel sehen können.

```
class ThreadWait {
        public static void main(String[ ] args) {
                ThreadNotify n = new ThreadNotify();
                n.start();
                System.out.println("Der letzte Eintrag ist: " + n.last);
        }
}

class ThreadNotify extends Thread {
        int last;

        public void run() {
                synchronized (this) {
                        for (int i = 0; i < 100; i++) {
                                last = i;
                        }
                        notify();
                }
        }
}
```

Ausgabe auf der Konsole:

Der letzte Eintrag ist: 0

d) Die Methode join()

Join() macht es möglich, daß ein Thread auf einen anderen wartet, bis dieser geendet hat und erst dann wird der wartende Thread ausgeführt. Die Methode join() kann einen Zeitparameter haben und muß in einem try/catch-Block stehen.

6. Übungen

(1) Frage

Welche der folgenden Methoden gehören zur Klasse java.lang.Object?

a) wait()
b) sleep()
c) join()
d) notify()
e) notifyAll()
f) start()
g) yield()
h) run()

(2) Frage

Welche der folgenden Methoden gehören zur Klasse java.lang.Thread?

a) wait()
b) sleep()
c) join()
d) notify()
e) notifyAll()
f) start()
g) yield()
h) run()

(3) Frage

Welche der folgenden Methoden gehören zum Interface Runnable?

a) wait()
b) sleep()
c) join()
d) notify()
e) notifyAll()
f) start()
g) yield()
h) run()

(4) Frage

Welche der folgenden Methoden sind statische Methoden der Klasse java.lang. Thread?

a) wait()
b) sleep()
c) join()
d) notify()
e) notifyAll()

f) start()
g) yield()
h) run()

(5) Frage

Welche dieser Aussagen trifft zu?

a) Thread.MAX_PRIORITY == 1
b) Thread.MAX_PRIORITY == 10
c) Thread.MAX_PRIORITY == 5
d) Thread.MIN_PRIORITY == 1
e) Thread.MIN_PRIORITY == 5
f) Thread.MIN_PRIORITY == 10
g) Thread.Min_PRIORITY == 0
h) Thread.MAX_PRIORITY == 15

(6) Frage

Welche dieser Aussagen trifft zu?

a) Thread.NORM_PRIORITY == 10
b) Thread.NORM_PRIORITY == 0
c) Thread.NORM_PRIORITY == 5
d) Thread.MIN_PRIORITY == 1
e) Thread.MAX_PRIORITY == 5
f) Thread.MAX_PRIORITY == 10
g) Thread.NORM_PRIORITY == 1
h) Thread.NORM_PRIORITY == 15

(7) Frage

Welche Methode muß bei public class MyRunnable implements Runnable{..} implementiert werden?

a) wait()
b) sleep()
c) join()
d) notify()
e) notifyAll()
f) start()
g) yield()
h) run()

(8) Frage

Welche Methode darf bei public class MyThread extends Thread{..} überschrieben werden?

a) wait()
b) sleep()

c) join()
d) notify()
e) notifyAll()
f) start()
g) yield()
h) run()

(9) Frage

Welche der folgenden Aussagen sind zutreffend?

a) Die Klasse Thread implements Runnable.
b) Die Methode join von java.lang.Thread ist static.
c) Die Methode wait() gehört zu der Klasse Object:
d) Ein Thread kann zweimal mit der Methode start() gestartet werden.
e) Ein Thread kann mit der Methode run() gestartet werden.

(10) Frage

In welcher Zeile tritt eine Exception auf?

```
public class Faden implements Thread{                    Zeile 1
        public static void main(String[ ] args) {        Zeile 2
                Faden faden = new Faden();               Zeile 3
                faden.start();                           Zeile 4
        }

        public void run() {                              Zeile 5
                System.out.println("Dies ist ein Faden!");  Zeile 6
        }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(11) Frage

In welcher Zeile tritt eine Exception auf?

```
public class Faden extends Thread{                       Zeile 1
        public static void main(String[ ] args) {        Zeile 2
                Faden faden = new Faden();               Zeile 3
                faden.start();                           Zeile 4
        }

        public void run() {                              Zeile 5
                System.out.println("Dies ist ein Faden!");  Zeile 6
```

```
        }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Keine dieser Möglichkeiten.

(12) Frage

In welcher Zeile tritt eine Exception auf?

```
public class Faden implements Thread{                    Zeile 1
        public static void main(String[ ] args) {        Zeile 2
                Faden faden = new Faden();               Zeile 3
                faden.start();                           Zeile 4
                faden.start();                           Zeile 5
        }

        public void run() {                              Zeile 6
                System.out.println("Dies ist ein Faden!");  Zeile 7
        }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Zeile 5
f) Zeile 6
g) Zeile 7
h) Keine dieser Möglichkeiten.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Run implements Runnable{
        public static void main(String[ ] args) {
                Run a = new Run();
                Thread b = new Thread(a);
                b.start();
        }

        public void run() {
        System.out.println("Dies ist ein Thread");}}
```

a) Es tritt eine Exception auf.

b) Keine Ausgabe.
c) Dies ist ein Thread.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Run implements Runnable{
        public static void main(String[ ] args) {
                Run a = new Run();
                Thread b = new Thread();
                b.start();
        }
        public void run() {
        System.out.println("Dies ist ein Thread");
}
}
```

a) Es tritt eine Exception auf.
b) Keine Ausgabe.
c) Dies ist ein Thread.

(15) Frage

Welche der folgenden Phasenübergänge zwischen den verschieden Threadstadien gibt es?

a) Vom beendeten Zustand zum Laufenden.
b) Vom ausführbaren Zustand in den Laufenden.
c) Vom Laufenden in den ausführbaren Zustand.
d) Vom Laufenden in den beendeten Zustand.
e) Vom wartenden in den laufenden Zustand.
f) Vom wartenden in den ausführbaren Zustand.

(16) Frage

Welche Methode führt nur unter Umständen in den ausführbaren Zustand?

a) Die yield()-Methode.
b) Die notify()-Methode.
c) Die join()-Methode.
d) Die sleep()-Methode.
e) Die start()-Methode.
f) Die run()-Methode.
g) Die wait()-Methode.
h) Die notifyAll()-Methode.

(17) Frage

Welche Methode bringt einen Thread zum Starten?

a) Die yield()-Methode.
b) Die notify()-Methode.
c) Die join()-Methode.
d) Die sleep()-Methode.
e) Die start()-Methode.
f) Die run()-Methode.
g) Die wait()-Methode.
h) Die notifyAll()-Methode.

(18) Frage

Welche Methode weckt einen wartenden Thread auf?

a) Die yield()-Methode.
b) Die notify()-Methode.
c) Die join()-Methode.
d) Die sleep()-Methode.
e) Die start()-Methode.
f) Die run()-Methode.
g) Die wait()-Methode.
h) Die notifyAll()-Methode.

(19) Frage

Welche der untenstehenden Methoden sind static?

a) Die yield()-Methode.
b) Die notify()-Methode.
c) Die join()-Methode.
d) Die sleep()-Methode.
e) Die start()-Methode.
f) Die run()-Methode.
g) Die wait()-Methode.
h) Die notifyAll()-Methode.

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class MyThread extends Thread{
        public static void main(String[ ] args) {
                MyThread myThread1 = new MyThread();
                myThread1.start();
                MyThread myThread2 = new MyThread();
                myThread2.start();
}
        public void run() {
                synchronized (this) {
                        for (int i=0; i<10; i++ ){
                        System.out.println(i);
                        }
```

```
                }
        }
}
```

a) Es tritt ein Kompilierfehler auf.
b) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben.
c) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben, aber in chaotischer Reihenfolge.
d) Es wird nichts ausgegeben.
e) Es wird ein Thread gestartet.

(21) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class MyThread extends Thread{
        public static void main(String[ ] args) {
                MyThread myThread1 = new MyThread();
                myThread1.start();
                MyThread myThread2 = new MyThread();
                myThread2.start();
}
        public void run() {
                        for (int i=0; i<10; i++ ){
                        System.out.println(i);
                        }
        }
}
```

a) Es tritt ein Kompilierfehler auf.
b) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben.
c) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben, aber in chaotischer Reihenfolge.
d) Es wird nichts ausgegeben.
e) Es wird ein Thread gestartet.

(22) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class MyThread extends Thread{
        private static Object Lock = new Object();
        public static void main(String[ ] args) {
                MyThread myThread1 = new MyThread();
                myThread1.start();
                MyThread myThread2 = new MyThread();
                myThread2.start();
}
        public void run() {
                synchronized (Lock) {
                        for (int i=0; i<10; i++ ){
```

```
                        System.out.println(i);
                        }
                }
        }
}
```

a) Es tritt ein Kompilierfehler auf.
b) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben.
c) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben, aber in chaotischer Reihenfolge.
d) Es wird nichts ausgegeben.
e) Es wird ein Thread gestartet.

(23) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class MyThread extends Thread{
        public static void main(String[ ] args) {
                MyThread myThread = new MyThread();
                myThread.start();
                myThread.start();
}
        public void run() {
                synchronized (this) {
                        for (int i=0; i<10; i++ ){
                        System.out.println(i);
                        }
                }
        }
}
```

a) Es tritt ein Kompilierfehler auf.
b) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben.
c) Es werden zweimal hintereinander die Zahlen 1 bis 10 ausgegeben, aber in chaotischer Reihenfolge.
d) Es wird nichts ausgegeben.
e) Es wird ein Thread gestartet.
f) Es wird eine IllegalThreadStateException zur Laufzeit geworfen.

7. Lösungen

(1) Frage

a, d, e

(2) Frage

b, c, f, g

(3) Frage
h

(4) Frage
b, g
(5) Frage
b, d

(6) Frage
c, d, f

(7) Frage
h

(8) Frage
h
(9) Frage
a

(10) Frage
a, Es muß public class Faden extends Thread { .. } heißen.

(11) Frage
g

(12) Frage
e, Die Methode start() darf nicht zweimal aufgerufen werden.

(13) Frage
c

(14) Frage
b, Es müsste das Runnable-Objekt, wie folgt übergeben werden: Thread b = new Thread(a);

(15) Frage
b, c, d, f

(16) Frage
a

(17) Frage
e

(18) Frage
b, h

(19) Frage
a, d

(20) Frage
c

(21) Frage
c

(22) Frage
b

(23) Frage
f

XII. Garbage Collection

1. Begriffserläuterung

In der Programmiersprache Java gibt es einen Papierkorb in Form eines automatischen Aufräumvorgangs, der nicht mehr referenzierte Objekte einsammelt und vernichtet. Auf diese Art und Weise wird nicht mehr benötigter Speicher freigegeben. Kommt ein Objekt für die Garbage Collection in Frage, kann nicht genau vorhergesagt werden, wann und ob das Objekt tatsächlich beseitigt wird.

2. Freigabe von Objekten zur Garbage Collection

Betrachten wir nun einige Beispiele, in denen gezeigt wird, zu welchem Zeitpunkt ein Objekt nicht mehr referenziert und somit für die Garbage Collection freigegeben wird.

a) Erneute Wertzuweisung

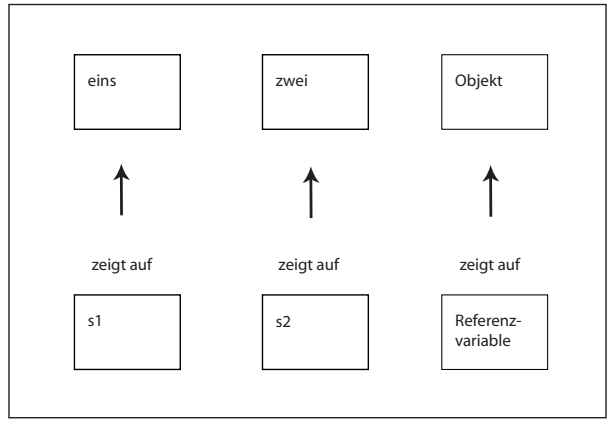
(1) Wertzuweisung von null

Wird einer Referenzvariablen erneut ein Wert zugewiesen, kann es sein, daß das zugehörige Objekt nicht mehr referenziert wird. In untenstehendem Beispiel wird s1 null zugewiesen, somit wird das String-Objekt nicht mehr referenziert, und das Objekt mit dem Inhalt „eins“ wird zur Garbage Collection freigegeben. Hierbei muß man sich vergegenwärtigen, daß das Objekt und die Referenzvariable nicht identisch sind, sondern die Referenzvariable nur auf das Objekt zeigt.

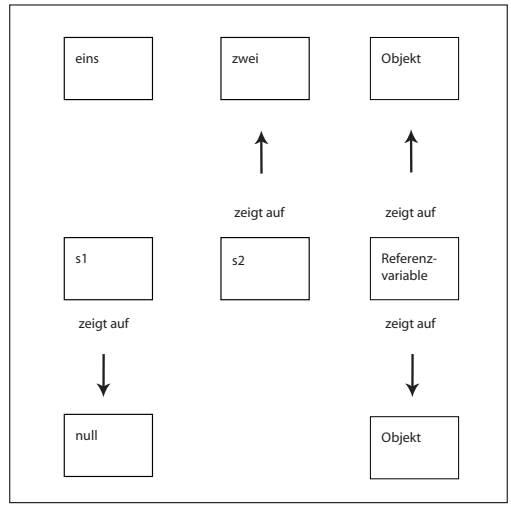
```
3  public class GarbageCollector {
4      public static void main(String[] args) {
5          String s1 = new String("eins");
6          String s2 = new String("zwei");
7          //Die Objekte s1 und s2 kommen nicht für die GC in Frage
8          s1 = null;
9          /*Da das Stringobjekt, das "eins" beinhaltet,
10         nicht mehr erreichbar ist, kommt es für die GC in Frage*/
11     }
12 }
```

Nun schauen wir uns Schritt für Schritt an, was genau passiert.

1. Schritt
In den Zeilen 5 und 6 sehen die Beziehungen zwischen Objekt und Referenzvariable wie folgt aus:



2. Schritt
In Zeile 8 wird der Objektreferenz s1 = null zugewiesen, so zeigt auf das Stringobjekt, das den Inhalt „eins“ hat, keine Referenzvariable mehr, d. h. es ist isoliert. Das Objekt ist nicht mehr erreichbar, und kommt aus diesem Grund für die Garbage Collection in Frage.



(2) Erneute Wertzuweisung durch einen anderen Wert

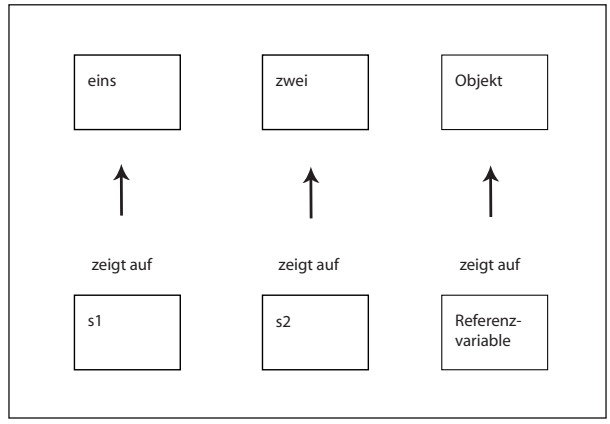
Wie Sie in untenstehendem Beispiel sehen können, wird bei einer erneuten Wertzuweisung zu einer Referenzvariablen, hier s2 = new String(„drei“);, das Stringobjekt, das den Wert „zwei“ beinhaltet, zur Garbage Collection freigegeben, da es nun isoliert ist. Die Referenzvariable s2 hat nun den Wert „drei“ und nicht mehr „zwei“. Der gleiche Effekt würde erreicht werden, wenn Sie, z. B. s2 = s1; zuweisen würden, das Ergebnis ist immer die Isolierung des Strings, der „zwei“ beinhaltet.

```
3  public class GarbageCollector {
4      public static void main(String[] args) {
5          String s1 = new String("eins");
6          String s2 = new String("zwei");
7          s2 = new String("drei");
8          /*Da nun das Stringobjekt, das "zwei" beinhaltet,
9          nicht mehr erreichbar ist, kommt es für die GC in Frage*/
10     }
11 }
```

Schauen wir uns diese Vorgänge wieder Schritt für Schritt an:

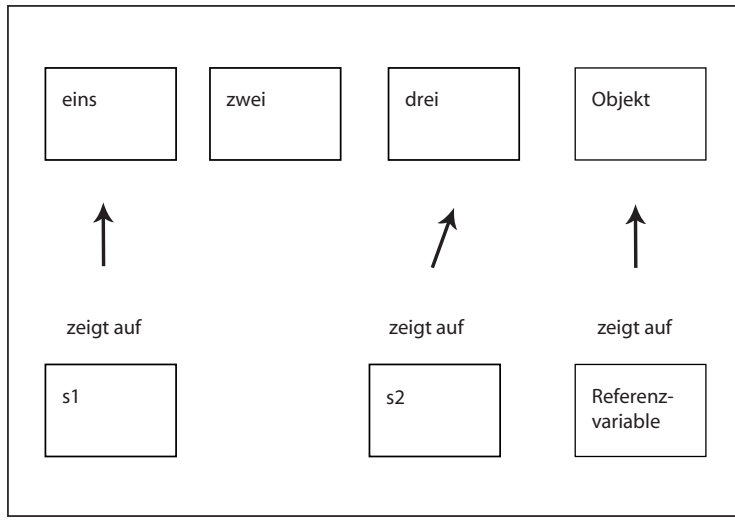
1. Schritt

In den Zeilen 5 und 6 sehen die Beziehungen zwischen Objekt und Referenzvariable wie folgt aus:



2. Schritt

In Zeile 7 wird eine erneute Zuweisung durchgeführt s2 = new String (“drei“), was untenstehende Veränderungen zur Folge hat. Nun kommt das Objekt, das den Inhalt „zwei“ hat, für die Garbage Collection in Frage.



(3) Übergabe an eine andere Methode

In untenstehendem Beispiel wird die Referenzvariable, bevor ihr ein neuer Wert zugewiesen wird, an eine Methode übergeben, von der wir nicht wissen, welchen Inhalt sie hat. In dieser Methode könnte also das String-Objekt, das den Wert „zwei“ besitzt, einer anderen Referenzvariablen zugewiesen werden. Somit wäre dieses Objekt nicht mehr isoliert und könnte nicht von dem „Müllmann“ eingesammelt werden. Es wäre also möglich, daß dieses Objekt nicht für die Garbage Collection in Frage kommt.

```
public class GarbageCollector {
        public static void main(String[ ] args) {
                String s1 = new String(”eins“);
                String s2 = new String(”zwei“);
                arbeite(s2);
                s2 = new String(”drei“);
        }

        private static void arbeite(String s2){
                //Hier steht viel Code!
        }
}
```

Wird allerdings der Code wie untenstehend verändert, kommt das Stringobjekt mit dem Inhalt „zwei“ bereits nach der Zeile 5 für die Garbage Collection in Frage, da nun das Stringobjekt mit dem Inhalt „zwei“ isoliert dasteht.

```
public class GarbageCollector {                          Zeile 1
        public static void main(String[ ] args) {        Zeile 2
                String s1 = new String("eins");          Zeile 3
                String s2 = new String("zwei");          Zeile 4
                s2 = new String("drei");                 Zeile 5
                arbeite(s2);                             Zeile 6
        }

        private static void arbeite(String s2){
                //Hier steht viel Code!
        }
}
```

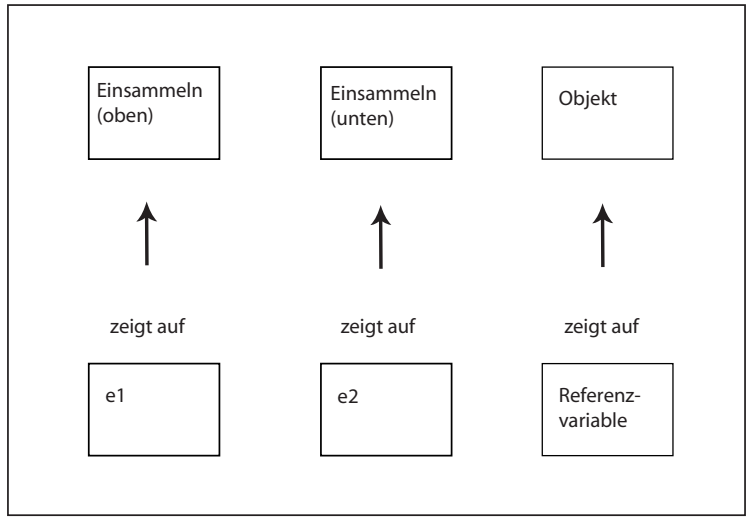
b) <u>Isolierte Inseln</u>

Sogenannte Isolierte Inseln sind Beispiele für Objekte, die für die Garbage Collection in Frage kommen, obwohl diese noch gültige Referenzen haben. Ein gutes Beispiel hierfür ist, wenn wie in untenstehendem Beispiel die Instanzvariable eines Objektes, eine Referenzvariable zu einer anderen Instanz der gleichen Klasse ist. Auf diese Art und Weise zeigt die Referenzvariable e1.e auf das Objekt e2 und die Referenzvariable e2.e auf das Objekt e1. Ab Zeile 5 und 6 sind diese zwei Objekte isoliert, da die Referenzen von e1 und e2 nicht mehr auf die ursprünglichen Objekte von Zeile 1 und 2 hindeuten. Es existieren zwar noch gültige Referenzen, diese können allerdings nicht mehr von außen erreicht werden. Nun sind e1.e und e2.e isoliert, aus Ihnen ist eine sogenannte Isolierte Insel geworden.

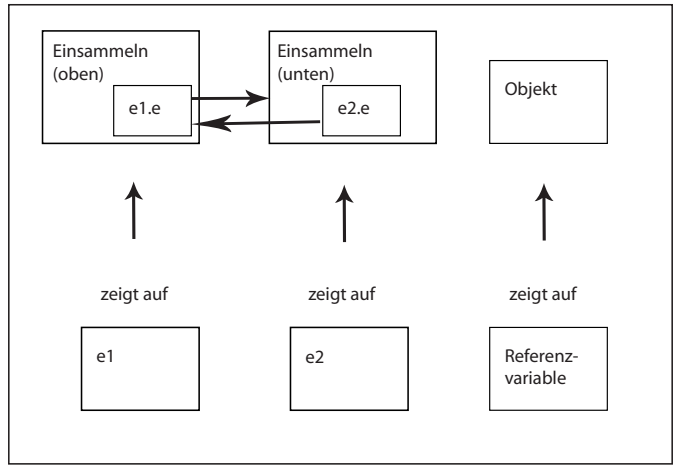
```
public class Einsammeln {
        Einsammeln e;
        public static void main(String[ ] args) {
                Einsammeln e1 = new Einsammeln();        Zeile 1
                Einsammeln e2 = new Einsammeln();        Zeile 2
                e1.e = e2;                               Zeile 3
                e2.e = e1;                               Zeile 4
                e1 = null;                               Zeile 5
                e2 = null;                               Zeile 6
        }
}
```

Lassen Sie uns nun schrittweise die einzelnen Vorgänge genauer betrachten:

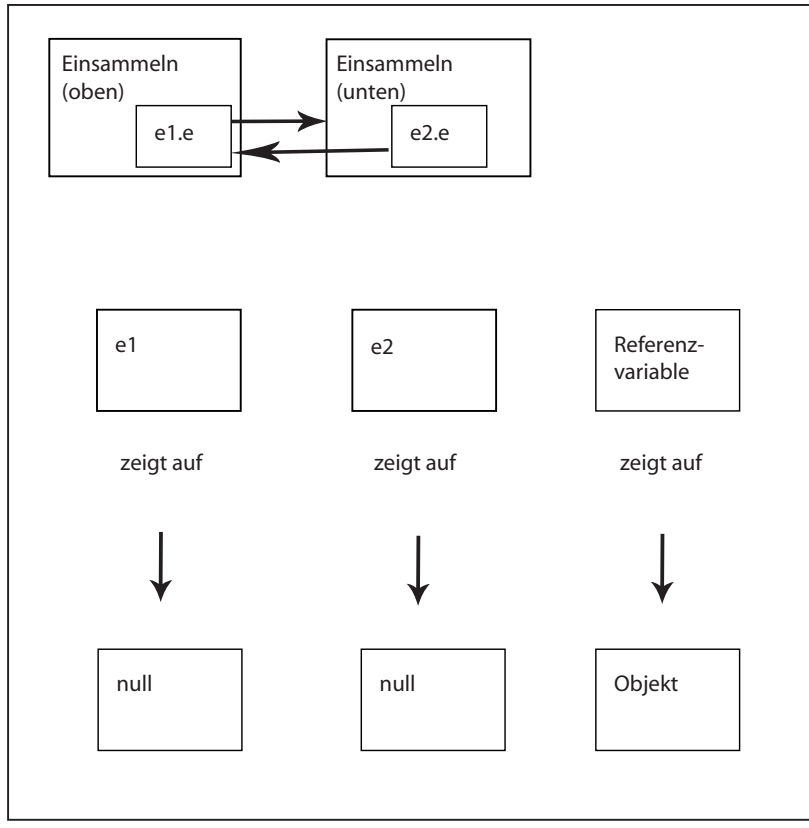
1. Schritt
In den Zeilen 1 und 2 sehen die Beziehungen zwischen Objekt und Referenzvariable wie folgt aus:



2. Schritt
Ab Zeile 3 und 4 weist die Referenzvariable e1.e auf das Objekt e2 und die Referenzvariable e2.e auf das Objekt e1.



3. Schritt
In Zeile 5 und 6 wird e1 und e2 dem Objekt null zugewiesen, wobei eine sogenannte Isolierte Insel ensteht. Von außen zeigt keine Referenz mehr auf e1.e und e2.e. Sie sind also von außen nicht mehr erreichbar, obwohl sie untereinander noch gültige Referenzen besitzen.



3. Object Finalization

In Java wird ein Objekt, das für die Garbage Collection in Frage kommt, darüber informiert, daß es demnächst von dem Garbage Collector eingesammelt wird, indem die Methode protected void finalize() throws Throwable der Klasse java.lang.Object aufgerufen wird, die jedes Objekt automatisch erbt. Diese Methode stellt also eine Art letzten Aufenthaltsort zur Verfügung. An diesem Ort befinden sich ein Objekt, kurz bevor es tatsächlich zerstört wird.

Die finalize( )-Methode darf mit noch genaueren und detaillierteren Vorschriften

zur Garbage Collection mit protected oder public überschrieben werden. Sie darf auch überladen werden. Wird die finalize()-Methoden allerdings überladen, muß trotzdem weiterhin immer die finalize()-Methode der Klasse Object bei der Garbage Collection aufgerufen werden und nicht die Überladene.

Die letzte Methode, die auf einem Objekt ausgeführt wird, das für die Garbage Collection in Frage kommt, ist die finalize()-Methode, erst danach wird das Objekt zerstört. Beachten Sie bitte, daß die finalize()-Methode nur einmal auf einem Objekt aufgerufen und ausgeführt wird. Man kann allerdings mit einer überschriebenen finalize()-Methode, die Freigabe für die Garbage Collection wieder rückgängig machen.

Eine Implementierung der finalize()-Methode nennt sich Finalizer. Ruft der Garbage Collector aber in jedem Fall den Finalizer auf? Nein! Man sollte sich also niemals darauf verlassen. Die finalize()-Methode wird u. U. niemals aufgerufen und die Garbage Collection nie durchgeführt. Sollte die finalize()-Methode eine Exception werfen, wenn sie vom Garbage Collector aufgerufen wird, und sollte diese nicht aufgefangen werden, so wird diese ignoriert und das Objekt bleibt weiterhin freigegeben für die Garbage Collection.

4. Die Methode void gc()

Die gc()-Methode stellt eine Möglichkeit dar, die Garbage Collection zu bitten, Müll einzusammeln, indem sie die finalize()-Methode aufruft. Wobei aber allgemein empfohlen wird, die statische Methode System.gc() zu verwenden. Denken Sie aber immer daran, daß nicht vorausgesagt werden kann, zu welchem Zeitpunkt und ob überhaupt die Garbage Collection durchgeführt wird. Diese Methode stellt also nur einen Versuch dar, die Garbage Collection zu beschleunigen, aber keine Garantie.

Hier ein Beispiel für die Methoden finalize() und System.gc(). Die Methode System.gc() bringt die Garbage Collection zum Laufen, die Methode finalize() wird aber erst beim zweiten Durchlauf der for-Schleife tatsächlich ausgeführt. Je nach Systemvoraussetzungen kann dieses Ergebnis allerdings anders aussehen. Die finalize()-Methode wird aber für jedes Objekt maximal einmal ausgeführt. Bei jeder for-Schleife wird erneut ein Objekt erstellt, so wird für jedes Objekt separat, die finalize()-Methode durchgeführt. Die Ausgabe muß jedoch nicht notwendigerweise so sein, da die Garbage Collection nicht erzwungen werden kann und nicht genau vorhergesagt werden kann, wann und ob überhaupt die Garbage Collection durchgeführt wird. So wird für das dritte Objekt keine finalize()-Methode durchgeführt.

```
public class Einsammeln {
        private String s;
        public void arbeite(String s){
                System.out.println(s);
        }
        protected void finalize (){
```

```
                System.out.println(”finalize“ );
        }
        public static void main(String[ ] args) {
                for (int i = 0; i < 3; i++){
                        Einsammeln e1 = new Einsammeln();
                        e1.arbeite(“ja”);
                        System.gc();
                }
        }
}
```

Ausgabe auf der Konsole:

```
ja
ja
finalize
ja
finalize
```

5. Übungen

(1) Frage

Welche der untenstehenden Bemerkungen sind richtig?

a) Nicht alle Objekte haben die finalize()-Methode.
b) Die finalize()-Methode kann nicht überschrieben werden.
c) Eine Implementierung der finalize()-Methode nennt sich Finalization.
d) Die letzte Methode, die auf einem Objekt ausgeführt wird, das für die Garbage Collection in Frage kommt, ist die finalize()-Methode, erst danach wird das Objekt zerstört.
e) Man sollte sich also immer darauf verlassen, daß die Garbage Collection tatsächlich durchgeführt wird.
f) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Bemerkungen sind richtig?

a) Die finalize()-Methode kann für jedes Objekt beliebig oft aufgerufen werden.
b) Ein Objekt, das für die Garbage Collection in Frage kommt, wird auf jeden Fall von der Garbage Collection zerstört.
c) Die finalize()-Methode darf nicht überladen werden.
d) Sogenannte Isolierte Inseln kommen nicht für die Garbage Collection in Frage.
e) Die Methode System.gc() erzwingt die Durchführung der Garbage Collection.
f) Keine dieser Möglichkeiten.

(3) Frage

Welche der untenstehenden Bemerkungen sind nicht richtig?

a) Die Methode System.gc() erbittet die Durchführung der Garbage Collection.
b) Die Methode finalize() wird von der Klasse java.lang.Object geerbt.
c) Die finalize()-Methode stellt also eine Art Zwischenstop zur Verfügung, kurz bevor ein Objekt tatsächlich eingesammelt wird.
d) Finalizer wird eine Implementierung der finalize()-Methode genannt.
e) Es ist nie genau sicher, wann die Garbage Collection durchgeführt wird.
f) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class E {
    private String s;
    public void hole(String s){
        System.out.println(s);
    }
    protected void finalize (){
```

```
            System.out.println("zerstöre" );
        }
        public static void main(String[ ] args) {
            for (int i = 0; i < 2; i++){
                E e1 = new E();
                e1.hole("hole");
                System.gc();
            }
        }
}
```

a) hole zerstöre hole zerstöre
b) hole hole zerstöre
c) Keine dieser Möglichkeiten!

(5) Frage

Wann kommt das Stringobjekt mit dem Inhalt „ich“, das in Zeile 1 erstellt wurde, für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String("ich");          Zeile 1
        String s2 = new String("bin");          Zeile 2
        arbeite(s2);                            Zeile 3
        s1 = s2;                                Zeile 4
    }
    private static void arbeite(String s) {
        // Hier steht viel Code!
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

(6) Frage

Wann kommt das Stringobjekt mit dem Inhalt „ich“, das in Zeile 1 erstellt wurde, für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String("ich");          Zeile 1
        String s2 = new String("bin");          Zeile 2
        arbeite(s1);                            Zeile 3
        s2 = s1;                                Zeile 4
    }
    private static void arbeite(String s) {
        // Hier steht viel Code!
    }}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

(7) Frage

Wann kommt das Stringobjekt mit dem Inhalt „ich“, das in Zeile 1 erstellt wurde, für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);          Zeile 1
        String s2 = new String(”bin“);          Zeile 2
        s1 = s2;                                Zeile 3
        arbeite(s1);                            Zeile 4
    }
    private static void arbeite(String s) {
        // Hier steht viel Code!
    }}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

(8) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);
        String s2 = new String(”bin“);
        String s3 = new String(”ein“);
        String s4 = new String(”Genie!“);
        s2 = s1;
        s3 = s4;
        s4 = s2;
    }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(9) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);
        String s2 = new String(”bin“);
        String s3 = new String(”ein“);
        String s4 = new String(”Genie!“);
        s2 = s1;
        s3 = s4;
        s4 = s2;
        s1 = null;
    }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(10) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);
        String s2 = new String(”bin“);
        String s3 = new String(”ein“);
        String s4 = new String(”Genie!“);
        s2 = s1;
        s3 = s4;
        s4 = s2;
        s3 = null;
    }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(11) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);
        String s2 = new String(”bin“);
        String s3 = new String(”ein“);
        String s4 = new String(”Genie!“);
        s2 = s1;
        s3 = s2;
        s4 = s2;
    }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(12) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class G {
    public static void main(String[ ] args) {
        String s1 = new String(”ich“);
        String s2 = new String(”bin“);
        String s3 = new String(”ein“);
        String s4 = new String(”Genie!“);
        s2 = s1;
        s3 = s2;
        s4 = new String(”Hugo“);
    }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(13) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class E {
    E e;
    public static void main(String[ ] args) {
        E e1 = new E();
        E e2 = new E();
        E e3 = new E();
        E e4 = new E();
```

```
            e1.e = e2;
            e2.e = e1;
            e3 = null;
            e4 = null;
      }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

(14) Frage

Wieviele Objekte kommen nach Ablauf dieses Codes für die Garbage Collection in Frage?

```
public class E {
      E e;
      public static void main(String[ ] args) {
            E e1 = new E();
            E e2 = new E();
            E e3 = new E();
            E e4 = new E();
            e1.e = e2;
            e2.e = e1;
            e1 = null;
            e2 = null;
            e3 = null;
            e4 = null;
      }
}
```

a) 1
b) 2
c) 3
d) 4
e) Keine dieser Möglichkeiten.

6. Lösungen

(1) Frage

d

(2) Frage

f

(3) Frage

f, Beachten Sie, daß dort *nicht* richtig steht.

(4) Frage

c, Keine dieser Möglichkeiten, da nicht genau vorhergesagt werden kann, wann und ob überhaupt die Garbage Collection durchgeführt wird.

(5) Frage

d, Vorsicht hier wird die Referenzvariable s2 an die Methode arbeite(s2) übergeben und nicht s1.

(6) Frage

e, Da die Referenzvariable s1 an die Methode arbeite(s1) übergeben wird und wir nicht wissen, was in dieser Methode geschieht, können wir auch nicht voraussagen, ab welchem Zeitpunkt dieses Objekt für die Garbage Collection in Frage kommt

(7) Frage

c, Da die Referenzvariable s1 erst nach der erneuten Zuweisung von s1 = s2 an die Methode arbeite(s1) übergeben wird, kommt ab Zeile 3 das Stringobjekt mit dem Inhalt „ich“ für die Garbage Collection in Frage.

(8) Frage

b

(9) Frage

b

(10) Frage

c

(11) Frage

c

(12) Frage

c

(13) Frage

b

(14) Frage

d, 4 Objekte kommen für die Garbage Collection in Frage, obwohl es noch gültige Referenzen gibt, aber es sind sogenannte isolierte Inseln entstanden, die von außen nicht mehr erreichbar sind.

XIII. Das Collections-Framework

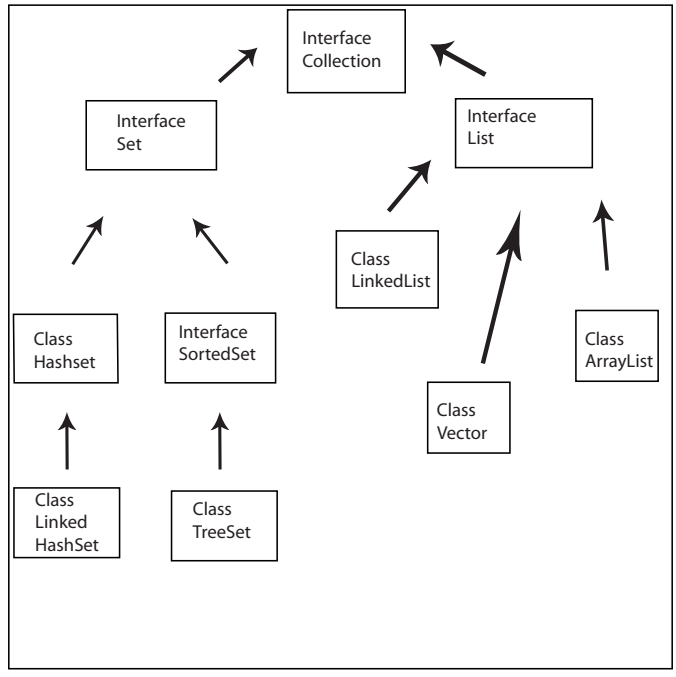
A. Begriffsbestimmung

Das Collections-Framework stellt Möglichkeiten, wie z. B. Sets, Maps oder Listen, zur Verfügung, die es Ihnen erlauben mehrere Objekte ein- und auszulesen. So ist beispielsweise ein Set ein Container, der mehrere String-Objekte enthält:

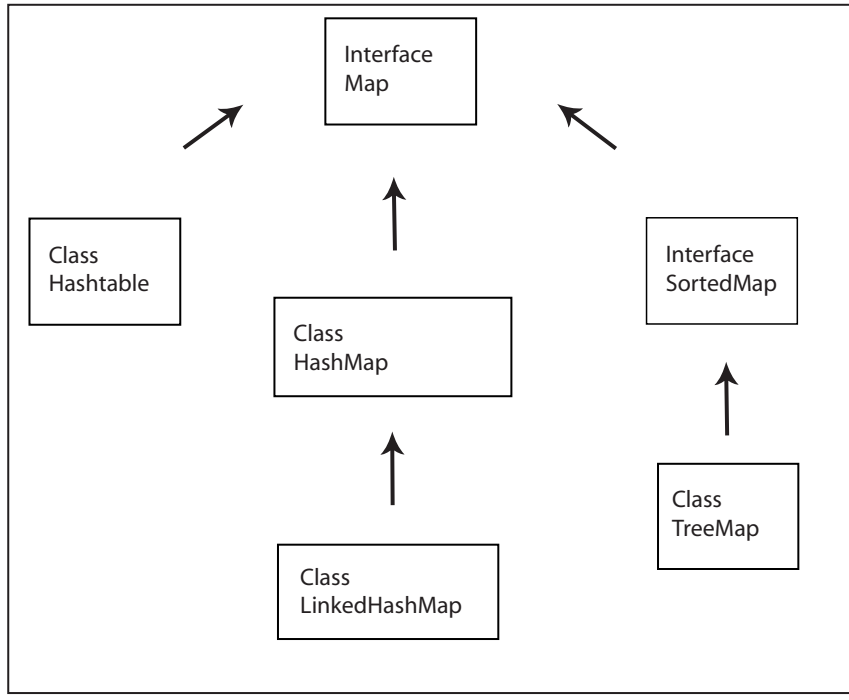
Set
String st = new String(”Hallo“);
String st1 = new String(”Bye“);
String st2 = new String(”Auf Wiedersehen“);
String st3 = new String(”Goodbye“);

Das Collections-Framework umfaßt sowohl das Interface Collection als auch das Interface Map. Direkte Subinterfaces des Interfaces Collection sind das Interface Set und List. Achtung: Das Collections-Framework wird am Ende mit s geschrieben, das Collection-Interface allerdings ohne.

1. Übersicht Interface Collection



2. Übersicht Interface Map



B. Das Interface Collection

1. Das Interface Set

Ein Set ist eine Ansammlung von Feldern ohne Index, wobei keine Duplikate erlaubt sind, sprich in einem Set darf kein Wert zweimal vorkommen.

Set
String st = new String(”Hallo“);
String st1 = new String(”Bye“);
String st2 = new String(”Auf Wiederssehen“);
String st3 = new String(”Goodbye“);

a) Das HashSet und LinkedHashSet

Die Klasse HashSet ist eine direkte Unterklasse des Interfaces Set. Die Klasse LinkedHashSet wiederum ist eine direkte Unterklasse der Klasse HashSet. Ein HashSet speichert seine Werte ungeordnet in einer Hash-Tabelle und Sie können

auf seine Elemente ähnlich schnell zugreifen wie bei einem Array.

Das LinkedHashSet gewährleistet bei der Ausgabe die gleiche Reihenfolge der Objekte wie bei der Eingabe (First in, First out). Das Gegenteil ist bei einem HashSet der Fall, es garantiert keinerlei Ordnung seiner Elemente.

Hier ein Beispiel wie Objekte in ein HashSet ein- und wieder ausgelesen werden können. Mit der Methode boolean add(Object o) wird das HashSet um ein Objekt ergänzt und die Methode gibt true zurück, wenn es das entsprechende Element noch nicht gibt und es dem HashSet hinzugefügt werden konnte. Handelt es sich allerdings um ein Duplikat, wird false zurückgegeben. Die Objekte werden mit der Methode next() der Klasse Iterator wieder ausgelesen. Die Methode hasNext() der Klasse Iterator stellt fest, ob es noch ein zusätzliches Element am Ende des HashSets gibt.

```
import java.util.HashSet;
import java.util.Iterator;

public class CollectionSet {
        public static void main(String[ ] args) {
                HashSet s = new HashSet();
                String st = new String(”Hallo“);
                String st1 = new String(”Bye“);
                String st2 = new String(”Auf Wiedersehen“);
                String st3 = new String(”Goodbye“);
                //Elemente werden hinzugegefügt
                s.add(st);
                s.add(st1);
                s.add(st2);
                s.add(st3);
                //Elemente werden wieder ausgegeben
                for (Iterator i = s.iterator(); i.hasNext();){
                        Object o = i.next();
                        System.out.println(o);
                }
        }
}
```

Ausgabe auf der Konsole:

```
Hallo
Goodbye
Bye
Auf Wiedersehen
```

Bitte beachten Sie, daß in das HashSet zwar String-Objekte eingegeben werden, aber Objekte der Klasse java.lang.Object ausgegeben werden. Wollen Sie das Object wieder in ein Stringobjekt umwandeln, erfordert dies einen Cast: String o1 = (String)i.next(); (vgl. Kapitel Subtyping und Casting von Objekten). Der Iterator ist nur in der Lage Objekte der Klasse Object auszulesen.

Hier ein Beispiel, in dem das HashSet durch ein LinkedHashSet ersetzt wurde: Jetzt ist die Reihenfolge der Ausgabe identisch mit der Reihenfolge der Eingabe.

```
import java.util.LinkedHashSet;
import java.util.Iterator;

public class CollectionSet {
	public static void main(String[ ] args) {
		LinkedHashSet s = new LinkedHashSet();
		String st = new String(”Hallo“);
		String st1 = new String(”Bye“);
		String st2 = new String(”Auf Wiedersehen“);
		String st3 = new String(”Goodbye“);
		s.add(st);
		s.add(st1);
		s.add(st2);
		s.add(st3);

		for (Iterator i = s.iterator(); i.hasNext();){
			Object o = i.next();
			System.out.println(o);
		}
	}
}
```

Ausgabe auf der Konsole:

```
Hallo
Bye
Auf Wiedersehen
Goodbye
```

b) TreeSet

Bei einem TreeSet werden die Elemente, im Gegensatz zu den anderen Sets, in eine natürliche Ordnung gebracht, sprich in eine alphabetische. Diese Reihenfolge hat aber zur Folge, daß ein TreeSet relativ langsam ist, da das alphabetische Sortieren der Elemente Zeit braucht. In ein TreeSet können Sie nur bestimmte Elemente einfügen, wie z. B. Objekte der Klasse String, die das Comparable Interface implementiert. Weiter unten im Kapitel Das Comparator und Comparable Interface gehe ich genauer auf die Voraussetzungen ein, die ein Objekte, das in ein TreeSet eingefügt wird, erfüllen muß.

```
import java.util.TreeSet;
import java.util.Iterator;

public class CollectionSet {
	public static void main(String[ ] args) {
```

```
                TreeSet s = new TreeSet();
                String st = new String("Hallo");
                String st1 = new String("Bye");
                String st2 = new String("Auf Wiedersehen");
                String st3 = new String("Goodbye");
                s.add(st);
                s.add(st1);
                s.add(st2);
                s.add(st3);

                for (Iterator i = s.iterator(); i.hasNext();){
                        Object o = i.next();
                        System.out.println(o);
                }
        }
}
```

Ausgabe auf der Konsole:

```
Auf Wiedersehen
Bye
Goodbye
Hallo
```

2. Das Interface List

Bei einer List gibt es Felder und einen Index, wie bei einem Array. Grundsätzlich sind doppelte Einträge erlaubt, da sie sich durch die Indexposition unterscheiden. Bei allen Listen stimmt die Reihenfolge der Ausgabe der Objekte mit der Reihenfolge der Eingabe überein.

List	
Indexposition	**Element**
0	String st = new String("delta");
1	String st1 = new String("aber");
2	String st2 = new String("beta");

a) LinkedList

Eine LinkedList ist zu bevorzugen, wenn öfter innerhalb, in der Mitte oder am Anfang und am Ende Elemente hinzugefügt oder gelöscht werden müssen, sofern nicht über die Indexposition zugegriffen wird. Der Zugriff auf die Elemente mit Hilfe der Indexposition ist sehr langsam, da es sich intern nicht um ein Array handelt, sondern um eine Zweifach-Verkettete-Liste, sprich jedes Element ist über seinen Vorgänger und Nachfolger innerhalb der Liste informiert.

Es ist die effizienteste Möglichkeit der identischen Reihenfolge der Ein- und

Ausgabe (FiFo = First in, First out) von allen Listen, Sets und Maps. Außerdem gibt es Methoden, wie z. B. void addFirst(Object o) oder Object removeFirst(), die es ermöglichen, am Anfang oder Ende Elemente hinzuzufügen oder zu löschen, ohne auf den Index zugreifen zu müssen.

```
import java.util.Iterator;
import java.util.LinkedList;

public class CollectionList {
	public static void main(String[ ] args) {
		LinkedList s = new LinkedList();
		String st = new String(”delta“);
		String st1 = new String(”aber“);
		String st2 = new String(”beta“);
		s.add(st);
		s.add(st1);
		s.add(st2);
		for (Iterator i = s.iterator(); i.hasNext(); ) {
			Object o = i.next();
			System.out.println( o );
		}
	}
}
```

Ausgabe auf der Konsole:

```
delta
aber
beta
```

Will man nicht alle Elemente auslesen, sondern nur ein Bestimmtes, an einer speziellen Indexposition, benötigt man die Methode Object get (int index). In untenstehendem Beispiel befindet sich an der Stelle 1 der String „aber“, da wie bei Arrays bei 0 angefangen wird zu zählen.

```
import java.util.LinkedList;

public class CollectionList {
	public static void main(String[ ] args) {
		LinkedList s = new LinkedList();
		String st = new String(”delta“);
		String st1 = new String(”aber“);
		String st2 = new String(”beta“);
		s.add(st);
		s.add(st1);
		s.add(st2);

		Object o = s.get(1);
		System.out.println( o );
	}
```

}

Ausgabe auf der Konsole:

aber

b) ArrayList

Die ArrayList ist in den meisten Fällen die beste Implementierung aller Listen. Man hat mit ihr einen indexierten und sehr schnellen wahlfreien (random access) Zugriff. Eine ArrayList basiert auf einem Array, das in der Größe flexibel und dynamisch wachsen und schrumpfen kann.

c) Vector

Der Vector ist ein Array, das sich von der Größe her verändern kann. Er hat zwar auch einen schnellen wahlfreien Zugriff (random access), ist aber langsamer als die ArrayList, da er synchronisiert, d. h. thread-safe (thread-sicher), ist (vgl. Kapitel Threads).

C. Das Interface Map

Maps bestehen aus Feldern und Schlüsseln (keys). Jeden Schlüssel gibt es nur einmal, was zur Folge hat, daß ein gleichlautender neuer Schlüssel einen alten ersetzt. Felder können allerdings Duplikate enthalten, wenn sie unterschiedliche Schlüssel haben. Bitte beachten Sie, daß das Interface Map (ebenso List und Set) zwar deklariert, aber nicht instanziiert werden kann, da man nur Klassen instanziieren kann und nicht Interfaces.

Map	
Schlüssel	**Element**
String s1 = ”1“	String w1 = ”hallo“;
String s2 = ”2“	String w2 = ”bye“;

1. Hashtable

Eine Hashtable ist synchronisiert und somit thread-sicher (thread-safe). Die Elemente werden nicht sortiert, somit ist die Reihenfolge der Eingabe und Ausgabe nicht identisch, wie Sie in untenstehendem Beispiel sehen können. Dieses Mal habe ich das Interface Enumeration zur Hilfe genommen, um die einzelnen Objekte des Hashtable auszulesen. Die Methode hasMoreElements() stellt fest, ob es noch mehr Elemente im Hashtable gibt, die Methode nextElement() gibt die Schlüssel zurück und die Methode get(o) wählt den Eintrag an der entsprechenden Schlüsselposition aus.

```
import java.util.Enumeration;
import java.util.Hashtable;

public class MapHashtable {
        public static void main(String[ ] args) {
                String s1 = ”1”;
                String s2 = ”2”;
                String w1 = ”hallo”;
                String w2 = ”bye”;

                Hashtable h = new Hashtable();
                h.put(s1, w1);
                h.put(s2, w2);

                for (Enumeration en = h.keys(); en.hasMoreElements(); ) {
                        Object o = en.nextElement();
                        System.out.println(“Schlüssel: “ + o );
                        System.out.println(“Wert: “ + h.get(o) );
                }
        }
}
```

Ausgabe auf der Konsole:

```
Schlüssel: 2
Wert: bye
Schlüssel: 1
Wert: hallo
```

Zu einer NullPointerException kommt es, wenn als Inhalt eines Feldes oder eines Schlüssels Standardwert null eingegeben wird. Bei der HashMap indes ist dies erlaubt.

```
 6 public class MapHashtable {
 7     public static void main(String[] args) {
 8         String s1 = "1";
 9         String s2 = "2";
10         String w1 = "hallo";
11         String w2 = null;
12
13         Hashtable h = new Hashtable();
14         h.put(s1, w1);
15         h.put(s2, w2);
16
17         for (Enumeration en = h.keys(); en.hasMoreElements(); ) {
18             Object o = en.nextElement();
19             System.out.println( "Schlüssel: " + o );
20             System.out.println( "Wert: " + h.get(o) );
21         }
```

```
Problems  Javadoc  Declaration  Console  Error Log
<terminated> MapHashtable [Java Application] C:\Programme\Java\j2re1.4.2_05\bin\javaw.exe (02.08.2005 10:52:21
java.lang.NullPointerException
        at java.util.Hashtable.put(Unknown Source)
        at scjp.MapHashtable.main(MapHashtable.java:15)
Exception in thread "main"
```

2. HashMap und LinkedHashMap

Die Elemente in einem HashMap sind nicht sortiert, die Elemente werden in einer anderen Reihenfolge ein- und wieder ausgelesen. Des weiteren ist null als Feld oder als Schlüssel erlaubt. Wohingegen bei der LinkedHashMap das Prinzip first in, first out (fifo) gilt. Die Reihenfolge der Eingabe ist identisch mit der Reihenfolge der Ausgabe. Es gibt aber auch einen zusätzlichen Konstruktor in der LinkedHashMap-Klasse, der eine Reihenfolge last in, first out ermöglicht, sprich das Element, das zuletzt eingegeben wurde, wird als erstes wieder ausgegeben.

3. TreeMap

Das TreeMap implementiert das SortedMap-Interface. Der Inhalt der Elemente wird nach dem Schlüssel (key) in einer natürlichen Reihenfolge sortiert, also alphabetisch oder bei Zahlen aufsteigend. Sowohl beim TreeMap als auch beim TreeSet können nur bestimmte Objekte eigefügt werden. Näheres hierzu finden Sie in untenstehendem Kapitel: Das Comparator und Comparable Interface.

D. Übersicht nach der internen Ordnung der Collections

Gleiche Ordnung Ein- und Ausgabe	**Keine Ordnung**	**Sortiert nach sogenannter natürlicher Ordnung (alphabetisch und nach Zahlen aufsteigend)**
• LinkedHashSet • ArrayList • LinkedList • Vector (thread-safe) • LinkedHashMap	• HashSet • HashMap • Hashtable (thread-safe)	• TreeSet • TreeMap (sortiert nach dem Schlüssel)

E. Das Comparator- und Comparable- Interface

Nun möchte ich Ihnen noch zwei nützliche Interfaces vorstellen, die bei Listen (ArrayList, Vector und LinkedList) anzuwenden sind, wenn Sie beabsichtigen nachträglich die Elemente zu sortieren: Das Comparator und Comparable Interface. Des weiteren sind sie unabdingbar, wenn Sie die Absicht haben, die Klasse TreeMap und TreeSet zu verwenden.

1. Das Comparable Interface

a) Das Comparable Interface und die Klassen TreeSet und TreeMap

Wenn Sie Elemente in ein TreeSet oder in ein TreeMap einfügen wollen, müssen die entsprechenden Objekte das Comparable Interface implementieren. Das Comparable Interface besitzt genau eine Methode, die Sie implementieren müssen: die Methode public int compareTo(Object o). Bitte beachten Sie, daß die Klasse String und die meisten Wrapper-Klassen, dieses Interface bereits von Hause aus implementieren. Untenstehend möchte ich Ihnen anhand der Klasse Buch zeigen, wie das Comparable Interface implementiert wird, und wie Sie anschließend Objekte der Klasse Buch, z. B. dem TreeSet, hinzufügen können.

```
public class Buch implements Comparable{
        String autor;
        public Buch(String a){
                setAutor( a);
        }

        //Dieser Methode muß als Parameter ein Objekt übergeben werden
        public int compareTo(Object o) {

        /*Das übergebene Objekt wird in ein Objekt der Klasse java.lang.Object
        umgewandelt und muß somit wieder in ein Buch gecastet werden */
                return autor.compareTo(((Buch)o).getAutor());
        }
        public String getAutor() {
                return autor;
        }
        public void setAutor(String autor) {
                this.autor = autor;
        }
}
```

Wir haben eine Klasse Buch erstellt, die das Comparable Interface implementiert: Objekte dieser Klasse können Sie nun ohne Probleme in ein TreeSet einlesen. Hätte die Klasse Buch, das Interface Comparable nicht implementiert, würde es zwar zur Kompilierzeit zu keinem Fehler führen, aber zur Laufzeit würde eine Exception (ClassCastException) geworfen. Für die Klasse TreeMap muß analog vorgegangen werden. Objekte, die in ein TreeSet oder in ein TreeMap eingefügt werden, werden alphabetisch sortiert wieder ausgegeben.

```
import java.util.TreeSet;

public class TreeSetSortieren{
        public static void main(String[ ] args) {
                TreeSet s = new TreeSet();
                Buch st = new Buch(”delta“);
                Buch st1 = new Buch(”aber“);
```

```
                Buch st2 = new Buch(”beta“);
                s.add(st);
                s.add(st1);
                s.add(st2);

                for (Object r:s){
                        System.out.println(((Buch)r).getAutor());
                }
        }
}
```

Ausgabe auf der Konsole:

```
aber
beta
delta
```

b) Das Comparable Interface und Listen

Wie verhält es sich mit den Ergebnissen der Listen (ArrayList, Vector und LinkedList)? Können diese auch sortiert werden? Ja: Sie brauchen die statische Methode sort(List list) der Klasse java.util.Collections und Objekte, die das Comparable Interface implementieren. Hätte die Klasse Buch, das Interface Comparable nicht implementiert, führt dies zu keinem Kompilierfehler, aber zu einer ClassCastException.

```
import java.util.ArrayList;
import java.util.Collections;

public class Sortieren{
        public static void main(String[ ] args) {
                ArrayList s = new ArrayList();
                Buch st = new Buch(”delta”);
                Buch st1 = new Buch(”aber“);
                Buch st2 = new Buch(”beta”);
                s.add(st);
                s.add(st1);
                s.add(st2);
                Collections.sort(s);

                for (Object r:s){
                        System.out.println(((Buch)r).getAutor());
                }
        }
}
```

Ausgabe auf der Konsole:

```
aber
beta
delta
```

2. Das Comparator Interface

a) Das Comparator Interface und die Klassen TreeSet und TreeMap

Welchen zusätzlichen Weg gibt es, Objekte einem TreeSet oder einem TreeMap zu übergeben? Sie implementieren das Comparator Interface und übergeben das so entstandene Objekt, dem überladenen TreeSet-Konstruktor als Parameter. Das Interface Comparator befindet sich im Package java.util und es beinhaltet die Methode int compare(Object o1, Object o2), die implementiert werden muß. Der Vorteil dieser Vorgehensweise ist, daß man die Klasse, hier Buch, nicht mit Methoden belastet, die eigentlich nichts mit der Aufgabe dieser Klasse zu tun haben. Die Methode int compare(Object o1, Object o2) wird in eine andere Klasse ausgelagert, der wir den Namen Buchvergleich geben wollen. Untenstehend nun die Klassen Buch und Buchvergleich.

```
public class Buch {
        String autor;
        public Buch(String a){
                setAutor( a);
        }
        public String getAutor() {
                return autor;
        }
        public void setAutor(String autor) {
                this.autor = autor;
        }
}
```

```
import java.util.Comparator;

public class BuchVergleich implements Comparator{

        public int compare(Object o1, Object o2) {

                /*Das übergebene Objekt wird in ein Objekt der Klasse
                java.lang.Object umgewandelt und muß somit wieder in ein Buch
                gecastet werden */
                return (((Buch)o1).getAutor().compareTo(((Buch)o2).getAutor()));
        }
}
```

Nun können Sie ein Objekt der so entstandenen Klasse, die das Comparator-Interface implementiert, dem überladenen TreeSet-Konstruktor als Parameter übergeben und diesem TreeSet können Sie anschließend Buchobjekte hinzufügen.

```
public class TreeSetSortieren{
        public static void main(String[ ] args) {
                BuchVergleich b = new BuchVergleich();
```

```
                TreeSet s = new TreeSet(b);
                Buch st = new Buch(”delta“);
                Buch st1 = new Buch(”aber“);
                Buch st2 = new Buch(”beta“);
                s.add(st);
                s.add(st1);
                s.add(st2);

                for (Object r:s){
                        System.out.println(((Buch)r).getAutor());
                }
        }
}
```

b) <u>Das Comparator Interface und Listen</u>

Wie verhält es sich nun, wenn Sie Listen mit dem Comparator Interface verwenden wollen? Es existiert die überladene Methode sort(List list, Comparator c), der als Parameter ein Objekt, das das Comparator-Interface implementiert, übergeben werden kann. Wollen Sie nun nachträglich das Ergebnis der ArrayList, der sie Objekte der Buchklasse übergeben haben, sortieren, können Sie dies nun mit der überladenen sort()-Methode tun. Untenstehend der veränderte Code, der Klasse Sortieren:

```
import java.util.ArrayList;
import java.util.Collections;

public class Sortieren{
        public static void main(String[ ] args) {
                BuchVergleich b = new BuchVergleich();
                ArrayList s = new ArrayList();
                Buch st = new Buch(”delta“);
                Buch st1 = new Buch(”aber“);
                Buch st2 = new Buch(”beta“);
                s.add(st);
                s.add(st1);
                s.add(st2);
                Collections.sort(s, b);

                for (Object r:s){
                        System.out.println(((Buch)r).getAutor());
                }
        }
}
```

F. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class auto{
}

public class Collections {
	public static void main(String[ ] args) {
		LinkedHashMap l = new LinkedHashMap();
		Auto auto = new Auto();
		System.out.println(l instanceof Auto);
		System.out.println(l instanceof Map);
		System.out.println(l instanceof Object);
	}
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
	public static void main(String[ ] args) {
		LinkedHashSet l = new LinkedHashSet();
		System.out.println(l instanceof Collection);
		System.out.println(l instanceof Map);
		System.out.println(l instanceof Object);
	}
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
	public static void main(String[ ] args) {
		LinkedList l = new LinkedList();
		System.out.println(l instanceof Collection);
		System.out.println(l instanceof Map);
		System.out.println(l instanceof Object);
	}
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
	public static void main(String[ ] args) {
		Hashtable l = new Hashtable();
		System.out.println(l instanceof Collection);
		System.out.println(l instanceof Map);
		System.out.println(l instanceof Object);
	}
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
	public static void main(String[ ] args) {
```

```
                ArrayList l = new ArrayList();
                System.out.println(l instanceof Collection);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof List);
        }
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
        public static void main(String[ ] args) {
                TreeMap l = new TreeMap();
                System.out.println(l instanceof Collection);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof SortedMap);
        }
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
        public static void main(String[ ] args) {
                Vector l = new Vector();
                System.out.println(l instanceof Set);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof List);
        }
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
        public static void main(String[ ] args) {
                LinkedList l = new LinkedList();
                System.out.println(l instanceof Set);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof List);
        }
}
```

a) true true true
b) true false true
c) true true false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
        public static void main(String[ ] args) {
                TreeSet l = new TreeSet();
                System.out.println(l instanceof SortedSet);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof List);
        }}
```

a) true true true
b) true false true
c) true false false
d) false true true
e) false false true
f) false true false

g) false false false
h) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Collections {
        public static void main(String[ ] args) {
                HashSet l = new HashSet();
                System.out.println(l instanceof SortedSet);
                System.out.println(l instanceof Map);
                System.out.println(l instanceof List);
        }}
```

a) true true true
b) true false true
c) true false false
d) false true true
e) false false true
f) false true false
g) false false false
h) Kompilierfehler

(11) Frage

Welche der untenstehenden Klassen ermöglicht es, die Elemente in der gleichen Reihenfolge auszugeben, wie sie eingegeben worden sind?

a) Hashtable
b) HashMap
c) TreeMap
d) HashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(12) Frage

Welche der untenstehenden Klassen ermöglicht es, die Elemente in der gleichen Reihenfolge auszugeben, wie sie eingegeben worden sind?

a) Hashtable
b) LinkedHashMap
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(13) Frage

Welche der untenstehenden Klassen, die das Map-Interface implementiert,

ermöglicht es, die Elemente in der gleichen Reihenfolge auszugeben, wie sie eingegeben worden sind?

a) Hashtable
b) LinkedHashMap
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(14) Frage

Welche der untenstehenden Klassen, die das Set-Interface implementiert, ermöglicht es, die Elemente in der gleichen Reihenfolge auszugeben, wie sie eingegeben worden sind?

a) Hashtable
b) LinkedHashMap
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(15) Frage

Welche der untenstehenden Klasse, die das List-Interface implementiert, ermöglicht es, die Elemente in der gleichen Reihenfolge auszugeben, wie sie eingegeben worden sind?

a) Hashtable
b) LinkedHashMap
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(16) Frage

Welche der untenstehenden Klassen ist die effizientes Umsetzung des Prinzips First in First out?

a) Hashtable
b) LinkedHashMap
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(17) Frage

Welche der untenstehenden Klassen ist die effizientes Umsetzung des Prinzips

First in First out?

a) Hashtable
b) LinkedList
c) TreeMap
d) LinkedHashSet
e) TreeSet
f) Keine dieser Möglichkeiten.

(18) Frage

Welche der untenstehenden Klassen ermöglicht es, am Anfang oder Ende Elemente hinzuzufügen oder zu löschen, ohne auf den Index zugreifen zu müssen?

a) Hashtable
b) Vector
c) LinkedList
d) ArrayList
e) LinkedHashSet
f) TreeSet
g) Keine dieser Möglichkeiten.

(19) Frage

Welche der untenstehenden Klassen ist relativ langsam, wenn auf die Elemente mit Hilfe eines Indexes zugegriffen wird?

a) Hashtable
b) Vector
c) LinkedList
d) ArrayList
e) LinkedHashSet
f) TreeSet
g) Keine dieser Möglichkeiten.

(20) Frage

Welche der untenstehenden Klassen ist ähnlich schnell wie ein Array und erlaubt keine Dublikate?

a) Hashtable
b) Vector
c) LinkedList
d) HashSet
e) LinkedHashSet
f) TreeMap
g) Keine dieser Möglichkeiten.

(21) Frage

Welches der untenstehenden Interfaces hat Felder und Schlüssel?

a) List
b) Set
c) Map
d) Keine dieser Möglichkeiten.

(22) Frage

Welches der untenstehenden Interfaces hat einen Index und Felder?

a) List
b) Set
c) Map
d) Keine dieser Möglichkeiten.

(23) Frage

Welches der untenstehenden Interfaces darf keine Dublikate enthalten?

a) List
b) Set
c) Map
d) Keine dieser Möglichkeiten.

(24) Frage

Welches der untenstehenden Interfaces hat keinen Index, sondern nur Felder?

a) List
b) Set
c) Map
d) Keine dieser Möglichkeiten.

(25) Frage

Welche der Listen ermöglicht es am schnellsten in der Mitte Elemente hinzuzufügen oder zu löschen?

a) Vector
b) LinkedList
c) ArrayList
d) Keine dieser Möglichkeiten.

(26) Frage

Welche der untenstehenden Klassen sind thread-safe?

a) Hashtable
b) Vector
c) LinkedList
d) HashSet

e) LinkedHashSet
f) TreeMap
g) Keine dieser Möglichkeiten.

(27) Frage

Welche der untenstehenden Klassen sind thread-safe?

a) TreeSet
b) HashSet
c) LinkedList
d) Hashtable
e) LinkedHashSet
f) TreeMap
g) Keine dieser Möglichkeiten.

(28) Frage

Welche der untenstehenden Klassen basieren auf einem Array, das sich von der Größe her anpassen kann?

a) TreeSet
b) HashSet
c) LinkedList
d) Hashtable
e) LinkedHashSet
f) TreeMap
g) Keine dieser Möglichkeiten.

(29) Frage

Welche der untenstehenden Klassen basieren auf einem Array, das sich von der Größe her anpassen kann?

a) TreeSet
b) HashSet
c) ArrayList
d) Hashtable
e) LinkedHashSet
f) Vector
g) Keine dieser Möglichkeiten.

(30) Frage

Welche der untenstehenden Klassen hat Felder und Schlüssel, erlaubt null als Element und die Elemente werden nicht sortiert?

a) TreeSet
b) HashSet
c) ArrayList
d) Hashtable

e) LinkedHashSet
f) Vector
g) Keine dieser Möglichkeiten.

(31) Frage

Welche der untenstehenden Klassen hat Felder und Schlüssel, erlaubt null nicht als Element und die Elemente werden nicht sortiert?

a) TreeSet
b) HashSet
c) ArrayList
d) Hashtable
e) LinkedHashSet
f) Vector
g) Keine dieser Möglichkeiten.

(32) Frage

Welche der untenstehenden Klassen sind thread-safe?

a) TreeSet
b) HashSet
c) ArrayList
d) Hashtable
e) LinkedHashSet
f) Vector
g) Keine dieser Möglichkeiten.

(33) Frage

Welche der untenstehenden Klassen ist synchronisiert und hat Felder und Schlüssel?

a) TreeSet
b) HashSet
c) ArrayList
d) Hashtable
e) LinkedHashSet
f) Vector
g) Keine dieser Möglichkeiten.

(34) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class C {
    public static void main(String[ ] args) {
        LinkedList s = new LinkedList();
        String st = new String(”eins“);
```

```
            String st1 = new String(”zwei“);
            String st2 = new String(”drei“);
            s.add(st);
            s.add(st1);
            s.add(st2);
            Object o1 = s.get(1);
            System.out.println(o1);
      }
}
```

a) eins
b) zwei
c) drei
d) eins zwei drei
e) drei eins zwei
f) Keine dieser Möglichkeiten.

(35) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Iterator;
import java.util.TreeSet;

public class CollectionSet {
      public static void main(String[ ] args) {
            TreeSet s = new TreeSet();
            String st = new String(”eins“);
            String st1 = new String(”zwei“);
            String st2 = new String(”drei“);
            s.add(st);
            s.add(st1);
            s.add(st2);

            for (Iterator i = s.iterator(); i.hasNext();){
                  Object o = i.next();
                  System.out.println(o);
            }
      }
}
```

a) eins
b) zwei
c) drei
d) eins zwei drei
e) drei eins zwei
f) Keine dieser Möglichkeiten.

(36) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Iterator;
import java.util.HashSet;

public class CollectionSet {
	public static void main(String[ ] args) {
		HashSet s = new HashSet();
		String st = new String(”eins“);
		String st1 = new String(”zwei“);
		String st2 = new String(”drei“);
		s.add(st);
		s.add(st1);
		s.add(st2);

		for (Iterator i = s.iterator(); i.hasNext();){
			Object o = i.next();
			System.out.println(o);
		}
	}
}
```

a) eins
b) zwei
c) drei
d) eins zwei drei
e) drei eins zwei
f) Keine dieser Möglichkeiten.

(37) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Iterator;
import java.util.LinkedHashSet;

public class CollectionSet {
	public static void main(String[ ] args) {
		LinkedHashSet s = new LinkedHashSet();
		String st = new String(”eins“);
		String st1 = new String(“zwei“);
		String st2 = new String(”drei“);
		s.add(st);
		s.add(st1);
		s.add(st2);

		for (Iterator i = s.iterator(); i.hasNext();){
			Object o = i.next();
			System.out.println(o);
		}
	}
}
```

a) eins

b) zwei
c) drei
d) eins zwei drei
e) drei eins zwei
f) Keine dieser Möglichkeiten.

G. Lösungen

(1) Frage

a, Die Klassen LinkedHashMap und Auto haben kein Verwandtschaftsverhältnis zueinander, deswegen kommt es zu einem Kompilierfehler (vgl. Kapitel Operator: instanceof)

(2) Frage

b

(3) Frage

b

(4) Frage

d

(5) Frage

b

(6) Frage

d

(7) Frage

e

(8) Frage

e

(9) Frage

c

(10) Frage

g

(11) Frage
f

(12) Frage
b, d

(13) Frage
b

(14) Frage
d

(15) Frage
f

(16) Frage
f

(17) Frage
b

(18) Frage
c

(19) Frage
c

(20) Frage
d

(21) Frage
c

(22) Frage
a

(23) Frage
b

(24) Frage
b

(25) Frage
b

(26) Frage
a, b

(27) Frage
d

(28) Frage
g

(29) Frage
c, f

(30) Frage
g

(31) Frage
d

(32) Frage
d, f

(33) Frage
d

(34) Frage
b

(35) Frage
e

(36) Frage
f, Bei einem HashSet kann nicht vorausgesagt werden, in welcher Reihenfolge die Elemente ausgelesen werden.

(37) Frage
d

H. Die hashCode()-Methode

1. Begriffsbestimmung

Wie stellen Sie fest, ob zwei Objekte gleich sind? Mit der hashCode()-Methode aus der Klasse Object. Es ist so ähnlich wie an der Garderobe, an der Sie Ihren Mantel abgeben, Sie bekommen eine Nummer, die es Ihnen später ermöglicht Ihren Mantel wieder abzuholen. Ihre Nummer ist der Hashcode. Wenn ein Objekt in einem Objekt des Collections-Framework gespeichert werden soll, müssen in dieser Klasse folgende Methoden korrekt implementiert werden: public int hashCode() und public boolean equals(Object o). Außerdem muß die Klasse, die die equals()-Methode überschreibt auch die hashCode()-Methode überschreiben. Alle Wrapper, alle Collections-Klassen und die String-Klasse überschreiben die hashCode()- und die equals()-Methode. Nur die StringBuffer-Klasse und die Klasse java.lang.Number überschreiben diese beiden Methoden nicht.

Es gibt mehrere Richtlinien, die Sie einhalten müssen, wenn Sie die hashCode()-Methode und die equals()-Methode in einer Klasse überschreiben. Überschreiben Sie die hashCode()-Methode nicht, erhält jedes Objekt einen unterschiedlichen HashCode, auch wenn der Inhalt der Objekte gleich ist.

(1) Richtlinie

Ein Objekt gibt solange ein Programm ausgeführt wird, immer den gleichen HashCode zurück. Bei jeder erneuten Ausführung darf ein anderer HashCode für das gleiche Objekt zurückgegeben werden.

(2) Richtlinie

Objekte, die im Vergleich mit der equals()-Methode gleich sind (vgl. Fundamental Classes), haben auch einen identischen HashCode.

```
public class HashCode {
        public static void main(String[ ] args) {
                String s = new String(”eins“);
                String s1 = new String(”eins“);

                boolean b1 = s.equals(s1);
                boolean b2 = s.hashCode() == s1.hashCode();
                System.out.println(b1);
                System.out.println(b2);
        }
}
```

Ausgabe auf der Konsole:

```
true
true
```

(3) Richtlinie

Der Umkehrschluß gilt allerdings nicht. Objekte mit identischen HashCodes sind nicht notwendigerweise auch beim Vergleich mit der equals()-Methode gleich. Sprich der gleiche Mantel muß die gleiche Nummer haben, aber an der gleichen Garderobennummer dürfen zwei verschiedene Mäntel hängen. Objekte, die im Vergleich mit der equals()-Methode nicht gleich sind, haben nicht notwendigerweise unterschiedliche HashCodes, d. h. Objekte mit gleichem HashCode sind im Vergleich mit der equals-Methode nicht unbedingt gleich. Sind aber die HashCodes nicht gleich, so dürfen zwei Objekte beim Vergleich mit equals() auch nicht gleich sein.

```
public class HashCode {
        public static void main(String[ ] args) {
                String s = new String(”eins“);
                String s1 = new String(”zwei“);

                boolean b1 = s.equals(s1);
                boolean b2 = s.hashCode() == s1.hashCode();
                System.out.println(b1);
                System.out.println(b2);
        }
}
```

Ausgabe auf der Konsole:

```
false
false
```

2. Übungen

(1) Frage

Welche der untenstehenden Feststellungen sind korrekt?

a) Objekte, die im Vergleich mit der equals()-Methode nicht gleich sind, haben notwendigerweise unterschiedliche HashCodes.
b) Ein Objekt gibt während der Ausführung eines Programms immer einen unterschiedlichen HashCode zurück.
c) Ein Objekt gibt während der Ausführung eines Programms nie den gleichen HashCode zurück.
d) Objekte mit identischen HashCodes, sind auch beim Vergleich mit der equals()-Methode gleich.
e) Keine dieser Möglichkeiten.

(2) Frage

Welche der untenstehenden Feststellungen sind korrekt?

a) Objekte, die im Vergleich mit der equals()-Methode gleich sind, haben notwendigerweise unterschiedliche HashCodes.
b) Ein Objekt gibt während der Ausführung eines Programms immer den gleichen HashCode zurück.
c) Objekte mit identischen HashCodes, sind auch beim Vergleich mit der equals()-Methode gleich.
d) Objekte, die im Vergleich mit der equals()-Methode gleich sind, haben identische HashCodes.
e) Keine dieser Möglichkeiten.

(3) Frage

Nehmen wir an s.equals(t) gibt true zurück, welche der untenstehenden Möglichkeiten trifft dann zu?

a) s.hashCode() == t.hashCode()
b) s.hashCode() != t.hashCode()
c) Keine dieser Möglichkeiten.

(4) Frage

Nehmen wir an s.hashCode() != t.hashCode() gibt true zurück, welche der untenstehenden Möglichkeiten trifft dann zu?

a) !s.equals(t)
b) s.equals(t)
c) Keine dieser Möglichkeiten.

(5) Frage

Welche der untenstehenden Klassen überschreiben die hashCode()- und die equals()-Methode nicht?

a) java.lang.StringBuffer

b) java.lang.String
c) java.lang.Long
d) java.util.Hashtable
e) java.util.LinkedList
f) Keine dieser Möglichkeiten.

(6) Frage

Welche der untenstehenden Klassen überschreiben die hashCode()- und die equals()-Methode nicht?

a) java.lang.String
b) java.lang.Double
c) java.lang.Integer
d) java.util.Vector
e) java.util.LinkedHashSet
f) Keine dieser Möglichkeiten.

(7) Frage

Nehmen wir an o.equals(p) gibt true zurück, welche der untenstehenden Möglichkeiten trifft dann zu?

a) o.hashCode() == p.hashCode()
b) o.hashCode() != p.hashCode()
c) Keine dieser Möglichkeiten.

(8) Frage

Welche der untenstehenden Feststellungen sind nicht korrekt?

a) Werden zwei Objekte mit der equals-Methode verglichen und sie gibt true zurück, so müssen auch die zugehörigen HashCodes im Vergleich true zurückgeben.
b) Werden zwei Objekte mit der equals-Methode verglichen und sie gibt false zurück, so müssen auch die zugehörigen HashCodes im Vergleich false zurückgeben.
c) Sind aber die HashCodes nicht gleich, so dürfen zwei Objekte beim Vergleich mit equals() auch nicht gleich sein.
d) Wenn Sie ein Objekt in einem Objekt des Collections-Framework speichern wollen, müssen in dieser Klasse folgende Methoden korrekt implementiert werden: public int hashCode() und public boolean equals(Object o).
e) Keine dieser Möglichkeiten.

(9) Frage

Welche der untenstehenden Feststellungen sind nicht korrekt?

a) Die equals-Methode muß überschreiben werden, wenn die hashCode-Methode überschrieben worden ist.
b) Alle Wrapper, alle Collections-Klassen und die String-Klasse überschreiben die hashCode()- und die equals()-Methode.
c) Objekte, die im Vergleich mit der equals()-Methode gleich sind, haben auch

einen identischen HashCode.

d) Die StringBuffer-Klasse überschreibt weder die equals-Methode noch die hashCode()-Methode.

e) Keine dieser Möglichkeiten.

(10) Frage

Nehmen wir an s.hashCode() == t.hashCode() gibt false zurück, welche der untenstehenden Möglichkeiten muß dann auch false zurückgeben?

a) !s.equals(t)

b) s.equals(t)

c) Keine dieser Möglichkeiten.

3. Lösungen

(1) Frage

e

(2) Frage

b, d

(3) Frage

a

(4) Frage

a

(5) Frage

a

(6) Frage

f

(7) Frage

a

(8) Frage

b, Achtung! In der Aufgabenstellung steht nicht!

(9) Frage

e, Achtung! In der Aufgabenstellung steht nicht!

(10) Frage

b

XIV. Innere Klassen

A. Statische und nicht-statische innere Klassen und Interfaces

1. Begriffserläuterung

a) Innere Klassen

Seither waren wir es gewohnt einem einzelnen Gegenstand, wie z. B. einem Auto jeweils eine eigene Klasse, zuzuordnen. Es gibt aber Fälle, bei denen es sinnvoll sein kann, Klassen ineinander zu schachteln, weil Objekte der realen Welt auch mehr oder weniger wie eine Schachtel, in der sich noch andere Schachteln befinden, aufgebaut sind: In einem Auto befinden sich Sitze und ein Motor. Oder: In einem Pfirsich gibt es einen Kern.

Zwei Klassen werden also ineinander gefügt. In untenstehendem Beispiel ist die Hülle: der Pfirsich und der Inhalt: der Pfirsichkern. Es existieren sowohl statische als auch nicht statische innere Klassen. Eine innere Klasse muß einen anderen Namen haben wie die äußere Klasse, andernfalls kommt es zu einem Kompilierfehler.

```
//Definition der äußeren Klasse
public class Pfirsich {
        //Definition der inneren Klasse
        public class Kern{

        }
}
```

Nicht nur Klassen, sondern auch Interfaces können innere Klassen besitzen.

b) Innere Interfaces

In einer Klasse kann sich auch ein inneres Interface befinden. Diese sind implizit static und das Schlüsselwort static kann weggelassen werden. Innere Interfaces können public, private, protected oder package local sein.

2. Instanziierung von inneren Klassen

a) Instanziierung von inneren Klassen innerhalb der äußeren Klasse

(1) Bei nichtstatischen inneren Klassen

Eine nicht-statische innere Klasse wird innerhalb der äußeren Klasse (Top-Level-Klasse) so instanziiert, wie Sie es bisher bei allen anderen Klassen auch gewohnt

sind.

```
public class Pfirsich {
        //Instanziierung einer inneren Klasse innerhalb einer äußeren Klasse
        Kern kern = new Kern();
        public class Kern{

        }
}
```

Wird die nicht-statische innere Klasse in der main()-Methode der äußeren Klasse instanziiert, kann auf die äußere Klasse nicht verzichtet werden. Da es ansonsten zu einem Kompilierfehler kommen würde. Sie müssen die innere mit der äußeren instanziieren: Pfirsich.Kern pfirsich = new Pfirsich().new Kern();.

```
public class Pfirsich {
        public class Kern{
        }

        public static void main(String args[ ]){
                Kern kern = new Kern(); //Kompilierfehler
                Pfirsich.Kern pfirsichKern = new Pfirsich().new Kern();
        }
}
```

Damit ist ein großer Nachteil verbunden: Es wird sowohl eine Instanz der äußeren Klasse als auch der inneren erstellt. Es werden beide Standardkonstruktoren ausgeführt und in untenstehendem Beispiel wird sowohl das Wort Pfirsich als auch das Wort Kern auf der Konsole ausgegeben.

```
public class Pfirsich {
        Pfirsich(){
                System.out.println("Pfirsich");
        }
        public class Kern{
                Kern(){
                        System.out.println("Kern");
                }
        }
        public static void main(String args[ ]){
                Pfirsich.Kern pfirsichKern = new Pfirsich().new Kern();
        }
}
```

Ausgabe auf der Konsole:

```
Pfirsich
Kern
```

Würde in der main()-Methode nur new Pfirsich pfirsich = new Pfirsich(); stehen, so würde nur der Ausdruck Pfirsich ausgegeben werden.

```
public class Pfirsich {
        Pfirsich(){
                System.out.println("Pfirsich");
        }
        public class Kern{
                Kern(){
                        System.out.println("Kern");
                }
        }
        public static void main(String args[ ]){
                Pfirsich pfirsich = new Pfirsich();
        }
}
```

Ausgabe auf der Konsole:

```
Pfirsich
```

(2) Bei statischen inneren Klassen

Statische innere Klassen werden innerhalb der äußeren Klasse genauso instanziiert wie nicht-statische innere Klassen.

```
public class Pflanze {
   //Instanziierung einer statischen inneren Klasse innerhalb einer äußeren Klasse
   Blatt blatt = new Blatt();
   public static class Blatt{

   }
}
```

Es gibt aber bei der Objekterstellung innerhalb der main()-Methode der äußeren Klasse einen wichtigen Unterschied zu den nicht-statischen inneren Klassen: Bei statischen inneren Klassen ist zusätzlich eine Instanziierung ohne die äußere Klasse möglich. Wird aber die innere Klasse mit der äußeren instanziiert, gibt es eine Abweichung in der Syntax: Es fehlen die runden Klammern nach new Pflanze auf der rechten Seite und nach Pflanze auf der linken Seite.

```
public class Pflanze {
        public static class Blatt{

        }
        public static void main(String[ ] args){
                Blatt blatt = new Blatt();
                Pflanze.Blatt pflanzeBlatt = new Pflanze.Blatt();
        }
}
```

Es liegt nahe zu fragen: Was wird bei den verschiedenen Instanziierungsarten auf dem Bildschirm ausgegeben? Die Ausgabe steht im Gegensatz zu den nicht-statischen inneren Klassen: In beiden Fällen wird nur das Wort Blatt ausgegeben. Wohingegen gleich bleibt: Eine Instanz der äußeren Klasse Pflanze gibt nur das Wort Pflanze aus.

```
public class Pflanze {
        Pflanze(){
                System.out.println("Pflanze");
        }

        public static class Blatt{
                Blatt(){
                        System.out.println("Blatt");
                }
        }

        public static void main(String[ ] args) {
                Pflanze pflanze = new Pflanze();
                Blatt blatt = new Blatt();
                Pflanze.Blatt pflanzeBlatt = new Pflanze.Blatt();
        }
}
```

Ausgabe auf der Konsole:

```
Pflanze
Blatt
Blatt
```

b) Instanziierung von inneren Klassen außerhalb der äußeren Klasse

Es gibt geringfügige Syntax-Unterschiede bei der Instanziierung von statischen und nicht-statischen inneren Klassen außerhalb der äußeren Klasse: Es fehlt bei der statischen inneren Klasse die runde Klammer nach Pflanze. Außerhalb der äußeren Klasse wird aber auf jeden Fall sowohl bei statischen als auch nicht-statischen inneren Klassen, die äußere Klasse bei der Instanziierung benötigt.

```
public class Obstkorb {
        public static void main(String[ ] args) {
                //Instanziierung einer nicht-statischen inneren Klasse
                Kern kern = new Kern(); //Kompilierfehler
                Pfirsich.Kern pfirsichKern = new Pfirsich().new Kern();
                //Instanziierung einer statischen inneren Klasse
                Blatt blatt = new Blatt(); //Kompilierfehler
                Pflanze.Blatt pflanzeBlatt = new Pflanze.Blatt();
        }
}
```

Die Ausgabe bei der Instanziierung von nicht-statischen inneren Klassen außerhalb der äußeren Klasse weicht von der Ausgabe bei statischen Klassen ab: Es wird zusätzlich eine Instanz der äußeren Klasse erstellt. Es erscheinen bei Pfirsich. Kern pfirsichKern = new Pfirsich().new Kern(); die Wörter „Pfirsich“ und „Kern“ auf der Konsole. Wohingegen bei der statischen inneren Klasse nur die innere Klasse instanziiert und somit auch nur Blatt ausgegeben wird.

```
public class Obstkorb {
        public static void main(String[ ] args) {
                Pfirsich.Kern pfirsichKern = new Pfirsich().new Kern();
                Pflanze.Blatt pflanzeBlatt = new Pflanze.Blatt();
        }
}
```

Ausgabe auf der Konsole:

```
Pfirsich
Kern
Blatt
```

3. Zugelassene Modifier für innere Klassen

Als Modifier für innere Klassen sind erlaubt: abstract, final, private, protected, static, public und package local (sprich ohne Modifier).

4. Zugriff auf Bestandteile der äußeren Klassen durch die innere Klasse

a) Statische innere Klassen

Statische innere Klassen können auf nicht-statische Variablen der äußeren Klasse gar nicht zugreifen: Es kommt zu einem Kompilierfehler.

```
4  public class Pflanze {
5      private String farbe = "grün";
6      public static class Blatt{
7          Blatt(){
8              System.out.println(farbe);
9          }
10     }
11
12 }
```

Wie können wir aber auf die Variable farbe der äußeren Klasse zugreifen? Wir ändern die Variable farbe in eine statische:

```
public class Pflanze {
        static private String farbe = ”grün“;

        public static class Blatt{
                Blatt(){
                        System.out.println(farbe);
                }
        }
}
```

b) Nicht-statische innere Klassen

Im Gegensatz zu den statischen inneren Klassen können nicht-statische innere Klassen auf alle statische und nicht-statische Variablen der äußeren Klasse zugreifen:

```
public class Pfirsich {
        static private String farbe = ”rosa“;
        private String groesse = ”klein“;

        public class Kern{
                Kern(){
                        System.out.println(groesse);
                        System.out.println(farbe);
                }
        }
}
```

5. Zugriff auf Variablen der inneren und äußeren Klasse mit this

a) Bei nicht-statischen Klassen

Wie kann man aber zwischen gleichnamigen Variablen der inneren und äußeren Klasse unterscheiden und innerhalb einer nicht-statischen inneren Klasse darauf zugreifen? Die Antwort ist einfach: mit this (vgl. Kapitel Objektreferenz und Konstruktoraufruf mit this und super).

In untenstehendem Beispiel können Sie innerhalb der nicht-statischen inneren Klasse auf die Variable farbe der Klasse Pfirsich nur mit Bezug auf die Klasse Pfirsich und mit this zugreifen: Pfirsich.this.farbe. Dies entspricht in etwa der Logik der Verwendung von super, wobei hier super aber nicht verwendet werden kann.

Auf die Variable farbe der Klasse Kern können Sie in der Klasse Kern auf zweierlei Arten zugreifen: mit farbe oder this.farbe.

```
public class Pfirsich {
    private String farbe = ”rosa“;

    public class Kern{
        private String farbe = “grau”;
        Kern(){
            System.out.println(“Farbe des Pfirsichs: “ + Pfirsich.this.farbe);
            System.out.println(“Farbe des Kerns: “ + this.farbe);
            System.out.println(“Farbe des Kerns: “ + farbe);
        }
    }
    public static void main(String args[ ]){
        Pfirsich.Kern pfirsichKern = new Pfirsich().new Kern();
    }
}
```

Ausgabe auf der Konsole:

```
Farbe des Pfirsichs: rosa
Farbe des Kerns: grau
Farbe des Kerns: grau
```

b) Bei statischen Klassen

Bei statischen Klassen gilt eine andere Syntax (vgl. Kapitel Klassen- und Instanzvariablen), wenn Sie auf die Variable farbe der äußeren Klasse zugreifen wollen: Pflanze.farbe. Wollen Sie allerdings auf die Variable der inneren Klasse zugreifen gelten weiterhin folgende zwei Alternativen: this.farbe und farbe.

```
public class Pflanze {
        static private String farbe = ”grün“;

        public static class Blatt{
            static private String farbe = “dunkelgrün”;
            Blatt(){
                System.out.println(“Farbe der Pflanze: “ + Pflanze.farbe);
                System.out.println(“Farbe des Blatts: “ + this.farbe);
                System.out.println(“Farbe des Blatts: “ + farbe);
            }
        }

        public static void main(String[ ] args) {
            Pflanze.Blatt pflanzeBlatt = new Pflanze.Blatt();
        }
}
```

Ausgabe auf der Konsole:

Farbe der Pflanze: grün
Farbe des Blatts: dunkelgrün
Farbe des Blatts: dunkelgrün

6. Zugelassene Deklarationen innerhalb von inneren Klassen

a) Nicht-statische innere Klassen

Welche Felder sind innerhalb der nicht-statischen inneren Klassen erlaubt? Kurz gesagt: nicht-statische Deklarationen und final static Felder.

```
5  public class Pfirsich {
6      public class Kern{
7          //nicht erlaubt sind statische Deklarationen
8          static int groesse = 1;
9          //erlaubt sind alle nicht-statische Deklarationen
10         private String farbe = "grau";
11         //erlaubt sind alle final static Deklarationen
12         final static String material = "Holz";
13     }
14 }
```

b) Statische innere Klassen

In statischen inneren Klassen sind alle Variablendeklarationen möglich.

7. Zusammenfassende Aufstellung: Innere Klassen

Stellen wir einige Unterschiede in bezug auf innere Klassen übersichtlich nebeneinander:

	Zugriff auf Bestandteile der äußeren Klasse	**Mögliche Deklarationen innerhalb der inneren Klasse**
Statische innere Klasse	nur statische	alle
Nicht-statische innere Klasse	alle	• nur nicht-statische • aber: final static

8. Übungen

(1) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse innerhalb der äußeren Klasse an der Stelle zu instanziieren, an der das Wort hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    hier
    public class BKlasse{
    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();
d) Keine dieser Möglichkeiten.

(2) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse innerhalb der äußeren Klasse an der Stelle zu instanziieren, an der das Wort hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    hier
    public static class BKlasse{
    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();
d) Keine dieser Möglichkeiten.

(3) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse innerhalb der main()-Methode der äußeren Klasse an der Stelle zu instanziieren, an der das Wort hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    public class BKlasse{
    }
    public static void main(String args[ ]){
        hier
    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();

d) Keine dieser Möglichkeiten.

(4) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse innerhalb der main()-Methode der äußeren Klasse an der Stelle zu instanziieren, an der das Wort hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    public static class BKlasse{
    }
    public static void main(String args[ ]){
        hier
    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();
d) Keine dieser Möglichkeiten.

(5) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse außerhalb der äußeren Klasse zu instanziieren, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    public class BKlasse{

    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();
d) Keine dieser Möglichkeiten.

(6) Frage

Welche von untenstehenden Möglichkeiten gibt es, untenstehende innere Klasse außerhalb der äußeren Klasse zu instanziieren, ohne daß es zu einem Kompilierfehler kommt?

```
public class AKlasse {
    public static class BKlasse{

    }
}
```

a) BKlasse b = new BKlasse();
b) AKlasse.BKlasse b = new AKlasse().new BKlasse();
c) AKlasse.BKlasse b = new AKlasse.BKlasse();
d) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    AKlasse(){
        System.out.println(”A“);
    }
    public class BKlasse{
        BKlasse(){
            System.out.println“B”);
        }
    }
    public static void main(String args[ ]){
        AKlasse.BKlasse b = new AKlasse().new BKlasse()    }
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    AKlasse(){
        System.out.println(”A“);
    }
    public class BKlasse{
        BKlasse(){
            System.out.println“B”);
        }
    }
    public static void main(String args[ ]){
        AKlasse a = new AKlasse();
    }
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
	AKlasse(){
		System.out.println(”A“);
	}
	public class BKlasse{
		BKlasse(){
			System.out.println(“B”);
		}
	}
	public static void main(String args[ ]){
		BKlasse b = new BKlasse();
	}
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
	AKlasse(){
		System.out.println(”A“);
	}
	public static class BKlasse{
		BKlasse(){
			System.out.println(“B”);
		}
	}
	public static void main(String args[ ]){
		AKlasse.BKlasse b = new AKlasse.BKlasse();
	}
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
```

```
    AKlasse(){
        System.out.println("A");
    }
    public static class BKlasse{
        BKlasse(){
            System.out.println("B");
        }
    }
    public static void main(String args[ ]){
        AKlasse a = new AKlasse();
    }
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    AKlasse(){
        System.out.println("A");
    }
    public static class BKlasse{
        BKlasse(){
            System.out.println("B");
        }
    }
    public static void main(String args[ ]){
        BKlasse b = new BKlasse();
    }
}
```

a) A B
b) B
c) A
d) Kompilierfehler

(13) Frage

Welche von untenstehenden Aussagen sind korrekt?

a) Innere Klassen können statisch und nicht-statisch sein.
b) Die Modifier abstract oder final sind für innere Klassen erlaubt.
c) Es gibt keine inneren Interfaces.
d) Innere Interfaces sind implizit native.
e) Eine äußere Klasse kann sowohl eine innere Klasse als auch ein inneres Interface beinhalten.
f) Ein Interface kann keine innere Klasse beinhalten.

(14) Frage

Welche von untenstehenden Aussagen sind nicht korrekt?

a) Eine innere Klasse wird innerhalb einer anderen Klasse deklariert.
b) Innere Interfaces sind implizit static.
c) Innere Klassen sind static oder nicht static.
d) Innere Klassen gibt es sowohl in Interfaces als auch in Klassen.
e) Private und protected können Modifier für innere Klassen sein.
f) Keine dieser Möglichkeiten.

(15) Frage

Welche der untenstehenden Modifier sind zugelassen für innere Klassen?

a) native
b) synchronized
c) transient
d) strictfp
e) Keine dieser Möglichkeiten

(16) Frage

Welche der untenstehenden Modifier sind zugelassen für innere Klassen?

a) synchronized
b) final
c) protected
d) static
e) Keine dieser Möglichkeiten

(17) Frage

Welche der untenstehenden Modifier sind zugelassen für innere Klassen?

a) public
b) package local
c) private
d) abstract
e) Keine dieser Möglichkeiten

(18) Frage

Welche der untenstehenden Modifier sind zugelassen für innere Klassen?

a) synchronized
b) final
c) transient
d) private
e) Keine dieser Möglichkeiten

(19) Frage

Welche von untenstehenden Aussagen sind korrekt?

a) Statische innere Klassen können nicht auf statische Variablen der äußeren Klasse zugreifen.
b) Nicht-statische innere Klassen können nicht auf statische Variablen der äußeren Klasse zugreifen.
c) Es gibt keine statischen innere Klassen
d) Nicht-statische innere Klassen können nicht auf Variablen der äußeren Klasse zugreifen.
e) Statische innere Klassen können nicht auf Variablen der äußeren Klasse zugreifen.
f) Keine dieser Möglichkeiten.

(20) Frage

Welche von untenstehenden Aussagen sind nicht korrekt?

a) Statische innere Klassen können auf statische Variablen der äußeren Klasse zugreifen.
b) Nicht-statische innere Klassen können auf alle Variablen der äußeren Klasse zugreifen.
c) Es gibt statische und nicht-statische innere Klassen
d) Nicht-statische innere Klassen können auf statische Variablen der äußeren Klasse zugreifen.
e) Statische innere Klassen können statische Variablen beinhalten.
f) Keine dieser Möglichkeiten.

(21) Frage

Auf welche Variablen können Sie innerhalb der inneren Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht?

```
public class AKlasse {
    private static String a;
    private final int b = 1;
    public class BKlasse{
        BKlasse(){
            System.out.println(hier);
        }
    }
}
```

a) a
b) b
c) Keine dieser Möglichkeiten.

(22) Frage

Auf welche Variablen können Sie innerhalb der inneren Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht?

```
public class AKlasse {
	private static String a;
	private final int b = 1;
	public static class BKlasse{
		BKlasse(){
			System.out.println(hier);
		}
	}
}
```

a) a
b) b
c) Keine dieser Möglichkeiten.

(23) Frage

Wie können Sie an der Stelle, an der hier steht, auf die Variable private String a = “a“; der äußeren Klasse zugreifen?

```
public class AKlasse {
	private String a = ”a“;
	public class BKlasse{
		private String a = ”b“;
		BKlasse(){
			System.out.println(hier);
		}
	}
	public static void main (String args[ ]){
		AKlasse.BKlasse ab = new AKlasse().new BKlasse();
	}
}
```

a) super.a
b) this.a
c) a
d) AKlasse.this.a
e) Keine dieser Möglichkeiten.

(24) Frage

Wie können Sie an der Stelle, an der hier steht, auf die Variable private String a = “b“; der inneren Klasse zugreifen?

```
public class AKlasse {
	private String a = ”a“;
	public class BKlasse{
		private String a = ”b“;
		BKlasse(){
			System.out.println(hier);
		}
	}
	public static void main (String args[ ]){
```

```
        AKlasse.BKlasse ab = new AKlasse().new BKlasse();
    }
}
```

a) super.a
b) this.a
c) a
d) AKlasse.this.a
e) Keine dieser Möglichkeiten.

(25) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    public static class BKlasse{
        final static int a = 1;         Zeile 1
        static String b;                Zeile 2
        String c;                       Zeile 3
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(26) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    public class BKlasse{
        final static int a = 1;         Zeile 1
        static String b;                Zeile 2
        String c;                       Zeile 3
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(27) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
```

```
    static String b;                    Zeile 1
    public static class BKlasse{
        final static int a = 1;         Zeile 2
        String c;                       Zeile 3
        int i;                          Zeile 4
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

(28) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    static String b;                    Zeile 1
    public class BKlasse{
        final static int a = 1;         Zeile 2
        String c;                       Zeile 3
        int i;                          Zeile 4
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

(29) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    static String b;                    Zeile 1
    final static int a = 1;             Zeile 2
    public class BKlasse{
        String c;                       Zeile 3
        static int i;                   Zeile 4
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4

e) Keine dieser Möglichkeiten.

(30) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AKlasse {
    static String b;                    Zeile 1
    final static int a = 1;             Zeile 2
    public static class BKlasse{
        String c;                       Zeile 3
        static int i;                   Zeile 4
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Zeile 4
e) Keine dieser Möglichkeiten.

9. Lösungen

(1) Frage

a

(2) Frage

a

(3) Frage

b

(4) Frage

a, c

(5) Frage

b

(6) Frage

c

(7) Frage

a

(8) Frage
c

(9) Frage
d, Die Instanziierung BKlasse b = new BKlasse(); wäre nur möglich, wenn diese Klasse eine statische innere Klasse wäre.

(10) Frage
b

(11) Frage
c

(12) Frage
b

(13) Frage
a, b, e

(14) Frage
f, Achtung, in der Frage steht welche Aussagen sind nicht korrekt!

(15) Frage
e

(16) Frage
b, c, d

(17) Frage
a, b, c, d

(18) Frage
b, d

(19) Frage
f

(20) Frage
f, Achtung, in der Frage steht welche Aussagen sind nicht korrekt!

(21) Frage
a, b

(22) Frage
a

(23) Frage
d

(24) Frage
b, c

(25) Frage
d

(26) Frage
b

(27) Frage
e

(28) Frage
e

(29) Frage
d

(30) Frage
e

B. Lokale Klassen

1. Begriffserläuterung

Eine Klasse, die sich innerhalb eines Blocks befindet, nennt sich lokale Klasse. Ein Block kann eine Methode, ein Konstruktor oder ein lokaler Block sein. Es wird zwischen zwei Arten von Blöcken unterschieden: statische und nicht-statische. Eine lokale Klasse selbst kann nicht statisch sein und innerhalb der lokalen Klasse dürfen auch keine statische Deklarationen stehen.

2. Instanziierung im statischen und nicht-statischen Kontext

Lokale Klassen können nur innerhalb des Blocks, in dem sie sich befinden und nach der Klassendefinition der inneren Klasse instanziiert werden. Instanziierungsversuche an anderer Stelle führen zu Kompilierfehlern. Hierbei ist es egal, ob die Klasse in einem statischen oder nicht-statischen Block steht. Hier ein Beispiel einer Instanziierung einer lokalen Klasse in einer Methode:

```
public class LokaleKlasse {
        void  arbeite(){
                class InnereLokale {
                }
                InnereLokale innereLokale = new InnereLokale();
        }
}
```

3. Zugelassene Modifier für lokale Klassen

Als Modifier für lokale Klassen sind nur final und abstract möglich. Sie können nicht gleichzeitig verwendet werden, da abstract und final sich gegenseitig widersprechen (siehe Kapitel Modifier).

4. Zugriff auf Bestandteile der äußeren Klasse durch die lokale Klasse

a) Innerhalb eines statischen Blocks

Befindet sich die lokale Klasse in einem statischen Block, so hat die lokale Klasse nur Zugriff auf die statischen Variable der äußeren Klasse. In unserem Beispiel ist dies die Variable groesse.

```
public class LokaleKlasse {
        static String groesse = “klein“;
```

```
        static void  arbeite(){
                class InnereLokale {
                        InnereLokale(){
                                System.out.println(groesse);
                        }
                }
                InnereLokale innereLokale = new InnereLokale();
        }
        public static void main (String args[ ]){
                //statischer Zugriff auf eine statische Methode
                LokaleKlasse.arbeite();
        }
}
```

Ausgabe auf der Konsole:

klein

Verändern Sie nun in obenstehendem Beispiel die statische Variable groesse in eine nicht-statische Variable, kommt es zu einem Kompilierfehler.

```
 5 public class LokaleKlasse {
 6     String groesse = "klein";
 7     static void  arbeite(){
 8         class InnereLokale {
 9             InnereLokale(){
10                 System.out.println(groesse);
11             }
12         }
13         InnereLokale innereLokale = new InnereLokale();
14     }
15     public static void main (String args[]){
16         LokaleKlasse.arbeite();
17     }
18 }
```

b) Innerhalb eines nicht-statischen Blocks

Befindet sich die lokale Klasse in einem nicht-statischen Block, so kann die lokale Klasse auf alle Variablen der äußeren Klasse zugreifen.

```
public class LokaleKlasse {
        static String groesse = “klein“;
        String farbe = ”rot“;

        void  arbeite(){
                class InnereLokale {
                        InnereLokale(){
                                System.out.println(groesse);
                                System.out.println(farbe);
                        }
                }
```

```
                InnereLokale innereLokale = new InnereLokale();
        }
        public static void main (String args[ ]){
                LokaleKlasse lokaleKlasse = new LokaleKlasse();
                lokaleKlasse.arbeite();
        }
}
```

Ausgabe auf der Konsole:

klein
rot

5. Zugriff auf Bestandteile des Blocks

Auf welche Bestandteile des Blocks können Sie in lokalen Klassen zugreifen? Auf lokale Variablen des Blocks, die final sind, wobei kein Unterschied zwischen lokalen Klassen in statischen und nicht-statischen Blöcken besteht. Wollen Sie auf andere Variablen des Blocks zugreifen, kommt es zu einem Kompilierfehler. Zur Erinnerung lokale Variablen sind in untenstehendem Fall, die Variablen, die sich innerhalb der Methode arbeite() befinden, und diese gelten nur innerhalb der Methode und sie sind außerhalb der Methode unbekannt (Vgl. Kapitel Geltungsbereich von Variablen).

```
5  public class LokaleKlasse {
6      void  arbeite(){
7          //lokale Variablen groesse und farbe
8          String groesse = "klein";
9          final String farbe = "rot";
10         class InnereLokale {
11             InnereLokale(){
12                 System.out.println(groesse);
13                 System.out.println(farbe);
14             }
15         }
16         InnereLokale innereLokale = new InnereLokale();
17     }
18     public static void main (String args[]){
19         LokaleKlasse lokaleKlasse = new LokaleKlasse();
20         lokaleKlasse.arbeite();
21     }
22 }
```

6. Zugelassene Deklarationen innerhalb von lokale Klassen

Welche Bestandteile dürfen in lokalen Klassen stehen? Es sind die folgenden: nicht-statische Deklarationen und final static Felder. Existiert ein Unterschied zwischen lokalen Klassen in statischen und nicht-statischen Blöcken? Nein!

```
5  public class LokaleKlasse {
6      void  arbeite(){
7          class InnereLokale {
8              static String groesse = "klein";
9              final static String farbe = "rot";
10             String s = "gut";
11         }
12     }
13 }
```

7. Zusammenfassende Aufstellung: Lokale Klassen

Fassen wir das bisher Gesagte in untenstehender Tabelle zusammen:

	Zugriff auf Bestandteile der äußeren Klasse	**Zugriff auf Bestandteile des Blocks (z. B. Methode)**	**Mögliche Deklarationen innerhalb der lokalen Klasse**
Lokale Klasse innerhalb eines statischen Blocks	nur statische	nur lokale Variablen, die final sind	• nur nicht-statische • aber: final static
Lokale Klasse innerhalb eines nicht-statischen Blocks	alle		

8. Übungen

(1) Frage

Welche der untenstehenden Modifier sind für lokale Klassen möglich?

a) synchronized
b) final
c) protected
d) static
e) Keine dieser Möglichkeiten

(2) Frage

Welche der untenstehenden Modifier sind für lokale Klassen möglich?

a) public
b) package local
c) private
d) abstract
e) Keine dieser Möglichkeiten

(3) Frage

Welche der untenstehenden Modifier sind für lokale Klassen möglich?

a) synchronized
b) public
c) transient
d) private
e) Keine dieser Möglichkeiten

(4) Frage

In welcher Zeile tritt ein Kompilierfehler auf, wenn man folgendes Programm kompiliert und startet?

```
public class LokaleAußen {
    void neu(){
        class Lokale {
            String s1 = ”blau“;                          Zeile 1
            final static String s2 = ”blau“;             Zeile 2
            String s = “gelb”;                           Zeile 3
        }
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(5) Frage

In welcher Zeile tritt ein Kompilierfehler auf, wenn man folgendes Programm kompiliert und startet?

```
public class LokaleAußen {
    void neu(){
        class Lokale {
            String s1 = “blau“;                    Zeile 1
            static String s2 = “blau“;             Zeile 2
            String s = “gelb“;                     Zeile 3
        }
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(6) Frage

In welcher Zeile tritt ein Kompilierfehler auf, wenn man folgendes Programm kompiliert und startet?

```
public class LokaleAußen {
    static void neu(){
        class Lokale {
            String s1 = ”blau”;                    Zeile 1
            final static String s2 = ”blau”;       Zeile 2
            String s = “gelb”;                     Zeile 3
        }
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(7) Frage

In welcher Zeile tritt ein Kompilierfehler auf, wenn man folgendes Programm kompiliert und startet?

```
public class LokaleAußen {
    void neu(){
        class Lokale {
            static String s1 = ”blau“;             Zeile 1
            final static String s2 = ”blau“;       Zeile 2
            String s = “gelb”;                     Zeile 3
        }
    }
}
```

a) Zeile 1
b) Zeile 2
c) Zeile 3
d) Keine dieser Möglichkeiten.

(8) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
    void neu(){
        String s1 = ”blau”;
        final String s2 = ”rot”;
        String s = ”gelb”;
            class Lokale {
                Lokale(){
                    System.out.println(hier);
                }
            }
    }
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(9) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
    static void neu(){
        String s1 = ”blau”;
        final String s2 = ”lila”;
        String s = ”gelb”;
            class Lokale {
                Lokale(){
                    System.out.println(hier);
                }
            }
    }
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(10) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
    void neu(){
        String s1 = “blau”;
        String s2 = ”rot”;
        String s = ”gelb”;
            class Lokale {
                Lokale(){
                    System.out.println(hier);
                }
            }
    }
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(11) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
    static String s1 = ”lila”;
    final String s2 = ”blau”;
    String s = ”gelb”;
    void neu(){
        class Lokale {
            Lokale(){
                System.out.println(hier);
            }
        }
    }
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(12) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
	static String s1 = ”lila”;
	final String s2 = ”blau”;
	String s = ”gelb”;
	static void neu(){
		class Lokale {
			Lokale(){
				System.out.println(hier);
			}
		}
	}
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(13) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
	String s1 = ”lila”;
	final String s2 = ”blau”;
	String s = ”gelb”;
	static void neu(){
		class Lokale {
			Lokale(){
				System.out.println(hier);
			}
		}
	}
}
```

a) s
b) s1
c) s2
d) Keine dieser Möglichkeiten.

(14) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
	static String a = ”eins”;
	static final String b = ”zwei”;
	String c = “drei”;
	void neu(){
```

```
            class Lokale {
                Lokale(){
                    System.out.println(hier);
                }
            }
        }
}
```

a) a
b) b
c) c
d) Keine dieser Möglichkeiten.

(15) Frage

Auf welche Variablen können Sie innerhalb der lokalen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class LokaleAußen {
    static String a = ”eins”;
    static final String b = ”zwei”;
    String c = “drei”;
    static void neu(){
        class Lokale {
            Lokale(){
                System.out.println(hier);
            }
        }
    }
}
```

a) a
b) b
c) c
d) Keine dieser Möglichkeiten.

9. Lösungen

(1) Frage

b

(2) Frage

d

(3) Frage

e

(4) Frage
d

(5) Frage
b

(6) Frage
d

(7) Frage
a

(8) Frage
c

(9) Frage
c

(10) Frage
d

(11) Frage
a, b, c

(12) Frage
b

(13) Frage
d

(14) Frage
a, b, c

(15) Frage
a, b

C. Anonyme Klassen

1. Begriffserläuterung

Anonyme Klassen werden hauptsächlich in der Programmierung von graphischen Oberflächen (GUI) verwendet und sind dort weit verbreitet. Sie dienen dazu, sogenannte Events zu implementieren. Events lösen Aktionen aus: Klicken Sie auf einen Button, öffnet sich ein Fenster. Oder: Sie erhalten eine Fehlermeldung, nachdem Sie etwas falsch gemacht haben.

2. Erstellen von anonymen Klassen

Anonyme Klassen können in Klassen, in Methoden oder als Argument eines Methodenaufrufs definiert werden. Sie können entweder auf Grundlage einer Klasse oder eines Interface erstellt werden, es dürfen aber nicht die Begriffe implements und extends dortstehen. Anonyme Klasse haben implizit eine extends-Beziehung zur Klasse Object und sie sind implizit final. Außerdem können weder die Modifier abstract noch static verwendet werden.

Bei anonymen Klassen werden zwei Schritte in einem durchgeführt: Es wird eine Klasse ohne Namen definiert und gleichzeitig eine Instanz dieser Klasse erstellt.

a) Erstellen von anonymen Klassen auf Basis von Klassen

Nehmen wir an, Sie haben untenstehende Klasse als Basis für eine anonyme Klasse:

```
public class Zimmer {
	public void wohne() {
		System.out.println("Ich wohne in einem Zimmer");
	}
}
```

Nun können Sie in der untenstehenden Klasse Haus eine anonyme Klasse definieren, die als Basis die Klasse Zimmer hat. Anonyme Klassen haben am Ende ein Semikolon, wenn Sie innerhalb einer Klasse oder einer Methode erstellt werden. Es wird bei diesem Vorgang implizit eine Subklasse der Klasse Zimmer erzeugt, und gleichzeitig kann deren Methode wohne() überschrieben werden. Sie muß es aber nicht! Sie kann auch weggelassen werden, ohne daß es zu einem Kompilierfehler kommt. Für anonyme Klassen macht es keinen Sinn eigene Methoden zu haben.

```
public class Haus {
	Zimmer z = new Zimmer(){
		public void wohne() {
			System.out.println("Ich wohne in einem Haus");
```

```
                }
        }; //Achtung! Strichpunkt nicht vergessen!
}
```

Fehlt am Ende der anonymen Klasse der Strichpunkt, kommt es zu einem Kompilierfehler.

```
4  public class Haus {
5      Zimmer z = new Zimmer(){
6          public void wohne() {
7              System.out.println("Ich wohne in einem Haus");
8          }
9      }
10 }
```

Eine Anonyme Klasse verhält sich wie eine gewöhnliche Subklasse einer Superklasse, da eine implizite extends-Beziehung existiert. Aber jetzt kommt eine Einschränkung: eine anonyme Klasse darf keinen Konstruktor haben!

b) <u>Erstellen von anonymen Klassen auf Basis von Interfaces</u>

Nehmen wir die zweite Möglichkeit, eine anonyme Klasse zu erstellen: auf Basis eines Interfaces (vgl. Kapitel Interfaces). Wir greifen als Beispiel auf das bereits existierende Interface Runnable zurück (vgl. Kapitel Threads), das die Methode public void run() beinhaltet, die implementiert werden muß. Erstellt man eine Anonyme Klasse, muß diese eine Methode public void run() enthalten. Es wird automatisch eine implizite implements-Beziehung erzeugt. Achten Sie bitte darauf, daß am Ende der anonymen Klasse ein Strichpunkt steht, ohne diesen Strichpunkt kommt es zu einem Kompilierfehler.

```
public class AnonymeKlasse {
    Runnable ru = new Runnable(){
        public void run() {
            System.out.println("Ich laufe!");
        }
    };  //Achtung! Strichpunkt nicht vergessen!
}
```

Statt Runnable ru = new Runnable() nur new Runnable() zu schreiben, reicht nicht aus, dies würde zu einem Kompilierfehler führen.

c) Erstellen von anonymen Klassen in Methoden

(1) Erstellen von anonymen Klassen auf Basis von Klassen innerhalb einer Methode

Wie kann man eine anonyme Klasse auf Basis einer Klasse in einer Methode formulieren? Es gibt zwei verschiedene Möglichkeiten, ohne daß es zu einem Kompilierfehler kommt:

```
public class Haus {
	public static void main(String[ ] args) {
		new Zimmer(); //Achtung! Strichpunkt nicht vergessen!
		Zimmer z = new Zimmer(); //Achtung! Strichpunkt nicht vergessen!
	}
}
```

(2) Erstellen von anonymen Klassen auf Basis von inneren Klassen innerhalb einer Methode

Sie fragen sich sicherlich, ob eine anonyme Klasse auf Basis von inneren Klassen in einer Methode definiert werden kann? Natürlich! Es geht sowohl mit statischen als auch mit nicht-statischen inneren Klassen. Erinnern wir uns an die Beispiele aus dem Kapitel zum Thema Innere Klassen und lassen Sie uns zwischen statischen und nicht-statischen inneren Klassen unterscheiden.

Sie können eine anonyme Klasse einer nicht-statischen inneren Klasse in der main()-Methode der äußeren Klasse wie untenstehend definieren. Wollen Sie das gleiche außerhalb der äußeren Klasse in einer Methode einer anderen Klasse tun, funktioniert dies genauso.

```
public class Pfirsich {
	public class Kern{
	}
	public static void main(String args[ ]){
		new Pfirsich().new Kern();
	}
}
```

Welche Unterschiede gibt es hierzu bei statischen inneren Klassen? Sie können anonyme Klassen auf zwei Arten erstellen und es gibt eine Abweichung in der Syntax: Es fehlt die Klammer nach new Pflanze.

```
public class Pflanze {
	public static class Blatt{
	}
```

```
        public static void main(String[ ] args) {
                new Pflanze.Blatt();
                new Blatt();
        }
}
```

Wie formuliert man eine anonyme Klasse in einer Methode außerhalb der äußeren Klasse? Die Antwort ist die folgende: dies geht nur mit new Pflanze.Blatt();.

(3) Erstellen von anonymen Klassen auf Basis von einem Interface innerhalb einer Methode

Untenstehend sehen Sie zwei verschiedene Wege eine anonyme Klasse auf Basis von einem Interface in einer Methode zu erzeugen, ohne daß es zu einem Kompilierfehler kommt.

Dies ist die erste Möglichkeit:

```
public class AnonymeKlasse {
    void arbeite() {
        new Runnable() {
            public void run() {
                System.out.println("Ich laufe!");
            }
        };  //Achtung! Strichpunkt nicht vergessen!
    }
}
```

Dies ist die zweite Möglichkeit:

```
public class AnonymeKlasse {
    void arbeite() {
        Runnable r = new Runnable() {
            public void run() {
                System.out.println("Ich laufe!");
            }
        };  //Achtung! Strichpunkt nicht vergessen!
    }
}
```

d) Erstellen von anonymen Klassen als Argument eines Methodenaufrufs

Wollen Sie eine anonyme Klasse als Argument eines Methodenaufrufs definieren, ist die Syntax anders: Die anonyme Klasse wird nicht mit einem Strichpunkt beendet. Der Semikolon, den Sie unten sehen, gehört zur Methode und schließt die Methode ab. In unserem Beispiel wird einer Schaltfläche, die Aktion Dokument öffnen zugewiesen. Die Basis für die untenstehende anonyme Klasse ist das Interface ActionListener mit seiner Methode public void actionPerformed(Action e),

die auf jeden Fall implementiert werden muß. Das Interface ActionListener befindet sich im Paket java.awt.event.ActionListener.

```
button.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                          openFile();
                    }
               }  // Achtung! Hier kein Strichpunkt!
);  //Achtung! Strichpunkt zum Abschließen der Methode!
```

(1) Ein kleines Anwendungsbeispiel

Da dies sicherlich sehr abstrakt klingt, jetzt ein kleines Anwendungsbeispiel aus dem Bereich der Programmierung von graphischen Oberflächen. Sobald Sie untenstehendes Programm starten, wird ein Fenster mit einem Button, mit der Aufschrift „Drück mich!“ geöffnet!

Kommen Sie der Aufforderung nach, öffnet unser ActionListener eine Message-Box mit der Meldung „Mach mich wieder zu!“.

Drücken Sie nun auf OK wird das Fenster wieder geschlossen. Und es erscheint wieder der Button Drück mich!.

Hier der dazugehörige Code:

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;


public class EigenerActionListener extends JFrame implements ActionListener{

	public EigenerActionListener() {
		setSize(600, 400);
		setVisible(true);
		getContentPane().setLayout(new BorderLayout());
		JButton button = new JButton();
		button.setText("Drück mich!");
		button.setActionCommand(„actionCommand");
		button.addActionListener( this );
		getContentPane().add(button, BorderLayout.CENTER);
		button.addActionListener(
				new ActionListener() {
					public void actionPerformed(ActionEvent e) {JOptionPane.
showMessageDialog(EigenerActionListener.this, "Mach mich wieder zu!", "Fehlermeldung",
JOptionPane.INFORMATION_MESSAGE);
					}
				}
		);
		pack();
	}
		public void actionPerformed(ActionEvent e) {
		}

		public static void main (String args[ ]) {
			EigenerActionListener e = new EigenerActionListener();
		}
}
```

Nun noch eine Bemerkung zu dem Ausdruck EigenerActionListener.this. Hiermit greift man auf die äußere Klasse zu. Nur this geht nicht, da man so nur auf die anonyme Klasse zugreifen würde (vgl. Kapitel Innere Klassen).

3. Zugriff auf Bestandteile der äußeren Klasse durch die anonyme Klasse

Anonyme Klassen verhalten sich in bezug auf ihren Zugriff zu Bestandteilen des Blocks und der äußeren Klasse und der möglichen Deklarationen innerhalb der anonymen Klasse wie lokale Klassen (vgl. Kapitel lokale Klassen).

	Zugriff auf Bestandteile der äußeren Klasse	**Zugriff auf Bestandteile des Blocks (z. B. Methode)**	**Mögliche Deklarationen innerhalb der anonymen Klasse**
Anonyme Klasse innerhalb eines statischen Blocks	nur statische	nur lokale Variablen, die final sind	• nur nicht-statische • aber: final static
Anonyme Klasse innerhalb eines nicht-statischen Blocks	alle		

4. Übungen

(1) Frage

Welche von untenstehenden Aussagen sind korrekt?

a) Anonyme Klassen haben Konstruktoren.
b) Anonyme Klassen sind implizit static und final.
c) Eine anonyme Klasse kann keine implizite implements-Beziehung zu einem Interface haben.
d) Anonyme Klassen können implements- und extends-Angabe haben.
e) Anonyme Klassen haben implizit keine extends-Beziehung zur Klasse Object.
f) Keine dieser Möglichkeiten.

(2) Frage

Welche von untenstehenden Aussagen sind korrekt?

a) Innerhalb von anonyme Klassen gibt es keine Methoden.
b) Anonyme Klassen sind implizit final.
c) Eine anonyme Klasse kann nicht in einer Methode stehen.
d) Anonyme Klassen können keine implements- und extends-Angabe haben.
e) Anonyme Klassen haben implizit eine extends-Beziehung zur Klasse Object.
f) Keine dieser Möglichkeiten.

(3) Frage

Was muß an der Stelle, an der hier steht, stehen?

```
public class Anonyme {
	hier {
		public void run() {
		}
	};
}
```

a) new Runnable()
b) Runnable r = new Runnable()
c) new Runnable
d) Runnable r = new Runnable

(4) Frage

Was muß an der Stelle, an der hier steht, stehen?

```
public class Anonyme {
	void a() {
		hier {
			public void run() {
			}
		};
	}
```

```
}
```

a) new Runnable()
b) Runnable r = new Runnable()
c) new Runnable
d) Runnable r = new Runnable

(5) Frage

Welche von untenstehenden Programmauszügen führt nicht zu einem Kompilierfehler?

a)
```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
        }
    }
);
```
b)
```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
        }
    };
)
```
c)
```
button.addActionListener(new ActionListener() {
    }
);
```
d)
```
button.addActionListener() {new ActionListener()
        public void actionPerformed(ActionEvent e) {
        }
    }
);
```
e) Keine dieser Möglichkeiten.

(6) Frage

Welche von untenstehenden Programmauszügen führt nicht zu einem Kompilierfehler?

a)
```
button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("Ich bin ein Event!");
        }
    };
)
```
b)
```
button.addActionListener(){new ActionListener()
        public void actionPerformed(ActionEvent e) {
            System.out.println("Ich bin ein Event!");
        }
    };
);
```
c)
```
button.addActionListener(new ActionListener() {
```

```
                    public void actionPerformed(ActionEvent e) {
                        System.out.println("Ich bin ein Event!");
                    }
                }
    );
d)  button.addActionListener {new ActionListener()
                    public void actionPerformed(ActionEvent e) {
                        System.out.println("Ich bin ein Event!");
                    }
                }
    );
e)  Keine dieser Möglichkeiten.
```

(7) Frage

Welche von untenstehenden Programmauszügen führt nicht zu einem Kompilierfehler?

```
a)  void m() {
            new Runnable() {
                public void run() {
                }
            };
    }
b)  void m() {
            Runnable r = new Runnable() {
                public void run() {
                }
            };
    }
c)  void m() {
            Runnable r = new Runnable() {
                public void run() {
                }
            }
    };
d)  void m() {
            Runnable r = new Runnable() {
            };
    }
e)  Keine dieser Möglichkeiten.
```

(8) Frage

Welche von untenstehenden Programmauszügen führt nicht zu einem Kompilierfehler?

```
a)  public class Anonyme {
            new Runnable() {
                public void run() {
```

```
                    }
                };
        }
b)      public class Anonyme {
                Runnable r = new Runnable() {
                    public void run() {
                    }
                };
        }
c)      public class Anonyme {
                Runnable r = new Runnable() {
                    public void run() {
                    }
                }
        };
d)      public class Anonyme {
                Runnable r = new Runnable() {
                };
        }
e)      Keine dieser Möglichkeiten.
```

(9) Frage

Auf welche Variablen können Sie innerhalb der anonymen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class Anonyme {
    int i;
    void m() {
        int a;
        final String b = ”String“;
        Runnable r = new Runnable() {
            String c;
            public void run() {
                System.out.println(hier);
            }
        };
    }
}
```

a) i
b) a
c) b
d) c
e) Keine dieser Möglichkeiten.

(10) Frage

Auf welche Variablen können Sie innerhalb der anonymen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem

Kompilierfehler kommt?

```
public class Anonyme {
	int i;
	static void m() {
		int a;
		final String b = ”String“;
		Runnable r = new Runnable() {
			String c;
			public void run() {
				System.out.println(hier);
			}
		};
	}
}
```

a) i
b) a
c) b
d) c
e) Keine dieser Möglichkeiten.

(11) Frage

Auf welche Variablen können Sie innerhalb der anonymen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class Anonyme {
	int i;
	static void m() {
		int a;
		final String b = ”String“;
		String c;
		Runnable r = new Runnable() {
			public void run() {
				System.out.println(hier);
			}
		};
	}
}
```

a) i
b) a
c) b
d) c
e) Keine dieser Möglichkeiten.

(12) Frage

Auf welche Variablen können Sie innerhalb der anonymen Klasse zugreifen, sprich welche Variablen können Sie dort einsetzen, wo hier steht, ohne daß es zu einem Kompilierfehler kommt?

```
public class Anonyme {
	int i;
	void m() {
		int a;
		String b = ”String“;
		String c;
		Runnable r = new Runnable() {
			public void run() {
				System.out.println(hier);
			}
		};
	}
}
```

a) i
b) a
c) b
d) c
e) Keine dieser Möglichkeiten.

5. Lösungen

(1) Frage
f

(2) Frage
b, d, e

(3) Frage
b

(4) Frage
a, b

(5) Frage
a

(6) Frage
c

(7) Frage
a, b

(8) Frage
b

(9) Frage
a, c, d

(10) Frage
c, d

(11) Frage
c

(12) Frage
a

XV. Neuerungen in Java 2 Plattform 5.0

A. Kontrollstrukturen

In Java 2 Plattform 5.0 ist zu den bisherigen for-Schleifen die sogenannte erweiterte for-Schleife hinzugekommen.

1. Die erweiterte for-Schleife

Welchen Vorteil bietet die erweiterte for-Schleife gegenüber der bisherigen? Sie vereinfacht es Ihnen, Elemente aus einem Array auszulesen. In untenstehendem Beispiel werden die Unterschiede der beiden for-Schleifen gegenübergestellt: Das Ergebnis ist bei beiden identisch, aber die Syntax hat sich verändert. Das int i : a in der zweiten for-Schleife hat zur Folge, daß das Array a ausgelesen wird und jeweils pro Schleifendurchlauf ein Element des Arrays in der Variablen i gespeichert wird. So gibt nun System.out.println(i); alle Elemente des Arrays aus. Bitte beachten Sie, daß das Array a vom Typ int ist, so muß auch die Variable i vom Typ int sein.

```
public class ForSchleife {
	public static void main(String[ ] args) {
		int[ ] a = {1, 2, 3};

		for(int i = 0; i < a.length; i++){
			System.out.println(a[i]);
		}

		//Datentyp von i und von a muß gleich sein
		for(int i:a){
			System.out.println(i);
		}
	}
}
```

Ausgabe auf der Konsole:

```
1
2
3
1
2
3
```

Entspricht der Rückgabetyp von i nicht dem von a, so kommt es zu einem Kompilierfehler (siehe unten). Die Variable i ist vom Typ her byte und somit kleiner als int (vgl. Kapitel Implizite und explizite Typanpassung).

```
5  public class ForSchleife {
6      public static void main(String[] args) {
7          int[] a = {1, 2, 3};
8
9          for (byte i:a){
10             System.out.println(i);
11         }
12     }
13 }
```

Eine weitere Frage: Was passiert, wenn der Rückgabetyp Variable i vom Typ her long ist? Die Antwort ist einfach: Der primitive Typ long ist größer als int und es findet automatisch eine implizite Typanpassung statt.

```
public class ForSchleife {
        public static void main(String[ ] args) {
                int[ ] a = {1, 2, 3};

                for (long i:a){
                        System.out.println(i);
                }
        }
}
```

Betrachten wir nun Arrays, die Objekte beinhalten. In untenstehendem Beispiel verwenden wir ein String-Array s, das der Variablen i, ebenfalls vom Typ String, übergeben wird.

```
public class ForSchleife {
        public static void main(String[ ] args) {
                String[ ] s = {new String(”1“), new String(”2“), new String(”3“)};

                for (String i:s){
                        System.out.println(i);
                }
        }
}
```

Ändern wir nun den Typ String von i in den Typ Object, ist dies ohne Probleme möglich, da die Klasse Object die Mutter aller Klassen ist (Vgl. Kapitel Subtyping und Casting von Objekten).

```
public class ForSchleife {
        public static void main(String[ ] args) {
                String[ ] s = {new String(”1“), new String(”2“), new String(”3“)};
```

```
                for (Object i:s){
                        System.out.println(i);
                }
        }
}
```

Wollen Sie allerdings ein Objekt der Klasse Object einem String zuweisen, führt dies, wie Sie unten sehen können, zu einem Kompilierfehler.

```
5  public class ForSchleife {
6      public static void main(String[] args) {
7          Object[] s = {new Object(), new Object(), new Object()};
8
9          for (String i:s){
10             System.out.println(i);
11         }
12     }
13 }
```

Weiter unten werden Sie sehen, daß uns diese neue for-Schleife auch gute Dienste beim Auslesen von Collections/Generics leistet.

2. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class FSchleife {
	public static void main(String[ ] args) {
		String[ ] s = {new String(”44“), new String(”56“), new String(”73“)};

		for (String i:s){
			System.out.println(i);
		}
	}
}
```

a) Keine Ausgabe.
b) 44 56 73
c) Kompilierfehler

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class FSchleife {
	public static void main(String[ ] args) {
		String[ ] s = {new String(”44“), new String(”56“), new String(”73“)};

		for (Object i:s){
			System.out.println(i);
		}
	}
}
```

a) Keine Ausgabe.
b) 44 56 73
c) Kompilierfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class FSchleife {
	public static void main(String[ ] args) {
		Object[ ] s = {new Object(), new Object(), new Object()};

		for (String i:s){
			System.out.println(i);
		}
	}
}
```

a) null null null
b) Keine Ausgabe
c) Kompiliefehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class FSchleife {
	public static void main(String[ ] args) {
		Object[ ] s = {new Object(), new Object(), new Object()};

		for (Object i:s){
			System.out.println(i);
		}
	}
}
```

a) null null null
b) Keine Ausgabe
c) Kompiliefehler

3. Lösungen

(1) Frage

b

(2) Frage

b

(3) Frage

c

(4) Frage

b

B. Autoboxing

1. Begriffsdefinition

Erinnern wir uns an das Kapitel Wrapper-Klassen: Die Wrapper-Klassen waren dazu da, primitive Datentypen in Objekte umzuwandeln. Seit Java 1.5 muß man dies nicht mehr tun, da die Typumwandlung nun automatisch geschieht. Dieser Vorgang nennt sich Autoboxing. Wird ein Wrapper-Objekt automatisch in einen primitiven Datentyp überführt, spricht man von Unboxing. Das englische Wort box kann mit Schachtel übersetzt werden. Dies ist eine begriffliche Entsprechung zu den englischen Verben wrap und unwrap, die die Bedeutung von ein- und auspacken haben (siehe Kapitel Die Wrapper-Klassen). Das automatische Aufblasen von primitiven Datentypen in Objekte kostet sehr viel Rechenzeit, wohingegen der umgekehrte Vorgang zu keinerlei Verzögerungen führt. Wie Sie unten sehen können, besteht jetzt die Möglichkeit einem Wrapper-Objekt, einen primitiven Datentyp direkt zuzuweisen:

```
public class AutoboxingAllgemein {
        public static void main(String[ ] args) {
                //seither: Instanziierung eines Integer-Wrapper-Objektes
                Integer i = new Integer(10);
                /*jetzt: Zuweisung primitiver Datentyp int zu einem
                 einen Integer-Wrapper*/
                Integer i1 = 10;
                //seither: Instanziierung eines Boolean-Wrapper-Objektes
                Boolean b = new Boolean(”true“);
                /*jetzt: Zuweisung primitiver Datentyp boolean zu einem
                 einen Boolean-Wrapper*/
                Boolean b1 = true;
        }
}
```

Weisen Sie einem Boolean-Wrapper einen primitiven Datentyp zu, kann dieser nur zwei Ausprägungen annehmen: true und false. So würde es zu einem Kompilierfehler kommen, wenn Sie dem Boolean-Wrapper einen String-Literal, wie z. B. Boolean b = “true“;, zuweisen würden.

Wenn Sie ein paar Seiten zum Kapitel Wrapper-Klassen zurückblättern, werden Sie feststellen, daß es zusätzliche Methoden zum Umwandeln von primitiven Datentypen in Objekte und umgekehrt, wie z. B. xxxValue() oder valueOf(String s), gibt. Diese Methoden können nach wie vor problemlos verwendet werden.

2. Autoboxing und Operatoren

Bei der Verwendung von allen Operatoren werden Wrapper-Objekte automatisch in Ihre primitive Typen überführt. Sollte der Rückgabetyp wieder ein Wrapper-Objekt sein, findet dann anschließend eine automatische Anpassung (Autoboxing) statt.

Hier ein Beispiel zu Boolean-Objekten:

```
public class AutoboxingBooleanOperatoren {
	public static void main(String[ ] args) {
		Boolean b = true;
		Boolean b1 = true;
		boolean b2 = false;
		Boolean b3 = b || b1 && b2;
		boolean b4 = b || b1 && b2;
		System.out.println(b3);
		System.out.println(b4);
	}
}
```

Ausgabe auf der Konsole:

```
true
true
```

Nehmen wir ein weiteres Beispiel zur Verwendung von Operatoren: Eine Berechnung mit einem int und einem Integer-Objekt:

```
public class AutoboxingIntegerOperatoren {
	public static void main(String[ ] args) {
		Integer in = 500;
		Integer i1 = new Integer(”500“);
		int i = 19;
		int ergebnis = i + in + i1;
		Integer ergebnisInteger = i + in + i1;
		System.out.println(ergebnis);
		System.out.println(ergebnisInteger);
	}
}
```

Ausgabe auf der Konsole:

```
1019
1019
```

3. Probleme und Besonderheiten

Im Folgenden möchte ich Sie auf einige Probleme und Besonderheiten aufmerksam machen, die die automatische Umwandlung mit sich bringt.

a) Zuweisung von null

Der Wert null ist kein gültiger Wert für einen primitiven Datentyp. Bei einer Zuweisung von null zu einem primitiven Datentyp kommt es zu einem Kompilierfehler und

falls Sie das Programm starten würden, würde eine NullPointerException geworfen werden.

```
2
3 public class AutoboxingNull {
4     public static void main(String[] args) {
5         int i = null;
6     }
7
8 }
```

b) Autoboxing: Vergleich von Objekten und primitiven Datentypen

(1) Vergleich von primitiven Datentypen mit Objekten der Wrapper-Klasse

Erinnern wir uns an das Kapitel Vergleich von Objekten und primitiven Datentypen: Was hat sich durch die Einführung des Autoboxings geändert? Seither konnten primitive Datentypen nicht mit ihren entsprechenden Wrapper-Objekten verglichen werden. Es kam sowohl beim Vergleich mit dem Vergleichsoperator (==) als auch mit der equals()-Methode zu einem Kompilierfehler. Nun können Sie beides, und das Ergebnis des Vergleichs mit dem Vergleichsoperator == wird vom primitiven Datentyp bestimmt: Das Integer-Objekt wird in eine primitive int-Zahl „unboxed“. Wohingegen bei der equals()-Methode die int-Zahl automatisch in ein Wrapper-Objekt umgewandelt wird. So erhalten Sie beides Mal das Ergebnis true:

```
public class AutoboxingVergleich {
        public static void main(String[ ] args) {
                int i = 5;
                Integer in = new Integer(”5“);
                System.out.println(i == in);
                System.out.println(in.equals(i));
        }
}
```

Ausgabe auf der Konsole:

```
true
true
```

Aber: Es kommt nach wie vor zu einem Kompilierfehler, wenn Sie versuchen eine Methode auf einem primitiven Datentyp auszuführen, da primitive Datentypen keine Methoden besitzen. Der primitive Datentyp wird in diesem Fall nicht automatisch in ein Wrapper-Objekt umgewandelt.

```
3  public class AutoboxingVergleich {
4      public static void main(String[] args) {
5          int i = 5;
6          Integer in = new Integer("5");
7          System.out.println(i == in);
8          System.out.println(i.equals(in));
9      }
10 }
```

(2) Unveränderbare Objekte (Immutable Objects): Vergleich von Byte-, Short-, und Integer-Objekten

Welche Besonderheiten sind beim Vergleich von Integer-Objekten zu beachten, denen ein primitiver int zugewiesen worden ist? Die int-Werte im Intervall von - 128 bis + 127 werden in einem Cache (Pool) als Immutable Objects gespeichert und nur einmal erzeugt, um Rechenzeit beim Autoboxing von int zu Integer einzusparen. Entsprechendes gilt bei Byte- und Short-Objekten. Wir erinnern uns in diesem Zusammenhang an die String-Literale, die es auch nur einmal im String-Pool gab (siehe Kapitel Die Klasse String und Kapitel Vergleich von Strings). String-Literale werden allerdings nicht in Objekte umgewandelt.

So erhalten Sie bei der Zahl -128 wider Erwarten unterschiedliche Ergebnisse: Beim Vergleich von Integer-Objekten, denen primitive int-Werte zugewiesen worden sind, erhalten Sie sowohl beim Vergleich mit dem Vergleichsoperator als auch beim Vergleich mit der equals()-Methode true als Ergebnis, da es diesen int-Wert im Cache nur einmal gibt. Wohingegen bei Integer-Objekten nach wie vor beim Vergleichsoperator (==) die Referenzen verglichen werden und nicht der Inhalt (siehe Kapitel Vergleich von Objekten und primitive Datentypen), so erhalten Sie false als Resultat.

```
public class AutoboxingInteger {
        public static void main(String[ ] args) {
                Integer i = -128;
                Integer in = -128;
                Integer i1 = new Integer(”-128“);
                Integer in1 = new Integer (”-128“);
                System.out.println(i == in);
                System.out.println(i.equals(in));
                System.out.println(i1 == in1);
                System.out.println(i1.equals(in1));
        }
}
```

Ausgabe auf der Konsole:

```
true
true
false
true
```

Bei der Zahl “-129” erhalten Sie wieder gleiche Ergebnisse: Bei Zahlen außerhalb dieses Pools werden beim Vergleichsoperator (==), wie zu erwarten, wieder die Referenzen miteinander verglichen.

```
public class AutoboxingInteger {
        public static void main(String[ ] args) {
                Integer i = -129;
                Integer in = -129;
                Integer i1 = new Integer(”-129“);
                Integer in1 = new Integer (”-129“);
                System.out.println(i == in);
                System.out.println(i.equals(in));
                System.out.println(i1 == in1);
                System.out.println(i1.equals(in1));
        }
}
```

Ausgabe auf der Konsole:

```
false
true
false
true
```

Vergleichen Sie dahingegen ein Integer-Objekt mit einem Integer-Objekt, dem ein primitiver Datentyp zugewiesen worden ist, werden wieder die Referenzen verglichen und nicht der Inhalt. Dieses Verhalten ist vollkommen unabhängig von der Größe der Zahl.

```
public class AutoboxingInteger {
        public static void main(String[ ] args) {
                Integer in = 5;
                Integer i1 = new Integer(“5”);
                System.out.println(in == i1);
                System.out.println(i1.equals(in));
        }
}
```

Ausgabe auf der Konsole:

```
false
true
```

(3) Unveränderbare Objekte (Immutable Objects): Vergleich von Boolean-Objekten

Ebenfalls im Pool gespeicherte Unveränderbare Objekte sind die boolean-Werte

true and false, was bei untenstehendem Vergleich mit dem Vergleichsoperator eine Ausgabe von true zur Folge hat.

```
public class AutoboxingBoolean {
	public static void main(String[ ] args) {
		Boolean b = true;
		Boolean b1 = true;

		System.out.println(b == b1);
		System.out.println(b.equals(b1));
	}
}
```

Ausgabe auf der Konsole:

```
true
true
```

(4) Zusammenfassung Immutable Objects

Hier nun eine Liste aller primitiven Datentypen, die als Unveränderbare Objekte im Cache gespeichert werden:

- alle byte-Werte
- short-Werte von –128 bis + 127
- int-Werte –128 bis + 127
- char-Werte von \u0000 bis \u007F
- boolean-Werte true und false

c) Autoboxing und Methoden

Wie Sie unten sehen können, ist Vorsicht geboten in bezug auf Methoden, denen Parameter übergeben werden: Oberstes Gebot ist hierbei die Kompatibilität zu Java 1.4. Betrachten wir nun genauer, wann welche Methode aufgerufen wird: Gibt es untenstehende drei überschriebene Methoden, und es wird eine Methode mit einem int-Parameter aufgerufen, so wird auch nur die überschriebene Methode mit dem int-Parameter durchgeführt. Hier findet kein Autoboxing statt, da es eine entsprechende Methode gibt.

```
public class AutoboxingMethoden {
	public void schreiben(long i) {
		System.out.println(”long“);
	}

	public void schreiben(int i) {
		System.out.println(”int“);
```

```
        }

        public void schreiben(Integer i) {
                System.out.println("Integer");
        }

        public static void main(String[ ] args) {
                AutoboxingMethoden a = new AutoboxingMethoden();
                // Der Methode wird ein int übergeben
                a.schreiben(9);
        }
}
```

Ausgabe auf der Konsole:

int

Gibt es allerdings keine entsprechende Methode, sondern nur noch überschriebene Methoden mit einem long- und einem Integer-Parameter, wird wider Erwarten die Methode mit dem long-Parameter aufgerufen. Es findet also kein Autoboxing statt, sondern eine implizite Typanpassung von int zu long (vgl. Kapitel Zuweisungen und Typumwandlung).

```
public class AutoboxingMethoden {
        public void schreiben(long i) {
                System.out.println("long");
        }

        public void schreiben(Integer i) {
                System.out.println("Integer");
        }

        public static void main(String[ ] args) {
                AutoboxingMethoden a = new AutoboxingMethoden();
                // Der Methode wird ein int übergeben
                a.schreiben(9);
        }
}
```

Ausgabe auf der Konsole:

long

Existiert nur eine Methode schreiben() mit einem Integer-Objekt, wird diese Methode aufgerufen, jetzt findet ein Autoboxing statt:

```
public class AutoboxingMethoden {
        public void schreiben(Integer i) {
                System.out.println("Integer");
        }
```

```
        public static void main(String[ ] args) {
                AutoboxingMethoden a = new AutoboxingMethoden();
                // Der Methode wird ein int übergeben
                a.schreiben(9);
        }
}
```

Ausgabe auf der Konsole:

Integer

Was passiert, wenn es nur eine Methode schreiben() mit einem int-Wert als Parameter gibt, und Sie der Methode ein Integer-Objekt übergeben? Jetzt findet ein Unboxing statt: Das Integer-Objekt wird automatisch in einen primitiven Datentyp int umgewandelt.

```
public class AutoboxingMethoden {
        public void schreiben(int i) {
                System.out.println("int");
        }

        public static void main(String[ ] args) {
                AutoboxingMethoden a = new AutoboxingMethoden();
                a.schreiben(new Integer(9));
        }
}
```

Ausgabe auf der Konsole:

int

Wollen Sie allerdings einer Methode schreiben() mit dem Parameter Float, einen Parameter int übergeben, kommt es zu einem Kompilierfehler. Es wird nicht zuerst, wie man vielleicht erwarten könnte, ein int in ein float, und dann anschließend dieser float-Wert automatisch in ein Float-Objekt umgewandelt, sondern es wird ein int-Wert automatisch in ein Integer-Objekt überführt. Ein Integer-Objekt kann man auch nicht einer Methode übergeben, die einen Parameter vom Typ Float vorgibt.

```
3  public class AutoboxingMethoden {
4      public void schreiben(Float f) {
5          System.out.println("Float");
6      }
7
8      public static void main(String[] args) {
9          AutoboxingMethoden a = new AutoboxingMethoden();
10         // Der Methode wird ein int übergeben
11         a.schreiben(9);
12     }
```

Im folgenden Fall kommt es weder zu einer automatischen Typanpassung noch zu einem Autoboxing: Die Umwandlung von einem int-Wert in einen byte-Wert würde einen expliziten Cast erfordern (siehe Kapitel Zuweisungen und Typumwandlung).

```
4  public class AutoboxingMethoden {
5      public void schreiben(byte b) {
6          System.out.println("byte");
7      }
8
9      public static void main(String[] args) {
10         AutoboxingMethoden a = new AutoboxingMethoden();
11         // Der Methode wird ein int übergeben
12         a.schreiben(9);
13     }
14 }
```

d) Automatische Typanpassung und Autoboxing

Weisen Sie den Wrappern Long, Double und Float ihren jeweiligen primitiven Datentyp zu, muß dieser als solcher gekennzeichnet sein (vgl. Kapitel Literale), da es anderenfalls zu einem Kompilierfehler kommt:

```
public class AutoboxingTypanpassung {
        public static void main(String[ ] args) {
                //ein long-Wert erfordert am Ende ein l
                Long l = 100l;
                Short s = 1000;
                Byte b = 100;
                //ein double-Wert erfodert ein „ .0” am Ende oder ein d
                Double d = 100.0;
                Double d1 = 100d;
                //ein float-Wert erfodert ein f am Ende oder ein „ .0f“
                Float f = 100.0f;
                Float f1 = 100f;
        }
}
```

e) Autoboxing und Arrays

Für Array-Typen sind dem Autoboxing Grenzen gesetzt: So kommt es zu einem Kompilierfehler, wenn Sie versuchen einem Integer-Array ein int-Array zuzuweisen. Ein int-Array wird nicht automatisch in ein Integer-Array umgewandelt.

```
3  public class AutoboxingArrays {
4
5
6      public static void main(String[] args) {
7          int[] i = {1, 2, 3, 4};
8          Integer[] in = i;
9      }
10
11 }
12
```

Multiple markers at this line
- Type mismatch: cannot convert from int[] to Integer[]
- in

Eine mögliche Lösung für dieses Problem ist der folgende Ansatz:

```
public class AutoboxingArrays {
	public static void main(String[ ] args) {
		int[ ] i = {1, 2, 3, 4};
		Integer[ ] in = new Integer[i.length];

		for(Integer aus:i){
			System.out.println(aus);
		}
	}
}
```

Ausgabe auf der Konsole:

```
1
2
3
4
```

4. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingBool {
	public static void main(String[ ] args) {
		Boolean b = true;
		boolean b2 = false;
		Boolean b3 = b || b2;
		boolean b4 = b && b2;
		System.out.println(b3);
		System.out.println(b4);
	}
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingB {
	public static void main(String[ ] args) {
		Boolean b = false;
		Boolean b1 = false;
		System.out.println(b == b1);
		System.out.println(b.equals(b1));
	}
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingB {
	public static void main(String[ ] args) {
```

```
                Boolean b = false;
                Boolean b1 = “false“;
                System.out.println(b == b1);
                System.out.println(b.equals(b1));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingB {
        public static void main(String[ ] args) {
                Boolean b = false;
                Boolean b1 = new Boolean(false);
                System.out.println(b == b1);
                System.out.println(b1.equals(b));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingB {
        public static void main(String[ ] args) {
                boolean b = false;
                Boolean b1 = new Boolean(false);
                System.out.println(b == b1);
                System.out.println(b1.equals(b));
        }
}
```

a) true false
b) false true
c) true true

d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingB {
        public static void main(String[ ] args) {
                boolean b = false;
                Boolean b1 = new Boolean(false);
                System.out.println(b == b1);
                System.out.println(b.equals(b1));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingInt {
        public static void main(String[ ] args) {
                Integer in = 128;
                Integer i1 = new Integer(”128“);
                System.out.println(in == i1);
                System.out.println(i1.equals(in));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingInt {
```

```
        public static void main(String[ ] args) {
                Integer in = 128;
                Integer i1 = 128;
                System.out.println(in == i1);
                System.out.println(i1.equals(in));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingInt {
        public static void main(String[ ] args) {
                Integer in = 127;
                Integer i1 = 127;
                System.out.println(in == i1);
                System.out.println(i1.equals(in));
        }
}
```

a) true false
b) false true
c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingInt {
        public static void main(String[ ] args) {
                Integer in = new Integer(127);
                Integer i1 = new Integer(127);
                System.out.println(in == i1);
                System.out.println(i1.equals(in));
        }
}
```

a) true false
b) false true

c) true true
d) false false
e) Kompilierfehler
f) Es wird Exception geworfen.

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingM {
        public void schreiben(int i) {
                System.out.println(i);
        }

        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);}}
```

a) 9
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingM {
        public void schreiben(long l) {
                System.out.println(l);
        }

        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);
        }
}
```

a) 9
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingM {
        public void schreiben(byte b) {
                System.out.println(b);
        }
```

```
        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);
        }
}
```

a) 9
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingM {
        public void schreiben(Float f) {
                System.out.println(f);
        }

        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);
        }}
```

a) 9
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(15) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class AutoboxingM {
        public void schreiben(long l) {
                System.out.println("1");
        }
                public void schreiben(Integer i) {
                System.out.println("2");
        }
        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);}}
```

a) 1
b) 2
c) Kompilierfehler
d) Es wird eine Exception geworfen.

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und

startet?

```
public class AutoboxingM {
        public void schreiben(long l) {
                System.out.println("1");
        }
                public void schreiben(int i) {
                System.out.println("2");
        }
        public static void main(String[ ] args) {
                AutoboxingM a = new AutoboxingM();
                a.schreiben(9);
        }
}
```

a) 1
b) 2
c) Kompilierfehler
d) Es wird eine Exception geworfen.

(17) Frage

Was darf an der Stelle, an der hier steht, hinzugefügt werden?

```
public class AutoboxingT {
        public static void main(String[ ] args) {
                hier
        }
}
```

a) Double d = 100d;
b) Double d = 100;
c) Double d = 100,0;
d) Double d = 100.0;

5. Lösungen

(1) Frage

a

(2) Frage

c

(3) Frage

e

(4) Frage
b

(5) Frage
c

(6) Frage
e

(7) Frage
b

(8) Frage
b

(9) Frage
c

(10) Frage
b

(11) Frage
a

(12) Frage
a

(13) Frage
b

(14) Frage
b

(15) Frage
a

(16) Frage
b

(17) Frage
a, d

C. Methoden und Konstruktoren mit variabler Argumentanzahl (Varargs)

1. Begriffsbestimmung

Wie erreicht man es, einer Methode oder einem Konstruktor null oder beliebig viele Parameter hinzufügen zu können? Mit Methoden oder Konstruktoren, die eine beliebige Anzahl an Argumenten zulassen, auch Varargs genannt: Einer Methode wird ein Paramtertyp, gefolgt von drei Auslassungspunkte und einem Parameternamen, hinzufügt: String... viele. Der Kompiler interpretiert diese Schreibweise als String-Array: String[ ] viele. Dieser Methode können null, beliebig viele Argumente oder ein String-Array übergeben werden.

```
public class VariableArguments {

        void aufrufen(String... viele){

        }

        public static void main(String[ ] args) {
                VariableArguments va = new VariableArguments( );
                va.aufrufen();
                va.aufrufen("eins", "zwei", "drei");
                va.aufrufen(new String[ ]{"hallo", "Tschüss"});
        }
}
```

Es ist auch weitherhin möglich, Parameter unterschiedlichen Typs der Parameterliste hinzuzufügen, wobei darauf geachtet werden muß, daß die variable Argumentanzahl am Schluß steht. Ist dies nicht der Fall kommt es zu einem Kompilierfehler.

```
public class VariableArguments {

        void aufrufen(int i, String... viele){

        }

        public static void main(String[ ] args) {
                VariableArguments va = new VariableArguments();
                va.aufrufen(1, "eins", "zwei", "drei");
                va.aufrufen(2, new String[ ]{"hallo", "Tschüss"});
        }
}
```

Auch zu einem Kompilierfehler kommt es, wenn Sie versuchen wie untenstehend, die Methode aufrufen() zu überladen: Der Ausdruck „String... viele“ wird intern wie „String[ ] viele“ vom Kompiler verarbeitet, somit sind für den Kompiler untenstehende

Methoden identisch:

```
2  public class VariableArguments {
3
4      void aufrufen(String... viele){
5          for(String s: viele){
6              System.out.println(s);
7          }
8      }
9
10     void aufrufen(String[] viele)   Duplicate method aufrufen(String[]) in type VariableArguments
11         for(String s: viele){
12             System.out.println(s);
13         }
14     }
15
16     public static void main(String[] args) {
17         VariableArguments va = new VariableArguments();
18         va.aufrufen("eins", "zwei", "drei");
19         va.aufrufen(new String[]{"hallo", "Tschüss"});
20     }
```

2. Auslesen einer variablen Argumentliste

Wie können Sie die variable Argumentliste auslesen? Mit der erweiterten for-Schleife, und auf die gleiche Art und Weise wie auch Arrays ausgelesen werden können (siehe Kapitel Erweiterte for-Schleife).

```
public class VariableArguments {

        void aufrufen(String... viele){
                for(String s: viele){
                        System.out.println(s);
                }
        }

        public static void main(String[ ] args) {
                VariableArguments va = new VariableArguments();
                va.aufrufen("eins", "zwei", "drei");
                va.aufrufen(new String[ ]{"hallo", "Tschüss"});
        }
}
```

Ausgabe auf der Konsole:

```
eins
zwei
drei
hallo
Tschüss
```

3. Autoboxing und Varargs

Wie Sie bereits im Kapitel Autoboxing gesehen haben, wird ein Integer-Array nicht automtisch in ein int-Array umgewandelt, deswegen kommt es in untenstehendem Fall zu einem Kompilierfehler. Sie können einer Methode void aufrufen (int... i) kein Integer-Array als Argument übergeben.

```
3
4  public class VariableArgumentsInteger {
5
6      void aufrufen(int... i){
7
8      }
9
10     public static void main(String[] args) {
11         VariableArgumentsInteger vai = new VariableArgumentsInteger();
12         vai.aufrufen(new Integer [] {1, 2, 3});
13     }
14 }
```

4. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class VariableArgs {
	void aufrufen(Integer... vieleZahlen){
		for(Integer i: vieleZahlen){
			System.out.println(i);
		}
	}

	public static void main(String[ ] args) {
		VariableArgs va = new VariableArgs();
		va.aufrufen(1, 2, 3);
	}
}
```

a) 1 2 3
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class VariableArgs {
	void aufrufen(Integer... vieleZahlen, int i){
		for(Integer i: vieleZahlen){
			System.out.println(i);
		}
	}

	public static void main(String[ ] args) {
		VariableArgs va = new VariableArgs();
		va.aufrufen(1, 2, 3);
	}
}
```

a) 1 2 3
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class VariableArgs {
	void aufrufen(Integer... vieleZahlen){
		for(Integer i: vieleZahlen){
```

```
                        System.out.println(i);
                }
        }

        public static void main(String[ ] args) {
                VariableArgs va = new VariableArgs();
                va.aufrufen(new Integer[ ] {1, 2, 3});
        }
}
```

a) 1 2 3
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class VariableArgs {
        void aufrufen(Integer... vieleZahlen){
                for(Integer i: vieleZahlen){
                        System.out.println(i);
                }
        }

        public static void main(String[ ] args) {
                VariableArgs va = new VariableArgs();
                va.aufrufen(int {1, 2, 3});
        }
}
```

a) 1 2 3
b) Kompilierfehler
c) Es wird eine Exception geworfen.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class VariableArgs {
        void aufrufen(Integer... vieleZahlen){
                for(Integer i: vieleZahlen){
                        System.out.println(i);
                }
        }
        void aufrufen(Integer[ ] vieleZahlen){
                for(Integer i: vieleZahlen){
                        System.out.println(i);
                }
        }
        public static void main(String[ ] args) {
                VariableArgs va = new VariableArgs();
```

```
                va.aufrufen(new Integer[ ] {1, 2, 3});
        }
}
```

a) 1 2 3
b) Kompilierfehler
c) Es wird eine Exception geworfen.

5. Lösungen

(1) Frage

a

(2) Frage

b

(3) Frage

a

(4) Frage

b

(5) Frage

b

D. Generics im Collections-Framework

1. Generics bei Listen

Welche Verbesserungen bringen uns Generics gegenüber der bisherigen Verwendung von Klassen des Collections Framework? Sehen wir uns erst einmal an, wie seither z. B. eine LinkedList funktioniert hat (vgl. Kapitel Das Collections-Framework): Die LinkedList ist ein Weg mehrere Objekte in einer Liste zu speichern, der allerdings einen großen Nachteil mit sich bringt: In die LinkedListed werden Stringobjekte ein- und Objekte der Klasse java.lang.Object ausgegeben: Object o = i.next();. Wollte man anstelle der Objekte der Klasse java.lang.Object ein Stringobjekt als Ergebnis haben, mußte man einen Cast hinzufügen: String o = (String)i.next();.

```
import java.util.LinkedList;

public class CollectionList {
        public static void main(String[ ] args) {
                LinkedList s = new LinkedList();
                String st = new String(”delta“);
                String st1 = new String(”aber“);
                String st2 = new String(”beta“);
                s.add(st);
                s.add(st1);
                s.add(st2);

                for(Iterator i = s.iterator(); i.hasNext();){
                        Object o = i.next();
                        System.out.println( o );
                }
        }
}
```

Ausgabe auf der Konsole:

```
delta
aber
beta
```

Mit der Einführung der Generics wird dieser Cast überflüssig: Es werden String-Objekte in die LinkedList ein- und wieder ausgegeben. Wie geschieht dies? Folgendermaßen: Das Object der Klasse LinkedList wird um den entsprechenden Typ <String> ergänzt. So wird sichergestellt, daß nur noch String-Objekte der LinkedList hinzugefügt werden dürfen. Hiermit wird eine sogenannte Typsicherheit gewährleistet, die nur für Objekte und nicht für primitive Datentypen gilt. Außerdem wird es durch die erweiterte for-Schleife einfacher, die Strings aus der LinkedList auszulesen, da der Iterator überflüssig wird. Verwenden Sie weiterhin den Iterator, werden auch in der neuen Java-Version, nur Objekte der Klasse java.lang.Object zurückgegeben.

```
import java.util.LinkedList;

public class GenericsCollectionList {
	public static void main(String[ ] args) {
		LinkedList<String> s = new LinkedList<String>();
		//oder:
		//LinkedList<String> s = new LinkedList();
		String st = new String("delta");
		String st1 = new String("aber");
		String st2 = new String("beta");
		s.add(st);
		s.add(st1);
		s.add(st2);
		for (String r:s){
			System.out.println(r);
		}
	}
}
```

Ausgabe auf der Konsole:

```
delta
aber
beta
```

Würden Sie die LinkedList wie folgt LinkedList<Object> s = new LinkedList<String>(); instanziieren, würde dies zu einem Kompilierfehler führen, da man links nur Objekte der Klasse java.lang.Object und rechts nur Objekte der Klasse java.lang.String einfügen darf. Es kommt zu Problemen, da ein String zwar ein Subtyp von Object ist, aber LinkedList<String> kein Subtyp von LinkedList<Object> ist. Die Klasse String ist eine Kindklasse von Klasse Object, wohingegen LinkedList<String> keine Kindklasse von LinkedList<Object> ist (vgl. Kapitel Subtyping und Casting von Objekten).

```
import java.util.LinkedList;

public class GenericsCollectionList {
	public static void main(String[ ] args) {
		//folgende Zeile würde zu einem Kompilierfehler führen:
		LinkedList<Object> s = new LinkedList<String>();
		String st = new String("delta");
		String st1 = new String("aber");
		String st2 = new String("beta");
		s.add(st);
		s.add(st1);
		s.add(st2);
		for (String r:s){
			System.out.println(r);
		}
	}
}
```

Sie können für die Variable r auch weiterhin den Typ Object definieren, dies funktioniert ohne weiteres, da Object eine Superklasse der Stringklasse ist (vgl. Kapitel Die erweiterte for-Schleife). Bitte beachten Sie, daß wenn Sie weiterhin den Iterator für Generics benutzen wollen, nach wie vor Objekte der Klasse java.lang. Object zurückgegeben werden.

In untenstehendem Beispiel kann man, obwohl es auf den ersten Blick dem bisher Gesagten widerspricht, einer LinkedList<Object> Stringobjekte hinzufügen. Dies ist möglich, da ein Objekt der Klasse String ein Subtyp der Klasse Object ist, oder will man es vereinfacht ausdrücken, ein String ist aufgrund der Verwandtschaftsverhältnisse ein Object, da die Klasse Object eine Superklasse der Klasse String ist. Sie erhalten jedoch Objekte der Klasse Object zurück, obwohl Sie Stringobjekte hinzufügen. Deshalb würde es zu einem Kompilierfehler führen, wenn Sie für r den Rückgabetyp String festlegen würden (vgl. Kapitel Die erweiterte for-Schleife).

```
import java.util.LinkedList;

public class GenericsCollectionList {
	public static void main(String[ ] args) {
		LinkedList<Object> s = new LinkedList<Object>();
		String st = new String(”delta“);
		String st1 = new String(”aber“);
		String st2 = new String(”beta“);
		s.add(st);
		s.add(st1);
		s.add(st2);

		for(Object r:s){
			System.out.println(r);
		}
	}
}
```

Ausgabe auf der Konsole:

```
delta
aber
beta
```

2. Generics bei Sets und Maps

Bei allen anderen Klassen des Collections Framework wird analog vorgegangen. Alle Maps haben allerdings zusätzlich zu Ihren Elementen noch einen Schlüssel, von dem Sie auch den Typ festlegen können. Nun möchte ich Ihnen anhand der Hashtable zeigen, wie eine Map instanziiert wird, und Elemente ein- und wieder ausgelesen werden.

```
import java.util.Hashtable;
import java.util.Map;

public class GenericsHashtable {
        public static void main(String[ ] args) {
                String s1 = ”1”;
                String s2 = “2”;
                String w1 = ”hallo”;
                String w2 = ”bye”;

                Hashtable<String, String> h = new Hashtable<String, String>();
                h.put(s1, w1);
                h.put(s2, w2);

                //Auslesen mit Hilfe der keySet()-Methode der Klasse Hashtable
                for(String key : h.keySet()){
                        System.out.println(“Schlüssel: “ + key );
                        System.out.println(“Wert: “ + h.get(key) );
                }

                //Auslesen mit Hilfe der entrySet()-Methode aus dem Interface
                //Map.Entry
                for(Map.Entry<String,String> entry : h.entrySet()){
                        System.out.println(“Schlüssel: “ + entry.getKey() );
                        System.out.println(“Wert: “ + entry.getValue() );
                }
        }
}
```

3. Der Einsatz von Wildcards

Wir haben also gesehen: Eine LinkedList<String> ist kein Subtyp von LinkedList<Object>. Jetzt kommt die nächste Frage: Ist LinkedList<String> ein Subtyp von List<Object>? Auch nicht! Und dies obwohl LinkedList ein Subtyp von List ist. Also: LinkedList<String> ist kein Subtyp von List<Object>! Diese Tatsache hat zur Folge, daß, wie in untenstehendem Fall, es zu einem Kompilierfehler kommt, wenn Sie eine LinkedList<String> an eine Methode übergeben wollen, die als Parameter eine List<Object> vorgibt (vgl. Kapitel Subtyping und Casting von Objekten).

```
3  import java.util.List;
4  import java.util.LinkedList;
5
6  public class GenericsCollectionList {
7      public static void main(String[] args) {
8          LinkedList<String> s = new LinkedList<String>();
9          String st = new String("delta");
10         String st1 = new String("aber");
11         String st2 = new String("beta");
12         s.add(st);
13         s.add(st1);
14         s.add(st2);
15         GenericsCollectionList g = new GenericsCollectionList();
16         g.auslesen(s);
17         }
18         public void auslesen(List<Object> l){
19             for (Object r:l){
20                 System.out.println(r);
21             }
22     }
23 }
```

Es liegt nahe zu fragen: Wie löse ich dieses Problem? Mit Wildcards. Ersetzen Sie nun List<Object> durch List<? extends Object> können Sie der Methode public void auslesen(List<? extends Object> l) alle Listen, die Objekte der Klasse Object und alle dazugehörigen Subklassen enthalten, als Parameter übergeben. Diese Beziehung heißt für Objekte und Interfaces extends: <? extends MeinObjekt> und <? extends MeinInterface>. Es besteht die Möglichkeit auch nur <?> zu schreiben, dann sind mit dem Fragezeichen alle Objekte aller Typen gemeint.

```
import java.util.List;
import java.util.LinkedList;

public class GenericsCollectionList {
        public static void main(String[ ] args) {
                LinkedList<String> s = new LinkedList<String>();
                String st = new String(”delta”);
                String st1 = new String(”aber”);
                String st2 = new String(”beta”);
                s.add(st);
                s.add(st1);
                s.add(st2);
                GenericsCollectionList g = new GenericsCollectionList();
                g.auslesen(s);

                }
                public void auslesen(List<? extends Object> l){
                        for (Object r:l){
                                System.out.println(r);
                        }
        }
}
```

Dies hat folgenden Nachteil: Es werden wieder Objekte der Klasse java.lang. Object ausgelesen. So muß die for-Schleife wieder angepaßt werden: Es muß dort Object r:l stehen. Hat r allerdings String als Rückgabetyp, kommt es zu einem Kompilierfehler.

```
3  import java.util.List;
4  import java.util.LinkedList;
5
6  public class GenericsCollectionList {
7      public static void main(String[] args) {
8          LinkedList<String> s = new LinkedList<String>();
9          String st = new String("delta");
10         String st1 = new String("aber");
11         String st2 = new String("beta");
12         s.add(st);
13         s.add(st1);
14         s.add(st2);
15         GenericsCollectionList g = new GenericsCollectionList();
16         g.auslesen(s);
17
18         }
19         public void auslesen(List<? extends Object> l){
20             for (String r:l){
21                 System.out.println(r);
22             }
23     }
24 }
```

Unter welchen Umständen kann man (List<? extends Object> l) durch (List<Object> l) ersetzen? Wenn Sie eine LinkedList<Object> übergeben wollen. Ein LinkedList<Object> ist sehr wohl ein Subtyp von List<Object>. Wohingegen eine LinkedList<String> kein Subtyp von einer List<Object> ist (siehe oben).

```
import java.util.List;
import java.util.LinkedList;

public class GenericsCollectionList {
        public static void main(String[ ] args) {
                LinkedList<Object> s = new LinkedList<Object>();
                String st = new String("delta");
                String st1 = new String("aber");
                String st2 = new String("beta");
                s.add(st);
                s.add(st1);
                s.add(st2);
                GenericsCollectionList g = new GenericsCollectionList();
                g.auslesen(s);
                }
                public void auslesen(List<Object> l){
                        for (Object r:l){
                                System.out.println(r);
                        }
        }
}
```

Was passiert, wenn Sie bereits bei der Instanziierung, wie unten, für den Typ auf der linken Seite LinkedList<? extends Object> s definieren? Zuerst geschieht nichts. Sie erhalten erst einen Kompilierfehler, wenn Sie mit der Methode add() ein Element hinzufügen wollen. Der Grund hierfür ist: Der Typ von LinkedList<? extends Object> s wurde nicht genau bestimmt, die Methode add() braucht aber einen genau festgelegten Typ des Parameters der LinkedList, um den Forderungen der Typsicherheit entsprechen zu können. Typsicherheit soll gewährleisten, daß nur bestimmte Objekte eingelesen werden dürfen.

```
3  import java.util.List;
4  import java.util.LinkedList;
5
6  public class GenericsCollectionList {
7      public static void main(String[] args) {
8          LinkedList<? extends Object> s = new LinkedList<String>();
9          String st = new String("delta");
10         String st1 = new String("aber");
11         String st2 = new String("beta");
12         s.add(st);
13         s.add(st1);
14         s.add(st2);
15         GenericsCollectionList g = new GenericsCollectionList();
16         g.auslesen(s);
17         }
18
19         public void auslesen(List<? extends Object> l){
20             for (Object r:l){
21                 System.out.println(r);
22             }
23     }
24 }
```

Was passiert, wenn Sie nachträglich Ihre LinkedList<String> einer Liste List<? extends Object> li = s; zuweisen? Dies ist ohne Probleme möglich, die Liste gibt nun allerdings keine String-Objekte mehr zurück.

```
import java.util.List;
import java.util.LinkedList;

public class GenericsCollectionList {
        public static void main(String[ ] args) {
                LinkedList<String> s = new LinkedList<String>();
                String st = new String(”delta”);
                String st1 = new String(”aber”);
                String st2 = new String(“beta”);
                s.add(st);
                s.add(st1);
                s.add(st2);
                List<? extends Object> li = s;
                        for (Object r:li){
                                System.out.println(r);
                        }
        }
}
```

Wenn die Elemente der List li ausgelesen werden, erhalten Sie Objekte der Klasse java.lang.Object zurück. Dies müssen Sie auch in der for-Schleife bedenken: Ändern Sie den Typ der Variable r vom Typ Object in den Typ String, kommt es zu einem Kompilierfehler.

```
3  import java.util.List;
4  import java.util.LinkedList;
5
6  public class GenericsCollectionList {
7      public static void main(String[] args) {
8          LinkedList<String> s = new LinkedList<String>();
9          String st = new String("delta");
10         String st1 = new String("aber");
11         String st2 = new String("beta");
12         s.add(st);
13         s.add(st1);
14         s.add(st2);
15         List<? extends Object> li = s;
16
17             for (String r:li){
18                 System.out.println(r);
19             }
20     }
21 }
```

Untenstehender Versuch der Instanziierung eines Generic-Objektes führt sofort zu einem Kompilierfehler, da eine so vage Festlegung des Typs, der zugrunde liegenden Idee der Generics widerspricht.

LinkedList<? extends Object> l = new LinkedList<? extends Object>(); **//Kompilierfehler**

4. Generische Interfaces, Klassen und Methoden

Es bleibt eine Frage zu klären: Kann man auch für Interfaces, Klassen und Methoden den Typ des Parameters sicherstellen? Ja! Beginnen wir mit einem Rückblick: Wenn Sie einige Seiten zum Thema Interface Comparator im Kapitel Collections und Maps zurückblättern, werden Sie feststellen, daß bei der Implementierung des Comparator-Interfaces einige Probleme auftauchen. Bisher war es nur möglich, der compare()-Methode Objekte als Parameter zu übergeben. Dies hatte folgenden Nachteil: Objekte, wie z. B. ein Buch, wurden automatisch in ein Objekt der Klasse java.lang.Object überführt, und mußten somit wieder in ein Objekt der Klasse Buch gecastet werden.

```
import java.util.Comparator;

public class BuchVergleich implements Comparator{
        public int compare(Object o1, Object o2) {

                /*Das übergebene Objekt wird in ein Objekt der Klasse
                java.lang.Object umgewandelt und muß somit wieder in ein Buch
                gecastet werden */
                return (((Buch)o1).getAutor().compareTo(((Buch)o2).getAutor()));
        }
}
```

Diese umständliche Vorgehensweise gehört nun der Vergangenheit an: Mit der neuen Java-Version wurden generische Interfaces eingeführt. Sie können jetzt bestimmte Typen <T> für Parameter, die Sie dem Interface übergeben, festlegen. Das Interface mit seiner compare()-Methode sieht nun wie folgt aus:

```
public interface Comparator<T>{
        public int compare(T o1, T o2);
}
```

Ändern wir unsere ursprüngliche Klasse Buchvergleich, die das Comparator-Interface implementiert hatte, entsprechend ab, sieht sie nun wie untenstehend aus. Wie Sie sehen können, erspart die so erlangte Typsicherheit jegliches Casten.

```
import java.util.Comparator;

public class BuchVergleich implements Comparator<Buch>{
        public int compare(Buch b1, Buch b2) {
                return (b1.getAutor().compareTo(b2.getAutor()));
        }
}
```

Einige wichtige Bemerkungen zum Schluß: Das <T>, mit dem die Typsicherheit bei Objekten garantiert wird, wird bei Klassen des Collections Frameworks durch ein <E> ersetzt. Außerdem können statische Bestandteile einer Klasse den Typ der Klasse nicht übernehmen, da es statische Bestandteile pro Klasse nur einmal gibt. So würden sich zwei Objekte mit unterschiedlichen Typen (Zahl<Integer> und Zahl<Float>) den gleichen statischen Bestandteil teilen, dies widerspräche der Typsicherheit. Eine statische Methode kann allerdings einen eigenen Typ haben.

5. Das Collections-Framework und Arrays

Die Klassen Collections und Arrays gab es bereits unter Java 1.4. Sie sind allerdings erst seit der Zertifizierung 1.5 prüfungsrelevant. Die sort()-Methode der Klasse

Collections war allerdings bereits für die Prüfung 1.4 notwendig (vgl. im Kapitel Das Collections-Framework das Unterkapitel Das Comparator- und Comparable-Interface). Diesen beiden Klassen wurden aber im Tiger-Release (Java 1.5) mehrere Methoden hinzugefügt. In meinen folgenden Ausführungen möchte ich mich auf die wichtigen Methoden für die Zertifzierung beschränken.

a) Umwandeln von Listen und Sets in Arrays

Wie wird eine Liste oder ein Set in ein Array umgewandelt? Mit Hilfe der toArray()-Methode. Da die toArray()-Methode ohne Parameter ein Object-Array zurückgibt, wir aber in unserem Beispiel ein String-Array brauchen, nehmen wir die überladene Methode toArray, die es Ihnen erlaubt ein String-Array mit der Länge des TreeSets (s.size()) zu erstellen.

```
import java.util.TreeSet;

public class CollectionSetArray {
        public static void main(String[ ] args) {
                TreeSet<String> s = new TreeSet<String>();
                String st = new String(”Hallo“);
                String st1 = new String(”Bye“);
                String st2 = new String(”Auf Wiedersehen“);
                String st3 = new String(”Goodbye”);
                s.add(st);
                s.add(st1);
                s.add(st2);
                s.add(st3);
                String[ ] ss = s.toArray( new String[ s.size() ] );

                for (String str: ss){
                        System.out.println(str);
                }
        }
}
```

Ausgabe auf der Konsole:

```
Auf Wiedersehen
Bye
Goodbye
Hallo
```

b) Umwandeln von Arrays in Listen

Sie wollen ein Array in eine Liste umwandeln? Welche Klasse und welche Methode benötigen Sie? Die asList()-Methode der Klasse Arrays, der Sie das Array als Parameter übergeben können.

```
import java.util.LinkedList;
import java.util.Arrays;
```

```
public class CollectionListArrays {
	public static void main(String[ ] args) {
		String[ ] s = {“eins”, “zwei”};
		LinkedList<String> ll = new LinkedList<String>(Arrays.asList(s));

		for(String st: ll){
			System.out.println( st );
		}
	}
}
```

Ausgabe auf der Konsole:

```
eins
zwei
```

c) Die Methode binarySearch() der Klasse Collections und Arrays

Wollen Sie den Inhalt eines Array sortieren und dann anschließend feststellen, an welcher Indexposition sich ein bestimmtes Element befindet? Dann brauchen Sie die Methoden sort() und binarySearch() der Klasse Arrays. Bitte beachten Sie, daß die Methode binarySearch() nur bei einem sortierten Array korrekt funktioniert. Wird das Array nicht sortiert, kann das genaue Ergebnis nicht exakt vorausgesagt werden.

```
import java.util.Arrays;

public class ArraysBinarySearch {
	public static void main(String[ ] args) {
		String[ ] s = {“eins”, “zwei”, “drei”};
		//die Elemente des Arrays werden alphabetisch sortiert
		Arrays.sort(s);
		//nun lautet die Reihenfolge des Arrays: drei, eins, zwei
		int i = Arrays.binarySearch(s, ”drei“);
		//das Element drei befindet sich nun an der Stelle 0
		System.out.println(i);
	}
}
```

Ausgabe auf der Konsole:

```
0
```

In der Klasse Collections gibt es auch eine Methode binarySearch(), die es Ihnen bei Listen ermöglicht, die Indexposition eines Elementes festzustellen. Auch hier muß zuerst die Liste sortiert werden. Vorsicht: Listen können Sie nur sortieren, wenn der Liste Elemente hinzugefügt worden sind, die das Comparable- oder Comparator-Interface korrekt implementieren (siehe Kapitel Comparator- und Comparable-Interface).

```
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListBinarySearch {
	public static void main(String[ ] args) {
		ArrayList<String> s = new ArrayList<String>();
		String st = new String(”delta”);
		String st1 = new String(”aber”);
		String st2 = new String(”beta”);
		s.add(st);
		s.add(st1);
		s.add(st2);
		Collections.sort(s);
		int i = Collections.binarySearch(s, ”delta”);
		System.out.println(i);
	}
}
```

Ausgabe auf der Konsole:

2

6. Das Interface Queue

a) Allgemeines

Seit Java 1.5 gibt es ein neues Interface im java.util-Package: Das Interface Queue, das in der Regel das Prinzip FIFO (first in, first out) umsetzt. Zwei der wichtigsten Klassen, die dieses Interface implementieren, sind die Klassen PriorityQueue und LinkedList. Das Interface Queue stellt zwei neue Methoden zur Verfügung, die die bisherige Verwendung der Methoden add() und remove() des Collection-Interface ergänzen sollen: die Methoden offer() und poll(). Die Methode offer() fügt wie die Methode add() der Queue Elemente hinzu, sie wirft aber keine Exception, wenn die Queue voll ist, sondern gibt false zurück. Ähnlich verhält es sich mit der Methode poll(), die im Gegensatz zur Methode remove() keine Exception wirft, wenn die Queue leer ist, sondern null als Ergebnis hat. Auch null gibt die Methode peek() zurück, wenn Sie an der Spitze der Queue kein Element findet, wohingegen die Methode element() beim gleichen Ergebnis eine Exception wirft.

b) Priority Queue

Die Klasse Priority Queue implementiert das Interface Queue, die allerdings wie die Klassen TreeSet und TreeMap ihre Elemente alphabetisch und nach Zahlen aufsteigend sortiert. Und die Elemente müssen ebenfalls entweder das Comparator- oder das Comparable korrekt umsetzen (vgl. Kapitel Comparator- und Comparable Interface).

7. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList<String>();          Zeile 1
                String s = new String("eins");
                ll.add(s);                                                 Zeile 2
                Generics g = new Generics();
                g.auslesen(ll);                                            Zeile 3
                }
                public void auslesen(List<? extends Object> l){            Zeile 4
                        for (Object r:l){                                  Zeile 5
                                System.out.println(r);
                        }
        }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
        public static void main(String[ ] args) {
                LinkedList<Object> ll = new LinkedList<String>();          Zeile 1
                String s = new String("eins");
                ll.add(s);                                                 Zeile 2
                Generics g = new Generics();
                g.auslesen(ll);                                            Zeile 3
                }
                public void auslesen(List<? extends Object> l){            Zeile 4
                        for (Object r:l){                                  Zeile 5
                                System.out.println(r);
                        }
        }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList<String>();        Zeile 1
                String s = new String("eins");
                ll.add(s);                                               Zeile 2
                Generics g = new Generics();
                g.auslesen(ll);                                          Zeile 3
                }
                public void auslesen(List<? > l){                        Zeile 4
                        for (Object r:l){                                Zeile 5
                                System.out.println(r);
                        }
        }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList<String>();        Zeile 1
                String s = new String("eins");
                ll.add(s);                                               Zeile 2
                Generics g = new Generics();
                g.auslesen(ll);                                          Zeile 3
```

```
                    }
                    public void auslesen(List<? extends Object> l){          Zeile 4
                        for (String r:l){                                    Zeile 5
                            System.out.println(r);
                        }
        }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
    public static void main(String[ ] args) {
        LinkedList<? extends Object> ll = new LinkedList<String>();      Zeile 1
        String s = new String("eins");
        ll.add(s);                                                       Zeile 2
        Generics g = new Generics();
        g.auslesen(ll);                                                  Zeile 3
        }
        public void auslesen(List<? extends Object> l){                  Zeile 4
            for (Object r:l){                                            Zeile 5
                    System.out.println(r);
        }
    }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;
```

```
public class Generics {
    public static void main(String[ ] args) {
        LinkedList<Object> ll = new LinkedList<Object>();          Zeile 1
        String s = new String("eins");
        ll.add(s);                                                 Zeile 2
        Generics g = new Generics();
        g.auslesen(ll);                                            Zeile 3
        }
        public void auslesen(List<? extends Object> l){            Zeile 4
            for (Object r:l){                                      Zeile 5
                    System.out.println(r);
        }
    }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
    public static void main(String[ ] args) {
        LinkedList<Object> ll = new LinkedList<Object>();          Zeile 1
        String s = new String("eins");
        ll.add(s);                                                 Zeile 2
        Generics g = new Generics();
        g.auslesen(ll);                                            Zeile 3
        }
        public void auslesen(List<Object> l){                      Zeile 4
            for (Object r:l){                                      Zeile 5
                    System.out.println(r);
        }
    }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics {
    public static void main(String[ ] args) {
        LinkedList<Object> ll = new LinkedList<Object>();          Zeile 1
        String s = new String("eins");
        ll.add(s);                                                 Zeile 2
        Generics g = new Generics();
        g.auslesen(ll);                                            Zeile 3
        }
        public void auslesen(List<Object> l){                      Zeile 4
            for (String r:l){                                      Zeile 5
                    System.out.println(r);
        }
    }
}
```

a) eins
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Generics2 {
        public static void main(String[ ] args) {
                LinkedList<Object> ll = new LinkedList<Object>();          Zeile 1
                String s = new String("eins“);
                String s1 = new String("zwei“);
                ll.add(s);                                                 Zeile 2
                ll.add(s1);                                                Zeile 3
                        for (Object r:ll){                                 Zeile 4
                                System.out.println(r);
                        }
        }
}
```

a) eins zwei
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Generics2 {
        public static void main(String[ ] args) {
                LinkedList<Object> ll = new LinkedList<Object>();          Zeile 1
                String s = new String("eins");
                String s1 = new String("zwei");
                ll.add(s);                                                 Zeile 2
                ll.add(s1);                                                Zeile 3
                        for (String r:ll){                                 Zeile 4
                                System.out.println(r);
                        }
        }
}
```

a) eins zwei
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Generics2 {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList();                  Zeile 1
                String s = new String("eins");
                String s1 = new String("zwei");
                ll.add(s);                                                 Zeile 2
                ll.add(s1);                                                Zeile 3
                        for (String r:ll){                                 Zeile 4
                                System.out.println(r);
                        }
        }
}
```

a) eins zwei
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Generics2 {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList<String>();          Zeile 1
                String s = new String("eins");
                String s1 = new String("zwei");
                ll.add(s);                                                 Zeile 2
                ll.add(s1);                                                Zeile 3
                List<? extends Object> l = ll;                             Zeile 4
                        for (Object r:ll){                                 Zeile 5
                                System.out.println(r);
                        }
        }
}
```

a) eins zwei
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(13) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class Generics2 {
        public static void main(String[ ] args) {
                LinkedList<String> ll = new LinkedList<String>();          Zeile 1
                String s = new String("eins");
                String s1 = new String("zwei");
                ll.add(s);                                                 Zeile 2
                ll.add(s1);                                                Zeile 3
                List<? extends Object> l = ll;                             Zeile 4
                        for (String r:ll){                                 Zeile 5
                                System.out.println(r);
                        }
        }
}
```

a) eins zwei
b) Kompilierfehler in Zeile 1
c) Kompilierfehler in Zeile 2
d) Kompilierfehler in Zeile 3
e) Kompilierfehler in Zeile 4
f) Kompilierfehler in Zeile 5

(14) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
```

```
import java.util.LinkedList;
import java.util.Collections;

public class Generics2 {
	public static void main(String[ ] args) {
		LinkedList<String> ll = new LinkedList();
		String s = new String(”omega”);
		String s1 = new String(”alpha”);
		ll.add(s);
		ll.add(s1);
		Collections.sort(ll);
			for (String r:ll){
				System.out.println(r);
			}
	}
}
```

a) omega alpha
b) alpha omega
c) Kompilierfehler, da in der Klasse Collections keine Methode sort() existiert.
d) Kompilierfehler, da Sie die Methode sort() der Klasse LinkedList hätten nehmen müssen.

(15) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.List;
import java.util.LinkedList;

public class Generics2 {
	public static void main(String[ ] args) {
		LinkedList<String> ll = new LinkedList();
		String s = new String(”omega”);
		String s1 = new String(”alpha”);
		ll.add(s);
		ll.add(s1);
		ll.sort( );
			for (String r:ll){
				System.out.println(r);
			}
	}
}
```

a) omega alpha
b) alpha omega
c) Kompilierfehler, da in der Klasse LinkedList keine Methode sort() existiert.

(16) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Hashtable;
import java.util.Map;

public class GenericsHash {
	public static void main(String[ ] args) {
		String s1 = ”1”;
		String s2 = ”1”;
		String w1 = ”yes”;
		String w2 = ”no”;

		Hashtable<String, String> h = new Hashtable<String, String>();
		h.put(s1, w1);
		h.put(s2, w2);

		for(String key : h.keySet()){
			System.out.println(key );
			System.out.println(h.get(key) );
		}
	}
}
```

a) 1 no
b) 1 yes
c) 1 no 1 yes
d) Kompilierfehler

(17) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Hashtable;
import java.util.Map;

public class GenericsHash {
	public static void main(String[ ] args) {
		String s1 = ”1”;
		String s2 = ”2”;
		String w1 = ”yes”;
		String w2 = ”no”;

		Hashtable<String, String> h = new Hashtable<String, String>();
		h.put(s1, w1);
		h.put(s2, w2);

		for(Map.Entry<String,String> entry : h.entrySet()){
			System.out.println(entry.getKey() );
			System.out.println(entry.getValue() );
		}
	}
}
```

a) 1 yes
b) 2 no
c) Die Reihenfolge der Eingabe der Elemente ist nicht identisch mit der Reihenfolge der Ausgabe.
d) Kompilierfehler.

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Hashtable;
import java.util.Map;

public class GenericsHash {
	public static void main(String[ ] args) {
		String s1 = ”1”;
		String s2 = ”1”;
		String w1 = ”yes”;
		String w2 = ”no”;

		Hashtable<String, String> h = new Hashtable<String, String>();
		h.put(s1, w1);
		h.put(s2, w2);

		for(StringBuffer key : h.keySet()){
			System.out.println(key );
			System.out.println(h.get(key) );
		}
	}
}
```

a) 1 no
b) 1 yes
c) 1 no 1 yes
d) Kompilierfehler

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Arrays;

public class ArraysB {
	public static void main(String[ ] args) {
		String[ ] s = {”alpha”, ”omega”, ”emil”};
		int i = Arrays.binarySearch(s, ”emil”);
		System.out.println(i);
	}
}
```

a) 0
b) 1
c) 2
d) Ergebnis kann nicht exakt vorausgesagt werden.

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Arrays;

public class ArraysB {
	public static void main(String[ ] args) {
		String[ ] s = {”alpha”, ”omega”, ”emil”};
		Arrays.sort(s);
		int i = Arrays.binarySearch(s, ”emil”);
		System.out.println(i);
	}
}
```

a) 0
b) 1
c) 2
d) Ergebnis kann nicht exakt vorausgesagt werden.

(21) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListB {
	public static void main(String[ ] args) {
		ArrayList<String> s = new ArrayList<String>();
		String st = new String(”computer”);
		String st1 = new String(”haus”);
		String st2 = new String(”baum”);
		s.add(st);
		s.add(st1);
		s.add(st2);
		Collections.sort(s);
		int i = Collections.binarySearch(s, ”haus”);
		System.out.println(i);
	}
}
```

a) 0
b) 1
c) 2
d) Ergebnis kann nicht exakt vorausgesagt werden.

(22) Frage

Durch welche Methode muß das Wort hier ersetzt werden?

```
import java.util.LinkedList;
import java.util.Arrays;

public class CollectionListA {
	public static void main(String[ ] args) {
		String[ ] s = {"links", "rechts"};
		LinkedList<String> ll = new LinkedList<String>(Arrays.hier);

		for(String st: ll){
			System.out.println( st );
		}
	}
}
```

a) toArray(s)
b) asList(s)
c) offer(s)
d) binarSearch(s)
e) sort(s)

(23) Frage

Was gibt die Methode poll() zurück, wenn die Queue leer ist?

a) Es wird eine Exception geworfen
b) false
c) null
d) true

8. Lösungen

(1) Frage

a

(2) Frage

b

(3) Frage

a

(4) Frage

f

(5) Frage
c

(6) Frage
a

(7) Frage
a

(8) Frage
f

(9) Frage
a

(10) Frage
e

(11) Frage
a

(12) Frage
a

(13) Frage
f

(14) Frage
b, Vgl. Kapitel Das Comparator und Comparable Interface.

(15) Frage
c, Vgl. Kapitel Das Comparator und Comparable Interface.

(16) Frage
a, Maps bestehen aus Feldern und Schlüsseln (keys). Jeden Schlüssel gibt es nur einmal, was zur Folge hat, daß ein gleichlautender neuer Schlüssel einen alten ersetzt.

(17) Frage
c, Vgl. Kapitel Collections Framework.

(18) Frage

d, Achtung: In der for-Schleife steht der falsche Rückgabetyp, es müßte dort String als Rückgabetyp stehen.

(19) Frage

d

(20) Frage

b

(21) Frage

c

(22) Frage

b

(23) Frage

c

E. Enums

1. Begriffsbestimmung

Nehmen wir einmal an, Sie brauchen immer wieder die gleichen drei Anredeformen für Ihr Online-Formular, in das Sie Adressen eingeben. Seit Java 5.0 geht dies mit den sogenannten Enums. Hier ein Beispiel einer Aufzählung von möglichen Anreden in einem Menü.

Enums werden wie untenstehend formuliert. Mit Enums haben Sie Gelegenheit eine Enumeration, zu deutsch Aufzählung, festzulegen. Bitte beachten Sie, daß bei einem Enum, nicht class steht, sondern enum. Wichtig: Die Aufzählung wird durch einen Semikolon abgeschlossen und die Werte eines Emus sind implizit public static final.

```
public enum Anrede {
        Herr, Frau, Familie;
}
```

In Java 1.4 muß eine Aufzählung noch viel aufwendiger und weniger komfortable bestimmt werden:

```
public class AnredeAlt {
        public static final int Herr = 1;
        public static final int Frau = 2;
        public static final int Familie = 3;
}
```

Wenn Sie ein Enum definieren, erstellen Sie eine neue Klasse, die implizit eine extends-Beziehung zu der Klasse java.lang.Enum erzeugt. Und da jede Klasse nur eine extends-Beziehung haben kann, kann ein Enum per Definition keine zusätzliche extends-Beziehung besitzen. Außerdem implementiert die Klasse Enum sowohl das Serializable als auch das Comparable Interface. Ein Enum kann nicht instanziiert, sondern nur einer Referenzvariablen vom Typ Anrede zugewiesen werden.

```
public enum Anrede {
        Herr, Frau, Familie;

        public static void main(String[ ] args){
                // Zuweisung eines Enumwertes zu einer Referenzvariablen
                Anrede herr = Anrede.Herr;
                System.out.println(herr);
        }
}
```

Ausgabe auf der Konsole:

```
Herr
```

Wollen Sie für Ihr Enum keine separate Klasse erstellen, ist es ebenfalls denkbar, das Enum innerhalb einer Klasse zu definieren. Das Ergebnis ist dann eine statische innere Klasse.

2. Auslesen von Werten eines Enums

Wie können alle Werte eines Enums ausgegeben werden? Es gibt eine einfache Antwort: mit der Methode values() der Klasse java.lang.enum.

```
public enum Anrede {
        Herr, Frau, Familie;

        public static void main(String[ ] args){

                for(Anrede a:Anrede.values()){
                        System.out.println(a);
                }

        }
}
```

Ausgabe auf der Konsole:

```
Herr
Frau
Familie
```

Sie können aber auch nur einen bestimmten Teil aller Enum-Werte auslesen: Dies ist mit der statischen Methode EnumSet.range() der Klasse java.util.EnumSet möglich, die Sie dazu importieren müssen. Da eine teilweise Wiedergabe bei drei Elementen keinen Sinn macht, lassen Sie uns das Enum Anrede um eine weitere Anrede ergänzen, nämlich um die Anrede Firma. Nun wollen wir nur alle privaten Anredeformen ausgeben. Dies tun wir in untenstehendem Beispiel.

```
import java.util.EnumSet;

public enum Anrede {
	Herr, Frau, Familie, Firma;

	public static void main(String[ ] args){

		for(Anrede a:EnumSet.range(Anrede.Herr, Anrede.Familie)){
			System.out.println(a);
		}
	}
}
```

Ausgabe auf der Konsole:

```
Herr
Frau
Familie
```

3. Methoden, Variablen und Konstruktoren in einem Enum

a) Methoden

(1) Hinzufügen von Methoden

Was ist nun, wenn Sie einer Person einen Brief schreiben wollen, und automatisch die richtige Anrede (z. B. Sehr geehrte Frau,) zu Beginn des Textes erscheinen soll? Sie können einem Enum eine Methode hinzufügen. In unserem Beispiel ist dies die Methode public String anschreiben(){return „Sehr geehrte Damen und Herren, „;}. Diese kann für jeden Enum-Wert überschrieben werden: Sie können hinter den einzelnen Elementen in einer geschweiften Klammer, eine gleichnamige Methode, mit anderem Inhalt, einfügen (vgl. Kapitel Überladen und Überschreiben von Methoden).

```
public enum Anrede {
	Herr {public String anschreiben(){
		return “Sehr geehrter Herr, “;}
	},
	Frau {public String anschreiben(){
		return “Sehr geehrte Frau, “;}
	},
	Familie {public String anschreiben(){
		return “Sehr geehrte Familie, “;}
	},
	Firma;

	public static void main(String[ ] args){
		for(Anrede a: Anrede.values()){
			System.out.println(a + “ “ + a.anschreiben());
		}
```

```
        }
        public String anschreiben(){
                return “Sehr geehrte Damen und Herren, “;
        }
}
```

Ausgabe auf der Konsole:

```
Herr Sehr geehrter Herr,
Frau Sehr geehrte Frau,
Familie Sehr geehrte Familie,
Firma Sehr geehrte Damen und Herren,
```

(2) Überschreiben der toString()-Methode

Zusätzlich gibt es noch die Möglichkeit, den Inhalt von der Variablen Anrede. Herr, der bis zu diesem Augenblick „Herr“ ist, zu ändern. Wie geschieht dies? Sie überschreiben die bereits existierende toString()-Methode der Klasse java.lang. Enum in der geschweiften Klammer nach dem Enumerationswert. Was ist der Unterschied zu der Lösung, eigene Methoden einem Enum hinzuzufügen? Der Inhalt von Anrede.Herr wird von „Herr“ in „Sehr geehrter Herr,“ geändert. Diese Lösung würde dann wie folgt aussehen:

```
public enum Anrede {
        Herr {public String toString(){
                return “Sehr geehrter Herr,”;}
        },
        Frau {public String toString(){
                return “Sehr geehrte Frau,”;}
        },
        Familie {public String toString(){
                return “Sehr geehrte Familie,”;}
        },
        Firma{public String toString(){
                return “Sehr geehrte Damen und Herren,”;}
        };

        public static void main(String[ ] args){
                for(Anrede a: Anrede.values()){
                        System.out.println(a + ” ” + a.toString());
                }
                System.out.println(“Dies ist der Inhalt von Herr: “ + Anrede.Herr);
        }
}
```

Ausgabe auf der Konsole:

```
Sehr geehrter Herr, Sehr geehrter Herr,
Sehr geehrte Frau, Sehr geehrte Frau,
Sehr geehrte Familie, Sehr geehrte Familie,
Sehr geehrte Damen und Herren, Sehr geehrte Damen und Herren,
```

Dies ist der Inhalt von Herr: Sehr geehrter Herr,

b) Hinzufügen von Variablen und Konstruktoren

Wollen Sie z. B. alle Monate mit der entsprechenden Anzahl an Tagen ausgeben, ist dies möglich, indem Sie einen Konstruktor für die Enum-Klasse definieren, dem Sie die Tage als Parameter übergeben. Da Enumerationen Java-Klassen sind, haben sie auch Konstruktoren und Variablen. Hierbei gilt es allerdings zu beachten, daß die Konstruktoren nicht public sind und es somit auch nicht möglich ist, Instanzen eines Enums zu erzeugen.

```
public enum Monate {
        Januar(“31“), Februar(“28“), März(“31“);

        private String tage;

        Monate(String tage){
                this.tage = tage;
        }
}
```

Außerdem müssen Sie anschließend eine getTage()-Methode erstellen, die die entsprechenden Tage wieder ausliest.

```
public enum Monate {
        Januar(“31“), Februar(“28“), März(“31“);

        private String tage;

        Monate(String tage){
                this.tage = tage;
        }

        public String getTage(){
                return tage;
        }

        public static void main(String[ ] args){
                for (Monate m: Monate.values()){
                        System.out.println(m + ” ” + m.getTage());
                }
        }
}
```

Ausgabe auf der Konsole:

```
Januar 31
Februar 28
März 31
```

4. Enums und die switch-Struktur

Nehmen wir nun an, es soll je nach Monat ein anderer Satz ausgegeben werden. Dann ist die switch-Struktur die richtige Wahl (vgl. Kapitel Die switch-Struktur). Seither durfte der Ausdruck in der Klammer nach switch in der switch-Struktur nur vom primitiven Datentyp char, byte, short oder int sein (vgl. Kapitel Die switch-Struktur), seit Java 1.5 darf dieser Ausdruck auch ein Enum sein.

```
public enum Monate {
        Januar(“31“), Februar(“28“), März(“31“);

        private String tage;

        Monate(String tage){
                this.tage = tage;
        }

        public String getTage(){
                return tage;
        }

        public static void main(String[ ] args){
                Monate m = Monate.Januar;
                switch (m){
                case Januar: System.out.println(“Es ist kalt”); break;
                case Februar: System.out.println(“Es ist immer noch kalt”); break;
                case März: System.out.println(“Frühling”);
                }
        }
}
```

Ausgabe auf der Konsole:

Es ist kalt

Nach case darf in der switch-Struktur statt Januar nicht Monate.Januar stehen, da es in diesem Fal, wie Sie unten sehen können, zu einem Kompilierfehler kommt. Außerdem darf in der Klammer nach switch auch nicht Monate stehen.

```
3  public enum Monate {
4      Januar("31"), Februar("28"), Marz("31");
5
6      private String tage;
7
8      Monate(String tage){
9          this.tage = tage;
10     }
11
12     public String getTage(){
13         return tage;
14     }
15
16     public static void main(String[] args){
17         Monate m = Monate.Januar;
18         switch (m){
19         case Monate.Januar: System.out.println("Es ist kalt"); break;
20         case Februar: System.out.println("Es ist immer noch kalt"); break;
21         case Marz: System.out.println("Frühling");
22         }
23     }
24 }
```

Steht die switch-Struktur in einer nicht-statischen Methode kann der Ausdruck in der Klammer nach switch auch this sein (vgl. Objektreferenz und Konstruktoraufruf mit this und super), wobei sich this auf das Enum Monate bezieht.

```
public enum Monate {
	Januar("31"), Februar("28"), März("31");

	private String tage;

	Monate(String tage){
		this.tage = tage;
	}
	public String getTage(){
		return tage;
	}
	public void waehlen(){
		switch (this){
		case Januar: System.out.println("Es ist kalt"); break;
		case Februar: System.out.println("Es ist immer noch kalt"); break;
		case März: System.out.println("Frühling");
		}
	}

	public static void main(String[ ] args){
		Monate m = Monate.Februar;
		m.waehlen();
	}
}
```

Ausgabe auf der Konsole:

Es ist immer noch kalt

5. Die Methode valueOf()

Die Methode valueOf() der Klasse Enum gibt den konstanten Enumwert zurück. Dieser ist in unserem Fall das Wort Montag und dieses wird auf der Konsole ausgegeben.

```
public enum Wochentage {
	Montag, Dienstag, Mittwoch, Donnerstag, Freitag;

	public static void main(String[ ] args){
		Wochentage w = Wochentage.valueOf("Montag");
		System.out.println(w);
	}
}
```

Ausgabe auf der Konsole:

Montag

Was passiert nun, wenn Sie den konstanten Enumwert, den die valueOf()-Methode zurückgibt, mit dem Enumwert Wochentage.Montag vergleichen? Das Ergebnis ist sowohl beim Vergleich mit == als auch mit equals() true.

```
public enum Wochentage {
        Montag, Dienstag, Mittwoch, Donnerstag, Freitag;

        public static void main(String[ ] args){
                Wochentage w = Wochentage.valueOf("Montag");
                System.out.println(w == Wochentage.Montag);
                System.out.println(w.equals(Wochentage.Montag));
        }
}
```

Ausgabe auf der Konsole:

```
true
true
```

6. Der statische Import

Lassen Sie uns nun zu unserem ursprünglichen Beispiel des Enums Anrede zurückkommen, das wie untenstehend aussieht. Nehmen wir nun zusätzlich an, daß es sich im Package scjp befindet und Sie von einer anderen Klasse darauf zugreifen wollen, die sich in einem anderen Package befindet.

```
package scjp;

public enum Anrede {
        Herr, Frau, Familie;
}
```

Dies ist u. a. durch einen sogenannten statischen Import möglich. Ich habe Ihnen die verschiedenen Kombinationen der Arten des statischen und nicht-statischen Imports des Enums Anrede und der zugehörigen Zugriffsarten innerhalb der Klasse in untenstehender Tabelle zusammengestellt:

Import der Klasse	Zugriffsarten
import scjp.Anrede;	Anrede.Frau
import scjp.*;	Anrede.Frau
import static scjp.Anrede.*; oder import static scjp.Anrede.Frau;	Frau

Der statische Import funktioniert nicht nur bei Enums, sondern auch bei statischen Bestandteilen anderer Klassen. Hier nun ein Beispiel der statischen Methode tan()

der Klasse java.lang.Math und wie seither darauf zugegriffen wurde:

```
import java.lang.Math;

public class StatischerImport {
        public static void main(String[ ] args){
                        System.out.println(Math.tan(60));
        }
}
```

Die Einführung des statischen Imports vereinfacht den Zugriff auf die statische tan()-Methode:

```
import static java.lang.Math.*;

public class StatischerImport {
        public static void main(String[ ] args){
                        System.out.println(tan(60));
        }
}
```

Der Ausdruck import static java.lang.Math.*; kann auch durch import static java.lang.Math.tan; ersetzt werden.

7. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Ampel {
	rot {public String toString(){
		return ”Stehenbleiben!“;}
	},
	orange {public String toString(){
		return ”Warten!“;}
	},
	grün {public String toString(){
		return ”Loslaufen!“;}
	};
	public static void main(String[ ] args){
		for(Ampel a: Ampel.values()){
			System.out.println(a);
		}
	}
}
```

a) rot orange grün
b) Stehenbleiben! Warten! Loslaufen!
c) Kompilierfehler

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Ampel {
	rot {public String umschalten(){
		return “Stehenbleiben!“;}
	},
	orange {public String umschalten(){
		return “Warten!“;}
	},
	grün {public String umschalten(){
		return “Loslaufen!“;}
	},
	aus;
	public static void main(String[ ] args){
		for(Ampel a: Ampel.values()){
			System.out.println(a);
		}
	}
	public String umschalten(){
		return “Ampel ist kaputt! “;
	}
}
```

a) rot orange grün aus
b) rot orange grün
c) Stehenbleiben! Warten! Loslaufen!
d) Stehenbleiben! Warten! Loslaufen! Ampel ist kaputt!
e) Kompilierfehler

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Ampel {
        rot {public String umschalten(){
                return “Stehenbleiben!“;}
        },
        orange {public String umschalten(){
                return “Warten!“;}
        },
        grün {public String umschalten(){
                return “Loslaufen!“;}
        },
        aus;
        public static void main(String[ ] args){
                for(Ampel a: Ampel.values()){
                        System.out.println(a.umschalten());
                }
        }
        public String umschalten(){
                return “Ampel ist kaputt! “;
        }
}
```

a) rot orange grün aus
b) rot orange grün
c) Stehenbleiben! Warten! Loslaufen!
d) Stehenbleiben! Warten! Loslaufen! Ampel ist kaputt!
e) Kompilierfehler

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Ampel {
        rot {public String umschalten(){
                return “Stehenbleiben!“;}
        },
        orange {public String umschalten(){
                return “Warten!“;}
        },
        grün {public String umschalten(){
                return ”Loslaufen!“;}
        },
```

```
        aus
        public static void main(String[ ] args){
                for(Ampel a: Ampel.values()){
                        System.out.println(a.umschalten());
                }
        }
        public String umschalten(){
                return “Ampel ist kaputt! “;
        }}
```

a) rot orange grün aus
b) rot orange grün
c) Stehenbleiben! Warten! Loslaufen!
d) Stehenbleiben! Warten! Loslaufen! Ampel ist kaputt!
e) Kompilierfehler

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.EnumSet;

public enum Ampel {
        rot , orange , grün , aus;
        public static void main(String[ ] args){
                for(Ampel a: EnumSet.range(Ampel.rot, Ampel.grün)){
                        System.out.println(a);}}}
```

a) rot orange grün aus
b) rot orange grün
c) rot orange
d) rot
e) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Ampel {
        rot , orange , grün , aus;
        public static void main(String[ ] args){
                        Ampel rot = Ampel.rot;
                        System.out.println(rot);
        }
}
```

a) rot orange grün aus
b) rot orange grün
c) rot orange
d) rot

e) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März("31"), April("30"), Mai("31");
        private String tage;
        Monate(String tage){
                this.tage = tage;
        }
        public String getTage(){
                return tage;
        }
        public void waehlen(){
                switch (this){
                case März: System.out.println("Frühling"); break;
                case April: System.out.println("Aprilwetter"); break;
                case Mai: System.out.println("Sonnenschein");
                }
        }
        public static void main(String[ ] args){
                Monate m = Monate.März;
                m.waehlen();
        }
}
```

a) Frühling Aprilwetter Sonnenschein
b) Frühling
c) Aprilwetter
d) Sonnenschein
e) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März("31"), April("30"), Mai("31");
        private String tage;
        Monate(String tage){
                this.tage = tage;
        }
        public String getTage(){
                return tage;
        }
        public void waehlen(){
                switch (this){
                case Monate.März: System.out.println("Frühling"); break;
                case Monate.April: System.out.println("Aprilwetter"); break;
                case Monate.Mai: System.out.println("Sonnenschein");
```

```
                }
        }
        public static void main(String[ ] args){
                Monate m = Monate.März;
                m.waehlen();
        }
}
```

a) Frühling Aprilwetter Sonnenschein
b) Frühling
c) Aprilwetter
d) Sonnenschein
e) Kompilierfehler

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März("31"), April("30"), Mai("31");
        private String tage;
        Monate(String tage){
                this.tage = tage;
        }
        public String getTage(){
                return tage;
        }
        public static void main(String[ ] args){
                Monate m = Monate.März;
                switch (m){
                case März: System.out.println("Frühling"); break;
                case April: System.out.println("Aprilwetter"); break;
                case Mai: System.out.println("Sonnenschein");
                }
        }
}
```

a) Frühling Aprilwetter Sonnenschein
b) Frühling
c) Aprilwetter
d) Sonnenschein
e) Kompilierfehler

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März("31"), April("30"), Mai("31");
        private String tage;
        Monate(String tage){
```

```
                this.tage = tage;
        }
        public String getTage(){
                return tage;
        }
        public static void main(String[ ] args){
                for(Monate m: Monate.values()){
                        System.out.println(m.getTage());
                }}
}
```

a) März April Mai
b) 31 30 31
c) Kokmpilierfehler

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März("31"), April("30"), Mai("31");
        private String tage;
        Monate(String tage){
                this.tage = tage;
        }
        public String getTage(){
                return tage;
        }
        public static void main(String[ ] args){
                for(Monate m: Monate.values()){
                        System.out.println(m);
                }}}
```

a) März April Mai
b) 31 30 31
c) Kokmpilierfehler

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public enum Monate {
        März, April, Mai;
        public static void main(String[ ] args){
                Monate m = Monate.valueOf("April");
                System.out.println(m == Monate.April);
                System.out.println(m.equals(Monate.April));
        }
}
```

a) true true

b) true false
c) false true
d) Kompilierfehler

8. Lösungen

(1) Frage
b

(2) Frage
a

(3) Frage
d

(4) Frage
e

(5) Frage
b

(6) Frage
d

(7) Frage
b

(8) Frage
e

(9) Frage
b

(10) Frage
b

(11) Frage
a

(12) Frage
a

F. Bearbeiten von Text

Die Klassen String und StringBuffer haben wir ja bereits im Kapitel Standardklassen kennengelernt. Mit der neuen Java-Version wurden zusätzliche Klassen und Methoden, die uns Mittel und Wege für den Umgang mit Text bereitstellen, eingeführt.

1. Die Klasse StringBuilder

Die neu eingeführte Klasse StringBuilder ist fast identisch mit der Klasse StringBuffer, beide implementieren das java.lang.Appendable Interface und ihre Objekte sind im Gegensatz zur Klasse String veränderbar (vgl. die Kapitel über die Klasse String und StringBuffer). Wodurch unterscheiden sich die Klassen StringBuilder und StringBuffer? Objekte der Klasse StringBuilder sind nicht threadsafe (vgl. Kapitel Threads), aber schneller. So ist es sinnvoller, Objekte der Klasse StringBuilder zu nehmen, wenn nur ein Thread auf Ihr Objekt zugreift. Dies ist problemlos möglich, da beide Klassen über die gleichen Methoden verfügen.

2. Die Klasse Scanner und die Verwendung von regulären Ausdrücken

a) Text aufsplitten mit Hilfe der Klasse Scanner

Die Klasse Scanner ermöglicht es Ihnen, Text nach bestimmten Zeichen oder Zeichenketten abzusuchen (to scan), um ihn dann anschließend in Einzelteile zu zerlegen. Dies ist z. B. notwendig, wenn im Internet Daten, wie Adressen, in ein Formular eingegeben und weitergesendet werden. Diese Daten werden durch ein „&“ verknüpft und auf die nächste Seite weitergeleitet, dort müssen die Adreßdaten wieder herausgefiltert werden. Dies geschieht mit einem Objekt der Scanner-Klasse und einem Delimiter (Trennsymbol).

Kommen wir zu einem Beispiel: Der String „Michael&Bauer&Sonnenweg 2-&10000&Berlin“ wird einem Scanner-Objekt übergeben, auf dem dann die Methode useDelimiter(String pattern) ausgeführt wird. Diese Methode splittet mit Hilfe des Trennungszeichens „&“ den String in seine Einzelteile und erstellt daraus ein Array. Zwei Methoden helfen Ihnen nun das Array auszulesen: Die Methode hasNext(), die am Ende jedes Elementes feststellt, ob noch ein zusätzliches Element existiert und die Methode next(), die jedes Element ausgibt.

```
import java.util.Scanner;

public class StringScanner {
	public static void main(String[ ] args) {
		String s = “Michael&Bauer&Sonnenweg 2&10000&Berlin”;
		String d = ”&“;
		Scanner scanner = new Scanner(s);
		scanner.useDelimiter(d);
		while (scanner.hasNext()){
```

```
                System.out.println(scanner.next());
            }
        }
}
```

Ausgabe auf der Konsole:

```
Michael
Bauer
Sonnenweg 2
10000
Berlin
```

Mit scanner.close() am Ende können eventuell geöffnete Datenströme (aus einer Datei oder von einem Netzwerk) wieder geschlossen werden.

b) Verwendung von regulären Ausdrücken

(1) Anwendungsbeispiele

Es gibt sogenannte reguläre Ausdrücke, die es Ihnen erleichtern sollen, Zeichen oder Zeichenketten zu suchen. Bleiben wir bei obenstehendem Beispiel: Wir stellen uns nun vor, die einzelnen Daten wären nicht durch ein „&“ getrennt, sondern durch ein Leerzeichen, dann wäre ein regulärer Ausdruck, der Leerzeichen aufspürt, sehr hilfreich. Der reguläre Ausdruck, der dies tut, ist folgender: „\s“. Bitte beachten Sie, daß in dem String d= „\\s“; zweimal der Backslash steht, dies ist kein Tippfehler, vielmehr Absicht, da der Compiler anderenfalls den regulären Ausdruck nicht als solchen erkennt und es zu einem Kompilierfehler kommen würde. Wie Sie sehen können, ist es in diesem Fall nicht sinnvoll, diesen regulären Ausdruck zu nehmen: Er spaltet nicht nur die Adresse in Einzelteile auf, sondern er trennt zusätzlich auch die Hausnummer von der Straße.

```
import java.util.Scanner;

public class StringScanner {
    public static void main(String[ ] args) {
        String s = “Michael Bauer Sonnenweg 2 10000 Berlin”;
        String d = ”\\s“;
        Scanner scanner = new Scanner(s);
        scanner.useDelimiter(d);
        while (scanner.hasNext()){
            System.out.println(scanner.next());
        }
    }
}
```

Ausgabe auf der Konsole:

```
Michael
Bauer
Sonnenweg
```

```
2
10000
Berlin
```

Nehmen wir an, die einzelnen Adreßdaten wären durch „xx“ von einander getrennt, dann könnte man die Zeichenkette mit x+ zerlegen. Der reguläre Ausdruck „+“ nach einem x sucht nach einer Zeichenkette, in der ein x, mindestens einmal vorkommt.

```
import java.util.Scanner;

public class StringScanner {
	public static void main(String[ ] args) {
		String s = “MichaelxxBauerxxSonnenweg 2xx10000xxBerlin”;
		String d = ”x+“;
		Scanner scanner = new Scanner(s);
		scanner.useDelimiter(d);
		while (scanner.hasNext()){
			System.out.println(scanner.next());
		}
	}
}
```

Ausgabe auf der Konsole:

```
Michael
Bauer
Sonnenweg 2
10000
Berlin
```

Was passiert, wenn Sie statt x+, x* schreiben? Das Ergebnis wird in einzelne char-Zeichen zerlegt und nicht in einzelne Strings.

```
import java.util.Scanner;

public class StringScanner {
		public static void main(String[ ] args) {
				String s = “MichaelxxBauerxxSonnenweg 2xx10000xxBerlin”;
				String d = ”x*“;
				Scanner scanner = new Scanner(s);
				scanner.useDelimiter(d);
				while (scanner.hasNext()){
					System.out.println(scanner.next());
				}
		}
}
```

Ausgabe auf der Konsole (Auszug):

M
i
c
h
a
e
l
B
a
u

Kann man auch eine bestimmte Abfolge von Zeichen als Delimiter einsetzen? Ja! So ist es möglich, z. B. nach 2 aufeinanderfolgenden Ziffern und einem anschließenden Leerzeichen mit folgendem regulärem Ausdruck suchen: String d = “\\d\\d\\s“;.

```
import java.util.Scanner;

public class StringScanner {
	public static void main(String[ ] args) {
		String s = ”11 elf 12 zwölf 13 dreizehn”;
		String d = ”\\d\\d\\s”;
		Scanner scanner = new Scanner(s);
		scanner.useDelimiter(d);
		while (scanner.hasNext()){
			System.out.println(scanner.next());
		}
	}
}
```

Ausgabe auf der Konsole:

```
elf
zwölf
dreizehn
```

(2) Die Methode findInLine()

Lassen Sie uns mit der Methode findInLine() fortfahren: Sie durchsucht einen String nach einer bestimmten Zeichenfolge. So findet in unserem Beispiel der reguläre Ausdruck „(\\w+) 2“ eine Zeichenkette, die aus Buchstaben oder Zahlen besteht, und von einem Leerzeichen und der Zahl 2 gefolgt wird: das Wort Sonnenweg. Und „(\\d+) Berlin“ findet mehrere Ziffern, die einem Leerzeichen dem Wort Berlin vorausgehen: die Zahl 10000.

```
import java.util.Scanner;
import java.util.regex.MatchResult;

public class StringScannerFindInLine {
```

```
        public static void main(String[ ] args) {
                String s = “Michael Bauer Sonnenweg 2 10000 Berlin”;
                Scanner scanner = new Scanner(s);
                //findInLine() findet eine bestimmte Zeichenfolge
                scanner.findInLine(“(\\w+) 2 (\\d+) Berlin“);
                //match() speichert die Ergebnisse in einem MatchResult
                MatchResult ms = scanner.match();
                //groupCount() zählt die gefundenen Ergebnisse
                for(int i = 1; i<=ms.groupCount();i++){
                        //group() gibt jedes einzelne Ergebnis aus
                        System.out.println(ms.group(i));
                }
        }
}
```

Ausgabe auf der Konsole:

```
Sonnenweg
10000
```

(3) Zusammenfassung wichtiger regulärer Ausdrücke

Hier eine zusammenfassende Aufstellung aller für die Zertifizierung wichtigen regulären Ausdrücke:

Regulärer Ausdruck	Beschreibung
\s	Findet alle Leerzeichen, Tabulatoren und Zeilenumbrüche.
\d	Findet alle Ziffern („0“ bis „9“).
\w	Findet alle Buchstaben und Ziffern.
.	Findet alles vom Typ character.
*	Findet null oder mehrere vom gleichen Zeichen.
+	Findet eines oder mehrere vom gleichen Zeichen.
?	Findet null oder eines vom gleichen Zeichen.
[a-e]	Findet die Buchstaben a, b, c, d und e.

3. Die Klassen Pattern und Matcher

a) Die Methoden compile() und matcher()

Wie sucht man nach einem bestimmten Muster (Pattern) in einem String? Mit Hilfe der Klasse Pattern und Matcher. Mit dem regulären Ausdruck (\\w+) 1 wird nach einer Zeichenreihenfolge von Buchstaben oder Ziffern mit einem nachfolgenden Leerzeichen und der Ziffer 1 gesucht. Die Methode compile() der Klasse Pattern erstellt daraus ein Muster, das als Parameter der Methode matcher() übergeben wird und ein Objekt der Klasse Matcher zurückgibt. Mit diesem Muster sucht die Methode find() der Klasse Matcher in einem String nach einer Entsprechung, die

mit Hilfe der Methode group() auf dem Bildschirm ausgegeben wird.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		//compile() erstellt ein Muster
		Pattern pa = Pattern.compile("(\\w+) 1");
		//Muster wird der Methode matcher() übergeben
		Matcher ma = pa.matcher("zwei 1 drei 1 vier");
		//find() findet die Ergebnisse
		while(ma.find()){
			//group() gibt jedes einzelne Ergebnis aus
			System.out.println(ma.group());
		}
	}
}
```

Ausgabe auf der Konsole:

```
zwei 1
drei 1
```

b) Die Methode replaceAll()

Ist es möglich, mehrere Zeichenketten durch eine andere zu ersetzen? Ja! Mit der Methode replaceAll() der Klasse Matcher. In untenstehendem Beispiel werden alle Entsprechungen des erstellten Musters gefunden und durch das Wort „neu“ ersetzt.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		Pattern pa = Pattern.compile("(\\w+) 1");
		Matcher ma = pa.matcher("zwei 1 drei 1 vier");
			System.out.println(ma.replaceAll("neu"));
	}
}
```

Ausgabe auf der Konsole:

```
neu neu vier
```

c) Die Methode matches()

Wenn Sie feststellen wollen, ob eine bestimmte Zeichenkette mit dem gesamten String identisch ist, können Sie dies mit der Methode matches() der Klasse Matcher. Sind zwei Strings vom Inhalt her identisch wird true zurückgeben, wenn nicht false.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		Pattern pa = Pattern.compile("zwei 1 drei 1 vier");
		Matcher ma = pa.matcher("zwei 1 drei 1 vier");
		boolean bo = ma.matches();
		System.out.println(bo);
		}
	}
}
```

Ausgabe auf der Konsole:

```
true
```

d) Die Methode split() der Klasse Pattern

Wollen Sie einen Text in Einzelteile zerlegen, können Sie dies mit der Methode split() der Klasse Pattern.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		Pattern pa = Pattern.compile("1");
		String[ ] st = pa.split("zwei1drei1vier");
		for(String i:st){
			System.out.println(i);
		}
	}
}
```

Ausgabe auf der Konsole:

```
zwei
drei
vier
```

4. Die Methode split() der Klasse String

Die Methode public String[ ] split(String regex) der Klasse String hat die gleiche Funktion wie die soeben kennengelernte gleichnamige Methode der Klasse Pattern.

```
public class StringSplit {
	public static void main(String[ ] args) {
		String s = ”H1u2p“;
		String d = ”\\d“;
		String[ ] st = s.split(d);
		for (String i:st){
			System.out.println(i);
		}
	}
}
```

Ausgabe auf der Konsole:

```
H
u
p
```

5. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		Pattern pa = Pattern.compile("(\\w+)@");
		Matcher ma = pa.matcher("mein@dein.de");
		System.out.println(ma.replaceAll("hallo"));
	}}
```

a) hallodein.de
b) meinhallodein.de
c) mein@dein.de
d) Keine dieser Möglichkeiten.

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
	public static void main(String[ ] args) {
		Pattern pa = Pattern.compile("(\\w+?@");
		Matcher ma = pa.matcher("mein@dein.de");
		while (ma.find()){
			System.out.println(ma.group());
		}
	}
}
```

a) @
b) mein@
c) @dein.de
d) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
```

```
        public static void main(String[ ] args) {
                Pattern pa = Pattern.compile("(\\w+)@");
                Matcher ma = pa.matcher("mein@dein.de");
                boolean bo = ma.matches();
                System.out.println(bo);
        }
}
```

a) mein@
b) @
c) true
d) false
e) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PattMatch {
        public static void main(String[ ] args) {
                Pattern pa = Pattern.compile("@");
                String[ ] s = pa.split("mein@dein.de");
                for(String st: s){
                        System.out.println(st);
                }
        }
}
```

a) mein dein.de
b) mein@dein.de
c) mein
d) dein.de

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class StringSplit {
        public static void main(String[ ] args) {
                String s = "hallo22neu33mein";
                String d = "\\d+";
                String[ ] st = s.split(d);
                for (String i:st){
                        System.out.println(i);
                }
        }
}
```

a) hallo neu mein
b) Das Ergebnis wird in einzelne char-Zeichen zerlegt und nicht in einzelne Strings.
c) Keine dieser Möglichkeiten.

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class StringSplit {
        public static void main(String[ ] args) {
                String s = ”hallo22neu33mein“;
                String d = ”\\d*“;
                String[ ] st = s.split(d);
                for (String i:st){
                        System.out.println(i);
                }}}
```

a) hallo neu mein
b) Das Ergebnis wird in einzelne char-Zeichen zerlegt und nicht in einzelne Strings.
c) Keine dieser Möglichkeiten.

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Scanner;

public class StringScanner {
        public static void main(String[ ] args) {
                String s = ”11 elf 12 zwölf”;
                String d = ”\\s”;
                Scanner scanner = new Scanner(s);
                scanner.useDelimiter(d);
                while (scanner.hasNext()){
                        System.out.println(scanner.next());
                }
        }
}
```

a) 11 elf 12 zwölf
b) elf zwölf
c) 11 12

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Scanner;
```

```
public class StringScanner {
	public static void main(String[ ] args) {
		String s = ”11elf12zwölf”;
		String d = ”\\d+”;
		Scanner scanner = new Scanner(s);
		scanner.useDelimiter(d);
		while (scanner.hasNext()){
			System.out.println(scanner.next());
		}
	}
}
```

a) 11 elf 12 zwölf
b) elf zwölf
c) 11 12

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Scanner;
import java.util.regex.MatchResult;

public class StringScannerFindInLine {
	public static void main(String[ ] args) {
		String s = “Michael Bauer Sonnenweg 2 10000 Berlin”;
		Scanner scanner = new Scanner(s);
		scanner.findInLine(“(\\w+) Bauer”);
		MatchResult ms = scanner.match();
		for(int i = 1; i<=ms.groupCount();i++){
			System.out.println(ms.group(i));
		}
	}}
```

a) Michael Bauer
b) Michael
c) Bauer
d) Michael Bauer Sonnenweg 2 10000 Berlin

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Scanner;
import java.util.regex.MatchResult;

public class StringScannerFindInLine {
	public static void main(String[ ] args) {
		String s = “Michael Bauer Sonnenweg 2 10000 Berlin”;
		Scanner scanner = new Scanner(s);
```

```
                    scanner.findInLine("(\\w+) 2");
                    MatchResult ms = scanner.match();
                    for(int i = 1; i<=ms.groupCount();i++){
                              System.out.println(ms.group(i));
                    }
          }
}
```

a) Sonnenweg
b) 10000
c) Sonnenweg 2

6. Lösungen

(1) Frage
a

(2) Frage
b

(3) Frage
d

(4) Frage
a

(5) Frage
a

(6) Frage
b

(7) Frage
a

(8) Frage
b

(9) Frage
b

(10) Frage
a

G. Formatieren von Text, Zahlen und Datum

1. Die Klasse Locale

Die Klasse Locale dient dazu, Ihnen eine Reihe von länderspezifischen Formatierungen zur Verfügung zu stellen. So wird Sie z. B. benötigt, um Zahlen in der entsprechenden Landeswährung (1.000,00 €) zu formatieren. Es gibt eine Anzahl an vordefinierten konstanten Locale-Kennungen. Sie bestehen aus dem Begriff Locale + "." + dem englischen Namen des Landes. Hier eine Auswahl:

Land	**Locale**
Frankreich	Locale.FRANCE
Deutschland	Locale.GERMANY
USA	Locale.US
Großbritannien	Locale.UK
Italien	Locale.ITALY
Japan	Locale.JAPAN

Es gibt leider keine Möglichkeit, diese auf der Konsole auszugeben, da die Methode getDisplayCountry() zwar dafür vorgesehen ist, aber der Fehler gemacht wurde, auch die Namen der Länder zu übersetzen. Dies ist an dieser Stelle nicht sinnvoll, da für die Locale der englische Name des entsprechenden Landes notwendig ist. Hier einige der wichtigsten Methoden der Klasse Locale:

```
import java.util.Locale;

public class Laenderkennzahlen {
   public static void main(String[ ] args) {
      //Abrufen aller zur Verfügung stehenden Locale-Konstanten
      for (Locale l: Locale.getAvailableLocales()){
         //Abrufen der Länder, Länderabkürzungen und der entsprechenden Sprache
         System.out.println(l.getDisplayCountry() + " " + l.getCountry()+ " + " +
         l.getLanguage());
      }
   }
}
```

Ausgabe auf der Konsole (ein Ausschnitt):

```
Belgien: BE + fr
Kanada: CA + fr
Schweiz: CH + fr
Frankreich: FR + fr
Luxemburg: LU + fr
```

Des weiteren können Sie mit getDefault() die jeweilige Länderkennung feststellen

und mit setDefault() können Sie sie verändern.

a) Konstruktoren der Klasse Locale

Wie können Sie eine Locale-Konstante erstellen? Mit einem Konstruktor der Klasse Locale. Es gibt zwei Konstruktoren:

```
import java.util.Locale;

public class KonstruktorenLocale {
        public static void main(String[ ] args) {
                //Sprachabkürzung als Parameter
                Locale l = new Locale("en");
                //Sprachabkürzung und Länderabkürzung als Parameter
                Locale l1 = new Locale("en", "US");
        }
}
```

Sie können aber im Falle von USA auf die bereits existierende Locale zugreifen und wie folgt ein neues Objekt erstellen: Locale l = Locale.US;

2. Formatieren von Text und Zahlen

a) Formatieren mit der Methode format() der Klasse String

Wie können Sie Integer-Zahlen mit einem Tausenderpunkt (100.000.000) versehen? Mit der Methode format() der Klasse String. Dieser Methode müssen Sie einen Format-Parameter und eine Integer-Zahl übergeben. Der Format-Parameter „%d" formatiert Integer-Zahlen, allerdings ohne Tausenderpunkte (vgl. Aufstellung weiter unten), und „%,d" formatiert die Integer-Zahl mit Tausenderpunkten. Warum steht dort ein Komma und kein Punkt? Werden Sie sich jetzt sicherlich fragen. Leider werden deutsche und amerikanische Zahlenformate nicht sauber getrennt. Sollten Sie die Prüfung auf englisch/amerikanisch machen, gehen Sie bitte immer von amerikanischen Zahlenformaten aus.

```
import java.util.Locale;

public class FormatierenTausend {
        public static void main(String[ ] args) {
                Integer i = new Integer(100000000);
                String st = String.format("%,d", i);
                System.out.println(st);
        }
}
```

Ausgabe auf der Konsole:

100.000.000

Wollen Sie jedoch tatsächlich eine Integer-Zahl in amerikanischem Zahlenformat formatieren, brauchen Sie die überladene format()-Methode, die es Ihnen ermöglicht, Zahlen mit einer Locale länderspezifisch anzupassen.

```
import java.util.Locale;

public class Formatieren {
  public static void main(String[ ] args) {
    Integer i = new Integer(100000000);
    String st = String.format("So wird i formatiert mit Locale.GERMANY: %,d", i);
    String st1 = String.format(Locale.US, "So wird i formatiert mit Locale.US: %,d", i);
    System.out.println(st);
    System.out.println(st1);
  }
}
```

Ausgabe auf der Konsole:

```
So wird ein Integer formatiert mit Locale.GERMANY:        100.000.000
So wird ein Integer formatiert mit Locale.US:             100,000,000
```

Zusätzliche Probleme gibt es auch beim Formatieren von Fließkommazahlen: Der Konstruktor des Wrappers Double erwartet eine Nachkommazahl im amerikanischen Zahlenformat, wohingegen die format()-Methode eine Zahl im deutschen Zahlenformat wiedergibt. Wollen Sie, daß eine Zahl im amerikanischen Zahlenformat formatiert wird, müssen Sie dies separat festlegen. Sie müssen in diesem Fall die überladene format()-Methode nehmen, die Sie zusätzlich die Locale festlegen läßt.

```
import java.util.Locale;

public class Formatieren {
    public static void main(String[ ] args) {
        Double d = new Double(3400.667);
        String str = String.format("So wird d formatiert mit Locale.GERMANY: %.2f", d);
        String str1 = String.format(Locale.US, "So wird d formatiert mit Locale.US: %.2f", d);
        System.out.println(str);
        System.out.println(str1);
    }
}
```

Ausgabe auf der Konsole:

```
So wird ein Double formatiert mit Locale.GERMANY:         3400,67
So wird ein Double formatiert mit Locale.US:              3400.67
```

Wie sie in obenstehenden Beispiel sehen, können Sie mit „%.2f“ Double-Zahlen formatieren, wobei „%.2f“ zwei Nachkommastellen festlegt und die Nachkommastellen aufgerundet werden. Der Formatierungs-Parameter enthält

einen Punkt und kein Komma, da er sich am amerikanischen Zahlenformat orientiert.

Jetzt kommt die nächste Frage: Kann man mit Formatierungs-Parametern auch Text formatieren? Ja, mit „%s“. Hierbei wird ein Objekt mit der toString()-Methode in ein String-Objekt umgewandelt. Wollen Sie als Ergebnis ein Wort haben, das nur aus Großbuchstaben besteht, müssen Sie %S nehmen.

```
public class FormatierenString {
   public static void main(String[ ] args) {
            String st = String.format("Es wird alles in Kleinbuchstaben formatiert: %s","ich");
            String st1 = String.format("Es wird alles in Großbuchstaben formatiert: %S","ich");
            System.out.println(st);
            System.out.println(st1);
   }
}
```

Ausgabe auf der Konsole:

```
Es wird alles in Kleinbuchstaben formatiert: ich
Es wird alles in Großbuchstaben formatiert: ICH
```

Ebenso ist eine Formatierung von Boolean-Werten mit „%b“ realisierbar.

```
public class FormatierenBoolean {
            public static void main(String[ ] args) {
                        Boolean bo = new Boolean("true");
                        String st = String.format("Dies ist der Wert der Variablen bo: %b", bo);
                        System.out.println(st);
            }
}
```

Ausgabe auf der Konsole:

```
Dies ist der Wert der Variablen bo: true
```

Die prüfungsrelevanten Format-Spezifizierer beschränken sich auf ein paar wenige, die in untenstehender Tabelle zusammengefaßt sind:

Format-Spezifizierer	Beschreibung
%b	formatiert einen Boolean-Wert
%c	formatiert einen Character-Wert
%d	formatiert einen Integer-Wert
%f	formatiert einen Fließkomma-Wert
%s	formatiert ein Objekt, indem es das Objekt mit der toString()-Methode in ein String-Objekt umwandelt

b) Formatieren mit Hilfe der Formatter-Klasse

(1) Formatieren von Objekten der Klasse StringBuffer und StringBuilder

In den Klassen StringBuilder und StringBuffer gibt es keine Methode format(). Wenn Sie also Objekte dieser Klasse formatieren wollen, müssen Sie die Objekte dieser beiden Klassen dem Konstruktor der Formatter-Klasse Formatter (Appendable a) als Parameter übergeben. Wie Sie sehen, muß man diesem Konstruktor ein Objekt übergeben, daß das Appendable-Interface implementiert, und dies trifft auf beide Klassen zu. Die Reihenfolge der Bearbeitung bei der Formatierung können Sie beeinflussen, indem Sie nach dem „%“ wie untenstehend „1$“ schreiben. Hiermit legen Sie fest, daß zuerst das Wort Paris formatiert wird und nicht Eiffelturm.

```
import java.util.Formatter;

public class FormatterKlasse {
   public static void main(String[ ] args) {
            StringBuilder sb = new StringBuilder();
            StringBuffer s = new StringBuffer();
            Formatter f = new Formatter(sb);
            Formatter fo = new Formatter(s);

            System.out.println(f.format(“Das zweite Wort %2$s, das erste Wort %1$s “,
            “Eiffelturm”, “Paris”) );
            System.out.println(fo.format(“Das erste Wort %s, das zweite Wort %s “,
            “Eiffelturm”, “Paris”) );
   }
}
```

Ausgabe auf der Konsole:

```
Das zweite Wort Paris, das erste Wort Eiffelturm
Das erste Wort Eiffelturm, das zweite Wort Paris
```

Wie können Sie nun Zahlen mit der Klasse Formatter länderspezifisch formatieren, wenn Sie ein StringBuilder- oder StringBuffer-Objekt haben? Mit dem Konstruktor der Klasse Formatter, dem Sie ein Objekt der StringBuilder-Klasse und die entsprechend Locale übergeben können.

```
import java.util.Formatter;
import java.util.Locale;

public class FormatterKlasseLocale {
	public static void main(String[ ] args) {
		StringBuilder sb = new StringBuilder();
		Formatter fo = new Formatter(sb, Locale.US);
		Double d = new Double(564.5667);
		fo.format(”%.2f”, d);
		System.out.println(fo);
	}
}
```

Ausgabe auf der Konsole:

```
564.57
```

(2) Formatieren von String-Objekten

Die Formatter-Klasse stellt auch eine Methode zur Verfügung mit der Sie ein String-Objekt formatieren können: die format()-Methode.

```
import java.util.Formatter;

public class FormatterKlasseString {
	public static void main(String[ ] args) {
		String s = new String(”ich“);
		Formatter f = new Formatter();
		f.format(“Bitte in Großbuchstaben %S”, s);
		System.out.println(f);
	}
}
```

Ausgabe auf der Konsole:

```
Bitte in Großbuchstaben ICH
```

c) Die Methode printf()- und format()-Methode der Klassen PrintStream und PrintWriter

Mit den Methoden printf() und format() der Klassen PrintStream und PrintWriter kann man auch Text und Zahlen formatieren (vgl. Kapitel Ein- und Ausgabe über Datenströme: java.io):

```
public class FormatierenPrintf {
        public static void main(String[ ] args) {
                Integer i = new Integer(100000000);

                System.out.format("So wird ein Integer formatiert: %,d", i);
                System.out.println("");
                System.out.printf("So wird ein Integer formatiert: %,d", i);
        }
}
```

Ausgabe auf der Konsole:

```
So wird ein Integer formatiert: 100.000.000
So wird ein Integer formatiert: 100.000.000
```

3. Das Datumsformat

Zu der Kategorie Datum gehören drei wichtige Klassen, die unterschiedliche Aufgaben erfüllen:

- **Date**
  Objekte der Klasse Date speichern Datums- bzw. Zeitwerte (als 64-Bit-Long-Variable) in Millisekunden (gerechnet seit dem 1.1.1970), wobei ein negativer Wert ein Datum vor 1970 darstellt. Objekte dieser Klasse werden nur noch selten verwendet und ein Großteil seiner Methoden sind deprecated, sprich es wird empfohlen, diese nicht mehr zu verwenden.

- **DateFormat**
  Mit dieser Klasse wird die Ein- und Ausgabe von Datums- und Zeitwerten formatiert.

- **Calendar**
  Diese Klasse basiert auf dem Gregorianischen Kalender. Sie ist die erste Wahl, wollen Sie Berechnungen mit Datums- und Zeitwerten durchführen.

a) Allgemeine Datumsformate der Klasse DateFormat

In untenstehendem Beispiel der Klasse Datum habe ich eine Instanz der Klasse Date erstellt, die unter Zuhilfenahme der Klasse DateFormat formatiert wurde. Nur mit Methoden der Klasse DateFormat können Sie, das Datum in eine „gefällige" Form bringen: Sie müssen ein Date-Objekt der Klasse DateFormat als Parameter übergeben. Berücksichtigen Sie bitte, daß es sich bei der Klasse DateFormat um eine abstrakte Klasse handelt, die sich nicht mit new instanziieren läßt. Es ist nur ein statischer Zugriff auf die Methoden dieser Klasse möglich (vgl. Kapitel Interfaces und abstrakte Klassen).

```
import java.text.DateFormat;
import java.util.Date;

public class Datum {
   public static void main(String[ ] args) {
      Date d = new Date();

      //statischer Zugriff auf Methoden einer abstrakten Klasse
      DateFormat f = DateFormat.getInstance();
      DateFormat fo = DateFormat.getDateInstance();
      DateFormat fom = DateFormat.getTimeInstance();
      DateFormat form = DateFormat.getDateTimeInstance();

      System.out.println("Unformatiertes Datum und Datumsformat:");
      System.out.println("Unformatiertes Datum: " + d);
      System.out.println("Unformatiertes Datumsformat: " + f);

      System.out.println("Formatiertes Datum:");
      System.out.println("Datum und Uhrzeit kurz (getInstance()): " + f.format(d));
      System.out.println("Datum (getDateInstance()): " + fo.format(d));
      System.out.println("Uhrzeit (getTimeInstance())): " + fom.format(d));
      System.out.println("Datum und Uhrzeit lang (getDateTimeInstance()): "+form.format(d));
   }
}
```

Ausgabe auf der Konsole:

```
Unformatiertes Datum und Datumsformat:
Unformatiertes Datum: Wed Feb 08 10:24:22 CET 2006
Unformatiertes Datumsformat: java.text.SimpleDateFormat@3dd497a
Formatiertes Datum:
Datum und Uhrzeit kurz (getInstance()): 08.02.06 10:24
Datum (getDateInstance()): 08.02.2006
Uhrzeit (getTimeInstance())): 10:24:22
Datum und Uhrzeit lang (getDateTimeInstance()): 08.02.2006 10:24:22
```

Eine andere Zeitzone, z. B. die von Los Angeles, können Sie festlegen, indem Sie sie mit den Methoden getTimeZone() und setTimeZone() verändern und mit getTime() wieder abrufen.

```
import java.text.DateFormat;
import java.util.Date;
import java.util.TimeZone;

public class DatumAendern {
    public static void main(String[ ] args) {
            Date d = new Date();
            //statischer Zugriff auf Methoden einer abstrakten Klasse
            DateFormat form = DateFormat.getDateTimeInstance();

            System.out.println("Uhrzeit Deutschland: " + form.format(d));
            form.setTimeZone(TimeZone.getTimeZone("America/Los_Angeles"));
```

```
            System.out.println("Uhrzeit in Los Angeles: " + form.format(d.getTime()));
            form.setTimeZone(TimeZone.getTimeZone("Japan"));
            System.out.println("Uhrzeit in Japan: " + form.format(d.getTime()));
    }
}
```

Ausgabe auf der Konsole:

```
Uhrzeit Deutschland: 08.02.2006 10:38:06
Uhrzeit in Los Angeles: 08.02.2006 01:38:06
Uhrzeit in Japan: 08.02.2006 18:38:06
```

b) Kurzes und langes Datumsformat

Wie Sie unten sehen können, sind Datumsformate in unterschiedlicher Länge realisierbar. Dies geschieht unter Verwendung der Klasse DateFormat und deren Konstanten DateFormat.LONG, DateFormat.SHORT, DateFormat.FULL, DateFormat.MEDIUM (= DateFormat.DEFAULT).

```
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class DatumFormat {
   public static void main(String[ ] args) {
      Date d = new Date();

      //statischer Zugriff auf Methoden einer abstrakten Klasse
      DateFormat df1 = DateFormat.getDateInstance(DateFormat.LONG, Locale.US);
      DateFormat df2 = DateFormat.getDateInstance(DateFormat.SHORT, Locale.US);
      DateFormat df3 = DateFormat.getDateInstance(DateFormat.FULL, Locale.US);
      DateFormat df4 = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.US);

      System.out.println("LONG: " + df1.format(d));
      System.out.println("SHORT: " + df2.format(d));
      System.out.println("FULL: " + df3.format(d));
      System.out.println("MEDIUM: " + df4.format(d));
   }
}
```

Ausgabe auf der Konsole:

```
LONG: December 6, 2005
SHORT: 12/6/05
FULL: Tuesday, December 6, 2005
MEDIUM: Dec 6, 2005
```

c) Die Methode parse()

Die Methode parse( ) der Klasse DateFormat wird dazu verwendet, ein formatiertes Datum vom Typ String in ein Objekt vom Typ Date umzuwandeln. Das Datum vom Typ String muß allerdings exakt den vorgegebenen Formatierungen der jeweiligen Ländereinstellungen entsprechen. Ist dies nicht der Fall, kommt es zu einer Exception. Dies ist auch der Grund, warum diese Methode in einem try/catch-Block stehen muß.

```
import java.text.DateFormat;
import java.text.ParseException;
import java.util.Date;
import java.util.Locale;

public class DatumParse {
   public static void main(String[ ] args) {
      String s = ”May 11, 2005”;

      try {
            Date d = DateFormat.getDateInstance(DateFormat.LONG, Locale.US).parse(s);
            System.out.println(d);
      } catch (ParseException e) {
            e.printStackTrace();
      }
   }
}
```

Ausgabe auf der Konsole:

```
Wed May 11 00:00:00 CEST 2005
```

4. Zahlen- und Währungsformate

a) Zahlenformat

Wie kann man einen String in eine Zahl umwandeln? Mit der Methode parse( ) der Klasse Number. Und wie wird diese Zahl formatiert? Mit der Methode format() der Klasse NumberFormat. Das so erhaltene Objekt der Klasse Number wird dann unter Zuhilfenahme der Methode intValue() in eine Integerzahl umgewandelt.

Welche Auswirkungen haben die unterschiedlichen Länderkennzahlen, die mit der Methode getInstance() der Klasse NumberFormat hinzugefügt werden? Im deutschen Zahlenformat bedeutet das Komma eine Trennung der Zahl von seinen Nachkommastellen, wohingegen es im amerikanischen die Funktion eines Tausenderpunktes hat. So erhalten Sie bei der Locale.GERMANY als Ergebnis für den String “23,545“ die Zahl 23,545 und bei der Locale.US die Zahl 23545. Was für ein Unterschied! Achten Sie darauf, daß die Methode parse() immer in einem try/catch-Block stehen muß und der geparste String immer eine Zahl im amerikanischen Zahlenformat, basierend auf der entsprechenden Locale, wiedergibt.

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class ZahlenFormat {
	public static void main(String[ ] args) {
		//statischer Zugriff auf Methoden der Klasse NumberFormat
		NumberFormat nf = NumberFormat.getInstance(Locale.GERMANY);
		NumberFormat nf1 = NumberFormat.getInstance(Locale.US);

		try {
			Number n = nf.parse(”23,545“);
			/*in String wurde in eine Zahl vom Typ Number
			 umgewandelt, basierend auf Locale.GERMANY*/
			System.out.println(“Locale.Germany mit dem String 23,545:“);
			System.out.println(“parse():  “ + n);
			/*die Zahl vom Typ Number wird im deutschen
			Zahlenformat ausgegeben*/
			System.out.println(“format(): “+ nf.format(n));
			//n wird in eine Integer-Zahl umgewandelt
			System.out.println(”intValue(): ”+ n.intValue());

			Number n1 = nf1.parse(”23,545“);
			System.out.println(“Locale.US mit dem String 23,545:”);
			System.out.println(“parse(): “ + n1);
			System.out.println(“format(): “ + nf1.format(n));
			System.out.println(”intValue(): ” + n1.intValue());

		} catch (ParseException e) {
			e.printStackTrace();
		}
	}
}
```

Ausgabe auf der Konsole:

```
Locale.Germany mit dem String 23,545:
parse(): 23.545
format(): 23,545
intValue(): 23
Locale.US mit dem String 23,545:
parse(): 23545
format(): 23.545
intValue(): 23545
```

b) Währungsformat

Sie wollen Ihren String mit der entsprechenden Landeswährung formatieren? So brauchen Sie die statische getCurrencyInstance()-Methode und die Methode format() der abstrakten Klasse NumberFormat. Wollen Sie allerdings den umgekehrten Vorgang durchführen, so benötigen Sie die Methode parse(), die

den formatierten String wieder in eine Zahl umwandelt. Die Methode parse() muß immer in einem try/catch-Block stehen.

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class Waehrungsformat {
   public static void main(String[ ] args) {
            //statischer Zugriff auf Methoden einer abstrakten Klasse
            NumberFormat nf = NumberFormat.getCurrencyInstance(Locale.GERMANY);
            NumberFormat nf1 = NumberFormat.getCurrencyInstance(Locale.US);

            try {
                        Number n = nf.parse(”23,50 €“);
                        String s = nf.format(23.5);
                        System.out.println(”23,50 € geparst(Locale.GERMANY): ” + n);
                        System.out.println(”23.5 formatiert(Locale.GERMANY): ” + s);
                        Number n1 = nf1.parse(”$23.50“);
                        String s1 = nf1.format(23.5);
                        System.out.println(”$23.50 geparst(Locale.US): ” + n1);
                        System.out.println(”23.5 formatiert(Locale.US): ” + s1);
            } catch (ParseException e) {
                        e.printStackTrace();
            }
   }
}
```

Ausgabe auf der Konsole:

```
23,50 € geparst(Locale.GERMANY): 23.5
23.5 formatiert(Locale.GERMANY): 23,50 €
$23.50 geparst(Locale.US): 23.5
23.5 formatiert(Locale.US): $23.50
```

Wie Sie oben sehen können, haben Währungsformate immer zwei Nachkommastellen.

5. Übungen

(1) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class F {
        public static void main(String[ ] args) {
                Integer i = new Integer(1000);
                String st = String.format("%d", i);
                System.out.println(st);
        }
}
```

a) 1000
b) 1.000
c) 1,000

(2) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class F {
        public static void main(String[ ] args) {
                Integer i = new Integer(1000);
                String st = String.format("%,d", i);
                System.out.println(st);
        }
}
```

a) 1000
b) 1.000
c) 1,000

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class F {
        public static void main(String[ ] args) {
                Integer i = new Integer(1000);
                String st = String.format(Locale.US, "%,d", i);
                System.out.println(st);
        }
}
```

a) 1000
b) 1.000
c) 1,000

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class F {
        public static void main(String[ ] args) {
                Double d = new Double(10.789);
                String st = String.format(”%.2f“, d);
                System.out.println(st);
        }
}
```

a) 10,79
b) 10.79
c) 10,789
d) 10.789

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class F {
        public static void main(String[ ] args) {
                Double d = new Double(10.789);
                String st = String.format(Locale.US, ”%.2f“, d);
                System.out.println(st);
        }
}
```

a) 10,79
b) 10.79
c) 10,789
d) 10.789

(6) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Formatter;

public class FormKlasse {
        public static void main(String[ ] args) {
                StringBuffer s = new StringBuffer();
                Formatter f = new Formatter(s);
                System.out.println(f.format(”%2$s, %1$s“, ”Hallo“, ”Tschüss“) );
        }
}
```

a) Hallo Tschüss
b) HALLO TSCHÜSS

c) Tschüss Hallo
d) TSCHÜSS HALLO

(7) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Formatter;

public class FormKlasse {
	public static void main(String[ ] args) {
		StringBuffer s = new StringBuffer();
		Formatter f = new Formatter(s);
		System.out.println(f.format("%s, %s", "Hallo", "Tschüss") );
	}
}
```

a) Hallo Tschüss
b) Tschüss Hallo
c) Kompilierfehler

(8) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Locale;

public class F {
	public static void main(String[ ] args) {
		Double d = new Double(3.567);
		System.out.printf("%.2f", d);
	}
}
```

a) 3.567
b) 3,567
c) 3.57
d) 3,57

(9) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Locale;

public class F {
	public static void main(String[ ] args) {
		Integer i = new Integer(3097);
		System.out.format("%,d", i);
	}
```

```
}
```

a) 3097
b) 3.097
c) 3,097

(10) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Formatter;
import java.util.Locale;

public class FormatterKlasseLocale {
	public static void main(String[ ] args) {
		StringBuilder sb = new StringBuilder();
		Formatter fo = new Formatter(sb, Locale.US);
		Double d = new Double(4.754);
		fo.format( ”%.2f”, d);
		System.out.println(fo);
	}
}
```

a) 4.75
b) 4,75
c) 4.754
d) 4,754

(11) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.util.Formatter;

public class FormKlasse {
	public static void main(String[ ] args) {
		StringBuffer s = new StringBuffer();
		Formatter f = new Formatter(s);
		System.out.println(f.format(”%2$S, %1$S“, ”Hallo“, ”Tschüss“) );
	}
}
```

a) Hallo Tschüss
b) HALLO TSCHÜSS
c) Tschüss Hallo
d) TSCHÜSS HALLO

(12) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class ZahlForm {
	public static void main(String[ ] args) {
		NumberFormat nf = NumberFormat.getInstance(Locale.GERMANY);
		try {
			Number n = nf.parse(”6,798”);
			System.out.println(nf.format(n));
			System.out.println(n.intValue());
		} catch (ParseException e) {
			e.printStackTrace();
		}
	}
}
```

a) 6,798 6798
b) 6,798 6
c) 6.798 6798
d) 6.798 6

(13) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class ZahlForm {
	public static void main(String[ ] args) {
		NumberFormat nf = NumberFormat.getInstance(Locale.GERMANY);
		try {
			Number n = nf.parse(”6,798”);
			System.out.println(n);
		} catch (ParseException e) {
			e.printStackTrace();
		}
	}
}
```

a) 6,798
b) 6.789
c) 6

(14) <u>Frage</u>

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.NumberFormat;
```

```
import java.text.ParseException;
import java.util.Locale;

public class Wformat {
	public static void main(String[ ] args) {
		NumberFormat n = NumberFormat.getCurrencyInstance(Locale.US);
		try {
			Number n1 = n.parse("$2.60");
			String s1 = n.format(1.5);
			System.out.println(n1);
			System.out.println(s1);
		} catch (ParseException e) {
			e.printStackTrace();
		}
	}
}
```

a) 2.60 1.5
b) $2.60 $1.50
c) $2.60 1.50
d) 2.6 $1.50

(15) Frage

Welche der untenstehenden Konstanten gibt alle zur Verfügung stehenden Locale-Konstanten wieder?

a) getDisplayCountry()
b) getCountry()
c) getLanguage()
d) getAvailableLocales()
e) getInstance()
f) getCurrencyInstance()

(16) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
public class FormBoolean {
	public static void main(String[ ] args) {
		Boolean bool = new Boolean("false");
		String st = String.format("%b", bool);
		System.out.println(st);
	}
}
```

a) true
b) false
c) Kompilierfehler

(17) Frage

Mit welcher der untenstehenden Methoden kann die Zeitzone verändert werden?

a) getDateInstance()
b) getInstance()
c) getDateTimeInstance()
d) setTimeZone()
e) getTime()
f) getTimeInstance()
g) Keine dieser Möglichkeiten.

(18) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class DatumFormat {
   public static void main(String[ ] args) {
      Date d = new Date();
      DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.US);
      System.out.println(df.format(d));
   }
}
```

a) December 6, 2005
b) 12/6/05
c) Tuesday, December 6, 2005
d) Dec 6, 2005
e) Keine dieser Möglichkeiten.

(19) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class DatumFormat {
   public static void main(String[ ] args) {
      Date d = new Date();
      DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.US);
      System.out.println(df.format(d));
   }
}
```

a) December 6, 2005
b) 12/6/05

c) Tuesday, December 6, 2005
d) Dec 6, 2005
e) Keine dieser Möglichkeiten.

(20) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.DateFormat;
import java.util.Date;
import java.util.TimeZone;

public class Dat {
	public static void main(String[ ] args) {
		Date d = new Date();
		DateFormat df = DateFormat.getDateInstance();
		System.out.println(df.format(d));
	}
}
```

a) 06.12.2005
b) 06.12.2005 9:15
c) 9:15:27
d) Keine dieser Möglichkeiten.

(21) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.text.DateFormat;
import java.util.Date;
import java.util.TimeZone;

public class Dat {
	public static void main(String[ ] args) {
		Date d = new Date();
		DateFormat df = DateFormat.getTimeInstance();
		System.out.println(df.format(d));
	}
}
```

a) 06.12.2005
b) 06.12.2005 9:15
c) 9:15:27
d) Keine dieser Möglichkeiten.

6. Lösungen

(1) Frage
a

(2) Frage
b

(3) Frage
c

(4) Frage
a

(5) Frage
b

(6) Frage
c

(7) Frage
a

(8) Frage
d

(9) Frage
b

(10) Frage
a

(11) Frage
d

(12) Frage
b

(13) Frage
b

(14) Frage
d

(15) Frage
d

(16) Frage
b

(17) Frage
d

(18) Frage
a

(19) Frage
d

(20) Frage
a

(21) Frage
c

H. Ein- und Ausgabe über Datenströme: java.io

1. Datenströme

Nehmen wir an, Sie wollen einen Brief verfassen und wieder abrufen, so geben Sie die Daten unter Verwendung der Tastatur in Ihr Textverarbeitungsprogramm ein und speichern diesen dann anschließend unter einem Namen. Das so erstellte Dokument läßt sich nun jederzeit wieder öffnen und bearbeiten. Analoge Vorgänge gibt es in Java, sie basieren auf Streams (Ströme), sie sind also „stromorientiert“. Streams transportieren Daten zeitlich hintereinander (sequenziell), sie fließen wie Wasser in einem Strom. Wollen Sie ein Dokument lesen oder schreiben, wird ein Strom zu einer Datenquelle geöffnet und wieder geschlossen. Die Klassen und Methoden, die Sie hierzu brauchen, stellt Ihnen das Package java.io zur Verfügung. Es lassen sich zwei verschiedene Arten von Datenströmen unterscheiden: Zeichen- und Byteströme. Sollten Sie sich bereits länger mit Java beschäftigen, werden Sie sich jetzt sicherlich wundern, aber all die folgenden Klassen und Methoden gibt es nicht erst seit dem Tiger-Release, aber sie waren in 1.4 nicht prüfungsrelevant, sind es aber seit der Zertifizierung zu 1.5.

a) Zeichenströme

Zeichenströme ermöglichen es Ihnen, Text in lesbarer Form in einem Dokument zu speichern. Zeichenströme versenden Folgen von 16-Bit-Daten, so ist es möglich die Daten characterweise (zeichenweise) zu übertragen (vgl. Kapitel Literale und char-Literale). Ein char-Literal (ein Zeichen) entspricht einem Unicode-Zeichen. Der Begriff Character-Streams wird in der Fachliteratur häufig als Synonym für Zeichenströme verwendet. Die entsprechenden Klassen des Packages java.io sind die abstrakten Klassen Reader und Writer.

b) Byteströme

Betrachten wir nun die Byteströme: Sie ermöglichen es Ihnen zwar auch Text in einer Datei zu speichern, der Text wird aber in Bytecode umgewandelt, so ist der Inhalt dieser Datei für Sie nicht mehr lesbar. Byteströme können nur 8-Bit Bytes-Datenpakete befördern. Die Daten werden byteweise durch den Strom weitergeleitet. Die abstrakten Klassen InputStream und OutputStream stellen die erforderlichen Methoden zur Verfügung.

2. Die Klasse File

Ein Objekt der Klasse File repräsentiert den Namen einer Datei und den dazugehörigen Pfad, aber nicht das tatsächliche Dokument oder dessen Daten. Wird ein neues Objekt der File-Klasse instanziiert, so wird kein Dokument erstellt, sondern nur der Name einer Datei. Ein File-Objekt kann auch existieren, obwohl es das dazugehörige Dokument nicht gibt. In der Klasse File gibt es keine Methoden, mit denen Sie in Dateien schreiben (write) oder lesen (read), können. Sie besitzt allerdings eine Methode createNewFile(), mit der Sie eine neue Datei erstellen

können. Der Konstruktor der File-Klasse lautet: public File(String pathname).

Hier eine tabellarische Aufstellung der wichtigsten Methoden der Klasse File:

Methode	Funktion
public boolean createNewFile()	Erstellt eine neue Datei mit dem Namen des File-Objektes.
public boolean delete()	Löscht die Datei, die zum File-Objekt gehöhrt.
public boolean exists()	Überprüft, ob eine Datei exisiert.
public String[ ] list()	Gibt ein String-Array (z. B. String[ ] s = f.list(), wobei f ein File-Objekt ist) aller Dateien und Verzeichnisse zurück, indem sich das File-Objekt befindet.
public boolean mkdir()	Legt ein neues Verzeichnis mit dem Namen des File-Objektes an.
public boolean renameTo(File dest)	Benennt die Datei um.
public boolean isDirectory()	Stellt fest, ob der Name des File-Objektes ein Verzeichnis ist.
public boolean canWrite()	Stellt fest, ob ein Dokument schreibgeschützt ist.
public boolean canRead()	Stellt fest, ob ein Dokument lesegeschützt ist.

3. Ein- und Ausgabe über Zeichenströme:

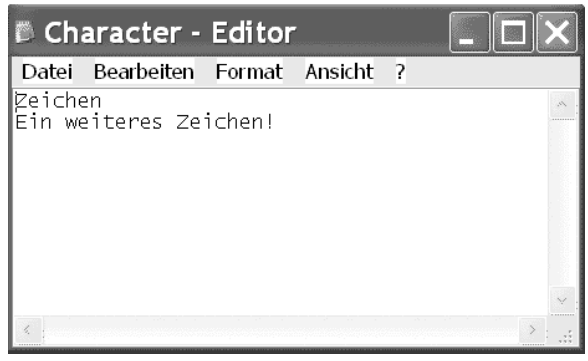
a) Die Klasse FileWriter und BufferedReader

Ein Objekt der Klasse FileWriter ist in der Lage Zeichen, char für char, in eine Datei zu schreiben. Dies ist keine optimale Lösung! Wie kann man diesen Prozess verbessern? Sie übergeben das FileWriter-Objekt als Parameter dem Konstruktor der Klasse BufferedWriter, dort werden die Zeichen zuerst an einen Puffer (Buffer) übergeben, wo sie zwischengelagert werden. Dieser Puffer dient als Zwischenspeicher, der seinen Inhalt erst an die entsprechende Datei übergibt, wenn der Puffer voll ist. Mit der Methode write(String s) werden die Zeichen in den Puffer geschrieben. Sollten Sie vorher Daten in die Datei schreiben wollen, geht dies unter Verwendung der Methode flush(). Am Ende ist es notwendig den Puffer mit der Methode close() zu schließen, da es sonst zu einem ungewollten Datenverlust kommen könnte. Da es zu unterschiedlichen Problemen kommen kann, wenn Sie Zeichen in den Puffer und in die Datei schreiben wollen, müssen diese Vorgänge in einen try/catch-Block geschrieben werden. Die entsprechende Exception, die bereits im Vorfeld aufgefangen werden muß, nennt sich IOException.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class SchreibenCharacter {
	public static void main(String[ ] args) {
		//Gibt Pfad zu Dokument an
		File f = new File("Character.txt");
		try {
			//Erstellt Datei Character.txt
			FileWriter fw = new FileWriter(f);
			//Daten werden dem Puffer übergeben
			BufferedWriter bw = new BufferedWriter(fw);
			//schreibt in den Puffer
			bw.write("Zeichen");
			//fügt Zeilenumbruch ein
			bw.newLine();
			//fügt zusätzliche Zeile eine
			bw.append("Ein weiteres Zeichen!");
			//schließt den Puffer und schreibt Daten in Datei
			bw.close();
		//fängt Fehler auf, der beim Schreiben in die Datei auftreten kann
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Bitte beachten Sie, daß die Klasse BufferedWriter keinen Konstruktor besitzt, dem man ein File-Objekt als Parameter übergeben kann. In folgender Abbildung sehen Sie das Ergebnis des obenstehenden Codes, die Daten wurden in die Datei Character.txt übertragen.



b) Die Klase FileReader

Objekte der Klasse FileReader und BufferedReader haben die Funktion, Daten aus einer Datei auszulesen. Die Methode readLine() ist dazu gedacht, den Inhalt der Datei zeilenweise auszugeben.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class LesenCharacter {
	public static void main(String[ ] args) {
		File f = new File(”Character.txt”);
		try {
			FileReader fw = new FileReader(f);
			BufferedReader bw = new BufferedReader(fw);
			String zeichen = null;
			while ((zeichen = bw.readLine()) != null){
				System.out.println(zeichen);
			}
			bw.close();
		//fängt Fehler auf, der beim Lesen der Datei auftreten kann
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Ausgabe auf der Konsole:

```
Zeichen
Ein weiteres Zeichen!
```

c) Die Klasse PrintWriter

Die Klasse PrintWriter stellt ergänzend zu der Klasse BufferedWriter die print()- und println()-Methode zur Verfügung, so bleibt es Ihnen erspart, separat einen Zeilenumbruch einfügen zu müssen. Außerdem ist es möglich, zusätzliche Datentypen, wie int, long, String und boolean einzulesen. Diese Daten werden dann als String in den Stream geschrieben.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class SchreibenPrintWriter {
	public static void main(String[ ] args) {
```

```
                File f = new File("PrintWriter.txt");
                try {
                        FileWriter fw = new FileWriter(f);
                        BufferedWriter bw = new BufferedWriter(fw);
                        /*FileWriter-Objekt wird als Parameter dem
                        PrintWriter-Konstruktor übergeben.*/
                        PrintWriter pw = new PrintWriter(bw);
                        pw.println("eins");
                        pw.println(24);
                        pw.close();
                //fängt Fehler auf, der beim Schreiben in die Datei auftreten kann
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Untenstehend lesen wir das soeben erstellte Dokument wieder aus.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class LesenPrintWriter {
        public static void main(String[ ] args) {
                File f = new File("PrintWriter.txt");
                try {
                        FileReader fw = new FileReader(f);
                        BufferedReader bw = new BufferedReader(fw);
                        String zeichen = null;
                        while ((zeichen = bw.readLine()) != null){
                                System.out.println(zeichen);
                        }
                        bw.close();
                //fängt Fehler auf, der beim Lesen der Datei auftreten kann
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Ausgabe auf der Konsole:

```
eins
24
```

4. Ein- und Ausgabe über Byteströme

a) Die Klasse ObjectOutputStream: Serialisierung von Objekten

Jetzt kommt die nächste Frage: Wie speichere ich Objekte? Oft ist es sinnvoll nicht nur Text oder Zahlen in Dateien zu speichern, sondern auch Objekte. Objekte können nur gespeichert (serialisiert) werden, wenn Sie das Interface Serializable implementieren.

Wie gehe ich Schritt für Schritt vor? Nehmen wir als Erstes unsere Klasse Auto, zu der wir eine implements-Beziehung zum Interface Serializable hinzufügen. Haben Sie dies getan, werden Sie feststellen, daß Sie keine Methode implementieren müssen, da das Serializable-Interface leer ist (vgl. Kapitel Interfaces und abstrakte Klassen).

```
import java.io.Serializable;

public class Auto implements Serializable {
	private String farbe;

	public String getFarbe() {
		return farbe;
	}

	public void setFarbe(String farbe) {
		this.farbe = farbe;
	}
}
```

Als Zweites wird ein FileOutputStream-Objekt instanziiert, das eine Datei erstellt, Dieses Objekt wird einem ObjectOutputStream-Objekt als Parameter übergeben, das dann die Objekte in Bytes umwandelt und in diesem Dokument speichert. Wollen Sie mehrere Objekte ein- und wieder ausgeben, geschieht dies unter Zuhilfenahme eines int-Wertes, den sie mit der Methode writeInt(int i) erst setzen müssen. Dies ist eine relativ komplizierte Methode, da Sie jedes Mal zählen müssen, wieviele Objekte es gibt. Die Klasse ObjectOutputStream besitzt aber leider keine Methode, die es Ihnen ermöglicht, problemlos mehrere Objekte aus Ihrem Dokument auszulesen. Sollten Sie mit diesem Trick nicht zufrieden sein, lernen Sie weiter unten eine andere Vorgehensweise kennen.

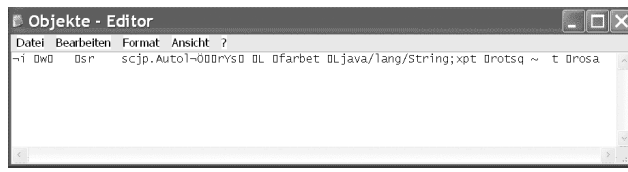
```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SchreibenObjekte {
	public static void main(String[ ] args) {
		//Gibt Pfad zu Dokument an
		File f = new File(”Objekte.txt“);
```

```
                //Erstellen von 2 Objekten
                Auto a = new Auto();
                a.setFarbe("rot");
                Auto b = new Auto();
                b.setFarbe("rosa");
                try {
                        //erstellt oder überschreibt Datei mit Namen Objekte.text
                        FileOutputStream fw = new FileOutputStream(f);
                        //wandelt Objekte und primitive Werte in Bytes um
                        ObjectOutputStream oos = new ObjectOutputStream(fw);
                        //schreibt einen primitiven int-Wert in Datei
                        oos.writeInt(2);
                        //schreibt Objekte in die Datei
                        oos.writeObject(a);
                        oos.writeObject(b);
                        oos.close();
                //fängt Fehler auf, der beim Schreiben in die Datei auftreten kann
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Versuchen Sie allerdings Objekte zu serialisieren, die das Interface Serializable nicht implementieren, wird eine NotSerializableException geworfen (vgl. Kapitel Exceptions).

Untenstehende Darstellung zeigt Ihnen, in welcher für den Menschen nicht lesbaren Form die Objekte gespeichert werden.



b) <u>Deserealisierung von Objekten</u>

Die Klassen FileInputStream und ObjectInputStream machen es Ihnen möglich serialisierte Objekte wieder auszulesen.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class LesenObjekte {
        public static void main(String[ ] args)  {
```

```
                File f = new File("Objekte.txt");
                try {
                        FileInputStream fi = new FileInputStream(f);
                        ObjectInputStream ois = new ObjectInputStream(fi);
                        //Auslesen der Anzahl der Objekte als int
                        int anzahl = ois.readInt();
                        //for-Schleife zum Auslesen mehrerer Objekte
                        for (int i = 0; i<anzahl; i++){
                                /*Cast ist notwendig, da die Methode
                                readObject() ein Object zurückgibt*/
                                Auto auto = (Auto) ois.readObject();
                                System.out.println(auto.getFarbe());
                        }
                        fi.close();
                //fängt Fehler auf, der beim Schreiben in die Datei auftreten kann
                } catch (IOException e) {
                        e.printStackTrace();
                /*fängt Fehler auf, der beim Casten (hier in ein Objekt Auto)
                auftreten kann*/
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
}
```

Ausgabe auf der Konsole:

```
rot
rosa
```

c) Serialisieren von Collections

Im vorangegangenen Abschnitt haben Sie gesehen, daß die Klasse ObjectOutputStream Ihnen leider keine komfortable Lösung zum Ein- und Auslesen von mehreren Objekten zur Verfügung stellt. Dieses Problem können Sie einfach lösen, indem Sie Ihre Objekte, bevor sie serialisiert werden, einer Collection, einer Map oder einem Array übergeben. Eine Collection, eine Map oder ein Array können aber nur serialisiert werden, wenn ihnen Objekte übergeben werden, die das Serializable-Interface implementieren. Dies möchte ich Ihnen nun anhand einer LinkedList<Auto> demonstrieren:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.LinkedList;

public class SchreibenObjekteList {
        public static void main(String[ ] args) {
                File f = new File("ObjekteList.txt");
                Auto a = new Auto();
```

```
                a.setFarbe("rot");
                Auto b = new Auto();
                b.setFarbe("rosa");
                LinkedList<Auto> s = new LinkedList<Auto>();
                s.add(a);
                s.add(b);
                try {
                        FileOutputStream fw = new FileOutputStream(f);
                        ObjectOutputStream oos = new ObjectOutputStream(fw);
                        oos.writeObject(s);
                        oos.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Hier die entsprechende Klasse, die die Daten aus der Datei wieder ausliest:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.LinkedList;

public class LesenObjekteList {
        public static void main(String[ ] args){
                File f = new File("ObjekteList.txt");
                try {
                        FileInputStream fi = new FileInputStream(f);
                        ObjectInputStream ois = new ObjectInputStream(fi);

                        for (Auto a: (LinkedList<Auto>)ois.readObject()){
                                System.out.println(a.getFarbe());
                        }
                        fi.close();
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
}
```

d) <u>Transiente und statische Variablen</u>

Welche Bestandteile eines Objektes werden bei der Serialisierung nicht mitgespeichert? Erstens: Statische Variablen, da sie zur Klasse gehören und nicht zum Objekt (vgl. Kapitel Klassen- und Instanzvariablen). Zweitens: Variablen, die um den Modifier transient ergänzt wurden. Betrachten wir ein Beispiel: Unsere

Variable farbe wird in transient geändert.

```
import java.io.Serializable;

public class Auto implements Serializable {

	transient private String farbe;

	public String getFarbe() {
		return farbe;
	}

	public void setFarbe(String farbe) {
		this.farbe = farbe;
	}
}
```

Was passiert, wenn jetzt Objekte der Klasse Auto serialisiert und deserialisiert werden? Es wird nur der entsprechende Standardwert eines Objektes ausgegeben: null.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.LinkedList;

public class LesenObjekteList {
	public static void main(String[ ] args){
		File f = new File(”ObjekteList.txt”);
		try {
			FileInputStream fi = new FileInputStream(f);
			ObjectInputStream ois = new ObjectInputStream(fi);

			for (Auto a: (LinkedList<Auto>)ois.readObject()){
				System.out.println(a.getFarbe());
			}
			fi.close();
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
}
```

Ausgabe auf der Konsole:

```
null
null
```

5. Übungen

(1) Frage

Durch welche Methode muß das Wort „Hier“ ersetzt werden?

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SchreibObjekt {
	public static void main(String[ ] args) {
		File f = new File(”Objekt.txt”);
		Auto a = new Auto();
		a.setFarbe(”blau”);
		try {
			FileOutputStream fw = new FileOutputStream(f);
			ObjectOutputStream oos = new ObjectOutputStream(fw);
			oos.Hier(a);
			oos.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

a) print
b) println
c) write
d) read
e) readObject
f) writeObject
g) Keine dieser Möglichkeiten.

(2) Frage

Durch welche Klasse muß das Wort „Hier“ ersetzt werden?

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.Hier;

public class LiesObjekt {
	public static void main(String[ ] args)  {
		File f = new File(”Objekt.txt”);
		try {
			FileInputStream fis = new FileInputStream(f);
			Hier ois = new Hier(fis);
			Auto auto = (Auto) ois.readObject();
			System.out.println(auto.getFarbe());
			fis.close();
		} catch (IOException e) {
```

```
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
}
```

a) FileReader
b) PrintWriter
c) BufferedReader
d) ObjectOutputStream
e) Serializable
f) Keine dieser Möglichkeiten.

(3) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.Serializable;

public class Auto implements Serializable {
        private String farbe;
        public String getFarbe() {
                return farbe;
        }
        public void setFarbe(String farbe) {
                this.farbe = farbe;
        }
}

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SchreibObjekt {
        public static void main(String[ ] args) {
                File f = new File("Objekt.txt");
                Auto a = new Auto();
                a.setFarbe("blau");
                try {
                        FileOutputStream fw = new FileOutputStream(f);
                        ObjectOutputStream oos = new ObjectOutputStream(fw);
                        oos.writeObject(a);
                        oos.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}

import java.io.File;
import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.io.ObjectInputStream;

public class LiesObjekt {
	public static void main(String[ ] args)  {
		File f = new File("Objekt.txt");
		try {
			FileInputStream fis = new FileInputStream(f);
			ObjectInputStream ois = new ObjectInputStream(fis);
			Auto auto = (Auto) ois.readObject();
			System.out.println(auto.getFarbe());
			fis.close();
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
}
```

a) blau
b) null
c) Es wird eine Exception geworfen.
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(4) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.Serializable;

public class Auto {
	private String farbe;
	public String getFarbe() {
		return farbe;
	}
	public void setFarbe(String farbe) {
		this.farbe = farbe;
	}
}

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SchreibObjekt {
	public static void main(String[ ] args) {
		File f = new File("Objekt.txt");
		Auto a = new Auto();
		a.setFarbe("blau");
		try {
```

```
                        FileOutputStream fw = new FileOutputStream(f);
                        ObjectOutputStream oos = new ObjectOutputStream(fw);
                        oos.writeObject(a);
                        oos.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class LiesObjekt {
        public static void main(String[ ] args)  {
                File f = new File("Objekt.txt");
                try {
                        FileInputStream fis = new FileInputStream(f);
                        ObjectInputStream ois = new ObjectInputStream(fis);
                        Auto auto = (Auto) ois.readObject();
                        System.out.println(auto.getFarbe());
                        fis.close();
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
}
```

a) blau
b) null
c) Es wird eine Exception geworfen.
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(5) Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
import java.io.Serializable;

public class Auto implements Serializable {
        transient private String farbe;
        public String getFarbe() {
                return farbe;
        }
        public void setFarbe(String farbe) {
                this.farbe = farbe;
        }
}
```

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SchreibObjekt {
	public static void main(String[ ] args) {
		File f = new File(”Objekt.txt”);
		Auto a = new Auto();
		a.setFarbe(”blau”);
		try {
			FileOutputStream fw = new FileOutputStream(f);
			ObjectOutputStream oos = new ObjectOutputStream(fw);
			oos.writeObject(a);
			oos.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class LiesObjekt {
	public static void main(String[ ] args)  {
		File f = new File(”Objekt.txt”);
		try {
			FileInputStream fis = new FileInputStream(f);
			ObjectInputStream ois = new ObjectInputStream(fis);
			Auto auto = (Auto) ois.readObject();
			System.out.println(auto.getFarbe());
			fis.close();
		} catch (IOException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
}
```

a) blau
b) null
c) Es wird eine Exception geworfen.
d) Kompilierfehler
e) Keine dieser Möglichkeiten.

(6) Frage

Welche Methoden gibt es in der Klasse PrintWriter?

a) readObject()
b) writeObject()
c) add()
d) delete()
e) mkdir()
f) Keine dieser Möglichkeiten.

(7) Frage

Welche Methoden gibt es in der Klasse BufferedWriter?

a) readObject()
b) writeObject()
c) add()
d) delete()
e) mkdir()
f) Keine dieser Möglichkeiten.

(8) Frage

Welche Methoden gibt es in der Klasse ObjectInputStream?

a) readObject()
b) writeObject()
c) add()
d) delete()
e) mkdir()
f) Keine dieser Möglichkeiten.

(9) Frage

Welche Methoden gibt es in der Klasse ObjectOutputStream?

a) readObject()
b) writeObject()
c) add()
d) delete()
e) mkdir()
f) Keine dieser Möglichkeiten.

(10) Frage

Welche Methoden gibt es in der Klasse File?

a) readObject()
b) writeObject()
c) add()
d) delete()
e) mkdir()
f) Keine dieser Möglichkeiten.

(11) Frage

Welche Aussagen treffen zu?

a) Ein Objekt der Klasse File repräsentiert den Namen einer Datei und den dazugehörigen Pfad.
b) Die Klasse File besitzt eine Methode readObject().
c) Die Klasse File besitzt eine Methode canWrite() und delete().
d) Die Begriffe Zeichenströme und Byteströme können synonym verwendet werden.
e) Die Klasse PrintWriter soll nicht verwendet werden.
f) Keine dieser Möglichkeiten.

(12) Frage

Welche Aussagen treffen zu?

a) Die abstrakten Klassen InputStream und OutputStream stellen die notwendigen Methoden für Zeichenströme zur Verfügung.
b) Die abstrakten Klassen Reader und Writer stellen die erforderlichen Methoden für Byteströme bereit.
c) Die Begriffe Character-Streams und Byteströme sind Synonyme.
d) Die Klasse File besitzt keine Methode createNewFile().
e) Will man ein Objekt speichern (serialisieren), muß es das Runnable-Interface implementieren.
f) Keine dieser Möglichkeiten.

(13) Frage

Welche Aussagen treffen zu?

a) Das Serializable-Interface besitzt keine Methoden.
b) Wollen Sie zu Objekten der Klasse BufferedWriter mit der Methode write() Zeichen hinzufügen, muß diese nicht in einem try/catch-Block stehen.
c) In der Klasse File gibt es Methoden, die das Lesen und Schreiben von Dateien möglich machen.
d) In der Klasse File gibt es eine Methode canRead().
e) Alle wichtigen Klassen, Datenströme betreffend, befinden sich im Package java.lang.
f) Keine dieser Möglichkeiten.

(14) Frage

Durch welche Methode muß das Wort „Hier“ ersetzt werden?

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class SchreibCharacter {
        public static void main(String[ ] args) {
```

```
                File f = new File("Text.txt");
                try {
                        FileWriter fw = new FileWriter(f);
                        BufferedWriter bw = new BufferedWriter(fw);
                        bw.Hier("Hallo!");
                        bw.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

a) print
b) println
c) write
d) writeObject

(15) Frage

Durch welche Klasse muß das Wort „Hier“ ersetzt werden?

```
import java.io.Hier;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class LiesCharacter {
        public static void main(String[ ] args) {
                File f = new File("Text.txt");
                try {
                        FileReader fw = new FileReader(f);
                        Hier bw = new Hier(fw);
                        String zeichen = null;
                        while ((zeichen = bw.readLine()) != null){
                                System.out.println(zeichen);
                        }
                        bw.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

a) ObjectOutputStream
b) PrintWriter
c) BufferedReader
d) BufferedWriter
e) OutputStream

(16) Frage

Durch welche Methode muß das Wort Hier ersetzt werden?

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class SchreibPrintWriter {
	public static void main(String[ ] args) {
		File f = new File("Neu.txt");
		try {

			FileWriter fw = new FileWriter(f);
			BufferedWriter bw = new BufferedWriter(fw);
			PrintWriter pw = new PrintWriter(bw);
			pw.Hier("Hallo");
			pw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

a) print
b) println
c) write
d) writeObject

6. Lösungen

(1) Frage

f

(2) Frage

f

(3) Frage

a

(4) Frage

c

(5) Frage

b

(6) Frage
f

(7) Frage
f

(8) Frage
a

(9) Frage
b

(10) Frage
d, e

(11) Frage
a, c

(12) Frage
f

(13) Frage
a, d

(14) Frage
c

(15) Frage
c

(16) Frage
a, b

Index

A

abs-Methode 230
abstrakte Klassen 52, 54
Anonyme Klassen 441
Api Documentation 45
append()-Methode 261
ArithmeticException 297
Arithmetischer Operator 103
ArrayIndexOutOfBoundsException 297
ArrayList 381
Arrays 195
asList()-Methode 493
assert-Statement 320–321
Assertions 319
auf- und abrunden 230
ausführbaren Zustand 344
Autoboxing 460

B

Bezeichner 16
Binäres Zahlensystem 134
binarySearch()-Methode 494
Boolean-Literale 26
Boolean Logischen Operatoren 115
BufferedReader 562
Byteströme 561, 566

C

Calendar 545
Casten von Objekten 90
ceil()-Methode 230
Char-Literale 24
charAt()-Methode 250
Checked Exceptions 295
ClassCastException 296
ClassNotFoundException 295
Collections-Framework 375
Comparable- Interface 383
Comparator-Interface 383
compile()-Methode 530
concat()-Methode 247
cos()-Methode 232

D

Daemon-Threads 340
Date 545
DateFormat 545
Datenströme 561
Debugging 319
default 42
Dekrement 127
delete()-Methode 263
Deserealisierung von Objekten 567
do-while-Schleife 181

E

Eclipse 5
eindimensionales Array 195
else-if-Struktur 164
Enums 510
equals()-Methode 277, 403
erweiterte for-Schleife 455
Exceptions 293

F

File 561
FileNotFoundException 295
FileReader 564
FileWriter 562
final 59
finalize()-Methode 365
finally-Block 299
findInLine()-Methode 529
Fließkommazahlen 26
floor()-Methode 230
for-Schleife 182
format()-Methode 540, 549
Formatieren 539
Formatter 543
Formatter-Klasse 543
Fundamental Classes 209

G

Garbage Collection 359
gc()-Methode 366
Geltungsbereich 44
Generics 484
Generische Interfaces 491
Generische Klassen 491
Generische Methoden 491
Getter- und Setter-Methoden 41
Gültigkeitsbereich 182

H

has-a-Relationship 50
hashCode()-Methode 403
HashMap 383
HashSet 376
Hashtable 381
Hexadezimales Zahlensystem 136
Hüllklassen 209

I

if-else-Struktur 164
if-Struktur 163
Immutable Objects 463
indexOf()-Methode 248
Initialisierungen von Variablen 24
Inkrement 127
Innere Klassen 409
insert()-Methode 262

Installation (Eclipse) 8
Installation (JDK) 5
instanceof 92
Instanz 43
Instanziierung 43
Instanziierung von inneren Klassen 409
Instanzvariable 44
Integer-Literale 25
Integer Bitweise Operatoren 134
Interface 52
Interface Collection 376
Interface List 379
Interface Map 381
Interface Queue 495
Interface Set 376
IOException 295
is-a-Relationship 49–50
Isolierte Inseln 363
Iterationen 180

J

java.io 561
Java Development Kit 5
Javadocs 45
Java Runtime Environment 5
Java Virtual Machine 5
JDK 5
join()-Methode 345, 348
JRE 5
JVM 5

K

Kapazität eines StringBuffers 260
Kapselung 41
Klasse Error 293
Klasse Exception 293
Klasse Formatter 543
Klasse Locale 539
Klasse Matcher 530
Klasse Math 228
Klassen 39
Klassen Pattern 530
Klassenvariable 44
Klasse Object 209
Klasse Scanner 526
Klasse String 246
Klasse StringBuffer 260
Klasse StringBuilder 526
Klasse Throwable 293
Kompiler 5
Kompilierfehler 11, 293
Konsole 10
Konstante 43
Konstruktor 42
Konstruktoraufruf mit super() 77
Konstruktoraufruf mit this() 82
Kontrollstrukturen 163

L

Länge eines StringBuffers 260
lastIndexOf()-Methode 248
laufender Zustand 344
LinkedHashMap 383
LinkedHashSet 376
LinkedList 379
List 379
Literale 24–27
Locale 539
Lock 342, 344
lokale Klassen 430
lokale Variable 182
lokale Variablen 45

M

main()-Methode 11
Main-Thread 340
Map 381
Matcher 530
matcher()-Methode 530
matches()-Methode 530
max()-Methode 228
Maximum 228
mehrdimensionales Array 198
Methode 41
Methoden und Konstruktoren mit variabler Argumentanzahl 478
min()-Methode 228
Minimum 228
Modifier 59
Modifier für innere Klassen 413
Modifier für lokale Klassen 430

N

native 59
notify()-Methode 346
notifyAll()-Methode 346
NumberFormat 548
NumberFormatException 210

O

Object Finalization 365
ObjectOutputStream 566
Objekte 39
Objektreferenz 43
Objektreferenz mit super 75
Objektreferenz mit this 76
Oktales Zahlensystem 135
Operator 103
Operator Modulo 104

P

Package 45
package local 42
Parameter 41
parse()-Methode 548
parseXXX(String s) 215

Pattern 530
Primitiver Datentyp 23
printf()-Methoden 544
PrintWriter 564
private 40
protected 42
public 40

Q

Queue 495

R

random()-Methode 231
reguläre Ausdrücke 527
relationaler Operator 134
replace()-Methode 247
replaceAll()-Methode 531
reverse()-Methode 263
round()-Methode 230
Rückgabetype 41
run()-Methode 338
Runnable Interface 338
RuntimeExceptions 296

S

Scanner 526
Schlüsselwörter 20
Schnittstelle 52
scope 45
Selektionen 163
Serialisierung von Objekten 566
Set 376
Shift-Operator 151
Short-Circuit-Auswertung 123
sin()-Methode 232
sleep()-Methode 345
sort()-Methode 494
split()-Methode 533
sqrt()-Methode 232
Standardklassen 209
start()-Methode 344
statischer Import 517
statische Variable 44
strictfp 59
String-Literal 246
String-Literale 26
StringBuilder 526
Stringkonkatenation 270
Subklasse 49
substring()-Methode 249
Subtyping 88
Superklasse 49
switch-Struktur 174
Synchronisation 341
Synchronisation von Blöcken 342
synchronized 59
System.gc()-Methode 366

T

tan()-Methode 232
Thread-Stadien 344
Thread.MAX_PRIORITY 337
Thread.NORM_PRIORITY 337
Thread_MIN_PRIORITY 337
Threads 337
throw-Ausdruck 301
throws-Klausel 303
toArray()-Methode 493
transient 59
Transiente Variablen 569
TreeMap 383
TreeSet 378
trim()-Methode 249
try- und catch-Blöcke 298
Typanpassung, explizite und implizite 34
Typumwandlung 34

U

Übergabewert 41
überladen 64
überschreiben 68
Unveränderbare Objekte 463

V

valueOf(String s) 213
Varargs 478
Variable 40
Variablendeklaration 24
Vector 381
Vererbung 49
Vergleichsoperator 277
Vergleich von Objekten und primitiven Datentypen 277
Verschiebeoperator 151

W

Währungsformat 549
wait()-Methode 346
wartender Zustand 344
while-Schleife 180
Wildcards 487
Wrapper-Klassen 209

X

xxxValue() 214

Y

yield()-Methode 345

Z

Zahlenformat 548

Zahlensysteme 134
Zeichenströme 561
Zugriffsbezeichner 42
zusammengesetzter Zuweisungsoperator 107
Zuweisung 34