

Antworten zu den Repetitionsfragen

-
- 1 Seite 28 Ein Objekt ist ein Bündel von zusammengehörigen Verhaltensweisen, die intern durch gemeinsam genutzte Daten unterstützt werden.
-
- 2 Seite 28 Eine Klasse beschreibt, wie sich gleichartige Objekte verhalten und welche Eigenschaften sie haben.
-
- 3 Seite 35 Die main()-Methode wird von der Java-Laufzeitumgebung gestartet, wenn die Klasse «laufen gelassen» wird.
-
- 4 Seite 35 Beispielsweise mittels Rechtsklick auf den SRC-Ordner und der anschließenden Auswahl von **Neu -> Package** im Kontextmenü. Im darauf folgenden Dialogfenster kann der Paketname eingegeben werden.
-
- 5 Seite 35 Die Klasse definiert die typischen Eigenschaften und Verhaltensweisen von Regisseuren. Ein Stelleninserat beschreibt, was der gesuchte Regisseur können und machen muss.
-
- 6 Seite 35

```
public class Klasse {...}
```
-
- 7 Seite 35 Mit dem Operator `new`. Beispiel: `Regisseur david = new Regisseur();`
-
- 8 Seite 35 Das Objekt `david` soll die Methode `go()` ausführen.
-
- 9 Seite 35 Obwohl Java grundsätzlich Umlaute erlaubt, sollte man davon absehen. Begründung: Anhand des Klassennamens wird eine Datei angelegt und nicht jedes Dateisystem kommt mit Umlauten im Dateinamen zurecht. Es kann auch vorkommen, dass die Zeichencodierung im Dateisystem anders ist als in Java und Umlaute zwar gleich aussehen, aber nicht demselben Zeichen entsprechen.
-
- 10 Seite 35 Die Klasse **Zeuge** erhält eine private Instanzvariable und eine Methode mit einem String als Übergabeparameter. So kann jedem Zeugen ein eigener Aussagetext gegeben werden.
-
- 11 Seite 35 Dem Polizistenobjekt `hercule` wird zur Zeugenbefragung jeweils eine Referenz auf ein Zeugenobjekt übergeben.
-
- 12 Seite 35 Objekte müssen untereinander Beziehungen haben, damit sie miteinander kommunizieren können.
-
- 13 Seite 35 Beziehungstypen:
- Aggregation
 - Assoziation
 - Komposition

-
- 14 Seite 45 Obwohl man im Prinzip direkt auf die Instanzvariablen eines Objekts zugreifen kann, sollte man das nicht tun, sondern das Objekt immer nach der gewünschten Information fragen. Begründung: Ein «Aussenstehender» muss nicht wissen, wie die betreffende Information im Objekt abgespeichert ist. Programmierer sind somit frei, die Art der Abspeicherung innerhalb des Objekts zu ändern (etwa um eine Performanceverbesserung zu erreichen), ohne dass «Benutzer» der Objekte davon betroffen sind.
-
- 15 Seite 45 Methoden:
- setter()-Methode
 - getter()-Methode
-
- 16 Seite 45 Mit ihrer Hilfe werden Informationen über Fehler oder Ausnahmen im Programmablauf kommuniziert.
-
- 17 Seite 45 Polymorphismus heisst wörtlich «Vielgestaltigkeit» und bedeutet in der OOP, dass ein und diesselbe Nachricht je nach Objekt unterschiedliche Methoden aktivieren kann. Beispiel: Obwohl ein Huhn ein Vogel-Objekt ist, werden «Huhn-spezifische» Methoden aufgerufen.
-
- 18 Seite 45 Ein Interface enthält keinen Code. Es sagt nur aus, dass eine Verhaltensweise bzw. eine Methode vorhanden ist, gibt aber nicht an, wie diese Verhaltensweise realisiert wird.
-
- 19 Seite 49 Das Klassendiagramm zeigt auf, zwischen welchen Klassen Beziehungen bestehen und welche Objekte «sich kennen».
-
- 20 Seite 49 Zwischen Objekten der **Klassen 1** und **2** gibt es eine Beziehung. Ein Objekt der **Klasse 1** kann zu keinem oder mehreren Objekten der **Klasse 2** eine Beziehung haben. Umgekehrt hat ein Objekt der **Klasse 2** immer genau eine Beziehung zu einem Objekt der **Klasse 1**.
-
- 21 Seite 49 Das Sequenzdiagramm zeigt auf, in welcher zeitlichen Abfolge (Sequenz) welches Objekt mit welchem anderen Objekt zusammenspielt (interagiert).
-
- 22 Seite 49 Durch einen Pfeil (mit gerader Linie mit leerem Dreieck), der von der Unter- zur Oberklasse zeigt.
-
- 23 Seite 62 Ganzzahlige primitive Datentypen: `byte`, `short`, `int`, `long`
-
- 24 Seite 62 Mögliche Datentypen: `float` oder `double`
-
- 25 Seite 62 In C sind logische Werte numerisch, d. h., man kann mit ihnen rechnen. 0 wird als `false` interpretiert, alles andere als `true`. In Java sind logische Werte echte logische Werte und keine «versteckten» Zahlen, d. h., man kann mit ihnen nicht rechnen.
-
- 26 Seite 62 Während `'a'` ein einzelnes Unicode-Zeichen darstellt, ist `"a"` ein String mit der Länge 1. Strings und Zeichen sind nicht dasselbe.
-
- 27 Seite 62 Ja. Das Pluszeichen zwischen den Strings wird als «aneinanderhängen» interpretiert.

-
- 28 Seite 62 Nein. Begründung: Es wird zwar eine Variable `zahlenreihe` erstellt, die eine Referenz auf ein Array mit `int`-Werten halten kann, doch das Array selbst existiert nicht. Die Zuweisung in der zweiten Zeile wird deshalb fehlschlagen. Korrekt wäre etwa: `int[] zahlenreihe = new int[10];`
-
- 29 Seite 62 Solche Klassen entsprechen einem primitiven Datentyp, d. h., es gibt zu jedem primitiven Datentyp eine Wrapper-Klasse.
-
- 30 Seite 62 Indem jede Objektvariable effektiv nur eine Referenz auf ein Objekt hält, «zeigt» sie auf das betreffende Objekt.
-
- 31 Seite 62 Der Java-Compiler übersetzt den Quellcode in sogenannten Byte-Code. Dieser wird danach durch die Java Virtual Machine ausgeführt.
-
- 32 Seite 62 Die virtuelle Java-Maschine ist eine Software, die wie ein Prozessor funktioniert und auf jeder Plattform eingesetzt werden kann. Dadurch ist Java Byte-Code ohne Änderungen oder erneute Kompilierung überall direkt lauffähig.
-
- 33 Seite 62

```
public static void main(String[] args) {...}
```
-
- 34 Seite 62 Ein Array in Java ist ein echtes Objekt und kennt seine Grösse.
-
- 35 Seite 62 Die Deklaration der Variablen muss innerhalb der Klasse stehen. Ausserhalb der Klassendeklaration ist kein Code erlaubt.
-
- 36 Seite 69 `&&`
-
- 37 Seite 69 Wenn `a` grösser als null ist, wird der Wert von `a` übernommen, ansonsten wird der negative Wert von `a` genommen. Mathematisch gesprochen wird hier der absolute Betrag von `a` bestimmt.
-
- 38 Seite 69 Das einfache Gleichheitszeichen ist eine Zuweisung, während das doppelte Gleichheitszeichen den Vergleichsoperator darstellt.
-
- 39 Seite 69 Eine kopfgesteuerte Schleife wird möglicherweise nie durchlaufen, da die Bedingung vor dem ersten Durchlauf geprüft wird. Eine fussgesteuerte Schleife wird mindestens einmal durchlaufen, da die Bedingung erst nach einem Durchlauf geprüft wird.
-
- 40 Seite 72 Erläuterungen:
- `pass by value`: Hier wird der effektive Wert an die Funktion übergeben. In der Funktion steht der Wert in einer eigenen unabhängigen Variablen.
 - `pass by reference`: Hier wird nur eine Referenz auf die Funktion übergeben, nicht der Wert bzw. das Objekt selbst. Die Funktion kann über diese Referenz auf das Objekt zugreifen und dieses auch verändern.

41 Seite 72 Unter Signatur einer Methode wird der Name der Methode plus Reihenfolge und Datentyp der Parameter verstanden.

42 Seite 72 Die zweite Methode wird aufgerufen. Grund: Die Zahl 3 kann zwar zu `double` konvertiert werden, passt aber besser zu `integer`, weil sie eine ganze Zahl ohne Kommastellen ist.

43 Seite 72 Die erste Methode wird aufgerufen. Grund: Die Zahl 3 ist ein primitiver `int`-Wert. Für einen Aufruf der zweiten Methode müsste die Zahl 3 zuerst in ein Integer-Objekt umgewandelt werden.

44 Seite 96

Klassenname	gültig	schlecht gewählt	ungültig
Grafik	X		
Mühle		Mit Umlaut	
Diesisteinlangername		Mangelhafte Lesbarkeit	
Waerme Pumpe			Mit Leerschlag
radioAufnahme		Beginnt mit Kleinbuchstaben	

45 Seite 96 Es handelt sich um eine Variable, die jedes Objekt einer Klasse hat. Da alle Objekte über einen eigenen Speicherplatz für Variablen verfügen, kann jedes Objekt einen eigenen Wert haben (unabhängig von den Werten bei den anderen Objekten).

46 Seite 96 Dieses Schlüsselwort stellt eine Referenz auf das aktuelle Objekt dar.

47 Seite 96 Es handelt sich um eine Klassenvariable. Den Speicherplatz für diese Variable gibt es genau einmal. Auf eine solche Variable kann zugegriffen werden, ohne dass man ein Objekt der Klasse benötigt.

48 Seite 96 Der Defaultkonstruktor ist ein Konstruktor ohne Parameter. Java erstellt ihn automatisch, wenn kein anderer Konstruktor vorhanden ist. Sobald der Programmierer einen eigenen Konstruktor schreibt, gibt es keinen Defaultkonstruktor mehr.

49 Seite 96 Mit `super()` wird der Konstruktor der Überklasse aufgerufen.

50 Seite 96 Dafür wird der Pfeil mit gerader Linie und leerem Dreieck verwendet. Beispiel: Ein Huhn ist ein Vogel wird so dargestellt: Huhn \longrightarrow Vogel

51 Seite 96 `public class Huhn extends Vogel {...}`

52 Seite 96 Zugriffsrechte:

Schlüsselwort	Klasse	Paket	Unterklasse	Alle andern
<code>public</code>	Zugriff	Zugriff	Zugriff	Zugriff
<code>protected</code>	Zugriff	Zugriff	Zugriff	Kein Zugriff
(default/keine Angabe)	Zugriff	Zugriff	Kein Zugriff	Kein Zugriff
<code>private</code>	Zugriff	Kein Zugriff	Kein Zugriff	Kein Zugriff

53 Seite 96 Mit dem Schlüsselwort `super`. Mit `super.fliegen()` greift die Methode `fliegen()` in der Klasse **Huhn** auf die Methode `fliegen()` in der Klasse **Vogel** zurück.

54 Seite 97

```
Huhn jakob = new Huhn();           // ein Huhn ist ein Huhn
Vogel einVogel = jakob;           // korrekt, Jakob ist ein Vogel
Vogel nochEinVogel = (Vogel)jakob; // ausdrückliche Typumwandlung
                                   // korrekt, weil Jakob ein Vogel ist

Ente ede = new Vogel();           // falsch, ein Vogel ist keine Ente

Vogel nochEinVogel = new Vogel(); // korrekt
Ente ede = (Ente)nochEinVogel;    // falsch, ein Vogel ist keine Ente
                                   // das kann auch nicht erzwungen werden

einVogel.fliegen();               // korrekt, jeder Vogel kann fliegen
jakob.fliegen();                  // korrekt, Jakob ist ein Huhn und
Hühner                             // können fliegen

Vogel ede = new Ente();           // ede ist eine Ente
ede.schwimmen();                  // falsch, ede wird als Vogel
angesprochen,                     // Vögel können nicht schwimmen

((Ente).ede).schwimmen();         // korrekt, ede wird als Ente
angesprochen,                     // Enten können schwimmen
```

55 Seite 97 Mit dem Operator `instanceof`.

56 Seite 97 Ja, jede Klasse erbt letzten Endes von der Klasse **Object**. Diese Klasse ist die einzige Klasse, die keine Oberklasse hat.

57 Seite 97 Genau gleich wie eine Klasse, jedoch mit dem Schlüsselwort `interface` anstatt `class`.

58 Seite 97 `public class MeineKlasse implements MeinInterface {...`

59 Seite 97 Es können keine Objekte der Klasse **Vogel** erzeugt werden, nur Objekte von Unterklassen, die von **Vogel** erben.

60 Seite 97 Mit dem Schlüsselwort `abstract` ohne Funktionskörper und Code.
Beispiel: `public abstract int rechne(int zahl1, double zahl2);`

61 Seite 97 Mit der Anweisung `throw`.

62 Seite 97 Mit einem `catch`-Block.

63 Seite 97 Jede Klasse hat eine Konstruktionsmethode, die Konstruktor genannt wird und die mit dem Schlüsselwort `new` aufgerufen werden kann.

-
- 64 Seite 97 Wichtige Exceptions-Methoden:
- `getMessage()`
 - `printStackTrace()`
-
- 65 Seite 97 In dieser Methode kann die Exception `xyz` geworfen werden. Sie wird nicht innerhalb der Methode abgefangen und muss deshalb weitergeleitet werden.
-
- 66 Seite 97 Diese Klasse enthält statische Methoden, um Benutzereingaben via Tastatur zu lesen.
-
- 67 Seite 102 Mit diesem Schlüsselwort kann eine Klasse in einem Paket «versorgt» werden. Der Eintrag `package paket.name;` muss als erste Zeile in einer Klassendeklaration stehen. Diese Klasse wird im Paket **paket.name** untergebracht.
-
- 68 Seite 102 Die Klasse, die in diesem Quellcode definiert wird, befindet sich im Paket `ch.meier.koole-app`. Pakete dienen der Ordnung.
-
- 69 Seite 102 Zugriffsoptionen:
- `import anderes.paket.name;`
 - `anderes.paket.name.Klasse`
-
- 70 Seite 102 **java.lang.***
-
- 71 Seite 102 Die Klasse **Random** aus dem Paket **java.math** wird bekannt gemacht. Nun kann auf diese Klasse einfach zugegriffen werden.
-
- 72 Seite 107 Vorgehen:
1. Java-Dokumentation öffnen (entweder online oder lokal).
 2. Im linken Bereich des Dialogfensters nach der Klasse **Runtime** suchen.
 3. Im linken Bereich des Dialogfensters den Namen der Klasse anklicken.
- Danach wird im rechten Bereich die zugehörige Dokumentation angezeigt.
-
- 73 Seite 107 Vorgehen:
1. In der Java-Dokumentation die Dokumentation für die Klasse **JFrame** suchen.
 2. Zuoberst in der Dokumentation die Paketzugehörigkeit ablesen.
-
- 74 Seite 107 Diese Funktionalität sollte nicht verwendet werden, da sie in einer späteren Java-Version nicht mehr zur Verfügung steht. In der Java-Dokumentation wird meist auf eine Alternative verwiesen, die stattdessen zu verwenden ist.
-
- 75 Seite 107 Im oberen Navigationsbereich des Dialogfensters gibt es einen separaten Menüpunkt für Hilfestellungen zur Dokumentation.
-
- 76 Seite 114 Ein Objekt der Klasse **File** beschreibt eine Datei oder ein Verzeichnis.

-
- 77 Seite 114 Ein `Reader` dient zum Einlesen von (Unicode-)Zeichen. Ein `BufferedReader` fasst diese Zeichen zeilenweise zusammen.
-
- 78 Seite 114 `System.out` ist der standardmässige Ausgabekanal. Dieser wird automatisch erstellt und ist normalerweise mit der Konsole (Bildschirm, Kommandozeilenfenster) verbunden.
-
- 79 Seite 114 Objekte der Klasse **Writer** schreiben verschiedene Datenarten in einen Ausgabestrom.
-
- 80 Seite 124 **[Ctrl]+[Leerschlag]**. Damit kann die Codierhilfe von Eclipse aufgerufen werden.
-
- 81 Seite 124 Beim «Reverse Engineering» wird aus dem Programmcode ein Klassendiagramm erzeugt.
-
- 82 Seite 124 CASE ist die Abkürzung von «Computer Aided Software Engineering» und bedeutet wörtlich computerunterstützte Softwareentwicklung.
-
- 83 Seite 124 Vorgehen:
1. Entweder über das Menü **File** oder über das Kontextmenü des jeweiligen Packages im Navigationsbereich von Eclipse (dort, wo die Klassen des Projekts angezeigt werden).
 2. Nach der Auswahl des Untermenüs **New** erscheint ein Dialogfenster, in dem alle notwendigen Angaben zur neuen Klasse gemacht werden können.
 3. Nach der Erfassung und Bestätigung dieser Angaben erstellt Eclipse eine Datei mit dem Gerüst einer Klassendeklaration.
-
- 84 Seite 145 Wovon ist die Rede? Welche «Dinge» werden immer wieder erwähnt?
-
- 85 Seite 145 Nationale Flüge weisen eine Flugnummer auf, die kleiner als 1000 ist.
-
- 86 Seite 145 Der Ausschnitt stellt die Klasse **Flughafen** dar. Diese Klasse hat zwei Attribute: eine Liste von Gates und einen Namen. Der Name ist von Datentyp `String`. Jeder Flughafen kann einen Flug landen und von einem Gate starten lassen und den Status eines Gates liefern.
-
- 87 Seite 145 Durch den Einsatz dieser Methode können die Regeln für nationale und internationale Flüge jederzeit geändert werden, ohne dass andere Klassen angepasst werden müssen, die diese Fluginformation abfragen bzw. benötigen.
-
- 88 Seite 145 Einen Array wie `Gate[] gates;`
-
- 89 Seite 145 Diese Methode überprüft, ob die Instanzvariable `flug` eine Referenz auf einen Flug hat (Wert = 0) oder nicht. Wenn kein Flug vorhanden ist, ist die Bedingung erfüllt und es wird `true` zurückgegeben: Das Gate ist zurzeit frei.
-
- 90 Seite 145 Wenn kein Flug vorhanden ist, kann in der letzten Zeile auch kein Flug nach dem Flugzeug abgefragt werden. Dies würde einen Laufzeitfehler verursachen und das Programm «zum Absturz» bringen.
-
- 91 Seite 145 Konstanten sind Variablen, die mit dem Schlüsselwort `final` angeführt werden.

-
- 92 Seite 145 Mit `static` wird aus einer Konstante eine Klassenkonstante, ohne dass Objekte dieser Klasse existieren müssen. Der Zugriff erfolgt durch das Voranstellen des Klassennamens.
-
- 93 Seite 154 Beim Unit-Test wird eine kleine Programmeinheit (meist eine Methode) getestet.
-
- 94 Seite 154 Der Test muss von der Klasse **TestCase** abgeleitet werden.
-
- 95 Seite 154 Ja: Jeder Testschritt bzw. jede Methode, die einen Testschritt implementiert, muss mit dem Wort «Test» beginnen.
-
- 96 Seite 160 `/**` (Schrägstrich plus Doppelstern)
-
- 97 Seite 160 Wichtige JavaDoc-Schlüsselwörter:
- Schlüsselwort für die Beschreibung von Übergabeparametern: `@param`
 - Schlüsselwort für die Beschreibung eines Rückgabewerts: `@return`
-
- 98 Seite 160 Jeweils direkt oberhalb des zu dokumentierenden Elements.
-
- 99 Seite 160 In der Dokumentation einer Methode, wenn diese eine (oder mehrere) Exception(s) werfen kann.

ConsoleReader

Unter Java kann die Benutzereingabe vereinfacht werden, indem eine entsprechende Klasse angelegt und implementiert wird. Für dieses Lehrmittel wurde dafür die Klasse **ConsoleReader** entwickelt, die eine Anzahl statischer Methoden zum typsicheren Einlesen von Benutzereingaben enthält. Folgende Methoden sind in **ConsoleReader** enthalten, um jeweils einen Wert des entsprechenden Datentyps vom Benutzer eingeben zu lassen:

- `readChar()`
- `readDouble()`
- `readInteger()`
- `readLong()`
- `readString()`

Jede Methode hat einen **String** als Eingabeparameter. Dieser String wird als Eingabeaufforderung angezeigt. Hier ein typischer Anwendungsfall. Es soll vom Benutzer das Geburtsjahr eingelesen werden. Hier bietet sich der Datentyp `int` an.

```
***  
int geburtsJahr = ConsoleReader.readInteger("Geben Sie Ihren Jahrgang ein");  
***
```

Der Programmcode der Klasse befindet sich in der Public Domain und steht unter der GPL (GNU General Public License). Dieser Code ist Bestandteil des Projekts **AirportScheduler**, das Sie von der Downloadseite des Verlags gratis herunterladen können. Vergleichen Sie dazu das Linkverzeichnis auf S. 11.

Nachfolgend sehen Sie das vollständige Listing von `ConsoleReader`^[1]:

```
/*  
 * This program is free software; you can redistribute it and/or  
 * modify it under the terms of the GNU General Public License  
 * as published by the Free Software Foundation; either version 2  
 * of the License, or (at your option) any later version.  
 *  
 * This program is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 * GNU General Public License for more details. http://www.gnu.org  
 */  
  
package ch.modul226.airport.view;
```

[1] Seit Java 5 gibt es Möglichkeiten, den Code der Klasse `ConsoleReader` zu vereinfachen. Neu gibt es die Klasse `Scanner`. Verschiedene Aufrufe der Methoden `doubleValue()`, `intValue()` etc. und der Methode `valueOf()` können ersatzlos entfallen, da es in Java 5 den Mechanismus `autoboxing/autounboxing` gibt.

```
import java.io.*;

/**
 * ConsoleReader, eine einfache utility-Klasse fuer Textkonsoleneingabe
 *
 * Diese Klasse dient dazu, in einem textkonsolenbasierten Java-Programm
 * einfache typsichere Eingaben zu machen.
 *
 * Verwendung:
 *     int zahl = utils.ConsoleReader.readInteger("Bitte ganze Zahl eingeben");
 *
 * analog geht es, wenn andere Datentypen vom Benutzer verlangt werden
 * sollen
 *
 * <p>
 * (C) Copyright 2004 by Markus Ruggiero, rucotec GmbH, Basel
 * @author Markus Ruggiero <markus@ruggiero.ch>
 */

/**
 * History:
 * -----
 * Sommer 2004 V0.9   mr  erstellt
 * Nov. 2004    V1.0   mr  fertig kommentiert
 */
public abstract class ConsoleReader {

    /**
     * Ein buffered reader als Singleton.
     *
     * Wird von allen Methoden verwendet, um von der Console zu lesen
     */
    private static BufferedReader reader = new BufferedReader(
        new InputStreamReader(
            System.in));

    /**
     * ConsoleReader default constructor.
     *
     * Ist private, damit es nicht moeglich ist, Objekte der Klasse zu erstellen.
     * Alle Methoden sind static
     */
    private ConsoleReader() {
        super();
    }

    /**
     * Liest ein einzelnes Zeichen von der Console ein
     *
     * @param prompt Eingabeaufforderung
     * @return das eingelesene Zeichen
     */
    public static char readChar(String prompt)
    {
        char value = 0;
        while (true)
        {
            try
            {
                System.out.print(prompt + ": ");
                value = (char) reader.read();
                // alternative value = (char)reader.readLine().charAt(0);
                break;
            }
            catch (IOException ex)
            {

```

```
System.out.println("*** Read Error ***");
        System.out.println("*** " + ex.getMessage());
        System.exit(1);
    }
}
return value;
}

/**
 * Liest einen double-Wert von der Console ein
 *
 * @param prompt Eingabeaufforderung
 * @return der eingegebene Wert
 */
public static double readDouble(String prompt)
{
    double value = 0.0;
    System.out.print(prompt + ": ");
    while (true)
    {
        try
        {
            value = Double.valueOf(reader.readLine()).doubleValue();
            break;
        }
        catch (IOException ex)
        {
            System.out.println("*** Read Error ***");
            System.out.println("*** " + ex.getMessage());
            System.exit(1);
        }
        catch (NumberFormatException ex)
        {
            System.out.println("*** Conversion Error ***, try again");
        }
    }

    return value;
}

/**
 * Liest einen int-Wert von der Console ein
 *
 * @param prompt Eingabeaufforderung
 * @return der eingegebene Wert
 */
public static int readInteger(String prompt)
{
    int value = 0;
    while (true)
    {
        try
        {
            System.out.print(prompt + ": ");
            value = Integer.valueOf(reader.readLine()).intValue();
            break;
        }

        catch (IOException ex)
        {
            System.out.println("*** Read Error ***");
            System.out.println("*** " + ex.getMessage());
            System.exit(1);
        }
        catch (NumberFormatException ex)
        {
            System.out.println("*** Conversion Error ***, try again");
        }
    }
}
```

```
return value;
    }

    /**
     * Liest einen long-Wert von der Console ein
     *
     * @param prompt Eingabeaufforderung
     * @return der eingegebene Wert
     */
    public static long readLong(String prompt)
    {
        long value = 0;
        while (true)
        {
            try
            {
                System.out.print(prompt + ": ");
                value = Long.valueOf(reader.readLine()).longValue();
                break;
            }
            catch (IOException ex)
            {
                System.out.println("**** Read Error ****");
                System.out.println("**** " + ex.getMessage());
                System.exit(1);
            }
            catch (NumberFormatException ex)
            {
                System.out.println("**** Conversion Error ****, try again");
            }
        }
        return value;
    }

    /**
     * Liest einen String von der Console ein
     *
     * @param prompt Eingabeaufforderung
     * @return der eingegebene Wert
     */
    public static String readString(String prompt)
    {
        String value = "";
        while (true)
        {
            try
            {
                System.out.print(prompt + ": ");
                value = reader.readLine();
                break;
            }
            catch (IOException ex)
            {
                System.out.println("**** Read Error ****");
                System.out.println("**** " + ex.getMessage());
                System.exit(1);
            }
        }
        return value;
    }
}
```

Stichwortverzeichnis

Symbols

«Ist-ein-Beziehungen» 83

A

Abgeleitete Klassen	38
Abstrakte Klassen	44
Abstrakte Klassen und Interfaces	44
Abstrakte Methoden	44, 89
Abstraktionen	31
Accessor	36
Aggregation	34
Airportscheduler	125
Anweisung	22
API Specification	103
API-Spezifikation	104
Arbeitsbereich für den Programmcode	116
Arbeitsverzeichnis	53
ArgoUML	122
ArithmeticException	95
Array-Index	58
Array-Länge	58
Array-Objekt	58
Arrays	58
Art der Ein- bzw. Ausgabe	109
Assoziation	34
Attribute	33, 84
Auf Instanzvariablen zugreifen	77
Ausgabestream	110
Autoboxing	60
Auto-Unboxing	60

B

Benutzerschnittstelle realisieren	133
Bereich zur Darstellung der Klassenübersicht	116
Beziehung	34
Beziehungsarten	34, 47
Blockkommentar	155
break-Anweisung	66
BufferedReader	111

byte	56
Byte-Code	54

C

Camel-Case	76
CASE-Tools	122
catch-Block	92
Checklisten	124
Class	105
Collections	61
Compiler	53
ConsoleReader	79, 110

D

Data Hiding	36, 80
Dateien lesen	113
Dateien schreiben	113
Dateinamen	112
Daten	31
Datenkapselung	36
Datentyp	47, 56
Datentyp für Methoden	70
Datentyp umwandeln	59
Datentypen	53
Datentypen für ganze Zahlen	56
Defaultkonstruktor	82, 85, 142
Defaultpaket	100
Defaultverzeichnis	100
Deklaration einer Klasse	76
Deklaration eines Interface	89
Denken in Funktionen und Datenstrukturen	32
Denken in Verhaltensweisen von Objekten	32
Deprecated	105
Diagrammtyp auswählen	122
do/while-Schleife	67
Dokumentation zur String-Klasse	104
Dynamische Programme	30

E

Eclipse	16, 115
---------	---------

Eigene Exceptions definieren	94
Eigenschaften der neuen Klasse definieren	119
Ein- und Ausgabenschnittstellen	108
Einfache Selektion	64
Einstieg	135
Einstieg in die Applikation	135
Einstiegspunkt	29, 54, 56
Entwicklungsumgebung	16, 115
Exception	40, 42, 92, 142
Exception auslösen	93
Exception erzeugen und werfen	142
Exception wird nicht abgefangen	93
Exception-Klasse	42, 94

F

Fachbegriffe	29
Fehlerabfragen	41
Fehlerbehandlung	40, 91
Fehlerstream	110
File	112
Flugnummer	125
Flugzeugtyp	125
Formatierung der Dokumente	159
for-Schleife	68
Funktion	31, 70
Funktionsbibliotheken	29, 30
Fussgesteuerte Schleife	67
Fussgesteuerte Schleife als Struktogramm	67

G

Ganzzahlen	56
Gatenummer	140
Gates	125
Generalisierung	37
Geschäftsklassen	134
Geschlossen	30
getter	36
Gültige Klassennamen	76

H

Hauptmenü	135
Help	105

HTML-Auszeichnungen	156
HTML-Dokumentation	157

I

if/else-Verzweigung	65
import-Anweisung	99
Index	105
index.html	158
Infobereich	116
Input-Klassen	108
InputStream (Methoden)	110
Instanz	21, 33
Instanziieren	44
Instanzmethoden	79
Instanzvariable	25, 33, 46, 77, 106
Instanzvariablen deklarieren	77
int	56
Integer	111
Interface	32, 42, 89
Internationale Gates	125
IOException	112
Ist-ein-Beziehung	37, 95

J

Java	101
Java Development Kit	53, 157
java.io	102, 108
java.io.InputStream	108
java.io.OutputStream	108
java.lang	101
java.math	101
java.swing	102
java.util	101
Java-Compiler	54
JavaDoc definieren	157
JavaDoc erstellen	157
JavaDoc-Anweisungen	156
JavaDoc-Kommentar	155
Java-Dokumentation	103
Java-Pakete	101
Java-Projekt	16

Java-Umgebung auf der Kommandozeile	53
javax	101
JDK	53, 157
JUnit	149

K

Kapselung	80
Kapselung realisieren	80
Keine Prototypen	70
Klartext der Exception	92
Klasse	20, 33, 53
Klasse Object	88
Klassenbeschreibung	106
Klassendefinition	55
Klassendeklaration	56
Klassendiagramm	46, 126, 129
Klassendiagramm definieren	123
Klassenframework	149
Klasseninformationen	106
Klassenmethoden implementieren	79
Klassenname	46, 76, 79
Klassenpfad	99
Klassenvariable	79
Klassenvariablen deklarieren	78
Kommandozeile	53
Kommazahlen	57
Kommentare in Java	155
Kompilervorgang	54, 55
Komposition	34
Konditionaler Operator	65
Konsoleneingabe	110, 112
Konsolenoberfläche	133
Konstanten	58
Konstantennamen	133
Konstruktionen für Daten und Funktionen	31
Konstruktor	60, 81, 84, 106
Kontrollstrukturen von Java	63
Kopfgesteuerte Schleife	66, 68
Kopfgesteuerte Schleife als Struktogramm	67

L

Lande- und Abflugzeit	125
Länge eines Arrays	58
Lauffähiges Programm erstellen	121
Leserichtung	129
Logische Werte	57
Logischen Verknüpfungsoperatoren	63
Lokale Variable	78
long	56

M

main()-Methode	17, 56
Mehrfachselektion	65
Mehrfachselektion als Struktogramm	66
Menüwahl	136
Methode	20, 33, 36, 46, 70, 84, 106
Methode aufrufen	81
Methode erstellen	70
Methode zur Abfrage einer Objektinformation	81
Methoden der Klasse Object	88
Methodenköpfe	90
Methodenrumpf	90
Methodensignatur	71
Modelle	29
model-Paket	134
Modularisierung	30
Module	30

N

Nachrichten	33
Nationale Gates	125
Neue Klasse erstellen	119
Neues Projekt erstellen	116
NumberFormatException	112

O

Oberbegriffe	37
Oberklasse	83
Object	88
ObjectOutputStream	109
Objektdiagramm	47

Objekte	31
ObjektInputStream	109
Objektname	47
Objektreferenz	78
Offen	30
OOD	9
OOP	9
Operationen	31
Operator	21
Ordnerstruktur	54
Output-Klassen	109
OutputStream (Methoden)	110
Overview	105

P

Package	105
Package-Explorer	116
Paket	16, 98
Paket java.io	102
Paket java.lang	101
Paket java.math	101
Paket java.util	101
Paket javax.swing	102
Paketangabe	98
Paketnamen	98
PAP	46
Parameter übergeben	70, 85
Parameterübergabe	71
Pass by reference	71
Pass by value	70
Passagierkapazität	125
Plug-ins	115
Polymorphismus	39, 44, 86, 139
Primitive Datentypen	56
Programmablaufpläne	46
Programmdokumentation	155
Projekt benennen	118
Projekttyp auswählen	117
Prozedurale Elemente der Programmiersprache	52

Q

Quellcode	54
Quelltext	54

R

Reader	110
Referenzaufruf	70
Regeln für Konstruktoren	82
Reverse Engineering	123
Rollenbezeichnungen	129
Routinen	29
Rückgabekanal	42
Run-Konfiguration	19
RuntimeExceptions	94

S

Schlecht gewählte Klassennamen	77
Schleifen	66
Schlüsselwort «this» verwenden	78
Schnittstellen zur Aussenwelt	108
Schreibweise für Kommentare	155
Semikolon	54
setter	36, 81
short	56
Shortcut-Operator	64
Signatur einer Methode	71
Softwarekrise	29
Sourcecode	54
Sourcecode schreiben	120
Sourcedatei erstellen	118
Sourceordner	118
Spezialisierung	37
Standard-Exception abfangen	143
Statische Methoden	111
Statische Programme	30
Statische Variablen	110
Steuerung des Hauptmenüs	136
String	25, 58, 104
Struktogramme	46
Struktur einer einfachen Selektion	64
Strukturen eines Java-Programms	55

Subklasse	37
Superklasse	37
switch/case-Anweisung	65
Syntax	9, 52
Syntaxfehler	120
System	110
System.err	110
System.in	110, 111
System.out	110

T

Templates	159
Ternärer Operator	65
Test auf Gleichheit	63
TestCase	152
Testcode	152
Testfall	150
Testfall erstellen	151
Testfallmethode bestimmen	151
Testfallmethoden	152
Testklasse erstellen	150
Testprojekt erstellen	149
Texteditor	53
Top-down-Ansatz	29
Tree	105
try-Block	42, 92
try-catch-Block	91, 110
try-catch-Konstruktion	42
Tutorials	103
Typumwandlung	86

U

Übergabeparameter	78
Überladen einer Methode	84
Überladene Methode	71
Umgang mit Exceptions	91
UML	37, 46
UML-Diagramme	46
Ungültige Klassennamen	77
Unicode-Zeichen	57
Unit-Test	149

Unit-Test fehlgeschlagen	153
Unit-Test ohne Fehler	153
Unit-Test starten	152
Unterbegriffe	37
Unterklasse	83
Use	105

V

Variable	21
Vererbung	37, 47, 83, 139
Vererbungsbeziehung	47
Vergleichsoperatoren	63
Verhalten der Objekte	32
Verknüpfungsoperatoren	63
Verzeichnis für die Dokumentation festlegen	158
Verzweigung	64
Verzweigung als Struktogramm	64
view-Paket	134
Virtuelle Java-Maschine	55
Vorgehensmethoden	29

W

Wertaufruf	70
while-Schleife	66
Wizard	150
Wizard für die Erstellung von Testklassen	150
Wrapper-Klasse	59
Writer	110

Z

Zeichenketten	57
Zeiger	41, 60
Zeilenkommentar	155
Ziel und Quelle der Ein- und Ausgabedaten	109
Zugriff auf Konstruktor der Oberklasse	85
Zugriff auf überschriebene Methode	85
Zugriffsmethode	36, 81
Zugriffsmethode definieren	81
Zugriffsmethoden regeln	81
Zugriffsmodifikatoren	88
Zugriffsrechte	89