

Inhalt

4	Einführung in die Programmiersprache Java (Teil IV)	4-2
4.7	<i>Strukturierung im Großen – Modularisierung</i>	4-2
4.7.1	Paradigmen	4-2
4.7.2	Objekte	4-2
4.7.3	Klassen zur Datenabstraktion	4-3
4.8	<i>Aufbau einer Klasse</i>	4-4
4.8.1	Objektvariablen und Objektmethoden	4-4
4.8.2	Beispiel „Würfel“	4-4
4.8.3	Klassenvariablen und Klassenmethoden	4-9
4.9	<i>Zusammenfassung</i>	4-11
4.10	<i>Konstanten und Methoden der Klasse <code>java.lang.Math</code></i>	4-12
4.11	<i>Methoden der Klasse <code>java.lang.String</code></i>	4-13
4.12	<i>Beispiel „Der einarmige Bandit“</i>	4-15

4 Einführung in die Programmiersprache Java (Teil IV)

4.7 Strukturierung im Großen – Modularisierung

4.7.1 Paradigmen

Imperative Sprachen unterstützen den herkömmlichen prozedurorientierter Ansatz:

Ein *komplexes Problem* wird schrittweise in genügend kleine *Teilprobleme* zerlegt (*Top-Down-Analyse*). Jedes dieser Teilprogramme wird als Prozedur in einer zur Verfügung stehenden Programmiersprache implementiert und dann in einer Hauptprozedur zu einem Programm zusammengefasst.

Algol ... *Algorithmic Language*, Fortran, Pascal, C

Objektorientierte Sprachen simulieren das menschliche Denkverhalten:

Das zu modellierende *Gesamtsystem* wird in *Teilsysteme* zerlegt, bestehend aus miteinander kommunizierenden Einheiten. Jeder solchen Einheit wird ein *Objekt* zugeordnet. Die Objekte verwalten ihre *Daten* und kommunizieren mit anderen Objekten.

Smalltalk

rein objektorientiert

Hybridsprachen

Sprachen, die zunächst das imperative Paradigma verfolgt haben, erhielten nachträglich eine objektorientierte Erweiterung. Sie besitzen demzufolge sowohl imperative als auch objektorientierte Sprachbestandteile und gestatten eine *nicht* streng objektorientierte Programmierung.

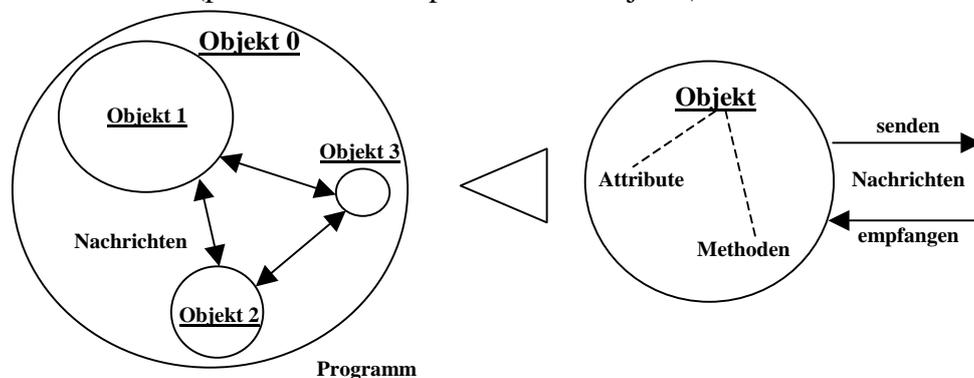
Objekt Pascal, C++, Java

sowohl imperativ als auch objektorientiert

4.7.2 Objekte

Ein **Objekt** im Sinne von **objektorientierten Programmierung (OOP)** besteht aus

- **Attributen:** *Daten* des Objekts (Strukturkomponente des Objekts).
- **Methoden:** *Prozeduren* zur Manipulation der Daten des Objekts (prozedurale Komponente des Objekts).



Ein objektorientiertes Programm ist ein System miteinander kommunizierender Objekte, d.h. sie selbst senden, empfangen und reagieren auf *Nachrichten*.

Schnittstelle eines Objektes: *Nachrichten, spezielle öffentlich zugängliche Methoden.*

Dieses Zusammenfassen von *Attributen* und *Methoden* wird als **Kapselung** (*information hiding*) bezeichnet, eine der wichtigsten Eigenschaften der OOP. Attribute und Methoden werden als ein *Modul* behandelt. **Schnittstellen** regeln die Interaktion mit anderen Objekten. Sie bestehen aus *öffentlich zugänglichen Methoden*. Die *Attribute* eines Objekts als gekapselte Datenstrukturen sollten *nicht öffentlich* und *nur durch eigene Methoden* manipulierbar sein. Eine Veränderung der Attribute von außen *widersprüche* dieser Programmierphilosophie.

Jedes Objekt verwaltet sich selbst!
--

4.7.3 Klassen zur Datenabstraktion

Objekte gleichen Aufbaus werden zu einer Klasse zusammengefasst.

Beispiel „Hund und Herrchen“

Eine *Klasse* **Hund** fasst alle Hunde verschiedener Rassen, Größe und Farbe zusammen. Jeder Hund selbst ist *Objekt* der Klasse und hat seine individuellen *Attribute*, wie Rasse: Riesenschnauzer, Farbe: schwarz, Größe: sehr groß. Hat ein Hund ein Herrchen, so ist dieses ein Objekt einer *anderen* Klasse, nennen wir sie **Mensch**. Das Herrchen gibt dem Hund den Befehl „Sitz“. Es schickt also eine *Nachricht* an den Hund. Der Hund reagiert darauf oder auch nicht. Er allein entscheidet, wie er sich auf den Befehl „Sitz“ verhält.

Klassen sind *Referenzdatentypen* im obigen Sinne und legen nicht nur Attribute sondern auch Methoden ihrer Objekte fest. Mit dem **new**-Operator erzeugt ein *Standardkonstruktor* ein neues Objekt der Klasse.

Vereinbarung einer Klasse

Klassenname *Objektnamen* ;

Objektnamen = **new** *Klassenname* () ;

oder

Klassenname *Objektnamen* = **new** *Klassenname* () ;

Beispiel

```
Hund hasso = new Hund() ;
```

```
Mensch hannes = new Mensch() ;
```

4.8 Aufbau einer Klasse

4.8.1 Objektvariablen und Objektmethoden

Objektvariablen und **Objektmethoden** sind die *objektspezifischen* Bestandteile einer Klasse.

Objekte werden auch als **Instanzen** bezeichnet. Deshalb spricht man auch von **Instanzvariablen** und **Instanzmethoden**.

Vereinbarung von Objektvariablen

Objektvariablen sind *Attribute* eines Objekts und wurden schon im Abschnitt über Referenzdatentypen - Klassen eingeführt. Sie fassen die *Daten eines Objekts* zusammen und bekommen mit ihrer Deklaration den Zusatz **private** (*nicht öffentlich*).

```
private Typ Variablenname [ = Ausdruck ];
```

Vereinbarung von Objektmethoden

Objektmethoden werden zur kontrollierten *Dateneingabe* (**set-Methoden**), gesicherten *Datenabfrage* (**get-Methoden**) und zur *Datenverarbeitung* (**service-Methoden**) *des Objektes* aufgenommen. Sie sollten *nur dann öffentlich* sein, wenn sie zur Kommunikation mit anderen Objekten notwendig sind. Öffentliche Methoden erhalten den Zusatz `public`, sonst `private`. In Java werden *Funktionen* als Methoden verwendet.

```
public / private Ergebnistyp Methodenname ( Parameterliste )      // Methodenkopf
{
  Anweisungen                                                    // Methodenrumpf
}
```

Zugriff

Da die Objektvariablen und Objektmethoden zum Objekt gehören, werden sie auch über den *Objektnamen* mittels **Punktoperator** aufgerufen, vorausgesetzt das Zugriffsrecht ist gegeben.

```
Objektname . Variablenname
Objektname . Methodenname ( Argumentenliste )
```

4.8.2 Beispiel „Würfel“

Ein Universalwürfel besitzt eine größte und kleinste Augenzahl. Diese sollen mit einer Methode eingetragen werden. Außerdem soll ein Würfel natürlich auch würfeln können.

Würfel	
-	augen : int
-	max : int
-	min : int
+	getAugen() : int
+	setWuerfel(minZahl : int, maxZahl : int) : void
+	wuerfeln() : int

Klassen stellt man grafisch in **Klassendiagrammen** dar. Neben dem Klassennamen werden Attribute mit ihrem Datentyp und Methoden mit der Parametertypliste und dem Ergebnistyp aufgeführt. Durch „-“ werden `private` und durch „+“ `public` als Zugriffsrechte gekennzeichnet.

Die Attribute `min` und `max` haben zunächst die Werte eines Standardwürfels. Mit der Methode `setWuerfel` kann man diese kontrolliert verändern. Die Methode `wuerfeln` erzeugt im Attribut `augen` einen gewürfelten Wert. Dieser ist mit der Methode `getAugen` zu erfragen.

Mit der Klasse **Wuerfel** kann man somit Würfel mit einer beliebigen Anzahl von Flächen simulieren, z. B. auch den Münzwurf.

Wuerfel.java

```
// Wuerfel.java                                MM 2009

/**
 * Wuerfel min .. max.
 */
public class Wuerfel
{
/* ----- */
// Attribute

/**
 * Minimale Augenzahl.
 */
    private int min = 1;

/**
 * Maximale Augenzahl.
 */
    private int max; = 6

/**
 * Aktuelle Augenzahl.
 */
    private int augen = min;

/* ----- */
// set-Methode

/**
 * Setzt Wuerfelattribute.
 * @param minZahl minimale Augenzahl
 * @param maxZahl maximale Augenzahl
 */
    public void setWuerfel( int minZahl, int maxZahl)
    {
        if( maxZahl < minZahl)
        {
            min = maxZahl;           // vertauschen
            max = minZahl;
        }
    }
}
```

```

    }
    else
    {
        min = minZahl;
        max = maxZahl;
    }

    wuerfeln();
}

/* ----- */
// get-Methode
/**
 * Liest aktuelle Augenzahl.
 * @return Augenzahl
 */
public int getAugen()
{
    return augen;
}

/* ----- */
// service-Methode
/**
 * Wuerfeln,
 * erzeugt zufällig ganze Zahl zwischen min und max.
 * @return Augenzahl
 */
public int wuerfeln()
{
    augen
    = (int)(Math.random() * ( max - min + 1)) + min;
    return augen;
}
}

```

Zu jeder Klasse gehört ein kleines Testprogramm. Wir wollen mit einem 1 .. 9 Würfel mehrmals würfeln und die mittlere Augenzahlen berechnen.

WuerfelTest.java

```

// WuerfelTest.java
import Tools.IO.*;
// Eingabe

/**
 * Testprogramm zur Klasse Wuerfel.
 */
public class WuerfelTest
{
/**
 * Wuerfelt mit einem 1..9 Wuerfel,
 * Berechnen der mittleren Augenzahl.
 */

```

```
public static void main( String[] args)
{
// Ueberschrift
  System.out.println();
  System.out.println( "Wuerfel 1..9, wuerfeln");

// Objekterzeugung Wuerfel
  Wuerfel wuerfelRot = new Wuerfel();
  wuerfelRot.setWuerfel( 1, 9);

// Anzahl Runden
  int anzahlRunden
  = IOTools.readInteger( "Wie oft wuerfeln? ");

// Summe
  int summe = 0;

// Wuerfeln
  for( int runde = 0; runde < anzahlRunden; runde++)
  {
    System.out.print( " " + wuerfelRot.wuerfeln(); );
    summe += wuerfelRot.getAugen();
  }

// Mittelwert
  double mittelwert = (double)summe / anzahlRunden;
  System.out.println();
  System.out.println( "Summe      : " + summe);
  System.out.println( "Mittelwert: " + mittelwert);

// Fertig
  System.out.println();
  System.out.println( "Programm beendet");
}
}
```

Mit dem nächsten Programm soll die Qualität des programmierten Würfels getestet werden. Wir verwenden den Standardwürfel. In einer Häufigkeitstabelle zählen wir die gewürfelten Augenzahlen bei einer genügend großen Anzahl von Versuchen (1Mrd.) und berechnen anschließend ihre relative Häufigkeit. Im Idealfall müsste sich für jede Augenzahl $1/6$ ergeben.

WuerfelFehler.java

```
// WuerfelFehler.java MM 2009
// Numerische Fehler
// (Datenfehler, Rechenfehler, Verfahrensfehler)

/**
 * Wuerfel 1..6,
 * experimentelle Ermittlung der relativen Haeufigkeiten
```

```
* der gewuerfelten Augenzahlen.
*/
public class WuerfelFehler
{
/**
 * Programm wuerfelt mehrmals mit dem Standardwuerfel,
 * erstellt eine Tabelle der absoluten Haeufigkeit
 * und berechnet die relativen Haeufigkeiten
 * der gewuerfelten Augenzahlen.
 */
    public static void main( String[] args)
    {
// Ueberschrift
        System.out.println( "Wuerfel 1..6");

// Wuerfel
        Wuerfel wuerfelBlau = new Wuerfel();

// Tabelle der absoluten Haeufigkeiten
        int[] tabelle = new int[ 6];

// Anzahl Runden
        int anzahlRunden = 1000000000;           // 1 Mrd.

// Wuerfeln
        System.out.println
        ( "ACHTUNG: Die Berechnung dauert einige Minuten!");
        for( int runde = 0; runde < anzahlRunden; runde++)
            tabelle[ wuerfelBlau.wuerfeln() - 1]++;

// Ausgabe der relativen Haeufigkeiten
        System.out.println
        ( "Tabelle der relative Haeufigkeiten");
        System.out.println();
        for( int i = 0; i < 6; i++)
        {
            System.out.print(( i + 1) + ": ");
            System.out.println
            ((double)tabelle[ i] / anzahlRunden);
        }

// Fertig
        System.out.println();
        System.out.println( "Programm beendet");
    }
}

/* ----- */
// Wahrscheinlichkeit ... beim idealen Wuerfel:
// 1 / 6 = 0.166667

// Wuerfel 1..6
```

```
// Tabelle der relative Haeufigkeiten

// 1: 0.166684948
// 2: 0.166656038
// 3: 0.1666613
// 4: 0.166657318
// 5: 0.166661714
// 6: 0.166678682
```

4.8.3 Klassenvariablen und Klassenmethoden

Klassenvariablen und **Klassenmethoden** gehören einer Klasse, sind also *klassenspezifische* Bestandteile. Sie existieren sogar, wenn es *keine* Objekte dieser Klasse gibt.

Ein typisches Beispiel für die Verwendung von *Klassenmethoden* wurde bereits mit der Klasse `java.lang.Math` eingeführt, `Math.sin()`. Alle Methoden dieser Klasse sind Klassenmethoden.

Klassenvariablen und *Klassenmethoden* werden von den *Objektvariablen* und den *Objektmethoden* durch das Schlüsselwort `static` unterschieden. Auch hier gilt Klassenvariablen sind spezielle *nicht öffentliche* Attribute, die *Klassendaten* speichern. Klassenmethoden dienen der *Bearbeitung* der *Klassendaten*. Sie sollten *nur dann öffentlich* sein, wenn sie zur Kommunikation notwendig sind.

Klassenvariablen

```
private static Typ Variablenname [= Ausdruck ];
```

Klassenkonstanten

```
public / private static final Typ Variablenname [= Ausdruck ];
```

Klassenmethoden

```
public / private static Ergebnistyp Methodename ( Parameterliste ) // Methodenkopf
{
    Anweisungen // Methodenrumpf
}
```

Zugriff

Da beide Komponenten zur Klasse gehören, erfolgt der Zugriff mittels **Punktoperator** über den *Klassennamen*.

```
Klassenname . Methodename ( Argumentenliste )
Klassenname . Variablenname
Klassenname . Konstantenname
```

Beispiele

main

Jedes Java-Programm ist eine Klasse. Eine der wichtigsten Klassenmethoden ist die Methode `main`. Sie kann als Programmstart oder auch als Testprogramm einer Klasse verwendet werden. Diese Methode ist die einzige, die *nicht direkt* aufgerufen, sondern bei der Ausführung einer Klasse aktiviert wird.

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

```
{  
}
```

In der Klasse `System` ist eine *Klassenvariable* `out` als Objekt einer Klasse `PrintStream` definiert. Mit der Anweisung `System.out.println()`; wird eine Objektmethode der Klasse `PrintStream` aufgerufen.

Mit der *Klassenmethode* `IOTools.readInteger()` der Klasse `IOTools` des Paketes `Tools` wird die Eingabe einer ganzen Zahl von der Tastatur ermöglicht.

Mit `Math.sin()` berechnet die entsprechende Winkelfunktion. `sin` ist eine Klassenmethode der Klasse `Math`, einer Klasse für mathematische Funktionen.

So wie man Klassenvariablen definieren kann, ist es auch möglich, *Klassenkonstanten* festzulegen. Klassenkonstanten sind spezielle, *nicht* veränderbare, Klassenvariablen.

`Math.PI` ist eine Klassenkonstante für die Ludolphsche Zahl π .

4.9 Zusammenfassung

Konzepte der OOP

- **Objekte** sind **Datengrundbausteine** (*Module*), aufgebaut aus **Attributen** (*Daten*) und **Methoden** (*Prozeduren*).
- *Objekte* kommunizieren miteinander über spezielle **öffentliche Methoden** (*Nachrichten*).
- **Klassen** sind Datentypen (*Datenabstraktion*) der Objekte. Sie fassen Objekte mit gleichem Aufbau zusammen und dienen der **Datenkapselung**.
- Bereits vorhandene **Klassenbibliotheken** bilden eine Basis für die Wiederverwendbarkeit von Softwarebausteinen.

Java Platform, All Classes:

<http://download.oracle.com/javase/7/docs/api/>

Faustregeln zur Modularisierung und Programmentwicklung

- ⇒ *Ein- und Ausgaben* sind im Hinblick auf grafische Oberflächen *nur* in der *main-Methode* oder in speziell dafür vorgesehenen Klassen enthalten.
- ⇒ *Kleine Klassen* sind durch Generalisierung bzw. Spezialisierung zu modellieren, dabei ist auf Wiederverwendbarkeit zu achten.
- ⇒ *Jede Klasse, jedes Attribut* und *jede Methode* in ihr ist mit einem *externen Kommentar* `/** ... */` für eine Dokumentation zu versehen.
- ⇒ *Methoden* sind gut zu strukturieren, *nicht länger* als eine A4-Seite. Ansonsten sind sie in kleinere Methoden aufzuteilen. Sie sind mit *internen Kommentaren* `// ...` bzw. `/* ... */` zu versehen.

4.10 Konstanten und Methoden der Klasse `java.lang.Math`

Die vordefinierte Klasse **Math** gehört zum Paket `java.lang` des Standardumfangs von Java. Dieses Paket wird beim Übersetzen automatisch eingebunden und muss deshalb *nicht* importiert werden. **Math** stellt mehrere mathematische **Klassenkonstanten** und **Klassenmethoden** zur Verfügung.

Klassenkonstanten

```
public static final double E;           // Leonard Euler
public static final double PI;        // Ludolf van Ceulen
```

Klassenmethoden (Auswahl)

Name	Parameteranzahl	Parametertyp	Ergebnistyp	Beschreibung
abs	1	double	double	Betrag eines Wertes
abs	1	float	float	Betrag eines Wertes
abs	1	long	long	Betrag eines Wertes
abs	1	int	int	Betrag eines Wertes
acos	1	double	double	Arcus Cosinus
asin	1	double	double	Arcus Sinus
atan	1	double	double	Arcus Tangens
ceil	1	double	double	ganzzahliges Aufrunden
cos	1	double	double	Cosinus
exp	1	double	double	E-Funktion
floor	1	double	double	ganzzahliges Abrunden
log	1	double	double	natürlicher Logarithmus
max	2	double	double	Maximum
max	2	float	float	Maximum
max	2	long	long	Maximum
max	2	int	int	Maximum
min	2	double	double	Minimum
min	2	float	float	Minimum
min	2	long	long	Minimum
min	2	int	int	Minimum
pow	2	double	double	Potenzfunktion
random	0		double	Zufallszahl zwischen 0 und 1
round	1	double	double	ganzzahliges Runden
sin	1	double	double	Sinus
sqrt	1	double	double	Quadratwurzel
tan	1	double	double	Tangens

Beispiel

$$y = \frac{(a-b)^3 - 1}{1 + \sin^2 x}$$

```
double a, b, x, y;
y = ( Math.pow( a - b, 3.0) - 1)
    / ( 1 + Math.sin( x) * Math.sin( x));
```

```
double x, y;
```

```
y = 1 + Math.sqrt( x) / 2
    + Math.sqrt( Math.sqrt( x)) / 24;
```

$$y = 1 + \sqrt{x}/2 + \sqrt{\sqrt{x}}/24$$

4.11 Methoden der Klasse `java.lang.String`

Zeichenketten werden in Java *nicht* mittels eines *elementaren Datentyps*, sondern in Form eines *Referenzdatentyps* namens `String` dargestellt. Diese vordefinierte Klasse gehört ebenfalls zum Paket `java.lang` des Standardumfangs von Java.

Eine Möglichkeiten zum **Erzeugen eines Stringobjekts** erfolgt mit der Zuweisung einer Zeichenkettenkonstante:

```
String s1 = "abc";
```

Die Klasse `String` stellt eine Reihe von Methoden für *Stringoperationen* bereit.

Klassenmethoden

Name	Parameteranzahl	Parameter-typ	Ergebnis-Typ	Beschreibung
valueOf	1	boolean char char[] double float int long Object	String	Umwandlung in einen String

Objektmethoden (Auswahl)

Name	Parameteranzahl	Parameter-typ	Ergebnis-Typ	Beschreibung
charAt	1	int	char	Zeichen an Stelle <code>int</code>
compareTo	1	String	1, 0, -1	lexikografischer Vergleich
endsWith	1	String	boolean	Vergleich Endung
equals	1	String	boolean	Vergleich auf Gleichheit
equalsIgnoreCase	1	String	boolean	Vergleich auf Gleichheit, ohne Groß-/Kleinschreibung
indexOf	1	String	int	Suche nach Teilstring
length	0		int	Stringlänge
replace	2	char char	String	Einsetzen
startsWith	1	String	boolean	Vergleich Anfang
substring	1	int	String	Teilstring ab Stelle <code>int</code>
substring	2	int int	String	Teilstring von Stelle <code>int</code> bis Stelle <code>int</code>
toCharArray	0		char[]	Umwandeln in ein char-Feld
toLowerCase	0		String	Umwandeln in Kleinbuchstaben

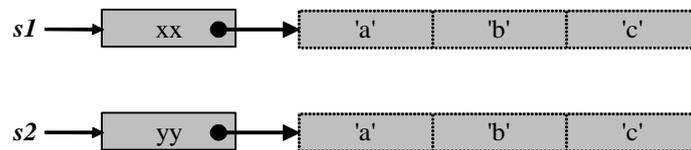
toUpperCase	0		String	Umwandeln in Großbuchstaben
--------------------	---	--	--------	-----------------------------

Im Gegensatz zu der Klasse `Math` sind diese *keine Klassenmethoden*, sondern *Objektmethoden*. Ihr Aufruf erfolgt mit dem Punktoperator, aber nicht über die Klasse sondern über ein aufrufendes Objekt. Ein Vergleich der gespeicherten Werte in zwei Zeichenketten erfolgt mit der Methode `equals()`:

```
String s2 = "abc";
s1.equals( s2)           =>           true
```

`s1` und `s2` sind Referenzvariablen, die auf unterschiedliche Stringobjekte verweisen. Referenzdatentypen liefern bei einem unmittelbaren Vergleich der Variablen mittels `==` stets den Wert **false**, da hier die Adressen der Referenzen verglichen werden:

```
s1 == s2                 =>           false
```



4.12 Beispiel „Der einarmige Bandit“

Spielregeln: Als Anfangskapital besitzt man 300 Euro. Der Einsatz wird festgelegt. Drei Zahlen zwischen 0 und 9 werden gewürfelt. Sind alle drei Zahlen gleich, wird das Vierfache, sind zwei Zahlen gleich, das Doppelte des Einsatzes zurückgezahlt, sonst wird der Einsatz vom Kapital abgezogen.

Das Spiel endet, falls das Kapital aufgebraucht wurde bzw. falls es 500 Euro überschreitet.

Bandit
- kapital: int
- runde: int
- auswertung(): void
- erlaeuterung(): void
- spielen(): void
+ starten(): void

Bandit.java

```
// Bandit.java
import Tools.IO.*;

/**
 * Spielautomat:
 * Der einarmige Bandit, Spiel fuer eine Person.
 * Anfangskapital: 300 Euro
 * Drei zufaellige Zahlen 0..9 werden gewuerfelt.
 * Sind alle drei Zahlen gleich, wird das Vierfache,
 * sind zwei Zahlen gleich, das Doppelte
 * des Einsatzes zurueckgezahlt.
 * Sonst wird der Einsatz vom Kapital abgezogen.
 * Das Spiel endet, falls das Kapital aufgebraucht
 * wurde bzw. falls es 500 Euro ueberschreitet.
 */
public class Bandit
{
    /* ----- */
    // Attribute

    /**
     * Kapital.
     */
    private int kapital = 300;

    /**
     * Spielrundenzaehler.
     */
    private int runde = 0;

    /* ----- */
    // Spielaufbau

    /**
     * Spiel starten.
     */
}
```

MM 2009

//Eingaben

```
public void starten()
{
// Erlaeuterung
    System.out.println();
    erlaeuterung();

    System.out.println();
    IOTools.readLine( "Weiter (ENTER) ");

// Spielstart
    char weiter = 'j';
    do
    {
        System.out.println(); // Spiel
        spielen();

        System.out.println(); // Auswertung
        auswertung();

        // Weiter
        weiter = IOTools.readChar( "Noch einmal (j/n)? ");
    } while( weiter == 'j');

// Beenden
    System.out.println();
    System.out.println( "Spiel beendet!");
}

/* ----- */
// Spiel
/**
 * Spielerlaeuterung.
 */
private void erlaeuterung()
{
    String str = "Der einarmige Bandit, ";
    str += "Spiel fuer eine Person.\n\n";

    str += "Anfangkapital: 300 Euro\n";
    str += "3 zufaellige Zahlen 0..9 werden gewuerfelt.\n";
    str += "Sind alle Zahlen gleich, wird das Vierfache,\n";
    str += "sind 2 Zahlen gleich, das Doppelte ";
    str += "des Einsatzes zurueckgezahlt.\n";
    str += "Sonst wird der Einsatz vom Kapital abgezogen.\n";
    str += "Das Spiel endet, ";
    str += "falls das Kapital aufgebraucht\n";
    str += "wurde bzw. falls es 500 Euro ueberschreitet.";

    System.out.println( str);
}

/**
 * Spielen
```

```
* Einsatz, Wuerfeln, Gewinnausschuetzung.
*/
private void spielen()
{
// Wuerfel
Wuerfel wuerfel = new Wuerfel();
wuerfel.setWuerfel( 0, 9);

// Spielen
runde = 0;
kapital = 300;
do
{
runde++;

int einsatz;
do
{
System.out.println // Einsatz
( "Dein Kapital: " + kapital + " Euro!");
einsatz
= IOTools.readInteger( "Dein Einsatz: ");
} while( kapital < einsatz || einsatz <= 0);
kapital -= einsatz;

int zahl1 = wuerfel.wuerfeln(); // Zug
int zahl2 = wuerfel.wuerfeln();
int zahl3 = wuerfel.wuerfeln();
System.out.println
( zahl1 + " " + zahl2 + " " + zahl3);

int f = 1; // Gewinnausschuetzung
if( zahl1 == zahl2) f++;
if( zahl1 == zahl3) f++;
if( zahl2 == zahl3) f++;
if( f != 1) kapital += f * einsatz;
} while( kapital > 0 && kapital < 500);
}

/**
* Spielauswertung, Ermittlung des Gewinners.
*/
private void auswertung()
{
String str = "Runde: " + runde + "\n";
if( kapital <= 0) str += "Du bist Pleite!";
if( kapital >= 500) str += "Der Bandit ist Pleite!\n";

System.out.println( str);
}

/* ----- */
```

```
                                                    // Programm
/**
 * Programm, erzeugt und startet einen Spielautomaten.
 */
public static void main( String[] args)
{
// Spielautomat erzeugen
    Bandit bandit = new Bandit();

// Spielautomat starten
    bandit.starten();
}
}
```