JavaKara

Den Marienkäfer Kara kann man selber programmieren. Dazu wird die Programmiersprache Java verwendet. Hier ist eine Anleitung in drei Schritten für JavaKara Programme. Dazu muss das JavaKara Programm

Mein eigenes JavaKara-Programm in 3 Schritten:

1. **Programm schreiben:**
   Unser JavaKara-Programm schreiben wir im Editor, der sich öffnet, wenn man auf „Programmieren“ klickt. Damit das JavaKara-Programm in die Kara-Umgebung eingebunden werden kann, müssen gewisse Dinge erfüllt sein. Der untenstehende Rahmen muss eingehalten werden. Bei einer neuen Datei wird er automatisch erstellt.

   ```
   import JavaKaraProgram;
   public class MyClass extends JavaKaraProgram {
       public void myProgram() {
             // hier kommen unsere Anweisungen...
       } //myProgram
   } //MyClass
   ```

   Vergesst nicht, das Programm zu speichern. Der Name der Datei muss gleich wie der Name der Klasse sein: Hier also MyClass.java .

   In JavaKara gibt es folgende Anweisungen:

`kara.move()`	Kara läuft einen Schritt vorwärts
`kara.turnRight()`	Kara dreht sich an Ort und Stelle nach rechts
`kara.turnLeft()`	Kara dreht sich nach links
`kara.putLeaf()`	Kara legt ein Kleeblatt ab (an der Stelle, wo Kara steht)
`kara.removeLeaf()`	Kara nimmt ein Kleeblatt auf (an der Stelle, wo er steht)

   In JavaKara gibt es Sensoren, mit denen die Umgebung überprüft werden kann:

`kara.treeFront()`	Gibt es einen Baum auf dem Feld unmittelbar vor Kara?
`kara.treeLeft()`	Gibt es einen Baum auf dem Feld links von Kara?
`kara.treeRight()`	Gibt es einen Baum auf dem Feld rechts von Kara?
`kara.mushroomFront()`	Gibt es einen Pilz auf dem Feld vor Kara?
`kara.onLeaf()`	Steht Kara auf einem Kleeblatt?

2. **Programm kompilieren:**
   Damit das Programm vom Computer ausgeführt werden kann, muss es mit einem Compiler übersetzt werden. Das geschieht bei JavaKara automatisch, wenn man im Fenster mit Karas Welt den „Play“ Knopf drückt (die Taste mit dem Symbol „>“). Man kann aber ein Programm auch kompilieren, um zu überprüfen, ob es schon korrekt ist. Dazu kann man den Knopf „Programm kompilieren“ verwenden. In beiden Fällen muss das Programm zuvor gespeichert werden wie in Schritt 1 erklärt.
   Das kompilierte Programm wird automatisch unter dem gleichen Namen mit der Endung .class abgelegt: Hier also MyClass.class .
   Falls es noch Fehler im Programm hat, so werden diese in der unteren Hälfte des Editors angezeigt. Ein Klick mit der Maus auf die entsprechnende Meldung zeigt den Ort des Fehlers im Programm an. Die Fehlermeldungen sind manchmal leider auch verwirrend. Schaue auch die Zeile vor- und nachher an, vielleicht hast du ja nur irgendwo einen Strichpunkt vergessen. Sobald das Programm fehlerfrei kompiliert werden kann, weiterfahren mit Punkt 3.

3. **Programm laufen lassen:**
   Jetzt brauchst du nur noch den Play-Knopf zu drücken, um dein Programm zu starten.

Erstes JavaKara Programm: Kara legt ein Kleeblatt ab

Kara soll an der Stelle, an der er steht, ein Kleeblatt ablegen und einen Schritt weiter gehen, so dass man das Kleeblatt sieht.
Startet JavaKara und tippt das Programm ein. Geht dabei gemäss den drei Schritten der JavaKara Anleitung vor. Ihr habt 20 Minuten Zeit. Falls Ihr noch Zeit habt, könnt Ihr das zweite JavaKara Programm (Kara sammelt Kleeblätter) auch noch ausprobieren.

```
import JavaKaraProgram;
public class DropLeaf extends JavaKaraProgram {
  public void myProgram() {
    kara.putLeaf();    // Kara legt ein Kleeblatt ab
    kara.move();        // Kara geht einen Schritt weiter
   }
 }
```

Zweites JavaKara Programm: Kara sammelt Kleeblätter

Kara soll alle Kleeblätter auflesen, über die er hinwegläuft. Das untenstehende Programm CollectLeaves erfüllt diese Aufgabe.

```
import JavaKaraProgram;
public class CollectLeaves extends JavaKaraProgram {
  public void myProgram() {
    if (kara.onLeaf())  {
      kara.removeLeaf();
    } //if
    while (!kara.treeFront()) {
      kara.move();
      if (kara.onLeaf())  {
        kara.removeLeaf();
      } //if
    } //while
  } //myProgram
} //CollectLeaves
```

Falls die Zeit noch reicht: Erweitert das Programm mit anderen Anweisungen. Z.B. kann Kara einen Zickzack-Weg gehen. Oder Kara kann jedesmal, wenn er ein Kleeblatt aufhebt, die Richtung ändern. Oder was könnte Kara tun, wenn ein Baum vor ihm steht?
Nutzt die verbleibende Zeit, um eigene Ideen umzusetzen!

Gruppenarbeit (Puzzlemethode)

Java lernen mit JavaKara.

Gestern habt Ihr den Marienkäfer Kara kennengelernt. Heute werden wir mit Hilfe der Programmiersprache Java den Käfer programmieren.

Jede von euch wird sich in einen kleinen Themenbereich einarbeiten und anschliessend ein paar Kolleginnen darüber unterrichten.

Ziele:
Bis heute Nachmittag werdet Ihr folgende vier Dinge in Java kennen lernen:

- Programme, die verzweigen. Je nach dem, ob eine Bedingung erfüllt ist, wird ein Programmteil ausgeführt oder nicht. Das sind die sogenannten if-Anweisungen.
- Die Bedingungen, wie man sie bei den if-Anweisungen findet, können auch komplizierter sein. Sie heissen Bool'sche Ausdrücke.
- Programmteile, die wiederholt ausgeführt werden. Das sind die Schleifen.
- Zusammenfassungen von Programmanweisungen zu Blöcken, die aufgerufen werden können. Das sind die sogenannten Methoden (in anderen Sprachen werden sie z.B. Prozeduren oder Funktionen genannt). Beispiel: "Kara, gehe um den Baum herum!". Kara weiss da genau, was zu tun ist. Er braucht keine weitere Informationen.

Arbeitsweise:

Teams (nach Farben)
Zuerst bilden wir Teams von je 4 Personen (bzw. 5 wenn es nicht aufgeht). Jedes Team erhält ein farbiges Team-Blatt (rot, blau, gelb, grün, ...). Die Teams erkennt man an den Farben.

Expertinnen
Das Thema rund um JavaKara wurde in 4 Teile eingeteilt. Jede Person in den Teams sucht sich nun eines dieser Themen aus (bei Fünfergruppen arbeiten zwei Anfängerinnen zusammen). Die Person wird sich mit diesem Thema auseinandersetzen (in der sogenannten Expertenrunde) und anschliessend ihre Teamkolleginnen darüber informieren (Unterrichtsrunde). Jede von euch erhält zudem ein Namensschild-Blatt. Das Namensschild-Blatt hat dieselbe Farbe wie das Team-Blatt (so wisst Ihr immer, zu welchem Team Ihr gehört). Auf dem Namensschildblatt steht, welche Expertin Ihr seid (A = if-Expertin, B = Expertin für Bool'sche Ausdrücke, C = Schleifen-Expertin oder D = Methoden-Expertin). Schreibt auch euren Namen auf das Namensschild-Blatt.

Achtung: Die Themen sind nicht alle gleich schwer. Wer schon Programmiererfahrung hat, nimmt bitte ein etwas schwierigeres Thema (C oder D). Die Themen A und B sind für Anfängerinnen besser geeignet.

Wir arbeiten in zwei Schritten:

1. Expertenrunde:
Zuerst werden sich je 7 von euch mit einem Thema vertraut machen. In einer sogenannten Expertenrunde können alle 7 die Schwierigkeiten dieses Themas diskutieren. Zugleich überlegt Ihr euch in der Expertenrunde, wie Ihr euren Kolleginnen euer Thema am besten präsentieren könnt. Dazu gibt es auf der folgenden Seite eine kurze Anleitung.
2. Unterrichtsrunde:
Anschliessend findet die Unterrichtsrunde statt: Die Teams setzten sich wieder nach Farben zusammen. Also alle roten an einen Tisch, alle blauen an einen Tisch etc. Jede Expertin berichtet nun den Kolleginnen während rund 10 Minuten über ihr Thema.

Fortsetzung Gruppenarbeit (Puzzlemethode)

Material:
Anleitung zur Puzzlemethode, Arbeitsblatt, Unterlagen zu den Themen, farbige Team-Blätter und farbige Namensschild-Blätter.

Lösung:
Wir erwarten, dass das Arbeitsblatt in der Expertenrunde gemeinsam von allen Expertinnen ausgefüllt wird. Das Thema muss den Teamkolleginnen in verständlicher Art und Weise mitgeteilt werden.
Alle sollen am Schluss dieses Puzzles einen Einblick in die verschiedenen Programmierkonstrukte if, while, Methoden und die Bool'schen Ausdrücke haben. Mittwoch bis Freitag werden wir diese Themen immer wieder vertiefend gebrauchen. Versucht also die Programmierkonstrukte zu verstehen, und stellt Fragen, wenn etwas unklar ist.

Zeitplan:

Bezeichnung	Aktivität	Zeit
Teams bilden und Expertinnen bestimmen	4-er Gruppen bilden und Themen verteilen. Pro Team möglichst erfahrene Programmiererinnen *und* absolute Anfängerinnen	5 Minuten
Expertinnen allein	Jede Expertin erarbeitet allein ihr Thema (falls es zu wenig Computer hat: zusammen mit einer gleichen Expertin aus einem anderen Team). Tippt die Programme aus den Unterlagen ab und probiert sie aus.	Etwa 40 Minuten
Dazwischen: PAUSE		15 Minuten (11:00 bis 11:15)
Expertinnenrunde	Alle Expertinnen eines Themas (z.B. alle if-Expertinnen) treffen sich. Probleme klären. Habt Ihr alles richtig verstanden? Arbeitsblatt gemeinsam bearbeiten. Überlegen, wie man das Thema den Kolleginnen im Team beibringen kann.	Etwa 40 Minuten
Mittagspause		
zurück zum Team	Alle Teams setzen sich nach Farben zusammen.	5 Minuten
Unterrichtsrunde	Jede Expertin präsentiert während 10 Minuten ihr Thema den Teamkolleginnen.	40 Minuten

Arbeitsblatt: Puzzlemethode

Ihr seid nun Expertin auf einem gewissen Thema. In der Unterrichtsrunde sollt Ihr euer Thema den Teamkolleginnen präsentieren. Dazu habt Ihr nur 10 Minuten Zeit. Ihr müsst euch also genau überlegen, wie Ihr euer Thema darstellen wollt.
Unten sind ein paar Fragen, die euch helfen sollen, euer Thema zu präsentieren. Es ist euch selbstverständlich frei gestellt, das Thema auf irgend eine andere Art zu präsentieren. Beantwortet die Frage jedoch ohnehin als Vorbereitung.

1. Wie lautet das Thema?
   ---
2. Was war die Aufgabe?

   ---
3. Wie habt Ihr sie gelöst?

   ---
4. Gab es dabei besondere Probleme?

   ---
5. Wie seid Ihr mit den Problemen umgegangen?

   ---
6. Was habt Ihr gelernt ?

   ---
7. Tipp für eure Kolleginnen

   ---
8. Bemerkungen:

   ---
9. Notizen für die Präsentation:

Bitte denkt daran, dass einige unter euch nur sehr wenig oder gar keine Programmiererfahrung haben. Stellt das Thema also so dar, dass auch Anfängerinnen verstehen können, worum es geht.

Methoden in JavaKara im Überblick

Klasse kara

`void move()`	Schritt vorwärts
`void turnLeft()`	90°-Linksdrehung
`void turnRight()`	90°-Rechtsdrehung
`void putLeaf()`	Kleeblatt hinlegen
`void removeLeaf()`	Kleeblatt aufnehmen
`boolean treeFront()`	Kara vor Baum?
`boolean treeLeft()`	Baum links von Kara?
`boolean treeRight()`	Baum rechts von Kara?
`boolean onLeaf()`	Kara auf Kleeblatt?
`boolean mushroomFront()`	Kara vor Pilz?
`void setPosition (int x, int y)`	Kara von Position (x,y)
`java.awt.Point getPosition()`	Koordinaten der aktuellen Position

Klasse world

`void clearAll()`	Alle Elemente entfernen
`void setLeaf (int x, int y, boolean putLeaf)`	Blatt legen oder entfernen
`void setTree (int x, int y, boolean putTree)`	Baum setzen oder entfernen
`void setMushroom (int x, int y, boolean putMushroom)`	Pilz setzen oder entfernen
`void setSize (int newSizeX, int newSizeY)`	Grösse der Welt ändern
`boolean isEmpty (int x, int y)`	Ist das Feld leer?
`boolean isTree (int x, int y)`	Ist ein Baum auf dem Feld?
`boolean isMushroom (int x, int y)`	Ist ein Pilz auf dem Feld?
`boolean isLeaf (int x, int y)`	Ist ein Blatt auf dem Feld?
`int getSizeX()`	Horizontale Grösse der Welt
`int getSizeY()`	Vertikale Grösse der Welt

Klasse tools

`void println (String string)`	String auf Standard-Ausgabe schreiben
`void showMessage (String message)`	String in Dialogfenster ausgeben
`int random (int bound)`	Liefert Zufallszahl aus Bereich [0..bound]
`void checkState()`	Schaut auf den Geschwindigkeitsregler
`void sleep (int ms)`	Schläft `ms` Millisekunden
`String stringInput (String title)`	String in einem Dialogfenster mit Titel `title` eingeben; gibt `null` zurück, wenn der Dialog mit Cancel abgebrochen wurde
`int intInput (String title)`	Ganzzahl in einem Dialogfenster mit Titel `title` eingeben; gibt `Integer.MIN_VALUE` zurück, wenn der Dialog mit Cancel abgebrochen wurde oder nicht eine Ganzzahl eingegeben wurde
`double doubleInput (String title)`	Fliesskommazahl in einem Dialogfenster mit Titel `title` eingeben; gibt `Double.MIN_VALUE` zurück, wenn der Dialog mit Cancel abgebrochen wurde oder nicht eine Fliesskommazahl eingegeben wurde

JAVA – Kurzreferenz für das Schnupperstudium

Das Wort Java ist altenglisch und bedeutet Kaffee (ein weit verbreitetes Getränk bei übermüdeten Programmierern und Programmiererinnen).
Java wurde um 1990 von James Gosling und Bill Joy bei der Firma SUN entwickelt. Ihr könnt Java unter http://java.sun.com herunterladen.

Der Kern von Java ist relativ klein. Je nachdem, welche Art von Programm man schreiben will, muss man die entsprechenden Zusätze importieren. Diese Zusätze nennt man Klassenbibliotheken (engl. Libraries). Jeweils am Anfang jedes Programmes (oder Applets) werden zuerst die nötigen Libraries importiert: z.B. `import java.awt.*;` importiert alles, was mit graphischen Darstellungen auf dem Bildschirm zu tun hat.

Kommentare

Um Programme leichter lesbar zu machen, schreibt man Kommentare, z.B. wozu man Variablen und Konstanten verwenden möchte, oder was man sich bei einem Algorithmus überlegt hat. Kommentare werden vom Computer ignoriert. Man kann somit schreiben was man will.
Kommentare in Java werden einen Schrägstrich und einen Stern begrenzt:
`/* Das ist ein Kommentar. */` Kommentare können auch über mehrere Zeilen gehen.

Wenn nur bis zum Ende der Zeile ein Kurzkommentar eingefügt wird, dann kann dieser auch durch zwei Schrägstriche // angekündigt werden. Das Ende des Kommentars ist automatsich das Zeilenende.
Also z.B. `// Kommentar bis ans Ende dieser Zeile.`

Gross- und Kleinschreibung unterscheiden!

In Java wird Gross- und Kleinschreibung unterschieden. Wenn man z.B.eine Variable count definiert und später Count verwendet, gibt es einen Fehler. Gross Count ist nicht definiert.

Konvention (wird oft eingehalten – man muss jedoch nicht, wenn man nicht will...)
Konstanten mit Grossbuchstaben bezeichnen: z.B. FENSTERGROESSE
Methoden mit Kleinbuchstaben beginnen: z.B. berechneFahrstrecke(..)
Klassen mit Grossbuchstaben beginnen: z.B. MeineKlasse .

Variablen und Konstanten

Variablen werden definiert, indem der Typ, der Name der Variable und ein erster Wert (Initialisierungswert) angegeben wird.

Hier einige Typen von Variablen (für unsere Übung sollten diese ausreichen):

- Integer (ganze Zahlen, auch negative): `int`
- Double (Gleitkommazahl mit doppelter Rechengenauigkeit): `double`
- Boolean (Wahrheitswert richtig oder falsch): `boolean`
- Character (ein einzelner Buchstabe): `char`
- String (eine Zeichenkette): `String`

Beispiele (Variablen):

```
int counter = 0;  /* Zählervariable, die auf Null gesetzt ist */
```

Konstanten werden prinzipiell gleich definiert wie Variablen. Bei den Konstanten steht jedoch der Begriff *final* davor. Das bedeutet, dass der Wert nicht verändert werden kann - also konstant ist.

Beispiele (Konstante):

```
final int MAXIMUM = 100; /* Konstante MAXIMUM vom Typ Integer mit Wert 100 */
final double PI = 3.1415926535; /* PI ist eine Konstante */
```

Kompatibilität von Typen:

Integer-Variablen können untereinander beliebig kombiniert werden (z.B. durch Addition, Subtraktion, Zuweisung, etc.). Double Variablen können ebenfalls untereinander beliebig kombiniert werden. Man kann auch einer Double-Variablen einen Integer-Wert zuweisen.
Das Umgekehrte geht allerdings nicht. In einer Integer-Variablen hat kein Double-Wert Platz.
Für dieses Problem gibt es die Typumwandlung.

Beispiel:

```
final double pi = 3.1415926535;
int ganzeZahl;
ganzeZahl := (int)pi;  /* Pi kann der ganzen Zahl nur zugewiesen werden, wenn es zuerst
                          in einen Integer umgewandelt wird, also 3. */
```

Der Klammerausdruck `(int)` bedeutet, dass die nachfolgende Variable zum Typ Integer umgewandelt wird. Die Typumwandlung braucht man z.B. wenn man einen Double Wert berechnet und diesen nachher am Bildschirm darstellen will. Die Koordinaten des Bildschirms sind jedoch vom Typ Integer.

Operatoren

In Java gibt es sehr viele Operatoren. Hier sind nur ein paar wenige aufgeführt.

Operator	Beschreibung	Beispiel
`++`	Wert wird um eins erhöht	`int i=2;` `i++;  /* i hat nun den Wert 3 */`
`--`	Wert wird um eins erniedrigt	`int i=2;` `i--; /* i hat nun den Wert 1 */`
`!`	Logische Negation, not-Operator	`if (!(i<4)) {...}` */* falls i nicht kleiner als 4 ist. Klar, das könnte man auch einfacher haben: if (i>=4) {...} */*
`* / + -`	Arithmetische Operationen	`i= (5-3)*12;` */* i hat den Wert 24 */*
`==`	Gleichheit	`while (i==7) {...}` */* Solange i den Wert 7 hat.. */*
`!=`	Ungleichheit	`if (i!=3) {...}` */* Falls i ungleich 3 ist... */*
`&&`	Logische Und-Verknüpfung	`while ((i<4) && (j>2)) {..}` */* Solange i kleiner 4 und j > 2 ist - d.h. solange i gleich 3 ist... */*
`||`	Logische Oder-Verknüpfung	`if ((i>7)||(j<4)) {..}` */* Falls i grösser als 7 oder j kleiner als 4 ist… */*

Anweisungen

In Java werden Anweisungen durch einen Strichpunkt "**;**" getrennt. Anweisungen können sich auch über mehrere Zeilen erstrecken (z.B. eine if-Anweisung). Umgekehrt können auch mehrere Befehle auf einer Zeile stehen (getrennt durch einen Strichpunkt).

Blöcke von mehreren Anweisungen werden mit geschweiften Klammern { } zusammengefasst. Blöcke werden z.B. in Schleifen und in if-Anweisungen verwendet.

Schleifen

Als Schleife bezeichnet man eine Gruppe von Programmieranweisungen, die mehrmals hintereinander ausgeführt werden.

Eine Schleife, die nie stoppt, heisst Endlosschleife.
Z.B. `while (2<3) {screen.DrawString("immer noch endlos", 10, 10); }`
/ solange 2 kleiner ist als 3 (und das ist ja immer der Fall) wird der obige String an der Position (10,10) auf den Bildschirm geschrieben. */*

for-Schleife
Bei der for-Schleife ist schon am Anfang bekannt, wie oft die Schleife durchlaufen werden soll. Eine Zählervariable wird auf einen Wert initialisiert (z.B. i=1). Bei jedem Durchlauf der Schleife wird diese Zählervariable verändert (z.B. um eins erhöht, das schreibt man i++).

Die for-Schleife wird durchlaufen, so lange das sogenannte Abbruchkriterium noch nicht erfüllt ist. D.h. ein boole'scher Ausdruck wahr ist(z.B. i<=10).

for (Anfangswert der Zählervariable; Abbruchkriterium; Veränderung der Zählervariable)
{Anweisungen in der Schleife}

Beispiele:

```
int i;
for (i=1; i<=3; i++) {kara.move();} /* Kara geht drei Schritte vorwärts */
```

```
int i;     /* Definition einer Zählervariablen i*/
int s=0;   /* Definition der Summe s, Initialisierung mit dem Anfangswert Null. */
for (i=1; i<=10; i++) {s=s+i;}
```
/ Die Zählervariable beginnt bei Eins, Die Zählervariable muss kleiner gleich 10 sein (d.h. i läuft von 1 bis und mit 10). Die Zählervariable wird bei jedem Durchlauf um eines erhöht (i++). In den geschweiften Klammern stehen die Anweisungen, die mehrmals ausgeführt werden. Bei jedem Durchgang wird i zur aktuellen Summe hinzu addiert. Es werden also die Zahlen von 1 bis 10 aufsummiert. */*

while-Schleifen

Bei der while-Schleife wird eine Gruppe von Anweisungen so lange ausgeführt, bis die Bedingung nicht mehr erfüllt ist.Achtung: Bei der while-Schleife ist man selber dafür verantwortlich, dass man eine Zählervariable definiert (wie bei der for-Schleife) und diese bei jedem Durchlauf so verändert, dass die Schleife irgendwann abgebrochen wird

while (Bedingung) { Anweisungen } */* Solange die Bedingung zutrifft, werden die Anweisungen in den geschweiften Klammern ausgeführt. */*

Beispiele:

```
while (!kara.treeFront()) {kara.move();} /* Kara geht vorwärts, solange kein Baum vor ihm steht */
```

```
int i=1;
int s=0;
while (i <=10) {s=s+i; i++;} /* Solange die Bedingung i <=10 zutrifft, wird die Schleife weiterausgeführt. Nicht vergessen die Variable i innerhalb der Schleife zu erhöhen! Sonst gibt es eine Endlosschleife. Diese while-Schleife macht übrigens genau dasselbe wie das Beispiel der FOR-Schleife oben. */
```

if-Anweisung

Mit der if-Anweisung können verschiedene Fälle unterschieden werden. Falls die Bedinung erfüllt ist, werden die einen Befehle ausgeführt. Falls die Bedingung nicht erfüllt ist, werden andere Befehle ausgeführt.

Beispiele:

```
if (!(kara.mushroomFront()) {kara.move();}
 /* Falls Kara nicht vor einem Pilz steht, geht er ein Feld weiter.*/
```

```
if (i<10) {i++;} /* Falls i kleiner ist als 10, wird i um eins erhöht */
else {screen.DrawString("i ist grosser gleich 10", 10, 10);} /* sonst wird auf dem Bildschirm geschrieben, dass i grösser gleich 10 ist */
```

Wenn nach dem if oder nach dem else nur eine einzige Anweisung folgt, können die geschweiften Klammern weggelassen werden. Also kann man schreiben:

```
if (i<10) i++;
else screen.DrawString("i ist grosser gleich 10", 10, 10);
```

Allerdings birgt dies auch die Gefahr von Fehlern. Will man dann plötzlich eine zweite Anweisung abarbeiten lassen, muss man an die Klammern denken, sonst hat man einen Fehler im Programm, den man möglicherweise fast nicht mehr findet.

Methoden

Methoden sind Programmabschnitte, welche Teile eines gesamten Programms bilden. Methoden entsprechen den Prozeduren oder Funktionen in anderen Programmiersprachen.
Die Methoden werden von anderen Methoden aufgerufen und können selber wieder andere Methoden aufrufen.
Einer Methode werden **Parameter** von der aufrufenden Methode übergeben. Sie gibt einen Wert zurück. Der sogenannte **Rückgabewert** kann auch Nichts sein (void). Der Begriff public bedeutet, dass die Methode überall sichtbar ist.

Beispiel 1: `public void paint(Graphics screen) {…}`

Die Methode paint ist überall sichtbar (public), gibt keinen Rückgabewert (void) bekommt jedoch von ihrem Aufrufer als Parameter einen Bildschirmausschnitt (Graphics screen) zum Zeichnen.

Beispiel 2: `public int count(double a, double b, double c) {…}`

Die Methode count ist überall sichtbar (public), liefert einen Integerwert (den Zählwert) zurück, und übernimmt drei Doublewerte a, b, c von ihrer aufrufenden Methode.

Wenn Ihr mehr wissen wollt:

Java-Dokumentation auf Deutsch:
http://www.selfjava.de/

Die offizielle Java-Homepage bei SUN auf Englisch:
http://java.sun.com/ im Menü links "Java Tutorial" anklicken.

Notizen
