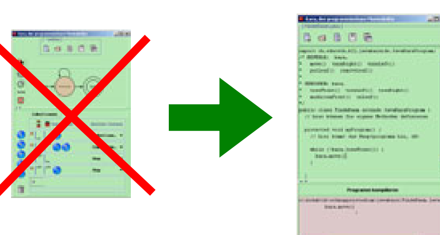
JavaKara

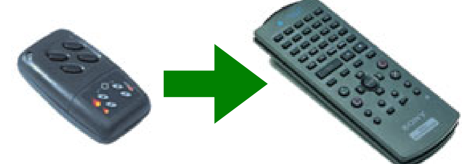
```
public class ZaehleKleeblaetter extends JavaKaraPro
{
  int zaehleK   blaetterBisBaum()
  {
    int klee  attZaehler = 0;
    while (!k          Front())

      kara.
      if (k
      {
        ka
        kl
      }
    }
    retu
  }

  public void
    int zaehl   = z       blaetterBisBaum();
    tools.show essa  ("Ich habe "+zaehler+
                      " Kleeblaetter gefunden.");
   }
}
```

Java? Kara

- **Anstatt endliche Automaten nun professionelle Programmiersprache Java**

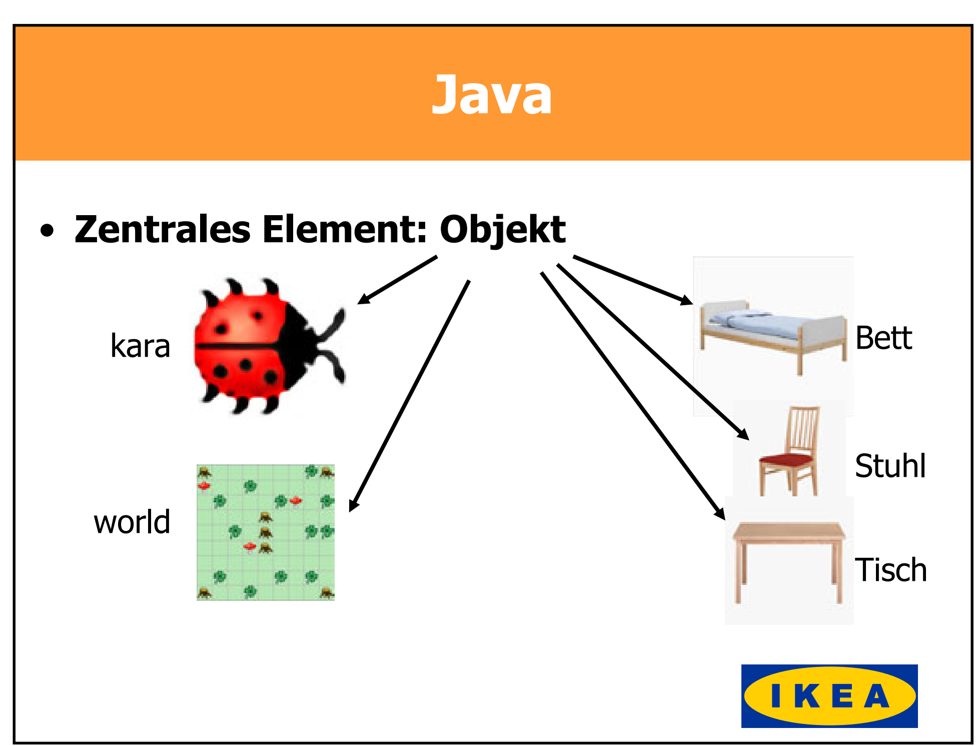


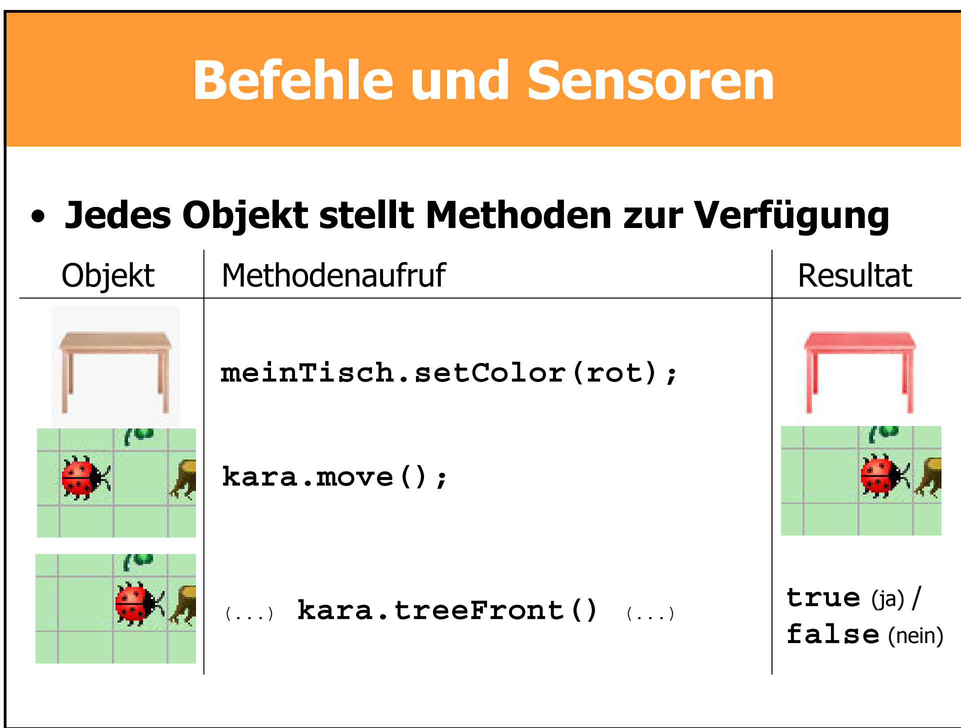
- **Professionell = viele tausend Befehle!**



Java

- **Zentrales Element: Objekt**





Was Kara alles kann

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**
kara.removeLeaf()	**Kara nimmt ein Kleeblatt auf**

Sensoren, um dieUmgebung zu überprüfen:

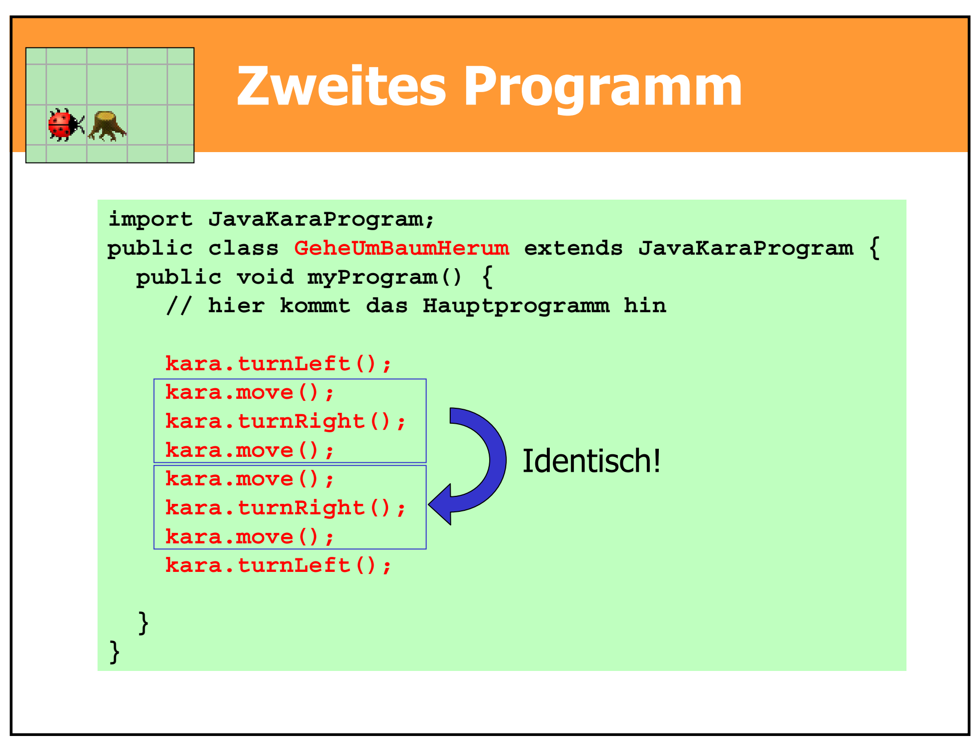
kara.treeFront()	**Gibt es einen Baum auf dem Feld vor Kara?**
kara.treeLeft()	**Gibt es einen Baum auf dem Feld links von Kara?**
kara.treeRight()	**Gibt es einen Baum auf dem Feld rechts von Kara?**
kara.mushroomFront()	**Gibt es einen Pilz auf dem Feld vor Kara?**
kara.onLeaf()	**Steht Kara auf einem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

Programme in JavaKara

- **Schablone ist vorgegeben**
- **Wir können also direkt loslegen!**

```java
import JavaKaraProgram;
public class SchrittVorwaerts extends JavaKaraProgram {
  public void myProgram() {
    // hier kommt das Hauptprogramm hin

    kara.move();

  }
}
```

Demo



Geschickter: Neue Methode

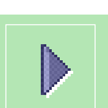
```
import JavaKaraProgram;
public class GeheUmBaumHerum extends JavaKaraProgram {

  void viertelDrehung() {
    kara.move();
    kara.turnRight();
    kara.move();
  }

  public void myProgram() {
    // hier kommt das Hauptprogramm hin

    kara.turnLeft();
    viertelDrehung();
    viertelDrehung();
    kara.turnLeft();

  }
}
```

Demo

Was haben wir gemacht ?

1. **Programm schreiben und speichern**

2. **Programm laufen lassen**



Unser Koch versteht nur Schwedisch

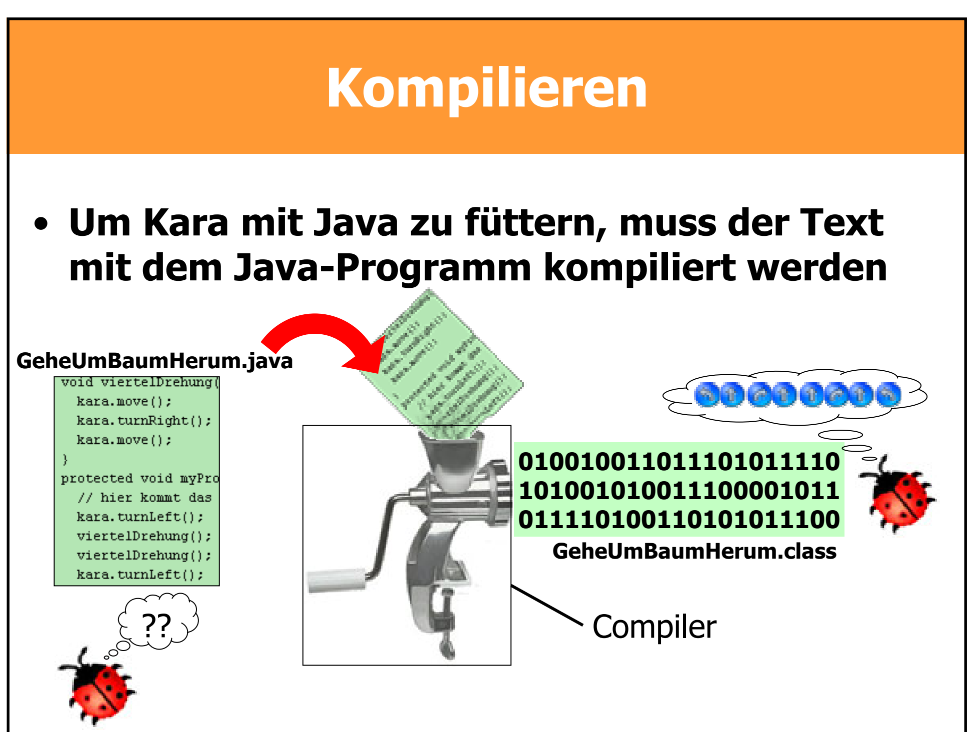
„Ägg i blå sås

Detta milt exotiska recept ingick i "Smekmånadens Kokbok", vilken Kooperativa Förbundet distribuerade till nygifta under 1930-talet. En verklig funkis-rätt! Var inte oroliga för metylenblått, den akuta giftigheten är låg."

Wir brauchen einen Übersetzer!
Bei Programmiersprachen heisst ein solcher Übersetzer Kompiler

Kompilieren

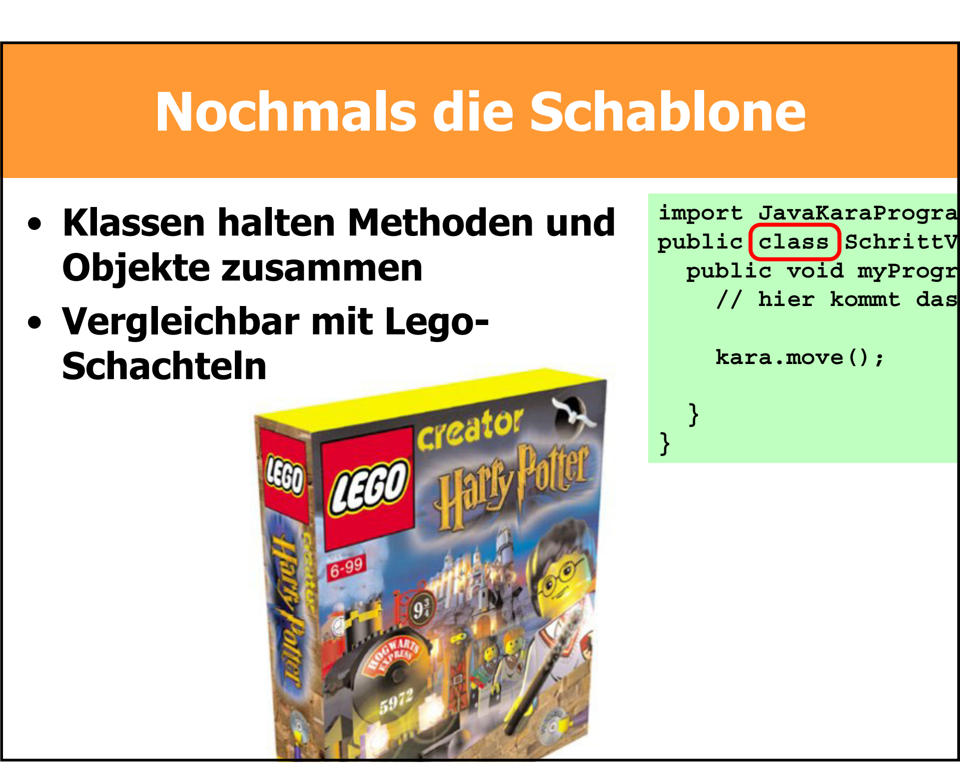
- **Um Kara mit Java zu füttern, muss der Text mit dem Java-Programm kompiliert werden**



Schritte bis sich Kara bewegt...

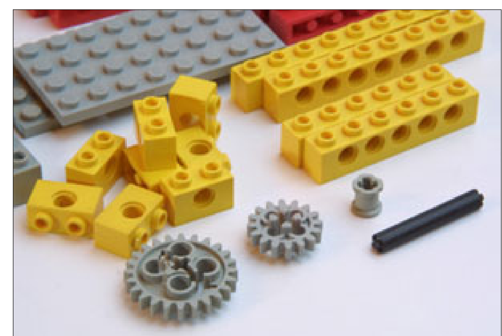
1. **Programm schreiben und speichern**
2. **Kompilieren**
3. **Programm ausführen**

Demo



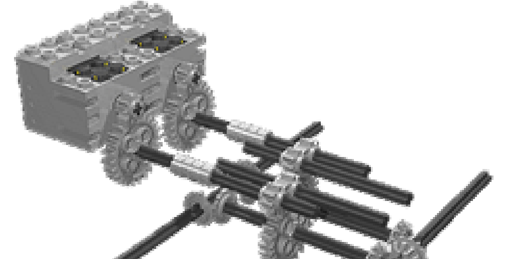
Lego - Schachteln ?

- **Lego-Bausteine sind Objekte**
- **Es gibt verschiedene Typen:**
  - **Grundsteine (verschiedene Grössen)**
  - **Zahnräder (kleine, mittlere, grosse)**
  - **Stangen**
  - **Usw.**
- **Von jedem Typ kann man beliebig viele Objekte haben**



Lego - Schachteln

- **Bedienungsanleitung**
- **Dort hat es eine Methode**
  `setzeHauptModellZusammen()`
- **Und vielleicht eine Methode**
  `erstelleKompliziertesGetriebe()`



Wir haben gesehen

- **Das Gerüst ist bei JavaKara fix vorgegeben**

```
import JavaKaraProgram;
public class ___________ extends JavaKaraProgram {
  public void myProgram() {
    // hier kommt das Hauptprogramm hin

    (...)

  }
}
```

- **Wir müssen nur noch „Befehle einfüllen"**

Befehle einfüllen

- **Die Befehle, die man aufruft, nennt man Methoden**
- **Die Befehle werden mit Strichpunkten abgeschlossen**

```
kara.move();
```

Geschweifte Klammern { }

- **Umklammern Programmblöcke**
- **Z.B. eine Methode oder ein Verzweigungs-Block**
- **Können auch verschachtelt angewendet werden -> Blöcke einrücken**

```
import JavaKaraProgram;
public class ___________ extends JavaKaraProgram {
  public void myProgram() {

      (...)

  }
}
```

Einrücken

import JavaKaraProgram; public class Slalom extends JavaKaraProgram { void eat() { if (kara.onLeaf()) { kara.removeLeaf(); } } void eat_right() { eat(); kara.move(); kara.turnRight(); } void eat_left() { eat(); kara.move(); kara.turnLeft(); } public void myProgram() { while (!kara.treeFront()) { eat_right(); eat_left(); } }

Einrücken

```
import JavaKaraProgram;

public class Slalom extends JavaKaraProgram {

  void eat() { if (kara.onLeaf()) { kara.removeLeaf(); } }

  void eat_right() { eat(); kara.move(); kara.turnRight(); }

  void eat_left() { eat(); kara.move(); kara.turnLeft(); }

  public void myProgram() { while (!kara.treeFront()) { eat_right(); eat_left(); } }

}
```

Bildschirm

Einrücken

```
import JavaKaraProgram;

public class Slalom extends
    JavaKaraProgram {

  void eat() {

    if (kara.onLeaf()) {
      kara.removeLeaf();
    }

  }

  void eat_right() {

    eat();
    kara.move();
    kara.turnRight();

  }
...
```

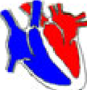
```
...

void eat_left() {

    eat();
    kara.move();
    kara.turnLeft();

 }

 public void myProgram() {

    while (!kara.treeFront()) {
      eat_right();
      eat_left();
    }
 }
}
```

Wir kennen nun schon

- **Boole`sche Ausdrücke**

- **Verzweigung (if)**
```
if (richtungBasel()) { einspuren(); }
```

- **Schleifen (while, for)**
```
while (lebeNoch()) { herzSchlag(); }
```



- **Methoden**

Herr Boole & Verzweigungen

- **Wichtig, wenn festgestellt werden muss, ob ein bestimmter Fall eingetreten ist.**
- **Mehrere Bedingungen können kombiniert werden**

```
if ( sonne.Scheint() && draussen.Warm() ) {
   marc.nimmLiegestuhHervor();
   marc.liegeAnDieSonne();
}
```

Objekt „marc"

„Solange-Bis“ Schleifen

- **while Schleife, wenn man etwas solange macht, bis etwas passiert**

```
while ( sonne.Scheint() ) {
  marc.bleibAufDemLiegestuhlLiegen();
}
```

„x-Mal“ Schleifen

- **for Schleife, bei einer Anzahl Schleifendurchgänge**

```
for (int i=1; i<17; i++) {
  marc.leseHarryPotterBand1Seite(i);
}
```

Methoden

- Methoden fassen mehrere Befehle zusammen
- Parameter erlauben es, Werte zu übergeben.
- Methoden können Werte zurückgeben

```
int erhoeheUmEins(int zahl) {
  int erhoehteZahl = zahl + 1;
  return erhoehteZahl;
}
```

Methoden

- Details: Mehrere Parameter werden durch Kommas getrennt
- Mehrere Werte zurückgeben: Geht nicht!

```
int erhoeheUm(int zahl, int wert) {
  int erhoehteZahl = zahl + wert;
  return erhoehteZahl;
}
```

Methoden

```
int erhoeheUmEins(int zahl) {
  int erhoehteZahl = zahl + 1;
  return erhoehteZahl;
}
```

- **Parameter werden in der Methode wie Variablen angeschaut**
- **Auf die Variablen des Hauptprogramms soll man nicht zugreifen**

Weitere Java-Elemente

- **Kommentare:**
  Die habt Ihr schon gesehen
- **2 Möglichkeiten:**

```
/* Das ist ein Kommentar.
   Der kann auch mehrere Zeilen umfassen. */

// Das ist auch ein Kommentar
// der geht aber nur bis zum Schluss der Zeile
```

Gross-/Kleinschreibung

- **Java unterscheidet zwischen Gross- und Kleinschreibung!**
  `Kara.Move();` **wird nicht akzeptiert**

Variablen

- **Schon kennengelernt bei for-Schleife**
- **Kennt Ihr von der Mathematik:**

$$3x+4 = 16$$
$$3x = 12$$
$$x = 4$$

- **Platzhalter für einen Wert**

Variablen

- **In Mathematik:**
  x kann ganze Zahl sein, oder reell, oder eine natürliche Zahl
- **In Java:**
  jede Variable hat einen Typ!
- **Variablen müssen definiert werden!**

Variablen definieren

```
int counter = 0;
```

Typ (int), Name (counter), Initialisierungswert (0)

```
int counter;
(...)
counter = 0;
```

Initialisierungswert kann auch weggelassen werden!

Verschiedene Typen

- **int: ganze Zahlen**
  `int counter = 2;`
- **double: Gleitkommazahl (rationale Zahlen)**
  `double pi = 3.14159;`
- **String: Zeichenketten (Achtung: grosses S)**
  `String bewertung = "Java ist cool!";`
- **boolean: wahr oder nicht wahr**
  `boolean schoenesWetter = true;`

Typen in der freien Wildbahn!

```
int erhoeheUmEins(int zahl) {
  int erhoehteZahl = zahl + 1;
  return erhoehteZahl;
}

void viertelDrehung() {
  kara.move();
  kara.turnRight();
  kara.move();
}

boolean warm() {
  if (draussen.temperatur() > 20) {
    return true;
  }
  return false;
}
```

Konstanten

- **Gibt's in der Mathematik auch:**
  Zahl π = 3.14159265358...
  Zahl e = 2.71828182845...

- **Wert wird vor dem Start zugewiesen – und nicht mehr verändert**

- **Auch Konstanten müssen definiert werden**

Konstanten definieren

- **Konvention: Für den Konstantennamen GROSSBUCHSTABEN verwenden**
- **Vor dem Typ muss `final` stehen**

```
final int FENSTERBREITE = 800;
```

Ablauf von JavaKara Programmen

- **Start:** `myProgram()` **wird gesucht**
- **Abarbeiten der Befehle in** `myProgram()`
- **Programm ist fertig, wenn die letzte Anweisung von** `myProgram()` **abgearbeitet wurde**

```
import JavaKaraProgram;
public class GeheUmBaumHerum

  void viertelDrehung() {
    kara.move();
    kara.turnRight();
    kara.move();
  }

  public void myProgram() {
    kara.turnLeft();
    viertelDrehung();
    viertelDrehung();
    kara.turnLeft();
  }
}
```

public – jetzt wird's kompliziert

- `public` **heisst "von aussen sichtbar"**
- **Brauchts, damit das JavaKara-Programm die Klasse** `GeheUmBaumHerum` **und darin das Program** `myProgram()` **"sieht"**

```
import JavaKaraProgram;
public class GeheUmBaumH

  void viertelDrehung()
    kara.move();
    kara.turnRight();
    kara.move();
  }

  public void myProgram(
    kara.turnLeft();
    viertelDrehung();
    viertelDrehung();
    kara.turnLeft();
  }
}
```

Beispiel Restaurant

- "Public"
  - `entrée(KLEINER_SALAT);`
  - `hauptgang(STEAK, MEDIUM);`
  - `rechnung(KREDITKARTE_VISA);`
- Nicht Public
  - `wascheSalat(GRUENDLICH);`
  - `schäleKartoffeln(FLOTT);`
  - `aufschlag("10%");`

public – jetzt wird's kompliziert

- **viertelDrehung() muss nicht public sein**
- **Wird ja auch nicht vom JavaKara Programm direkt aufgerufen**

```
import JavaKaraProgram;
public class GeheUmBaumH

  void viertelDrehung()
    kara.move();
    kara.turnRight();
    kara.move();
  }

  public void myProgram(
    kara.turnLeft();
    viertelDrehung();
    viertelDrehung();
    kara.turnLeft();
  }
}
```

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.

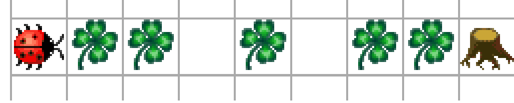
Notizen

Verzweigungs-Expertin

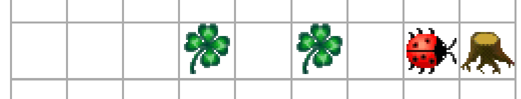
Aufgabe

Vor Kara hat es Felder, die entweder leer oder mit einem Kleeblatt belegt sind. Diese „Spur“ endet vor einem Baum. Kara soll alle Kleeblätter aufheben und bei allen Lücken ein Kleeblatt legen.

Situation vor dem Start des Programms



Situation nachher



Das Neue

Kara muss 2 Fälle unterscheiden:

- Ist ein Kleeblatt unterhalb von Kara, so muss er dieses aufheben.
- Ist noch kein Kleeblatt unter Kara, dann legt er ein Kleeblatt hin.

Lösung in JavaKara

```
import JavaKaraProgram;
public class InvertiereKleeblaetter extends JavaKaraProgram
{ // Anfang von InvertiereKleeblaetter
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      kara.move();
      if (kara.onLeaf())
      {
        kara.removeLeaf();
      }
      else
      {
        kara.putLeaf();
      }
    }
  } // Ende von myProgramm
} // Ende von InvertiereKleeblaetter
```

Bemerkung:

Die Zeile `while (!kara.treeFront())` heisst soviel wie „solange kein Baum vor Kara ist, mache folgendes:“

Erläuterungen

1. **Verzweigungen: grundlegende Ablaufstruktur beim Programmieren**
   Das Prinzip von Verzweigungen gibt es nicht nur bei Kara, sondern überall wo programmiert wird. Stell dir zum Beispiel eine Maschine vor, die abschalten muss, wenn das Kühlsystem defekt ist. Oder einen Getränkeautomaten, der anders reagieren soll, wenn man ihn mit thailändischen 10-Baht-Münzstücken füttert anstelle der gleich schweren und gleich grossen 2-Euro-Münzen. Oder bei einem Kalender: Je nach Monat hat es eine unterschiedliche Anzahl Tage.

2. **Die Syntax einer Verzweigung**

   `if (Bedingung) { Anweisungen1 } else { Anweisungen2 }`

   oder

   `if (Bedingung) { Anweisungen }`

   Beim zweiten Fall werden die Anweisungen nur abgearbeitet, wenn die Bedingung zutrifft. Wenn die Bedingung nicht zutrifft, wird das Programm einfach nach den geschweiften Klammern fortgesetzt.

   Beispiel:
   ```
   if (kara.onLeaf()) { kara.removeLeaf(); }
   ```

3. **Mögliche Darstellungen**
   Es ist nicht übersichtlich, mehrere Anweisungen auf eine Zeile zu schreiben. Deshalb macht es Sinn, die geschweiften Klammern auf eine neue Zeile zu setzen und die Befehle für die Fallunterscheidung einzurücken.

   ```
   if (Bedingung)
   {
     Anweisung1;
     Anweisung2;
     Anweisung3;
   }
   ```

   Man nennt das `if` mit der Bedingung und allen Anweisungen (inklusive des `else` Teils) einen `if`-Block.

4. **Geschachtelte `if`-Anweisungen**
   In den Anweisungen eines `if`-Blockes können natürlich weitere `if`-Anweisungen stehen.

   ```
   if (Bedingung1)
   {
     if (Bedingung2)
     {
       Anweisung1;
       Anweisung2;
     }
   ```

```
  else
  {
    Anweisung3;
  }
}
else
{
  Anweisung4;
  Anweisung5;
}
```

Beachte, dass Einrückungen den Programmablauf nicht beeinflussen, aber die Lesbarkeit wesentlich verbessern.

Fragen

1. Betrachten wir untenstehendes Programm

```
(...)
while (!kara.treeFront())
{
  if (kara.onLeaf())
  {
    kara.removeLeaf();
  }
  else
  {
    kara.putLeaf();
  }
  kara.move();
}
(...)
```

   Im Gegensatz zur Lösung 1 wurde die Anweisung `kara.move();` hinter den `if`-Block gestellt. Ändert sich dadurch die Funktionalität des Programmes?

2. Das Programm auf der ersten Seite arbeitet nicht in allen Situationen korrekt.
   a. Versuche den Spezialfall zu finden, bei welchem das Programm nicht richtig arbeitet.
   b. Entwirf auf Papier eine verbesserte Version des Programms, die auch den Spezialfall berücksichtigt.
   c. Teste am Computer, ob deine Lösung funktioniert.

   Hinweis: Teste dein Programm für Spezialfälle: Kara steht zu Beginn auf einem Kleeblatt, oder vor dem Baum liegt ein Kleeblatt, etc.

3. Kara steht in einer Welt ohne Bäume. Er soll nun einen Schritt nach vorne machen. Liegt dort ein Kleeblatt, so soll Kara es aufnehmen, sich nach links drehen und einen Schritt nach vorne machen. Ist auf dem Feld kein Kleeblatt, so soll Kara ein Kleeblatt hinlegen, sich nach rechts drehen und einen Schritt nach vorne machen.

   Schreibe ein JavaKara Programm, das obige Aufgabe erfüllt und teste es anschliessend

am Computer.

4. Wozu könnte folgendes Programmschema gut sein?
   Hinweis: Stell dir vor, du musst dir die Ampelfarbe einer Lichtsignalanlage anschauen.

```
(...)
if (Bedingung1) { Anweisungen1 }
else {
  if (Bedingung2) { Anweisungen2 }
  else {
    if (Bedingung3) { Anweisungen3 }
    else { Anweisungen4 }
  }
(...)
```

5. *Knifflige Zusatzaufgabe:*

   Kara soll folgendes tun, wenn er auf einem Kleeblatt steht:
   - Zuerst soll er das Kleeblatt aufnehmen.
   - Ist vor ihm ein Baum, dann soll Kara eine Drehung um 180° machen.
   - Ist vorne kein Baum, dann soll Kara ein Feld nach vorne schreiten.

   Steht Kara auf keinem Kleeblatt, soll er nichts tun.

   Obige Aufgabe wurde bereits programmiert. Leider befindet sich ein Fehler im Programm.

```
 (...)
if (kara.onLeaf()) {
  // Kara ist auf Kleeblatt: Also aufnehmen
  kara.removeLeaf();
  if (kara.treeFront()) {
    // Kara steht vor Baum, drehe um
    kara.turnRight();
    kara.turnRight(); }
  }
  else {
    // Kara steht nicht vor einem Baum
    kara.move();
}
// und sonst mache gar nichts
(...)
```

   Finde den Fehler und verbessere das Programm.

Expertin für Boole'sche Ausdrücke

Aufgabe 1

Kara hat geradeaus vor sich einen Baum. Er möchte bis zum Baum entlang laufen und auf allen Feldern Kleeblätter deponieren.

Situation vor dem Start des Programms

Situation nachher

Das Neue

Wir brauchen eine Methode `kara.notOnLeaf()`, damit wir feststellen können, dass sich kein Kleeblatt unterhalb von Kara befindet. Diese Methode gibt es zwar nicht, aber wir haben eine Methode `kara.onLeaf()`, die uns das Gegenteil zurück gibt. Der not-Operator `!` von Java dient dazu, eine Aussage umzudrehen. Er kehrt einen wahren Wert um in einen falschen und umgekehrt.

Lösung in JavaKara

```
import JavaKaraProgram;
public class LegeKleeblaetterAufWeg extends JavaKaraProgram
{ // Anfang von LegeKleeblaetterAufWeg
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      kara.move();
      if (!kara.onLeaf())
      {
        kara.putLeaf();
      }
    }
  } // Ende von myProgramm
} // Ende von LegeKleeblaetterAufWeg
```

Bemerkung

Die Anweisung `if (!Ausdruck)` bedeutet: „wenn Ausdruck *nicht wahr* ist, dann mache folgendes:"

Puzzle: Expertin B

Erläuterungen

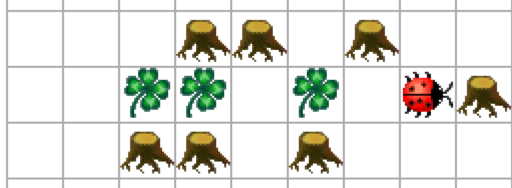
Die Methode `kara.onLeaf()` gibt einen Wahrheitswert zurück, der entweder `true` oder `false`, also wahr oder falsch, sein kann. Der `!` Operator ändert im Prinzip nur das „Vorzeichen“: Aus `true` wird `false` und umgekehrt.

Aufgabe 2

Kara soll nun nur ein Kleeblatt legen, wenn rechts ein Baum steht.

Situation vor dem Start des Programms

Situation nachher



Das Neue

Um ein Kleeblatt zu legen, müssen nun zwei Bedingungen erfüllt sein. Immer noch soll kein Kleeblatt unterhalb von Kara liegen, zusätzlich muss aber noch rechts ein Baum stehen.

Lösung in JavaKara

```
import JavaKaraProgram;
public class LegeKleeblaetterAufWeg2 extends JavaKaraProgram
{ // Anfang von LegeKleeblaetterAufWeg2
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      kara.move();
      if (!kara.onLeaf() && kara.treeRight())
      {
        kara.putLeaf();
      }
    }
  } // Ende von myProgramm
} // Ende von LegeKleeblaetterAufWeg2
```

Bemerkungen

Die `&&` Operation dient dazu, zwei Bedingungen mit einem „und“ zu verknüpfen. Sowohl `!kara.onLeaf()` wie auch `kara.treeRight()` liefern `true` oder `false` zurück. Die `&&`

Puzzle: Expertin B

Operation verknüpft die beiden Bedingungen und gibt nur `true` zurück, wenn beide Bedingungen `true` sind. Alle anderen Fälle ergeben `false`.
Diese Operation wird im Fachjargon auch „logisches Und" genannt.

Folgendes Schema zeigt die Funktionsweise von `&&`. Wir werden ihm später wieder begegnen.

`false && false`	`false`
`false && true`	`false`
`true && false`	`false`
`true && true`	`true`

Aufgabe 3

Kara soll nun ein Kleeblatt legen, wenn links oder rechts ein Baum steht. Achtung, das bedeutet, dass auch ein Kleeblatt gelegt werden muss, falls auf beiden Seiten ein Baum steht.

Situation vor dem Start des Programms

Situation nachher

Das Neue

Um ein Kleeblatt zu legen, muss nun zusätzlich zur einen Kleeblatt-Bedingung *mindestens eine* der beiden Baum-Bedingungen `treeLeft()` und `treeRight()` zutreffen.

Lösung in JavaKara

```
import JavaKaraProgram;
public class LegeKleeblaetterAufWeg3 extends JavaKaraProgram
{ // Anfang von LegeKleeblaetterAufWeg3
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      kara.move();
      if ( !kara.onLeaf() &&
           ( kara.treeLeft() || kara.treeRight() ) )
      {
        kara.putLeaf();
      }
    }
```

```
  } // Ende von myProgramm
} // Ende von LegeKleeblaetterAufWeg3
```

Erläuterungen

Neu kommt ein weiterer Operator hinzu. Der `||` Operator dient dazu, zwei Bedingungen mit einem „oder“ zu verknüpfen. `||` gibt `true` zurück wenn mindestens eine der beiden Bedingungen (`kara.treeLeft()`, `kara.treeRight()`) wahr ist.
Diese Operation wird im Fachjargon „logisches Oder“ genannt.

Folgendes Schema zeigt die Funktionsweise von `||`.

false || false	false
false || true	true
true || false	true
true || true	true

Bei der Verknüpfung von mehreren Bedingungen spricht man von einem Boole’schen Ausdruck. Benannt sind die Ausdrücke nach dem englischen Mathematiker George Boole (1815-1864). Seine Theorie bildet einen Grundstein der Informatik, ist aber auch in der Mathematik von grosser Bedeutung.

Du hast gesehen, dass es teilweise notwendig ist, Boole’sche Ausdrücke mit Klammern zu ordnen. Zähle deshalb beim Schreiben von Java Programmen wenn nötig die Klammern.

Fragen

1. Wie müsstest du das dritte Programm ändern, wenn Kara nur Kleeblätter legen dürfte, wenn sich links *und* rechts ein Baumstamm befindet? Probiere deine Lösung auf dem Computer aus!

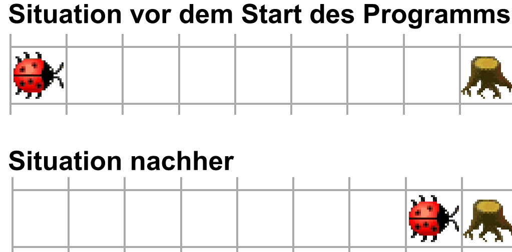
2. Das dritte Programm soll erweitert werden. Nun soll ein Kleeblatt nur gelegt werden, wenn sich entweder nur links ein Baum oder nur rechts ein Baum befindet. Wenn sich links und rechts ein Baum befindet, dann soll nichts gemacht werden.
   Probiere deine Lösung wiederum am Computer aus!
   Hinweis: Die Formulierung des Boole’schen Ausdruckes ist nicht ganz leicht. Versuche die Lösung aus Aufgabe 3 mit deiner Lösung zur vorhergehenden Frage zu kombinieren.

Schleifen-Expertin

Aufgabe

Kara soll geradeaus laufen, bis er vor einem Baum steht.

Situation vor dem Start des Programms



Situation nachher

Das Neue

Mit einer `while` Schleife können Befehle oder Befehlsblöcke wiederholt werden, solange eine Bedingung erfüllt ist.

Lösung in JavaKara

```
import JavaKaraProgram;
public class FindeBaum extends JavaKaraProgram
{ // Anfang von FindeBaum
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      kara.move();
    }
  } // Ende von myProgramm
} // Ende von FindeBaum
```

Bemerkung

Das Ausrufezeichen bei `!kara.treeFront()` ist der not-Operator. Er kehrt die Aussage von `kara.treeFront()` um. Also wenn ein Baum vor Kara ist, wird der Ausdruck *falsch* anstatt *wahr*. Entsprechend, wenn Kara nicht vor einem Baum steht, *wahr* anstelle von *falsch*.

Erläuterungen

1. **Schleifen: grundlegende Ablaufstruktur beim Programmieren**
   Der Computer eignet sich zum wiederholten Ausführen von Instruktionen. Durch diese Schleifen im Programmablauf kann man viele Aufgaben automatisieren. Roboter können zum Beispiel ein Auto zusammenbauen oder andere „Fliessbandarbeiten“ übernehmen.

2. **Syntax einer `while` Schleife**

   `while (`Schleifenbedingung`)`

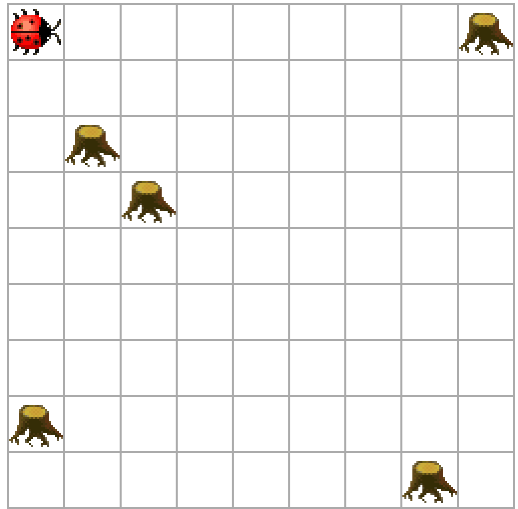
```
{
   Anweisung1
   Anweisung2
   Anweisung3
}
```

Solange die Bedingung erfüllt ist, werden die verschiedenen Anweisungen der Reihe nach abgearbeitet. Dabei wird stets *vor* dem Abarbeiten der ersten Anweisung geprüft, ob der Anweisungsblock überhaupt noch ausgeführt werden muss.

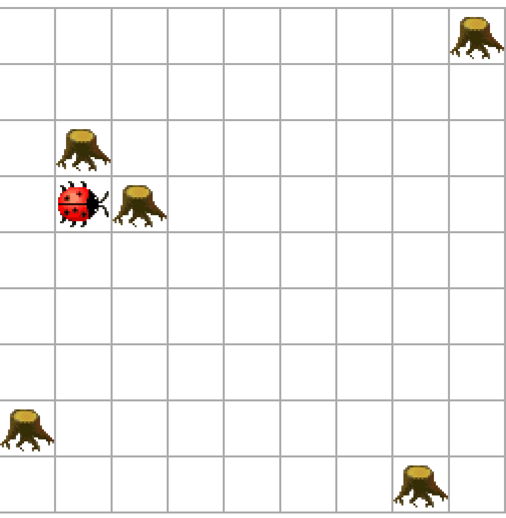
Aufgabe

Nun soll Kara beim Erreichen des Baumstammes eine Rechtsdrehung machen und Weiterlaufen bis zum nächsten Baumstamm. Das soll er so lange machen, bis er nach der Rechtsdrehung unmittelbar wieder vor einem Baum steht.

Situation vor dem Start des Programms



Situation nachher



Das Neue

`while` Schleifen müssen teilweise auch verschachtelt angewandt werden.

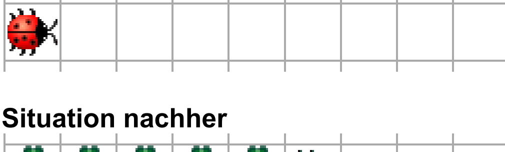
Lösung in JavaKara

```
import JavaKaraProgram;
public class ImKreisHerum extends JavaKaraProgram
{ // Anfang von ImKreisHerum
  public void myProgram()
  { // Anfang von myProgramm
    while (!kara.treeFront())
    {
      while (!kara.treeFront())
      {
        kara.move();
      }
      kara.turnRight();
    }
  } // Ende von myProgramm
} // Ende von ImKreisHerum
```

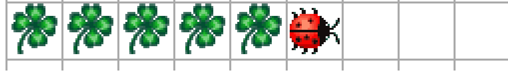
Aufgabe

Kara möchte 5 Kleeblätter in die leere Welt setzen, die alle hintereinander angeordnet sein sollen.

Situation vor dem Start des Programms



Situation nachher



Das Neue

Mit einer `for` Schleife kann ein Block von Anweisungen eine bestimmte Anzahl mal durchlaufen werden.

Lösung in JavaKara

```
import JavaKaraProgram;
public class FuenfKleeblaetter extends JavaKaraProgram
{ // Anfang von FuenfKleeblaetter
  public void myProgram()
  { // Anfang von myProgramm
    for (int i=1; i<=5; i++)
    {
      kara.putLeaf();
      kara.move();
    }
  } // Ende von myProgramm
} // Ende von FuenfKleeblaetter
```

Erläuterungen

1. **Die Syntax einer `for` Schleife:**

```
for (Initialisierung; Bedingung; Aktualisierung)
{
  Anweisung1
  Anweisung2
  Anweisung3
}
```

Die Initialisierungs-Anweisung (im obigen Beispiel `int i=1`) wird vor Beginn der Schleife einmal ausgeführt. Sie wird oft dazu verwendet, Zählervariablen auf Anfangswerte zu setzen.
Die Bedingung (im obigen Beispiel `i<=5`) wird genau wie bei der `while` Schleife vor der ersten Anweisung getestet. Trifft sie nicht (mehr) zu, dann wird die Schleifen-ausführung gestoppt und mit dem restlichen Programmablauf fortgefahren.
Am Ende eines Schleifendurchlaufes wird – bevor die Bedingung neu getestet wird – die Aktualisierungs-Anweisung (im obigen Beispiel `i++`) ausgeführt. Meistens wird

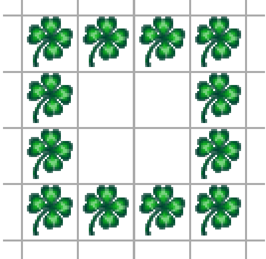
sie dazu verwendet, um die Zählvariable zu erhöhen.

2. Den Ausdruck `int` braucht es in der Initialisierung, damit Java weiss, dass unsere Zählvariable `i` eine ganze Zahl darstellt.

3. Die Aktualisierungs-Anweisung `i++` erhöht die Zählvariable um eins. Man hätte auch `i=i+1` schreiben können. Entsprechend kann man auch `i--` anstelle von `i=i-1` schreiben.

Fragen

1. Kara möchte ein Rechteck zeichnen, indem er den Rand einer 4x4 Felder grossen Fläche mit Kleeblättern markiert. Überlege dir, ob sich wieder eine `while` oder eine `for` Schleife eignet und schreibe das entsprechende Programm. Teste es anschliessend am Computer.



2. Neben der Methode `kara.putLeaf()` gibt es bei JavaKara auch die Methode `world.setLeaf(int x, int y, boolean putLeaf)`. Der Vorteil dabei ist, dass wir Kara gar nicht mehr brauchen und trotzdem Kleeblätter in der Welt legen können. Das Beispiel `world.setLeaf(0,2,true)` setzt ein Kleeblatt an die Koordinate (0,2).
   Fülle mit dieser Methode eine Fläche von 5x5 Feldern in der Kara-Welt.
   Zusatzaufgabe: Versuche herauszufinden, was das `true`/`false` bei der Methode `world.setLeaf` bedeutet. Benutze dazu deine Kursunterlagen.

3. Kara soll ebenfalls eine 5x5 Felder grosse Fläche mit Kleeblätter bedecken. Entwerfe die Programme zuerst auf Papier, teste anschliessend am Computer.
   a. Lass deinen Kara immer 5 Kleeblätter horizontal legen und anschliessend die 5 Felder zurück laufen, bevor er sich auf die nächste Zeile wagt.
   b. Nun kann Kara den Weg noch optimieren. Er kann bereits beim zurück spazieren wieder 5 Kleeblätter auf der nächsten Zeile legen.
   Hinweis: Diese Teilaufgabe ist einiges schwieriger als die Teilaufgabe a.

Methoden-Expertin

Aufgabe 1

Kara steht vor einem Baum, der alleine in der Welt steht. Hinter dem Baum hat es ein Kleeblatt, das Kara aufheben soll. Danach soll Kara wieder zum Ausgangsort zurückkehren.

Situation vor dem Start des Programms

Situation nachher

Das Neue

Kara kann gewisse Abläufe lernen und unter einem neuen Kommando speichern. Diese Kommandos werden in Java Methoden genannt. Kara lernt in dieser Aufgabe, was er bei `geheUmBaumHerum()` und bei `dreheUm180Grad()` machen muss.

Lösung in JavaKara

```
import JavaKaraProgram;
public class GeheUmBaumHerumUndNimmKleeblattAuf extends
JavaKaraProgram
{ // Anfang von GeheUmBaumHerumUndNimmKleeblattAuf
  void geheUmBaumHerum() // Methodenkopf
  {
    kara.turnLeft();
    kara.move();
    kara.turnRight();
    kara.move();
    kara.move();
    kara.turnRight();
    kara.move();
    kara.turnLeft();
  }

  void dreheUm180Grad() // Methodenkopf
  {
    kara.turnRight();
    kara.turnRight();
  }
```

Puzzle: Expertin D

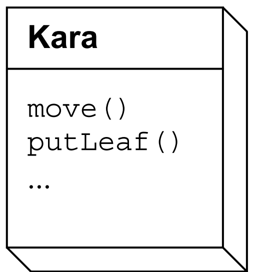
```
  public void myProgram()
  { // Anfang von myProgramm
    this.geheUmBaumHerum(); // Methodenaufruf
    kara.removeLeaf();
    this.dreheUm180Grad();  // Methodenaufruf
    this.geheUmBaumHerum(); // Methodenaufruf
    this.dreheUm180Grad();  // Methodenaufruf
  } // Ende von myProgramm
} // Ende von GeheUmBaumHerumUndNimmKleeblattAuf
```

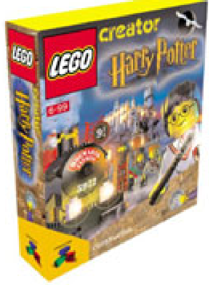
Erläuterungen

1. **Der Aufruf der Methode**
   Mit `this.geheUmBaumHerum();` wird Kara mitgeteilt, dass er die Methode mit dem Namen geheUmBaumHerum ausführen soll. Das `this` vor dem Methodennamen sagt Kara, dass sich die Methode in der aktuellen Klasse befindet. Dies ist aber nicht unbedingt nötig. Möglich wäre auch:

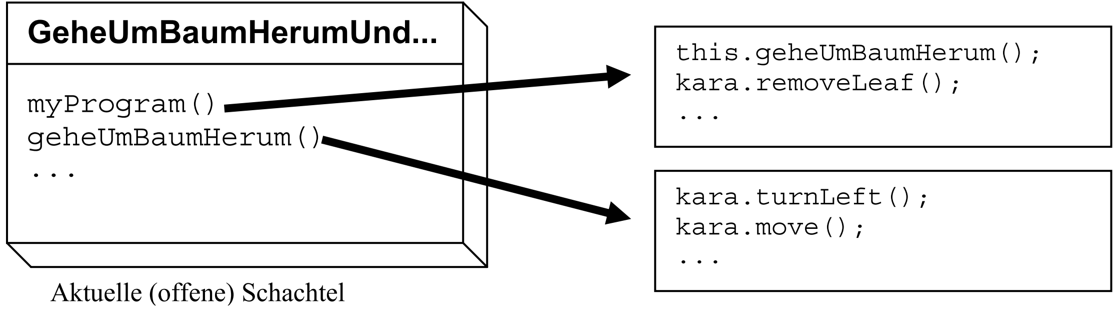
```
(...)
geheUmBaumHerum();
(...)
```

   Uff, das tönt alles ein bisschen abstrakt! Du musst dir das einfach so vorstellen:





Verschlossene Schachtel



Aktuelle (offene) Schachtel

   Die Anleitung `geheUmBaumHerum()` steckt nicht in der Kara-Schachtel, sondern in der aktuellen Schachtel. Erst die Anleitung ruft einen Befehl aus der Kara-Schachtel auf.

2. **Das leere Klammerpaar**
   Das Klammerpaar am Schluss des Methodenaufrufs bzw. des Methodenkopfs bedeutet, dass man keine Parameter übergeben möchte. Mit Hilfe von Parametern könnte man dem Kara z.B. mitteilen, wie viele Bäume hintereinander stehen, um die er herumgehen soll. Doch dazu kommen wir später wieder.

3. **Die Methode**
   Beim Methodenkopf wird dem Namen das Schlüsselwort `void` vorangestellt. Dieses Schlüsselwort gibt an, was für einen Wert wir an das Hauptprogramm zurückgeben wollen. Void bedeutet im Englischen „leer“, wir möchten also keinen Wert zurückgeben.
   Aus dem Mathematik-Unterricht kennen wir aber viele Beispiele, wo wir einen Wert zurückbekommen: Der Methode „Sinus“ übergeben wir einen Wert (z.B. $\pi/2$) und wir bekommen die Zahl 1 zurück. Auch darauf werden wir erst später eingehen.

Aufgabe 2

Nun soll Kara Rechtecke beliebiger Grösse zeichnen können. Dafür wäre es nützlich, eine Methode zu haben, mit der Kara eine Seite zeichnen kann, d.h. mehrere Kleeblätter hintereinander legen kann.

Das Neue

In einer neuen Methode `legeKleeblaetter` soll Kara die Kleeblätter legen. Die Anzahl der zu legenden Blätter wird mit einem Parameter übergeben.

Lösung in JavaKara

```
import JavaKaraProgram;
public class ZeichneRechteck extends JavaKaraProgram
{ // Anfang von ZeichneRechteck
  void legeKleeblaetter(int anzahl)
  {
    for (int i=1; i<=anzahl; i++)
    {
      kara.putLeaf();
      kara.move();
    }
  }
  public void myProgram()
  { // Anfang von myProgramm
    // 6x4 Rechteck zeichnen...
    legeKleeblaetter(5); // 5 Kleeblätter legen
    kara.turnRight();
    legeKleeblaetter(3); // 3 Kleeblätter legen
    kara.turnRight();
    legeKleeblaetter(5); // 5 Kleeblätter legen
    kara.turnRight();
    legeKleeblaetter(3); // 3 Kleeblätter legen
  } // Ende von myProgramm
} // Ende von ZeichneRechteck
```

Bemerkung

Die Anweisung `for (int i=1; i<=anzahl; i++)` sorgt dafür, dass die beiden Anweisungen `kara.putLeaf()` und `kara.move()` nicht nur einmal, sondern *anzahl*-mal ausgeführt werden.

Puzzle: Expertin D

Erläuterungen

Bis jetzt sind immer nur Methoden vorgekommen, denen keine Parameter übergeben worden sind. Das waren nur Spezialfälle, darum waren die Klammern auch immer leer. Es können aber auch mehrere Parameter übergeben werden, die durch Kommas voneinander getrennt werden:

```
void zeichneRechteck(int breite, int hoehe)
```

Für jeden Parameter, den man einer Methode „mitgibt", muss man seinen Datentyp angeben. Für die Anzahl der Kleeblätter haben wir den Typ `int` angegeben, der für integer steht. Integer sind ganze Zahlen. Wir werden in dieser Woche aber auch noch anderen Datentypen begegnen:

`double`: Das sind Fliesskommazahlen, wie z.B. 54013.35 oder 3.14159265358979

`String`: Dort können Zeichenketten gespeichert werden, also z.B. „Kara ist super!"

Weitere Typen sind `boolean`, `Graphics` und noch viele andere mehr. Ihnen werden wir teilweise später wieder begegnen.

Aufgabe 3

Kara soll alle Kleeblätter bis zum nächsten Baum einsammeln. Kara möchte die Kleeblätter zählen und danach anzeigen, wie viele er aufgenommen hat.

Das Neue

Eine Methode `zaehleKleeblaetterBisBaum` gibt das Ergebnis des Zählvorgangs zurück.

Lösung in JavaKara

```
import JavaKaraProgram;
public class ZaehleKleeblaetter extends JavaKaraProgram
{ // Anfang von ZaehleKleeblaettter
  int zaehleKleeblaetterBisBaum()
  {
    int kleeblattZaehler = 0;
    while (!kara.treeFront())
    {
      kara.move();
      if (kara.onLeaf())
      {
        kara.removeLeaf();
        kleeblattZaehler++;
      }
    }
    return kleeblattZaehler;  // Wert zurueckgeben
  }

  public void myProgram()
  { // Anfang von myProgram
    int zaehler = zaehleKleeblaetterBisBaum();
    tools.showMessage("Ich habe "+zaehler+
                      " Kleeblaetter gefunden.");
  } // Ende von myProgram
} // Ende von ZaehleKleeblaettter
```

Bemerkung

Die Anweisung `while (!kara.treeFront())` sorgt dafür, dass Kara die folgenden Anweisungen solange ausführt, bis er vor einem Baum steht.

Erläuterungen

Bisher mussten wir immer `void` vor unsere Methoden stellen. Jetzt sehen wir, was dies genau bedeutet. Für jede Methode gibt man an, was für einen Typ der Rückgabewert hat. Hat eine Methode keinen Rückgabewert, so schreibt man `void`. Diese Methoden sind also eigentlich Spezialfälle.
Die Rückgabe eines Wertes wird mit dem Aufruf von `return kleeblattZaehler;` gemacht.

Mit `tools.showMessage` kann ein Text (vom Typ `String`) ausgegeben werden. Mit `+` werden dabei verschiedene Texte aneinandergehängt.

Fragen

1. **Rechteck-Methode**
   Kreiere eine Methode `void zeichneRechteck(int breite, int hoehe)`, die mit Kleeblättern ein Rechteck zeichnet. Zeichne damit ein Rechteck der Grösse 4x3.
   Hinweis: Benutze das JavaKara Programm von Aufgabe 2 als Hilfe.
   Teste das Programm am Computer.

2. Was passiert, wenn man beim Rechteck eine Grösse von 1x3 zeichnet? Wie könnte man das Problem beseitigen?
   Probiere deine Lösung am Computer aus!

3. *Zusatzaufgabe:* Wir möchten nun die Aufgabe 3 so verändern, dass Kara nicht bis zum nächsten Baum läuft, sondern nur eine bestimmte Anzahl Schritte macht. Dabei soll er wie gehabt die aufgenommenen Kleeblätter zählen. Der Methodenkopf soll also wie folgt aussehen:
   `int zaehleKleeblaetter(int anzahlSchritte)`
   Entwirf die Methode auf Papier und teste sie anschliessend am Computer.