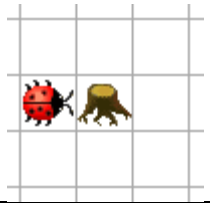
Grundlegende Programmierkonzepte: Anweisungen und Methoden

Einleitung: Eigene Befehle definieren

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**
	

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.

In JavaKara sieht das Programm wie folgt aus:

```
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();
  }

  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
```

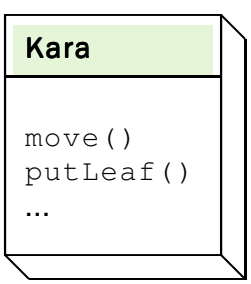
Erläuterungen

1. Der Aufruf der Methode

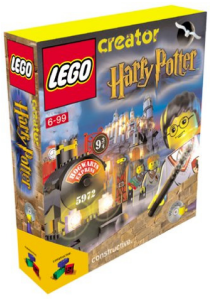
Mit `this.geheUmBaumHerum();` wird Kara mitgeteilt, dass er die Methode mit dem Namen geheUmBaumHerum ausführen soll. Das `this` vor dem Methodennamen sagt Kara, dass sich die Methode in der aktuellen Klasse befindet. Dies ist aber nicht unbedingt nötig. Möglich wäre auch:

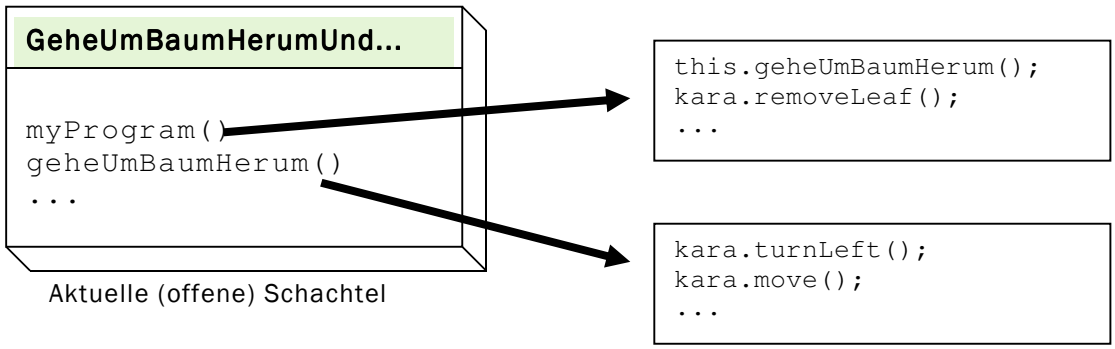
```
geheUmBaumHerum();
```

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



Verschlossene





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.

Beispiel: Eigene Methode mit Parameter definieren

Betrachten wir eine Methode, die Kara eine Anzahl Kleeblätter legen lässt:

```
void legeKleeblaetter(int anzahl) {
  for (int i = 1; i <= anzahl; i++) {
    kara.putLeaf();
    kara.move();
  }
}
```

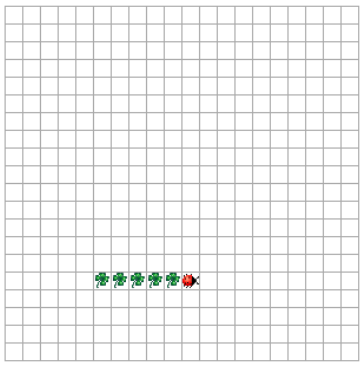
Hinweis: Die Übergabe von Parametern („int anzahl“) sowie Wiederholungen (hier eine Schleife mit „for“) werden wir später genauer kennenlernen.

Diese Methode können Sie in einem eigenen Hauptprogramm aufrufen:

```
public void myProgram() {
  world.clearAll(); // lösche alle Objekte aus Welt
  kara.setPosition(5, 15);
    // setze Kara an (x=5, y=15) nach rechts schauend
  legeKleeblaetter(5); // Methodenaufruf
}
```

Hinweis: Die ersten beiden Befehle sind nützlich, damit sie nicht jedes Mal von Hand die Welt wieder herstellen müssen.

In einer 20x20 Felder grossen Welt ausgeführt, sieht die Welt nach Programmausführung wie folgt aus:

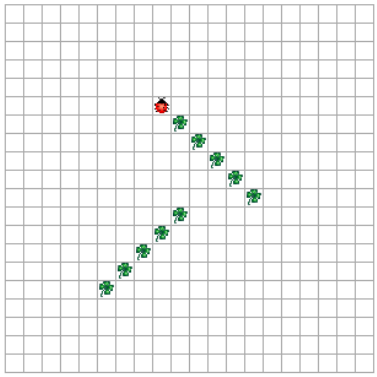


Aufgabe 1: Lassen Sie Kara diagonal Kleeblätter legen und „leer“ laufen

Wir wollen neue Methoden einführen, die Kara laufen lassen, ohne Kleeblätter zu legen, und die Kara Kleeblätter in der Diagonale legen lassen.

Das folgende Hauptprogramm soll dann die Grafik rechts erzeugen:

```
public void myProgram() {
  kara.setPosition(5, 15);
  kara.turnLeft(); // schaue nach oben
  diagonaleRechts(5);
  kara.turnRight(); // schaue nach rechts
  laufe(3);
  kara.turnLeft(); // schaue nach oben
  diagonaleLinks(5);
}
```



Damit diese Hauptprogramm starten kann, müssen Sie die fett markierten Methoden schreiben:

1. Schreiben Sie eine Methode, die Kara einfach nur eine bestimmte Anzahl Felder laufen lässt, ohne Kleeblätter zu legen:

```
void laufe(int anzahl) {
  // Ihr Code
}
```

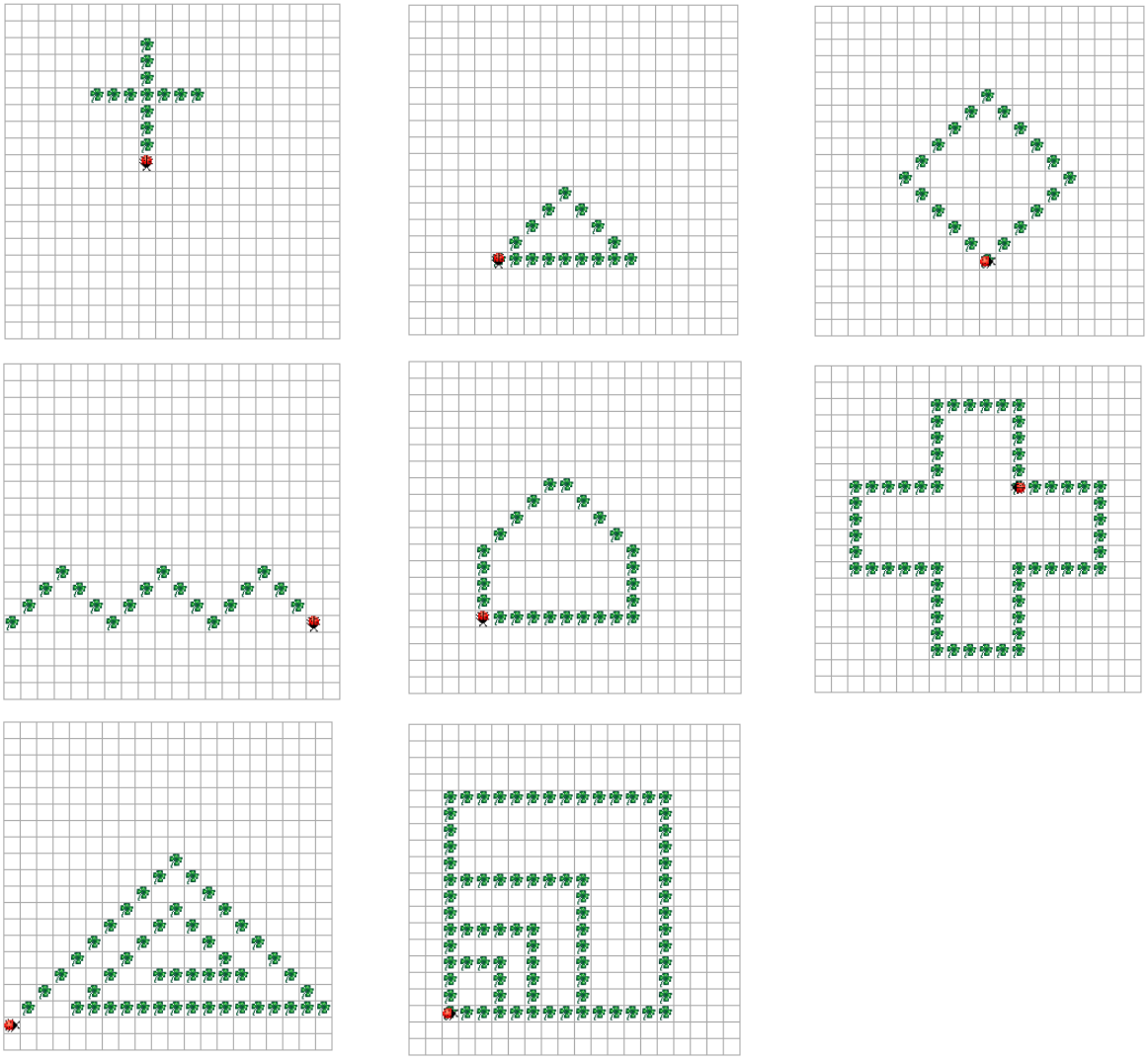
2. Schreiben Sie eine Methode, die Kara diagonal nach rechts Kleeblätter legen lässt, und eine Methode, die Kara diagonal nach links Kleeblätter legen lässt:

```
void diagonaleLinks(int anzahl) {
  // Ihr Code
}

void diagonaleRechts(int anzahl) {
  // Ihr Code
}
```

Aufgabe 2: Lassen Sie Kara Figuren zeichnen!

Mit den bisher vorgestellten und von Ihnen in Aufgabe 1 programmierten Methoden lassen sich schon diverse Figuren erzeugen, hier ein paar Beispiele:



Schreiben Sie ein paar JavaKara-Programme, die diese oder andere Figuren erstellen!

Folgende Methode könnte Ihnen nützlich sein: Sie legt wie legeKleeblaetter eine Anzahl Kleeblaetter, schaut aber jeweils, ob ein Kleeblatt auf gelegt werden kann. Das heisst, es gibt keinen Fehler, wenn „Kleeblaetter über Kleeblaetter“ gelegt werden sollen:

```
void legeKleeblaetterFallsMoeglich(int anzahl) {
  for (int i = 1; i <= anzahl; i++) {
    if (!kara.onLeaf()) {
      kara.putLeaf();
    }
    kara.move();
  }
}
```