Grundlegende Programmierkonzepte: Abläufe mit Wiederholungen (Schleifen)

Wiederholung mit Abbruchbedingung (while-Schleife)

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

Situation vor dem Start des Programms	Situation nachher
	

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

```
public void myProgram() { // mit Eclipse: myMainProgram
  while (!kara.treeFront()) {
    kara.move();
  }
}
```

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*.

Struktur und Syntax einer `while` Schleife

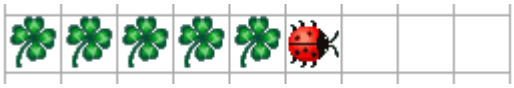
```
while (Schleifenbedingung) {
  Anweisung1
  Anweisung2
  Anweisung3
}
```

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.

Schleifen können geschachtelt werden: Eine while-Schleifen kann eine andere while-Schleife beinhalten, die wiederum eine while-Schleife beinhalten könnte, und so weiter.

Anzahl-basierte Wiederholungen (for-Schleife)

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

Situation vor dem Start des Programms	Situation nachher
	

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

```
public void myProgram() { // mit Eclipse: myMainProgram
  for (int i=1; i<=5; i++) {
    kara.putLeaf();
    kara.move();
  }
}
```

Struktur und 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 Schleifenausführung 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.

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

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.

JavaKara:
Aufgaben zu Schleifen, Wiederholungen

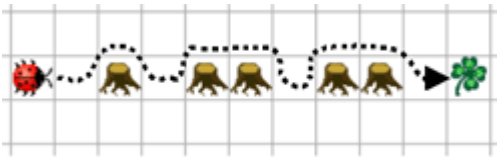
Kara, der Tunnelsucher I

Kara sucht den Eingang eines geraden Tunnels (Feld 2a). Schreiben Sie ein Programm, das ihn auf dem ersten Feld im Tunnelinnern anhalten lässt. Aber Achtung: manche Tunnels haben zunächst eine einseitige Wand, manche links, manche rechts.

Kara, der Tunnelsucher II

Kara will den Ausgang des Tunnels finden (Feld 2b). Dazu muss er zunächst den Tunnel durchqueren. Schreiben Sie ein Programm, das ihn auf dem ersten Feld nach dem Tunnel anhalten lässt – er soll nicht bis zum Ende der Gallerie laufen!

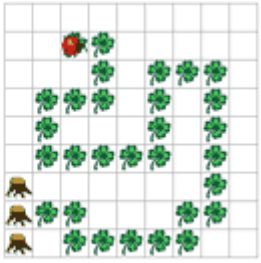
Kleeblattsuche im Wald II



Erweitern Sie Ihr Programm von „Kleeblattsuche im Wald I“ so, dass Kara auch mit mehreren nebeneinander stehenden Bäumen fertig wird!

Hinweis: Zur Lösung dieser Aufgabe müssen Sie mit geschachtelten Schleifen arbeiten. Eine Schleife für die Suche nach Kleeblatt in der unteren Zeile, eine Schleife für die Suche in der oberen Zeile nach dem Weg zurück in die untere Zeile.

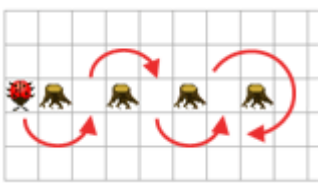
PacMan



Programmieren Sie Kara so, dass er die Spur von Kleeblättern "auffrisst"! Da Sie wissen, dass die Spur nie entlang eines Baumes geht, kann das Programm beendet werden, sobald Kara auf einem Kleeblatt vor einem Baum steht. Sie können selbst bestimmen, ob Sie auf einem Kleeblatt oder davor starten wollen.

Hinweis: Diese Aufgabe ist mit Java schon ganz schön tricky. Überlegen Sie sich zuerst genau, wie Ihr Lösungsverfahren aussehen soll, bevor Sie zu programmieren beginnen!

Slalom

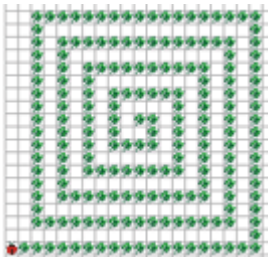


Kara möchte zwischen den Bäumen Slalom fahren. Der Anfang des Slaloms ist im Bild eingezeichnet. Programmieren Sie Kara so, dass er den Slalom endlos hin- und zurück fährt. Am Anfang ist Kara immer so platziert, dass er zuerst eine Linkskurve machen muss.

Wie lange der Parcour ist (wieviele Bäume der Slalom hat), weiss Kara zu Beginn natürlich nicht. Es soll ihm auch egal sein, ob die Bäume horizontal oder vertikal nebeneinander stehen.

Hinweis: Eventuell hilft es, wenn Sie Karas Bewegungen in „Vierteldrehungen“ um die Bäume zerlegen.

Spirale zeichnen



Programmieren Sie Kara so, dass er eine Kleeblatt-Spirale wie die obige zeichnet. Von innen nach aussen ist jede Kante der Spirale um eins länger als die vorangehende.

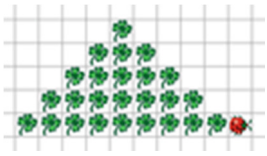
Bilder invertieren

Programmieren Sie Kara so, dass er ein "Negativbild" von dem Kleeblattbild innerhalb des Baumrechtecks erstellt. Wo ein Kleeblatt liegt, soll er es aufnehmen, und wo keines liegt, soll er eines hinlegen. Kara startet immer oben links in der Ecke mit Blick nach rechts.

Hinweis: Diese Aufgabe ist ziemlich umständlich zu lösen, wenn Kara durch die Welt gesteuert werden muss. Sie können die Lösung vereinfachen, wenn Sie direkt in der Welt arbeiten (mit world.isLeaf(x,y) und world.setLeaf(x,y,true) bzw. world.setLeaf(x,y,false)).

Hinweis: Lösen Sie diese Aufgabe zunächst nur für eine einzelne Zeile der Welt, etwa y=1 (die erste Zeile mit Kleeblättern). Dazu brauchen Sie eine Schleife. Erweitern Sie in einem zweiten Schritt Ihr Programm um eine zweite Schleife, damit alle Zeilen bearbeitet werden.

Dreiecke zeichnen



Programmieren Sie Kara so, dass er Dreiecke zeichnet!

Hinweis: Diese Aufgabe ist ziemlich umständlich zu lösen, wenn Kara durch die Welt gesteuert werden muss. Sie können die Lösung vereinfachen, wenn Sie direkt in der Welt arbeiten (mit world.isLeaf(x,y) und world.setLeaf(x,y,true) bzw. world.setLeaf(x,y,false)).

Hinweis: Die Lösung ist recht nahe bei der Lösung des Rechtecks. Überlegen Sie sich, wo der Unterschied liegt!

Labyrinthe (schwierig)

Führen Sie Kara durch das Labyrinth zum Kleeblatt. Jede horizontale Baumreihe, ausser der untersten, hat genau einen Ausgang, der in die nächst höhere Zeile führt. Diesen muss Kara jeweils finden. Hinter dem letzten Ausgang wartet das Kleeblatt auf ihn.

Programmieren Sie Kara so, dass er das Kleeblatt findet und aufnimmt. Dabei soll er nie an einem Ausgang vorbeilaufen, ohne ihn zu benutzen! Zu Beginn schaut Kara immer nach rechts.