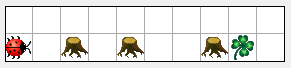
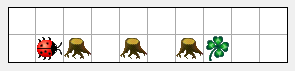
Kapitel 2: Programmfluss steuern

# Das Flussdiagramm

Um schwierige Aufgaben beim Programmieren zu lösen, ist es oft hilfreich, den Programmablauf in einem Flussdiagramm darzustellen. Folgende Symbole werden in einem Flussdiagramm verwendet:

|  |  |
| --- | --- |
| **Symbole** | **Bedeutung** |
|  | Start bzw. Stopp |
|  | Aktivität |
|  | Entscheidung / Bedingung |

Aufgabe 9: Um Baum Herum II



1. Ergänzen Sie das Flussdiagramm so, dass Kara das Kleeblatt in allen Welten mit den folgenden Eigenschaften erreicht:

* Das Kleeblatt liegt immer gerade vor ihm – er muss nur um die Bäume herumlaufen.
* Es stehen nie zwei Bäume nebeneinander

*Hinweis: Start bedeutet hier, dass der Act-Knopf gedrückt wird.*

act()-Start

treeFront()

no

goAroundTree()

move()

yes

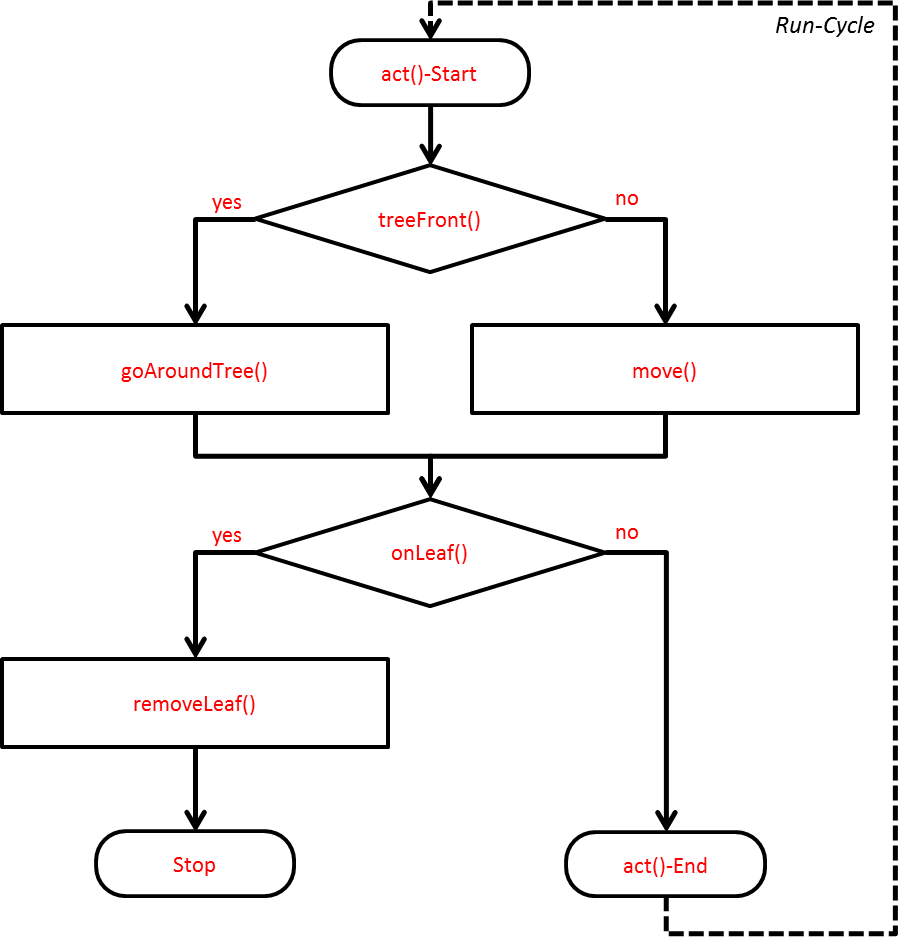
act()-End

*Run-Wiederholungen*

|  |
| --- |
| **Kara** |
| ***Aktionen****:* move() turnLeft() turnRight()  putLeaf()  removeLeaf()  ***Sensoren****:*  treeFront()  treeLeft()  treeRight()  onLeaf()  mushroomFront() |

1. Zeichnen Sie ein erweitertes Diagramm, so dass Kara das Kleeblatt am Schluss auch aufliest, wenn er es findet. Rechts finden Sie nochmals die verfügbaren Methoden von Kara:

Lösungsvorschlag:



# Bedingte Anweisungen in Java[[1]](#footnote-1)

Bedingung

**if** (treeFront()) {

Anweisungsblock 1  
wenn Bedingung *true* ist.

turnLeft();

} **else** {

Anweisungsblock 2  
wenn Bedingung *false* ist.

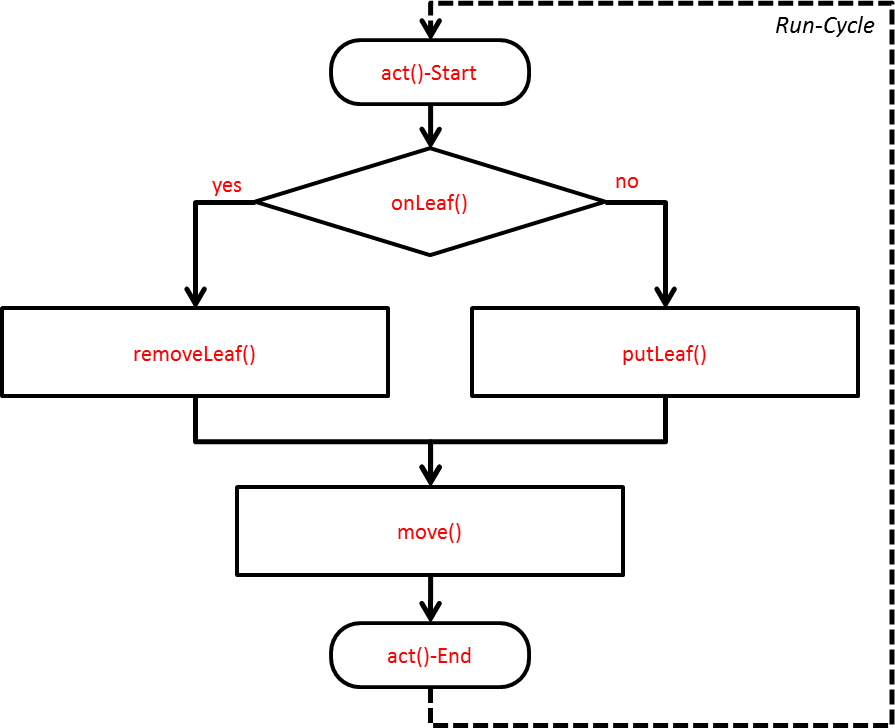
move();

}

**Hinweis: Der else-Teil (Anweisungsblock 2) kann, wenn er nicht benötigt wird, weggelassen werden.**

Aufgabe 10:

Beschreiben Sie zuerst mit Worten, was die folgenden Codebeispiele bewirken. Skizzieren Sie diese jeweils auch als Flussdiagramm.

1. 

if (onLeaf()) {

removeLeaf();

} else {

putLeaf();

}

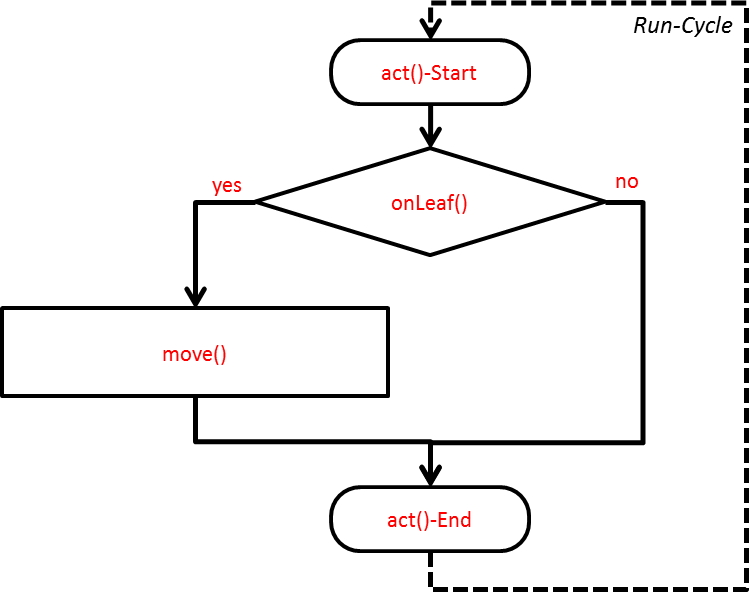
move();

Nimmt ein Blatt weg, wenn es eines hat, legt

ein Blatt hin, wenn keines dort ist (Invertieren).



if (onLeaf()) {



move();

}

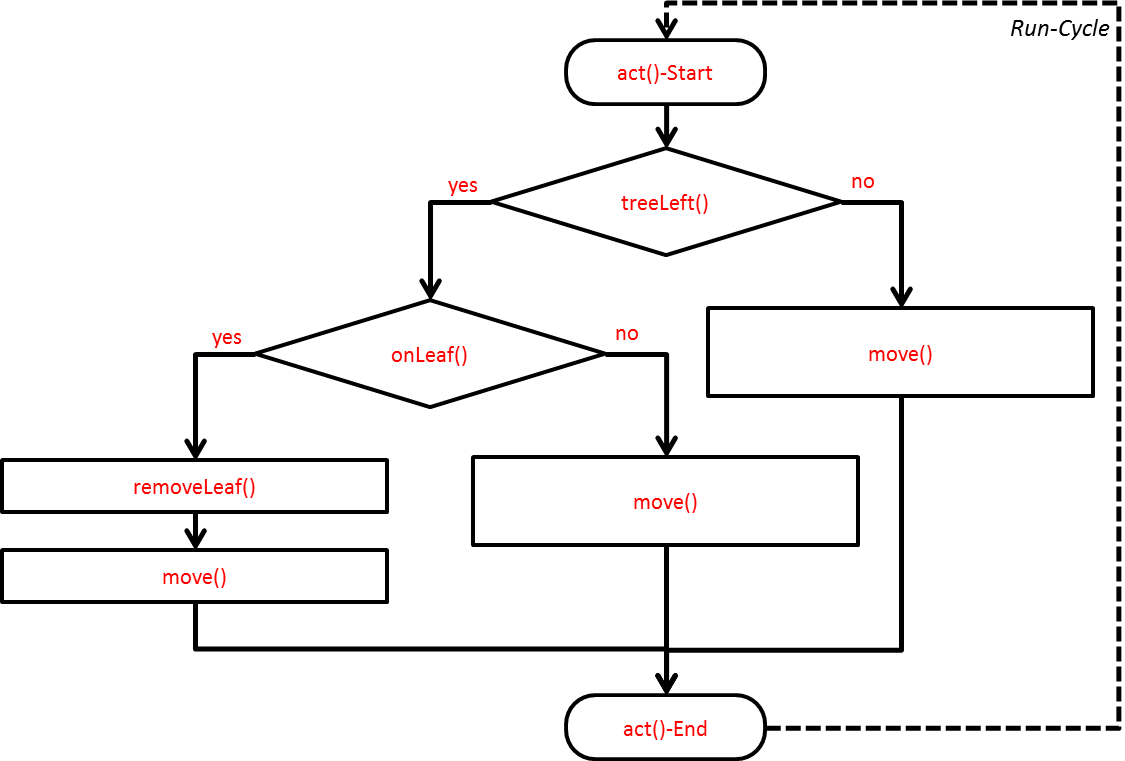
Wenn er auf einem Blatt ist, geht er einen Schritt vorwärts.

***Aufgabe 11: Verschachtelte Bedingungen***

1. Bedingte Anweisungen können auch ineinander verschachtelt sein. Beschreiben Sie was beim Ausführen des Programms in der gegebenen Kara-Wiese passiert. Zeichnen Sie ein entsprechendes Flussdiagramm.

if (treeLeft()) {

if (onLeaf()) {

 removeLeaf();

move();

} else {

move();

}

} else {

move();

}

Nimmt das Blatt auf, wenn links ein Baum steht.

## Implementieren

Nun können Sie die Übungen, welche Sie gezeichnet haben am Computer umsetzen. Sie sollen wie folgt vorgehen:

Zu Aufgabe 9:

* Öffnen Sie das Szenario ***Kara 09 …*** aus dem Ordner ***scenarios-chapter-2***. In diesem Szenario ist die Methode goAroundTree()bereits programmiert und ein Teil der act()-Methode ist vorbereitet.
* Programmieren Sie nun den Ablauf, welchen Sie unter Aufgabe 9b als Flussdiagram gezeichnet haben in der act()-Methode.
* In diesem Szenario haben Sie **verschiedene Wiesen (WorldSetup)** zur Verfügung (a, b und c).
* Ihr Programm sollte auf jeder Wiese **ohne Fehlermeldungen** funktionieren.

Zu Aufgabe 11:

* Laden Sie das Szenario ***Kara 11 …*** in Greenfoot und programmieren Sie das unter Aufgabe 11a skizzierte Programm.  
  Formen Sie das Programm so um, dass Kara die Kleeblätter nur dann aufliest, wenn kein Baum neben ihm ist.

# Logische Operatoren

Unser Kara kann nun schon mehr als einfach nur simple Befehle abzuarbeiten. Er kann durch die Benutzung von Bedingungen auf die jeweiligen Ergebnisse eines Sensors unterschiedlich reagieren. Es sollte Kara aber natürlich auch möglich sein, auf zwei oder mehrere Sensoren gleichzeitig zu reagieren.

Die folgende Tabelle zeigt die drei wichtigsten logischen Operatoren in Java:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Beschreibung** | **Beispiel** | |
| **&&** | **und** | treeFront() && onLeaf() | Ist nur erfüllt (true) , wenn beide Aussagen erfüllt sind, d.h. wenn Kara vor einem Baum und auf einem Blatt steht. |
| **||** | **oder** | treeFront() || onLeaf() | Ist dann erfüllt (true), wenn entweder die eine oder die andere Aussage oder beide erfüllt sind. |
| **!** | **nicht** | !treeFront() | Ändert einen Ausdruck von true in false und umgekehrt. Diese Aussage wäre also dann erfüllt (true), wenn Kara nicht vor einem Baum steht. |

Ein Beispiel in Java würde wie folgt aussehen:

if (treeLeft() && onLeaf()) {

// Mache etwas…

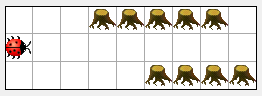
}

oder auch kombiniert:

if (treeLeft() && !treeRight()) {

// Mache etwas…

}

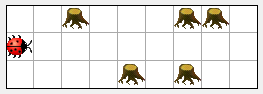


Tunneleingang

Aufgabe 12: Angst vor Tunnel[[2]](#endnote-1)

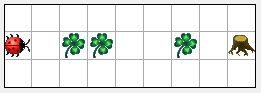
Kara hat etwas Angst vor Tunneln. Er soll auf jedem Feld überprüfen, ob es ein Tunneleingang ist (d.h. ob es auf beiden Seiten Bäume hat). Ist dies der Fall, so lässt er vor Schreck gleich ein Kleeblatt fallen.

Laden Sie das Szenario ***Kara 12*** *…*, schreiben Sie das Programm und testen Sie es mit allen drei Wiesen.

Aufgabe 13: Blatt beim Baum

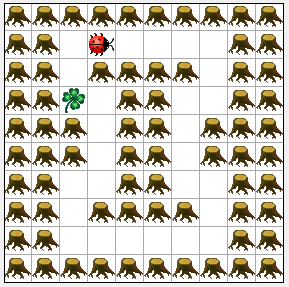
Nun soll Kara wieder geradeaus gehen und überall dort ein Blatt legen, wo entweder links oder rechts oder auf beiden Seiten ein Baum steht.

Laden Sie das Szenario ***Kara 13*** *…* und schreiben Sie das Programm dazu.

Aufgabe 14: Blätter legen bis zum Baum

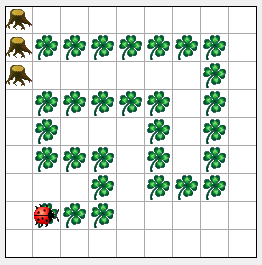
Kara soll vorwärts laufen und dabei überall ein Blatt legen, wo keines ist. Wenn er beim Baum angelangt ist, soll er nichts mehr machen (auch wenn der Act- oder der Run-Knopf nochmals gedrückt wird).

Laden Sie das Szenario ***Kara 14*** *…*, schreiben Sie das Programm und testen Sie es.

Aufgabe 15: Rundgang

Kara geht in einem Rundgang auf die Suche nach einem Kleeblatt (und liest es auf). Jedes Feld im Rundgang hat genau zwei freie benachbarte Felder. Eines liegt immer hinter Kara, von diesem Feld aus ist er auf das aktuelle Feld gekommen.

Laden Sie das Szenario ***Kara 15*** *…* und verfassen Sie ein passendes Programm dazu. Testen Sie Ihr Programm mit allen drei Wiesen!

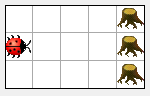
*Tipp: Stellen Sie sich vor, was nach jedem Drücken des Act-Knopfes geschehen muss. Zeichnen Sie dafür als Hilfe ein Flussdiagramm.*

Zusatzaufgabe 16 (schwierig): Kara spielt Pacman

Kara spielt Pacman: Er steht auf dem ersten Kleeblatt einer langen Spur von Kleeblättern, die vor einem Baum endet. Er soll alle Kleeblätter auffressen und vor den Bäumen stoppen.

Schreiben Sie zur besseren Übersicht für gewisse Programmteile eigene Methoden.

# Schleifen

Kara kann jetzt nach von uns festgelegten Regeln auf verschiedene Situationen reagieren. Er ist allerdings noch nicht in der Lage, Anweisungen bis zum Eintreten eines bestimmten Ereignisses zu **wiederholen**. Zum mehrfachen Ausführen von Anweisungsblöcken werden Schleifen verwendet.

Als Beispiel möchten wir folgendes tun:   
*Kara soll sich solange vorwärts bewegen, bis er auf einen Baum trifft.*

Im Flussdiagramm sieht man, dass move() immer wieder ausgeführt wird,   
solange kein Baum vor Kara steht.

Start

! treeFront()

no

move()

yes

Stop

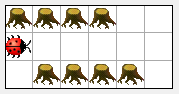
**Die Schreibweise in Java:**

while (!treeFront()) {

move();

}

Aufgabe 17:



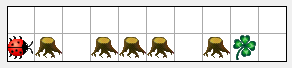
**1 2 3 4 5**

Gegeben ist die folgende Situation: Kara steht vor einem Tunnel.

Beschreiben Sie, was die folgenden Schleifen bewirken und wie viele Schritte Kara macht.

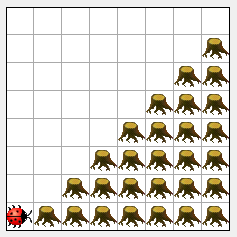
|  |  |  |
| --- | --- | --- |
| **Code** | **Beschreibung** | **Anzahl Schritte** |
| while (treeLeft()) {  move();  } | Solange links ein Baum steht, mache einen Schritt. | 4 |
| while (treeRight()) {  move();  } | Solange rechts ein Baum steht, mache einen Schritt. | 0 |
| while (treeLeft() || treeRight()) {  move();  } | Solange entweder links oder rechts ein Baum steht, mache einen Schritt. | 5 |
| if (treeLeft()) {  move();  } while (treeLeft() && treeRight()) {  move();  } | Zuerst ein Schritt, wenn links ein Baum.  Solange links und rechts ein Baum steht, mache einen Schritt. | 4 |
| while (!treeFront) {  if (treeLeft()) {  move();  }  } | Solange kein Baum vor Kara steht  Wenn links ein Baum, mache einen   Schritt  **Achtung: Endlosschleife!** | 4 |

Aufgabe 18: Um Baum Herum III



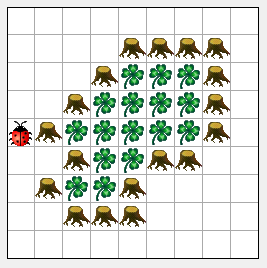
Dies ist die ähnliche Übung, wie in Aufgabe 9: Kara soll ein Kleeblatt finden, das geradeaus vor ihm liegt. Nun können aber eine beliebige Anzahl Bäume hintereinander stehen.

1. Laden Sie das Szenario ***Kara 18*** *…* und verbessern Sie die Methode goAroundTree() so, dass Kara um mehrere Bäume herumgehen kann. Testen Sie Ihr Programm in allen vorgegebenen Wiesen.
2. Ändern Sie die act()-Methode so ab, dass man nur noch einmal auf den Act-Knopf drücken muss. Kara soll dann automatisch um die Bäume laufen, bis er das Kleeblatt erreicht. Am Schluss soll er es dann wieder auffressen.

Aufgabe 19: Treppensteigen

Kara soll eine beliebig lange Treppe hochlaufen.

Schreiben Sie eine Methode oneStepUp(), wo Sie Kara eine einzelne Stufe hochsteigen lassen. Überlegen Sie Sich, wie Kara erkennen kann, ob er noch eine Stufe steigen muss.

Zusatzaufgabe 20 (schwierig): Kara als Wächter

Kara will einen Wald bewachen. Er soll endlos aussen am Waldrand entlang laufen. Als Hilfe können Sie Sich ein Flussdiagramm zeichnen.

*Hinweis: Für eine Endlosschleife können wir auf den Run-Knopf drücken.*

1. Basiert auf: Thomas Kempe und David Tepasse, Informatik 1 - Softwareentwicklung mit Greenfoot und BlueJ, 1. Aufl (Paderborn: Schöningh, 2010), S. 36. [↑](#footnote-ref-1)
2. ***Hinweis zu Kara:***

   * Ideen und Konzepte von Kara wurden entwickelt von Jürg Nievergelt, Werner Hartmann, Raimond Reichert et al., <http://www.swisseduc.ch/informatik/karatojava/>, abgerufen Februar 2011.
   * Einige Kara-Übungen basieren auf Unterlagen von Horst Gierhardt, <http://www.swisseduc.ch/informatik/karatojava/javakara/material/>, abgerufen Februar 2011.

   [↑](#endnote-ref-1)