Michael Kölling

Einführung in Java mit Greenfoot

Spielerische Programmierung mit Java

PEARSON
Schule

Michael Kölling

Einführung in Java mit Greenfoot

Spielerische Programmierung mit Java

PEARSON Schule

ein Imprint von Pearson Education
München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Java unterstützt verschiedene Arten von Variablen. Die erste, die wir hier betrachten wollen, heißt *Objektvariable* oder *Zustandsfeld* (diese beiden Begriffe bedeuten das Gleiche). Die anderen Arten von Variablen werden wir später kennenlernen.

Eine Objektvariable ist ein kleiner Speicherblock, der zum Objekt gehört (daher auch der Name). Alles, was in dieser Variablen gespeichert ist, steht so lange zur Verfügung, wie das Objekt existiert, und kann später abgerufen werden.

Die Art der Variablen muss im Quelltext der Klasse beschrieben werden. Dass nennt man Deklaration. Die Deklaration einer Objektvariablen wird eingeleitet durch das Schlüsselwort **private** gefolgt von dem Typ der Variablen und dem Variablennamen:

```
private variablentyp variablenname;
```

Konzept

Objektvariablen (auch **Zustandsfelder** genannt) dienen zum Speichern von Informationen (Objekte oder Werte), die später benötigt werden.

Der Typ der Variablen definiert, was wir darin speichern wollen. Da wir in unserem Fall Objekte vom Typ **GreenfootImage** in unserer Variablen speichern wollen, sollte der Typ **GreenfootImage** lauten. Der Variablenname gibt uns die Möglichkeit, der Variablen einen Namen zu geben, den wir später verwenden können, um auf die Variable Bezug zu nehmen. Er sollte beschreiben, wofür die Variable verwendet wird.

Werfen wir zur Veranschaulichung einen Blick auf unsere Klasse **Crab** (Listing 4.2):

Listing 4.2
Die Klasse **Crab** mit zwei Objektvariablen.

```
import greenfoot.*; // (Actor, World, Greenfoot, GreenfootImage)

// Kommentar ausgelassen

public class Crab extends Animal
{
   private GreenfootImage image1;
   private GreenfootImage image2;

   // Methoden ausgelassen

}
```

In diesem Beispiel haben wir zwei Variablen in unserer Klasse **Crab** deklariert. Beide sind vom Typ **GreenfootImage** und heißen **image1** und **image2**.

Wir setzen die Deklarationen unserer Objektvariablen immer ganz an den Anfang unserer Klasse, noch vor die Konstruktoren und Methoden. Java schreibt dies zwar nicht zwingend vor, doch es ist guter Stil, der ein schnelles Nachschlagen von Variablendeklarationen erlaubt.

Übung 4.7 Bevor du diesen Code hinzufügst, klicke mit der rechten Maustaste auf ein Krabben-Objekt in deiner Welt und wähle aus dem aufspringenden Kontextmenü der Krabbe den Befehl INSPIZIEREN. Notiere dir alle Variablen, die im Krabben-Projekt angezeigt werden.

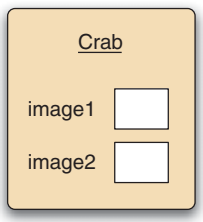
Übung 4.8 Warum glaubst du, hat die Krabbe überhaupt irgendwelche Variablen, auch wenn wir keine in unserer Klasse **Crab** deklariert haben?

Übung 4.9 Füge die Variablendeklarationen aus Listing 4.2 in deine Version der Klasse **Crab** ein. Achte darauf, dass sich die Klasse übersetzen lässt.

Übung 4.10 Nachdem du die Variablen ergänzt hast, erzeuge und inspiziere erneut dein Krabben-Objekt. Notiere dir die Variablen und ihre Werte (die in den weißen Kästchen angezeigt werden).

Beachte, dass wir durch die Deklaration dieser beiden **GreenfootImage**-Variablen noch keine zwei **GreenfootImage**-Objekte erhalten, sondern lediglich zwei leere Kästchen, in denen wir zwei Objekte speichern können, Abbildung 4.3 (in der Abbildung sind die beiden Objektvariablen als zwei weiße Kästchen dargestellt).

Abbildung 4.3
Ein Krabben-Objekt mit zwei leeren Objektvariablen.



Als Nächstes müssen wir die beiden Bildobjekte erzeugen und in den Variablen speichern. Wie diese Objekte erzeugt werden, wurde bereits oben gezeigt. Das von uns dazu verwendete Codefragment lautete:

```
new GreenfootImage("crab2.png")
```

Um das Objekt in einer Variablen zu speichern, benötigen wir ein Java-Konstrukt, das als *Zuweisung* bezeichnet wird.

4.6 Zuweisung

Eine Zuweisung ist eine Anweisung, die es uns ermöglicht, irgendetwas in einer Variablen zu speichern. Für die Zuweisung wird ein Gleichheitszeichen verwendet:

```
variable = ausdruck;
```

Links des Gleichheitszeichens steht der Name der Variablen, in der wir etwas speichern wollen, und rechts das, was wir speichern wollen. Da das Gleichheitszeichen für Zuweisung steht, wird es auch als *Zuweisungssymbol* bezeichnet. Wir lesen es in der Regel als „wird zu“, zum Beispiel: „Variable wird zu Ausdruck“.

In unserem Krabben-Beispiel schreiben wir:

```
image1 = new GreenfootImage("crab.png");
image2 = new GreenfootImage("crab2.png");
```

Diese zwei Codezeilen erzeugen die beiden Bilder, die wir verwenden wollen, und speichern sie in unseren beiden Variablen **image1** und **image2**. Nach Ausführung dieser Anweisungen haben wir drei Objekte (eine Krabbe und zwei Bilder), wobei die Variablen der Krabbe Verweise auf die Bilder enthalten. Abbildung 4.4 verdeutlicht dies.

Konzept

Eine **Zuweisungsanweisung** weist einer Variablen ein Objekt oder einen Wert zu.

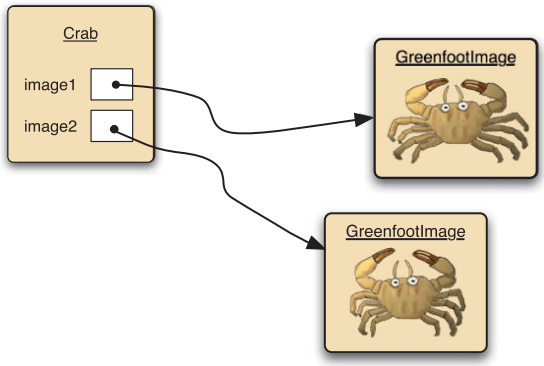


Abbildung 4.4
Ein Krabben-Objekt mit zwei Variablen, die auf Bildobjekte zeigen.

Als Nächstes stellt sich uns natürlich die Frage, wo der Code einzufügen ist, der die Bilder erzeugt und in Variablen speichert. Da dies nur einmal bei der Erzeugung des Krabben-Objekts erfolgen soll und nicht bei jedem Aktionsschritt, können wir den Code nicht in der **act**-Methode unterbringen. Stattdessen fügen wir den Code in einen Konstruktor ein.

Konzept

Wenn ein Objekt einer Variablen zugewiesen wird, enthält die Variable eine **Referenz** auf dieses Objekt.

4.7 Die Konstruktoren der Akteur-Klassen

Am Anfang dieses Kapitels haben wir gesehen, wie mithilfe des Konstruktors der Welt-Klasse die Welt initialisiert wurde. Auf ähnliche Weise können wir mit einem Konstruktor einer Akteur-Klasse den Akteur initialisieren. Der Code im Konstruktor wird nur einmal ausgeführt, und zwar wenn der Akteur erzeugt wird. Listing 4.3 zeigt einen Konstruktor für die Klasse **Crab**, der die beiden Objektvariablen initialisiert, indem er Bilder erzeugt und diese den Variablen zuweist.

Listing 4.3:
Die Variablen im Konstruktor initialisieren.

```
import greenfoot.*; // (Actor, World, Greenfoot, GreenfootImage)

// Kommentar ausgelassen

public class Crab extends Animal
{
   private GreenfootImage image1;
   private GreenfootImage image2;

   /**
   * Erzeugt eine Krabbe und initialisiert ihre beiden Bilder.
   */
   public Crab()
   {
      image1 = new GreenfootImage("crab.png");
      image2 = new GreenfootImage("crab2.png");
      setImage(image1);
   }

   // Methoden ausgelassen
}
```

Die gleichen Regeln, die für den **World**-Konstruktor beschrieben wurden, gelten auch für den **Crab**-Konstruktor:

- Die Signatur eines Konstruktors beinhaltet keinen Rückgabetyp.
- Der Name des Konstruktors lautet wie der Name der Klasse.
- Der Konstruktor wird automatisch ausgeführt, wenn ein Krabben-Objekt erzeugt wird.

Die letzte Regel – die automatische Ausführung des Konstruktors – stellt sicher, dass die Bildobjekte automatisch erzeugt und zugewiesen werden, wenn eine Krabbe erzeugt wird. Nach der Erzeugung der Krabbe entspricht die Situation also der in Abbildung 4.4.

Die letzte Zeile des Konstruktors legt das erste der beiden erzeugten Bilder als das aktuelle Bild der Krabbe fest:

```
setImage(image1);
```

Dies zeigt, wie der Name der Variablen (**image1**) jetzt dazu verwendet werden kann, um auf das darin gespeicherte Bildobjekt Bezug zu nehmen.

Übung 4.11 Füge diesen Konstruktor in deine **Crab**-Klasse ein. Du wirst keine Verhaltensänderungen bei der Krabbe beobachten können, aber die Klasse sollte sich übersetzen lassen und du solltest in der Lage sein, Krabben zu erzeugen.

Übung 4.12 Inspiziere noch einmal dein Krabben-Objekt. Beachte auch hier wieder die Variablen und ihre Werte. Vergleiche sie mit denen, die du zuvor notiert hast.

4.8 Die Bilder wechseln

Damit sind wir jetzt so weit, dass uns zwei Bilder der Krabbe für die Animation zur Verfügung stehen. Doch die Animation selbst haben wir noch nicht in Angriff genommen. Dies ist jetzt allerdings relativ einfach.

Für die Animation müssen wir lediglich zwischen unseren beiden Bildern hin- und herwechseln. Mit anderen Worten: Bei jedem Schritt wollen wir, wenn wir gerade **image1** zeigen, zu **image2** wechseln und umgekehrt. In Pseudocode sähe das folgendermaßen aus:

```
if (wenn unser aktuelles Bild image1 ist) then
    verwende jetzt image2
else
    verwende jetzt image1
```

Pseudocode, wie er hier verwendet wird, ist eine Technik, eine Aufgabe so auszudrücken, dass die Struktur zum Teil echter Java-Code ist und zum Teil einfaches Deutsch. Oft ist Pseudocode eine gute Hilfe, um herauszufinden, wie der reale Code aussehen sollte. Wir können unser Problem jetzt in richtigem Java-Code ausdrücken (Listing 4.4).

Listing 4.4
Zwischen zwei Bildern wechseln.

```
if ( getImage() == image1 )
{
    setImage(image2);
}
else
{
    setImage(image1);
}
```

In diesem Codefragment fallen uns einige neue Elemente auf:

- Die Methode **getImage** kann dazu verwendet werden, um das aktuelle Bild des Akteurs zu erhalten.
- Der Operator **==** (zwei Gleichheitszeichen) kann dazu verwendet werden, um einen Wert mit einem anderen zu vergleichen. Das Ergebnis ist entweder **true** oder **false**.
- Die **if**-Anweisung hat eine Form, der wir bisher noch nicht begegnet sind. Bei dieser Form folgen auf den ersten Rumpf der **if**-Anweisung das Schlüsselwort **else** und ein weiterer Anweisungsblock. Im nächsten Abschnitt wollen wir diese neue Form der **if**-Anweisung genauer unter die Lupe nehmen.

Konzept

Mithilfe des doppelten Gleichheitszeichens (**==**) können wir prüfen, ob zwei Dinge **gleich** sind.

Fallstricke

Ein immer wiederkehrender Fehler ist es, den Anweisungsoperator (**=**) mit dem Operator, der auf Gleichheit prüft (**==**), zu verwechseln. Wenn du prüfen möchtest, ob zwei Werte oder Variablen gleich sind, musst du zwei Gleichheitszeichen schreiben.

4.9 Die if/else-Anweisung

Bevor wir fortfahren, wollen wir die **if**-Anweisung einmal näher betrachten. Wie wir gerade gesehen haben, kann eine **if**-Anweisung auch in der folgenden Form geschrieben werden:

```
if ( bedingung )
{
   anweisungen;
}
else
{
   anweisungen;
}
```

Konzept

Die **if/else**-Anweisung führt ein Codefragment aus, wenn eine gegebene Bedingung wahr ist, und ein anderes Codefragment, wenn die Bedingung falsch ist.

Diese **if**-Anweisung enthält zwei Blöcke (d.h. zwei Paare geschweifter Klammern um jeweils eine Liste von Anweisungen): die *if-Klausel* und die *else-Klausel* (in dieser Reihenfolge).

Wenn diese **if**-Anweisung ausgeführt wird, wird zuerst die Bedingung ausgewertet. Ist die Bedingung wahr, wird die **if**-Klausel ausgeführt und die Ausführung setzt mit dem Code unterhalb der **else**-Klausel fort. Ist die Bedingung falsch, wird statt der **if**-Klausel die **else**-Klausel ausgeführt. Auf diese Weise wird immer einer der beiden Anweisungsblöcke ausgeführt, aber nie beide.

Der **else**-Teil mit dem zweiten Block ist optional. Wenn wir ihn weglassen, erhalten wir die kürzere Version der **if**-Anweisung, die wir bereits von früher kennen.

Damit verfügen wir über alle nötigen Informationen, um unsere Aufgabe zu meistern. Jetzt ist es Zeit, wieder zur Tastatur zurückzukehren und das Gelernte in die Praxis umzusetzen.

Übung 4.13 Füge den Code aus Listing 4.4 zum Alternieren der Bilder in die **act**-Methode deiner **Crab**-Klasse ein. Versuche es einmal selbst. (Wenn du einen Fehler einbaust, behebe ihn. Das Beispiel sollte funktionieren.) Klicke in Greenfoot auch ab und zu auf den ACT-Button anstatt auf den RUN-Button – damit kannst du das Verhalten etwas besser beobachten.

Übung 4.14 In ***Kapitel 3*** haben wir darüber gesprochen, dass man für Unteraufgaben eigene Methoden definieren sollte, anstatt immer mehr Code in die **act**-Methode zu packen. Übertrage dieses Prinzip auf den Code zum Alternieren des Bildes: Erzeuge eine neue Methode namens **switchImage**, verschiebe in diese deinen Code und rufe die Methode von der **act**-Methode aus auf.

Übung 4.15 Rufe die Methode **switchImage** interaktiv von dem Kontextmenü der Krabbe aus auf. Funktioniert das?

4.10 Würmer zählen

Kommen wir jetzt zu unserem letzten Thema im Zusammenhang mit den Krabben: dem Zählen. Wir wollen Funktionalität hinzufügen, die die Krabbe mitzählen lässt, wie viele Würmer sie gefressen hat. Wenn sie acht Würmer gefressen hat, haben wir das Spiel gewonnen. Im letzteren Fall möchten wir dann auch noch einen kurzen „Gewonnen-Sound“ abspielen.

Um unsere Pläne zu realisieren, müssen wir einige Ergänzungen an unserem Krabben-Code vornehmen. So benötigen wir

- eine Objektvariable, um die Zahl der aktuell gefressenen Würmer zu speichern,
- eine Zuweisung, die diese Variable am Anfang mit 0 initialisiert,
- Code, um unsere Zählung jedes Mal, wenn wir einen Wurm gefressen haben, um eins zu erhöhen (inkrementieren),
- und Code, der prüft, ob wir acht Würmer gefressen haben, und, wenn ja, das Spiel anhält und einen Sound abspielt.

Lass uns die Aufgaben in der Reihenfolge erledigen, in der wir sie aufgeführt haben.

In Anlehnung an das Beispiel aus Abschnitt *Objektvariablen (Zustandsfelder)* definieren wir eine neue Objektvariable. Unter unseren bereits vorhandenen Objektvariablen fügen wir die folgende Zeile ein:

```
private int wormsEaten;
```

Das Schlüsselwort **private** leitet all unsere Definitionen von Objektvariablen ein. Die folgenden zwei Wörter geben den Typ und den Namen unserer Variablen an. Der Typ **int** weist darauf hin, dass wir Integer (ganze Zahlen) in dieser Variablen speichern wollen, und der Name **wormsEaten** verrät uns, welchen Zweck diese Variable erfüllt.

Als Nächstes fügen wir folgende Zeile am Ende unseres Konstruktors ein:

```
wormsEaten = 0;
```

Damit initialisieren wir die Variable **wormsEaten** bei der Erzeugung der Krabbe mit 0. Eigentlich ist diese Zeile überflüssig, da Objektvariablen vom Typ **int** automatisch mit 0 initialisiert werden. Doch manchmal benötigen wir einen Anfangswert ungleich 0, sodass das Schreiben einer eigenen Initialisierunsanweisung ruhig zur Gewohnheit werden sollte.

Was uns noch bleibt, ist, die Würmer zu zählen und zu prüfen, ob wir acht erreicht haben. Dies muss jedes Mal erfolgen, wenn wir einen Wurm fressen. Deshalb suchen wir nach unserer Methode **lookForWorm**, die den Code zum Fressen der Würmer enthält. Hier fügen wir eine Codezeile hinzu, die die Wurmzählung inkrementiert:

```
wormsEaten = wormsEaten + 1;
```

In dieser Zuweisung wird zuerst die Seite rechts des Zuweisungssymbols ausgewertet (**wormsEaten + 1**). Wir lesen also den aktuellen Wert von **wormsEaten** und addieren dazu eine 1. Anschließend weisen wir diesen Wert wieder der Variablen **wormsEaten** zu. Als Folge wird die Variable um 1 inkrementiert.

Jetzt benötigen wir noch eine **if**-Anweisung, die prüft, ob wir schon acht Würmer gefressen haben, und in diesem Fall den Sound abspielt und die Ausführung anhält.

Listing 4.5 enthält den vollständigen Code der Methode **lookForWorms**. Die hier verwendete Sounddatei (*fanfare.wav*) liegt bereits im *sounds*-Ordner deines Szenarios, sodass sie einfach abgespielt werden kann.

Übung 4.16 Füge den zuvor besprochenen Code in dein Szenario ein, teste ihn und stelle sicher, dass er funktioniert.

Listing 4.5
Würmer zählen und prüfen, ob wir gewonnen haben.

```
/**
 * Prüft, ob wir auf einen Wurm gestoßen sind.
 * Wenn ja, wird er gefressen. Wenn nein, passiert nichts.
 * Wenn wir acht Würmer gefressen haben, haben wir gewonnen.
 */
public void lookForWorm()
{
    if ( canSee(Worm.class) )
    {
        eat(Worm.class);
        Greenfoot.playSound("slurp.wav");

        wormsEaten = wormsEaten + 1;
        if (wormsEaten == 8)
        {
            Greenfoot.playSound("fanfare.wav");
            Greenfoot.stop();
        }
    }
}
```

Übung 4.17 Als zusätzliche Kontrolle kannst du den Objektinspektor für dein Krabben-Objekt öffnen (wähle dazu einfach INSPIZIEREN aus dem Kontextmenü der Krabbe), bevor du mit dem Spielen anfängst. Lass den Inspektor geöffnet, während du spielst, und beobachte die Variable **wormsEaten**.

4.11 Weitere Ideen

Das Szenario *little-crab-5* im Szenarien-Ordner des Buchkapitels ist eine Version des Projekts, das alle hier besprochenen Erweiterungen beinhaltet.

Wir werden uns jetzt von diesem Szenario verabschieden und einem anderen Beispiel zuwenden, auch wenn es noch viele naheliegende (oder weniger naheliegende) Möglichkeiten für Verbesserungen gibt. So könntest du zum Beispiel

- verschiedene Bilder für den Hintergrund und die Akteure verwenden;
- weitere Arten von Akteuren einführen;
- dich nicht automatisch vorwärtsbewegen, sondern nur, wenn die Taste ↑ gedrückt wird;
- ein Spiel entwickeln, das von zwei Spielern gespielt wird, indem du eine zweite tastaturgesteuerte Klasse einführst, die auf andere Tasten lauscht;
- festlegen, dass für jeden gefressenen Wurm (oder zu beliebigen Zeiten) neue Würmer auftauchen
- und dir noch vieles mehr überlegen.

Übung 4.18 Das Bild der Krabbe ändert sich beim Laufen sehr schnell, was die Krabbe ein wenig hyperaktiv wirken lässt. Vielleicht wäre es netter, wenn sich das Bild der Krabbe nur bei jedem zweiten oder dritten **act**-Schritt ändern würde. Versuche einmal, dies zu implementieren. Dazu könntest du einen Zähler hinzufügen, der in der **act**-Methode inkrementiert wird. Jedes Mal, wenn dieser Zähler 2 (oder 3) erreicht, ändert sich das Bild und der Zähler wird wieder auf 0 zurückgesetzt.

Zusammenfassung der Programmiertechniken

In diesem Kapitel haben wir uns mit einer Reihe von neuen Programmierkonzepten vertraut gemacht. Wir haben gesehen, wie Konstruktoren dazu genutzt werden können, um Objekte zu initialisieren – Konstruktoren werden immer ausgeführt, wenn ein neues Objekt erzeugt wird. Wir haben gelernt, wie mittels Objektvariablen (auch *Zustandsfelder* genannt) und Zuweisungsanweisungen Informationen gespeichert werden und wie man später auf diese Informationen zugreift. Wir haben die Anweisung **new** verwendet, um neue Objekte im Quelltext zu erzeugen, und zum Schluss haben wir eine vollständige Version der **if**-Anweisung kennengelernt, die einen **else**-Teil umfasst, der ausgeführt wird, wenn die Bedingung nicht wahr ist.

Mithilfe all dieser Techniken können wir bereits eine ganze Menge Code schreiben.

Zusammenfassung der Konzepte

- Der **Konstruktor** einer Klasse ist eine besondere Art Methode, die immer automatisch ausgeführt wird, wenn ein Objekt dieser Klasse erzeugt wird.
- Java-Objekte können im Programmquelltext mithilfe des Schlüsselwortes **new** erzeugt werden.
- Greenfoot-Akteure verwalten ihr sichtbares Bild in Form eines Objekts vom Typ **GreenfootImage**.
- **Objektvariablen** (auch **Zustandsfelder** genannt) dienen zum Speichern von Informationen (Objekte oder Werte), die später benötigt werden.
- Eine **Zuweisungsanweisung** weist einer Variablen ein Objekt oder einen Wert zu.
- Wenn ein Objekt einer Variablen zugewiesen wird, enthält die Variable eine **Referenz** auf dieses Objekt.
- Mithilfe des doppelten Gleichheitszeichens (**==**) können wir prüfen, ob zwei Dinge **gleich** sind.
- Die **if/else**-Anweisung führt ein Codefragment aus, wenn eine gegebene Bedingung wahr ist, und ein anderes Codefragment, wenn die Bedingung falsch ist.

EXKURS

1 Szenarien teilen

In diesem Abschnitt wollen wir keine neuen Programmiertechniken einführen, sondern einen kleinen Abstecher machen und besprechen, wie du das, was du entwickelt hast, mit anderen teilen kannst. Die „anderen" können dein Freund sein, der neben dir sitzt, oder ein anderer Greenfoot-Programmierer am anderen Ende der Welt. In unseren Zeiten des globalen Internets macht dies eigentlich keinen großen Unterschied mehr.

E1.1 Dein Szenario exportieren

Wenn du die Entwicklung eines Szenarios – sei es ein Spiel oder eine Simulation – abgeschlossen hast, möchtest du vielleicht auch andere daran teilhaben lassen. Diese Benutzer sollten das Spiel starten (und neu starten) können, sie benötigen jedoch keinen Zugriff auf das Klassendiagramm oder den Quelltext. Sie sollen das Spiel nicht ändern, sondern lediglich damit spielen.

In Greenfoot musst du dazu das Szenario exportieren. Hierfür steht dir der Befehl EXPORTIEREN im Menü SZENARIO zur Verfügung. Damit rufst du ein Dialogfeld auf, in dem du zwischen drei Exportoptionen wählen kannst: PROGRAMM, WEB-SEITE und PUBLIZIEREN.

E1.2 In ein Programm exportieren

Die erstgenannte Exportoption ist der Export in eine Anwendung. Eine Anwendung ist ein unabhängiges Programm, das der Benutzer lokal auf seinem Rechner ausführen kann.

Wähle hierfür in deinem EXPORTIEREN-Dialogfeld die Option PROGRAMM. Anschließend kannst du einen Speicherort und einen Namen für das ausführbare Szenario angeben, das du damit erzeugst (Abbildung E1.1).

Damit ein Programm nach dem Export wie gewünscht funktioniert, ist es wichtig, dass alle Akteure, die zu Beginn des Spiels auf dem Bildschirm angezeigt werden sollen, automatisch erzeugt werden. Der Benutzer hat keine Möglichkeit, Objekte interaktiv zu erzeugen. Das bedeutet in der Regel, dass deine Welt zum Bevölkern eine „populate"-Methode aufweisen sollte – so wie im Fall unseres Krabben-Spiels.

Copyright