EUGEN REINTJES SCHULE HAMELN	BG12 IT BG13 Wdh.	Programmieren

Programmierung

Inhalt

Greenfoot allgemein, Klassen der objektorientierten Programmierung

Greenfoot ist in drei Teile unterteilt: Welt, Klassendiagramm (siehe Konstruktor) und Greenfoot-Steuerung. Auf der Welt wird das gesamte Programm ausgeführt und ist für die Benutzer sichtbar, was passiert. Die Greenfoot-Steuerung dient zur Ausführung der verfügbaren Methoden der Klassen. Ein Klick auf Act führt alle Methoden einmal aus. Ein Klick auf Run wiederholt die Methoden so lange aus, bis sie gestoppt werden.

Datentypen

Typname	Größe[1]	Wrapper-Klasse	Wertebereich	Beschreibung
boolean	undefiniert[2]	java.lang.Boolean	true / false	Boolescher Wahrheitswert, Boolescher Typ[3]
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')	Unicode-Zeichen (UTF-16)
byte	8 bit	java.lang.Byte	-128 ... 127	Zweierkomplement-Wert
short	16 bit	java.lang.Short	-32.768 ... 32.767	Zweierkomplement-Wert
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647	Zweierkomplement-Wert
long	64 bit	java.lang.Long	-2^{63} bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$[4]	Zweierkomplement-Wert
float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38	32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen.
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308	64-bit IEEE 754, doppelte Genauigkeit

Datentypen sind für die Bestimmung von **Variablen** notwendig. Die am meisten verwendeten Datentypen sind **boolean**, **char**, **int** und **double**. Boolean ist ein **Wahrheitswert** (WAHR oder FALSCH), char steht für **Character** (Buchstaben), int steht für **Integer** (Ganzzahlen) und double steht für **Dezimalzahlen** (z. B. 3,1415…).

Wofür gibt es die anderen Datentypen? Auch byte, short und long stehen für Ganzzahlen; float steht auch für Dezimalzahlen. Der Unterschied zwischen den Datentypen ist die Anzahl der bits, wodurch sich der maximale Wertebereich der Datentypen ändert (z. B. byte = 2^8 = 256 -> -128 bis 127).

Als Beispiel hatten wir bei Snake einen Timer für die Verzögerung des automatischen Spawns eines Apfels.

Zugriffsmodifikator	Datentyp	Variablenbez.	Zuweisung	Wert
public private	int	timer1	=	0;

Da wir für die Variable Timer Ganzzahlen benötigen, verwenden wir den Datentypen int. Mittels timer1=0 weisen wir der Variable timer1 den Wert 0 zu. Wir können timer1 auch den Wert 10 zuweisen, indem wir timer1=10 schreiben.

Rechnen mit Variablen

Die objektorientierte Programmierung ermöglicht das einfach Rechnen, sowohl mit reinen Zahlen als auch das Rechnen mit Variablen. Das Ergebnis der Rechnung kann wiederum in einer bestehenden oder in einer neuen Variable gespeichert werden.

Beispiel: Variable1=Variable1+5; oder Variable2=Variable1*10;

Klassendefinition

Zugriffs-modifikator	Einleitung Klasse	(Unter) Klassenname	Unterordnung	Oberklasse
public private	class	Schlange	extends	Actor { }

Anhand dieser Schlüsselworte werden **Klassen** erstellt. *Public* ist ein **Zugriffsmodifikator**. Zugriffsmodifikatoren erlauben bzw. verhindern den Zugriff von anderen Klassen (die keine Unterklasse sind) auf diese eine Klasse. Durch *public* ist der Zugriff anderer Klassen auf Schlange(nmethoden) möglich, wie beispielsweise der Punktezähler. Ein weiterer Zugriffsmodifikator ist *private*. *Private* verhindert den Zugriff und die Sichtbarkeit dieser aktuellen Klasse durch andere Klassen.

Class leitet die Erstellung der Klasse ein. *Extends* Actor bedeutet, dass diese aktuelle Klasse eine Unterklasse der Klasse Actor ist.

Wir stellen uns analog dazu vor, dass Schulbücher als *public* definiert sind, damit jeder Schüler darauf zugreifen kann; Schülernoten als private, weil wir sonst gegen den Datenschutz verstoßen.

Zugriffsmodifaktoren

Access Levels				
Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Attributendeklaration

Wir können den **Klassen Attribute (Eigenschaften) zuweisen**. Beispielsweise ist die Schlange grün oder die Staatsangehörigkeit eines Schülers.

Zugriffsmodifikator	Attributtyp	Attributname
public private	String int	schlangenfarbe; schlangenalter;

Methodendefiniton

Zugriffsmodifikator	Rückgabetyp	Methodenname
public	void boolean int	sucheNachApfel(){Anweisung}

Wir können den Klassen **Methoden** zuweisen. Methoden sind ausführbare Operationen der Klassen, die Interaktionen der Objekte auf der Welt erlauben (Die Schlange kann sich bewegen, die Schlange kann Äpfel fressen.). Alle real platzierten Objekte der Klasse Schlange haben dieselben Methoden.

Wozu gibt es Rückgabetypen? Wir können diese unter anderem vor Methoden schreiben, damit ein Rückgabetyp entsteht. Beispielsweise boolean move(). Da Boolean ein Wahrheitswert ist, testet er, ob sich die Schlange bewegen kann. Als Rückgabe gibt es WAHR oder FALSCH. Void steht dafür, dass kein Rückgabewert erfolgt. Wir möchten nicht, dass das Spiel ständig unterbrochen wird, während die Schlange nach einem Apfel sucht.

```
1 public class Schlange extends Actor
2 {
3       private String schlangenfarbe;
4       //weitere Attribute…
5       public void sucheNachApfel()
6       {
7          Anweisung
8       }
9 }
```

Eingaben bei Methodendefinitionen

Bei der Methodendefinition können Variablen deklariert werden. Die Deklaration erfolgt in den runden Klammern hinter dem Methodennamen.

```
1 public void sucheNachApfel(int punkte, int anzahl)
2       {
3           punkte = 0;
4           anzahl = 10;
5       }
```

Alternativ kann die Wertzuweisung bei der Verwendung eines Konstruktors erstellt werden.

Klassen, Objekte, Konstruktoren

Das **Klassendiagramm** enthält bei uns unter anderem die **Klassen** „Schlange“, „Apfel“, „Kirsche“, „Stein“, … Klassen beschreiben das allgemeine Konzept einer Schlange oder eines Apfels. Fügen wir in unserer Welt eine Klasse Schlange hinzu, so wird aus der Klasse ein **Objekt Schlange** erzeugt. Wir können von der Klasse Schlange mehrere Objekte hinzufügen. Intern erhalten diese eine sequentielle Nummerierung, wie Schlange1, Schlange2, Schlange3, … Analog ist vorstellbar, dass wir in der Schule eine Klasse Schüler haben, wovon in jeder Schulklasse mehrere Objekte Schüler vorhanden sind.

Konstruktor erlauben die Erstellung einer neuen Instanz einer bestimmten Klasse, z. B. die Platzierung eines Apfels auf der Welt oder wir legen einen neuen Schüler mit Schülerakte in der Schule an. Mit dem Befehl „new“ und dem Klassennamen wird der Konstruktor abgerufen und die Initialisierung des Objektes durchgeführt. Sollen über den Konstruktor Werte zugewiesen werden (siehe letzten Teil der Methodendefinition), kann dies innerhalb der runden Klammern erfolgen.

```
1 //Beispiel Konstruktor
2 sucheNachApfel(0,10);
```

Objekt(nummer)	Zuweisung	Befehl zur Erstellung	Klassenname
schlange1	=	new	Schlange();

Verzweigungen durch Bedingungen

Verzweigungen erlauben nach einer Prüfung von Werten zu entscheiden, wie weiter verfahren wird. In der Schule gibt es den Fall, wenn ein Schüler eine gewisse Anzahl an Fünfen bzw. Unterkurse hat **(Prüfung)**, dann muss er das Schuljahr wiederholen **(if-Teil)** oder kommt in die nachfolgende Jahrgangsstufe **(else-Teil)**. Ähnlich verhält es sich bei Snake. Wenn der Timer eine bestimmte Zahl erreicht hat, spawnt er einen Apfel, ansonsten zählt er so lange weiter.

Timer1 wurde ganz oben im Quellcode dem Wert 0 zugewiesen. In der act()-Methode steht timer1++, also erfolgt bei jeder Methodenausführung eine Erhöhung des timers1 um 1. Nun wird gleichzeitig immer die **Methode spawn()** ausgeführt, die die **Verzweigung** einleitet. Erreicht der timer1 den Wert 20, ist die Prüfung erfüllt und führt die Befehle unterhalb aus.

```
public class MyWorld extends World
{
    private int timer1=0;
    /**
     * Constructor for objects of class MyWorld.
     *
     */
    public MyWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        //music1.playLoop();
        prepare();

    }
    public void act()
    {
        timer1++;
        spawn1();
    }
    private void spawn1()
    {
        if(timer1==20)
        {
            if(ApfelRichtung>=1)
            {
                addObject(new Apfel(),0,Greenfoot.getRandomNumber(400));
                timer1=0;
                ApfelRichtung=Greenfoot.getRandomNumber(2);
            }
            else
            {
                addObject(new Apfel(),300,Greenfoot.getRandomNumber(400));
                timer1=0;
                ApfelRichtung=Greenfoot.getRandomNumber(2);
            }
        }
    }
}
```

Verzweigung	Variablenprüfung	Math. Operator	Zu prüfender Wert	Befehl
if	(timer1	==	20)	{Anweisung}
else	(entfällt)	(entfällt)	(entfällt)	{Anweisung}
		>, <, ==, >=, <=		

Es können mehrere if-Verzweigungen hintereinandergeschaltet werden (siehe Bild), ohne dass ein else erfolgt. Der else-Teil muss nicht zwingend erstellt werden.

Des Weiteren können Bedingungen durch ein UND bzw. ODER wie bei Excel miteinander verknüpft werden.

```
1  //Beispiel UND-Verknüpfung
2  if(timer1 == 20 && anzahl == 10)
3       {
4           Anweisung
5       }
6  //Beispiel ODER-Verknüpfung
7  if(timer1 == 30 || anzahl == 20)
8       {
9           Anweisung
10      }
```

Beispiel für eine UND-Verknüpfung: *if (timer1==20 && anzahl==10) {Anweisung}*

Beispiel für eine ODER-Verknüpfung: *if (timer2==30 || anzahl==20) {Anweisung}*

Verzweigungen mit mehreren Bedingungen

if mit else if (Siehe „Programm1“-Klasse)

```
if(Bedingung 1)
     {
         Anweisung 1
     }
else if(Bedingung 2)
     {
         Anweisung 2
     }
else if(Bedingung 3)
     {
         Anweisung 3
     }
else
     {
         Anweisung 4
     }
```

Werden Verzweigungen mit else if gebildet, können mehrere bzw. unterschiedliche Bedingungen formuliert werden. Die Bedingungen werden von oben nach unten überprüft. Trifft also die erste Bedingung nicht zu, wird else if mit der Bedingung 2 überprüft. Falls kein if oder else if zutrifft, wird else, also Anweisung 4 ausgeführt.

Verzweigungen mit Verschachtelung von if-Statements

if-Statements können ineinander verschachtelt werden. Nur wenn die Bedingung 1 erfüllt ist, kann die verschachtelte if-Bedingung 1.1 überprüft werden.

```
if(Bedingung 1)
     {
     if(Bedingung 1.1)
          {Anweisung 1.1}
     else
          {Anweisung 1.1}
     }
else
     {Anweisung}
```

Schleifen

Schleifen sind dazu da, dass eine ständige Wiederholung (endlos) oder bis zu einem bestimmten Zustand geschieht. Bei Snake wurde die Schleife zur endlosen Durchschaltung der Animation der Schlange verwendet.

Die **Schleife** bei der Durchschaltung der Animation entsteht dadurch, dass die Variable frame beim Wert 1 startet und bei jeder erfüllten Bedingung um den Wert 1 bis zum Wert 3 erhöht wird und letztendlich in der letzten Zeile wieder auf den Wert 1 gesetzt wird. Somit ergibt sich eine Endlosschleife.

```
public class Schlange extends Actor
{
    GreenfootImage Schlange1 = new GreenfootImage("snake2.png");
    GreenfootImage Schlange2 = new GreenfootImage("snake1.png");
    GreenfootImage Schlange3 = new GreenfootImage("snake3.png");
    public int frame = 1;
    public int animierenZaehler = 0;
    /**
     * Act - do whatever the Schlange wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        animieren();
    }
    public void animieren()
    {
        if(frame == 1)
        {
            setImage(Schlange1);
            frame = 2;
        }
        else if(frame == 2)
        {
            setImage(Schlange2);
            frame = 3;
        }
        else if(frame == 3)
        {
            setImage(Schlange3);
            frame = 1;
        }
    }
```

Schleifenarten

While-Schleife (Siehe „Programm1“-Klasse)

```
1 while(zahl2>1)
2 {
3 zahl1 = zahl1 * zahl3;
4 zahl2 = zahl2 -1;
5 }
```

Die While-Schleife überprüft zuerst die Variable *„zahl2“*, ob sie größer als der Wert *„1“* ist. Trifft dies zu, dann werden die Anweisungen in der geschweiften Klammer ausgeführt. Die Schleife ergibt sich dadurch, dass dieser Programmteil so lange ausgeführt wird, bis die Prüfung nicht mehr zutrifft.

For-Schleife

```
1 for(a = 0; a<= 60; a++)
2 {
3 System.out.println(„a ist aktuell bei dem Wert“ + a);
4 }
```

Die For-Schleife besteht in der runden Klammer aus drei Teilen:

for(Initialisierung; Test bzw. Prüfung; Fortsetzung)

{

Anweisung

}

Bei der Initialisierung wird die Variable definiert. Der Test bzw. die Prüfung prüft eine bestimmte Variable auf einen bestimmten Wert. Im Gegensatz zur While-Schleife erfolgt die Fortsetzung schon während der ersten Prüfung. Also nimmt die Variable *„a“* schon beim ersten Durchgang den Wert *„1“* an.

Ausgaben

Die Ausgabe kann z. B. über *System.out.println* erfolgen. Innerhalb der runden Klammer erfolgt der Ausgabetext innerhalb der Anführungsstriche. Wird allerdings auf eine Variable zugegriffen, darf die Variable nicht innerhalb von zwei Anführungsstrichen stehen (also einleitende Anführungsstriche und Anführungsstriche zum Beenden). Vor der Variable wird ein Pluszeichen verwendet, damit die Variable interpretiert werden kann. Folgt nach der Variablen ein zusätzlicher Ausgabetext, wird dies wiederum mit einem Pluszeichen verknüpft (siehe Beispiel 2). An die Leerzeichen innerhalb der Anführungszeichen sollte gedacht werden, da dies sonst zusammengeschrieben wird.

```
1 //Beispiel 1
2 System.out.println(„Timer = “ + timer1);
3 //Beispiel 2
4 System.out.println(„Timer = “ + timer1 + „Sekunden “);
```