Programmierung mit GREENFOOT: Kurzübersicht

Wichtige Begriffe / Konzepte

Klasse

Vorlage bzw. Schablone, aus der Objekte erzeugt werden. Beispiel für eine Klasse in JAVA:

```
public class Smiley
{
}
```

Objekt

Ein Objekt ist ein konkretes Exemplar, das aus einer Klasse erzeugt wurde (z.B. ein Smiley, eine Schlange, eine Rakete…). Objekte können sich Dinge merken (*Attribute*) und können Dinge tun (*Methoden*).

Methode

Das, was die Objekte tun können (z.B. setLocation).

Beispiel für eine Methode:

```
public void act()
  {
     // Hier können wir  Befehle erteilen!
  }
```

public: Methode ist öffentlich, andere Objekte können diese Methode also aufrufen
void: Diese Methode gibt nichts zurück (void = leer)

Übergabeparameter

Manche Methoden brauchen einen Wert, damit sie wissen, was sie genau tun sollen.
Beispiel für Methode mit *zwei* Übergabeparametern: *setLocation(**10**, **20**)*

Variable

Wenn man sich etwas merken möchte (z.B. eine Zahl), dann verwendet man eine Variable, in der man die Information speichert. Vereinfacht ausgedrückt: Eine Variable ist so etwas wie eine Kiste, in der ich Dinge (hier: Informationen) aufheben kann.

Beispiel: Wir wollen uns die x-Koordinate eines Objekts merken

```
int x;
x = this.getX();
```

Attribut

Das, was sich alle Objekte einer Klasse „merken“. Präziser: Das, was wir bei allen Objekten einer Klasse speichern. Ein *Attribut* ist also eigentlich dasselbe wie eine *Variable*. Einziger Unterschied: *Attribute* werden oben in der Klasse erstellt (*Fachausdruck: deklariert*) und damit können alle Methoden auf dieses *Attribut* zugreifen. Bei *Variablen* ist das anders: Nur die Methode, in der die *Variable* erstellt wurde, kennt die *Variable*!

Beispiel: Im Attribut Munition speichern wir, wie oft wir noch schießen können

```
public class Rocket extends Actor
{
   // Attribut 'shots': Vorrat an Munition
   private int shots = 10;
```

Datentypen

Wenn man sich etwas in einem Attribut oder in einer Variablen merkt, muss man angeben, welche Art von Daten man speichern möchte (z.B. Zahlen, Buchstaben, etc.). Hier die wichtigsten Datentypen:

int	Ganze Zahlen. Den Datentyp *int* spricht man *Integer* aus.
double	Kommazahlen
String	Zeichenketten, also Buchstaben, Zahlen, Sonderzeichen…

Typumwandlung (Type Casting)

Manchmal muss man den Wert einer Variablen oder eines Attributs in einen anderen Datentyp umwandeln.

Beispiel 1: Eine Kommazahl soll in eine Ganzzahl umgewandelt werden

```
int ganzzahl;
double kommazahl = 3.0;

ganzzahl = (int) kommazahl;
```

Beispiel 2: Ein Objekt der Klasse *World* soll in ein Objekt der Klasse *MeteorWorld* konvertiert werden:

```
MeteorWorld meineWelt;
meineWelt = (MeteorWorld) this.getWorld();
```

Die getWorld-Methode liefert immer ein Objekt der Klasse World

Compiler

Wir programmieren in JAVA. Diese Programmiersprache ist so etwas wie eine Mini-Version der englischen Sprache. Die versteht der Computer aber nicht! Deshalb muss es ein Programm geben, das unseren Programmcode in eine Sprache übersetzt, die der Computer (eigentlich der Prozessor) versteht. Dieses Programm heißt *Compiler*. Der Compiler übersetzt aber nicht nur, sondern analysiert unseren Programmcode und kann uns auf Fehler hinweisen, die das Programm zum Absturz bringen würden.

Klassen, die uns GREENFOOT zur Verfügung stellt:

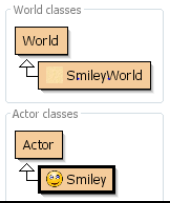
WORLD

Diese Klasse ist eine Super-Vorlage, aus der wir eigene Welten erstellen können.

ACTOR

Super-Vorlage, aus der man eigene ACTOR-Klassen erstellen kann. Aus einer solchen Klasse erstellen wir Objekte, die wir auf dem Spielfeld sehen.

Vererbung



Die Pfeile im Bild links zeigen, dass die von uns erstellte Klasse *SmileyWorld* alle Attribute und Methoden von der Klasse *World* erbt. Das gleiche gilt für die Klasse *Smiley*: Sie erbt alles von der Klasse *Actor*. In JAVA programmiert man die Vererbung so:

```
public class Smiley  extends Actor
{
}
```

Achtung: Attribute und Methoden, die *private* sind, werden nicht vererbt!

Programmierung mit GREENFOOT: Kurzübersicht

Wie funktioniert GREENFOOT?

1.	Zuerst erstellt Greenfoot die Welt
2.	Sobald man die Run-Schaltfläche anklickt, wird bei allen Objekten in der Welt nacheinander die **act-Methode** aufgerufen. Damit man sich das besser vorstellen kann, kann man sagen, dass die act-Methode jede Millisekunde aufgerufen wird. Das ist aber nur eine Metapher (also eine bildhafte Vorstellung). Die Geschwindigkeit, in der das passiert hängt nämlich auch davon ab, wieviele Objekte es in der Welt gibt (je mehr, desto langsamer).

Objekte erzeugen

Smiley meinSmiley;	Wir lassen für das zu erstellende Objekt Speicherplatz reservieren. Das Objekt existiert aber noch nicht!
meinSmiley = new Smiley();	Im zweiten Schritt erzeugen wir das Objekt. Man kann beides übrigens auch in einem einzigen Schritt machen: Smiley meinSmiley = new Smiley();

Methoden ausführen

Allgemein:	Objekt.Methode(Übergabeparameter);	
Beispiele:	this.setLocation(10, 20);	*Diese Methode hat zwei Übergabeparameter*
	this.moveLeft();	*Methode ohne Übergabeparameter*

Eigene Methoden schreiben

Eigene Methoden sehen so aus:

```
public Rückgabewert NameDerMethode()
{
}
```

Rückgabewert: Wenn eine Methode nur etwas tut und nichts zurückgibt, ist der Rückgabewert *void*.

Beispiel für eine Methode, die nicht zurückgibt:

```
public void checkCollision()
{
}
```

Der Konstruktor

Der Konstruktor ist eine besondere Methode, die genauso heißt wie die Klasse, in der der Konstruktor enthalten ist. Wird der Konstruktor aufgerufen, **wird ein Objekt der Klasse erzeugt**!

Den Konstruktor sieht man nicht immer, er ist aber trotzdem immer vorhanden: Wenn man den Konstruktor nicht selbst programmiert, erzeugt JAVA den Konstruktor automatisch im Hintergrund. Warum sollte man den Konstruktor dann programmieren? Weil man so festlegen kann, was alles passieren soll, wenn Objekte erzeugt werden!

Beispiel für einen Konstruktor, in dem nicht nur ein Objekt erzeugt wird, sondern mehr passiert:

```
public SmileyWorld()
{
   // Eine Welt mit 12x12 Feldern. Ein Feld hat die Größe 60 Pixel.
   super(12, 12, 60);

   // Die Welt mit dem Smiley, Schlangen und dem Haus (=Ziel) füllen
   this.fillWorld();
}
```

Beispiel: Kollision abfragen

Im folgenden Beispiel wird in der Klasse ROCKET ermittelt, ob ein Objekt der Klasse METEOR mit einer Rakete kollidiert ist:

```
// Speicherplatz für ein Objekt reservieren:
Actor meinMeteor;

// Das Objekt erzeugen, das sich mit mir überschneidet:
meinMeteor = this.getOneIntersectingObject(Meteor.class);
if(meinMeteor != null)
 {
    // Mach irgendwas!
 }
```

Beispiel: Ein Objekt „beschafft“ sich seine Welt

Will ein Objekt ein anderes Objekt erscheinen oder verschwinden lassen, so muss man sich zuerst die Welt „beschaffen“ mit *this.getWorld()* – Danach kann man die Methoden der Welt aufrufen.

Einfügen des Objekts meineExplosion*:*

```
this.getWorld().addObject(meineExplosion, 10, 20);
```

Ein Objekt entfernt sich selbst aus der Welt:

```
this.getWorld().removeObject(this);
```