Spiel mit Programmieren mit Java

für Anfänger

unter Linux Fedora Gnome

Schule	Johann-Conrad-Schlaun-Gymnasium
Auftragnehmer	Dean Schneider
Auftraggeber	Herr M. Schmidt
Bildungsgang	Berufliches Gymnasium Technik
Klasse	GYTU
Fach	Informatik
Bearbeitungszeitraum	18.08.2023 – 05.02.2024

Inhaltsverzeichnis

1. Erste schritte

1.1. Wichtige hinweise

Das Betriebssystem ist auf Englisch. Variablen haben Englische Namen da Deutsche Sonderzeichen nicht erlaubt sind, es wird davon ausgegangen das man Englisch Kenntnisse hat.

1.2. Benötigte Programme installieren

Programme über den App Store:

- Office z.B. LibreOffice
- IDE (Entwicklungsumgebung) z.B. NetBeans
- UML-Diagramme z.B. Dia Diagram Editor

Weitere Programme:

- Java Runtime/Development JDK

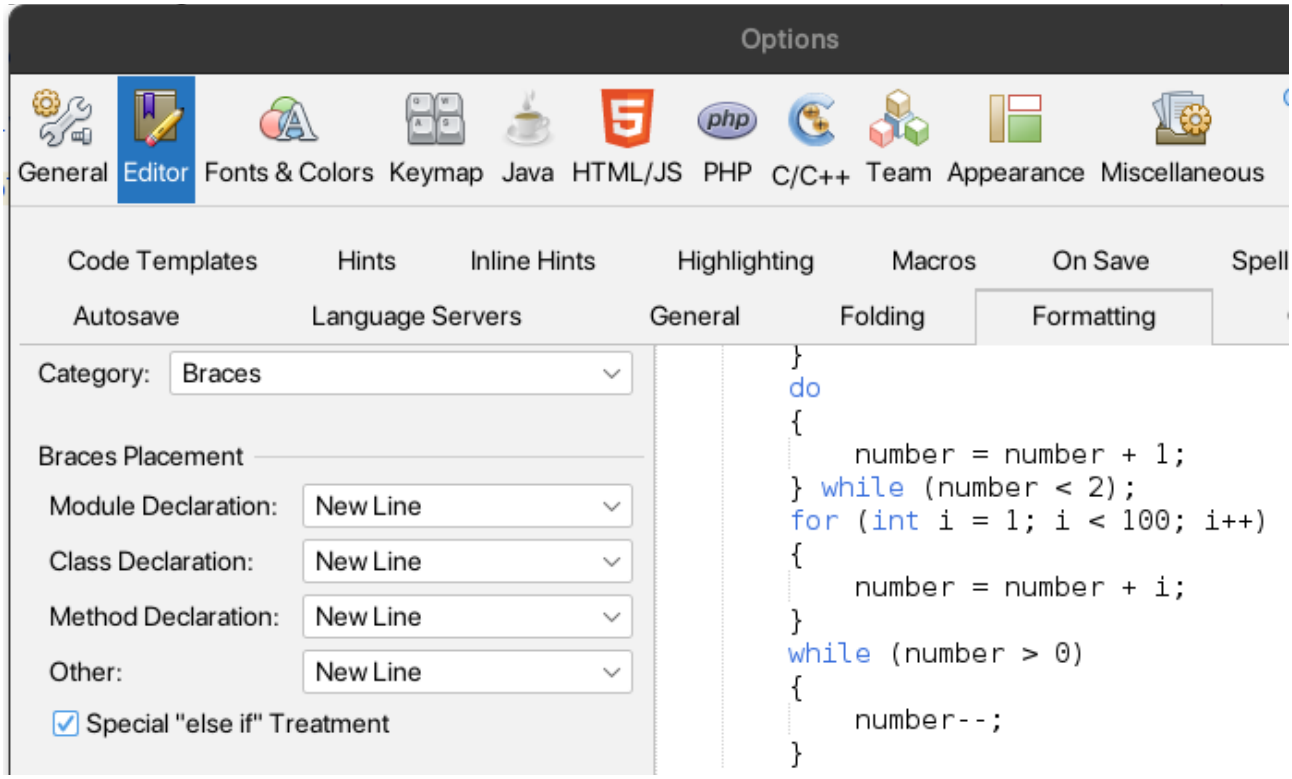
```
sudo apt update
sudo apt install default-jdk
```

1.3. NetBeans Einstellungen

Geschweifte Klammern müssen für einen übersichtlichen Code untereinander geschrieben werden.

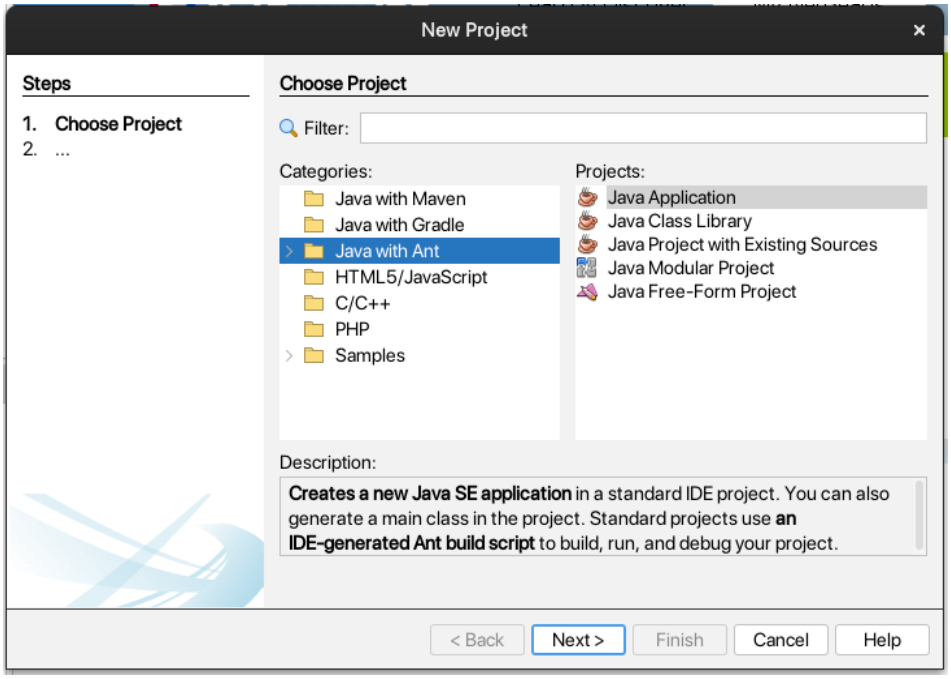
```
public class GamePanel
{
}
```

NetBeans hat die Funktion den Code mit einem Tastendruck zu formatieren, also ihn richtig einzurücken. Die automatische Code-Formatierung kann hier geändert werden:
`Tools → Options → Editor → Formatting`



1.4. Projekt erstellen

1. 

2. 

3. Project Name: Game
   Project Location: an einem guten Ort speichern
   Create main class: Ja

4. Die Main.java Datei sollte so aussehen:

```
package game;

public class Game
{
     public static void main(String[] args)
     {
          System.out.println("Hello World");
     }
}
```

   Jedes Programm beginnt bei einer „main“ also Hauptfunktion. Dieses Programm ist bereits voll funktionsfähig. Alles zwischen den geschweiften klammern {} wird ausgeführt.

5. Den Code Kompilieren und starten. Beim Kompilieren wird aus dem Java Code eine startbare Datei erstellt. In der Konsole wird nun „Hello World“ ausgeben.

2. Grundkenntnisse

In diesem Hauptkapitel werden die Grundkenntnisse von Java anhand von ausschnitten aus dem Spiel behandelt. Dadurch das das Spiel später viele Grundkenntnisse auf einmal verlangt, werden sie hier einzeln und kurz erklärt. Um es besser zu verstehen, empfiehlt es sich die Beispiele zu kopieren und mit ihnen zu experimentieren. Egal ob Java oder andere Programmiersprachen die folgenden Themen übertragen sich bis auf die Syntax[1] auf die meisten anderen Programmiersprachen.

2.1. Variablen

Variablen kennt vielleicht noch der ein oder andere aus dem Mathematikunterricht. Nicht großartig anders ist das beim Programmieren.

Initialisiert

```java
package game;

public class Game
{
	public static void main(String[] args)
	{
		System.out.println("Hello World");

		int fps = 60;
		int frames_per_minute = fps * 60;

		System.out.println("fps: " + fps);
		System.out.println("fpm: " + frames_per_minute);
	}
}
```

Ausgabe:

```
Hello World
fps: 60
fpm: 3600
```

- Code der zwischen den geschweiften Klammern **{}** steht gehört zu **main()** und wird von oben nach unten ausgeführt. Die Variablen sind nur innerhalb der Klammern **{}** verfügbar.
- Jede Zeile oder jeder Befehl wird mit einem Semikolon **;** beendet.

1 Schreibweise von

- System.out.println ist eine sogenannte Funktion, dazu später mehr. „println“ steht für „print line“ also etwas Ausgeben und einen neuen Zeilenumbruch hinzufügen. Text, Zahlen oder Variablen die in den Klammern **()** stehen werden ausgegeben.
- An **int** erkennt man eine Form von Variablen. **int** steht für „integer“, übersetzt Ganzzahl. In Java braucht jede Variable einen Datentyp. Der Datentyp gibt an, wie gespeichert wird.
- Plus, Minus, Mal und Geteilt funktionieren nach mathematischer Reihenfolge.

Deklariert

```
package game;

public class Game
{
	public static void main(String[] args)
	{
		double fps = 0.0;
		double deltaTime;
		long last = System.nanoTime();

		last -= 9000000; // 9ms

		deltaTime = (double) (System.nanoTime() - last) * 1e-6;
		fps = 1e3 / deltaTime;

		System.out.println("fps: " + fps);
	}
}
```

Ausgabe:

```
fps: 111.10833340277604
```

- Die Variable „deltaTime“ hat hier erst einen Wert, wenn man ihn zuweist, dies nennt man Deklaration ohne Initialisierung. Wichtig ist das der Wert nicht 0 ist, sondern **null** das heißt wirklich nichts.
- **double** wird verwendet, um Kommazahlen zu speichern.
- **long** wird verwendet, um besonders große Ganzzahlen speichern.
- System.nanoTime(); gibt die aktuelle System Zeit in Nanosekunden an. Dies wird verwendet um die Differenzzeit für einen Programm Durchlauf zu berechnen.
- In diesem Beispiel ist das Programm zu schnell. Durch -= wird eine simulierte Zeit in Nanosekunden abgezogen.
- // ist ein Kommentar, der Text wird von dem Programm ignoriert.

- Durch **(double)** wird **long** zu **double** umgewandelt.
- **1e-6** steht für 10^{-6} und **1e3** steht für 10^{3}.

Primitive Datentypen

Alle Variablen brauchen einen Datentyp. Abgesehen von int gibt es auch weitere, diese sind hier gelistet. Man kann sogar seine eigenen Datentypen mit Klassen erstellen. Die Zahlentypen in Java sind standardmäßig mit Vorzeichen, also positiv und negativ.

Keyword	Bits	Bytes	Beschreibung	Wertebereich
boolean	1	1	wahr oder falsch, **true** oder **false**	0 oder 1
byte	8	1	ein Byte	-128 bis 127
char	16	2	ein Buchstabe	
short	16	2	ganze Zahlen	-32768 bis 32767
int	32	4	Integer - positive und negative ganze Zahlen	-2 147 483 648 bis 2 147 483 647
long	64	8	Für besonders große positive und negative ganze Zahlen	2^{64}
double	64	8		Geeignet für 6 Nachkommastellen
float	32	4	Die gängige Variante um Kommazahlen darzustellen	Geeignet für 15 Nachkommastellen
void	0	0	Unbestimmter Inhalt oder keine Rückgabe	

2.2. Abzweigungen

Ab diesem Kapitel zeige ich nicht mehr den „public class Game“ Code, er muss allerdings trotzdem in Ihrem Programm stehen.

If-Anweisungen

„if“ bedeutet übersetzt „wenn“. Wenn die Bedingung Wahr ist, führe den dazu gehörigen Block aus.

```
public static void main(String[] args)
{
     boolean key_up = true;

     if (key_up)
     {
          System.out.println("Spieler geht nach oben");
     }
}
```

- Der Variablentyp **boolean** hält entweder **true** oder **false**, also wahr oder falsch.
- Im Spiel wird die Variable „key_up“ automatisch geändert, je nachdem ob die Taste für nach oben gedrückt ist oder nicht.
- Eine If-Anweisung funktioniert nur mit **boolean**.
- Variablen die innerhalb der **if** Klammern **{}** erstellt wurden, sind nur innerhalb der Klammern **{}** verfügbar. Dies nennt man auch Scope.

```
public static void main(String[] args)
{
     boolean spieler_tot = false;
     int x = 0;
     int bounds_left = 1;

     if (!spieler_tot && x < bounds_left)
     {
          x = bounds_left;
     }

     System.out.println("horizontale Spieler Position x: " + x);
}
```

Ausgabe:

```
horizontale Spieler Position: 1
```

- Ein ähnlicher Code wird später benutzt um den Spieler innerhalb der Spielfeld Grenzen zu halten.
- Mit „spieler_tot“ wird im Spiel angegeben ob der Spieler Tot ist, dann müssen keine Überprüfungen mehr gemacht werden.
- Mit einem Ausrufezeichen **!** wird die **boolean** umgekehrt.
- Um mehrere Abfragen in einem zu machen, wird **&&** für „und“ und || für „oder“ verwendet.

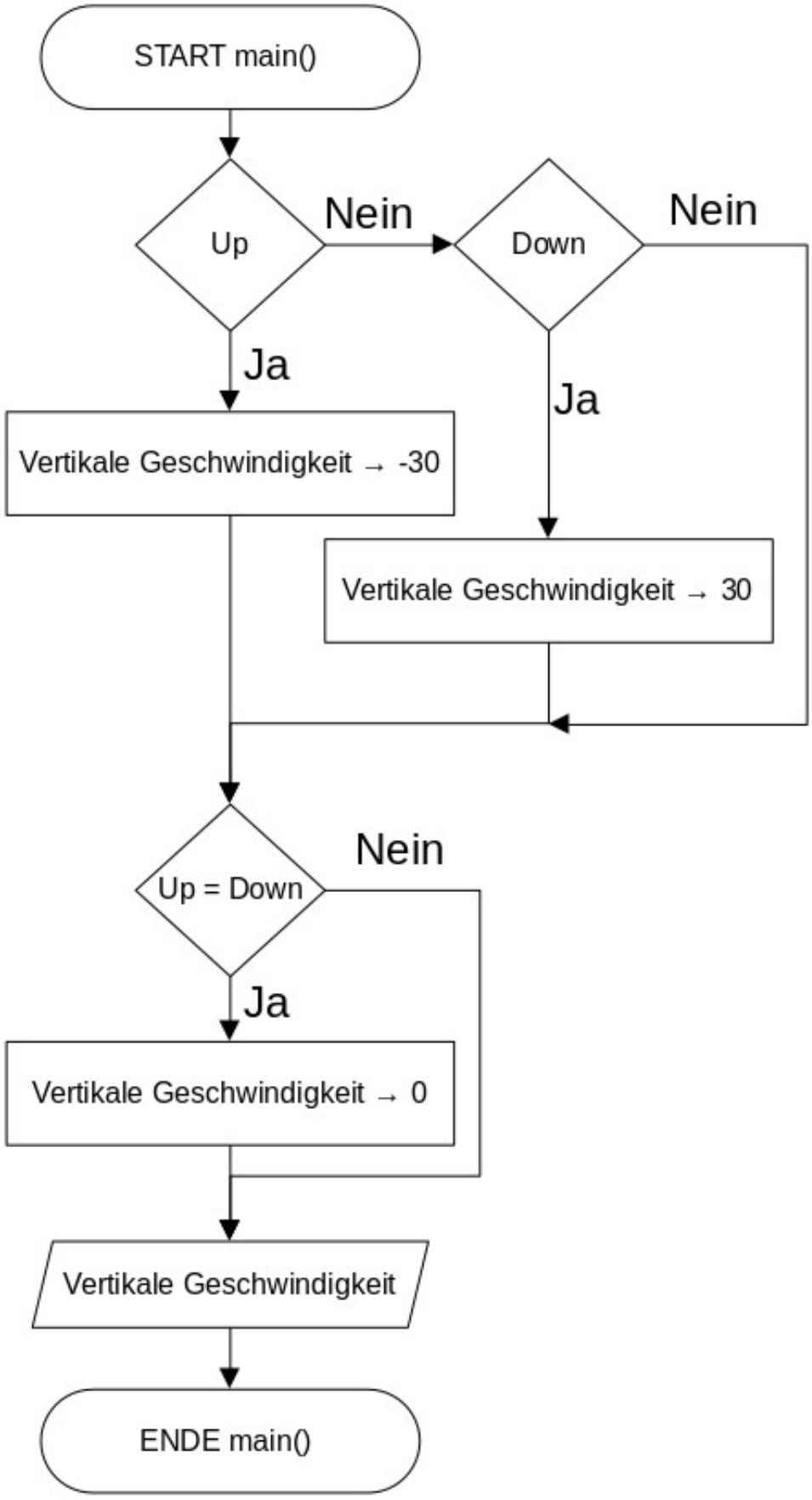
Else-Anweisung

```
public static void main(String[] args)
{
     boolean key_up = true;
     boolean key_down = false;
     double y_velocity = 0.0;
     double speed = 30.0;

     if (key_up)
          y_velocity = -speed;
     else if (key_down)
          y_velocity = speed;

     if (key_up == key_down)
          y_velocity = 0.0;

     System.out.println(y_velocity);
}
```

Ausgabe:

```
-30.0
```

- **if** kann auch ohne Klammern {} benutzt werden.
- Wenn die Bedingung nicht Wahr ist, wird **else** aufgerufen.
- Die Anweisung **else** kann optional auch mit einer weiteren **if**-Abfrage kombiniert werden.
- Wenn beide Tasten gedrückt sind oder beide nicht gedrückt sind soll die Geschwindigkeit 0.0 sein, mit anderen Worten, wenn beide gleich == sind.
- Beim Vergleichen von Zahlen und dem Verwenden von Kleiner gleich oder Größer gleich ist es wichtig <= zu schreiben und nicht =<.


START main()
Up
Nein
Down
Nein
Ja
Ja
Vertikale Geschwindigkeit → -30
Vertikale Geschwindigkeit → 30
Up = Down
Nein
Ja
Vertikale Geschwindigkeit → 0
Vertikale Geschwindigkeit
ENDE main()

Switch case

Wenn **switch** möglich ist sollte es verwendet werden, da es schneller und übersichtlicher ist als **if**.

```
public static void main(String[] args)
{
     int taste = 1;

     switch (taste)
     {
     case 0:
          System.out.println("Taste W wurde gedrückt");
          break;
     case 1:
          System.out.println("Taste S wurde gedrückt");
          break;
     default:
          System.out.println("Unbekannte Taste");
     }
}
```

Ausgabe:

```
Taste S wurde gedrückt
```

- **case** funktioniert nur in **switch** und bedeutet nichts anderes als „if (0 == taste)“.
- Um **switch** zu verlassen wird **break** benutzt.
- **default** wird aufgerufen, wenn kein **case** zu getroffen hat. Hier wird kein **break** benötigt da der **switch** Block zu ende ist.

2.3. Schleifen

Schleifen für den selben Code mehrmals aus.

While

```
public static void main(String[] args)
{
     boolean is_window_visible = true;
     int i = 0;
     boolean game_over = false;

     while (is_window_visible)
     {
          if (i == 3)
          {
               break;
          }

          if (game_over)
          {
               System.out.println("Spiel wird beendet");
               is_window_visible = false;
          }

          System.out.println("Bewegung würde berechnet werden");

          i++;
     }

     System.out.println("Loop vorbei");
}
```

Ausgabe:

```
Bewegung würde berechnet werden
Bewegung würde berechnet werden
Bewegung würde berechnet werden
Loop vorbei
```

- Die Schleife **while** läuft so lange bis der boolean Wert **false** ist.
- Durch **break** kann aus der Schleife ausgebrochen werden.

- **i++** ist dasselbe wie **i += 1**. Es gibt noch **++i**. Zwischen **i++** und **++i** besteht ein Unterschied, wenn allerdings der Operator Einzelt verwendet wird also wie in dem Beispiel, ist der Unterschied irrelevant.

For

```
public static void main(String[] args)
{
     boolean is_window_visible = true;
     boolean game_over = false;

     for (int i = 0; i < 3; i++)
     {
          System.out.println("Bewegung würde berechnet werden");

          if (game_over)
          {
               System.out.println("Spiel wird beendet");
               break;
          }
     }

     System.out.println("Loop vorbei");
}
```

Ausgabe:

```
Bewegung würde berechnet werden
Bewegung würde berechnet werden
Bewegung würde berechnet werden
Loop vorbei
```

- **for** enthält in der Regel eine Laufvariable wie im Beispiel.
- Sie hat drei Teile welche durch Semikolons ; getrennt sind.
  1. Wird einmal zu beginn ausgeführt.
  2. Wird vor jedem Durchlauf ausgeführt und wenn **false** dann wird die Schleife beendet.
  3. Wird nach der Durchlauf der Schleife ausgeführt.

2.4. Arrays

Arrays halten mehrere Variablen des gleichen Datentyps. Man könnte ein Array auch Liste nennen.

Bekannte Größe (Standard)

```
public static void main(String[] args)
{
     int picCount = 4;
     char[] pics = new char[picCount];

     pics[0] = '-';
     pics[1] = '/';
     pics[2] = '-';
     pics[3] = '\\';

     for (int i = 0; i < pics.length; i++)
     {
          System.out.println(pics[i]);
     }
}
```

Ausgabe:

```
-
/
-
\
```

- Der Datentyp **char** hält ein Zeichen.
- Im Spiel werden anstatt von Zeichen, Bilder in dem Array gespeichert.
- Mit **[x]** wird auf das Element x zugegriffen.
- Um ein „\“ darzustellen muss in Java „\\“ verwendet werden.
- pics.length = picCount

Unbekannte Größe (Vector)

```
package game;

import java.util.Vector;

public class Game
{
     public static void main(String[] args)
     {
          Vector<String> actors = new Vector<String>();

          actors.add("Spieler");
          actors.add("Gegner 1");
          actors.add("Gegner 2");

          for (String it : actors)
          {
               System.out.println("Verarbeite: " + it);
          }
     }
}
```

Ausgabe:

```
Verarbeite: Spieler
Verarbeite: Gegner 1
Verarbeite: Gegner 2
```

- Um Code von anderen zu benutzen wird **import** verwendet, es ist beim Programmieren üblich Code von anderen zu benutzten. In diesem fall wird von Standard Java importiert.
- **Vector** ist beim Erstellen generell langsamer als Standard Arrays.
- **String** ist ein Datentyp der eine Zeichenkette hält.
- Diese besondere Form der **for**-Schleife geht durch jedes Element des Arrays. „it“ ist eine normale Variable und in diesem Fall ist es eine Referenz das heißt Änderungen an dem Element von „it“ sind auch im Vector geändert.

2.5. Klassen

Mit Klassen können eigene Datentypen mit Hilfe von primitiven Datentypen erstellt werden. String oder Vector sind zum Beispiel Klassen die jemand erstellt hat.

Sprite.java

```
package game;

public class Sprite
{
     public double x;
     public double y;
}
```

Game.java

```
package game;

public class Game
{
     public static void main(String[] args)
     {
          Sprite sprite = new Sprite();
          sprite.x = 3.0;
          sprite.y = 0.0;

          System.out.println("x Position: " + sprite.x);
          System.out.println("y Position: " + sprite.y);
     }
}
```

Ausgabe:

```
x Position: 3.0
y Position: 0.0
```

- In Java braucht jede Klasse ihre eigene Datei und die Datei hat denselben Namen wie die Klasse mit der Endung .java.
- Die Klasse Sprite ist wie eine Blaupause, die immer wieder verwendet werden kann.
- Die heraus kommenden Variable nennt man Objekt.

Konstruktoren und Final

Sprite.java

```java
package game;

public class Sprite
{
	public double x;
	public double y;
	public final double scale;

	public Sprite(double x, double y, double scale)
	{
		this.x = x;
		this.y = y;
		this.scale = scale;
	}
}
```

Game.java

```java
package game;

public class Game
{
	public static void main(String[] args)
	{
		Sprite cloud = new Sprite(3.0, 0.0, 2.0);

		// error: sprite.scale = 1.0;
		System.out.println("Scale: " + cloud.scale);
	}
}
```

- **final** kann nur initialisiert erstellt werden und ist danach nicht änderbar.
- Der Konstruktor ist spezielle Form von Funktionen die im nächsten Kapitel behandelt werden. Der Konstruktor kann genau wie **main()** Code ausführen und wird immer ausgeführt beim Erstellen des Objektes.
- Die Parameter Variablen in den Klammern **()** des Konstruktors sind optional.
- Da die Parameter Variablen den selben Namen haben wie die Variablen in der Klasse muss **this** verwendet werden, **this** ist eine Referenz zu dem aktuellen Objekt.

Package

Pakete ermöglichen es viele Klassen besser in Ordnern zu organisieren.

structs/Bounds.java

```java
package structs;

public class Bounds
{
	public double top;
	public double bottom;
	public double left;
	public double right;
}
```

game/Game.java

```java
package game;

import structs.Bounds;

public class Game
{
	public static void main(String[] args)
	{
		Bounds bounds = new Bounds();
		// ...
	}
}
```

- Wenn die Klasse außerhalb des aktuellen Package liegt, wird import benötigt.

2.6. Funktionen

Funktionen oder auch Methoden genannt, ermöglichen Wiederverwertung von Code.

Einfache Funktion und static

Eine Funktion wird immer benutzt, die **main()** Funktion.

```
package game;

import structs.Bounds;

public class Game
{
	int cloud = 0;

	void init()
	{
		cloud = 1;
		// ...
	}

	public static void main(String[] args)
	{
		Game game = new Game();
		game.init();

		game.cloud = 0;
		game.init();

		System.out.println(game.cloud);
	}
}
```

Ausgabe:

```
1
```

- **main()** ist eine sogenannte statische Funktion, das bedeutet sie gehört zwar der Klasse „Game“ an allerdings hat sie kein Zugriff auf die Elemente, außer diese sind auch **static**.
- **void name()** signalisiert eine Funktion.
- Funktion Namen werden genau wie Variablen kleingeschrieben.

Übergabeparameter

```
package game;

public class Game
{
     boolean key_up;
     boolean key_left;
     boolean key_down;
     boolean key_right;

     void update_keys(int key, boolean pressed)
     {
          switch (e.getKeyCode())
          {
          case 0:
               key_up = pressed;
               break;
          case 1:
               key_left = pressed;
               break;
          case 2:
               key_down = pressed;
               break;
          case 3:
               key_right = pressed;
               break;
        }
    }

     public static void main(String[] args)
     {
          Game game = new Game();
          game.update_keys(0, true);
     }
}
```

- Alle beim Erstellen der **boolean** Werte wurde eine verkürzte Schreibweise verwendet.
- Diese Funktion wird später im Spiel verwendet, in dem Zusammenhang wird der Nutzen deutlicher.

Rückgabewert

Sprite.java

```
package game;

public class Sprite
{
	public double x;
	public double y;
	public double radius;

	public Sprite(double x, double y, double radius)
	{
		this.x = x;
		this.y = y;
		this.radius = radius;
	}

	public double distance(Sprite to)
	{
		double a = to.x - x;
		double b = to.y - y;

		return Math.sqrt(a * a + b * b) - radius - to.radius;
	}
}
```

- Ohne Rückgabe Wert würde dort anstatt von **double distance()** → **void distance()** stehen.
- Jede Funktion wird durch **return** beendet, doch bei **void** Funktionen kann der return Befehl weggelassen werden, ähnlich wie bei Mathematik das Mal-Zeichen steht es aber trotzdem da. Mit dieser Schreibweise gibt wird etwas aus der Funktion zurückgegeben.
- Konstruktoren sind immer **void** Funktionen.

Game.java

```
package game;

public class Game
{
	public static void main(String[] args)
	{
		Sprite player = new Sprite(10.0, 10.0, 5.0);
		Sprite enemy = new Sprite(6.0, 6.0, 5.0);

		double d = player.distance(enemy);
		System.out.println("Distanz: " + d);

		if (d <= 0.0)
		{
			System.out.println("Gegner wurde getroffen!");
		}
	}
}
```

Ausgabe

```
Distanz: -4.343145750507619
Gegner wurde getroffen!
```

- Sowohl der Spieler als auch der Gegner haben einen Radius, die XY Koordinaten sind mittig in diesem Beispiel.
- Wenn die Distanz im Minusbereich liegt berühren sie sich. Im Spiel bedeutet das Game over.

Public vs. Private

Sprite.java

```
package game;

public class Sprite
{
     public double x_velocity;
     private int current_pic = 0;
     private int pics_length = 4;

     void compute_animation()
     {
          current_pic++;

          if (current_pic >= pics_length)
          {
               current_pic = 0;
          }
     }
}
```

Game.java

```
package game;

public class Game
{
     public static void main(String[] args)
     {
          Sprite cloud = new Sprite();
          cloud.x_velocity = 2.0;
          // error: current_pic++;

           cloud.compute_animation();
     }
}
```

- Elemente die **private** sind, sind nur innerhalb der Klasse verfügbar.
- Elemente die keinen Zugriffsmodifikator haben, sind nur innerhalb des selben Packages verfügbar.
- public ist auch außerhalb des Packages verfügbar.

Try und catch

```
package game;

import java.io.File;
import java.io.IOException;

public class Game
{
     public static void main(String[] args)
     {
          File pic = new File("some_picture.png");
          BufferedImage source = null;

          try
          {
               source = ImageIO.read(pic);
          }
          catch (IOException e)
          {
               System.out.println("Couldn't read the picture");
               return;
          }

          // ...
     }
}
```

- Es gibt viele Möglichkeiten für einen Fehler, in diesem Beispiel geht es um das Lesen eines Bildes. Falls die Datei nicht existiert oder aus einem anderen Grund nicht geöffnet werden kann wirft die Funktion ImageIO.read() einen Fehler. Damit das Programm nicht abstürzt benutzt man **try** und **catch**.
- Im Fall eines Fehlers im **try** Block wird sofort in den **catch** Block gesprungen.
- Nur im Fall eines Fehlers wird der **catch** Block aufgerufen.
- In IOException ist der Fehler gespeichert, IOException ist ein Fehlertyp speziell für das Dateisystem.
- **return** verlässt die main() Methode und damit auch das Programm vorzeitig.

3. Spiel Programmierung

3.1. Fenster Vorbereitung

Es muss eine weitere Datei mit dem Symbol oben links erstellt werden. Alles bleibt bei den Standard-Einstellungen bis auf den Namen. Der Klassen Name ist „GamePanel“.

```
package game;

public class Game
{
     public static void main(String[] args)
     {
          // Macht die Anzeige des Fensters Flüssig
          System.setProperty("sun.java2d.opengl", "true");

          System.out.println("Working Directory: " +
               System.getProperty("user.dir"));
          new GamePanel(800, 600);
     }
}
```

```java
package game;

public class GamePanel extends JPanel implements Runnable
{
	JFrame frame;

	double fps = 0.0;
	double deltaTime;

	public GamePanel(int w, int h)
	{
		this.setPreferredSize(new Dimension(w, h));
		this.setBackground(new Color(89, 108, 171, 255));
		frame = new JFrame("Fluffy");
		frame.setLocation(100, 100);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.add(this);
		frame.addKeyListener(this);
		frame.pack();
		frame.setVisible(true);

		init();

		Thread t = new Thread(this);
		t.start();
	}
}
```

3.2. Spieler anzeigen

3.3. Spieler Bewegung

3.4. Gegner spawnen

3.5. Kollision

~~4. Spickzettel~~

4.1. Symbole

;	Befehl ende – beendet ein Befehl
//	Kommentar – Diese Zeile wird vom Programm ignoriert
/** **/	Kommentar Block – Kommentar über mehrere Zeilen
{}	Block/Scope – In dem Scope erstellte Variablen werden bei dem Ende gelöscht
[]	Array – Liste von Elementen

4.2. Keywords

public	Alle im selben package haben Zugriff.
private	Nur im aktuallen Scope.
protected	Wird nicht vererbt.
static	Der Wert ist Objekt übergreifend.
package	Verhindert Namen dopplung.
new	Initialisiert ein Objekt im Speicher.
try und catch	Fängt Fehler ab damit das Programm nicht abstürzt.
final	Wert kann nach der Initialisierung nicht mehr geändert werden.
while	Schleife sollange die Frage Wahr ist.
for	Schleife sollange die Frage Wahr ist und eine Laufvariabele kann in der Klammer gesetzt werden.
super	Greift auf die Mutterklasse zu.
import	Importiert classen aus einem Package
extends	Erbt die Funktionen und Variabelen aus einer Klasse.
implements	Verwendet ein Interface in der Klasse.
return	Gibt einen Wert aus der Funktion zurück und beendet die Funktion.
this	Eine Verknüpfung zu dem aktuellen Objekt.

4.3. Datentypen

Keyword	Bits	Bytes	Beschreibung	Wertebereich
boolean	1	1	wahr oder falsch, true oder false	0 oder 1
byte	8	1	ein Byte	0 bis 255
char	16	2	ein Buchstabe	
String			ist eigentlich kein Datentyp. Array von Buchstaben.	
short	16	2	ganze Zahlen	-32768 bis 32767
int	32	4	Integer - positive und negative ganze Zahlen	-2 147 483 648 bis 2 147 483 647
long	64	8	Für besonders große positive und negative ganze Zahlen	2^{64}

double	64	8	Zwei ints um eine Kommazahl darzustellen	Geignet für 6 Nachkommastellen
float	32	4	Die gängige Variante um Kommazahlen darzustellen	Geignet für 15 Nachkommastellen
void	0	0	Kein Inhalt oder keine Rückgabe	

Die Zahlentypen in Java sind standardmäßig mit Vorzeichen, also positiv und negativ.