Frank Roth (frankred@web.de) - letzte Aktualisierung: 02.04.2011

Slick Tutorial – Einstieg in die 2D Spiele Entwicklung

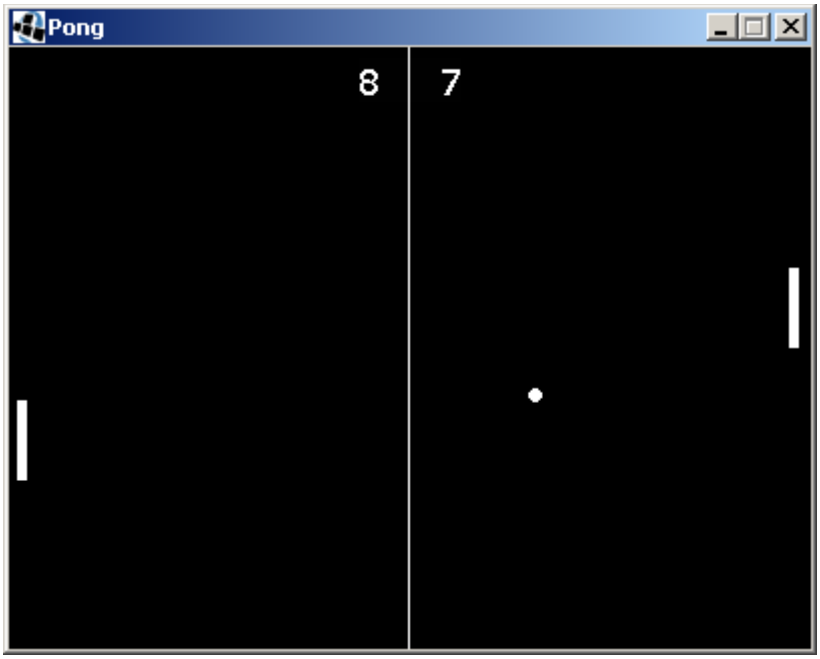
Einführung

Was ist Slick?

Slick ist eine Java Spiele Bibliothek die auf LWJGL(Lightweight Java Game Library) basiert. LWJGL bietet Schnittstellen um die Grafikbibliotheken OpenGL (Open Graphics Libary) und OpenAL(Open Audio Libary) zu verwenden.

Lernziel

In diesem „Tutorial“ sollt ihr den Umgang mit Slick, anhand eines Pong-Klons, erlernen um somit kleinere 2D Spiele selbst realisieren zu können.

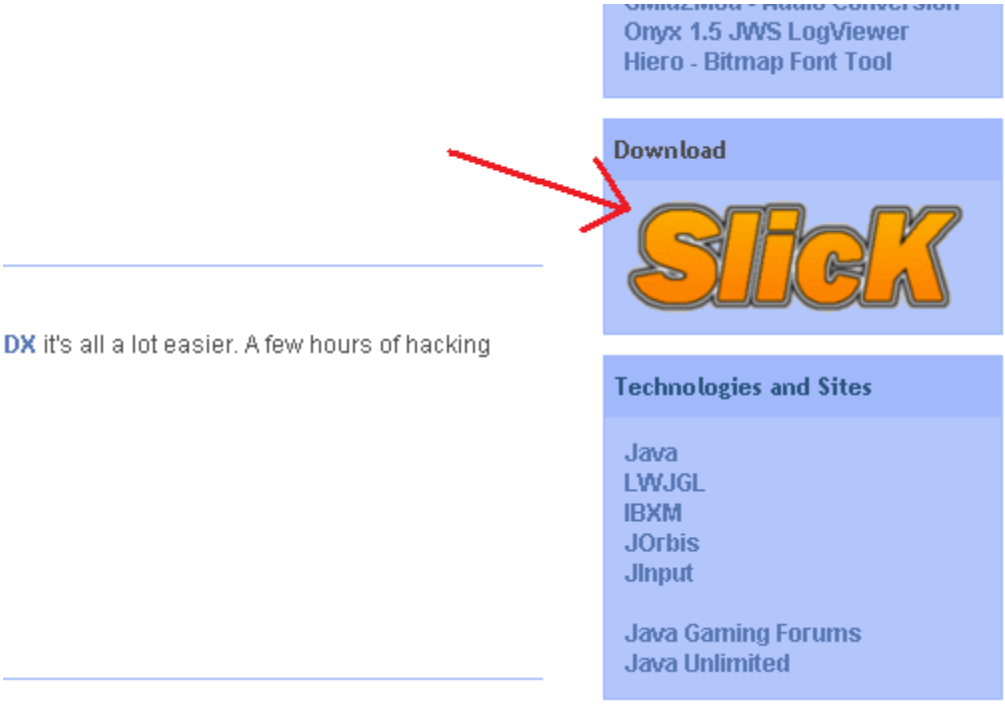


1 Screenshot des fertigen Pong-Klon

Slick einrichten

Slick herunterladen

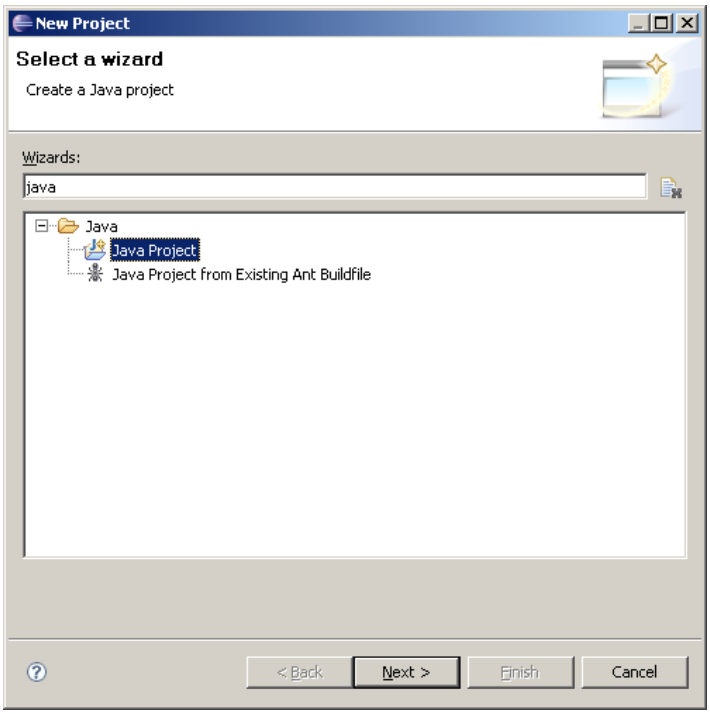
Auf http://slick.cokeandcode.com/ kann Slick heruntergeladen werden. Das Zip Archiv beinhaltet alles was man benötigt um sofort loszulegen.



2 Startseite mit Download Hinweis

Eclipse Projekt erstellen

- Neues Java Projekt erstellen

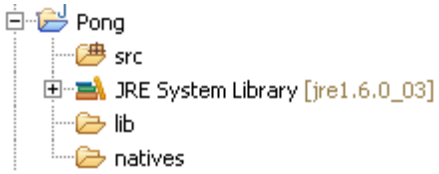


3 Neues Java Projekt

Bibliotheken einbinden

Java Bibliotheken kopieren

Verzeichnis „**lib**“ für die Slick Bibliotheken und das Verzeichnis „**natives**“ für die plattformabängigen (zum Beispiel *.dll – Dateien für Windows) Bibliotheken erstellen.



Anschließend entpackt ihr die heruntergeladene „slick.zip“ – Datei. Im entpackten Archiv befindet sich der Ordner „lib“. Folgende Jar-Archive müsst ihr in euer „lib“ Verzeichnis im Projekt kopieren.

- slick.jar
- lwjgl.jar

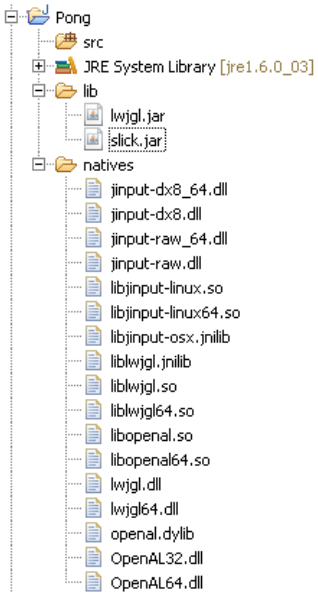
Native Bibliotheken kopieren

Im entpackten Archiv im Verzeichnis „lib“ befinden sich drei Jar-Dateien mit nativen Bibliotheken die wir ebenfalls benötigen:

- natives-linux.jar
- natives-mac.jar
- natives-win32.jar

Diese Jar-Archive müsst ihr in den Ordner „natives“ im Projekt entpacken. Schließlich sollte euer Projekt so aussehen:

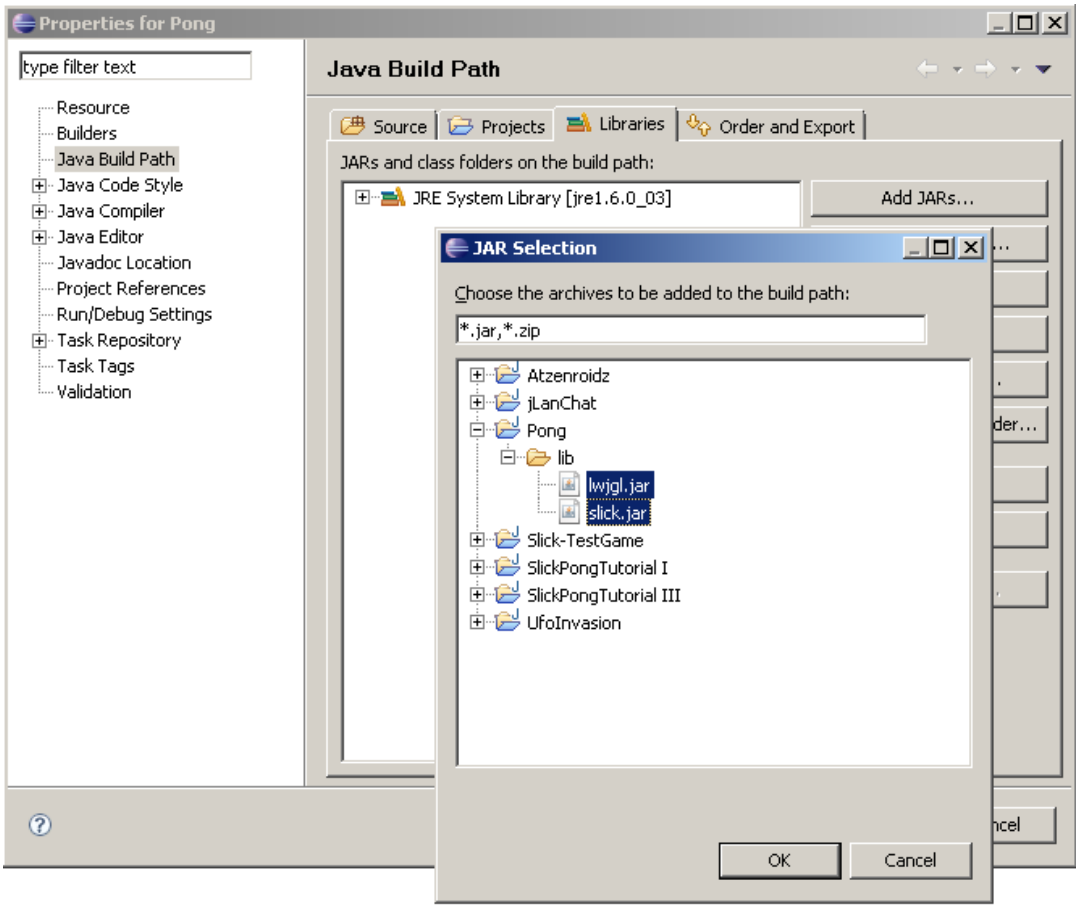
Bibliotheken ins Projekt einbinden



4 Projektverzeichnis

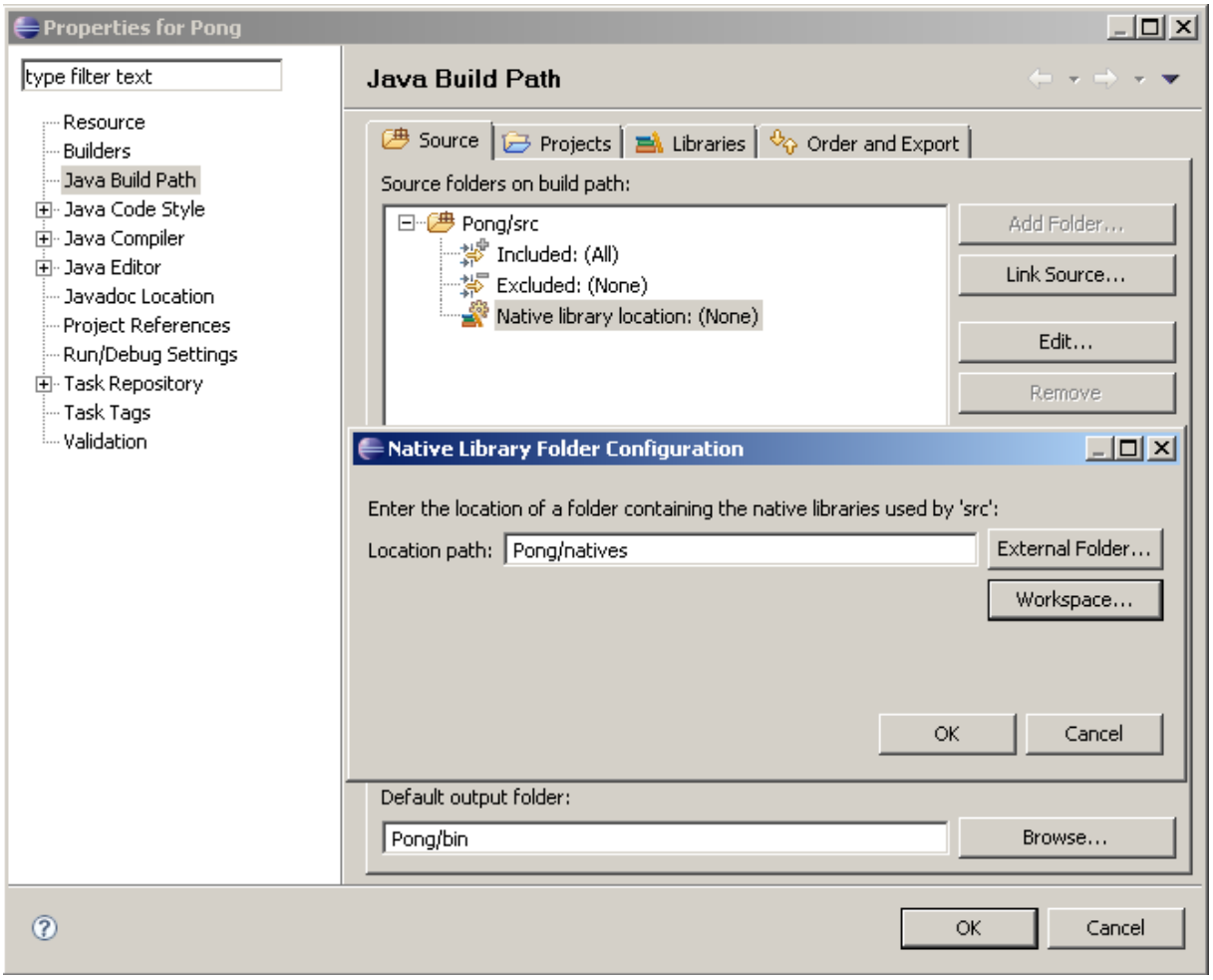
Das ganzen Bibliotheken müssen jetzt noch ins Projekt eingebunden werden:

- Rechtsklick aufs Projekt -> Java Build Path -> Libraries -> Add JARs…
- lwjgl.jar und slick.jar auswählen



5 Bibliotheken einbinden

- Rechtsklick aufs Projekt -> Java Build Path -> Libraries -> Source
- Aufklappen und “Native library location” editieren
- Pfad “natives” angeben



6 Native Bibliotheken einbinden

Grundaufbau einer Slick Anwendung

AppGameContainer

PongGame extends BasicGame

```
// Initialisierungsmethode wird nur einmal beim Start ausgeführt
public void init(GameContainer gc) throws SlickException {
    // Zum Beispiel: Spielfeld erzeugen, Spieler erzeugen ...
}

// Updatemethode wird immer wieder in einem bestimmtem Zeitabstand
ausgeführt
public void update(GameContainer gc, int delta) throws SlickException {
    // Zum Beispiel: Berechnung der Flugbahn des Balles, Tasten abfangen und
}   verarbeiten

// Grafisches rendern der Spielobjekte, wird so oft es geht ausgeführt
public void render(GameContainer gc, Graphics g) throws SlickException {
   // Zum Beispiel: Draw-Methoden von Linien, Polygonen, Kreisen, Bilder ...
}
```

7 Logischer Grundaufbau einer Slick Anwendung

Der AppGameContainer

Im AppGameContainer lassen sich **Einstellungen** über unser eigentliches **Spiel** vornehmen. Hier eine kleine Auswahl der wichtigsten Methoden des Containers.

Maximale FPS Anzahl setzen definieren	`setTargetFrameRate(60);`
Darstellung der aktuellen FPS	`setShowFPS(true);`
Vollbildmodus aktivieren/deaktivieren	`setFullscreen(true);`
VSync aktivieren/deaktivieren	`setVSync(true);`
Auflösung defnieren	`setDisplayMode(400, 300,false);`

Pong-Klon

Grundgerüst

Das Grundgerüst unserer ersten Slick Anwendung sieht wie folgt aus.

```
import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.BasicGame;
import org.newdawn.slick.GameContainer;
import org.newdawn.slick.Graphics;
import org.newdawn.slick.Input;
import org.newdawn.slick.SlickException;

public class PongGame extends BasicGame {

	public static final int WIDTH = 400;
	public static final int HEIGHT = 300;

	public PongGame() {
		super("Pong");
	}

	public void init(GameContainer gc) throws SlickException {

	}

	public void update(GameContainer gc, int delta) throws SlickException {
		// Abfangen der Eingabegaräte
		Input input = gc.getInput();
	}

	public void render(GameContainer gc, Graphics g) throws SlickException {

	}

	public static void main(String[] args) throws SlickException {
		AppGameContainer pong = new AppGameContainer(new PongGame());
		pong.setDisplayMode(WIDTH, HEIGHT, false);
		pong.setVSync(true);
		pong.setShowFPS(false);
		pong.start();
	}
}
```

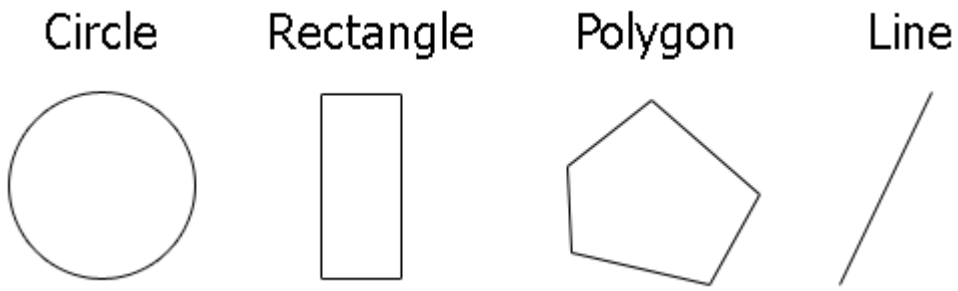
Da der „GameContainer“ unsere Eingabe von der Tastatur entgegennimmt und wir diese Eingabe später verarbeiten wollen legen wir uns in der Methode „update“ ein Input Objekt an. Mittels diesem können wir dann einfach entsprechende Benutzereingaben auswerten. Mehr dazu später.

Geometrien

Geometrische Formen

Die Hauptelemente beim Pong-Spiel sind die Bewegungen der Spieler repräsentiert durch einen rechteckigen Balken. Genau diesen wollen wir nun erzeugen. Slick bietet hier genauso wie „AWT“ verschiedene Geometrische Formen, repräsentiert durch Klassen, zur Auswahl an. Diese Objekte bieten eine Vielzahl von Methoden die uns später sehr hilfreich sein werden. So kann man zum Beispiel mit der Methode „intersect“ überprüfen ob sich zwei Geometrische Objekte in einem Koordinatenfeld schneiden.

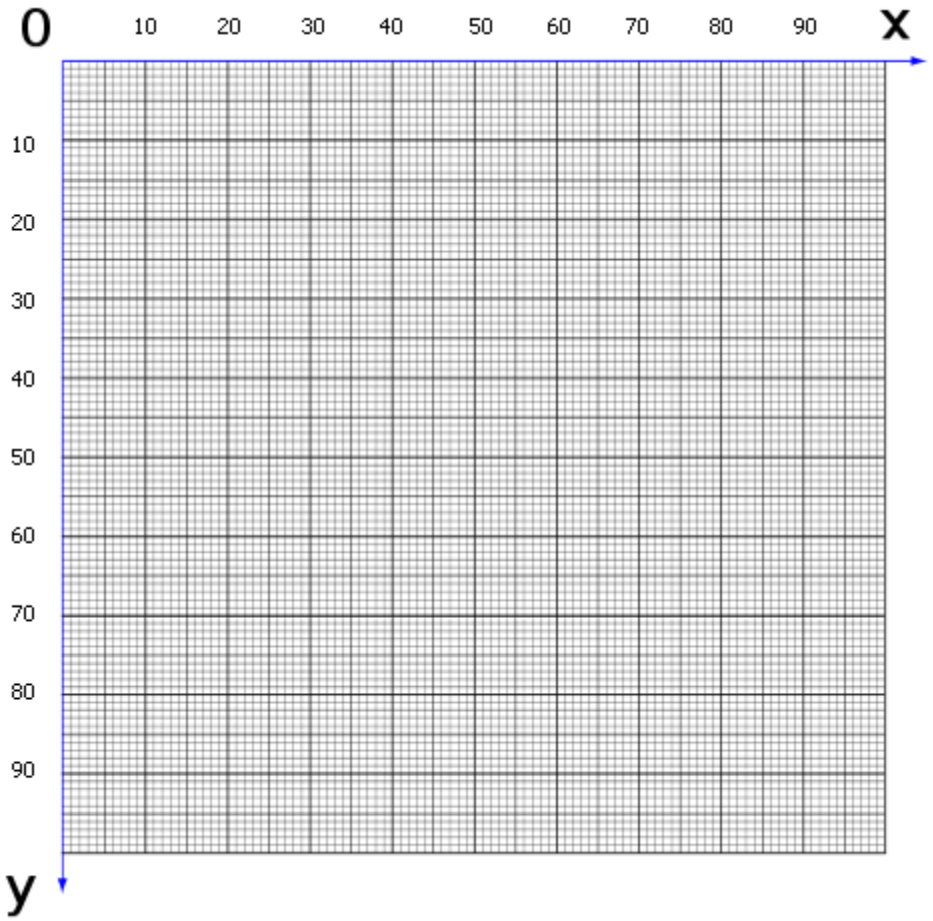
Shapes



8 Slick Geometrische Formen

Das Koordinatensystem

Das Koordinatensystem funktioniert in der Computerwelt ein wenig anders. Der Ursprung befindet sich, auf ein Fenster, bezogen links oben. Im Vergleich zum gewöhnlichen Koordinatensystem ist dieses an der X-Achse gespiegelt. Also nach unten wird der Wert größer und nach oben kleiner.



9 Koordinatensystem bei Slick

Objektmodell

Player

Ein Spieler (Player) wird repräsentiert durch das „Player“ Objekt. Dieses beinhaltet lediglich ein **Rechteck** und den **aktuellen Punktestand** des Spielers.

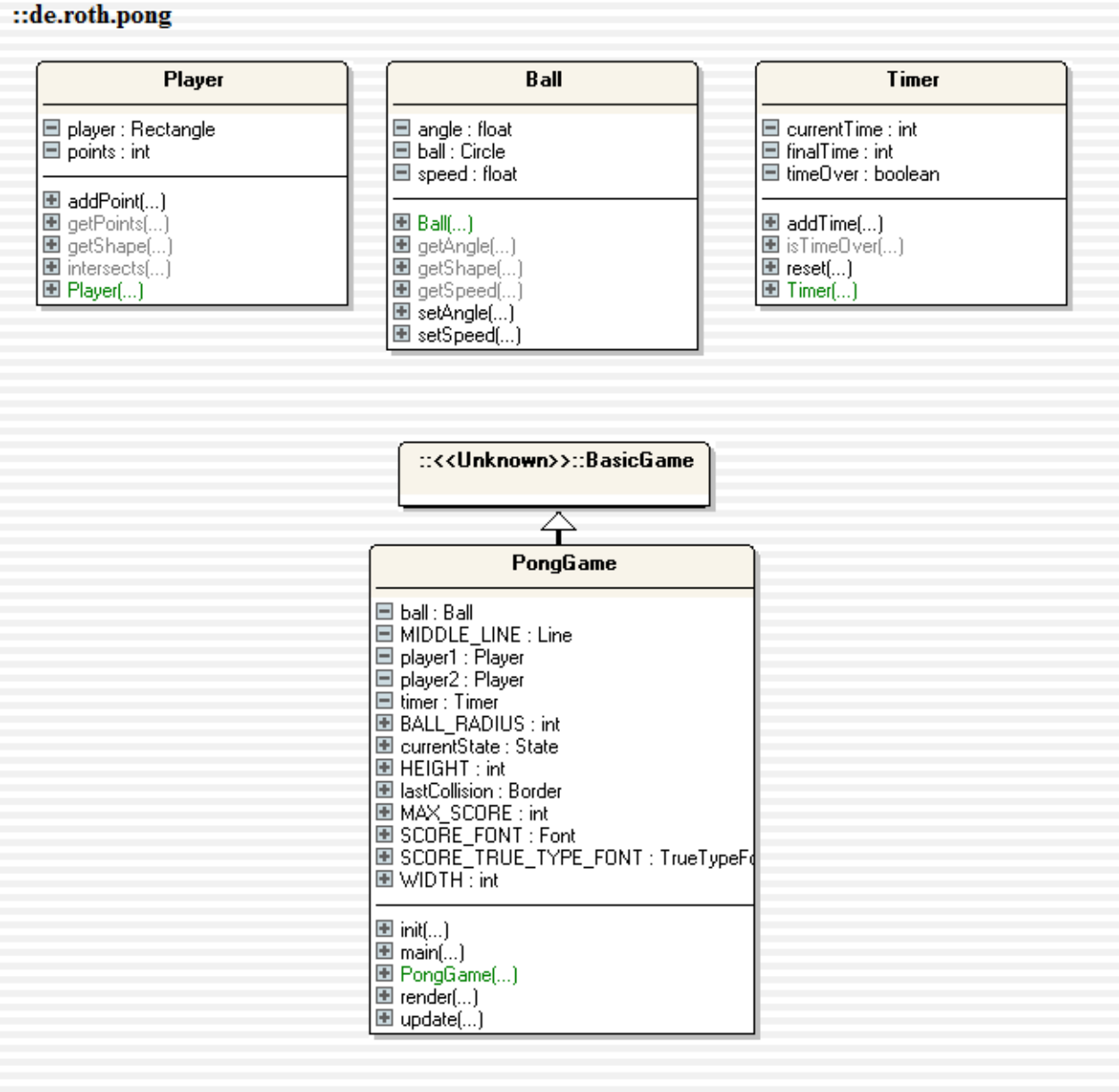
Ball

Der Ball wird ebenfalls durch ein Objekt repräsentiert. Dargestellt wird er mittels einem „Circle“. Hinzu kommen aktuelle Geschwindigkeit und Flugwinkel.

Timer

Ein Zeitgeber wird deswegen benötigt weil der Spielball nach einem gewissen Zeitabstand (eine Sekunde) an Geschwindigkeit zulegen soll. Dadurch dass unsere **Update-Methode** den Abstand, in Millisekunden, des letzten Frames als „**delta**“ übergeben bekommt, kann ein Timer einfach realisiert werden. Bei 60 FPS zum Beispiel beträgt dieser Abstand (das Delta) ungefähr 17ms (1s / 60). Dieser Timer kann dann im schlimmsten Fall eine Ungenauigkeit von 17ms annehmen, für unsere Bedürfnisse ist das aber vollkommen in Ordnung.

Klassendiagramm



10 Klassendiagramm

Quellcode

Player.java

```
package de.roth.pong;

import org.newdawn.slick.geom.Rectangle;
import org.newdawn.slick.geom.Shape;

/**
 * Repräsentiert den Spieler
 *
 * @author 933
 *
 */
public class Player {

	private Rectangle player;
	private int points;

	/**
	 * Konstruktor generiert ein Spieler als Rechteck an einer Position x,y
	 * (links oben vom Rechteck) mit einer Breite und einer Höhe.
	 * Die Punktezahl wird auf 0 gesetzt.
	 *
	 * @param x
	 * @param y
	 * @param width
	 * @param height
	 */
	public Player(int x, int y, int width, int height) {
		player = new Rectangle(x, y, width, height);
		this.points = 0;
	}

	public boolean intersects(Shape shape) {
		return this.player.intersects(shape);
	}

	/**
	 * Spieler als geometrische Form
	 *
	 * @return Shape - Spielerform
	 */
	public Rectangle getShape() {
		return player;
	}

	public int getPoints() {
		return points;
	}

	public void addPoint() {
		this.points = points + 1;
	}
}
```

Ball.java

```java
package de.roth.pong;

import java.util.Random;
import org.newdawn.slick.geom.Circle;
import org.newdawn.slick.geom.Shape;

/**
 * Repräsentiert den Spielball
 *
 * @author 933
 *
 */
public class Ball {

	private Circle ball;
	private float speed;
	private float angle;

	/**
	 * Konstruktor generiert einen zufällig fliegenden Ball, entweder zu
	 * Spieler1 oder zu Spieler 2, dabei beträgt die anfängliche
	 * Zusatzgeschwidigkeit 0.
	 *
	 * @param x - Startpunkt X-Koordinate
	 * @param y - Startpunkt Y-Koordinate
	 */
	public Ball(int x, int y) {
		Random r = new Random();
		this.ball = new Circle(x, y, PongGame.BALL_RADIUS);
		this.speed = 0;

		// Ball fliegt zu Spieler1 (40°-120°)
		if (r.nextBoolean()) {
			this.angle = r.nextInt(80) + 40;
		}
		// Ball fliegt zu Spieler2 (230°-310°)
		else {
			this.angle = r.nextInt(80) + 230;
		}
	}

	/**
	 * Ball als geometrische Form
	 *
	 * @return Shape - Ballform
	 */
	public Shape getShape() {
		return ball;
	}

	public float getSpeed() {
		return speed;
	}

	public void setSpeed(float speed) {
		this.speed = speed;
	}

	public float getAngle() {
		return angle;
	}

	public void setAngle(float angle) {
		this.angle = angle;
	}
}
```

Timer.java

```
package de.roth.pong;

public class Timer {

	private int finalTime;
	private int currentTime;
	private boolean timeOver;

	public Timer(int timeInMillis){
		this.finalTime = timeInMillis;
		this.timeOver = false;
		this.currentTime = 0;
	}

	public void addTime(int timeInMillis){
		this.currentTime = this.currentTime + timeInMillis;
		if(currentTime >= finalTime){
			timeOver = true;
		}
	}

	public boolean isTimeOver() {
		return timeOver;
	}

	public void reset(){
		this.timeOver = false;
		this.currentTime = 0;
	}
}
```

Da im Spiel unterschieden werden muss wo der Ball zuletzt angestoßen ist, um Mehrfachkollisionen zu vermeiden, führen wie noch eine Enum Border.java ein. Mehr dazu später.

Border.java

```
package de.roth.pong;

public enum Border {
	TOP, RIGHT, BOTTOM, LEFT, NONE

}
```

Auch mehrere Spielzustände müssen unterschieden werden. Befindet sich das Spiel im Startzustand oder ist ein Ball herrausgeflogen oder hat einer der beiden Spieler gewonnen. Dafür definieren wir ebenenfalls eine Enum, State.java.

State.java

```
package de.roth.pong;

public enum State {
	Start, Play, BallIsOut, Player1Wins, Player2Wins
}
```

Hier der Quellcode zum Herzstück der Anwendung, die Spiellogik.

PongGame.java

```
package de.roth.pong;
import java.awt.Font;

import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.BasicGame;
import org.newdawn.slick.GameContainer;
import org.newdawn.slick.Graphics;
import org.newdawn.slick.Input;
import org.newdawn.slick.SlickException;
import org.newdawn.slick.TrueTypeFont;
import org.newdawn.slick.geom.Line;

public class PongGame extends BasicGame {

	// Höhe/Breite des Spielfelds in Pixel
	public static final int WIDTH = 400;
	public static final int HEIGHT = 300;

	// Radius vom Ball
	public static final int BALL_RADIUS = 3;

	// Punkteanzahl zum gewinnen
	public static final int MAX_SCORE = 11;

	// Schrift für den Punktestand
	public static Font SCORE_FONT;
	public static TrueTypeFont SCORE_TRUE_TYPE_FONT;

	// Mittellinie
	private static Line MIDDLE_LINE = new Line(WIDTH / 2, 0, WIDTH / 2, HEIGHT);


	// Spielkomponenten
	private Player player1;
	private Player player2;
	private Ball ball;

	// Zeitmesser da nach einer gewissen Zeit der Ball schneller fliegen soll
	private Timer timer;

	// Wo war die Letzte Kollision, verhinderung von endlos Kollisionen
	Border lastCollision;

	// Aktueller Spielzustand
	State currentState = State.Start;

	public PongGame() {
		super("Pong");
	}

	/**
	 * Initialiserungsmethode wird lediglich einmal beim Start ausgeführt
	 */
	public void init(GameContainer gc) throws SlickException {
		// Spieler 1 wird erzeugt, Rechteckposition ( x = 5, y = 130, breite = 5, höhe =
40 );
		player1 = new Player(5, 130, 5, 40);

		// Spieler 2 wird erzeugt, Rechteckposition ( x = 390, y = 130, breite = 5, höhe
= 40 );
		player2 = new Player(390, 130, 5, 40);

		// Ball wird erzeugt in der Mitte vom Fenster
		ball = new Ball(WIDTH / 2 - BALL_RADIUS / 2, HEIGHT / 2 - BALL_RADIUS/ 2);

		// Timer um den Ball nach einer gewissen Zeit schneller zu machen, läuft
		// nach einer Sekunde ab.
```

```
            timer = new Timer(1000);

            // Es gab keine letzte Kollision
            lastCollision = Border.NONE;

            SCORE_FONT = new Font("Verdana", Font.PLAIN, 18);
            SCORE_TRUE_TYPE_FONT = new TrueTypeFont(SCORE_FONT, false);

      }

      /**
       * Wird immer wieder in einer Schleife ausgeführt, bei 60 FPS ungefähr alle
       * 19ms. int delta = Zeitabstand zur letzten Ausführung
       */
      public void update(GameContainer gc, int delta) throws SlickException {
            Input input = gc.getInput();

            // START-ZUSTAND
            if(currentState == State.Start){
                  if (input.isKeyDown(Input.KEY_ENTER)) {
                        currentState = State.Play;
                  }
            }

            // SPIEL-ZUSTAND
            if(currentState == State.Play || currentState == State.BallIsOut){
                  // Steuerung Spieler 1
                  if (input.isKeyDown(Input.KEY_W)) {
                        if (player1.getShape().getMinY() > 0) {
                              double hip = 0.4f * delta;
                              player1.getShape().setY((float) (player1.getShape().getY() -
(hip)));
                        }
                  }
                  if (input.isKeyDown(Input.KEY_S)) {
                        if (player1.getShape().getMaxY() < HEIGHT) {
                              double hip = 0.4f * delta;
                              player1.getShape().setY((float) (player1.getShape().getY() +
(hip)));
                        }
                  }

                  // Steuerung Spieler 2
                  if (input.isKeyDown(Input.KEY_UP)) {
                        if (player2.getShape().getMinY() > 0) {
                              double hip = 0.4f * delta;
                              player2.getShape().setY((float) (player2.getShape().getY() -
(hip)));
                        }
                  }
                  if (input.isKeyDown(Input.KEY_DOWN)) {
                        if (player2.getShape().getMaxY() < HEIGHT) {
                              double hip = 0.4f * delta;
                              player2.getShape().setY((float) (player2.getShape().getY() +
(hip)));
                        }
                  }

                  if(currentState == State.Start){
                        if(input.isKeyPressed(Input.KEY_ENTER)){
                              currentState = State.Play;
                        }
                  }

                  if(currentState == State.Play){
                        // Ball Flugbahn berechnen
                        float hip = 0.3f * delta + ball.getSpeed();
                        ball.getShape().setX((float) (ball.getShape().getX() + hip
                                    * Math.sin(Math.toRadians(ball.getAngle()))));
                        ball.getShape().setY((float) (ball.getShape().getY() - hip
                                    * Math.cos(Math.toRadians(ball.getAngle()))));
```

```
                        // Zeitabstand für den Timer hinzufügen, delta = Abstand in
Millisekunden zu der letzten Frame
                        timer.addTime(delta);

                        // Wenn die Zeit abgelaufen wird die Geschwindigkeit des Balles
erhöht
                        // und der Timer startet von neuem
                        if (timer.isTimeOver()) {
                              ball.setSpeed(ball.getSpeed() + 0.05f);
                              timer.reset();
                        }

                        // Ball stoßt oben an
                        if (ball.getShape().getMinY() <= 0 && lastCollision != Border.TOP) {
                              // Einfallswinkel = Ausfallswinkel
                              ball.setAngle((float) (-1 * (ball.getAngle() + Math.PI +
180)));
                              lastCollision = Border.TOP;
                        }
                        // Ball stoßt unten an
                        if (ball.getShape().getMaxY() >= HEIGHT && lastCollision !=
Border.BOTTOM) {
                              // Einfallswinkel = Ausfallswinkel
                              ball.setAngle((float) (-1 * (ball.getAngle() + Math.PI +
180)));
                              lastCollision = Border.BOTTOM;
                        }

                        // Spieler rechts trifft den Ball
                        if (player1.intersects(ball.getShape()) && lastCollision !=
Border.LEFT) {
                              // Einfallswinkel = Ausfallwinkel
                              ball.setAngle((float) (-1 * (ball.getAngle() + Math.PI)));
                              lastCollision = Border.LEFT;
                        }

                        // Spieler links trifft den Ball
                        if (player2.intersects(ball.getShape()) && lastCollision !=
Border.LEFT) {
                              // Einfallswinkel = Ausfallwinkel
                              ball.setAngle((float) (-1 * (ball.getAngle() + Math.PI)));
                              lastCollision = Border.LEFT;
                        }


                        // Ball fliegt links aus dem Bildschirm -> Punkt für Spieler Rechts
                        if (ball.getShape().getMaxX() < 0) {
                              player2.addPoint();
                              currentState = State.BallIsOut;
                              lastCollision = Border.NONE;

                              if(player2.getPoints() >= MAX_SCORE){
                                    currentState = State.Player2Wins;
                              }
                        }

                        // Ball fliegt rechts aus dem Bildschirm -> Punkt für Spieler Links
                        if (ball.getShape().getMinX() > WIDTH) {
                              player1.addPoint();
                              currentState = State.BallIsOut;
                              lastCollision = Border.NONE;

                              if(player1.getPoints() >= MAX_SCORE){
                                    currentState = State.Player1Wins;
                              }
                        }
                  }

                  if(currentState == State.BallIsOut){
```

```
                        // Falls der Ball aus dem Spielfeld ist und man [Enter] drückt gehts
weiter
                        if(input.isKeyDown(Input.KEY_ENTER)){
                              ball = new Ball(WIDTH / 2 - BALL_RADIUS / 2, HEIGHT / 2    -
BALL_RADIUS / 2);
                              currentState = State.Play;
                        }
                  }
            }

            // SPIEL VORBEI - ZUSTAND
            if(currentState == State.Player1Wins || currentState == State.Player2Wins ){
                  if(input.isKeyPressed(Input.KEY_ENTER)){
                        currentState = State.Play;
                        init(gc);
                  }
            }
      }

      /**
       * Grafisches rendern der Spielobjekte, wird so oft es geht ausgeführt.
       */
      public void render(GameContainer gc, Graphics g) throws SlickException {

            if(currentState == State.Start){
                  String pressEnterToStart = "[PRESS ENTER TO START]";
                  SCORE_TRUE_TYPE_FONT.drawString(WIDTH / 2 -
SCORE_TRUE_TYPE_FONT.getWidth(pressEnterToStart) / 2, HEIGHT/2 -
SCORE_TRUE_TYPE_FONT.getHeight(pressEnterToStart)/2 ,pressEnterToStart);

            }

            // Spieler Rechtecke füllen
            g.fill(player1.getShape());
            g.fill(player2.getShape());

            // Spieler Rechteck zeichnen
            g.draw(player1.getShape());
            g.draw(player2.getShape());

            // Mittellinie Zeichnen
            g.draw(MIDDLE_LINE);

            // Falls Ball im Spiel, Ball füllen und zeichnen
            if(ball != null){
                  g.fill(ball.getShape());
                  g.draw(ball.getShape());
            }

            // Punktestand mittig zeichnen
            String scoreText = player1.getPoints() + "     " + player2.getPoints();
            SCORE_TRUE_TYPE_FONT.drawString(WIDTH / 2 -
SCORE_TRUE_TYPE_FONT.getWidth(scoreText) / 2, 5,scoreText);

            // Meldung für neuen Ball mittig zeichnen
            if(currentState == State.BallIsOut){
                  String pressEnterForNewBall = "[PRESS ENTER FOR NEW BALL]";
                  SCORE_TRUE_TYPE_FONT.drawString(WIDTH / 2 -
SCORE_TRUE_TYPE_FONT.getWidth(pressEnterForNewBall) / 2, HEIGHT/2 -
SCORE_TRUE_TYPE_FONT.getHeight(pressEnterForNewBall)/2 ,pressEnterForNewBall);
            }

            // Meldung für Spieler1 hat gewonnen
            if(currentState == State.Player1Wins){
                  String player1Wins = "Player1 won! [NEW GAME - ENTER]";
                  SCORE_TRUE_TYPE_FONT.drawString(WIDTH / 2 -
SCORE_TRUE_TYPE_FONT.getWidth(player1Wins) / 2, HEIGHT/2 -
SCORE_TRUE_TYPE_FONT.getHeight(player1Wins)/2 ,player1Wins);
            }

            // Meldung für Spieler2 hat gewonnen
```

```
            if(currentState == State.Player2Wins){
                  String player2Wins = "Player2 won! [NEW GAME - ENTER]";
                  SCORE_TRUE_TYPE_FONT.drawString(WIDTH / 2 -
SCORE_TRUE_TYPE_FONT.getWidth(player2Wins) / 2, HEIGHT/2 -
SCORE_TRUE_TYPE_FONT.getHeight(player2Wins)/2 ,player2Wins);
            }
      }

      /**
       * Main Methode, Spiel wird gestartet
       * @param args
       * @throws SlickException
       */
      public static void main(String[] args) throws SlickException {
            AppGameContainer pong = new AppGameContainer(new PongGame());
            pong.setDisplayMode(WIDTH, HEIGHT, false);
            pong.setShowFPS(false);
            pong.start();
      }
}
```

Download

Das ganze Projekt kannst du dir hier herunterladen: Slick-PongTutorial.zip