Eclipse und Roboter Simulation

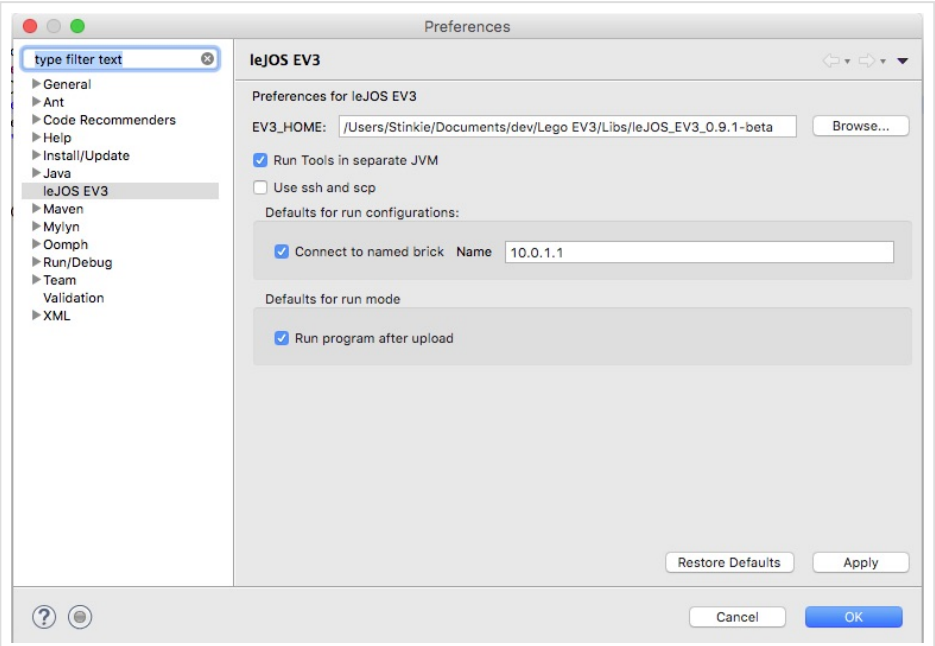
Die Bibliothek EV3JLIB enthält zusätzlich einen Simulations-Modus. Mit dem Simulation-Modus könnt ihr eure EV3-Programme auf eurem Rechner ohne den EV3 laufen lassen. Der EV3 wird dann per Software simuliert und die Bewegung wird in einem Java-Fenster graphisch dargestellt. Hier möchte ich euch zeigen, wie ihr euer EV3-Java-Programm im Simulations-Modus zum laufen bringt.

RobotSim – Dateien in den Eclipse-Workspace kopieren

Damit ihr den Simulation-Modus der EV3JLIB – Bibliothek nutzen könnt, müsst ihr euch eine zusätzliche Bibliothek herunterladen, die in dem Paket JGameGrid enthalten ist. JGameGrid könnt ihr euch von hier herunterladen: Download JGameGrid.

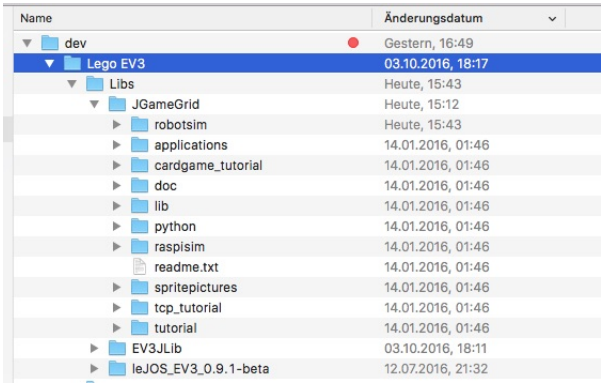
Nach dem Download solltet ihr die Dateien noch in euren Eclipse-Workspace für eure EV3-Projekte kopieren. Auch diese Dateien aus JGameGrid hab ich wieder in das Lib-Verzeichnis von leJOS kopiert.

Euer aktuelles leJOS-Verzeichnis findet ihr z.B. unter den Einstellungen für das EV3-Plugin in Eclipse unter: *Eclipse -> Einstellungen -> leJOS EV3 -> EV3_HOME*



EV3_HOME – Verzeichnis

Auf der gleichen Ebene des EV3_HOME-Verzeichnis hab ich dann wieder ein neues Verzeichnis mit dem Namen JGameGrid angelegt und den gesamten Inhalt aus der heruntergeladenen JGameGrid-ZIP-Datei in das neue Verzeichnis kopiert.



JGameGrid mit RoboSim

Im Verzeichnis *robotsim/lib* befindet sich das Package RobotSim.jar, welches ihr für eure Java-Programme benötigt um die EV3-Simulation zu verwenden. Dann gibt es im Verzeichnis *robotsim* u.a. noch diese Verzeichnisse:

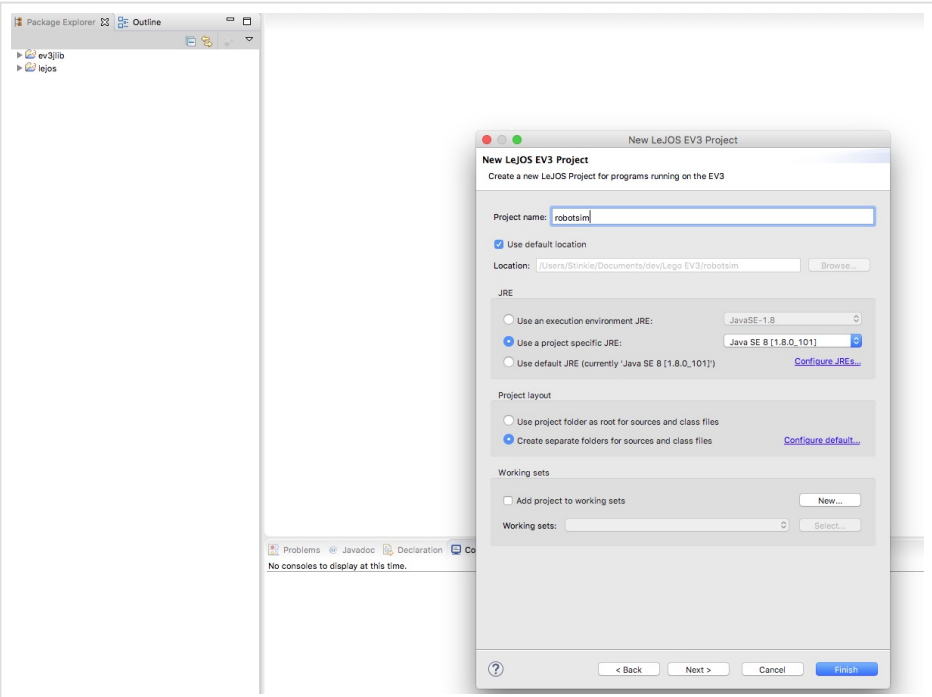
- *javadoc*: Die Dokumentation für RobotSim im HTML-Format
- *examples*: Java-Beispiele für RobotSim
- *src*: Der gesamte Quellcode der Bibliothek RobotSim

Neues Projekt anlegen

Ein neues Projekt müsst ihr in diesem Fall nicht unbedingt anlegen. Ihr könnt auch einfach, wie weiter unten beschrieben, die beiden benötigen Bibliotheken robotsim.jar und JGameGrid.jar in eure bestehende Projekte einfügen. Zum erklären und verstehen der Funktionsweise der Roboter-Simulation ist es aber übersichtlicher mit einem neuen Projekt zu starten.

Als nächstes legen wir in dem gleichen Eclipse-Workspace, wie das Projekt leJOS, ein neues EV3-Projekt mit *File -> New -> Project.. -> leJOS EV3 Project* an. Unter Projekt-Name geben wir den Namen des EV3-Projekts ein. Ich nenne das diesmal *robotsim*. Unter JRE wählen wir

wieder *Use a project specific JRE* aus. Und als JRE wählen nehmen wir das im vorherigen Schritt konfigurierte JDK 1.8.

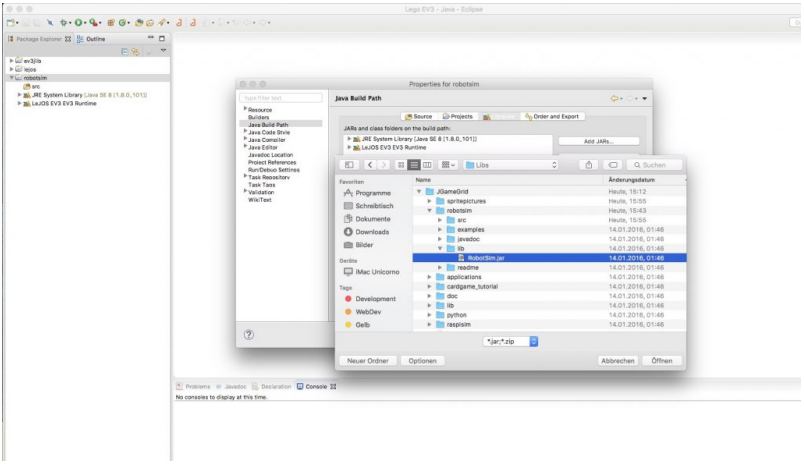


Neues Projekt robotsim

Nach dem Klick auf den *Finish* – Button wird das Projekt angelegt und enthält bereits wieder die für leJOS benötigten JAVA-Bibliotheken.

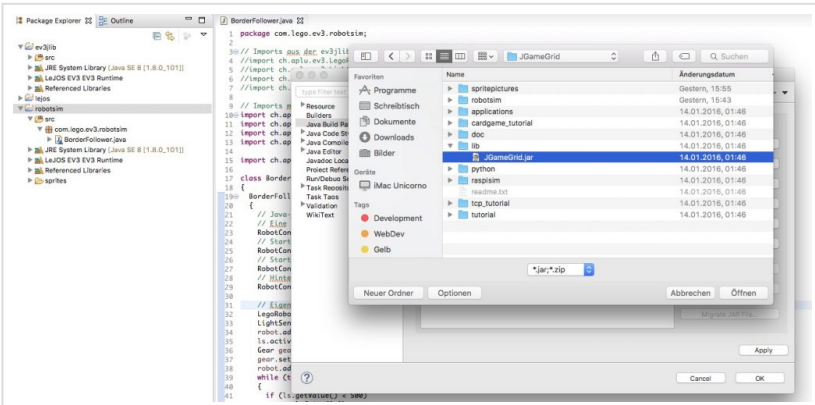
Zusätzlich benötigt ihr zum Programmieren mit *RobotSim* jetzt noch die Bibliotheken *robotsim.jar* und *JGamegrid.jar* (für die grafische Darstellung im Java-Fenster) im Build-Path eures Projekts. Die Bibliotheken könnt ihr in Eclipse mit einem Klick der rechten Maustaste auf euer neues robotsim-Projekt im Package-Explorer -> *Properties -> JAVA Build Path -> Libraries -> Add External jars.. hinzufügen.*

Zuerst navigiert ihr zu der Bibliothek *robotsim.jar* im Verzeichnis von JGameGrid unter *robotsim/lib* und wählt sie aus.



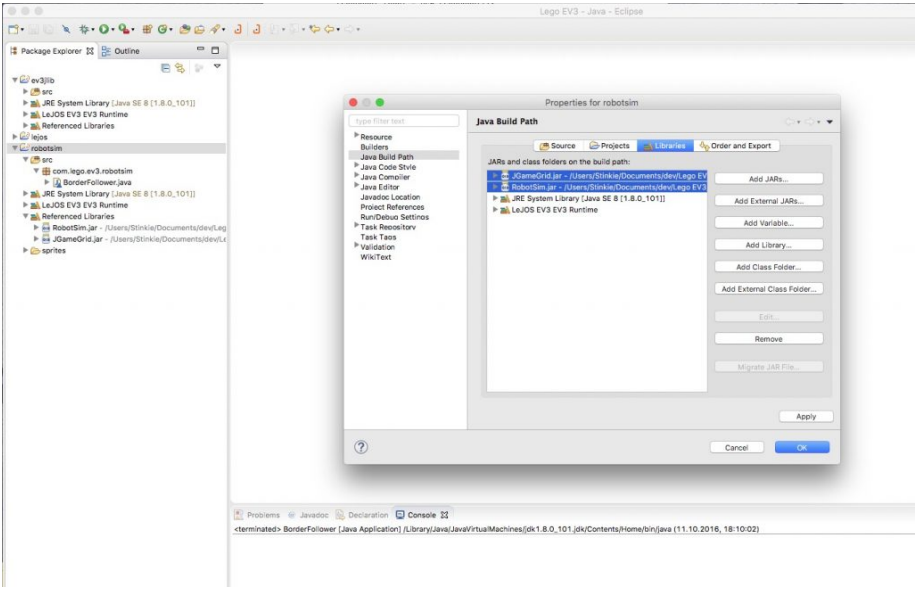
Bibliothek robotsim.jar

Danach wählt ihr auf die gleiche Weise noch die Bibliothek *JGameGrid.jar* im Pfad *lib* im Verzeichnis *JGameGrid* aus.



Bibliothek JGameGrid

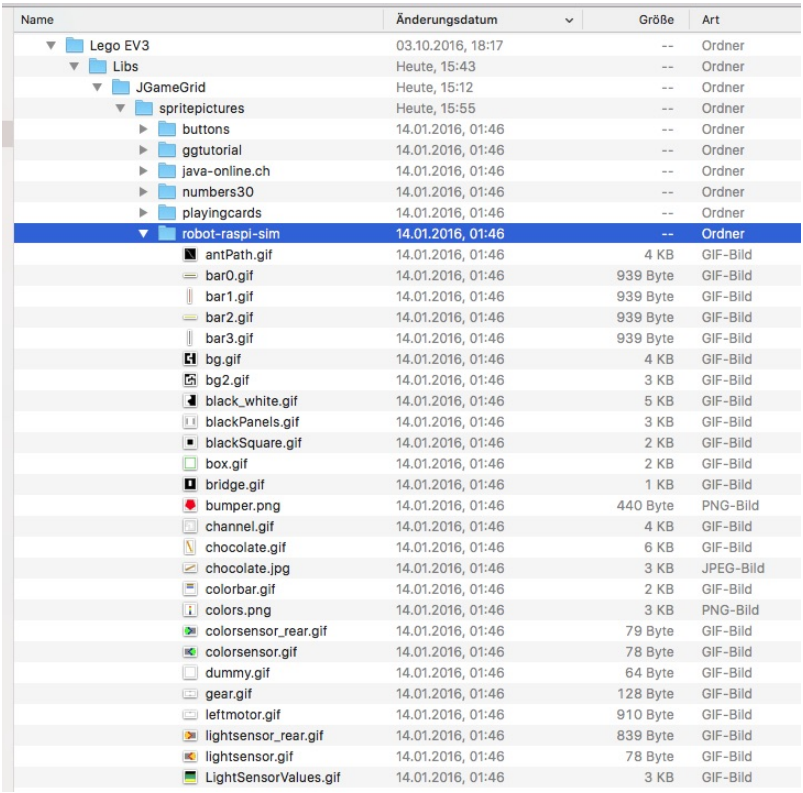
Danach sollten die Bibliotheken *robotsim.jar* und *JGameGrid.jar* in eurem Build-Path und im Package-Explorer von Eclipse auftauchen.



RobotSIm-Bibliotheken in Eclipse

Zur graphischen Darstellung des Roboters, von Hindernissen oder Hintergründen im Java-Fenster benötigt die Bibliothek *robotsim.jar* noch kleine Grafiken und Hintergründe, sogenannte Sprites. Die Hintergründe werden z.B. zur Darstellung von Bahnen, auf denen der Roboter entlang fahren soll, verwendet. Ihr könnt die vorgefertigten Hintergründe aus dem Sprites-Verzeichnis verwenden oder euch eigene erstellen.

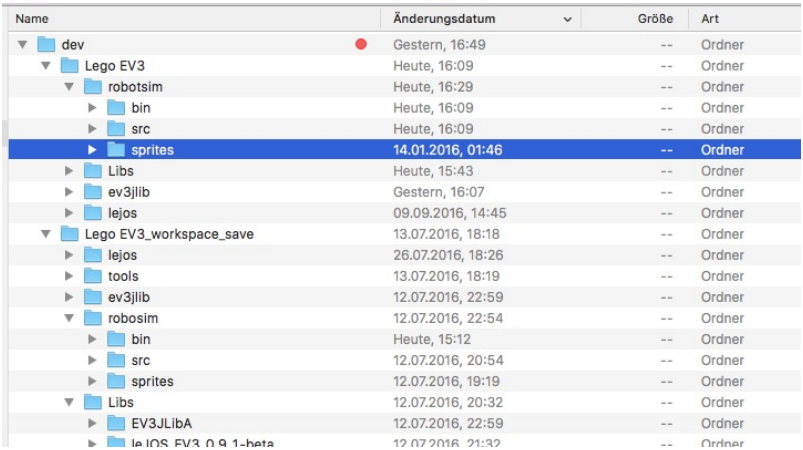
Die Sprites für *robotsim.jar* befinden sich im Verzeichnis von JGamesGrid unter *spritepictures/robot-raspi-sim*.



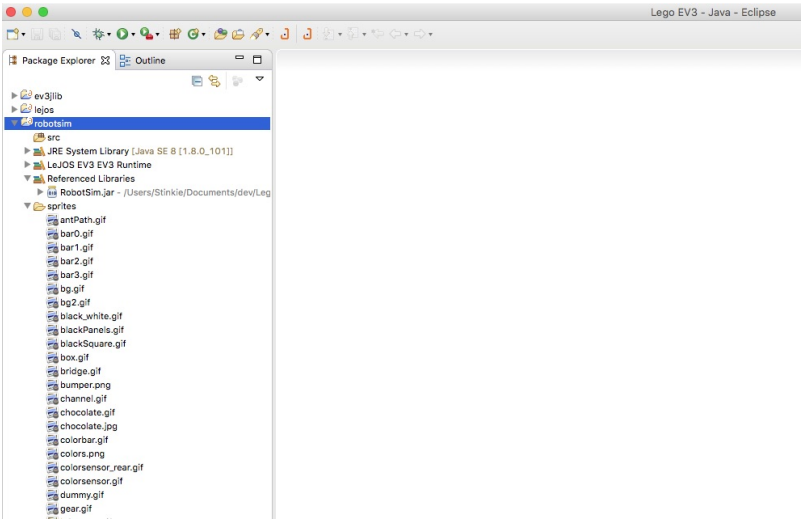
Name	Änderungsdatum	Größe	Art
Lego EV3	03.10.2016, 18:17	--	Ordner
Libs	Heute, 15:43	--	Ordner
JGameGrid	Heute, 15:12	--	Ordner
spritepictures	Heute, 15:55	--	Ordner
buttons	14.01.2016, 01:46	--	Ordner
ggtutorial	14.01.2016, 01:46	--	Ordner
java-online.ch	14.01.2016, 01:46	--	Ordner
numbers30	14.01.2016, 01:46	--	Ordner
playingcards	14.01.2016, 01:46	--	Ordner
robot-raspi-sim	14.01.2016, 01:46	--	Ordner
antPath.gif	14.01.2016, 01:46	4 KB	GIF-Bild
bar0.gif	14.01.2016, 01:46	939 Byte	GIF-Bild
bar1.gif	14.01.2016, 01:46	939 Byte	GIF-Bild
bar2.gif	14.01.2016, 01:46	939 Byte	GIF-Bild
bar3.gif	14.01.2016, 01:46	939 Byte	GIF-Bild
bg.gif	14.01.2016, 01:46	4 KB	GIF-Bild
bg2.gif	14.01.2016, 01:46	3 KB	GIF-Bild
black_white.gif	14.01.2016, 01:46	5 KB	GIF-Bild
blackPanels.gif	14.01.2016, 01:46	3 KB	GIF-Bild
blackSquare.gif	14.01.2016, 01:46	2 KB	GIF-Bild
box.gif	14.01.2016, 01:46	2 KB	GIF-Bild
bridge.gif	14.01.2016, 01:46	1 KB	GIF-Bild
bumper.png	14.01.2016, 01:46	440 Byte	PNG-Bild
channel.gif	14.01.2016, 01:46	4 KB	GIF-Bild
chocolate.gif	14.01.2016, 01:46	6 KB	GIF-Bild
chocolate.jpg	14.01.2016, 01:46	3 KB	JPEG-Bild
colorbar.gif	14.01.2016, 01:46	2 KB	GIF-Bild
colors.png	14.01.2016, 01:46	3 KB	PNG-Bild
colorsensor_rear.gif	14.01.2016, 01:46	79 Byte	GIF-Bild
colorsensor.gif	14.01.2016, 01:46	78 Byte	GIF-Bild
dummy.gif	14.01.2016, 01:46	64 Byte	GIF-Bild
gear.gif	14.01.2016, 01:46	128 Byte	GIF-Bild
leftmotor.gif	14.01.2016, 01:46	910 Byte	GIF-Bild
lightsensor_rear.gif	14.01.2016, 01:46	839 Byte	GIF-Bild
lightsensor.gif	14.01.2016, 01:46	78 Byte	GIF-Bild
LightSensorValues.gif	14.01.2016, 01:46	3 KB	GIF-Bild

Sprites für robotsim.jar

Die Sprites müsst ihr in euer Eclipse-Projekt-Verzeichnis *robotsim* in ein neues Verzeichnis mit dem Namen *sprites* kopieren, damit die Bibliothek *robotsim.jar* die Sprites beim Programmstart findet. Nach dem kopieren müsst ihr sehr wahrscheinlich in Eclipse erst mal einen Refresh (Rechte Mausklick auf das Projekt *robotsim -> Refresh* oder einfach F5 drücken) im Package-Explorer durchführen, damit euch Eclipse die Sprites auch anzeigt.



Sprites im Projektverzeichnis



Sprites in Eclipse-Projekt

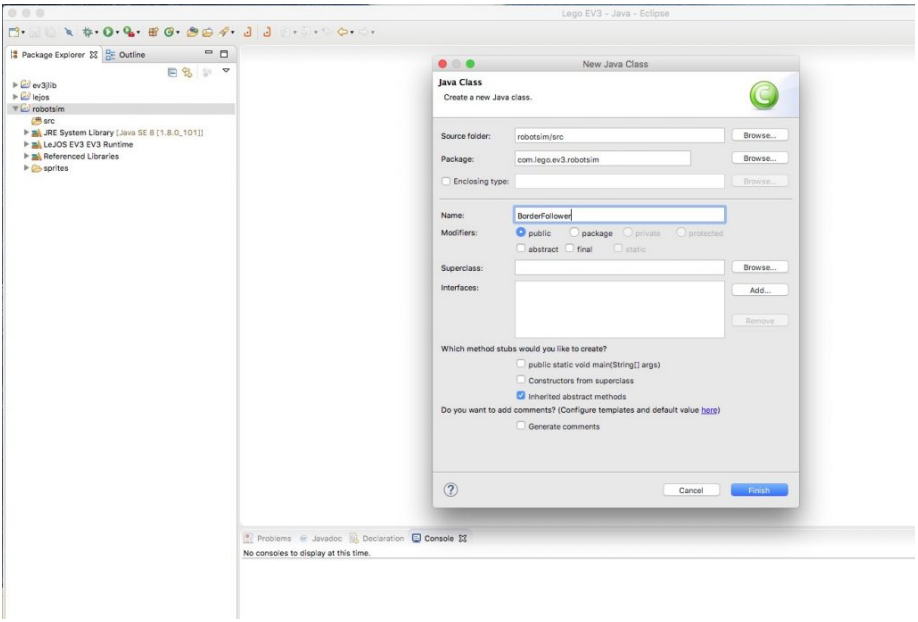
Wenn ihr in Eclipse ein JDK größer als 1.7 verwendet, müsst ihr die Compiler-Einstellung diesmal nicht unbedingt auf die Compiler-Version 1.7 umstellen. Die Simulation läuft auch mit neueren JDK-Versionen.

BorderFollower

Für das erste Beispiel könnt ihr das Programm BorderFollower aus dem letzten Kapitel mit der EV3JLIB verwenden. Die Klassen- und Methoden-Namen für die Steuerung des EV3 sind in beiden Bibliotheken (*ev3jlib.jar* und *robotsim.jar*) gleich. D.h. Ihr könnt zunächst eure Algorithmen mit der Simulation testen und wenn alles gut funktioniert, tauscht ihr einfach die Import-Direktiven auf die Klassen in der *robotsim.jar* mit den Import-Direktiven aus der Bibliothek *ev3jlib.jar* aus. Danach läuft das Programm auf eurem EV3 ohne das ihr den Algorithmus umschreiben müsst. Cool oder nicht?

Mit dem Programm BorderFollower machen wir hier das Gleiche, nur umgedreht.

Zunächst legen wir eine neue Klasse an. Das geht wie gehabt über *File -> New -> Class*. Als Package-Namen hab ich diesmal z.B. *com.lego.ev3.robotsim* vergeben, weil das Programm diesmal mit den Klassen aus der Bibliothek *robotsim* zusammen laufen soll. Als Namen wähle ich wieder *BorderFollower*.



Klasse BorderFollower anlegen

Mit Klick auf *Finish* wird die Java-Klasse angelegt.

Kopiert nun folgenden Beispiel-Code in eure neu angelegte Klasse:

```
package com.lego.ev3.robotsim;

// Imports aus der ev3jlib aus
//import ch.aplu.ev3.LegoRobot
//import ch.aplu.ev3.LightSens
//import ch.aplu.ev3.SensorPor
//import ch.aplu.ev3.Gear;

// Imports mit den gleichen Kl
import ch.aplu.robotsim.LegoRobot
import ch.aplu.robotsim.LightSens
import ch.aplu.robotsim.SensorPor
import ch.aplu.robotsim.Gear;

import ch.aplu.robotsim.RobotContext;

class BorderFollower {
```

```
BorderFollower() {
  // Java-Fenster für die Si

  // Eine Navigations-Leiste
  RobotContext.showNavigationBar();
  // Start-Position des Robc
  RobotContext.setStartPosition(220, 46
  // Start-Richtung des Robc
  RobotContext.setStartDirection(-90);
  // Hintergrund mit einer S
  RobotContext.useBackground("sprites/

  // Eigentliche Roboter-Ste
  LegoRobot robot = new LegoRo
  // Lichtsensor an Eingang
  LightSensor ls = new Light
  robot.addPart(ls);
  ls.activate(true);
```