```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package trafficsimulator.core;

import java.util.List;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import trafficsimulator.utils.Point;

/**
 *
 * @author snorri
 */
public class JunctionTest {

  private Lane lane1;
  private Lane lane2;

  @Before
  public void setUp() {

    Road r1 = new Road(new Point(20, 20), new Point(500, 20));
    lane1 = new Lane(Lane.Direction.IDENTICAL);
    lane2 = new Lane(Lane.Direction.IDENTICAL);
    r1.addLane(lane1);
    r1.addLane(lane2);
  }

  /**
   * Test whether a lane can connect to itself at a junction. This test is not
   * complete
   */
  @Test
  public void testLanesJunction1() {
    System.out.println("Opposite lanes at a junction");

    Junction junction = new Junction();
    junction.connect(lane1, lane1);

    fail(); // We shouldn't be able to get to this point

  }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package trafficsimulator.core;

import org.junit.Test;
import static org.junit.Assert.*;
import trafficsimulator.utils.Point;

/**
 *
 * @author snorri
 */
public class LaneTest {

  private Lane lane1;
  private Lane lane2;
  private Road road;

  public void setUp(Point start, Point end, Lane.Direction dir1, Lane.Direction
dir2) {

    road = new Road(start, end);
    lane1 = new Lane(dir1);
    lane2 = new Lane(dir2);
    road.addLane(lane1);
    road.addLane(lane2);

  }

  /**
   * This test checks if right and left parameters are calculated correctly if a
   * horizontal road is created and its direction is to the right like this: ->
   * with two lanes that are both IDENTICAL.
   */
  @Test
  public void testLanesHorizontalRight() {
    System.out.println("Lanes horizontal right");

    Point startLeft = new Point(100, 100);
    Point endLeft = new Point(400, 100);

    setUp(startLeft, endLeft, Lane.Direction.IDENTICAL, Lane.Direction.IDENTICAL
);
    /*
     System.out.println("lane1LeftStartPointX: " + lane1.getLeftStartPoint().get
X());
     System.out.println("lane1LeftStartPointY: " + lane1.getLeftStartPoint().get
Y());
     System.out.println("lane1LeftEndPointX: " + lane1.getLeftEndPoint().getX())
;
     System.out.println("lane1LeftEndPointY: " + lane1.getLeftEndPoint().getY())
;
     System.out.println("lane1RightStartPointX: " + lane1.getRightStartPoint().g
etX());
     System.out.println("lane1RightStartPointY: " + lane1.getRightStartPoint().g
etY());
```

```
     System.out.println("lane1RightEndPointX: " + lane1.getRightEndPoint().getX(
));
     System.out.println("lane1RightEndPointY: " + lane1.getRightEndPoint().getY(
));
     System.out.println("lane2LeftStartPointX: " + lane2.getLeftStartPoint().get
X());
     System.out.println("lane2LeftStartPointY: " + lane2.getLeftStartPoint().get
Y());
     System.out.println("lane2LeftEndPointX: " + lane2.getLeftEndPoint().getX())
;
     System.out.println("lane2LeftEndPointY: " + lane2.getLeftEndPoint().getY())
;
     System.out.println("lane2RightStartPointX: " + lane2.getRightStartPoint().g
etX());
     System.out.println("lane2RightStartPointY: " + lane2.getRightStartPoint().g
etY());
     System.out.println("lane2RightEndPointX: " + lane2.getRightEndPoint().getX(
));
     System.out.println("lane2RightEndPointY: " + lane2.getRightEndPoint().getY(
));
     */
    double expXLeftStartLane1 = startLeft.getX();
    double expYLeftStartLane1 = startLeft.getY();
    double expXRightStartLane1 = startLeft.getX();
    double expYRightStartLane1 = startLeft.getY() + lane1.laneWidth;
    double expXLeftEndLane1 = endLeft.getX();
    double expYLeftEndLane1 = endLeft.getY();
    double expXRightEndLane1 = endLeft.getX();
    double expYRightEndLane1 = endLeft.getY() + lane1.laneWidth;
    double expXLeftStartLane2 = startLeft.getX();
    double expYLeftStartLane2 = startLeft.getY() + lane1.laneWidth;
    double expXRightStartLane2 = startLeft.getX();
    double expYRightStartLane2 = startLeft.getY() + lane1.laneWidth + lane2.lane
Width;
    double expXLeftEndLane2 = endLeft.getX();
    double expYLeftEndLane2 = endLeft.getY() + lane1.laneWidth;
    double expXRightEndLane2 = endLeft.getX();
    double expYRightEndLane2 = endLeft.getY() + lane1.laneWidth + lane2.laneWidt
h;

    double resultXLeftStartLane1 = lane1.getLeftStartPoint().getX();
    double resultYLeftStartLane1 = lane1.getLeftStartPoint().getY();
    double resultXRightStartLane1 = lane1.getRightStartPoint().getX();
    double resultYRightStartLane1 = lane1.getRightStartPoint().getY();
    double resultXLeftEndLane1 = lane1.getLeftEndPoint().getX();
    double resultYLeftEndLane1 = lane1.getLeftEndPoint().getY();
    double resultXRightEndLane1 = lane1.getRightEndPoint().getX();
    double resultYRightEndLane1 = lane1.getRightEndPoint().getY();
    double resultXLeftStartLane2 = lane2.getLeftStartPoint().getX();
    double resultYLeftStartLane2 = lane2.getLeftStartPoint().getY();
    double resultXRightStartLane2 = lane2.getRightStartPoint().getX();
    double resultYRightStartLane2 = lane2.getRightStartPoint().getY();
    double resultXLeftEndLane2 = lane2.getLeftEndPoint().getX();
    double resultYLeftEndLane2 = lane2.getLeftEndPoint().getY();
    double resultXRightEndLane2 = lane2.getRightEndPoint().getX();
    double resultYRightEndLane2 = lane2.getRightEndPoint().getY();

    assertEquals(expXLeftStartLane1, resultXLeftStartLane1, 2.1);
    assertEquals(expYLeftStartLane1, resultYLeftStartLane1, 2.1);
```

```
    assertEquals(expXRightStartLane1, resultXRightStartLane1, 2.1);
    assertEquals(expYRightStartLane1, resultYRightStartLane1, 2.1);
    assertEquals(expXLeftEndLane1, resultXLeftEndLane1, 2.1);
    assertEquals(expYLeftEndLane1, resultYLeftEndLane1, 2.1);
    assertEquals(expXRightEndLane1, resultXRightEndLane1, 2.1);
    assertEquals(expYRightEndLane1, resultYRightEndLane1, 2.1);

    assertEquals(expXLeftStartLane2, resultXLeftStartLane2, 2.1);
    assertEquals(expYLeftStartLane2, resultYLeftStartLane2, 2.1);
    assertEquals(expXRightStartLane2, resultXRightStartLane2, 2.1);
    assertEquals(expYRightStartLane2, resultYRightStartLane2, 2.1);
    assertEquals(expXLeftEndLane2, resultXLeftEndLane2, 2.1);
    assertEquals(expYLeftEndLane2, resultYLeftEndLane2, 2.1);
    assertEquals(expXRightEndLane2, resultXRightEndLane2, 2.1);
    assertEquals(expYRightEndLane2, resultYRightEndLane2, 2.1);

  }

  /**
   * This test checks if right and left parameters are calculated correctly if a
   * horizontal road is created and its direction is to the right like this: ->
   * with two lanes where one is IDENTICAL and the other OPPOSITE.
   */
  @Test
  public void testLanesHorizontalRight2() {
    System.out.println("Lanes horizontal right");

    Point startLeft = new Point(100, 100);
    Point endLeft = new Point(400, 100);

    setUp(startLeft, endLeft, Lane.Direction.IDENTICAL, Lane.Direction.OPPOSITE)
;
    /*
     System.out.println("lane1LeftStartPointX: " + lane1.getLeftStartPoint().get
X());
     System.out.println("lane1LeftStartPointY: " + lane1.getLeftStartPoint().get
Y());
     System.out.println("lane1LeftEndPointX: " + lane1.getLeftEndPoint().getX())
;
     System.out.println("lane1LeftEndPointY: " + lane1.getLeftEndPoint().getY())
;
     System.out.println("lane1RightStartPointX: " + lane1.getRightStartPoint().g
etX());
     System.out.println("lane1RightStartPointY: " + lane1.getRightStartPoint().g
etY());
     System.out.println("lane1RightEndPointX: " + lane1.getRightEndPoint().getX(
));
     System.out.println("lane1RightEndPointY: " + lane1.getRightEndPoint().getY(
));
     System.out.println("lane2LeftStartPointX: " + lane2.getLeftStartPoint().get
X());
     System.out.println("lane2LeftStartPointY: " + lane2.getLeftStartPoint().get
Y());
     System.out.println("lane2LeftEndPointX: " + lane2.getLeftEndPoint().getX())
;
     System.out.println("lane2LeftEndPointY: " + lane2.getLeftEndPoint().getY())
;
     System.out.println("lane2RightStartPointX: " + lane2.getRightStartPoint().g
etX());
```

```
     System.out.println("lane2RightStartPointY: " + lane2.getRightStartPoint().g
etY());
     System.out.println("lane2RightEndPointX: " + lane2.getRightEndPoint().getX(
));
     System.out.println("lane2RightEndPointY: " + lane2.getRightEndPoint().getY(
));
     */
    double expXLeftStartLane1 = startLeft.getX();
    double expYLeftStartLane1 = startLeft.getY();
    double expXRightStartLane1 = startLeft.getX();
    double expYRightStartLane1 = startLeft.getY() + lane1.laneWidth;
    double expXLeftEndLane1 = endLeft.getX();
    double expYLeftEndLane1 = endLeft.getY();
    double expXRightEndLane1 = endLeft.getX();
    double expYRightEndLane1 = endLeft.getY() + lane1.laneWidth;
    double expXLeftStartLane2 = road.getRightEndPoint().getX();
    double expYLeftStartLane2 = road.getRightEndPoint().getY();
    double expXRightStartLane2 = road.getRightEndPoint().getX();
    double expYRightStartLane2 = road.getRightEndPoint().getY() - 22;
    double expXLeftEndLane2 = road.getRightStartPoint().getX();
    double expYLeftEndLane2 = road.getRightStartPoint().getY();
    double expXRightEndLane2 = road.getRightStartPoint().getX();
    double expYRightEndLane2 = road.getRightStartPoint().getY() - 22;

    double resultXLeftStartLane1 = lane1.getLeftStartPoint().getX();
    double resultYLeftStartLane1 = lane1.getLeftStartPoint().getY();
    double resultXRightStartLane1 = lane1.getRightStartPoint().getX();
    double resultYRightStartLane1 = lane1.getRightStartPoint().getY();
    double resultXLeftEndLane1 = lane1.getLeftEndPoint().getX();
    double resultYLeftEndLane1 = lane1.getLeftEndPoint().getY();
    double resultXRightEndLane1 = lane1.getRightEndPoint().getX();
    double resultYRightEndLane1 = lane1.getRightEndPoint().getY();
    double resultXLeftStartLane2 = lane2.getLeftStartPoint().getX();
    double resultYLeftStartLane2 = lane2.getLeftStartPoint().getY();
    double resultXRightStartLane2 = lane2.getRightStartPoint().getX();
    double resultYRightStartLane2 = lane2.getRightStartPoint().getY();
    double resultXLeftEndLane2 = lane2.getLeftEndPoint().getX();
    double resultYLeftEndLane2 = lane2.getLeftEndPoint().getY();
    double resultXRightEndLane2 = lane2.getRightEndPoint().getX();
    double resultYRightEndLane2 = lane2.getRightEndPoint().getY();

    assertEquals(expXLeftStartLane1, resultXLeftStartLane1, 2.1);
    assertEquals(expYLeftStartLane1, resultYLeftStartLane1, 2.1);
    assertEquals(expXRightStartLane1, resultXRightStartLane1, 2.1);
    assertEquals(expYRightStartLane1, resultYRightStartLane1, 2.1);
    assertEquals(expXLeftEndLane1, resultXLeftEndLane1, 2.1);
    assertEquals(expYLeftEndLane1, resultYLeftEndLane1, 2.1);
    assertEquals(expXRightEndLane1, resultXRightEndLane1, 2.1);
    assertEquals(expYRightEndLane1, resultYRightEndLane1, 2.1);

    assertEquals(expXLeftStartLane2, resultXLeftStartLane2, 2.1);
    assertEquals(expYLeftStartLane2, resultYLeftStartLane2, 2.1);
    assertEquals(expXRightStartLane2, resultXRightStartLane2, 2.1);
    assertEquals(expYRightStartLane2, resultYRightStartLane2, 2.1);
    assertEquals(expXLeftEndLane2, resultXLeftEndLane2, 2.1);
    assertEquals(expYLeftEndLane2, resultYLeftEndLane2, 2.1);
    assertEquals(expXRightEndLane2, resultXRightEndLane2, 2.1);
    assertEquals(expYRightEndLane2, resultYRightEndLane2, 2.1);
```

```
  }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package trafficsimulator.core;

import org.junit.Test;
import static org.junit.Assert.*;
import trafficsimulator.utils.Point;

/**
 *
 * @author snorri
 */
public class RoadTest {

  private Road road;

  public void setUp(Point start, Point end) {
    road = new Road(start, end);
    Lane lane1 = new Lane(Lane.Direction.IDENTICAL);
    Lane lane2 = new Lane(Lane.Direction.OPPOSITE);
    road.addLane(lane1);
    road.addLane(lane2);
  }

  /**
   * This test checks if right parameters are calculated correctly if a
   * horizontal road is created and its direction is to the right, like this: ->
   * .
   */
  @Test
  public void testRoadHorizontalRight() {
    System.out.println("Road horizontal right");

    setUp(new Point(100, 100), new Point(400, 100));

    double expYStart = road.getLeftStartPoint().getY() + road.calculateWidth();
    double expXStart = road.getLeftStartPoint().getX();
    double expYEnd = road.getLeftEndPoint().getY() + road.calculateWidth();
    double expXEnd = road.getLeftEndPoint().getX();

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a
   * horizontal road is created and its direction is to the left, like this: <-
   * .
```

```
   */
  @Test
  public void testRoadHorizontalLeft() {
    System.out.println("Road horizontal left");

    setUp(new Point(400, 100), new Point(100, 100));

    double expYStart = road.getLeftStartPoint().getY() - road.calculateWidth();
    double expXStart = road.getLeftStartPoint().getX();
    double expYEnd = road.getLeftEndPoint().getY() - road.calculateWidth();
    double expXEnd = road.getLeftEndPoint().getX();

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a vertical
   * road is created and its direction is to the up, like this: ^ | .
   */
  @Test
  public void testRoadVerticalUp() {
    System.out.println("Road vertical up");

    setUp(new Point(100, 400), new Point(100, 100));

    double expYStart = road.getLeftStartPoint().getY();
    double expXStart = road.getLeftStartPoint().getX() + road.calculateWidth();
    double expYEnd = road.getLeftEndPoint().getY();
    double expXEnd = road.getLeftEndPoint().getX() + road.calculateWidth();

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a vertical
   * road is created and its direction is to the right, like this: | v .
   */
  @Test
  public void testRoadVerticalDown() {
    System.out.println("Road vertical down");
```

```
    setUp(new Point(100, 100), new Point(100, 400));

    double expYStart = road.getLeftStartPoint().getY();
    double expXStart = road.getLeftStartPoint().getX() - road.calculateWidth();
    double expYEnd = road.getLeftEndPoint().getY();
    double expXEnd = road.getLeftEndPoint().getX() - road.calculateWidth();

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a road
   * that has a downward slope is created and its direction is to the right,
   * like this: \ v .
   */
  @Test
  public void testRoadDownwardRight() {
    System.out.println("Road downward right");

    setUp(new Point(100, 100), new Point(400, 400));

    double expYStart = road.getLeftStartPoint().getY() + 31;
    double expXStart = road.getLeftStartPoint().getX() - 31;
    double expYEnd = road.getLeftEndPoint().getY() + 31;
    double expXEnd = road.getLeftEndPoint().getX() - 31;

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a road
   * that has a downward slope is created and its direction is to the left, like
   * this: / v .
   */
  @Test
  public void testRoadDownwardLeft() {
    System.out.println("Road downward left");

    setUp(new Point(400, 100), new Point(100, 400));

    double expYStart = road.getLeftStartPoint().getY() - 31;
```

```
    double expXStart = road.getLeftStartPoint().getX() − 31;
    double expYEnd = road.getLeftEndPoint().getY() − 31;
    double expXEnd = road.getLeftEndPoint().getX() − 31;

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a road
   * that has an upward slope is created and its direction is to the right, like
   * this: ^ / .
   */
  @Test
  public void testRoadUpwardRight() {
    System.out.println("Road upward right");

    setUp(new Point(100, 400), new Point(400, 100));

    double expYStart = road.getLeftStartPoint().getY() + 31;
    double expXStart = road.getLeftStartPoint().getX() + 31;
    double expYEnd = road.getLeftEndPoint().getY() + 31;
    double expXEnd = road.getLeftEndPoint().getX() + 31;

    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

  /**
   * This test checks if right parameters are calculated correctly if a road
   * that has an upward slope is created and its direction is to the left, like
   * this: ^ \.
   */
  @Test
  public void testRoadUpwardLeft() {
    System.out.println("Road upward left");

    setUp(new Point(400, 400), new Point(100, 100));

    double expYStart = road.getLeftStartPoint().getY() − 31;
    double expXStart = road.getLeftStartPoint().getX() + 31;
    double expYEnd = road.getLeftEndPoint().getY() − 31;
    double expXEnd = road.getLeftEndPoint().getX() + 31;
```

```
    double resultYStart = road.getRightStartPoint().getY();
    double resultXStart = road.getRightStartPoint().getX();
    double resultYEnd = road.getRightEndPoint().getY();
    double resultXEnd = road.getRightEndPoint().getX();

    assertEquals(expYStart, resultYStart, 0.001);
    assertEquals(expXStart, resultXStart, 0.001);
    assertEquals(expYEnd, resultYEnd, 0.001);
    assertEquals(expXEnd, resultXEnd, 0.001);

  }

}
```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package trafficsimulator.core;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

//JUnit Suite Test
@RunWith(Suite.class)
@Suite.SuiteClasses({
  RoadTest.class,
  VehicleTest.class,
  JunctionTest.class,
  LaneTest.class
})

/**
 *
 * @author snorri
 */
public class TrafficSimulatorTestSuite {
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package trafficsimulator.core;

import org.junit.Test;
import static org.junit.Assert.*;

import trafficsimulator.utils.Point;
import trafficsimulator.utils.Size;
import trafficsimulator.vehicles.*;

/**
 *
 * @author snorri This class will be changing a lot. After the changes more
 * thorough testing will be done.
 */
public class VehicleTest {

  /**
   * Test the height of a reckless car.
   */
  @Test
  public void testHeightRecklessCar() {
    System.out.println("Height of a reckless car");

    Lane lane = new Lane(Lane.Direction.IDENTICAL);
    Vehicle recklessCar = new Car();

    Size expResult = new Size(14, 8);
    Size result = recklessCar.getSize();

    assertEquals(expResult.height, result.height, 0.001);
  }

  /**
   * Test the height of a reckless bus.
   */
  @Test
  public void testHeightRecklessBus() {
    System.out.println("Height of a reckless bus");

    Lane lane = new Lane(Lane.Direction.IDENTICAL);
    Vehicle recklessBus = new Bus();

    Size expResult = new Size(20, 10);
    Size result = recklessBus.getSize();

    assertEquals(expResult.height, result.height, 0.001);
  }

  /**
   * Test if a reckless bus moves
   */
  @Test
  public void testRecklessBusMovement() {
    System.out.println("Movement of a reckless bus");
```

```
    Road road = new Road(new Point(20, 20), new Point(500, 20));
    Lane lane = new Lane(Lane.Direction.IDENTICAL);
    road.addLane(lane);
    Vehicle recklessBus = new Bus();

    double initialPos = recklessBus.getPosition().getX();
    recklessBus.step();
    double finalPos = recklessBus.getPosition().getX();

    assertTrue(finalPos > initialPos);
  }

  /**
   * Test vehicle outside of road boundaries
   */
  @Test
  public void testRecklessBusOutsideRoad1() {
    System.out.println("Creation of vehicle outside of Road parameter");

    Road road = new Road(new Point(20, 20), new Point(500, 20));
    Lane lane = new Lane(Lane.Direction.IDENTICAL);
    road.addLane(lane);
    Vehicle recklessBus = new Bus();

    double roadStartX = road.getLeftStartPoint().getX();
    double roadStartY = road.getLeftStartPoint().getY();
    double roadEndX = road.getLeftEndPoint().getX();
    double roadEndY = road.getLeftEndPoint().getY();
    double recklessBusX = recklessBus.getPosition().getX();
    double recklessBusY = recklessBus.getPosition().getY();

    assertTrue((recklessBusX >= roadStartX && recklessBusX <= roadEndX)
            || (recklessBusY >= roadStartY && recklessBusY <= roadEndY));
  }

  @Test
  public void testRecklessBusOutsideRoad2() {
    System.out.println("Movement of a vehicle outside of road");

    final Road road = new Road(new Point(20, 20), new Point(500, 20));
    final Lane lane = new Lane(Lane.Direction.IDENTICAL);
    road.addLane(lane);
    final Vehicle recklessBus = new Bus();

    Simulation s = new Simulation() {

      @Override
      protected void init() {
        map.addRoad(road);
        addVehicle(recklessBus, lane, 1);
      }
    };

    double roadStartX = road.getLeftStartPoint().getX();
    double roadStartY = road.getLeftStartPoint().getY();
    double roadEndX = road.getLeftEndPoint().getX();
    double roadEndY = road.getLeftEndPoint().getY();
    double recklessBusX = recklessBus.getPosition().getX();
```

```
    double recklessBusY = recklessBus.getPosition().getY();

    assertTrue((recklessBusX >= roadStartX && recklessBusX <= roadEndX)
            || (recklessBusY >= roadStartY && recklessBusY <= roadEndY));
  }
}
```