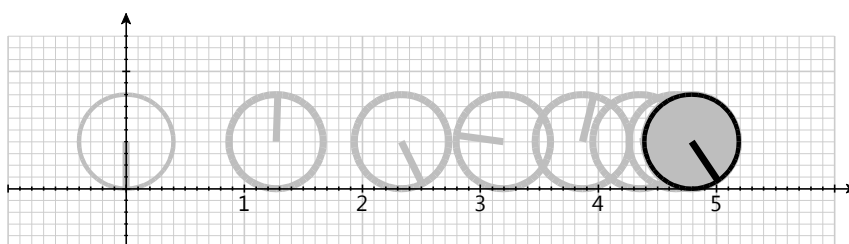


Exercise Sheet 5

Rolling Ball



This exercise sheet simulates a ball rolling on a horizontal surface. Let $r = 0,4\text{ m}$ be the radius of the ball and $m = 5\text{ kg}$ be its mass. The ball moves along the x-axis. Variable x describes the position of the ball along the x-axis and variable v represents its velocity. At the beginning of the simulation, i. e. at the time $t = 0$, x has the value 0 and v the value $2,3 \frac{\text{m}}{\text{s}}$.

The ball experiences rolling resistance during the rolling movement. The rolling resistance $F_r = -m \cdot g \cdot c_r$ is directed against the movement and slows down the ball. In this equation $g = 9,81 \frac{\text{m}}{\text{s}^2}$ is the acceleration due to earth gravity and c_r is the rolling resistance coefficient. The rolling resistance coefficient depends on the material of the ball and the base material. Let us assume that the resistance coefficient be $c_r = 0,05$. Let a be the acceleration, that applies to the ball. The following applies: $a = \frac{F_r}{m}$.

The rolling resistance acts as long as the ball is moving. As soon as the ball comes to a standstill, this force no longer applies. The moment the ball comes to a stop, the value of the variable F_r jumps abruptly from $-m \cdot g \cdot c_r$ to 0.

Exercise 1

Program a physical system named *RollingBall*. Insert all relevant physical variables into the *RollingBall* class. Provide the variables with the correct initial values, physical units and derivation relationships. It shall be assumed that variable F_r has a constant value $F_r = -m \cdot g \cdot c_r$ throughout the simulation.

The relevant physical variables shall be displayed in the plotter. Complete the program accordingly!

Start the simulation!

Exercise 2

A graphics component named *RollingBallTVG* is to be programmed for the physical system. The following program code shows the basic structure for the class *RollingBallTVG*. Copy this program code.

```
import de.physolator.usr.tvg.TVG;
import de.physolator.usr.tvg.Shape;

public class RollingBallTVG extends TVG {

    private RollingBall rb;


    public RollingBallTVG(RollingBall rollingBall) {
        rb = rollingBall;
        geometry.setUserArea(-1, 6, -0.5, 1.3);
        geometry.setRim(30, 30, 30, 30);
        scalesStyle.visible = true;
    }

    public void paint() {
        style.useUCS = true;
        style.strokeWidth = 3;
        // Platz für Zeichenbefehle
    }
}
```

Die Grafikkomponente *RollingBallTVG* soll angezeigt werden, sobald das physikalische System *RollingBall* geladen wird. Ergänzen Sie dazu den Programmcode von *RollingBall* um die folgenden Zeilen:

The graphics component *RollingBallTVG* is to be displayed as soon as the physical system *RollingBall* is loaded. To achieve this, add the following lines to the program code of *RollingBall*:

```
public void initGraphicsComponents(GraphicsComponents g) {
    g.addTVG(new RollingBallTVG(this));
}
```

With this addition and after reloading the physical system *RollingBall* (reload button ) a graphics component appears in the Physolator. The graphics component shows the two coordinate axes and the grid lines. The drawing area ranges from -1 to 6 in the x direction and from -0.5 to 1.3 in the y direction.

In this graphics component, the current state of the physical system is to be plotted in the following exercises. To do this, appropriate drawing commands shall be inserted at the marked position in the paint method. The physical system *RollingBall* can be accessed from within the graphics component *RollingBallTVG*. The variable that contains the physical system has the name *rb* and is of type *RollingBall*. If you want to access the current position of the ball in the paint method, you can find it under *rb.x*, the current speed can be found under *rb.v*, etc.

Explanation of the program code

In contrast to the graphics from the previous exercise sheets, this graphics shows the current state of a physical system. For the graphics component to have access to the physical system, the physical system is passed to the constructor of the physical system:

```
new RollingBallTVG(this)
```

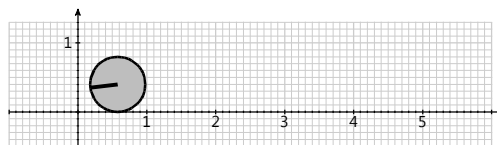
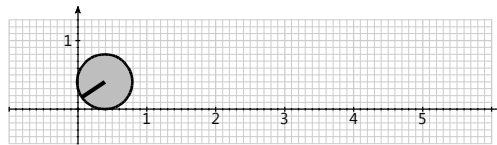
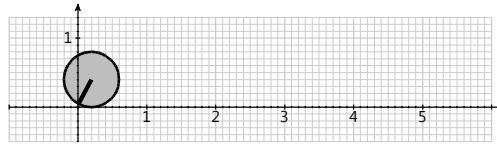
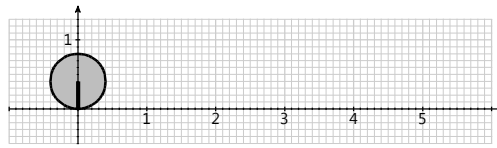
In the above program code “this” represents the actual instance of *RollingBall*, i. e. the current physical system. The constructor of *RollingBallTVG* saves this instance to *rb* using the following assignment:

```
rb = rollingBall;
```

As a result, the physical system can be accessed from within the paint method of the graphics component.

Exercise 3

The ball shall now be drawn into the graphics at the current position *rb.x*. The base on which the ball rolls is the x-axis. The center of the ball is therefore at the position (*rb.x*, *rb.r*).



The ball shall be represented by a circle. To improve visualization of the rolling movement, the ball shall be marked with a line. The line extends from the middle of the ball to the circle line and points downwards at the beginning. The rolling motion of the ball turns this line clockwise. Note: The direction of the marking depends only on the current position of the ball. First determine the rotation angle of the marking depending on the current position of the ball.

Draw the ball including its marking into the graphics. To do this, insert appropriate drawing commands into the *paint* method.

<code>drawLine(x1,y1,x2,y2);</code>	draws a line from $(x1,y1)$ to $(x2,y2)$
<code>drawCircle(x,y,r, Shape.POLYGON_LINE_LOOP);</code>	draws a filled circle with its center located at (x,y) and radius r

Draw the marking line with a pixel width of 5 to make it easier to recognize this line. To do this, insert the following assignment before the drawing command for marking:

```
style.strokeWidth = 5;
```

Exercise 4

The aim of this exercise is to find out how long the ball will roll and how far it will go.



So far, the ball has experienced an acceleration $F_r = -m \cdot g \cdot c_r$ during the entire simulation. The variable receives this value at the beginning of the simulation and it is not changed during the simulation. Now the value of F_r shall be set to 0 exactly at the time when v reaches speed 0. To do this, insert the following piece of program code into the *RollingBall* class.

```


public void g(double t, AfterEventDescription afterEventDescription) {
    if (v < 0)
        afterEventDescription.reportEvent() -> {
            Fr = 0;
        };
}

```

This addition in the program code causes the simulation to determine the time at which v falls below 0. The Physolator calculates this point in time by interval nesting and it calculates the state at this point in time. As soon as the physical system has reached this state, the assignment $Fr=0$; is executed. Such a point in time is called a physical event.

Move the mouse to the plotter. The button  for the settings appears. Press this button. A dialog appears with the TVG parameters. Go to plot and check "show time line". An additional time axis appears below the plot area. Now carry out the simulation in single steps by pressing the button  repeatedly. You can see that the simulation is proceeding in steps of $0.05s$. Each state is represented on the time axis by a small gray line. At the time of the physical event, an additional time and the state of the physical system is calculated at this time. On the time axis, the additional points in time of the physical events are represented by somewhat larger black lines.



Observe the state of the variables in the area  Structure and go through step by step to the time of the physical event. At the physical event, there are two states both exactly at the same point in time: a state immediately before the stopping event and a state immediately after the stopping. You can see that the value of Fr is changed to 0 between these two states.

Read the time t at which the event occurred and the x -value at this time.

Alternatively, the values of t and x can also be printed on the console during simulation. In the subsequent variant of g , not only Fr is set to 0 when the physical event occurs, but t and x are also printed on the standard output with the command `System.out.println`.

```

public void g(double t, AfterEventDescription afterEventDescription) {
    if (v < 0)
        afterEventDescription.reportEvent() -> {
            System.out.println("Stoß t=" + t + " x=" + x);
            Fr = 0;
        };
}

```

The Physolator determines the points in time of physical events by interval nesting. The accuracy of the time values is 12 decimal places. This is very accurate, but not exact. For this reason, the value of v is not exactly 0 at this point in time, but differs slightly from 0. To force the ball to stop moving after the physical event - not even slightly - the value of v shall be set to 0 at this point in time. Change your program code as follows and restart the simulation.

```

public void g(double t, AfterEventDescription afterEventDescription) {
    if (v < 0)
        afterEventDescription.reportEvent() -> {
            System.out.println("Stoß t=" + t + " x=" + x);
            Fr = 0;
            v = 0;
        };
}

```

Exercise 5

The flow resistance (air resistance) has not been taken into account so far. Now the question is to be answered again, how long the ball will roll and how far it will go - but this time both the rolling resistance and the flow resistance shall be taken into account.

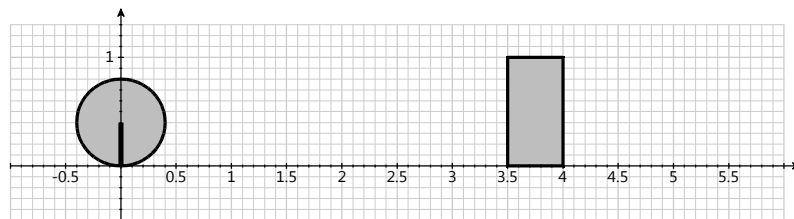
To do this, expand the existing physical system in such a way that when calculating the force, not only the rolling friction but also the flow resistance is taken into account. The following equation applies to the flow resistance F_L . Make sure that the flow resistance F_L is always directed in the opposite direction to the direction of movement v .

$$F_L = \frac{1}{2} A c_w \rho v^2$$

In this equation $A = r^2 \pi$ is the cross-sectional area of the ball, $c_w = 0,4$ is the flow resistance coefficient for a spherical body and $\rho = 1,2041 \frac{\text{kg}}{\text{m}^3}$ is the density of air.

Note that the formula $F_L = \frac{1}{2} A c_w \rho v^2$ always returns a positive value. To ensure that the force always opposes the direction of movement v , multiply this value by $-\text{signum}(v)$. The signum function maps positive numbers to 1, negative numbers to -1 and 0 to 0. In the programming language Java the signum function is part of the package `java.lang.Math`.

Exercise 6



In addition, there is now a wall at the location $x_w = 3,5 \text{ m}$. Complement the graphics component `RollingBallTVG` so that this wall is drawn according to the illustration above.

<code>drawRectangle(x1,y1,x2,y2,Shape.POLYGON_LINE_LOOP);</code>	draws an axis-parallel rectangle where (x1, y1) is the lower left corner, (x2, y2) the upper right corner
--	---

When the ball bounces against the wall, it is struck by an impact with restitution coefficient $k = 0,7$. When hitting the wall, the ball abruptly changes direction and then rolls in the opposite direction. The impact causes the ball to lose speed. If the speed is v immediately before impact, then the speed is $-k v$ immediately after impact.

Once again, it is to be calculated how long the ball will roll and at which position it will come to a standstill. Add the program code accordingly.

Hints

The method `g` already contains an *if*-construct that implements the physical event of the stop. In the same way, add another *if*-construct that implements the impact of the ball against the wall.

In this scenario - unlike the previous exercise - there are two directions of movement: the ball moves to the right before the impact, then to the left. Whether the ball crosses the threshold to standstill was previously checked in the program code using the boolean expression $v < 0$. It is not that simple anymore. Before the impact, the ball moves to the right and it must be checked if $v < 0$ is true, and after the shot, the ball moves to the left and it must be checked if $v > 0$ is true.

Recommendation: Add a boolean variable with the name *rightHandMovement*. To do this, insert the following line into the program code of the *RollingBall* class.

```
public boolean rightHandMovement = true;
```

At the beginning the variable gets the value *true*. Now, the impact event shall not only change the value of v to $-k v$ but also set the *rightHandMovement* variable to false. This boolean value can be accessed to decide, together with the value v , whether the ball crosses the threshold to standstill.