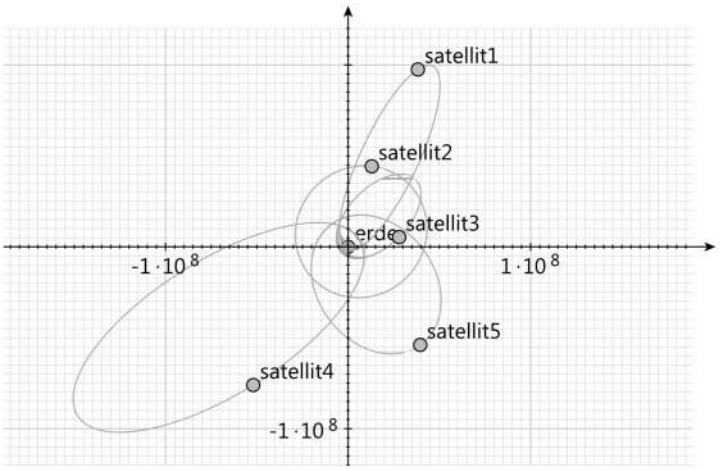
Exercise Sheet 9

Satellites, Object Oriented Programming



Exercise 1

In exercise sheet 8, a physical system was programmed in which a satellite orbits around the earth. All physical variables were scalar variables, i. e. single double values. In this exercise sheet, the same task is to be processed again. However, we will work with vectors. The position of the satellite, its speed and its acceleration shall be represented by vectors. The physical formulas shall be expressed using vector operations (vector addition, scalar product, etc.).

The position of the satellite was previously expressed by the two scalar variables *x* and *y*, its velocity by *vx* and *vy* and its acceleration by *ax* and *ax*. Remove these variables from your program code and add the following object attribute declarations for *r*, *v* and *a* according to the following program code. The variable *r* is to represent the position of the satellite, *v* stands for its speed and *a* for the satellite's acceleration. All three variables have the type *Vector2D*. The three variables *r*, *v,* and *a* are therefore pairs of numbers. Each of the three variables has an *x* and a *y* part of type *double*. The Java expression *r.x* stands for the *x*-part of position *r* and *r.y* stands for the *y*-part of *r*. Correspondingly, *v.x*, *v. y*, *a.x* and *a.y* stand for the *x* and *y* parts of *v* and *a*.

```
import de.physolator.usr.components.*;
import static de.physolator.usr.components.VectorMath.*;
...

public class Satellites extends PhysicalSystem {
    ...

    @V(unit = "m", derivative = "v")
    public Vector2D r = new Vector2D(4e7, 0);

    @V(unit = "m/s", derivative = "a")
    public Vector2D v = new Vector2D(0, 4000);

    @V(unit = "m/s^2")
    public Vector2D a = new Vector2D(0, 0);

    ...
}
```

Note, that the class *Vector2D* is part of the Java package *de.physolator.usr.components*. Therefore, the corresponding import statement is required in the initial area of the program code.

During initialization, values must be assigned to the vector variables. Vector values are generated with the expression:

new Vector2D(x, y)

In Java, this kind of expression is called a constructor invocation. The constructor invocation generates a vector and then assigns two values *x* and *y* to the vector. For *r* these are the values *4e7* and *0*, for *v* they are *0* and *4000* and for *a* they are *0* and *0*.

In exercise sheet 8, annotations were used to define derivation relationships between scalar variables. Derivation relationships can also be defined between vectors. In the above program code, the annotations determine that *v* is the first derivative of *r* and that *a* is the first derivative of *v*. This is done in the same way as for scalar variables. Physical units can also be assigned to vectors. For example, if it is specified for the physical variable *r* that the physical unit is to be *m* (meters), then this not only means that *r* receives the unit *m*, but it also means that the parts of *r*, namely *r.x* and *r.y*, also receive this unit.

From your Java program you can access the *x*- and *y*-parts of your vectors. Therefore, you can perform common scalar arithmetic operations with these *double*-values (addition, multiplication, sinus,...). Alternatively, you can also calculate on the level of the vectors. Rewrite the formulas in method *f* so that you work with vector operations. The class *VectorMath* provides vector operations for this purpose. The vector operations are listed in the following table. Use these operations when implementing the method f.

add(a,b)	Vector addition. Adds two vectors (*Vector2D* objects). The return value is a vector (another *Vector2D* objekt).
add(a,b,c) add(a,b,c,d) add(a,b,c,d,e) ...	Vector addition with three or more vectors.
sub(a,b)	Vector subtraction
mult(p,a)	Scalar product. Multiplies the scalar double value *p* with vector *a*. The return value is a vector.
abs(a)	Computes the amount of some vector *a*. The return value is a scalar value of type *double*.
normalize(a)	Normalization. Computes a vector with the same direction as *a*, but with an amount of 1.
dist(a,b)	Distance between two points *a* and *b*. *a* and *b* are vectors of type *Vector2D,* the return value is of type double.

When programming physical systems for the Physolator, there is one limitation: you must not assign a vector *p* to a vector *v* by an ordinary assignment:

```
v = p;
```

Instead you have to invoke the *set*-method of vector *v* and pass *p* as parameter of the *set*-method.

```
v.set(p);
```

Note this rule when implementing the method *f*. This rule applies only to vectors, not to the scalar variables of type double. If you want to assign a value to any variable z of the type double, this is done with an assignment of the form *z=....;*

In the previous implementation, the satellite was displayed graphically using the *MechanicsTVG* class. The command

```
t.addPointMass("x", "y", "vx", "vy", "ax", "ay");
```

defined that a point-shaped mass is to be represented, which is defined by these six scalar variables. This command can now be deleted without substitution. In the case of physical systems that work with vectors with the names *r*, *v* and a in the form shown above, the *MechanicsTVG* class automatically detects that they are point-shaped masses and displays them without further action.

Exercise 2

Copy the following class *CelestialBody*.

```
import de.physolator.usr.V;
import de.physolator.usr.components.Vector2D;

public class CelestialBody {

    @V(unit = "kg")
    public double m = 5.974E24;

    @V(unit = "m", derivative = "v")
    public Vector2D r = new Vector2D(0, 0);

    @V(unit = "m/s", derivative = "a")
    public Vector2D v = new Vector2D(0, 0);

    @V(unit = "m/s^2")
    public Vector2D a = new Vector2D(0, 0);

    public CelestialBody(double x, double y, double vx, double vy, double ax, double ay,
                         double m) {
        r.set(x,y);
        v.set(vx,vy);
        a.set(ax,ay);
        this.m = m;
    }
}
```

The class *CelestialBody* represents objects that have a mass *m*, a position *r*, a velocity *v*, and an acceleration *a*. *m* is a scalar quantity and *r*, *v* and *a* are vectors. Physical units have already been assigned to the appropriate physical variables and the derivation relationships are also specified.

Both the earth and the satellite are now to be described as instances of the class *CelestialBody*. Copy the following program code:

```
import static de.physolator.usr.components.VectorMath.*;
import static java.lang.Math.pow;
import mechanics.tvg.MechanicsTVG;
import de.physolator.usr.*;

public class Satellites extends PhysicalSystem {

    @V(unit = "m^3/kg s^2")
    double G = 6.67428e-11;

    public CelestialBody earth = new CelestialBody(0, 0, 0, 0, 0, 0, 5.974E24);
    public CelestialBody satellite = new CelestialBody(4e7, 0, 0, 4000, 0, 0, 100);

    public void f(double t, double h) {
        // space for your formulas
    }

    public void initGraphicsComponents(GraphicsComponents g, Structure s, Recorder r,
            SimulationParameters sp) {
        MechanicsTVG t = new MechanicsTVG(this, s, r);
        double p = 1.2e8;
        t.geometry.setUserArea(-p, p, -p, p);
        t.showPaths = true;
        t.showVelocity = false;
        t.showAcceleration = false;
        t.showLabels = true;
        g.addTVG(t);
    }

    public void initSimulationParameters(SimulationParameters s) {
        s.fastMotionFactor = 20000;
    }
}
```

Two instances of the class *CelestialBody* have already been created in this program code. One of the two instances is stored in the variable *earth*, the other in the variable *satellite*. The constructor invocations

new CelestialBody (...)

generate the instances of the *CelestialBody* class. The parameter values of the constructor calls determine the initial values of the objects: their position *r*, their velocity *v*, their acceleration *a* and their mass *m*.

Both the earth and the satellite each have an *r* attribute, a *v* attribute, an *a* attribute and an *m* attribute. These attributes can be accessed via the variables *earth* and *satellite*. Thus, *satellite.r* stands for the position of the satellite, *satellite.v* for its speed, *satellit.a* for its acceleration and *satellit.m* for its mass. Analogous to this is *earth.r*, *earth.v*, *earth.a* and *earth.m* stand for the position, speed, acceleration and mass of the earth.

Calculate the acceleration of the satellite in the method f using the positions and masses of the moon and earth!

Annotations

The class *MechanicsTVG* automatically draws all point-shaped masses of the physical system, i. e. all physical subcomponents that have an *r* component, a *v* component and an *a* component. In our case, *MechanicsTVG* automatically draws earth and satellite. If, as in this case, several point-shaped masses are drawn on the screen, there could be confusion. The assignment

t.showLabels = true;

ensures that the point-shaped masses are labeled with their variable names.

Exercise 3

In the *CelestialBody* class, add the following method *gravitationalAcceleration*.

```
public Vector2D gravitationalAcceleration(Vector2D p) {
    return ...; // space for your program code
}
```

The *gravitationalAcceleration* method is part of every *CelestialBody* object. The method is intended to determine which gravitational acceleration a point-shaped mass experiences through the *CelestialBody* object when the point-shaped mass is located at the position *p*. The *CelestialBody* object determines the gravitational acceleration of a point-shaped mass. The method accesses the object attributes of the *CelestialBody* object *r*, *v*, *a* and *m* and the parameter *p* and calculates the acceleration in the form of a vector. The calculated acceleration is returned as the method's return value. Recommendation: In the *CelestialBody* class, add a physical variable *G* with the gravitational constant.

This new method in the class *CelestialBody* can now be used in all *CelestialBody* objects. In the class Satellites, earth *gravitationalAcceleration(p)* can be used to determine the gravitational acceleration originating from the earth at any point *p* and, in analogy *satellite.gravitationalAcceleration(p)* computes the gravitational acceleration originating from the satellite. Use this new method when programming f!

Exercise 4

So far, only one satellite orbits the Earth. Now, five satellites are to orbit the Earth. To do this, create five variables *satellite1*, satellite2,... *satellite5* in the class *Satellites* and assign different starting positions and speeds to these satellites! Make sure that their acceleration is calculated in method *f*!