Lab 1 in TND002

Mark Eric Dieckmann

January 17, 2023

1 Summary

The aim of this lab is to familiarize you with object-oriented programming and how it differs from structured programming. Structured programming is based on loops, decisions, and subprograms. Object-oriented programming groups variables and methods that belong together into classes. You implement methods that deal with vector operations. You implement a structured program in the first part of the lab and an object-oriented one in the second. Revise lectures 1-4 and lessons 1a/b before the lab.

2 First part

Create in the Java project *TND002Labs* (or another name of your choice) the package *lab1* in a directory that is permanent (not on temporary disk space). Create the class **Lab1a** with a method *main(..)*. We consider in this part vectors, which are static (fixed size) arrays with three elements of type *double*. Declare and initialize in *main(..)* the vector *da1* with the values (1.0, 2.0, 3.0) and *da2* with the values (0.0, -0.5, -2.0). You can initialize the array either directly when you declare the reference variable:

```
double[] da1 = {1.0, 2.0, 3.0};
```

or by instantiating the array first and then filling it up element by element:

```
double[] da1=new double[3]; double[0]=1.0; double[1]=2.0; double[2]=3.0;
```

What is the result of *System.out.println(da1);* Explain that to the lab assistant.

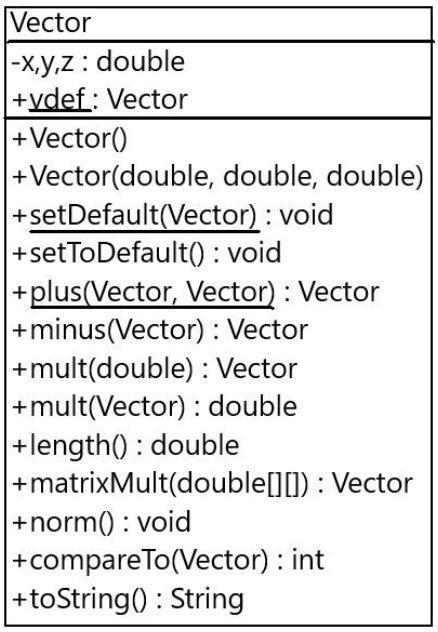
Implement the static method *addition(arg1, arg2)* in the class **Lab1a**, where *arg1* and *arg2* are vectors. Add up both vectors in that method and return the result. The arguments and the return type are the same as you use for declaring the vectors. Implement a second method, which subtracts *arg2* from *arg1* and returns the result vector.

Implement the method *printVector(arg)* that takes in the vector *arg* and has no return value. It should check first if the vector has three elements. If this is not the case, it should do nothing. The method creates a formatted string "Vector = (x, y, z)", where the vector components *x, y, z* are expressed as doubles with two digits before and one after the "decimal point" (a decimal point is a ',' in Java). Print the result to the console in this method.

3 Second part

Class Vector:

The obvious object in our code is the vector. It makes sense to design a dedicated class that contains the attributes (variables) *x,y,z* of the vector and methods that apply only to vectors. Create in *lab1* the class **Vector** and the class **Lab1b** with the main method (You can have two classes with *main()* in the same package but you can only run one at a time). **Vector** should contain the following variables and methods.



The instance variables *x,y,z* correspond to the coordinates of the vector.

The class variable *vdef* is a vector that is initialized as (0.0, 0.0, 0.0).

The default constructor *Vector*() calls *setToDefault()*.

Vector(d1, d2, d3) sets the values of *x, y, z* to those in the argument list.

setDefault(arg) sets *vdef* to *arg*.

setToDefault() sets the values of *x, y, z* to those of *vdef*. Remember that private variables in a class are visible to other instances of the same class.

plus(arg1,arg2) adds up *arg1* and *arg2* and returns the result.

minus(arg) subtracts *arg* from the vector that calls this method (calling vector) and returns the result.

mult(double arg) multiplies *arg* with the coordinates of the calling vector and returns the result. The second method with the same name *mult(Vector arg)* calculates the dot product between the calling vector and *arg* and returns the result.

length() returns the length of the calling vector.

norm() normalizes the calling vector to the return value of *length()*.

matrixMult(arg) takes the 3 by 3 matrix *arg*, multiplies it to the calling vector, and returns the result. It should return the calling vector if *arg* is not a 3 by 3 matrix.

compareTo(arg) compares the lengths of the calling vector and *arg*. It returns 0 if both lengths are equal, it returns 1 if the calling vector is longer than *arg* and -1 otherwise.

toString() returns a formatted string that equals the one in *printVector()* in **Lab1a**.

Class Lab1b:

You implement the class **Lab1b**, which contains only the method *main(..)*. Use *System.out.println(..)* to write a vector or the return value of a method to the console. Implement the following lines that test all methods of **Vector**.

Write out the content of *vdef* before you create the first instance of **Vector**.

Create the vector *v1* using the default constructor and write out its content.

Change the coordinates of the default vector *vdef* to (1.0, 2.0, 3.0), use the method *setToDefault()* of *v1*, and write out the content of *v1*.

Create the vector *v2* with the coordinates (1.0, 1.0, 2.0) and write out its content.

Write out the string "Length: " followed by the length of *v1*.

Initialize the matrix *m1* with the elements ((1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)), and write out the result of calling the method *matrixMult(m1)* of *v1*.

Initializing a 2D array works in the same way as initializing a 1D array (first part of the lab). The declaration uses *double[][]* and you initialize it with the construct

```
{{a11, a12, a13}, {a21, a22, a23}, {a31, a32, a33}}
```

Initialize the matrix *m2* with the elements ((1.0, 0.0), (0.0, 1.0)), and write out the result of calling the method *matrixMult(m2)* of *v1*.

Initialize the matrix *m3* with the elements ((0.0, 1.0, 0.0), (1.0, 0.0, 0.0), (0.0, 0.0, 1.0)), and write out the result of *matrixMult(m3)* of *v1*.

Write the result of adding *v1* and *v2* to the console.

Write the result of subtracting *v2* from *v1* to the console.

Write the result of 2**v1* to the console.

Write the result of the dot product between *v1* and *v2* to the console.

Write the result of the length comparison of *v1* and *v2* to the console.

Normalize *v1* and *v2* and write the scalar products of (a) *v1* and *v2* and (b) *v2* with itself to the console.