Lab exercise 1: Implementing a class
24292-Object Oriented Programming

1 Introduction

The aim of this lab session is to implement the design of Seminar 1, i.e. classes representing geometrical points and distance matrices. It is also necessary to create and manipulate instances of these classes to ensure that the classes are correctly defined. Optionally, you can create an instance of the class `DisplayMatrix` which is provided as part of the code package for Lab Session 1. The class `DisplayMatrix` makes it possible to display a distance matrix as part of a graphical interface. The session is mandatory and you have to hand in the source code of the Java project and a document describing the implementation.

2 Geometric Point

The first thing to do is to implement the code for the class that represents geometric points. Once this implementation is finished, you should create a class called `TestPoint` whose purpose is to test that the class is correctly implemented. In `TestPoint` you should define a main method where you create multiple instances of the class, apply the available methods and print the results of these methods to the screen. Hint: the static method `Math.sqrt` computes the square root of a given number.

3 Distance Matrix

To implement the class that represents distance matrices it is a good idea to use the existing Java class `LinkedList`. The following code illustrates how to use the `LinkedList` class:

```
import java.util.LinkedList;
public class TestLinkedList {
        public static void main(String[] args) {
                LinkedList<String> list;
                list = new LinkedList<String>();
                list.add("anElement");
                list.add("anotherElement");
                System.out.println(list.size());
                String firstElement = list.get(0);
                System.out.println(firstElement);
        }
}
```

The type of list element (`String` in the example) can be changed to any existing class if desired. It is not possible, however, to create a `LinkedList` with elements of a basic type such as `int`.

To help with the implementation of the distance matrix class we provide a sample design of the class below:

DistanceMatrix
−cities: list of Point −matrix: array of array of real
+DistanceMatrix(): void +addCity(x: real, y: real, name: string): void +getCityName(index: integer): string +getNoOfCities(): integer +createDistanceMatrix(): void +getDistance(index1: integer, index2: integer): real

You are free to use your own design if you want. Be aware, however, that the class `DisplayMatrix` will only work if the methods of `DistanceMatrix` are exactly those listed above.

Just as for the class for geometric points, you should create a new class called `TestDistanceMatrix` with a main method that tests whether the class `DistanceMatrix` has been correctly implemented. Again, you should create instances of the class, call the different methods and print the results to the screen. If necessary, invent the coordinates of fictional cities.

4 (Optional) Display the Distance Matrix

To display the distance matrix we have created a graphical interface represented by the class `DisplayMatrix` in the compressed file `P1Code.zip` which you can download from the Aula Global. To make the program work you have to follow these steps:

1. Decompress the file `P1Code.zip`.
2. Copy the files `DisplayMatrix.java` and `Matrix.java` to the source code directory of your Netbeans project (where the other Java files are located).
3. If necessary, change the package of each class so that it corresponds to the package defined for the project.
4. Modify the class header of `DistanceMatrix` as follows:

```
public class DistanceMatrix implements Matrix {
```

5. If necessary, modify the methods of `DistanceMatrix` such that their definitions correspond *exactly* to the methods in `Matrix.java`.

6. Create a new class `TestDisplayMatrix` with a main method that contains the following code:

```
DistanceMatrix matrix = new DistanceMatrix();
DisplayMatrix display = new DisplayMatrix( matrix );
display.setVisible( true );
```

7. Finally compile and execute the main method of `TestDisplayMatrix`.

5 Documentation

Apart from the source code, you should also hand in a document that outlines the solution of the problem. To elaborate the document you can use the following guidelines regarding the content:

1. An introduction where the problem is described. For example, what should the program do? Which classes do you have to define? Which methods do you have to implement for these classes?

2. A description of possible alternative solutions that were discussed, and a description of the chosen solution and the reason for choosing this solution rather than others. It is also a good idea to mention the related theoretical concepts of object-oriented programming that were applied as part of the solution.

3. A conclusion that describes how well the solution worked in practice, i.e. did the tests show that the classes were correctly implemented? You can also mention any difficulties during the implementation as well as any doubts you might have had.

6 Submission

To submit your solution (code + document), simply create a directory "Lab1" in your Git repository and upload all files there (remember that you have to add all files, commit the changes and push the new version of the repo). The deadline for submission is Tuesday 26 October (prior to the next lab session).