Pablo Fernández
Clara López
S101, P101

Lab 1 Report

Introduction

This report presents the work done in the first laboratory project of the Object-Oriented Programming course.

The aim of this project was to get more familiar with the basic concepts of programming with an object-oriented language, as well as to create an interesting program while applying the concepts learned in theory sessions about classes, attributes, methods, how classes are related to one another, etc... More specifically, we had to implement the classes we had designed in a previous seminar session, which were Geometric Point and Distance Matrix. These classes, or the program in general, should be able to compute some basic operations related to Geometry, mainly the distance between two points, and store the respective results in a matrix. Some other classes were created, whose goal was to test the code and check if the previous classes were correctly implemented.

When creating a class, it is important to bear in mind that there are some fundamental methods. The most important one is the constructor method, as we would not be able to create instances of a class otherwise. Other important methods are the getters and setters, to modify or use the values of the attributes without making the attributes public. In this case, they are only implemented in the first-mentioned class because they are not needed afterwards. Moreover, the methods we had to implement for the second class were established beforehand.

Pablo Fernández
Clara López
S101, P101

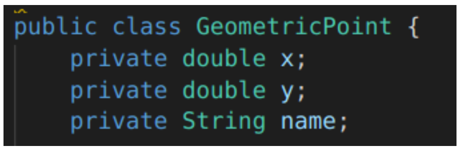
Description

We actually did not discuss multiple solutions, as our approach was just to code together and to build the code step by step. In the end, we only discussed small unimportant details. Anyway, here is the description of our solution:

Chosen solution

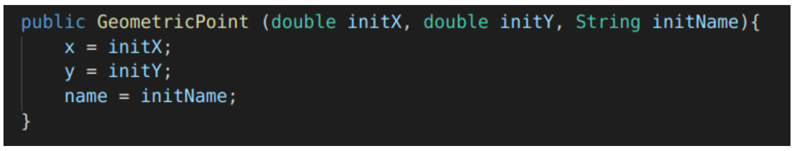
1. Geometric Point

The first part consisted in implementing a code for the *GeometricPoint* class.



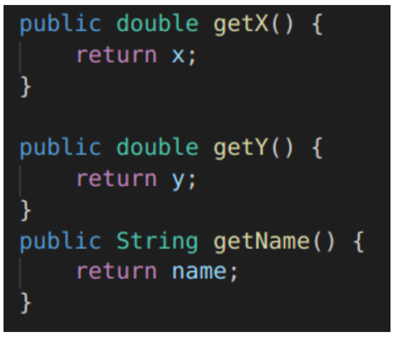
```java
public class GeometricPoint {
    private double x;
    private double y;
    private String name;
```

Here we have implemented the attributes of the class.



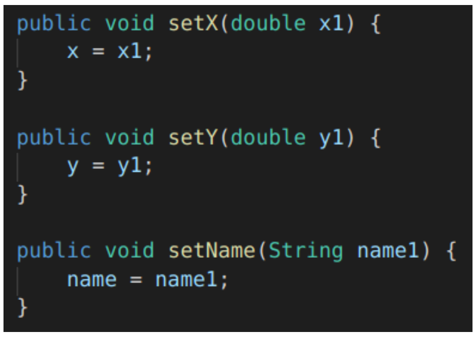
```java
public GeometricPoint (double initX, double initY, String initName){
    x = initX;
    y = initY;
    name = initName;
}
```

Constructor method, for creating instances and initializing them with specific values .



```java
public double getX() {
    return x;
}

public double getY() {
    return y;
}
public String getName() {
    return name;
}
```

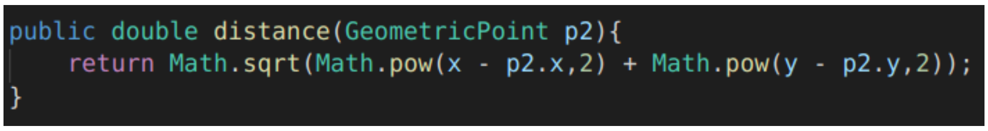
Getters, for using the values of the different attributes. We shall create one for each attribute.



```java
public void setX(double x1) {
    x = x1;
}

public void setY(double y1) {
    y = y1;
}

public void setName(String name1) {
    name = name1;
}
```

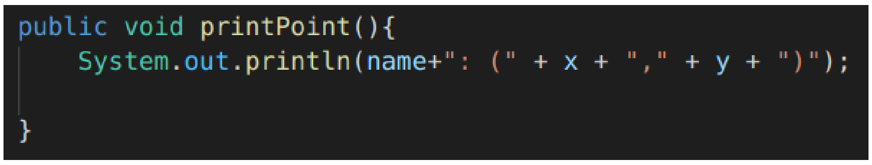
Setters, for changing the values of the different attributes. Once again, one for each attribute.

This class also has another two methods, needed to perform the tasks we are working on:



```java
public double distance(GeometricPoint p2){
    return Math.sqrt(Math.pow(x - p2.x,2) + Math.pow(y - p2.y,2));
}
```

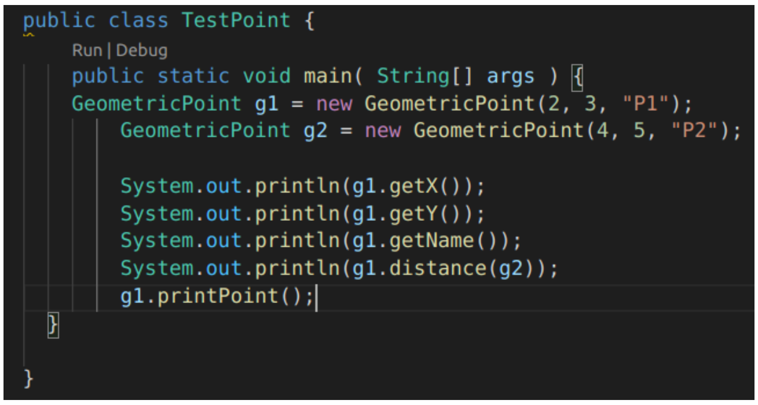
distance is used to calculate the distance between two points, by applying the formula. It needs to receive another point as a parameter, otherwise it has no distance to calculate as there would only be one point.



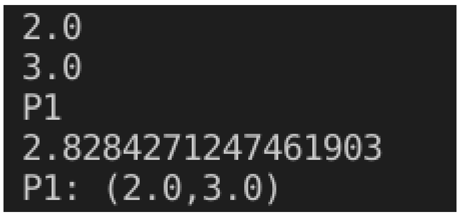
```java
public void printPoint(){
    System.out.println(name+": (" + x + "," + y + ")");

}
```

printPoint prints on the screen the name followed by the coordinates X and Y of the geometric point, using the adequate standard notation.

Then, we created a class called *TestPoint* where we defined the main method, in which we created different instances of the class in order to see that *GeometricPoint* worked.



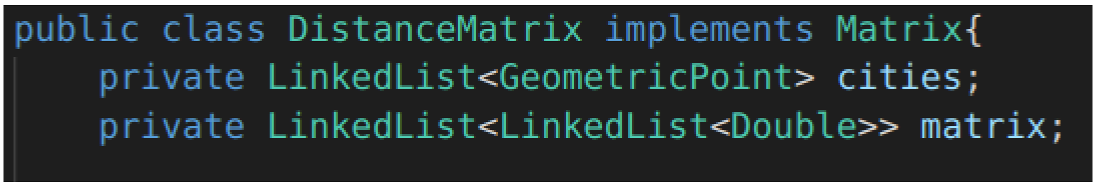
```java
public class TestPoint {
    Run | Debug
    public static void main( String[] args ) {
    GeometricPoint g1 = new GeometricPoint(2, 3, "P1");
        GeometricPoint g2 = new GeometricPoint(4, 5, "P2");

        System.out.println(g1.getX());
        System.out.println(g1.getY());
        System.out.println(g1.getName());
        System.out.println(g1.distance(g2));
        g1.printPoint();
  }

}
```



```
2.0
3.0
P1
2.8284271247461903
P1: (2.0,3.0)
```

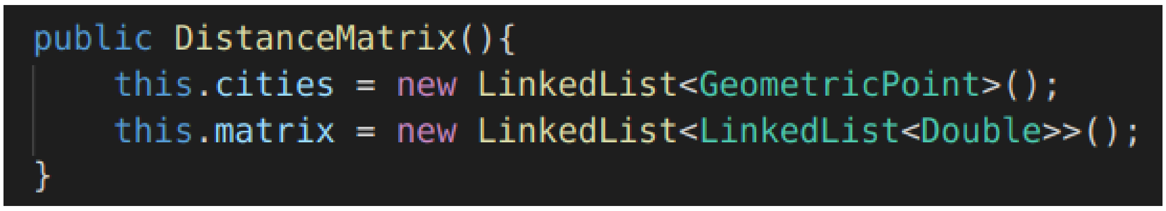
2. Distance Matrix

At this point, we had to implement a class that represents in a matrix the distance between cities. In this exercise, we have used an existing Java class, *LinkedList*. The code is already modified as suggested in the optional part.



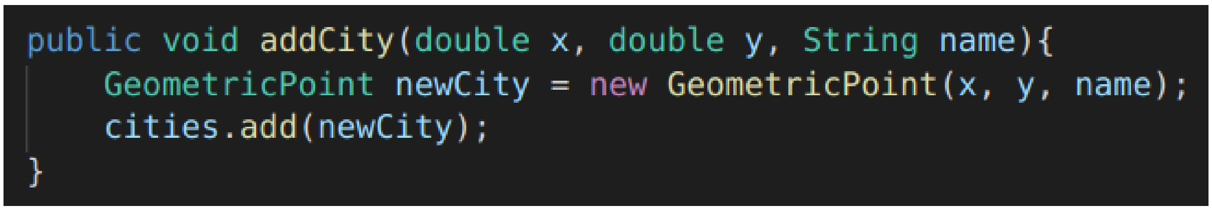
```java
public class DistanceMatrix implements Matrix{
    private LinkedList<GeometricPoint> cities;
    private LinkedList<LinkedList<Double>> matrix;
```

Here we have implemented the attributes of the class.



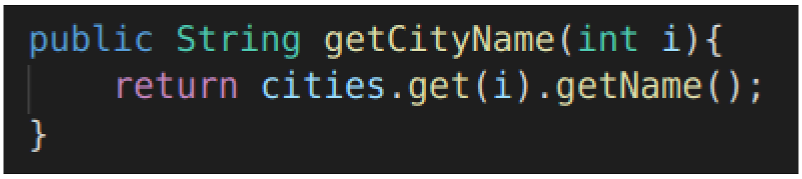
```java
public DistanceMatrix(){
    this.cities = new LinkedList<GeometricPoint>();
    this.matrix = new LinkedList<LinkedList<Double>>();
}
```

Constructor method. We use the keyword *this* so that the program knows what we are referring to, in case there are repeated names (even though this is not the case).



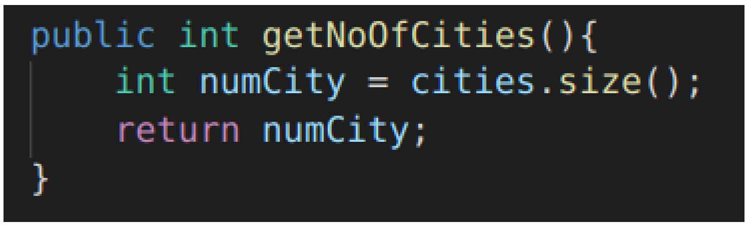
```java
public void addCity(double x, double y, String name){
    GeometricPoint newCity = new GeometricPoint(x, y, name);
    cities.add(newCity);
}
```

This method adds a point to the cities array by receiving its attributes.



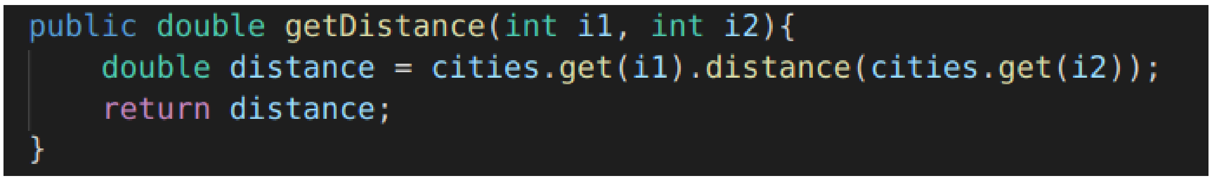
```java
public String getCityName(int i){
    return cities.get(i).getName();
}
```

This method returns the name of the city whose index is received as an input.



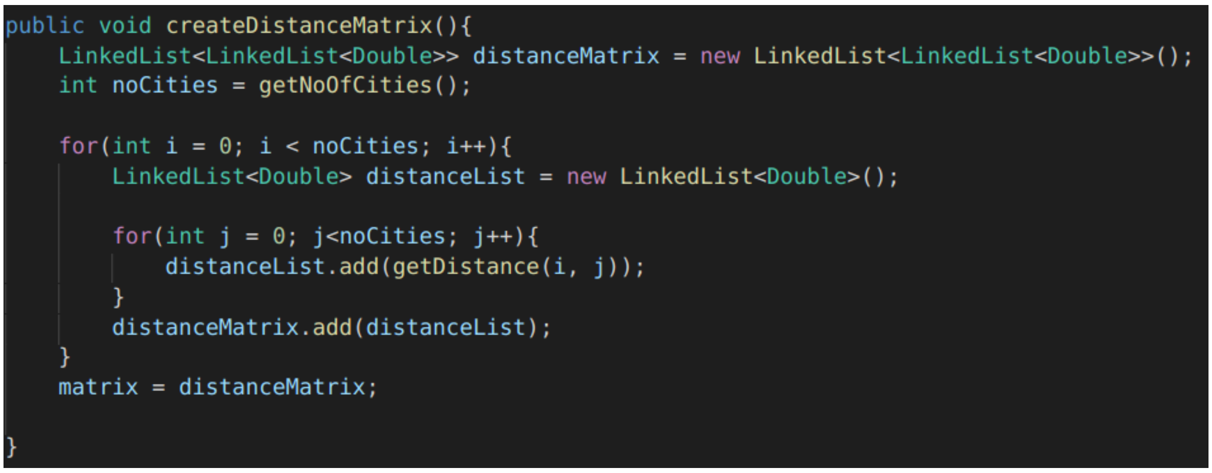
```java
public int getNoOfCities(){
    int numCity = cities.size();
    return numCity;
}
```

This method uses the method size(), defined for the LinkedList class, to return the number of elements in the cities array.



```java
public double getDistance(int i1, int i2){
    double distance = cities.get(i1).distance(cities.get(i2));
    return distance;
}
```

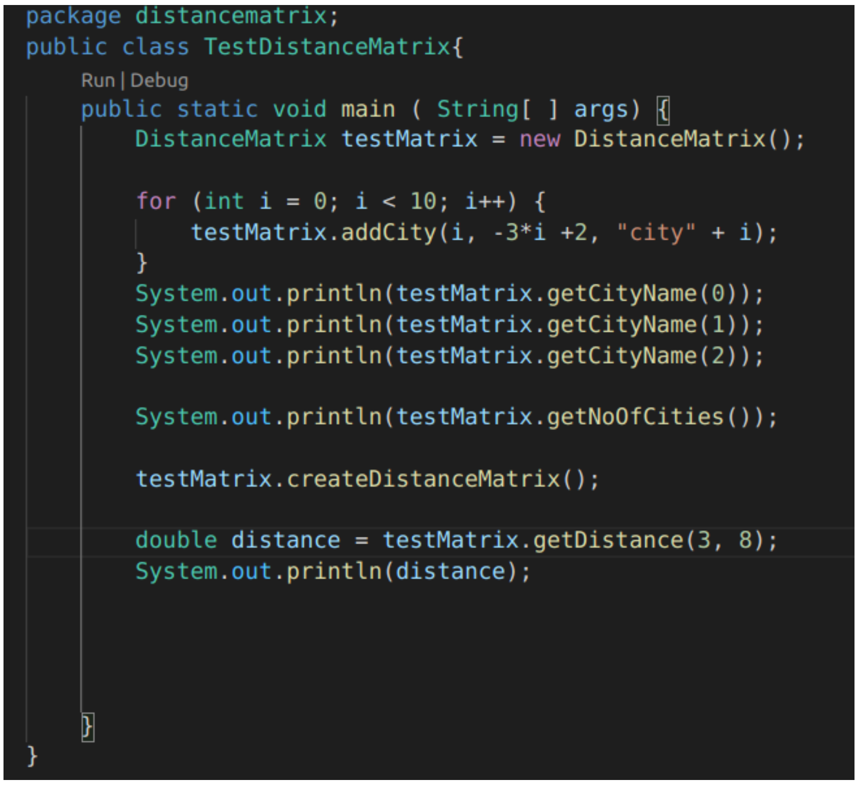
This method uses the previously defined method for GeometricPoint that finds the distance between two points, and applies it to the array (through the indexes)



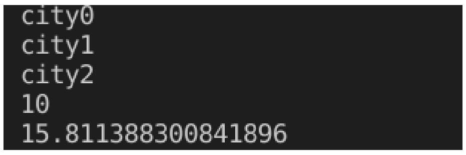
```java
public void createDistanceMatrix(){
    LinkedList<LinkedList<Double>> distanceMatrix = new LinkedList<LinkedList<Double>>();
    int noCities = getNoOfCities();

    for(int i = 0; i < noCities; i++){
        LinkedList<Double> distanceList = new LinkedList<Double>();

        for(int j = 0; j<noCities; j++){
            distanceList.add(getDistance(i, j));
        }
        distanceMatrix.add(distanceList);
    }
    matrix = distanceMatrix;

}
```

In this method, we iterate through the elements in the array in order to get the distance between themselves and add it to the matrix we are creating. We create an auxiliary variable, distanceList, because we have a list of distances for each element in the array, and the resulting matrix is formed by all the lists.

Then, we created a class called *TestDistanceMatrix* where we defined the main method, in which we created different instances of the class in order to see that *DistanceMatrix* worked.



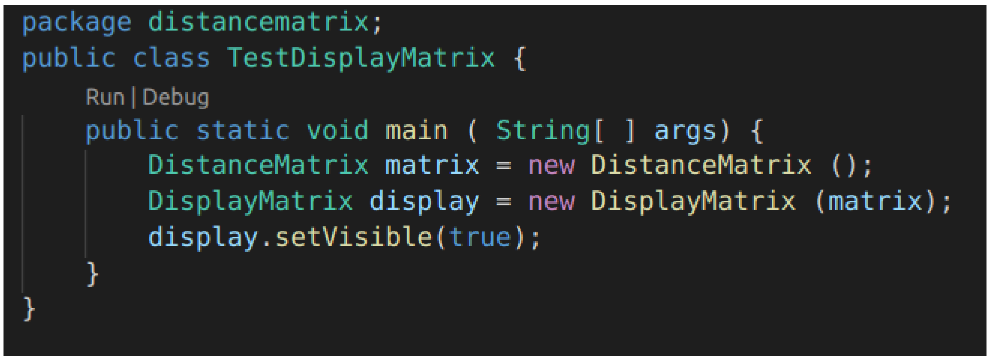
```
package distancematrix;
public class TestDistanceMatrix{
    Run | Debug
    public static void main ( String[ ] args) {
        DistanceMatrix testMatrix = new DistanceMatrix();

        for (int i = 0; i < 10; i++) {
            testMatrix.addCity(i, -3*i +2, "city" + i);
        }
        System.out.println(testMatrix.getCityName(0));
        System.out.println(testMatrix.getCityName(1));
        System.out.println(testMatrix.getCityName(2));

        System.out.println(testMatrix.getNoOfCities());

        testMatrix.createDistanceMatrix();

        double distance = testMatrix.getDistance(3, 8);
        System.out.println(distance);




    }
}
```



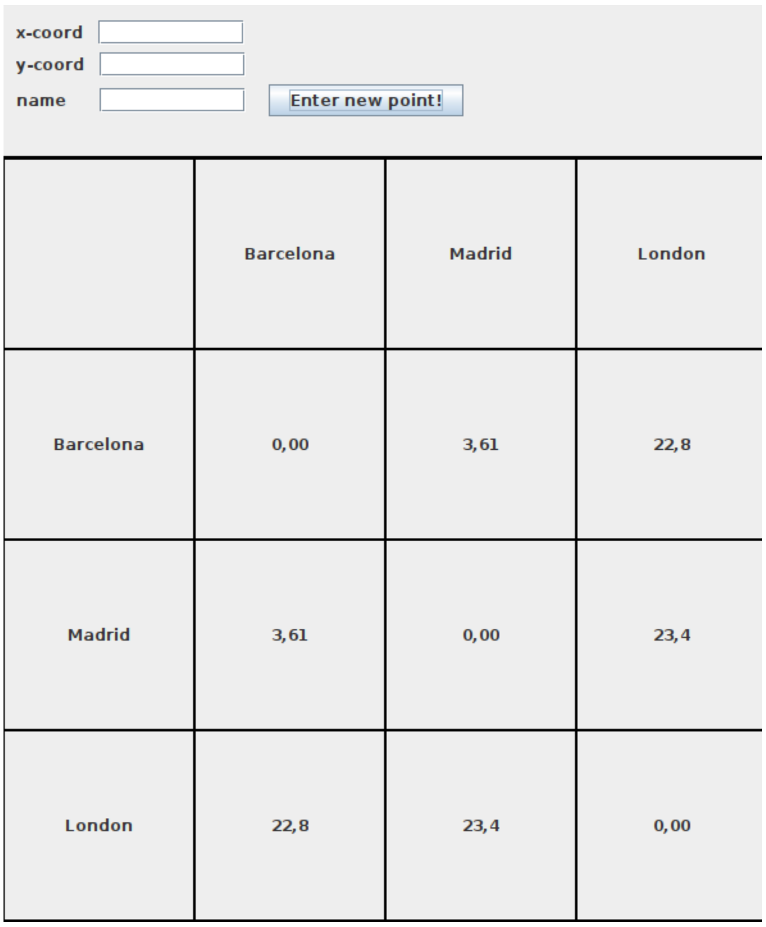
```
city0
city1
city2
10
15.811388300841896
```

3. Display the distance matrix

This part consisted only in following the instructions in the exercise document.



```
package distancematrix;
public class TestDisplayMatrix {
    Run | Debug
    public static void main ( String[ ] args) {
        DistanceMatrix matrix = new DistanceMatrix ();
        DisplayMatrix display = new DisplayMatrix (matrix);
        display.setVisible(true);
    }
}
```



	Barcelona	Madrid	London
Barcelona	0,00	3,61	22,8
Madrid	3,61	0,00	23,4
London	22,8	23,4	0,00

Conclusion

To sum up, we are really satisfied with our work in this lab because, even though we were a little lost at the beginning, we put some effort into it and in the end everything went well. It was very useful to create the Test classes, as we could see exactly where our code was not working properly and try to correct.