GROUP 4

Lab exercise 1: Implementing a class

24292-Object Oriented Programming

1-. Introduction

For this lab session we are going to base on our design we did on Seminar one with its correct implementations and definitions. For this lab we can divide our work in three parts which are the following. The first one is the creation of a class called GeometricPoint, with its testing file. The second one is the creation of a class called DistanceMatrix, with its testing file. Lastly, we created a class called DisplayMatrix, with its testing file, as a bonus point.

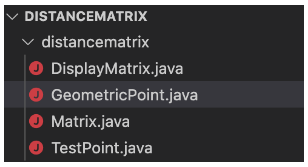
Furthermore, we will apply theoretical and practical concepts seen in the theory and seminar sesion. We will also have special consideration with LinkedLists.

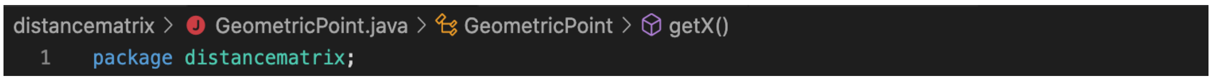
All the files are related between them. As forwarding we will explain:

The first part is the definition of a GeometricPoint. It has a coordinate x, y and a name. Some properties and actions (especially to calculate distance). In the second part DistanceMatrix GeometricPoint will be the same as a city as they have coordinates(x,y) and what we want to have in the array is the distance between the list of cities. We implemented this and then we can see it in a more visual way in the bonus implementation DisplayMatrix.

2-. Description

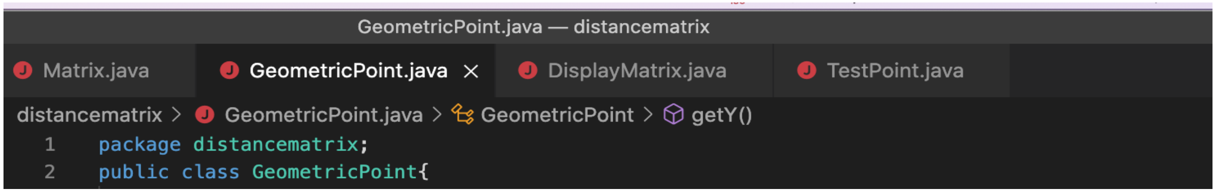
First of all we need for its correct performance to have in mind that all the files we create will have this first line in common. Which makes all to be correlated and belong to the same package *distancematrix,* which is the folder where all of our files will be stored.




distancematrix > GeometricPoint.java > GeometricPoint > getX()

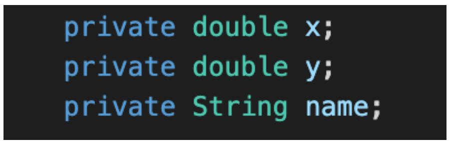
```
package distancematrix;
```

Then, we create a file in java code that will include the class that represents geometric points. It has to be named the same way as the public, because we want it to be accessible for the user to see it and operate with it. This class is part of the project ***GeometricPoint.*** The correct notation for classes is that the first letter is in uppercase and if it is composed by more words these also will have its first letter in uppercase.


GeometricPoint.java — distancematrix
Matrix.java
GeometricPoint.java
DisplayMatrix.java
TestPoint.java
distancematrix > GeometricPoint.java > GeometricPoint > getY()

```
package distancematrix;
public class GeometricPoint{
```

Consequently, the class *GeometricPoint* is composed of two parts that need to be defined inside which are: the attributes, which define the part of characteristics, and the methods, which define the functioning part. Doing a design on paper is the best way of organizing these ideas beforehand. The notation for attributes and methods is lowercase and if it's composed by more words these will have its first letter in uppercase. We follow our design on GeometricPoint done in Seminar 1:

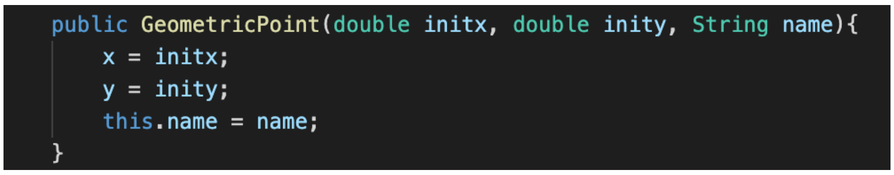


```
private double x;
private double y;
private String name;
```

On the one hand, we will start as usual by defining the attributes. We do not want them to be accessible to manipulate for the user as it will change the object itself and would be dangerous for our code. Usually all attributes are private for all projects. We foment encapsulation in this subject, leaving visible only the important details. For a GeometricPoint when we think about it we think essentially in a coordinate system X and another Y (in 2D) of type double, as we can have real numbers with decimals. Also another important characteristic to differentiate it is the name that would be of type String.

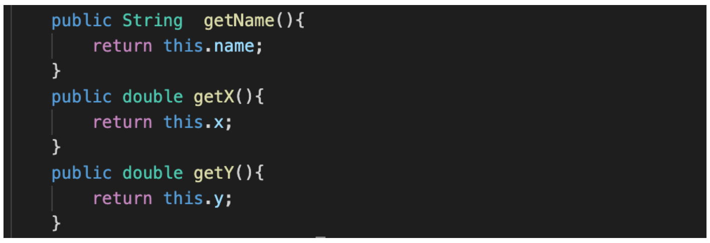
On the other hand, we have the methods. These will manage the class actions. There are different types of methods, as we can have:

1-. **The constructor:** each class needs a constructor. It is essential to be public. This takes the same name as the class and is defined by its attributes. It also creates the instances. To create an instance for the variable y, we use "inity" and the same for the variable x, which inits the variable and creates the instance in the same way. For the variable name we use "this." to refer to this concrete instance.



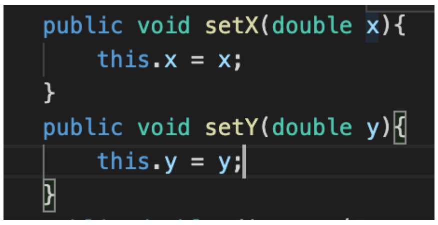
```
public GeometricPoint(double initx, double inity, String name){
    x = initx;
    y = inity;
    this.name = name;
}
```

2-. **Getters:** makes the user able to see the value of the attribute on which is based the getter in the moment it is called: returns the attribute's value. We are going to implement three, one for each of the attributes. Nevertheless in our design of Seminar 1 there was not one getter for the attribute name. That is why now, we have implemented one in the code as it was necessary. We can notice in our code that it is public as we want to make it accessible to the user, the keyword "this." is used to access each of the attributes and the type of each getter is the same as the values that we will be returned.



```
public String  getName(){
    return this.name;
}
public double getX(){
    return this.x;
}
public double getY(){
    return this.y;
}
```

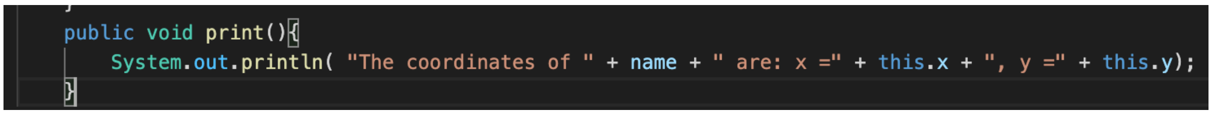
3-. **Setters:** allows to change an attribute value. We are going to implement two attributes as we do not want to change the name in any of the cases. These are public for the user, as we want to make it accessible. Then they are type void as are functions that change, we also use "this.", and after using it, we equalize it to each of the values in the parenthesis(doubles) as that would be the new value to the variables y or x. Also, the setters do not return anything.



```java
public void setX(double x){
    this.x = x;
}
public void setY(double y){
    this.y = y;
}
```

4-. **Other methods that act:** all are voids and public, meaning that they are accessible to users.

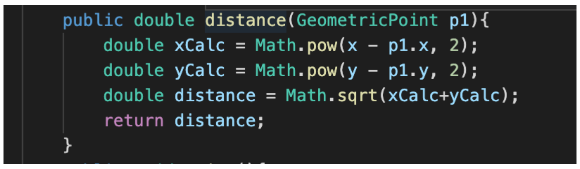
- **print**: this function prints out of the system by the library java util which we import and t we use `System.out.println()` . After this all the essential information we wanted from our GeometricPoint will be printed, concatenated with a + and using "this." to insert and refer to attributes. Then, the coordinates of the GeometricPoint will be printed.



```java
public void print(){
    System.out.println( "The coordinates of " + name + " are: x =" + this.x + ", y =" + this.y);
}
```

- **distance**: will compute the distance between two GeometricPoints by the following formula for which we are going to use Math.pow and Math.sqrt(to make the root). An alternative solution was to multiply it two times but this way was much more visual. Then, the code will look like the following image:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



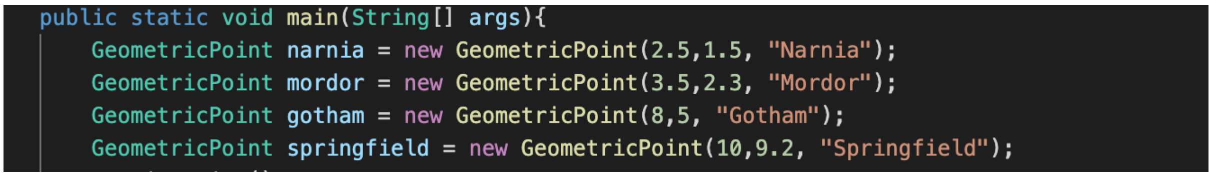
```java
public double distance(GeometricPoint p1){
    double xCalc = Math.pow(x - p1.x, 2);
    double yCalc = Math.pow(y - p1.y, 2);
    double distance = Math.sqrt(xCalc+yCalc);
    return distance;
}
```

Some annotations about the code:

- We return a double value that we called it equal to the method. Therefore after setting the privacity as public, we set the type as double.
- p1.x: from p1 that is the one we give as an input in order to access its attribute x.
- x and y from the other point from which we want to measure the distance. We will call the method inside the class of that point as it is the one we have accessed. By its method distance. With itself coordinates and the ones of p1.

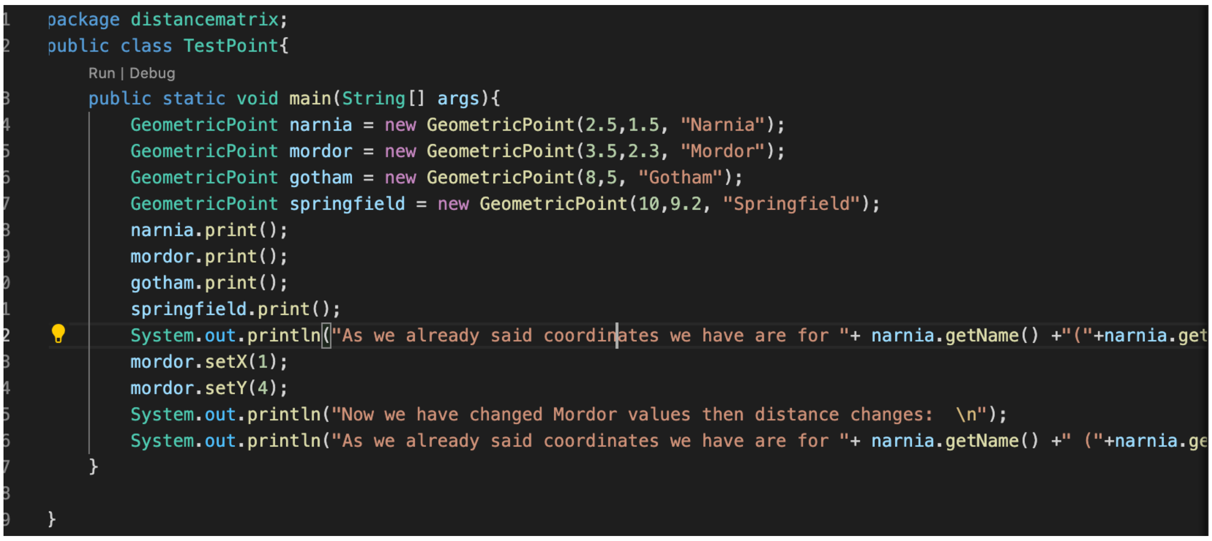
To test the code in order to work properly and is undoubtedly well-implemented we create a new file called ***TestPoint*** which will be a class in which we are going to test points we are going to declare. Already know from the last lesson, that the type of new points will be GeometricPoint.

GROUP 4



```java
public static void main(String[] args){
    GeometricPoint narnia = new GeometricPoint(2.5,1.5, "Narnia");
    GeometricPoint mordor = new GeometricPoint(3.5,2.3, "Mordor");
    GeometricPoint gotham = new GeometricPoint(8,5, "Gotham");
    GeometricPoint springfield = new GeometricPoint(10,9.2, "Springfield");
```

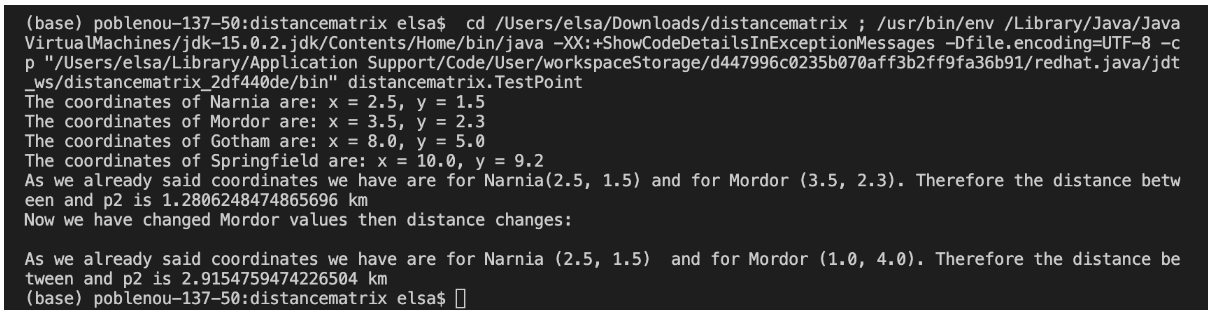
With a little bit of sense of humor and the correct attributes we create 4 new GeometricPoints: Narnia, Mordor, Gotham and Springfield.



```java
package distancematrix;
public class TestPoint{
    Run | Debug
    public static void main(String[] args){
        GeometricPoint narnia = new GeometricPoint(2.5,1.5, "Narnia");
        GeometricPoint mordor = new GeometricPoint(3.5,2.3, "Mordor");
        GeometricPoint gotham = new GeometricPoint(8,5, "Gotham");
        GeometricPoint springfield = new GeometricPoint(10,9.2, "Springfield");
        narnia.print();
        mordor.print();
        gotham.print();
        springfield.print();
        System.out.println("As we already said coordinates we have are for "+ narnia.getName() +"("+narnia.get
        mordor.setX(1);
        mordor.setY(4);
        System.out.println("Now we have changed Mordor values then distance changes:  \n");
        System.out.println("As we already said coordinates we have are for "+ narnia.getName() +" ("+narnia.ge
    }

}
```

Then, in order to implement the code and test our class, GeometricPoint, we opted for the following way. First, try print() with each of the points. Second, using getters to make an introduction before computing the distance and then compute the distance by trying the distance method. Finally we change the values for a GeometricPoint by using the setters and then repeat the procedure which tests with the setters values.

The output of the previous code is the following:

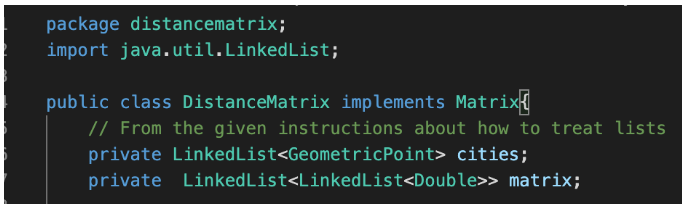


```
(base) poblenou-137-50:distancematrix elsa$  cd /Users/elsa/Downloads/distancematrix ; /usr/bin/env /Library/Java/Java
VirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -c
p "/Users/elsa/Library/Application Support/Code/User/workspaceStorage/d447996c0235b070aff3b2ff9fa36b91/redhat.java/jdt
_ws/distancematrix_2df440de/bin" distancematrix.TestPoint
The coordinates of Narnia are: x = 2.5, y = 1.5
The coordinates of Mordor are: x = 3.5, y = 2.3
The coordinates of Gotham are: x = 8.0, y = 5.0
The coordinates of Springfield are: x = 10.0, y = 9.2
As we already said coordinates we have are for Narnia(2.5, 1.5) and for Mordor (3.5, 2.3). Therefore the distance betw
een and p2 is 1.2806248474865696 km
Now we have changed Mordor values then distance changes:

As we already said coordinates we have are for Narnia (2.5, 1.5)  and for Mordor (1.0, 4.0). Therefore the distance be
tween and p2 is 2.9154759474226504 km
(base) poblenou-137-50:distancematrix elsa$ 
```

As we said in the introduction, our code is divided in two parts. Now we are going to go into the second part. In this one we will not go much further into privacity and theoretical concepts as we considered that this is a matter deeply introduced in the first part.

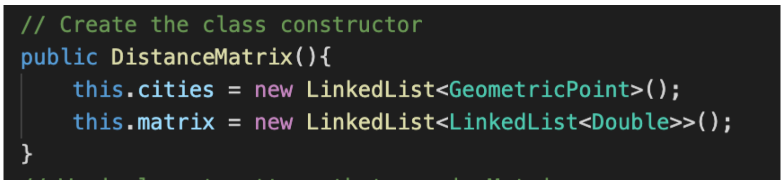
GROUP 4

Now, we will introduce the next project which is *DistanceMatrix.* This is a java file where we create our public class with the same name.



```java
package distancematrix;
import java.util.LinkedList;

public class DistanceMatrix implements Matrix{
    // From the given instructions about how to treat lists
    private LinkedList<GeometricPoint> cities;
    private  LinkedList<LinkedList<Double>> matrix;
```

Therefore we will import this java util library at the very beginning and through this concept we create the attributes cities that will be GeometricPoints and our matrix where we will store the distances a LinkedList of LinkedList that stores doubles(distances).

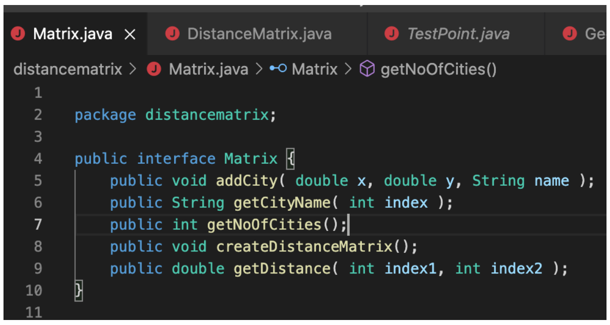


```java
// Create the class constructor
public DistanceMatrix(){
    this.cities = new LinkedList<GeometricPoint>();
    this.matrix = new LinkedList<LinkedList<Double>>();
}
```

As in any class we will have to create the constructor having nothing inside the parenthesis similar to an init.

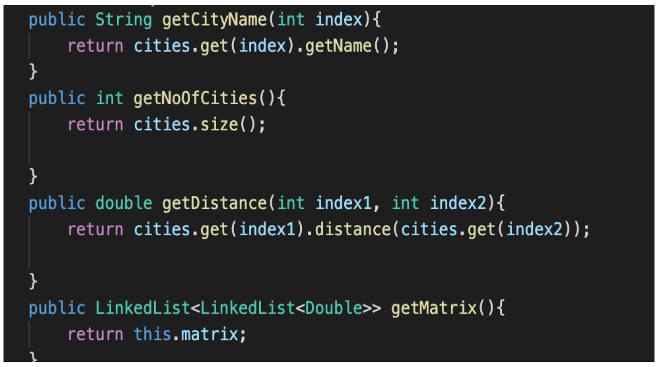
This is the code that we have already forwarded in the file Matrix and we have defined. Nevertheless, there are not implemented methods in the file matrix . We implement the methods:

(Notice there are no setters)



```java

package distancematrix;

public interface Matrix {
    public void addCity( double x, double y, String name );
    public String getCityName( int index );
    public int getNoOfCities();
    public void createDistanceMatrix();
    public double getDistance( int index1, int index2 );
}

```

1-. <u>**Getters**</u>: from the Matrix code we have seen we will have to implement in *DistanceMatrix*:

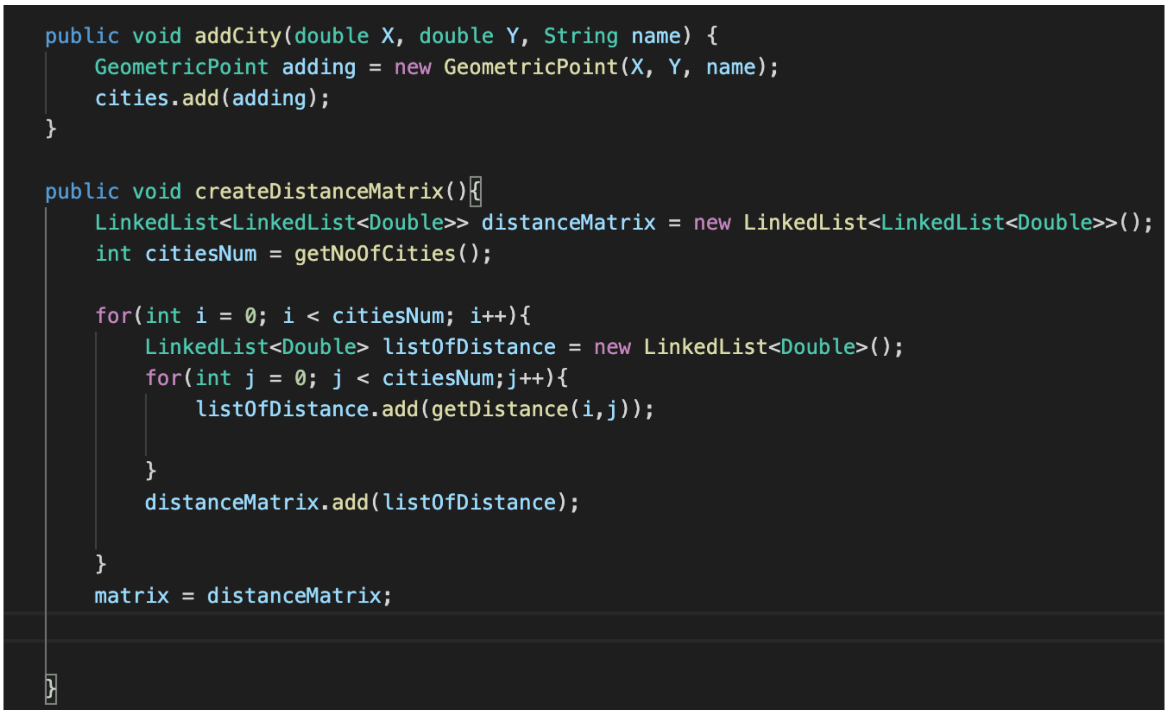


```java
public String getCityName(int index){
    return cities.get(index).getName();
}
public int getNoOfCities(){
    return cities.size();

}
public double getDistance(int index1, int index2){
    return cities.get(index1).distance(cities.get(index2));

}
public LinkedList<LinkedList<Double>> getMatrix(){
    return this.matrix;
```

- **getCityName(int index):** We return the name by accessing the index of the city and the attribute name void function we defined as a getter in GeometricPoint.
- **getNoOfCities():** We want to know how many elements we have in the list -> the cities. In conclusion, to get the size we will have to access the list and by the default java function size() we obtain the size.
- **getDistance(int index1, int index2):** we will have to return the distance that is of type double. So, as we have a matrix that stores the distance between, we access the cities indexes(i and j) and obtain the distance.
- **getMatrix():** to print the matrix returns the matrix of distances that we created, which we said is a list of lists that stores double values.

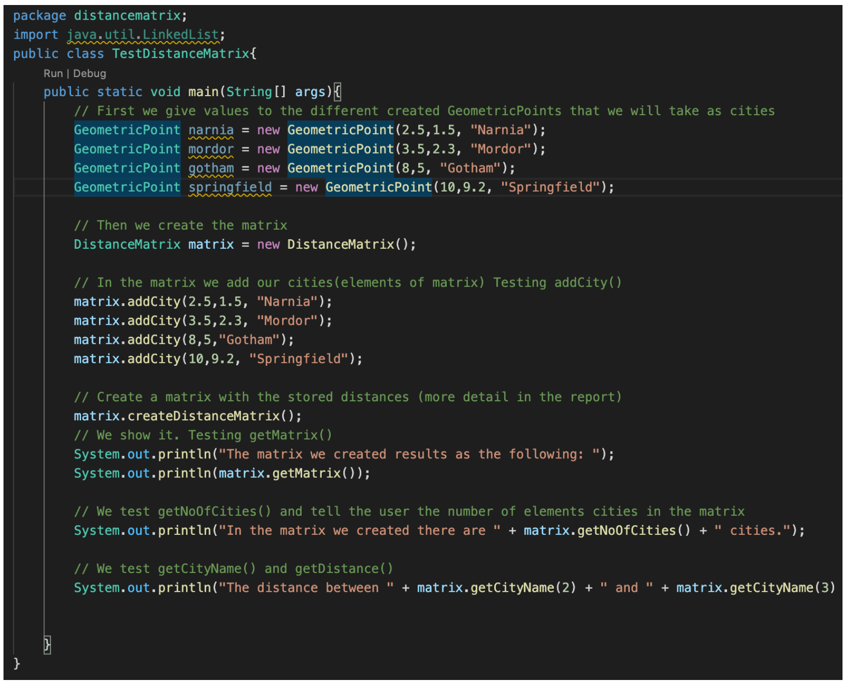
2.- **Functions:** void as they are functions.

- **createDistanceMatrix()**: To create the matrix we will have to make a new list of lists that stores doubles.
    - The size will depend on the elements we have. Ex.: n elements n*n. This will be the limit to stop creating the matrix. A matrix is composed of columns(i) and rows(j) we will be using to move through the matrix by increasing them. We are going to use two for ins by defining i and j as integer values equal to 0. In the same way we could use two whiles with a different implementation.
    - To store, we will use the function add() and to compute distances we will use the getter getDistance().
    - We will always have 4 values that are 0 as the distance between an element to the element itself will be 0.
- **addCity(**`double X, double Y, String name`**):** this function will add the elements to the list cities. Therefore we will insert new GeometricPoints with their corresponding attributes and with the default function add() we add but first it accesses the list.



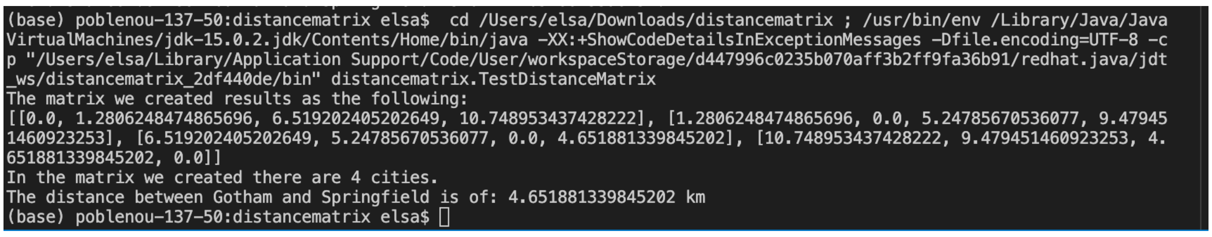
```
public void addCity(double X, double Y, String name) {
    GeometricPoint adding = new GeometricPoint(X, Y, name);
    cities.add(adding);
}

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

    for(int i = 0; i < citiesNum; i++){
        LinkedList<Double> listOfDistance = new LinkedList<Double>();
        for(int j = 0; j < citiesNum;j++){
            listOfDistance.add(getDistance(i,j));

        }
        distanceMatrix.add(listOfDistance);

    }
    matrix = distanceMatrix;


}
```

We created a new file TestDistanceMatrix to test our functions inside the DistanceMatrix file. We could do it in so many ways but we decided to do it as the following image:



```java
package distancematrix;
import java.util.LinkedList;
public class TestDistanceMatrix{
    Run | Debug
    public static void main(String[] args){
        // First we give values to the different created GeometricPoints that we will take as cities
        GeometricPoint narnia = new GeometricPoint(2.5,1.5, "Narnia");
        GeometricPoint mordor = new GeometricPoint(3.5,2.3, "Mordor");
        GeometricPoint gotham = new GeometricPoint(8,5, "Gotham");
        GeometricPoint springfield = new GeometricPoint(10,9.2, "Springfield");

        // Then we create the matrix
        DistanceMatrix matrix = new DistanceMatrix();

        // In the matrix we add our cities(elements of matrix) Testing addCity()
        matrix.addCity(2.5,1.5, "Narnia");
        matrix.addCity(3.5,2.3, "Mordor");
        matrix.addCity(8,5,"Gotham");
        matrix.addCity(10,9.2, "Springfield");

        // Create a matrix with the stored distances (more detail in the report)
        matrix.createDistanceMatrix();
        // We show it. Testing getMatrix()
        System.out.println("The matrix we created results as the following: ");
        System.out.println(matrix.getMatrix());

        // We test getNoOfCities() and tell the user the number of elements cities in the matrix
        System.out.println("In the matrix we created there are " + matrix.getNoOfCities() + " cities.");

        // We test getCityName() and getDistance()
        System.out.println("The distance between " + matrix.getCityName(2) + " and " + matrix.getCityName(3)


    }
}
```

OUTPUT

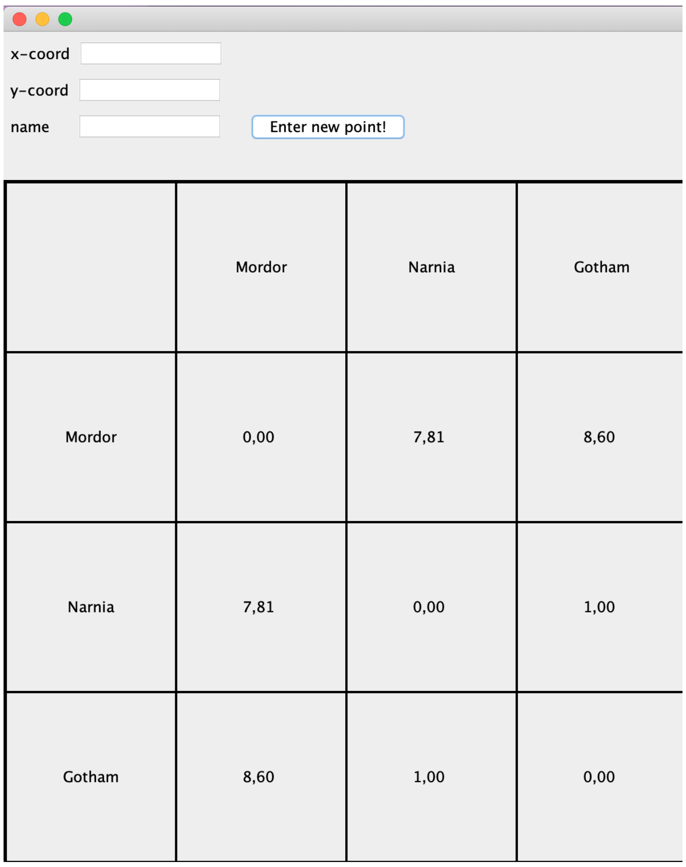


```
(base) poblenou-137-50:distancematrix elsa$  cd /Users/elsa/Downloads/distancematrix ; /usr/bin/env /Library/Java/Java
VirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -c
p "/Users/elsa/Library/Application Support/Code/User/workspaceStorage/d447996c0235b070aff3b2ff9fa36b91/redhat.java/jdt
_ws/distancematrix_2df440de/bin" distancematrix.TestDistanceMatrix
The matrix we created results as the following:
[[0.0, 1.2806248474865696, 6.519202405202649, 10.748953437428222], [1.2806248474865696, 0.0, 5.24785670536077, 9.47945
1460923253], [6.519202405202649, 5.24785670536077, 0.0, 4.651881339845202], [10.748953437428222, 9.479451460923253, 4.
651881339845202, 0.0]]
In the matrix we created there are 4 cities.
The distance between Gotham and Springfield is of: 4.651881339845202 km
(base) poblenou-137-50:distancematrix elsa$ 
```

Finally, the third part of the task was to do a project as a bonus point which is *DisplayMatrix.*

Then, we created a new file called TestDisplayMatrix, which will be related to the new public class DisplayMatrix. Moreover, we have on the first line the package distancematrix. Then, we create the class in the same name as the name of the file and to make it possible we create the part of the matrix. In addition, the display will include the matrix of type DisplayMatrix. So they are defined inside DisplayMatrix and to start it and to interact we set it visible.

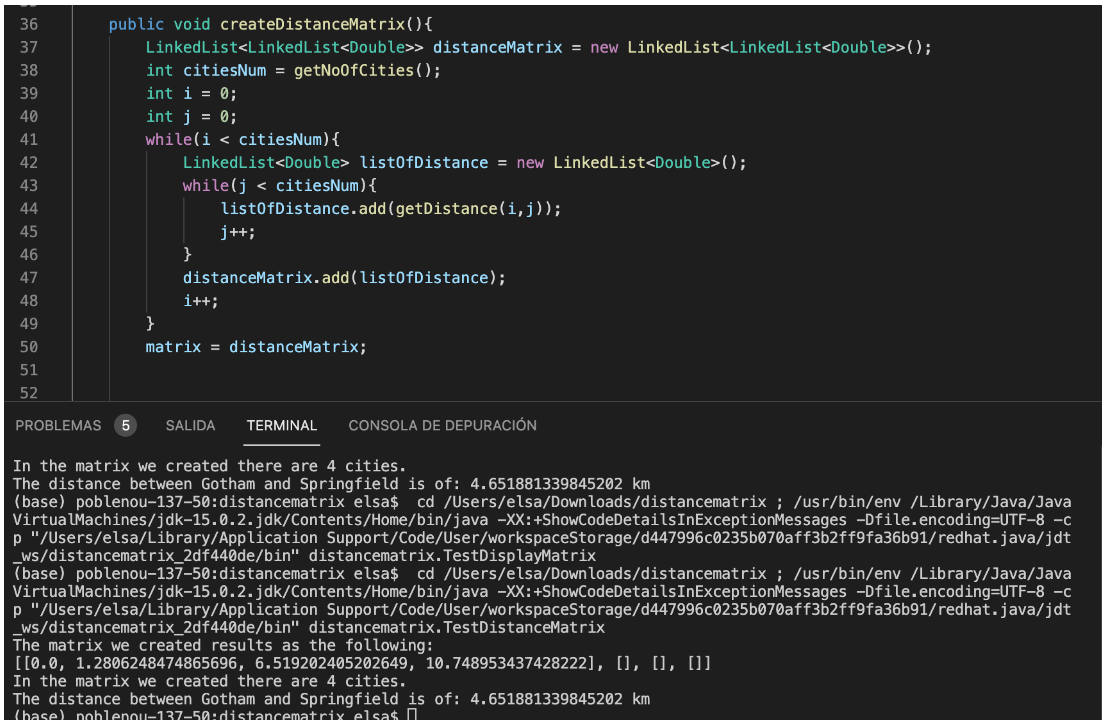
OUTPUT EXAMPLE



	Mordor	Narnia	Gotham
Mordor	0,00	7,81	8,60
Narnia	7,81	0,00	1,00
Gotham	8,60	1,00	0,00

2-. Problems

- We couldn't run anything in the very beginning as we had a problem with the way it was distributed. We had to add everything inside a folder named distance matrix. We did it a couple of times. Nevertheless it did not work correctly and it could not run. Eventually with the help of the bulbs and doing what the bulbs said, we were able to solve the problem and have everything inside the correct package.
- We left out a method that was defined inside the file Matrix. Nevertheless we clicked in the bulb again and did an override. We could not see any error or a clue about where it was. Eventually we noticed what the error was and implemented the function correctly.
- Tried to do createDistanceMatrix with a while and we got the following image instead of the correct solution. We think that the problem was that we had to redefine int j = 0 in the first while loop, and we did not do it before.



```java
    public void createDistanceMatrix(){
        LinkedList<LinkedList<Double>> distanceMatrix = new LinkedList<LinkedList<Double>>();
        int citiesNum = getNoOfCities();
        int i = 0;
        int j = 0;
        while(i < citiesNum){
            LinkedList<Double> listOfDistance = new LinkedList<Double>();
            while(j < citiesNum){
                listOfDistance.add(getDistance(i,j));
                j++;
            }
            distanceMatrix.add(listOfDistance);
            i++;
        }
        matrix = distanceMatrix;

```

```
In the matrix we created there are 4 cities.
The distance between Gotham and Springfield is of: 4.651881339845202 km
(base) poblenou-137-50:distancematrix elsa$  cd /Users/elsa/Downloads/distancematrix ; /usr/bin/env /Library/Java/Java
VirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -c
p "/Users/elsa/Library/Application Support/Code/User/workspaceStorage/d447996c0235b070aff3b2ff9fa36b91/redhat.java/jdt
_ws/distancematrix_2df440de/bin" distancematrix.TestDisplayMatrix
(base) poblenou-137-50:distancematrix elsa$  cd /Users/elsa/Downloads/distancematrix ; /usr/bin/env /Library/Java/Java
VirtualMachines/jdk-15.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -c
p "/Users/elsa/Library/Application Support/Code/User/workspaceStorage/d447996c0235b070aff3b2ff9fa36b91/redhat.java/jdt
_ws/distancematrix_2df440de/bin" distancematrix.TestDistanceMatrix
The matrix we created results as the following:
[[0.0, 1.2806248474865696, 6.519202405202649, 10.748953437428222], [], [], []]
In the matrix we created there are 4 cities.
The distance between Gotham and Springfield is of: 4.651881339845202 km
(base) poblenou-137-50:distancematrix elsa$ 
```

3-. Conclusion

For both of us this has been our first java well-structured project and we have found it really interesting. Definitely, we have found several errors and in many moments we had no idea as to which was the next step to follow, but after working on the code we could manage to overcome all of the problems. We did not know anything from programming in java, but after this lab we have learnt a lot. Especially about linked lists and pointers. Despite everything, we managed to learn and we have no doubts that we will continue to overcome all of the different problems that could happen and we will learn new aspects of programming. As a conclusion, we did a good job and learned so much. We hope to continue improving every day.