Rubén Vera
Eneko Treviño
25/10/2021

Programación Orientada a Objetos

Lab 1 – Documentación

1. Introducción

El principal objetivo de esta práctica era plasmar lo trabajado en el seminario 1 al "visual code" en forma de código, donde el usuario introduce unas coordenadas, una distancia y un nombre de un punto, en este caso será de ciudad. Después del seminario 1, las ideas principales de Geometric Point y de Distance Matrix las teníamos diseñadas en una hoja, por lo que lo único que teníamos que hacer era traducirlo a código. Los nombres de los correspondientes atributos y métodos ya los teníamos desde el comienzo de la práctica gracias al ya mencionado trabajo del Seminario. Era nuestro turno para entender que debería hacer cada método y programarlo de la mejor manera posible. En el report, explicamos el código de tres clases, los métodos que hemos utilizado y los problemas que hemos tenido. Al final hemos explicado, una breve conclusión describiendo los retos que hemos tenido que superar en el proceso y nuestra impresión sobre la práctica.

2. Explicando el código

a. Clase GeometricPoint

```java
public class GeometricPoint {
    private double x;
    private double y;
    private String name;
    public GeometricPoint(double x, double y, String name){
        this.x = x;
        this.y = y;
        this.name = name;
    }
    public void setX(double x) {
        this.x = x;
    }
    public void setY(double y) {
        this.y = y;
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
    public String getName() {
        return name;
    }
    public double distance(GeometricPoint p){
        //Formula de la distancia
        double cateto_1 = Math.pow(p.x-x, 2);
        double cateto_2 = Math.pow(p.y-y, 2);
        double resultado = Math.sqrt(cateto_1+cateto_2);
        return resultado;
    }
}
```

Imagen 2.0

Hemos establecido tres atributos a la clase GeometricPoint. La primera de todas, es un string llamado **name.** Esta contiene el nombre de los puntos, ciudades en este caso, que vamos a ejecutar. El segundo y el tercero son números reales llamado **x** e **y** e indica las coordenadas. Y como hemos visto en clase, los atributos han de ser privados.

Además, también tenemos nuestros setters **setX y setY**, que establecen el valor a nuestros atributos. Sin embargo, también utilizamos nuestros getters **getX, getY y getName** para devolver el valor de los atributos.

El método **distance** devuelve un número real que es la distancia → $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ que hay entre los puntos introducidos por el usuario.

b. **Clase DistanceMatrix**

```
lab1 > DistanceMatrix.java > ...
1   import java.util.LinkedList;
2   public class DistanceMatrix implements Matrix {
3       LinkedList <GeometricPoint>cities;//Lista de GeometricPoints
4       LinkedList <LinkedList<Double>> matrix;
5       public DistanceMatrix(){
6           this.cities = new LinkedList<GeometricPoint>();
7           this.matrix = new LinkedList<LinkedList<Double>>();
8       }
9       public double getDistance(int index1, int index2){
10          return cities.get(index1).distance(cities.get(index2));
11      }
12      private void AddtoArray(GeometricPoint p){
13          cities.add(p);
14      }
15      public void createDistanceMatrix(){//El ultimo con el resto,,, esto en principio bien
16          matrix.add(new LinkedList<Double>());//Nueva fila en la matriz
17          for(int i = 0; i < cities.size(); i++){
18              double distance = cities.getLast().distance(cities.get(i));
19              matrix.getLast().add(distance);
20              if(i != cities.size()-1){
21                  matrix.get(i).add(distance);
22              }
23          }
24      }
25      public void addCity(double x, double y, String name){
26          GeometricPoint p = new GeometricPoint(x, y, name);
27          AddtoArray(p);
28      }
29      public int getNoOfCities(){
30          return cities.size();
31      }
32      public String getCityName(int index){
33          return cities.get(index).getName();
34      }
35  }
```

Imagen 2.1

La primera parte del programa nos ha resultado bastante sencilla. Hay dos atributos, que en este caso son dos **Linked List**. Una es una lista de GeometricPoints y la otra es una Linked List dentro de una Linked List para definir la matriz. Es decir, en el constructor inicilizamos las linked list que están vacías.

De la misma forma, también tenemos nuestros getters **getDistance, getNoOfCities, getCityName** para retornar el valor de los atributos.

Los métodos **AddToArray y addCity** son muy simples. Definimos una instrucción con el código y el parámetro introducido y lo añadimos a la lista. En el caso de **addCity**, creamos un nuevo GeometricPoint y pasamos la información a la función **AddToArray.**

En el caso del método **createDistanceMatrix** hemos tenido algunas complicaciones. Primeramente, pensábamos que a la hora de calcular la distancia había que hacerlo la primera ciudad con el resto, pero nos dimos cuenta de que era más sencillo hacerlo de la forma inversa ya que cuando se añade un punto, este se coloca al final de la lista. Además, también tuvimos problemas a la hora de establecer la última distancia en la matriz, ya que la establecíamos bien, pero como el código no estaba del todo bien, establecía el mismo valor en la siguiente columna. Sin embargo, pudimos solucionarlo con las líneas de código 20 y 21 que se ven en la imagen 2.1.

En efecto, el método añade una fila en la matriz, y entra en un for loop, donde va calculando las distancias entre las ciudades introducidas por el usuario hasta que la i sea mayor o igual que el tamaño de la lista de ciudades. Y si la i es igual al tamaño de la lista de ciudades establece la distancia que le pertenece en su respectiva posición.

```
lab1 > Matrix.java > ...
1
2   //package distancematrix;
3
4   public interface Matrix {
5       public void addCity( double x, double y, String name );
6       public String getCityName( int index );
7       public int getNoOfCities();
8       public void createDistanceMatrix();
9       public double getDistance( int index1, int index2 );
10  }
11
```

Imagen 2.2

3. Conclusión

Desde el primer momento que empezamos a trabajar en este proyecto sabíamos que la clave para poder hacerlo lo mejor posible era comprobar el código creado ya que nos facilitaría mucho el trabajo, ya que, el código sería más funcional, limpio y entendible. Utilizando el método de prueba error hemos modelado nuestras funciones, hemos corregido varias funciones ya que principalmente lo pensamos de una manera, pero a medida que hemos ido avanzando con el trabajo hemos cambiado aquellas funciones que veíamos que podíamos hacerlas más simples y visuales.

Hemos creado varios programas para testear nuestras clases, como TestDistanceMatrix, TestPoint, testDisplayMatrix, que contienen un main, donde hemos ejecutado varias muestras para asegurarnos que nuestros métodos estaban bien.

Para nosotros, el método más difícil de implementar nos ha resultado el método createDistanceMatrix, como ya hemos explicado antes. Sin embargo, hemos sabido solventar esas dudas, problemas con eficacia y trabajo en equipo.

Esta práctica ha sido un gran reto para nosotros, ya que nos costó un poco entender lo que nos pedían en la práctica. Después de eso, podemos decir que hemos producido un código de calidad, producto de nuestras horas de trabajo de calidad, trabajo en equipo y el método prueba error.