Rubén Vera
Eneko Treviño
23/11/2021

PROGRAMACIÓN ORIENTADA A OBJETOS

PRÁCTICA 3 – DOCUMENTACIÓN

1. Introducción

El objetivo de esta tercera práctica era implementar el programa diseñado en los seminarios dos y tres, es decir, extender el diseño de country, continents y world del seminario 2. En esta tarea, hemos estructurado nuestras clases, de manera que estén relacionados. Eventualmente, nuestro deber era principalmente relacionar las nuevas clases con las del seminario dos y decidir cómo cada función debería funcionar para así poder relacionarlas correctamente entre las diferentes partes del programa y crear una experiencia simple y funcional para el usuario. En este report, hemos implementado nuevas clases como **City, Country, GeoPoint** y **EllipsoidalRegion.** Además, hemos hecho cambios en las siguientes **Continent** y **MyMap**. Sin embargo, las demás clases las hemos mantenido igual que en la práctica dos. Aparte, hemos explicado brevemente nuestras adversidades y retos que nos ha supuesto este trabajo. Hemos puesto punto y final con una corta pero clara conclusión, donde explicamos cómo hemos distribuido el trabajo y nuestros sentimientos sobre el trabajo.

Rubén Vera
Eneko Treviño
23/11/2021

2. Explicando el código

- **Clase City**

```java
import java.awt.Graphics;
public class City extends GeoPoint{
    private int numhab;//que hacer con esto
    public City(double xi, double yi, int h, String n) {
        super(xi, yi, n);
        this.numhab = h;
    }
    public void draw(Graphics g){
        super.draw(g);
    }

}
```

Nos encontramos con una clase llamada **City**, donde esta clase hereda todos los métodos de **GeoPoint** a través de **extends.** Nos encontramos con un atributo, por teoría sabemos que tiene que ser privado, que es un entero. En esta práctica no lo hemos tenido que utilizar. En el constructor, inicializamos el número de habitantes, y con la función super traemos la información proporcionada por GeoPoint, un punto x, un punto y un nombre, que será el nombre del país.

- **Clase Country**

```java
import java.util.LinkedList;
import java.awt.Graphics;

public class Country extends PolygonalRegion{
    private String name;
    private LinkedList< City > cities;
    private LinkedList< Country > neighbors;
    private City capital;
    public Country(LinkedList<Point> l, City capital) {
        super(l);
        this.capital = capital;
        cities = new LinkedList<City>();
        neighbors = new LinkedList<Country>();
    }
    public void addCity(City c){
        this.cities.add(c);
    }
    public void addNeighbor(Country c){
        neighbors.add(c);
    }
    public void draw(Graphics g){
        super.draw(g);
        for (int i = 0; i < cities.size(); i++){
            cities.get(i).draw(g);
        }
    }

}
```

Hemos establecido los atributos a la clase Country en el cual podemos encontrar 4**: name (string), capital (City), Linked List de City y Linked List de Country**. En esta clase también extendemos, pero en este caso hereda los métodos de **PolygonalRegion.** En el constructor inicializamos la Linked List de Country. Dentro del método de Country también inicializamos la Linked List de City. Aparte de esto tenemos tres métodos, **addCity**, añade ciudades a la Linked List City**,addNeighbor,**añade países vecinos a la Linked List Country**, draw**, unimos cada ciudad con en el siguiente creado mediante un for loop.

Rubén Vera
Eneko Treviño
23/11/2021

- **Clase GeoPoint**

```java
import java.awt.Graphics;
public class GeoPoint extends Point{
    private String name;
    public GeoPoint(double xi, double yi, String n) {
        super(xi, yi);
        this.name = n;
    }
    public void draw(Graphics g){
        int x = (int) this.getX();
        int y = (int) this.getY();
        g.drawString(name, x, y);
        g.fillOval( x+5, y, 10, 10);

    }
    public String getName() {
        return name;
    }
}
```

Nos encontramos con una clase llamada **GeoPoint**, donde esta clase hereda todos los métodos de **Point** a través de **extends.** Tenemos un único atributo que es una string llamada **name.** En el constructor inicializamos los valores **x** e **y**. Y también el nombre. En esta clase tenemos dos métodos; un getter, **getName**, para devolver el valor, es decir, el nombre y un método draw, donde hay dos getters más que son **getX()** y **getY()** que devuelven las coordenadas introducidas. Además, utilizamos las funciones **drawString**, para dibujar el nombre y **fillOval,** para dibujar un círculo pequeño.

- **Clase EllipsoidalRegion**

```java
import java.awt.Graphics;
public class EllipsoidalRegion extends Region{
    private Point c;//Centro
    private double r1;
    private double r2;
    public EllipsoidalRegion(Point c, double r1, double r2) {
        this.c = c;
        this.r1 = r1;
        this.r2 = r2;
    }
    public double getArea(){
        return Math.PI*r1*r2;
    }
    @Override
    public void draw(Graphics g){
        g.fillOval((int)c.getX(),(int) c.getY(),(int) r1,(int) r2);
    }
}
```

Tenemos una clase llamada **EllipsoidalRegion** que a través de la palabra extends hereda los métodos de **Region.** Está compuesta por tres atributos que son **r1**, **r2**, **c.**
R1 y **r2** son dos puntos y la **c** es el centro. En el constructor inicializamos los valores **c**, **r1**, **r2.** Utilizamos un getter, **getArea** para devolver el valor del área. Para finalizar sobre escribimos el método **draw**.

Rubén Vera
Eneko Treviño
23/11/2021

- **Clase Continent**

Código lab2

```java
import java.util.LinkedList;
import java.awt.Graphics;

public class Continent {
    //Atributos
    private LinkedList< PolygonalRegion > regions;
    //Ctor
    public Continent(LinkedList< PolygonalRegion > c) {
        this.regions = c;
    }
    //Metodos
    public double getTotalArea(){
        double total_area = 0;
        //Para cada region calculamos su area usando getArea
        for (int i = 0; i < regions.size(); i++){
            total_area += regions.get(i).getArea();
        }
        return total_area;
    }
    public void draw(Graphics g){
        //Dibujamos cada region usando el draw de la clase P
        for (int i = 0; i < regions.size(); i++){
            regions.get(i).draw(g);
        }
    }
}
```

Código lab3

```java
import java.util.LinkedList;
import java.awt.Graphics;

public class Continent {
    //Atributos
    private LinkedList< Country > countries;
    //Ctor
    public Continent(LinkedList< Country > c) {
        this.countries = c;
    }
    //Metodos
    public double getTotalArea(){
        double total_area = 0;
        //Para cada region calculamos su area usando getArea de la clase PolygonalRegion y las vamos sumando
        for (int i = 0; i < countries.size(); i++){
            total_area += countries.get(i).getArea();
        }
        return total_area;
    }
    public void draw(Graphics g){
        //Dibujamos cada region usando el draw de la clase PolygonalRegion
        for (int i = 0; i < countries.size(); i++){
            countries.get(i).draw(g);
        }
    }
}
```

En esta clase, como se puede observar en las imágenes de arriba, hemos hecho unos cambios pequeños para poder enlazar todas las clases. Hemos establecido los atributos a la clase Continent. En esta clase solo podemos encontrar uno, que es una Linked List de **Country**. En el constructor inicializamos la Linked List, que empieza vacia. Además también tenemos métodos, **getTotalArea(), draw(Graphics g).** En el método getTotalArea(), calculamos para cada país su área mediante un for loop. Sin embargo, en el método draw(Graphics g) dibujamos cada país usando el draw de la clase countries .

- **Clase MyMap**

```java
import java.util.LinkedList;

public class MyMap extends javax.swing.JPanel {
    private World world;
    public MyMap() {
        initComponents();
        //Region 1
        LinkedList< Point > points1 = new LinkedList< Point >();//Creamos una lista de puntos
        points1.add(new Point(10, 100));
        points1.add(new Point(150, 10));
        points1.add(new Point(300, 100));
        points1.add(new Point(300, 200));
        points1.add(new Point(150, 300));
        points1.add(new Point(10, 200));



        City bcn = new City(290, 100, 100, "BCN");
        City mad = new City(150, 150, 10, "MAD");
        City bil = new City(290, 190, 100, "BIL");

        Country spain = new Country(points1, mad);
        spain.addCity(bcn);
        spain.addCity(mad);
        spain.addCity(bil);
        //Country spain2 = new Country(points1, bcn);
        //Country spain3 = new Country(points1, bil);

        LinkedList< Country > countries = new LinkedList< Country >();
        countries.add(spain);
        //countries.add(spain2);
        //countries.add(spain3);

        LinkedList< Continent > conts = new LinkedList< Continent >();
        Continent europe = new Continent(countries);
        conts.add(europe);
```

```java
        //World
        world = new World(conts);//Creamos el mundo compuesto por todos los continentes
    }

    private void initComponents() {
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 1000, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 1000, Short.MAX_VALUE)
        );
    }

    public void paint( java.awt.Graphics g ) {
        super.paint( g );
        world.draw( g );
        }
}
```

La clase MyMap(), es una clase que fue administrada en la práctica 2, ya que proporcionan el esqueleto de la interfaz gráfica de usuario. Sin embargo, hemos tenido que modificar el método ya que ahora en vez de añadir puntos de PolligonalRegion añadimos ciudades, continentes y creamos un mundo compuesto por todos los continentes.

Rubén Vera
Eneko Treviño
23/11/2021

3. Conclusión

La tercera práctica para nosotros no ah sido la más complicada, pero en el seminario a la hora de plantear alguna que otra relación se nos ha hecho difícil. El principal reto era decidir las relaciones entre las clases, y el uso de cada método con el fin de alcanzar la implementación más eficiente. Entre todos los métodos, ninguno nos ha resultado especialmente difícil de implementar, sin embargo, vale la pena destacar que el método *draw*, ha sido el método que más rompecabezas nos ha surgido, ya que la teníamos que implementar de distintas maneras en cada clase. El método MyMap() no era muy difícil, pero nos ha llevado bastante tiempo debido a que no algún problema que otro.

Para verificar que nuestro código opera correctamente hemos utilizado la simulación de MyWindow() donde está el main. Por eso es que en MyMAp hemos tenido que hacer cambios para que a la hora de ejecutarse los nombres dados fueran de ciudades. Más tarde, era hora de comprobar cada método tenía el output correcto. Evidentemente, las primeras veces nos resulto difícil identificar todos los errores debido a que había muchos. Sin embargo, gracias al debug y nuestro trabajo en equipo supimos corregir todos los errores.

Como equipo, hemos trabajado una vez más juntos en el código y nos hemos ayudado el uno al otro dándonos una nueva perspectiva cuando nos quedábamos atascados. Nos ha requerido bastante paciencia y constancia, pero al final conseguimos llevarlo acabo con un código claro y limpio. Además, esperamos que nuestras líneas reflejen todas las horas de discusión y trabajo que hemos dedicado a este proyecto.