Rubén Vera
Eneko Treviño
09/11/2021

PROGRAMACIÓN ORIENTADA A OBJETOS

LAB 2 – DOCUMENTACIÓN

1.INTRODUCCIÓN

El objetivo de la segunda práctica era implementar lo planteado en el seminario 2. Es decir, aprender a diseñar un programa entero, en el cual clases múltiples interactúan para solucionar un problema. La asignación de esta práctica era programar las clases **World, PoligonalRegion, Point, MyWindow, MyMap y Continent.** Como estudiantes hemos tenido que trabajar de manera equitativa cada clase, cada una de ellas con sus problemas y dificultades. En resumen, nuestro trabajo ha sido decidir, debatir como cada clase debería funcionar y llevarlo a cabo de forma correcta en forma de código. En este report, explicamos brevemente la implementación de las clases nombradas anteriormente y los retos que hemos tenido que superar para poder hacer la práctica. Acabamos con una pequeña conclusión que detalla como nos hemos organizado el trabajo de la práctica y nuestra opinión sobre el trabajo que hemos hecho.

2. EXPLICACIÓN DEL CÓDIGO

- Clase Continent

```
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);
        }
    }
}
```

Hemos establecido los atributos a la clase Continent. En esta clase solo podemos encontrar uno, que es una Linked List de PolygonalRegions. 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 región su área mediante un for loop. Sin embargo, en el método draw(Graphics g) dibujamos cada región usando el draw de la clase PolygonalRegion.

- Clase MyMap

```
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
        LinkedList< Point > points2 = new LinkedList< Point >();//Creamos una lista de puntos
        //Añadimos 3 puntos a la lista de puntos1
        points1.add( new Point( 10, 100 ) );
        points1.add( new Point( 150, 10 ) );
        points1.add( new Point( 290, 100 ) );
        //Añadimos 4 puntos a la lista de puntos2
        points2.add( new Point( 290, 200 ) );
        points2.add( new Point( 150, 290 ) );
        points2.add( new Point( 25, 145 ) );
        points2.add( new Point( 10, 200 ) );
        LinkedList< PolygonalRegion > countries1 = new LinkedList< PolygonalRegion >();//Creamos la primera region
        //Añadimos las dos listas de puntos a la primera region
        countries1.add(new PolygonalRegion(points1));
        countries1.add(new PolygonalRegion(points2));


        //Region2
        LinkedList< Point > points3 = new LinkedList< Point >();//Creamos una lista de puntos
        //Añadimos 5 puntos a la lista de puntos 3
        points3.add( new Point( 160, 40 ) );
        points3.add( new Point( 50, 700 ) );
        points3.add( new Point( 59, 450 ) );
        points3.add( new Point( 460, 240 ) );
        points3.add( new Point( 360, 170 ) );
        LinkedList< PolygonalRegion > countries2 = new LinkedList< PolygonalRegion >();//Creamos la segunda region
        //Añadimos los puntos a la segunda region
        countries2.add(new PolygonalRegion(points3));
```

La clase MyMap(), es una clase que ha sido administrada, ya que proporcionan el esqueleto de la interfaz gráfica de usuario. Sin embargo, hemos tenido que modificar un poco el método para poder comprobar que todas las funciones hiciesen lo esperado. Hemos creado diferentes continentes y lo que ello comporta para poder ejecutar el draw del world.

Rubén Vera
Eneko Treviño
09/11/2021

```
//Region3
LinkedList< Point > points4 = new LinkedList< Point >();//Creamos una lista de puntos
//Añadimos 3 puntos a la lista de puntos 4
points4.add( new Point( 400, 150 ) );
points4.add( new Point( 600, 700 ) );
points4.add( new Point( 20, 30 ) );
LinkedList< PolygonalRegion > countries3 = new LinkedList< PolygonalRegion >();//Creamos la tercera region
//Añadimos los puntos a la tercera region
countries3.add(new PolygonalRegion(points4));

//Continents
LinkedList< Continent > conts = new LinkedList< Continent >();//Creamos la lista de continentes
//Continent1
conts.add(new Continent(countries1));//Añadimos la primera region a los continentes

//Continent2
conts.add(new Continent(countries2));//Añadimos la segunda region a los continentes

//Continent3
conts.add(new Continent(countries3));//Añadimos la tercera region a los continentes


//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 );
    }
```

- Clase MyWindow()

```
public class MyWindow extends javax.swing.JFrame {

    public MyWindow() {
        initComponents();
    }

    private void initComponents() {
        setDefaultCloseOperation( javax.swing.WindowConstant
    }

    Run | Debug
    public static void main( String[] args ) {
        java.awt.EventQueue.invokeLater( new Runnable() {
            public void run() {
                MyWindow w = new MyWindow();
                MyMap m = new MyMap();
                w.add( m );
                w.setVisible( true );
                w.pack();
            }
        } );
    }

}
```

La clase MyWindow(), es una clase que también ha sido administrada, ya que proporcionan el esqueleto de la interfaz gráfica de usuario. Al ejecutar el main que está implícito en esta clase, obtenemos visualizar una ventana con el Word con sus pertinentes regiones dibujadas dentro de ella.

- Clase Point

```
public class Point {
    //Atributos
    private double x;
    private double y;
    //Ctor
    public Point(double xi, double yi) {
        this.x = xi;
        this.y = yi;
    }
    //Metodos, getters en este caso
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
}
```

Hemos establecido dos atributos double a la clase Point, que son **x** e **y**, que son las coordenadas del punto. Como hemos visto en las clases de teoría son privados estos atributos. En el constructor inicializamos los valores de **x** e **y**. Utilizamos getters, **getX()** y **getY()** en el apartado de los métodos, para devolver el valor de los atributos y poder acceder a ellos desde fuera de la clase.

Rubén Vera
Eneko Treviño
09/11/2021

- Clase PolygonalRegion

```java
public class PolygonalRegion {
    //Atributos
    private LinkedList< Point > points;
    //Ctor
    public PolygonalRegion(LinkedList< Point > l) {
        this.points = l;
    }
    //Metodos
    //Calculamos el area de la region usando la formula pertinente.
    public double getArea(){
        double det1 = 0;
        double det2 = 0;
        for(int i = 0; i < points.size()-1; i++){
            det1 = det1 + (points.get(i).getX()*points.get(i+1).getY());
            det2 = det2 + (points.get(i).getY()*points.get(i+1).getX());
        }
        det1 = det1 + (points.getLast().getX()*points.getFirst().getY()) ;
        det2 = det2 + (points.getLast().getY()*points.getFirst().getX());
        double det = det1-det2;
        double area = 0.5 * det;
        return area;
    }
```

Nos encontramos con un atributo que es una Linked List de Point. En el constructor, como lo hemos hecho anteriormente en la clase **Continent** inicializamos la Linked List que estará vacía. En el apartado de los métodos tenemos un getter **getArea()**, para calcular el área de la región usando la formula pertinente, y devolvemos el valor de los atributos en forma de área. Por otro lado, en el método draw(Graphics g) unimos cada punto con el siguiente creado mediante un for loop, y creamos la rama que conecta el último punto con el primero.

```java
    public void draw(Graphics g){
        //Cada punto se unira con el siguiente creado
        for(int i = 0; i < points.size()-1; i++){
            g.drawLine( (int) points.get(i).getX(), (int)points.get(i).getY(), (int) points.get(i+1).getX(), (int) points.get(i+1).getY());
        }
        //Creamos el enlace que conecte el ultimo punto con el primero
        g.drawLine((int) points.getLast().getX(), (int) points.getLast().getY(), (int) points.getFirst().getX(), (int) points.getFirst().getY());

    }
}
```

- Clase World

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

public class World {
    //Atributos
    private LinkedList< Continent > conts;
    //Ctor
    public World(LinkedList< Continent > c) {
        this.conts = c;
    }
    //Metodos
    public void draw(Graphics g){
        //Dibujamos cada continente usando la funcion draw d
        for (int i = 0; i < conts.size(); i++){
            conts.get(i).draw(g);
        }
    }

}
```

Tenemos un atributo que es una Linked List de Continents. En el constructor hemos inicializado la lista que esta vacia. En los métodos volvemos a utilizar el método **Draw(Graphics g)**, donde dibujamos cada continente usando la función de la clase Continent.

3. CONCLUSIÓN

Desde que tuvimos esta tarea, comenzamos a trabajar con un orden establecido previamente, para así poder crear e implementar código claro y limpio. Como siempre, el método de prueba error ha sido la columna vertebral de nuestro equipo en la dinámica de desarrollo.

La creación e implementación de la clase PolygonalRegion ha sido la clave para poder comprobar si nuestro programa iba a estar bien estructurado. Tuvimos problemas en implementar la función de draw en la clase PolygonalRegion, ya que primero entendimos que cada punto debía ser unido con todos los siguientes. Pero después de consultarlo en clase con el profesor de prácticas llegamos a la conclusión de que debíamos unirlo con el siguiente creado y no con todos.

Sin embargo, el hecho de haber tenido exámenes parciales ha dificultado el trabajo continuo de la práctica, debido a que no hemos podido reunirnos muchas veces. Pero por otro lado, nuestra buena gestión de los tiempos y la buena organización nos ha llevado a producir un trabajo de calidad gracias a que los dos integrantes del equipo hemos puesto de nuestra parte.