Lab exercise 3: Extending the World design
24292-Object Oriented Programming

Jordi Polo Tormo : 239980
Simón Gasión Medina: 240126

1.Introduction

En esta práctica debemos de extender el diseño de la práctica anterior, implementando la clase “Country” la cual viene heredada de “PolygonalRegion”, siendo “Country” más completa al contener una lista de ciudades, una capital y una lista de vecinos.

Para las ciudades de cada país, creamos una clase llamada “City” la cual coge como atributo únicamente el número de habitantes de la ciudad ya que el resto viene heredado de la clase “GeoPoint” creada también para esta práctica, teniendo esta como atributo un nombre para el punto y heredando los valores de las coordenadas de la clase “Point” de la anterior práctica.

Las ciudades recibirán las coordenadas de “Point”, el nombre de “GeoPoint” y el número de habitantes de la propia clase. Siendo esto para el método constructor, construyendo de esta manera una ciudad.

Para la parte optativa creamos las dos clases “Lake” y “Ocean”, siento estas heredadas de “PolygonalRegion”, recibiendo en el constructor una lista de “Point” e implementando también una función “draw”, siendo los lagos de azul cian y los océanos de azul oscuro.

2.Description

Country:
Para implementar “Country” debemos definir como atributos la capital de tipo “City”, una lista de países vecinos, una lista de ciudades y una lista de lagos para la parte optativa de la práctica. La lista de fronteras no se usa en ningún método, pero al tomar como referencia el trabajo realizado en las clases de seminario hemos considerado añadirlo por sí es útil en alguno de los siguientes labs; tampoco és la única instancia de esto, podemos ver en esta misma clase que no se usa el atributo “Capital”, de lo que avisa el mismo IDE. Posteriormente implementamos los métodos, los cuales son el constructor que coje una lista de puntos, una ciudad para darle el papel de capital, implementando esta en una lista de ciudades ya que cada país debe tener como mínimo una capital, y una lista de lagos para la parte optativa. Seguidamente implementamos la función “addCity” la cual añade una ciudad a la lista de ciudades pasandole una ciudad, un “addNeighbor” el cual añadirá un país a la lista de países vecinos, y finalmente la función draw la qual dibujara tanto los paises como los lagos, gracias a la función draw implementada en la clase “PolygonalRegion”. Cambiando posteriormente en la clase “Continent” la lista de “PolygonalRegion” por una de tipo “Country”.

```java
import java.util.LinkedList;

public class Country extends PolygonalRegion {
    //Atributos//
    private City capital;
    private LinkedList<Country> neighbors;
    private LinkedList<City> cities;
    private LinkedList<Lake> lakes;
    //Métodos//
    public Country(LinkedList<Point> l, City cap, LinkedList<Lake> w) {
        super(l);
        this.capital = cap;
        LinkedList<City> c = new LinkedList<City>();
        c.add(cap);
        this.cities = c; //A country will always contain at least one city (capital)
        this.lakes = w;
    }
    public void addCity(City c) {
        this.cities.addLast(c);
    }
    public void addNeighbor(Country c) {
        this.neighbors.addLast(c);
    }
    public void draw(java.awt.Graphics g) {
        super.draw(g);
        int lakesize = this.lakes.size(); //draw lakes
        for (int i = 0; i < lakesize ; i++) {
            this.lakes.get(i).draw(g);
        }
        int size = this.cities.size(); //draw cities
        for (int i = 0; i < size; i++) {
            this.cities.get(i).draw(g);
        }
    }
}
```

City:
Para la implementación de “City” definimos como atributo el número de habitantes y creamos dos métodos, el constructor el cual coge las coordenadas y el nombre gracias a GeoPoint y el número de habitantes, y el método draw, el cual utiliza la función definida en “GeoPoint” para implementarlo en “City”.

```
public class City extends GeoPoint {

    //Attributes
    private int numhab;

    //Methods
    //constructor
    public City(double xi, double yi, String n, int h){ /
        super(xi, yi, n); //constructor of GeoPoint
        this.numhab = h;
    }
    public void draw(java.awt.Graphics g) { //asumimos qu
        super.draw(g); //imprimir una City y imprimir un
    }
}
```

GeoPoint:
Como hemos mencionado anteriormente utilizamos también la clase “GeoPoint” para “City”, por tanto debimos implementar esta, la qual tiene como atributo únicamente un nombre pero viene heredado de “Point”, utilizando las coordenadas de este en el constructor y el nombre definido en la clase propia como atributo. Implementamos también el método “draw” el cual utiliza las coordenadas y la función para dibujar elipses. Nos hemos topado con un problema al intentar implementar este método debido a que el enunciado sugiere que usemos la función “fillOval” de la librería “graphics” no admite coordenades en reales, tan solo en enteros; para solucionarlo, como en labs anteriores, hemos encontrado una función similar en la librería “graphics2D” que sí podemos usar con precisión sub-pixel.

```java
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class GeoPoint extends Point {

    //Attributes
    private String name;

    //Methods
    //constructor
    public GeoPoint(double xi, double yi, String n){
        super(xi, yi); //constructor of Point
        this.name = n;
    }
    public void draw(java.awt.Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        g2.draw(new Ellipse2D.Double(this.x, this.y, 5, 5)); //we
        g2.drawString(this.name, (int) this.x + 5, (int) this.y);
    }
}
```

Lake:

Para la parte optativa de la practica tuvimos que definir las clases “Lake” y “Ocean”, para el caso de “Lake” implementamos el metodo constructor el qual coge una lista de puntos gracias a que viene heredada por “PolygonalRegion”, y el metodo “draw” el qual utiliza la misma función de “PolygonalRegion” que habiamos usado para “Country” pero en este caso de color “CYAN” es decir de azul cian, para distinguir los lagos de los océanos.

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

public class Lake extends PolygonalRegion {
    //Atributos//
    //En el seminario existia una lista de Cout
    //Métodos//
    public Lake(LinkedList<Point> l) {
        super(l);
    }
    public void draw(java.awt.Graphics g) {
        g.setColor(Color.CYAN); //color azul má
        super.draw(g);
        g.setColor(Color.BLACK);
    }
}
```

Ocean:

Para el caso de “Ocean” la implementación es prácticamente igual a la de “Lake”, cambiando únicamente el el color de este ya que “Ocean” usara “BLUE” el cual es más oscuro que el azul de los lagos, representando la profundidad del océano.

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

public class Ocean extends PolygonalRegion {
    //Atributos//
    //En el seminario existia una lista de Cout
    //Métodos//
    public Ocean(LinkedList<Point> l) {
        super(l);
    }
    public void draw(java.awt.Graphics g) {
        g.setColor(Color.BLUE); //color azul má
        super.draw(g);
        g.setColor(Color.BLACK);
    }
}
```

Finalmente tuvimos que cambiar la clase “MyMap” para dibujar las ciudades, los lagos y los océanos. Aparte de también cambiar las regiones por países de tipo “Country” en vez de “PolygonalRegion”. Para esto creamos ciudades varias pasándose coordenadas “x” e “y”, un nombre para la ciudad y un número de habitantes. Y situandolas en diferentes países, definiendo en la creación del país una de las ciudades como capital y luego utilizando la función “addCity” añadimos el resto de ciudades que queremos que pertenezcan al país.

Como parte optativa también tuvimos que crear varios lagos y situar estos en el país que queramos que les corresponda. Para crear estos definimos varios puntos y los situamos en una lista, pasando esta al lago con el método constructor, situando los lagos o el lago en una lista de tipo “Lake” y pasando ésta al constructor del país.

En el caso de “Ocean” varía en que en vez de ser pasada la lista de tipo “Ocean” a un país, esta es pasada al “World”, pero por el resto la creación de los océanos es exactamente igual a la de los lagos, definiendo una lista de puntos para el constructor del océano y colocando los océanos en una lista de tipo “Ocean”.

La creación de los países es la misma que la práctica anterior, solo que ahora se le pasa una ciudad como capital y una lista de lagos. En cuanto a los continentes tienen la misma creación que la anterior práctica, pasando una lista de países.

Ejemplo de la creación de ciudades y lagos para la creación del país:

```java
import java.util.LinkedList;

public class MyMap extends javax.swing.JPanel {

    private World world;

    public MyMap() {
        initComponents();
        //Creamos 3 Regiones con distintos puntos para generar el primer continente

        LinkedList< Point > points1 = new LinkedList< Point >();
        points1.add( new Point( 10, 100) );
        points1.add( new Point( 150, 10) );
        points1.add( new Point( 290, 100) );
        points1.add( new Point( 290, 200 ) );
        points1.add( new Point( 150, 290 ) );
        points1.add( new Point( 10, 200 ) );
        //Creamos varias ciudades para el primer país de Europa
        City Madrid = new City(100, 100, "Madrid", 3223);
        City Barcelona = new City(200, 150, "Barcelona",1622);
        City Cadiz = new City (50, 200, "Cadiz", 166);

        //Creamos dos lagos para el pais
        LinkedList< Point > Lake1= new LinkedList< Point >();
        Lake1.add(new Point(100, 50));
        Lake1.add(new Point(150, 50));
        Lake1.add(new Point(125, 75));
        Lake CasaDeCampo = new Lake(Lake1);

        LinkedList< Point > Lake2= new LinkedList< Point >();
        Lake2.add(new Point(180, 180));
        Lake2.add(new Point(120, 120));
        Lake2.add(new Point(120, 200));
        Lake Ciutadella = new Lake(Lake2);

        //Creamos la lista conde almazenar los lagos del pais
        LinkedList<Lake> LakeEsp = new LinkedList<Lake>();
        LakeEsp.add(CasaDeCampo);
        LakeEsp.add(Ciutadella);

        //Creamos el pais y añadimos las otras ciudades
        Country Spain = new Country(points1, Madrid,LakeEsp);
        Spain.addCity(Barcelona);
        Spain.addCity(Cadiz);
        System.out.println("Area España:"+Spain.getArea() );
```

Creación de los océanos para el mundo:

```java
//Creamos Oceanos para crear el mundo
LinkedList<Point> Ocean1 = new LinkedList<Point>();
Ocean1.add(new Point(850, 500));
Ocean1.add(new Point(600, 380));
Ocean1.add(new Point(400, 700));
Ocean Pacifico = new Ocean(Ocean1);

LinkedList<Point> Ocean2 = new LinkedList<Point>();
Ocean2.add(new Point(10,290));
Ocean2.add(new Point(400, 290));
Ocean2.add(new Point(400, 700));
Ocean Atlantico = new Ocean(Ocean2);

LinkedList<Point> Ocean3 = new LinkedList<Point>();
Ocean3.add(new Point(10,990));
Ocean3.add(new Point(600, 990));
Ocean3.add(new Point(400, 700));
Ocean Indico = new Ocean(Ocean3);


LinkedList<Ocean> Ocean = new LinkedList<Ocean>();
Ocean.add(Pacifico);
Ocean.add(Atlantico);
Ocean.add(Indico);

this.world = new World(world, Ocean);
```

3.Conclusión

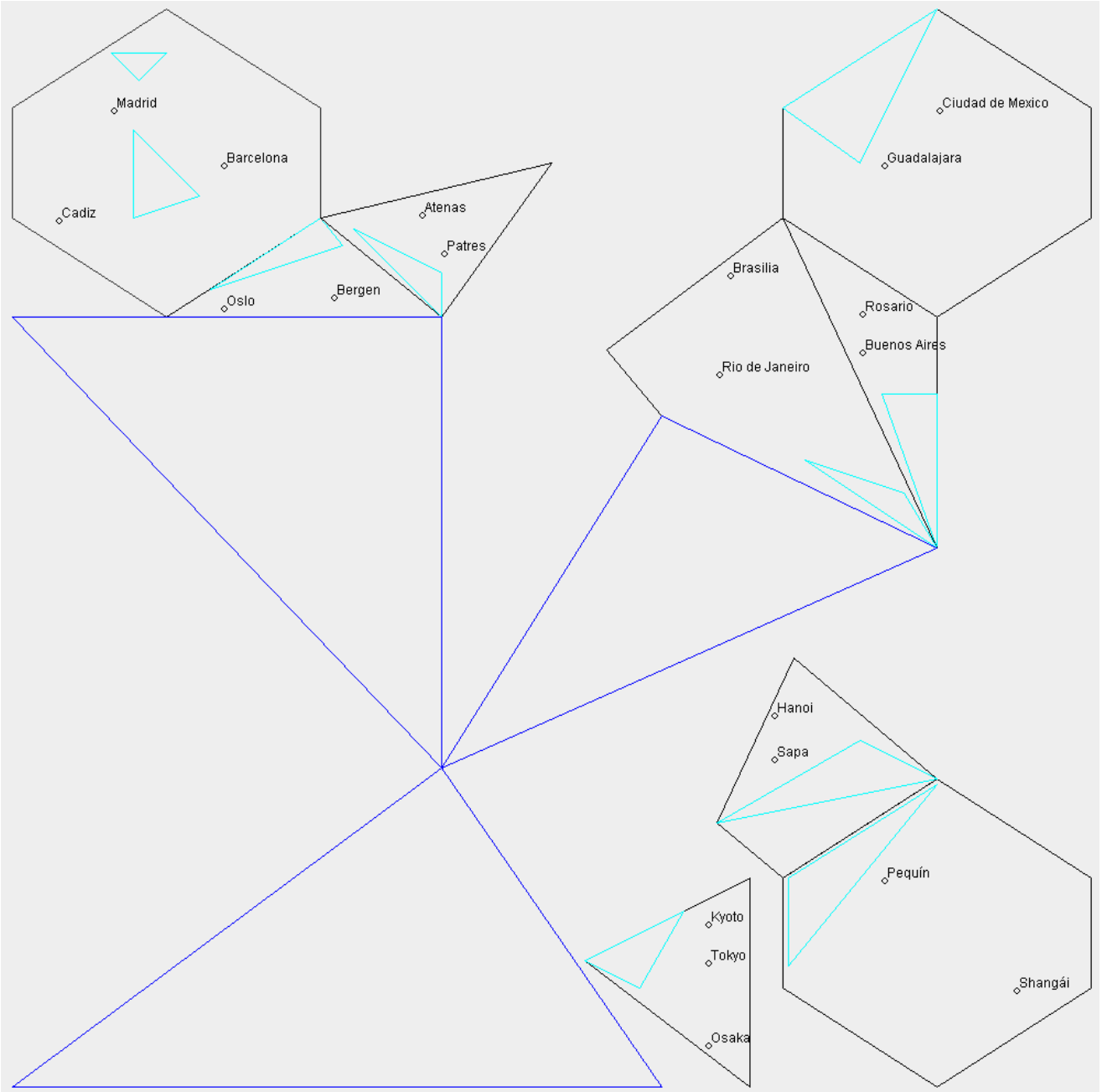
La práctica ha estado realizada de la manera correcta, dibujando las ciudades, los países, los lagos y los océanos correctamente, pudiendo probar esto al realizar varias ciudades por país, varios lagos y concretamente tres océanos para el mundo.

No hemos tenido grandes dificultades para realizar este lab, en parte porque consiste en expandir un trabajo previo y en parte porque cada vez estamos más familiarizados con el entorno de programación orientada a objetos; la gran mayoría de obstáculos con los que nos hemos encontrado vienen a causa de usar reales como coordenadas para el mapa, lo que muy probablemente hace que la implementación de nuestras funciones “draw” de las clases base del programa, es decir, “PolygonalRegion” y “GeoPoint”, sean bastante diferentes a las de otros trabajos.

Algo que ha requerido más tiempo del que esperabamos ha sido situar las ciudades y los lagos de manera correcta en cada país, para que no haya ciudades dentro de lagos, o lagos fuera de el pais, etc…
Siendo este proceso bastante lento al haber tantos lagos y tantas ciudades.

El resultado final al ejecutar el programa, dibujando todos sus continentes y océanos, es este:

Mapa final:


Madrid
Barcelona
Cadiz
Atenas
Patres
Oslo
Bergen
Ciudad de Mexico
Guadalajara
Brasilia
Rosario
Buenos Aires
Rio de Janeiro
Hanoi
Sapa
Pequín
Kyoto
Tokyo
Osaka
Shangái