

REPORT PRÀCTICA 2

- 1. An introduction where the problem is described. For example, what should the program do? Which classes do you have to define? Which methods do you have to define? Which methods do you have to implement for these classes?**

L'objectiu d'aquesta segona pràctica ha estat, principalment, implementar els conceptes adquirits en el Seminari 2. Hem començat, per tant, implementant les classes Regió, Continent i Món per estructurar i distribuir el nostre món. Addicionalment, hem implementat una classe Mapa per poder dibuixar la nostra distribució de classes, esmentades anteriorment.

Com a explicació general, en aquesta pràctica hem creat una classe anomenada PolygonalRegion, que crea figures geomètriques de tipus PolygonalRegion (regió). D'altra banda la classe Continent conté una llista de regions (el que fa referència als països [countries]). Finalment, la nostra classe World està definida per una llista de continents. Un cop creades totes les classes que fan referència al nostre món (relacionades entre elles mitjançant composició), a la classe MyMap (donada pel codi) creem instàncies per crear el nostre món i poder dibuixar-lo gràcies a l'altra classe, MyWindow, també donada a la pràctica.

Finalment les classes PolygonalRegion, Continents i World estan relacionades a la classe Graphics, donada per Java, per dependència. Aquesta funció és necessària per poder crear imatges fora del IDE i, per tant, representar (amb diversos mètodes explicats a continuació) instàncies d'aquestes.

Com a punt introductori, també volem esmentar que hem definit una classe inicial, la classe Point, que és la que anem implementant a les altres classes que defineixen el nostre món. Aquesta classe ja la vam definir en la pràctica 1 (GeometricPoint).

1.1. DEFINICIÓ DE POINT

Aquest ha estat la primera classe definida. Hem volgut seguir l'estructura que vam fer a la primera pràctica i, per tant, vam definir els atributs com coordenades X i Y de tipus double. El que ens interessa d'aquesta classe és que serà crucial per la posterior implementació de Point a les següents classes d'organització territorial (regions, continents i món). Un cop definits els atributs (en privat), hem definit els mètodes especials, és a dir, el constructor i els getters i setters (en públic).

Pel constructor passem per paràmetre els atributs de la classe i retornem les posicions. Pel que fa als getters, no passem per paràmetre cap variable i tan sols retornem el valor dels atributs. Cal destacar (com es veu a les fotografies), que hem creat un getter per cadascun dels atributs. Per tant tenim un getX i un getY. El mateix passa amb els setters, amb diferència que ara, passem per paràmetre l'atribut i establim un nou valor.

```
//Constructor
public Point(double x, double y){
    this.x = x;
    this.y = y;
}

public double getX(){
    return this.x;
}

public double getY(){
    return this.y;
}
```

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

Point és la classe més ‘primitiva’ de la pràctica. En altres paraules, no implementem cap mètode o classe en aquesta, sinó que és aquesta classe la que serà implementada posteriorment en altres classes. Per tant, és suficient amb definir els mètodes especials i no ens cal afegir cap mètode extra.

1.2. DEFINICIÓ I IMPLEMENTACIÓ DE POLYGONALREGION

Aquesta classe va ser la segona en definir. PolyogonalRegion ja és una classe més complexa de definir, doncs els seu únic atribut implica la implementació d’una variable de classe Point. Definim points (l’únic atribut de la classe) com una LinkedList de tipus Point. Això ens permet definir cada instància de tipus PolygonalRegion com una seqüència de punts. Indirectament, amb aquesta llista, estem definint quina forma geomètrica tindrà cada regió amb la quantitat de variables de tipus Point que contingui la llista.

Pel que fa als mètodes, comencem, com sempre, amb el constructor. La funció és la de sempre, crear una instància de tipus PolygonalRegion. Passem per paràmetre l’atribut i el declarem.

```
//Constructor  
public PolygonalRegion (LinkedList<Point> points){  
    this.points = points;  
}
```

A més, en aquesta classe sí necessitem diversos mètodes extra. D’una banda necessitem calcular l’àrea de la instància que estem definint en qüestió i, d’altra banda, necessitem una funció draw per poder dibuixar posteriorment les regions:

- a. getArea():** Per fer aquest mètode, ens hem basat en la fórmula proporcionada a l’enunciat de la pràctica. Necessitem, principalment, calcular l’àrea d’un polígon a partir dels punts donats. Seguim, per tant, la fórmula del càlcul de **l’àrea d’un polígon convex**:

$$\text{Area} = \frac{1}{2} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_n & y_n \\ x_1 & y_1 \end{vmatrix} = \frac{1}{2} [(x_1y_2 + x_2y_3 + x_3y_4 + \dots + x_ny_1) - (y_1x_2 + y_2x_3 + y_3x_4 + \dots + y_nx_1)]$$

Per implementar-ho en la nostra funció, vam utilitzar diverses variables i mètodes:

- **area:** És el nostre contador de l’àrea. Comença a 0 i es va actualitzant amb un for in.
- **n:** És la llargada de la llista de punts. Tenim el nombre de punts amb que està definida la nostra regió. També declarem fins on ha d’iterar el for in.

- j: Com el càlcul de l'àrea ha de ser en contra de les agulles del rellotge, veiem que j ha de ser el l'índex anterior a i. Aquest índex és necessari ja que veiem en la fórmula com hi ha multiplicacions de valors adjacents ($x_1y_2 + x_2y_3 + \dots + x_ny_1$).

Els mètodes utilitzats són el get(variable), implementat ja en Java, per obtenir cada posició de la LinkedList i els getters de la classe Point, per obtenir la coordenada desitjada per cada posició.

Finalment, el mètode al codi queda de la següent forma:

```
public double getArea(){  
    double area = 0;  
    int n = points.size();  
    int j = n - 1;  
    for (int i = 0; i < n; i++) {  
        area += (points.get(j).getX() + points.get(i).getX()) * (points.get(j).getY() - points.get(i).getY());  
        j = i;  
    }  
    return Math.abs(area / 2);  
}
```

- b. **drawP(Graphics g)**: En aquest mètode, creem tres variables de tipus int: nPoints (llargada de la llista de punts), xPoints (les coordenades X) i yPoints (les coordenades Y). Un cop obtinguts els valors de les X i les Y, cridem a la funció g.drawPolygon(nPoints, xPoints, yPoints), una funció ja implementada a Graphics, per dibuixar la nostra instància de tipus PolygonalRegion.

```
public void drawP(Graphics g) {  
    int nPoints = points.size();  
    int xPoints[] = new int[nPoints];  
    int yPoints[] = new int[nPoints];  
  
    for (int i = 0; i < points.size(); i++) {  
        xPoints[i] = (int) points.get(i).getX();  
        yPoints[i] = (int) points.get(i).getY();  
    }  
    g.drawPolygon(xPoints, yPoints, nPoints);  
}
```

1.3. DEFINICIÓ I IMPLEMENTACIÓ DE CONTINENT

La següent classe a definir va ser la classe Continent. Aquesta, seguint la jerarquia territorial, està composada de regions. Això ho apliquem a l'hora de definir l'atribut countries, una LinkedList de tipus PolygonalRegion. Aquesta classe, com l'anterior, consisteix del constructor, un mètode per calcular l'àrea total i un altre mètode per dibuixar el continent.

Definim el constructor passant per paràmetre countries, l'atribut i, actualitzant l'atribut.

```
//Constructor  
public Continent(LinkedList<PolygonalRegion> countries){  
    this.countries = countries;  
}
```

- a. **getTotalArea()**: Calculem l'àrea total del continent de la mateixa forma que hem declarat el mètode de càlcul de l'àrea a la classe PolygonalRegion:
- **totalArea**: És el contador de l'àrea del continent.

- **m:** És el nombre d'iteracions que haurem de fer en el for in. En altres paraules, és el nombre de regions que hi ha a la llista countries.

Implementem la funció getArea, per poder calcular directament l'àrea de continents. Això ho fem ja que, com una instància de classe Continent està feta d'instàncies de classe PolygonalRegion, podem cridar-la i obtenir el valor total de l'àrea per, finalment, retornar-ho.

```
public double getTotalArea(){  
    double totalArea = 0;  
    int m = countries.size();  
    for (int i = 0; i < m; i++){  
        totalArea += countries.get(i).getArea();  
    }  
    return totalArea;
```

- b. **drawC(Graphics g):** En aquest mètode, també cridem a la funció drawP per dibuixar el nostre continent (per la mateixa raó que podem cridar el getTotalArea). nCountries és el nombre de regions que hi ha a la llista countries. Amb el for in, per cada valor (coordenades de una regió) de la llista countries, cridem a la funció drawP (ja que cada posició de countries és una instància de tipus PolygonalRegion), per mostrar per pantalla les regions contingudes en un continent.

```
public void drawC(Graphics g) {  
    int nCountries = countries.size();  
    for(int i = 0; i < nCountries; i++){  
        countries.get(i).drawP(g);  
    }  
}
```

1.4. DEFINICIÓ I IMPLEMENTACIÓ DE WORLD

La classe World va ser la última classe a definir, relacionades amb l'estruatura de territoris. Els atributs d'aquesta és una LinkedList de tipus Continents, anomenada cons, on estan tots les instàncies de tipus Continents.

Pel que fa als mètodes, creem primer el constructor, on li passem per paràmetre l'atribut i, com els altres, l'actualitzem.

```
//Constructor  
public World(LinkedList<Continent> conts){  
    this.conts = conts;  
}
```

Com a mètode extra, fem el draw del nostre món, un mètode essencial posteriorment a la classe MyMap. En aquest cas, fem una implementació del mètode drawC. Cridem a continent perquè l'atribut de la classe world és de tipus Continent. Això fa que a l'hora de cridar a drawC, alhora es cridi a drawP, que aquest mateix crida a drawPolygon i, per tant, d'aquesta forma accedim a totes les instàncies de qualsevol classe del nostre món.

```
public void drawW(Graphics g) {  
    int nContinents = conts.size();  
    for(int i = 0; i < nContinents; i++){  
        conts.get(i).drawC(g);  
    }  
}
```

1.5. MODIFICACIÓ DE MYMAP

Un cop creades totes les classes, vam modificar el mòdul de MyMap. Aquí modifiquem la part de codi de **public MyMap()** i **public void paint(java.awt.Graphics g)**.

- a. **public MyMap():** En aquest mòdul, el principal objectiu va ser definir tot d'instàncies de diferents tipus per agrupar-les i, posteriorment, dibuixar-les. Per començar, vam decidir crear instàncies de tipus Point per guardar-les en diferents instàncies de tipus PolygonalRegion. Seguidament, vam fer llistes de tipus PolygonalRegion per guardar-les a instàncies de tipus continent. Finalment creem una instància de tipus world, que conté una llista de les tres instàncies de tipus continent que hem creat.
- b. **public void paint(java.awt.Graphics g):** Un cop definides les instàncies i implementades unes amb altres amb el mètode **classe.add(nomdelaclass)**, hem afegit la instrucció “world.drawW(g)” per poder dibuixar, gràcies al mòdul MyWindow, tot el nostre món, creat prèviament.

```
LinkedList< Point > points1 = new LinkedList< Point >();  
points1.add( new Point( 100, 190 ) );  
points1.add( new Point( 240, 100 ) );  
points1.add( new Point( 380, 190 ) );  
points1.add( new Point( 380, 290 ) );  
points1.add( new Point( 240, 380 ) );  
points1.add( new Point( 100, 290 ) );  
  
LinkedList< Point > points2 = new LinkedList< Point >();  
points2.add( new Point( 500, 540 ) );  
points2.add( new Point( 580, 500 ) );  
points2.add( new Point( 660, 540 ) );  
points2.add( new Point( 610, 700 ) );  
points2.add( new Point( 550, 700 ) );  
  
LinkedList< Point > points3 = new LinkedList< Point >();  
points3.add( new Point( 500, 100 ) );  
points3.add( new Point( 700, 100 ) );  
points3.add( new Point( 700, 300 ) );  
points3.add( new Point( 500, 300 ) );  
  
LinkedList< Point > points4 = new LinkedList< Point >();  
points4.add( new Point( 300, 500 ) );  
points4.add( new Point( 375, 630 ) );  
points4.add( new Point( 225, 630 ) );  
  
PolygonalRegion region1 = new PolygonalRegion( points1 );  
PolygonalRegion region2 = new PolygonalRegion( points2 );  
PolygonalRegion region3 = new PolygonalRegion( points3 );  
PolygonalRegion region4 = new PolygonalRegion( points4 );  
  
LinkedList< PolygonalRegion > countries1 = new LinkedList< PolygonalRegion >();  
countries1.add(region1);  
  
LinkedList< PolygonalRegion > countries2 = new LinkedList< PolygonalRegion >();  
countries2.add(region3);  
countries2.add(region4);  
  
LinkedList< PolygonalRegion > countries3 = new LinkedList< PolygonalRegion >();  
countries3.add(region2);  
  
Continent continent1 = new Continent(countries1);  
Continent continent2 = new Continent(countries2);  
Continent continent3 = new Continent(countries3);  
  
LinkedList< Continent > continents = new LinkedList<Continent>();  
continents.add(continent1);  
continents.add(continent2);  
continents.add(continent3);  
  
world = new World(continents);  
  
public void paint( java.awt.Graphics g ) {  
    super.paint( g );  
    world.drawW(g);  
}
```

2. A description of possible alternative solutions that were discussed, and a description of the chosen solution and the reason for choosing this solution rather than others. It is also a good idea to mention the related theoretical concepts of object-oriented programming that were applied as part of the solution.

Durant la pràctica, ens hem trobat amb diverses dificultats i incògnites que ens han fet redefinir certs mètodes.

Principalment, vam incloure directament tots els mòduls que hem necessitat per fer la pràctica en una carpeta conjunta, per relacionar-los directament i no tenir problemes de package, com ens va passar a l'altre pràctica.

Seguidament, vam haver d'importar diverses classes per poder realitzar correctament la pràctica. D'una banda, vam haver d'importar LinkedList a les classes PolygonalRegion, Continent i World, ja que són les classes que tenen implementats atributs amb llistes. D'altra banda, també vam haver d'importar la classe Graphics en aquestes mateixes classes perquè la funció draw funcionés correctament. Tot i així, al mòdul MyMap no el vam haver d'implementar perquè només l'utilitzem a la funció paint i ja ve importat com a paràmetre.

Un altre aspecte a mencionar ha estat els atributs de la classe Point. Inicialment els vam establir de tipus int. Tot i així, ens funcionava bé fins arribar als mètodes de càcul de l'àrea. Vam veure que, al establir les variables de tipus int, l'àrea no era exacte, ja que truncava el resultat. Per tant, vam decidir declarar els atributs X i Y com a doubles, així com les variables locals de les funcions d'àrea i, després, a la funció draw, els vam castejar com a ints de forma explícita. Això ho vam fer degut a que la funció drawPolygon() només accepta paràmetres de tipus int.

```
for (int i = 0; i < points.size(); i++) {  
    xPoints[i] = (int) points.get(i).getX();  
    yPoints[i] = (int) points.get(i).getY();  
}
```

Conversió explícita de double a int:
Indiquem, entre parèntesis, el tipus de variable que volem forçar a convertir.

Finalment, en un primer moment vam decidir utilitzar la funció drawLine (int x_1 , int y_1 , int x_2 , int y_2) a la funció draw, però vam veure que no era tan eficient com la de drawPolygon. Com a drawLine necessitàvem dos parelles de punts, vam creure que no era convenient perquè havíem de treballar amb masses variables. Per tant, vam decidir reemplaçar la funció per la de drawPolygon, que només necessita un array de punts X, un array de punts Y i un nombre total de punts. Per nosaltres era més fàcil, ja que teníem una matriu de coordenades 2XN, en altres paraules, una llista de punts. Per tant, vam dividir totes les coordenades en dos arrays (xPoints[N] i yPoints[N]) i el size de la llista era el nostra nombre total de punts (nPoints).

3. A conclusion that describes how well the solution worked in practice, i.e. did the tests show that the classes were correctly implemented? You can also mention any difficulties during the implementation as well as any doubts you might have had

Com a conclusions generals de la pràctica, ens hem vist més solvents i coordinats relacionats amb la primera. Creiem que, un cop ja havent entrat en matèria i realitzat dos seminaris i un laboratori previ, hem trobat més fàcil aquest laboratori. Hem pogut traslladar amb més facilitat els conceptes abstractes apresos a teoria i els mecanismes pràctics escrits a full dels seminaris. A més, hem notat que cada cop estem més familiaritzats amb l'entorn Visual Studio Code i el llenguatge de programació en sí.

D'altra banda, la redacció dels informes (sobretot el d'aquesta pràctica) ens ha ajudat a entendre bé què ha fet cadascun de la pràctica i hem compartit moltes idees en comú a l'hora de redactar-ho, pel qual creiem que hem fet un bon treball a nivell organitzatiu i cooperatiu.

Tot i així, volem destacar un parell de dubtes que s'han generat mentre hem estat fent la pràctica que, creiem que hem resolt, però no hem acabat d'entendre si la manera de resoldre-ho era la correcta.

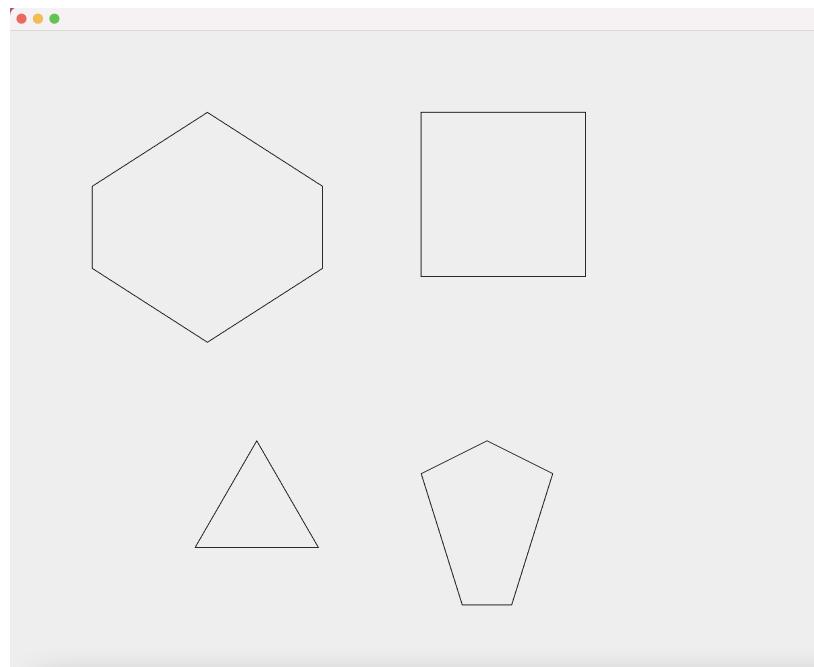
Principalment, al punt 3.3 de l'enunciat de la pràctica, diu que s'ha d'editar el mòdul de MyMap per demostrar que totes les classes i mètodes estan ben definides i implementades. Per tant vam programar totes les instàncies al mòdul MyMap. Tot i així, a l'enunciat s'esmenta que el main method ha de crear totes les regions, continents i el món. En aquest punt ens va generar el dubte de si s'havia d'editar el MyWindow, ja que allà apareix el main principal. No obstant, com ens funcionava tot havent-ho editat al MyMap, vam decidir no modificar res.

Altrament, volíem demostrar que les funcions getArea() i getTotalArea() funcionaven correctament. Vam crear les funcions i vam fer unes instàncies d'exemple abans de seguir amb les següents classes. Vam printejar les àrees (calculades al codi) i, simultàniament, vam calcular-les de forma manual. Així doncs, quan vam tenir diversos exemples que ens coincidien, vam continuar amb el codi, concloent que les funcions estaven correctes.

```
Àrea de la regió 1: 53200.0 km^2 -> hexàgon irregular.  
Àrea de la regió 2: 20800.0 km^2 -> pentàgon irregular.  
Àrea de la regió 3: 40000.0 km^2 -> quadrat.  
Àrea de la regió 4: 9750.0 km^2 -> triangle equilàter.  
Continent 1 = Hexàgon irregular.  
Àrea del continent 1: 53200.0 km^2.  
Continent 2 = Quadrat + Triangle equilàter.  
Àrea del continent 2: 49750.0 km^2.  
Continent 3 = Pentàgon irregular.  
Àrea del continent 3: 20800.0 km^2.
```

Finalment, el resultat del nostre programa apareix de la següent forma:

PRINT DEL NOSTRE MÓN



INTERPRETACIÓ DE LA DISTRIBUCIÓ DEL NOSTRE MÓN

