PRIS – Source Monitor

Curso 2019/2020

Juan José Pallarés Sánchez

Índice

1. Introducción

En este documento se recogen los datos sobre los diferentes checkpoints para la mejora del código del proyecto “Polígonos”. Este proyecto está compuesto de 5 clases las cuales hacen referencias a las diferentes partes que conforman una figura geométrica.

2. Baseline 1

En primer lugar, tras la primera ejecución podemos observar que de las 5 clases existen 2 que son razonablemente mas complejas que las demás (Tabla 1).

File Name	*Lines*	*Statements*	*% Branches*	*Calls*	*% Comments*
Poligono.java	68	39	15,4	19	7,4
Linea.java	30	17	5,9	1	3,3
Punto.java	33	17	0	1	0
UsoPoligono.java	57	41	0	24	5,3
Vector.java	33	17	0	8	0

Classes	*Methods/Class*	*Avg Stmts/Method*	*Max Complexity*	*Max Depth*	*Avg Depth*	*Avg Complexity*
1	4	7,75	4	4	1,97	2,5
1	4	2,5	2	3	1,53	1,25
1	6	1,17	1	2	1,29	1
1	1	37	1	2	1,83	1
1	6	1,17	1	2	1,29	1

Tabla 1 Base Line 1

Entrando paso por paso podemos descartar las clases “Punto.java” y “Vector.java” que simplemente son clases modelos que instancia los objetos base que conforma una figura geométrica.

Llegados a este punto, pasamos a analizar las 3 clases restantes “Polígono, Línea y UsoPoligono” de mayor complejidad a menor.

2.1. Poligono

Partiendo de este punto, vemos que el código está compuesto por 68 líneas de las cuales 39 son sentencias y 19 son llamadas, compuestos por 4 métodos. Al analizar el código se observa que solamente 2 de esos 4 métodos tienen la total complejidad.

Se debe de mantener cierta convención a la hora de implementar el código y sobre todo utilizar un solo idioma.

```
    ArrayList<Punto> poligono;

    public Poligono(ArrayList<Punto> poligon) {
        poligono = poligon;
    }
```

Además, los comentarios en medio del código, aunque pueden estar bien para aclararlo es mejor usar una sintaxis más limpia y no crear comentarios que induzcan a error. Por último, si se debe devolver un balón puede transformar las ultimas 5 líneas en un solo return más simple y sencillo.

```
public boolean concavo(){
        Vector vec1, vec2;

        //Comprobamos el signo del primer producto vectorial
        int i = 0;
        //Creamos vectores a partir de los primeros puntos
        vec1 = new Vector(poligono.get(i), poligono.get(i+1));
        vec2 = new Vector(poligono.get(i+1), poligono.get(i+2));
        double prodVectorial = vec1.productoVectorial(vec2);
        boolean positivo = prodVectorial >= 0;

        //Calculamos el signo de los dem�s productos vectoriales
        for (i = 1; i < poligono.size()-2; i++) {
            vec1 = new Vector(poligono.get(i), poligono.get(i+1));
            vec2 = new Vector(poligono.get(i+1), poligono.get(i+2));
            prodVectorial = vec1.productoVectorial(vec2);
            //Si tienen distinto signo es c�ncavo
            if (positivo != (prodVectorial >= 0)) {
                return true;
            }
        }
        //Calculamos el producto vectorial del �ltimo y el primer vector
        vec1 = new Vector(poligono.get(i+1), poligono.get(0));
        prodVectorial = vec2.productoVectorial(vec1);
        if (positivo != (prodVectorial >= 0)) {
            return true;
        }
        return false;
    }
```

Agregar que los métodos culla carga de operaciones aritméticas sea extremadamente alta, es posible realizar una buena indentación

```
    public Punto centroide(){
        double X=0, Y=0, A=0;
        int mod = poligono.size();
        for (int k = 0; k < poligono.size(); k++) {
            A+=poligono.get(k).getPosX()*poligono.get((k+1)%mod).getPosY(
)-poligono.get((k+1)%mod).getPosX()*poligono.get(k).getPosY();
        }
        A=A/2;

        for (int k = 0; k < poligono.size(); k++) {
```

```
            X+=(poligono.get(k).getPosX()+poligono.get((k+1)%mod).getPosX
())*(poligono.get(k).getPosX()*poligono.get((k+1)%mod).getPosY()-
poligono.get((k+1)%mod).getPosX()*poligono.get(k).getPosY());
        }
        X=X/(6*A);

        for (int k = 0; k < poligono.size(); k++) {
            Y+=(poligono.get(k).getPosY()+poligono.get((k+1)%mod).getPosY
())*(poligono.get(k).getPosX()*poligono.get((k+1)%mod).getPosY()-
poligono.get((k+1)%mod).getPosX()*poligono.get(k).getPosY());
        }
        Y=Y/(6*A);
        return new Punto((int) X, (int)Y);
    }
```

Todos estos problemas, hacen que una clase simple de 4 métodos tenga la carga principal del código.

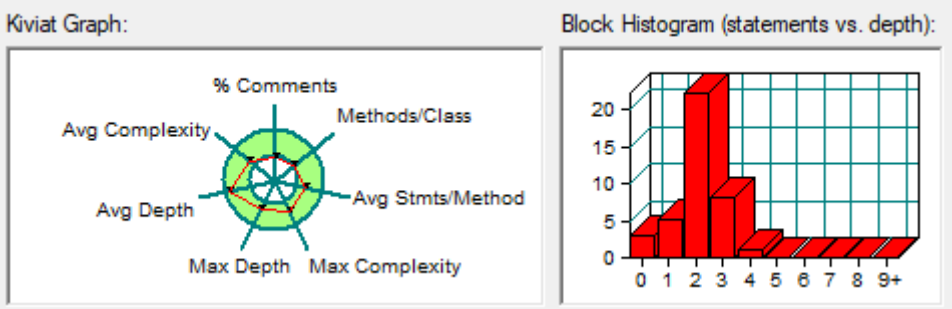

Ilustración 1 Graficos

2.2. Linea

Esta clase se trata de otra clase modelo que debe de tener una complejidad baja. No obstante, aunque bien hecho un desglose del código puede llegar a mejorar la comprensión de este. Es mejor en casos como este, comprimir el código y meter la operación aritmética bien indentada dentro de la ultima línea. Además, es posible comprimir el primer return también en una sola línea más limpia y fácil de leer.

```
    public String puntoCorte(Linea otra){
        double coordenadaX, coordenadaY;
        if (this.x == otra.x) {
            return "paralelas o coincidentes";
        }
        coordenadaX = (this.n-otra.n)/(this.x-otra.x)*(-1);
        coordenadaY = this.x*coordenadaX+this.n;
        return "("+coordenadaX+", "+coordenadaY+")";
    }
```

2.3. UsoPoligono

Esta clase se trata de la clase “main” que hace uso de las diferentes funciones de los polígonos. Aunque está orientada a un uso simple e intensivo debería de mantener un nombre descriptivo de clase y de variables. Concretamente por convenio las clases principales suelen denominarse “APP o PROGRAM” como clase principal.

```
public class UsoPoligono {

    public static void main(String[] args) {
```

Por último, no es necesario crear 11 líneas distintas para imprimir un solo bloque de código, en su lugar es más eficiente un solo string bien formado y sin problemas.

```
        System.out.println(puntoA);
        System.out.println(puntoB);
        System.out.println(puntoC);
        System.out.println(vector1);
        System.out.println(vector2);
        System.out.println(poligono1.concavo());
        System.out.println(poligono2.concavo());
        System.out.println(poligono2.centroide());
        System.out.println(ln);
        System.out.println(ln2);
        System.out.println(ln.puntoCorte(ln2));
```