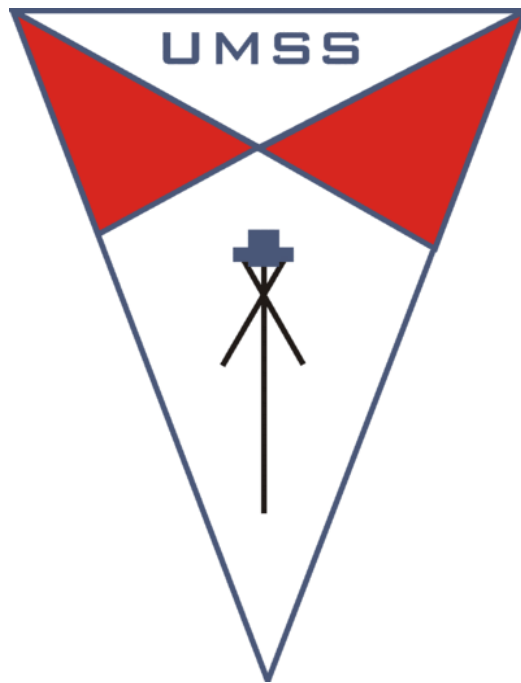


UNIVERSIDAD MAYOR DE SAN SIMÓN  
FACULTAD DE CIENCIAS Y TECNOLOGÍA



## **PRACTICA NRO 4 - BAD SMELLS : TRIANGULO**

### **MATERIA: PROGRAMACIÓN**

NOMBRE:

MORUNO RODRIGUEZ TATIANA ANDREA

PARA REVISION DE CODIGO BAD SMELLS:

GARCES HUAPALLA PAMELA MARIEL

CARRERA:INGENIERÍA INFORMÁTICA

DOCENTE: LIC. ROSEMARY TORRICO BASCOPE

FECHA: 13-JUNIO-2016

COCHABAMBA - BOLIVIA

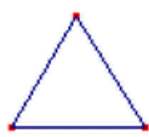






## I. PLANTEAMIENTO DEL PROBLEMA

En base a los conceptos de bad smells, implementar un sistema para representar un Triángulo, donde se pueda identificar su clasificación. Y propiedades como lados y ángulos.

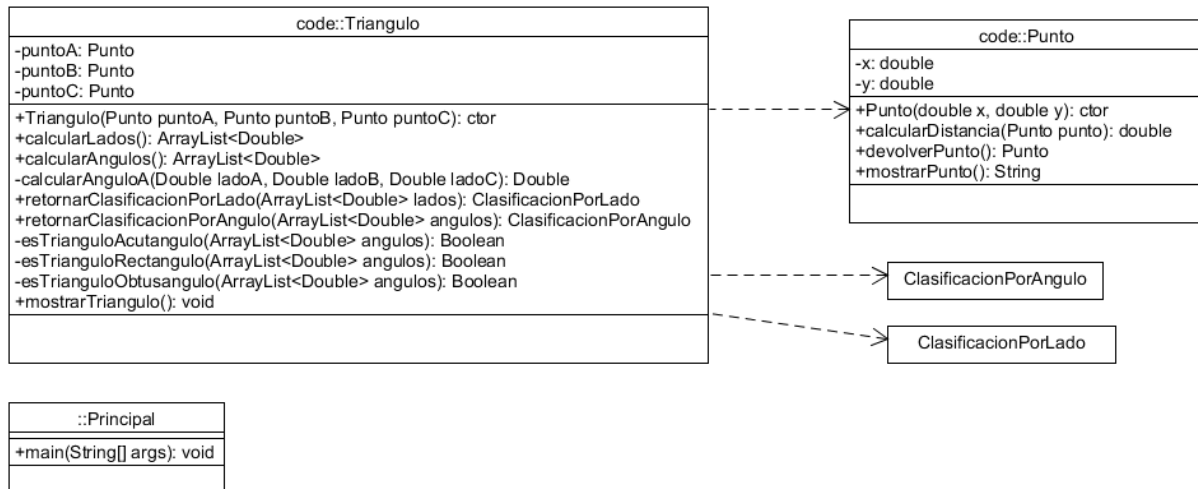
## II. DESCRIPCIÓN (ESTRATEGIA DE SOLUCIÓN)

En base al problema planteado, se implementó una solución en el lenguaje de programación Java. El proyecto partió de un diseño de clases y se aplicaron las siguientes estrategias:

- Una clase Triangulo es representada a través de 3 puntos.
- Se implementa la clase punto, la cual se representa a través de 2 coordenadas: x,y .
- La clase punto presenta métodos que calculan la distancia entre 2 puntos.
- La clase Triangulo presenta métodos para identificar sus lados, ángulos y en base a ellos, se identifica la clasificación por lado y por ángulo
- Las posibles clasificaciones por lado son: Equilátero, Isósceles, Escaleno.
- Las posibles clasificaciones por ángulo son: Acutángulo, Rectángulo, Obtusángulo.
- En la siguiente imagen se describe las posibles combinaciones de las clasificaciones:

CLASIFICACIÓN DE LOS TRIÁNGULOS			
Según sus lados Según sus ángulos	EQUILÁTERO 3 lados iguales 3 ángulos iguales	ISÓSCELES 2 lados iguales 2 ángulos iguales	ESCALENO 3 lados desiguales 3 ángulos desiguales
ACUTÁNGULO 3 ángulos agudos			
RECTÁNGULO 1 ángulo recto 2 ángulos agudos	No existe		
OBTUSÁNGULO 1 ángulo obtuso 2 ángulos agudos	No existe		

### III. DIAGRAMA DE CLASES



### IV. CÓDIGO FUENTE (código original, sin refactorización)

#### Clase Principal

```
import java.util.Random;
import code.Punto;
import code.Triangulo;

public class Principal {

    public static void main(String[] args) {

        Random rand = new Random();
        double min = -20;
        double max = 20;

        // Equilatero/acutangulo
        Punto p1 = new Punto(7.3, 3.1);
        Punto p2 = new Punto(1.0, -4.0);
        Punto p3 = new Punto(-2.0, 5.0);

        Triangulo trianguloEquilateroAcutangulo = new Triangulo(p1, p2, p3);
        trianguloEquilateroAcutangulo.mostrarTriangulo();

        // Isosceles/acutangulo
        p1 = new Punto(-2.0, -2.0);
        p2 = new Punto(3.0, -2.0);
        p3 = new Punto(0.0, 2.0);

        Triangulo trianguloIsoscelesAcutangulo = new Triangulo(p1, p2, p3);
        trianguloIsoscelesAcutangulo.mostrarTriangulo();

        // Escaleno/acutangulo
        p1 = new Punto(2.0, 2.0);
        p2 = new Punto(3.0, 8.0);
        p3 = new Punto(7.0, 2.0);
```

```

Triangulo trianguloEscalenoAcutangulo = new Triangulo(p1, p2, p3);
trianguloEscalenoAcutangulo.mostrarTriangulo();

// Isosceles/rectangulo
p1 = new Punto(6.0, 0.0);
p2 = new Punto(0.0, 6.0);
p3 = new Punto(0.0, 0.0);

Triangulo trianguloIsoscelesRectangulo = new Triangulo(p1, p2, p3);
trianguloIsoscelesRectangulo.mostrarTriangulo();

// Escaleno/rectangulo
p1 = new Punto(2.0, 2.0);
p2 = new Punto(2.0, 8.0);
p3 = new Punto(7.0, 2.0);

Triangulo trianguloEscalenoRectangulo = new Triangulo(p1, p2, p3);
trianguloEscalenoRectangulo.mostrarTriangulo();

// Isosceles/obtusangulo
p1 = new Punto(-5.12, 3.12);
p2 = new Punto(0.0, 0.0);
p3 = new Punto(6.0, 0.0);

Triangulo trianguloIsoscelesObtusangulo = new Triangulo(p1, p2, p3);
trianguloIsoscelesObtusangulo.mostrarTriangulo();

// Escaleno/acutangulo
/*p1 = new Punto(2.0, -9.0);
p2 = new Punto(-10.0, -17.0);
p3 = new Punto(-18.0, 17.0);

Triangulo trianguloEscalenoAcutangulo = new Triangulo(p1, p2, p3);
trianguloEscalenoAcutangulo.mostrarTriangulo();*/

// Escaleno/obtusangulo
p1 = new Punto(-9.0, 2.0);
p2 = new Punto(-10.0, -17.0);
p3 = new Punto(-18.0, 17.0);

Triangulo trianguloEscalenoObtusangulo = new Triangulo(p1, p2, p3);
trianguloEscalenoObtusangulo.mostrarTriangulo();

// triangulos randomicos
for(int i=0; i<10; i++){

    double randomNumX = (max - min) * rand.nextDouble() + min;
    double randomNumY = (max - min) * rand.nextDouble() + min;
    Punto punto1 = new Punto(randomNumX, randomNumY);

    randomNumX = (max - min) * rand.nextDouble() + min;
    randomNumY = (max - min) * rand.nextDouble() + min;
    Punto punto2 = new Punto(randomNumX, randomNumY);

    randomNumX = (max - min) * rand.nextDouble() + min;
    randomNumY = (max - min) * rand.nextDouble() + min;
    Punto punto3 = new Punto(randomNumX, randomNumY);

    Triangulo triangulo = new Triangulo(punto1, punto2, punto3);
    triangulo.mostrarTriangulo();
}

```

```
    }  
    }  
}
```

---

### **Clase Punto**

```
package code;  
  
public class Punto {  
    private double x;  
    private double y;  
  
    public Punto(double x,double y) {  
        this.x = Math rint(x*100)/100;  
        this.y = Math rint(y*100)/100;  
    }  
  
    public double calcularDistancia(Punto punto) {  
        return Math rint(Math.sqrt(Math.pow(punto.x - this.x,2) +  
        Math.pow(punto.y - this.y, 2))*100)/100;  
    }  
  
    public Punto devolverPunto() {  
        return this;  
    }  
  
    public String mostrarPunto() {  
        return ("x:" + this.x + " y:" + this.y);  
    }  
}
```

---

### **Clase Triangulo**

```
package code;  
  
import java.util.ArrayList;  
import lib.ClasificacionPorAngulo;  
import lib.ClasificacionPorLado;  
  
public class Triangulo {  
    private Punto puntoA;  
    private Punto puntoB;  
    private Punto puntoC;  
  
    public Triangulo(Punto puntoA, Punto puntoB, Punto puntoC) {  
        this.puntoA = puntoA;  
        this.puntoB = puntoB;  
        this.puntoC = puntoC;  
    }  
  
    public ArrayList<Double> calcularLados() {  
        ArrayList<Double> lados = new ArrayList<Double>();  
  
        Double ladoAB = puntoA.calcularDistancia(puntoB);  
        Double ladoBC = puntoB.calcularDistancia(puntoC);  
        Double ladoCA = puntoC.calcularDistancia(puntoA);  
  
        lados.add(ladoAB);  
    }  
}
```

```

        lados.add(ladoBC);
        lados.add(ladoCA);

        return lados;
    }

    public ArrayList<Double> calcularAngulos() {
        ArrayList<Double> angulos = new ArrayList<Double>();
        ArrayList<Double> lados = calcularLados();

        Double ladoA = lados.get(0);
        Double ladoB = lados.get(1);
        Double ladoC = lados.get(2);

        Double anguloCAB = calcularAnguloA(ladoA, ladoB, ladoC);
        Double anguloABC = calcularAnguloA(ladoB, ladoC, ladoA);
        Double anguloBCA = calcularAnguloA(ladoC, ladoA, ladoB);

        angulos.add(anguloCAB);
        angulos.add(anguloABC);
        angulos.add(anguloBCA);

        return angulos;
    }

    private Double calcularAnguloA(Double ladoA, Double ladoB, Double ladoC) {
        Double numerador = Math.pow(ladoA, 2) - Math.pow(ladoB, 2) -
Math.pow(ladoC, 2);
        Double denominador = -2 * ladoB * ladoC;
        Double cosenoA = Math.acos(numerador/denominador);
        return Math rint(Math.toDegrees(cosenoA)*100)/100;
    }

    public ClasificacionPorLado retornarClasificacionPorLado(ArrayList<Double> lados)
{
    int repetidos = 0;
    for (int i=0; i<= lados.size(); i++) {
        if (i+1 < lados.size()){
            for(int j=i+1; j<lados.size();j++){

                if (lados.get(i).equals(lados.get(j)))
                    repetidos++;
            }
        }
    }

    switch (repetidos) {
        case 0:
            return ClasificacionPorLado.ESCALENO;

        case 1:
            return ClasificacionPorLado.ISOSCELES;

        case 2:
            return ClasificacionPorLado.EQUILATERO;
        case 3:
            return ClasificacionPorLado.EQUILATERO;
        default:
            return ClasificacionPorLado.DESCONOCIDO;
    }
}

    public ClasificacionPorAngulo retornarClasificacionPorAngulo(ArrayList<Double>
angulos) {

        if (esTrianguloAcutangulo(angulos))

```

```

        return ClasificacionPorAngulo.ACUTANGULO;
    else if (esTrianguloRectangulo(angulos))
        return ClasificacionPorAngulo.RECTANGULO;
    else if (esTrianguloObtusangulo(angulos))
        return ClasificacionPorAngulo.OBTUSANGULO;
    else
        return ClasificacionPorAngulo.DESCONOCIDO;
}

private Boolean esTrianguloAcutangulo(ArrayList<Double> angulos) {
    int agudos = 0;
    for (int i=0; i< angulos.size(); i++){
        if (angulos.get(i)<90 && angulos.get(i) > 0)
            agudos++;
    }

    return (agudos == 3);
}

private Boolean esTrianguloRectangulo(ArrayList<Double> angulos) {
    int recto = 0;
    for (int i=0; i< angulos.size(); i++){
        if (angulos.get(i).intValue() ==90)
            recto++;
    }

    return (recto == 1);
}

private Boolean esTrianguloObtusangulo(ArrayList<Double> angulos) {
    int agudos = 0;
    int obtuso = 0;

    for (int i=0; i< angulos.size(); i++){
        if (angulos.get(i)<90 && angulos.get(i) > 0)
            agudos++;
        if (angulos.get(i)>90)
            obtuso++;
    }

    return (agudos == 2 && obtuso == 1);
}

public void mostrarTriangulo(){
    ArrayList<Double> lados = calcularLados();
    ArrayList<Double> angulos = calcularAngulos();

    System.out.println("Datos del Triángulo:");
    System.out.println("\t PuntoA: " + puntoA.mostrarPunto() +
        "\t PuntoB: " + puntoB.mostrarPunto() +
        "\t PuntoC: " + puntoC.mostrarPunto());
    System.out.println("\t Lados: " + lados.toString());
    System.out.println("\t Ángulos: " + angulos.toString());
    System.out.println("\t Clasificación por lado: " +
retornarClasificacionPorLado(lados).toString());
    System.out.println("\t Clasificación por ángulo: " +
retornarClasificacionPorAngulo(angulos).toString());
}
}

```

---

### **Clase ClasificacionPorAngulo**

```
package lib;
```

```
public enum ClasificacionPorAngulo {  
    ACUTANGULO, RECTANGULO, OBTUSANGULO, DESCONOCIDO  
}
```

---

### ***Clase ClasificacionPorLado***

```
package lib;  
  
public enum ClasificacionPorLado {  
    EQUILATERO, ISOSCELES, ESCALENO, DESCONOCIDO  
}
```



## V. BAD SMELLS DETECTADOS

```
        lados.add(ladoBC);
        lados.add(ladoCA);

        return lados;
    }

    public ArrayList<Double> calcularAngulos() {
        ArrayList<Double> angulos = new ArrayList<Double>();
        ArrayList<Double> lados = calcularLados();

        Double ladoA = lados.get(0);
        Double ladoB = lados.get(1);
        Double ladoC = lados.get(2);

        Double anguloCAB = calcularAnguloA(ladoA, ladoB, ladoC);
        Double anguloABC = calcularAnguloA(ladoB, ladoC, ladoA);
        Double anguloBCA = calcularAnguloA(ladoC, ladoA, ladoB);

        angulos.add(anguloCAB);
        angulos.add(anguloABC);
        angulos.add(anguloBCA);

        return angulos;
    }

    private Double calcularAnguloA(Double ladoA, Double ladoB, Double ladoC) {
        Double numerador = Math.pow(ladoA, 2) - Math.pow(ladoB, 2) -
        Math.pow(ladoC, 2);
        Double denominador = -2 * ladoB * ladoC;
        Double cosenoA = Math.acos(numerador/denominador);
        return Math rint(Math.toDegrees(cosenoA)*100)/100;
    }

    public ClasificacionPorLado retornarClasificacionPorLado(ArrayList<Double> lados)
    {
        int repetidos = 0;
        for (int i=0; i<= lados.size(); i++) {
            if (i+1 < lados.size()){
                for(int j=i+1; j<lados.size();j++){
                    if (lados.get(i).equals(lados.get(j)))
                        repetidos++;
                }
            }
        }

        switch (repetidos) {
            case 0:
                return ClasificacionPorLado.ESCALENO;

            case 1:
                return ClasificacionPorLado.ISOSCELES;

            case 2:
                return ClasificacionPorLado.EQUILATERO;
            case 3:
                return ClasificacionPorLado.EQUILATERO;
            default:
                return ClasificacionPorLado.DESCONOCIDO;
        }
    }

    public ClasificacionPorAngulo retornarClasificacionPorAngulo(ArrayList<Double>
    angulos) {

        if (esTrianguloAcutangulo(angulos))
    }
```

*porque no son atributos de la clase?*

*nombre no es muy claro*

*no es necesario tener ambos casos (codigo innecesario)*

## VI. CODIGO REFACTORIZADO CON CORRECCIONES DE BAD SMELLS

**ROJO** – representa el código eliminado

**VERDE** – representa el código agregado

```
10 ■■■■ src/Principal.java View
❖ @@ -58,15 +58,7 @@ public static void main(String[] args) {
58 58
59 59     Triangulo trianguloIsoscelesObtusangulo = new Triangulo(p1, p2, p3);
60 60     trianguloIsoscelesObtusangulo.mostrarTriangulo();
61 -
62 -     // Escaleno/acutangulo
63 -     /*p1 = new Punto(2.0, -9.0);
64 -     p2 = new Punto(-10.0, -17.0);
65 -     p3 = new Punto(-18.0, 17.0);
66 -
67 -     Triangulo trianguloEscalenoAcutangulo = new Triangulo(p1, p2, p3);
68 -     trianguloEscalenoAcutangulo.mostrarTriangulo();*/
69 -
70 61 +
70 62     // Escaleno/obtusangulo
71 63     p1 = new Punto(-9.0, 2.0);
72 64     p2 = new Punto(-10.0, -17.0);
❖
```

```
6 ■■■■ src/code/Punto.java View
❖ @@ -12,11 +12,7 @@ public Punto(double x,double y) {
12 12     public double calcularDistancia(Punto punto) {
13 13         return Math rint(Math.sqrt(Math.pow(punto.x - this.x,2) + Math.pow(punto.y - this.y, 2))*100)/100;
14 14     }
15 -
16 -     public Punto devolverPunto() {
17 -         return this;
18 -     }
19 -
20 15 +
20 16     public String mostrarPunto() {
21 17         return ("x:" + this.x + " y:" + this.y);
22 18     }
❖
```

```
6 ■■■■ src/code/Punto.java View
❖ @@ -12,11 +12,7 @@ public Punto(double x,double y) {
12 12     public double calcularDistancia(Punto punto) {
13 13         return Math rint(Math.sqrt(Math.pow(punto.x - this.x,2) + Math.pow(punto.y - this.y, 2))*100)/100;
14 14     }
15 -
16 -     public Punto devolverPunto() {
17 -         return this;
18 -     }
19 -
20 15 +
20 16     public String mostrarPunto() {
21 17         return ("x:" + this.x + " y:" + this.y);
22 18     }
❖
```

20		src/code/Triangulo.java	<div>View</div>	
		⚡	@@ -37,9 +37,9 @@ public Triangulo(Punto puntoA, Punto puntoB, Punto puntoC) {	
37	37		Double ladoB = lados.get(1);	
38	38		Double ladoC = lados.get(2);	
39	39			
40		-	Double anguloCAB = calcularAnguloA(ladoA, ladoB, ladoC);	
41		-	Double anguloABC = calcularAnguloA(ladoB, ladoC, ladoA);	
42		-	Double anguloBCA = calcularAnguloA(ladoC, ladoA, ladoB);	
	40	+	Double anguloCAB = calcularAngulo(ladoA, ladoB, ladoC);	
	41	+	Double anguloABC = calcularAngulo(ladoB, ladoC, ladoA);	
	42	+	Double anguloBCA = calcularAngulo(ladoC, ladoA, ladoB);	
43	43			
44	44		angulos.add(anguloCAB);	
45	45		angulos.add(anguloABC);	
		⚡	@@ -48,11 +48,12 @@ public Triangulo(Punto puntoA, Punto puntoB, Punto puntoC) {	
48	48		return angulos;	
49	49		}	
50	50			

20		src/code/Triangulo.java	<div>View</div>	
		@@ -37,9 +37,9 @@ public Triangulo(Punto puntoA, Punto puntoB, Punto puntoC) {		
50	50			
51	-	private Double calcularAnguloA(Double ladoA, Double ladoB, Double ladoC) {		
52	-	Double numerador = Math.pow(ladoA, 2) - Math.pow(ladoB, 2) - Math.pow(ladoC, 2);		
53	-	Double denominador = -2 * ladoB * ladoC;		
54	-	Double cosenoA = Math.acos(numerador/denominador);		
55	-	return Math rint(Math.toDegrees(cosenoA)*100)/100;		
	51	private Double calcularAngulo(Double ladoCatetoOpuestoA, Double ladoCatetoAdyacenteB, Double ladoCatetoAdyacenteC) {	+	
	52	Double numerador = Math.pow(ladoCatetoOpuestoA, 2) - Math.pow(ladoCatetoAdyacenteB, 2) -	+	
	53	Math.pow(ladoCatetoAdyacenteC, 2);	+	
	54	Double denominador = -2 * ladoCatetoAdyacenteB * ladoCatetoAdyacenteC;	+	
	55	Double arcosenoA = Math.acos(numerador/denominador);	+	
	56	return Math rint(Math.toDegrees(arcosenoA)*100)/100;	+	
56	57	}		

20		src/code/Triangulo.java
⚡		@@ -37,9 +37,9 @@ public Triangulo(Punto puntoA, Punto puntoB, Punto puntoC) {
58	59	public ClasificacionPorLado retornarClasificacionPorLado(ArrayList<Double> lados) {
⚡		@@ -71,8 +72,7 @@ public ClasificacionPorLado retornarClasificacionPorLado(ArrayList<Double> lados
71	72	return ClasificacionPorLado.ESCALENO;
72	73	case 1:
73	74	return ClasificacionPorLado.ISOSCELES;
74	-	case 2:
75	-	return ClasificacionPorLado.EQUILATERO;
	75	case 2:
76	76	case 3:
77	77	return ClasificacionPorLado.EQUILATERO;
78	78	default: