



UNIVERSIDAD  
**SAN IGNACIO  
DE LOYOLA**

**ACTIVIDAD**

**Docente:**

GIANNY ROMIE ALFARO GUTIERREZ

**Alumna:**

BAUTISTA ARRILUCEA, ROMINA ZORAIDA PERLA

2024-2

### Actividad 1.

1. Crear una clase llamada Círculo la cual contiene:

a. Dos variables de instancia privada: radio (de tipo double) y color (de tipo String), con valor por defecto de 12.5 y "azul", respectivamente.

```
public class Círculo {  
    private double radio= 12.5;  
    private String color="azul";  
}
```

b. Dos constructores sobrecargados (overloaded): un constructor predeterminado o por default sin argumentos y otro constructor que tiene un argumento de input double para el valor del radio.

```
public Círculo(){}; //constructor sin argumentos  
  
public Círculo(double radio){ //constructor con argumento  
    this.radio=radio;  
};
```

c. Dos métodos públicos: getRadio() y getArea(), que devuelven el radio y el área de esta instancia, respectivamente. Para calcular el área de un círculo, deberás seguir la siguiente fórmula:  $\text{area} = \text{radius} * \text{radius} * \text{Math.PI}$

```
public double getRadio(){  
    return radio;  
}  
  
public double getArea(){  
    return radio * radio * Math.PI;  
}  
  
}
```

Todo el código:

```
7 public class Círculo {  
8     private double radio= 12.5;  
9     private String color="azul";  
10  
11     public Círculo(){}; //constructor sin argumentos  
12  
13     public Círculo(double radio){ //constructor con argumento  
14         this.radio=radio;  
15     };  
16  
17     public double getRadio(){  
18         return radio;  
19     }  
20  
21     public double getArea(){  
22         return radio * radio * Math.PI;  
23     }  
24  
25 }
```

2. Compilar y ejecutar la clase "Circulo.java". ¿Puedes ejecutar la clase "Circulo.java"? ¿Por qué?

La clase Circulo no se puede ejecutar sola porque no tiene un main. Básicamente, es solo un molde para crear objetos, pero no le dice a Java qué hacer. Para que funcione, hay que crear otra clase con un main que cree instancias de Circulo y las use.

3. Crear una clase CirculoTest, esta clase contendrá el método main() para poder visualizar los siguientes datos:

Utilizando un constructor por defecto para imprimir:

- a. El radio de un círculo.
- b. El área de un círculo.

Utilizando un constructor sobrecargado, donde radio es igual a 6.55, para imprimir:

- a. El radio de un círculo.
- b. El área de un círculo.

```
10 public class CirculoTest {
11     public static void main(String[] args) {
12         Circulo c1 = new Circulo();           // constructor sin argumentos
13         Circulo c2 = new Circulo(6.55);       // constructor con radio
14
15
16         System.out.println("Usando constructor por defecto:");
17         System.out.println("Radio de c1: " + c1.getRadio());
18         System.out.println("Area de c1: " + c1.getArea());
19
20
21         System.out.println("Usando constructor sobrecargado: ");
22         System.out.println("Radio de c2: " + c2.getRadio());
23         System.out.println("Area de c2: " + c2.getArea());
24     }
25 }
26
```

**¿Qué pasos en común debe realizar para ejecutar cada uno de los ejercicios?**

**Describir conceptos del paradigma orientada a objetos.**

La clase Circulo actúa como una plantilla que define atributos y métodos. Los objetos creados a partir de esta clase, como c1 y c2, son instancias con sus propios valores. El encapsulamiento se aplica al declarar los atributos como private, evitando su acceso directo y permitiendo la manipulación a través de métodos públicos como getRadio() y getArea(). El uso de constructores permite la inicialización de los objetos, con un constructor predeterminado que asigna valores por defecto y otro sobrecargado que permite definir un radio personalizado. Finalmente, los métodos de la clase permiten obtener información sobre cada objeto sin modificar sus valores internos, asegurando una correcta gestión de los datos dentro del programa.

#### **4. Ejecutar la clase “CirculoTest.java” y responde las siguientes preguntas:**

##### **a. ¿Qué es el constructor y para qué sirve?**

Un constructor en Java es un método especial que inicializa objetos de una clase al momento de su creación. No tiene tipo de retorno y su nombre coincide con el de la clase. Puede asignar valores por defecto o personalizados a los atributos, asegurando que el objeto tenga un estado válido desde el inicio.

##### **b. ¿Con qué método recupero el radio del círculo?**

Se utiliza el método `getRadio` que devuelve el valor del atributo de radio. Este método permite acceder al radio sin modificarlo, siguiendo el principio de encapsulamiento en la programación orientada a objetos.

##### **c. ¿Por qué los atributos son privados y los métodos son públicos?**

Los atributos son privados para proteger los datos y evitar modificaciones directas desde fuera de la clase, garantizando el principio de encapsulamiento. En cambio, los métodos son públicos para permitir el acceso controlado a los atributos a través de métodos como getters y setters, asegurando que los datos solo se modifiquen de manera segura y consistente.

##### **d. ¿Qué pasa si accedes directamente a los atributos del círculo desde el método `main()`?**

El compilador mostrará un error porque los atributos son privados y solo pueden ser accedidos dentro de la misma clase.

##### **e. ¿Qué pasa si modificas directamente el atributo área del círculo desde el método `main()`?**

El compilador mostrará un error porque los atributos son privados y solo pueden ser accedidos dentro de la misma clase.

##### **f. En la clase “CirculoTest.java”, ejecuta la siguiente instrucción:**

`System.out.println(c1.radio)`, donde `c1` es una instancia del objeto Círculo. ¿Qué ocurrió? Explique el mensaje de error.

El compilador mostrará un error porque el atributo `radio` es privado en la clase `Circulo`. Esto significa que no se puede acceder directamente desde otra clase.

##### **g. En la clase “CirculoTest.java”, ejecuta la siguiente instrucción: `c1.radio=42.0`, donde `c1` es una instancia del objeto Círculo. ¿Qué ocurrió? Explique el mensaje de error.**

El compilador mostrará un error porque el atributo `radio` es privado en la clase `Circulo`. Esto significa que no se puede acceder directamente desde otra clase. El acceso está restringido debido al principio de encapsulamiento en Java.

h. ¿Con qué método modificas el valor de un atributo de la clase “Circulo.java”?

Para modificar el valor de un atributo en la clase Circulo.java, se usa un método setter, que permite cambiar el valor de un atributo privado desde fuera de la clase.

i. Crear un objeto de tipo Círculo y modifica los valores por defecto.

```
7 public class Circulo {
8     private double radio= 12.5;
9     private String color="azul";
10
11     public Circulo(){}; //constructor sin argumentos
12
13     public Circulo(double radio){ //constructor con argumento
14         this.radio=radio;
15     }
16
17     public double getRadio(){
18         return radio;
19     }
20
21     public double getArea(){
22         return radio * radio * Math.PI;
23     }
24
25     public void setRadio(double radio) {
26         this.radio = radio;
27     }
28
29     public void setColor(String color) {
30         this.color = color;
31     }
32
33     public String getColor() {
34         return color;
35     }
36
37 }
```

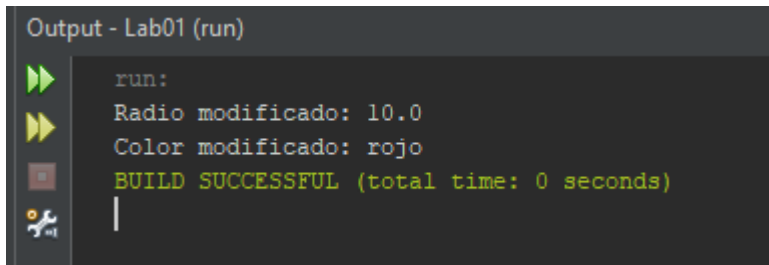
```
public class CirculoTest {
    public static void main(String[] args) {

        //objeto tipo circulo
        Circulo cl = new Circulo();

        // Modificar sus valores
        cl.setRadio(10.0);
        cl.setColor("rojo");

        // Mostrar los valores actualizados
        System.out.println("Radio modificado: " + cl.getRadio());
        System.out.println("Color modificado: " + cl.getColor());

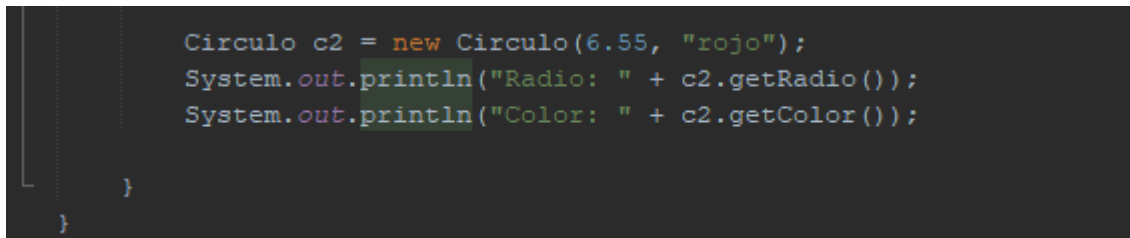
    }
}
```



```
Output - Lab01 (run)
run:
Radio modificado: 10.0
Color modificado: rojo
BUILD SUCCESSFUL (total time: 0 seconds)
```

j. ¿Para qué sirve la palabra reservada "this"? Modifique el constructor con argumentos (radio y color) y utiliza "this" y métodos setters en la clase "Circulo.java".

La palabra reservada "this" se usa en Java para referirse a la instancia actual de la clase. Se utiliza para diferenciar entre los atributos de la clase y los parámetros del constructor o métodos cuando tienen el mismo nombre.



```
Circulo c2 = new Circulo(6.55, "rojo");
System.out.println("Radio: " + c2.getRadio());
System.out.println("Color: " + c2.getColor());
```

k. ¿Para qué sirve el método toString()?

El método toString() sirve para representar un objeto como una cadena de texto. Se usa comúnmente para mostrar información sobre el estado de un objeto de manera legible. Si no se sobrescribe, se usa la implementación por defecto, que devuelve el nombre de la clase y un código hash. Al sobrescribir toString(), se puede personalizar la salida para mostrar valores específicos de los atributos de la clase.

l. Incluir el método toString() en la clase "Circulo.java".

```

13 public Circulo(double radio, String color) {
14     this.radio = radio;
15     this.color = color;
16
17 }
18
19 public double getRadio() {
20     return radio;
21 }
22
23 public void setRadio(double radio) {
24     this.radio = radio;
25 }
26
27 public String getColor() {
28     return color;
29 }
30
31 public void setColor(String color) {
32     this.color = color;
33 }
34
35 public double getArea() {
36     return radio * radio * Math.PI;
37 }
38
39 @Override
40 public String toString() {
41     return "Circulo{" + "radio=" + radio + ", color=" + color + '}';
42 }
43

```

```

public class CirculoTest {
    public static void main(String[] args) {

        Circulo c = new Circulo(5.0, "rojo");
        System.out.println(c);

    }
}

```

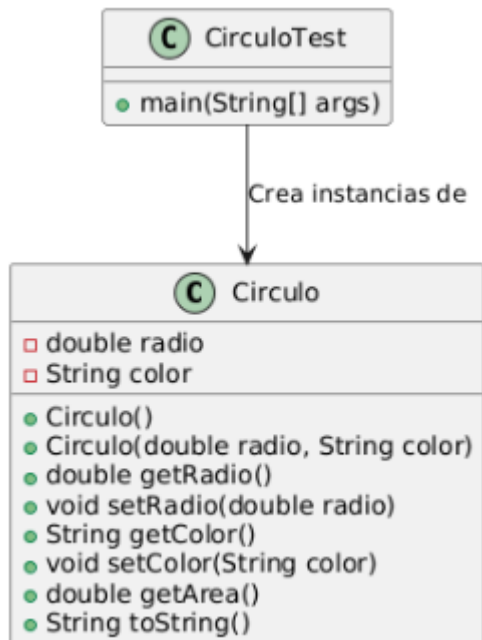
put - Lab01 (run)

```

run:
Circulo{radio=5.0, color=rojo}
BUILD SUCCESSFUL (total time: 0 seconds)
|

```

5. Trabaja en el diagrama de clase para la clase Circulo utilizando la herramienta online Visio o PlantUML (<https://plantuml.com/es/class-diagram>).

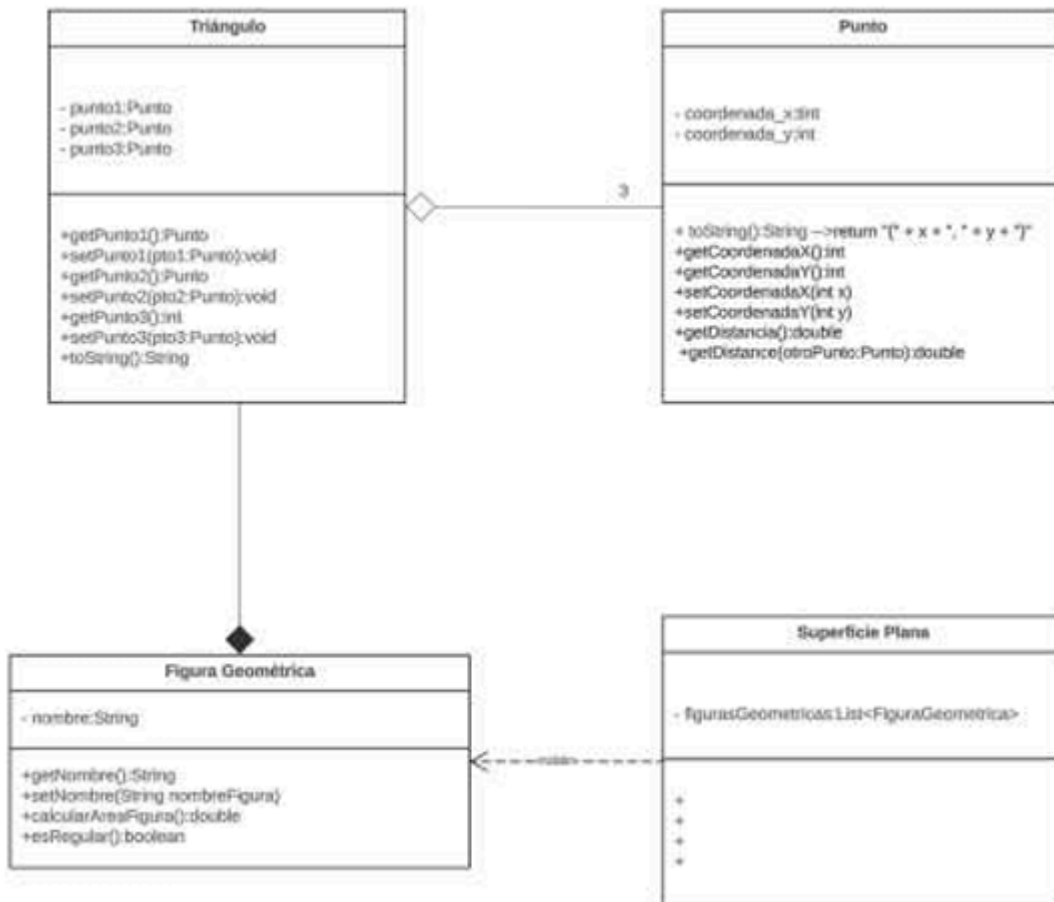


## Actividad 2.

1. Se tiene las siguientes clases:

- Crear la clase Punto.
- Crear la clase Triángulo.
- Crear la clase Figura Geométrica. Completar con 2 métodos abstractos: calcular el área y determinar si es una figura geométrica regular o no. Adicionalmente, incorporar métodos de accesos y manipulación.
- Crear la clase Superficie Plana. Esta clase tendrá un método que devolverá las áreas de cada figura geométrica. Completarlo.
- Crear una clase genérica para invocar a todas las clases mencionadas. Esta clase genérica debe contener el método **main()**.





## Clase Punto

```
7 public class Punto {
8     public int coordenada_x;
9     public int coordenada_y;
10
11     public Punto(int x, int y){
12         this.coordenada_x=x;
13         this.coordenada_y=y;
14     }
15
16     public int getCoordenada_x() {
17         return coordenada_x;
18     }
19
20     public int getCoordenada_y() {
21         return coordenada_y;
22     }
23
24     public void setCoordenada_x(int coordenada_x) {
25         this.coordenada_x = coordenada_x;
26     }
27
28     public void setCoordenada_y(int coordenada_y) {
29         this.coordenada_y = coordenada_y;
30     }
31
32     public double getDistancia(Punto otro){
33         return Math.sqrt(Math.pow(otro.coordenada_x - this.coordenada_x, 2) +
34                             Math.pow(otro.coordenada_y - this.coordenada_y, 2));
35     }
36
37     @Override
38     public String toString() {
39         return "Punto{" + "coordenada_x=" + coordenada_x + ", coordenada_y=" + coordenada_y + '}';
40     }
41 }
```

## Clase figuraGeometrica

```
abstract class FiguraGeometrica {
    private String nombre;

    public FiguraGeometrica(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public abstract double calcularAreaFigura();
    public abstract boolean esRegular();
}
```

## Clase Triangulo

```
6  class Triangulo extends FiguraGeometrica {
7      private Punto punto1, punto2, punto3;
8
9      public Triangulo(String nombre, Punto p1, Punto p2, Punto p3){
10         super(nombre);
11         this.punto1= p1;
12         this.punto2= p2;
13         this.punto3= p3;
14     }
15
16     public Punto getPunto1() {
17         return punto1;
18     }
19
20     public Punto getPunto2() {
21         return punto2;
22     }
23
24     public Punto getPunto3() {
25         return punto3;
26     }
27
28     public void setPunto1(Punto punto1) {
29         this.punto1 = punto1;
30     }
31
32     public void setPunto2(Punto punto2) {
33         this.punto2 = punto2;
34     }
```

```
36     public void setPunto3(Punto punto3) {
37         this.punto3 = punto3;
38     }
39
40     @Override
41     public double calcularAreaFigura() {
42         double a = punto1.getDistancia(punto2);
43         double b = punto2.getDistancia(punto3);
44         double c = punto3.getDistancia(punto1);
45         double s = (a + b + c) / 2;
46         return Math.sqrt(s * (s - a) * (s - b) * (s - c));
47     }
48
49     @Override
50     public boolean esRegular() {
51         double a = punto1.getDistancia(punto2);
52         double b = punto2.getDistancia(punto3);
53         double c = punto3.getDistancia(punto1);
54         return a == b && b == c;
55     }
56
57 }
```

2. Para la clase Punto considerar:

- El método toString() devolverá lo siguiente: "El punto tiene las siguientes coordenadas: " + x + "," + y .
- Un constructor sin parámetros y otro con parámetros (coordenaX, coordenaY).
- Incluir todos sus métodos de acceso y manipulación.
- La clase Punto tiene un método calcular distancia que se sobrescriben, el que no cuenta con parámetros retornará el resultado del otro método con parámetro.

```
1  public class Punto {
2      private int coordenada_x;
3      private int coordenada_y;
4
5      public Punto() {}
6
7      public Punto(int x, int y) {
8          this.coordenada_x = x;
9          this.coordenada_y = y;
10     }
11
12     public int getCoordenada_x() {
13         return coordenada_x;
14     }
15
16     public int getCoordenada_y() {
17         return coordenada_y;
18     }
19
20     public void setCoordenada_x(int coordenada_x) {
21         this.coordenada_x = coordenada_x;
22     }
23
24     public void setCoordenada_y(int coordenada_y) {
25         this.coordenada_y = coordenada_y;
26     }
27
28     public double calcularDistancia(Punto otro) {
29         return Math.sqrt(Math.pow(otro.coordenada_x - this.coordenada_x, 2) +
30                            Math.pow(otro.coordenada_y - this.coordenada_y, 2));
31     }
32
33     public double calcularDistancia() {
34         return calcularDistancia(new Punto(0, 0));
35     }
36
37     @Override
38     public String toString() {
39         return "El punto tiene las siguientes coordenadas: " + coordenada_x + "," + coordenada_y;
40     }
41 }
```

3. Para la clase Triángulo considerar:

- a. El método toString() devolverá lo siguiente: "Triángulo: " + getNombre() + " tiene 3 Puntos: "+ getPunto1().toString() + "," + getPunto2().toString()+ "," + getPunto3().toString();
- b. Un constructor sin parámetros y otro con parámetros pero éste debe invocar al constructor de la superclase.
- c. Incluir todos sus métodos de acceso y manipulación.
- d. Sobreescibir los métodos heredados de la clase Figura Geométrica.

```
1 public class Triangulo extends FiguraGeometrica {
2     private Punto punto1, punto2, punto3;
3
4     public Triangulo() {
5         super("Triángulo");
6         this.punto1 = new Punto();
7         this.punto2 = new Punto();
8         this.punto3 = new Punto();
9     }
10
11     public Triangulo(String nombre, Punto p1, Punto p2, Punto p3) {
12         super(nombre);
13         this.punto1 = p1;
14         this.punto2 = p2;
15         this.punto3 = p3;
16     }
17
18     public Punto getPunto1() {
19         return punto1;
20     }
21
22     public void setPunto1(Punto punto1) {
23         this.punto1 = punto1;
24     }
25
26     public Punto getPunto2() {
27         return punto2;
28     }
29
30     public void setPunto2(Punto punto2) {
31         this.punto2 = punto2;
32     }
```

```

34 public Punto getPunto3() {
35     return punto3;
36 }
37
38 public void setPunto3(Punto punto3) {
39     this.punto3 = punto3;
40 }
41
42
43 @Override
44 public double calcularAreaFigura() {
45     double a = punto1.calcularDistancia(punto2);
46     double b = punto2.calcularDistancia(punto3);
47     double c = punto3.calcularDistancia(punto1);
48
49     double s = (a + b + c) / 2;
50     return Math.sqrt(s * (s - a) * (s - b) * (s - c));
51 }
52
53 @Override
54 public boolean esRegular() {
55     double a = punto1.calcularDistancia(punto2);
56     double b = punto2.calcularDistancia(punto3);
57     double c = punto3.calcularDistancia(punto1);
58
59     return a == b && b == c;
60 }
61
62 @Override
63 public String toString() {
64     return "Triángulo: " + getNombre() + " tiene 3 Puntos: "
65         + getPunto1().toString() + ", "
66         + getPunto2().toString() + ", "
67         + getPunto3().toString();
68 }
69 }

```

4. Identificar las clases que heredan, ¿qué atributos y métodos heredan?

La clase Triangulo hereda de la clase abstracta FiguraGeometrica, lo que significa que hereda el atributo nombre (que almacena el nombre de la figura) y los métodos getNombre() y setNombre(), además de los métodos abstractos calcularAreaFigura() y esRegular(), que deben ser implementados obligatoriamente en Triangulo. La clase Punto no hereda de ninguna otra, pero se utiliza dentro de Triangulo, estableciendo una relación de composición, ya que un triángulo está compuesto por tres puntos.

5. Identificar las clases que conforman una composición.

En el diagrama, la composición se da entre las clases Triangulo y Punto. La composición indica que un Triangulo está compuesto por tres objetos de tipo Punto, y si el Triangulo deja de existir, los Punto también desaparecen.

6. ¿Qué es una superclase y una subclase?

Una superclase es una clase que sirve como base para otras clases, es decir, una clase padre de la cual otras clases pueden heredar atributos y métodos. Por otro lado, una subclase es una clase que hereda de otra, es decir, una clase hija que puede usar y sobrescribir los métodos de la superclase.

7. ¿Por qué usamos abstract? ¿Se puede dejar de heredar un método de una clase abstracta?

Se usa abstract en Java porque permite crear clases que sirven solo como base y no se pueden instanciar directamente. En mi código, FiguraGeometrica es abstracta porque define métodos como calcularAreaFigura(), que obligan a las subclases, como Triangulo, a implementarlos. No se puede dejar de heredar un método abstracto a menos que la subclase también sea abstracta, lo que significa que la implementación se pospone para más adelante. Esto me ayuda a organizar mejor el código y asegurarme de que todas las figuras geométricas tengan su propio cálculo de área sin repetir código.

8. ¿Qué anotación utilizo para sobrescribir métodos?

Se utiliza la anotación @Override para sobrescribir métodos en Java. Esta anotación indica al compilador que un método en una subclase está reemplazando un método de la superclase o de una interfaz. En el código, se utiliza en la clase Triangulo para redefinir los métodos calcularAreaFigura(), esRegular() y toString(), asegurando que implementen correctamente la funcionalidad específica de la clase sin errores.

9. Los atributos de la clase Figura Geométrica conviértalas en protected. ¿En qué condición convierte a los atributos? ¿Es posible acceder a los atributos protegidos sin utilizar una invocación a super() o sin método get?.

Si convertimos los atributos de la clase FiguraGeometrica en protected, significa que podrán ser accedidos directamente desde las subclases, como Triangulo, pero seguirán sin ser accesibles desde otras clases externas. En esta condición, los atributos dejan de ser privados y permiten un acceso más flexible dentro de la jerarquía de herencia. Además, sí es posible acceder a los atributos protegidos sin utilizar super() o un método get(), ya que una subclase puede referenciar directamente los atributos protected heredados sin necesidad de invocaciones adicionales.

10. ¿Cómo aplicarías polimorfismo?

El polimorfismo se aplicaría en este caso a través de la clase FiguraGeometrica, que define métodos abstractos como calcularAreaFigura() y esRegular(), los cuales son sobrescritos en la clase Triangulo con implementaciones específicas. Esto permite que, al trabajar con una referencia de tipo FiguraGeometrica, se pueda llamar a estos métodos y obtener el comportamiento correspondiente según la subclase que se esté utilizando. De esta manera, si en el futuro se crean otras figuras geométricas como Cuadrado o Círculo, todas podrán implementar sus propias versiones de estos métodos sin necesidad de modificar el código que las usa.

