**Documentación del Proyecto "Figuras"**

# Introducción

El proyecto "Figuras" tiene como objetivo ejemplificar los conceptos fundamentales de la Programación Orientada a Objetos (POO) a través de una jerarquía de clases que representan distintas formas geométricas. Este enfoque permite demostrar la utilización de herencia, encapsulamiento y polimorfismo en un contexto práctico. A continuación, se detallan las clases y su funcionalidad.

# Superclase: Formas

La clase `Formas` es la superclase que define propiedades y métodos comunes a todas las formas geométricas. Esta clase encapsula el atributo `color` y provee métodos para dibujar y establecer el color de una forma.

## Código:

public class Formas {  
 private String color;  
  
 public Formas() {  
 // Constructor por defecto  
 }  
  
 public void dibujar() {  
 System.out.println("Dibujar forma");  
 }  
  
 public void establecerColor(String color) {  
 this.color = color;  
 }  
  
 public String obtenerColor() {  
 return color;  
 }  
}

## Propiedades:

- `color`: Define el color de la forma. Encapsulamos esta propiedad para que sólo pueda ser modificada mediante métodos específicos.

## Métodos:

- `dibujar()`: Este método es genérico y se espera que sea redefinido en las subclases. La implementación actual simplemente imprime "Dibujar forma". Es decir, proporciona una acción básica que puede ser específica en cada subclase.  
- `establecerColor(String color)`: Este método permite establecer el color de la forma, facilitando la personalización de cada instancia.  
- `obtenerColor()`: Este método devuelve el color actual de la forma, permitiendo así acceder a esta propiedad de manera controlada.

# Clase Circulo

La clase `Circulo` hereda de `Formas` y añade la propiedad específica `radio`. Además, redefine el método `dibujar()` e introduce el método `calcularRadio()`.

## Código:

public class Circulo extends Formas {  
 private double radio;  
  
 public Circulo(double radio) {  
 this.radio = radio;  
 }  
  
 public void dibujar() {  
 System.out.println("Círculo");  
 }  
  
 public double calcularRadio() {  
 return radio;  
 }  
}

## Propiedades:

- `radio`: Define el radio del círculo. Esta propiedad es específica de la clase `Circulo` y no se encuentra en la superclase `Formas`.

## Métodos:

- `dibujar()`: Este método redefine el método `dibujar()` de la superclase para que imprima "Círculo", indicando que se está dibujando un círculo. En otras palabras, proporciona una implementación específica para esta forma.  
- `calcularRadio()`: Este método devuelve el radio del círculo. No realiza cálculos adicionales ya que el radio se establece al crear el objeto. De este modo, se asegura que el valor del radio sea accesible cuando sea necesario.

# Clase Linea

La clase `Linea` también hereda de `Formas` y añade la propiedad `largo`. Redefine el método `dibujar()` e introduce el método `obtenerLargo()`.

## Código:

public class Linea extends Formas {  
 private double largo;  
  
 public Linea(double largo) {  
 this.largo = largo;  
 }  
  
 public void dibujar() {  
 System.out.println("Línea");  
 }  
  
 public double obtenerLargo() {  
 return largo;  
 }  
}

## Propiedades:

- `largo`: Define el largo de la línea. Esta propiedad es específica de la clase `Linea`.

## Métodos:

- `dibujar()`: Este método redefine el método `dibujar()` de la superclase para que imprima "Línea", indicando que se está dibujando una línea. Así, proporciona una implementación específica para esta forma geométrica.  
- `obtenerLargo()`: Este método devuelve el largo de la línea, permitiendo acceder a esta propiedad específica.

# Clase Triangulo

La clase `Triangulo` hereda de `Formas` y añade la propiedad `angulo`. Redefine el método `dibujar()` e introduce el método `calcularArea()`.

## Código:

public class Triangulo extends Formas {  
 private double angulo;  
  
 public Triangulo(double angulo) {  
 this.angulo = angulo;  
 }  
  
 public void dibujar() {  
 System.out.println("Triángulo");  
 }  
  
 public double calcularArea() {  
 return (angulo \* angulo) / 2;  
 }  
}

## Propiedades:

- `angulo`: Define el ángulo del triángulo. Esta propiedad es específica de la clase `Triangulo`.

## Métodos:

- `dibujar()`: Este método redefine el método `dibujar()` de la superclase para que imprima "Triángulo", indicando que se está dibujando un triángulo. Por lo tanto, ofrece una implementación específica para esta forma.  
- `calcularArea()`: Este método calcula el área del triángulo utilizando una fórmula simplificada `(angulo \* angulo) / 2`. Es importante notar que en un caso real, el cálculo del área dependería de la base y la altura, o del tipo de triángulo. Así, se proporciona una función específica para esta forma, aunque sea un ejemplo simplificado.

# Clase Cuadrado

La clase `Cuadrado` hereda de `Formas` y añade la propiedad `lado`. Redefine el método `dibujar()` e introduce el método `calcularArea()`.

## Código:

public class Cuadrado extends Formas {  
 private double lado;  
  
 public Cuadrado(double lado) {  
 this.lado = lado;  
 }  
  
 public void dibujar() {  
 System.out.println("Cuadrado");  
 }  
  
 public double calcularArea() {  
 return lado \* lado;  
 }  
}

## Propiedades:

- `lado`: Define el lado del cuadrado. Esta propiedad es específica de la clase `Cuadrado`.

## Métodos:

- `dibujar()`: Este método redefine el método `dibujar()` de la superclase para que imprima "Cuadrado", indicando que se está dibujando un cuadrado. De este modo, proporciona una implementación específica para esta forma geométrica.  
- `calcularArea()`: Este método calcula el área del cuadrado multiplicando el valor del lado por sí mismo. En consecuencia, se proporciona una función específica que permite calcular el área de esta forma particular.

# Conclusión

El diseño de este proyecto utiliza la herencia para definir una jerarquía clara de formas geométricas, donde cada subclase extiende las funcionalidades de la superclase `Formas`. Esta estructura permite reutilizar código y añadir características específicas a cada tipo de forma, demostrando así la flexibilidad y potencia de la Programación Orientada a Objetos. Además, cada subclase redefine métodos y añade propiedades específicas para cumplir con sus respectivas necesidades, ejemplificando el uso de herencia y polimorfismo en un entorno práctico.