Clases abstractas

Clases abstractas

- **Clase abstracta**. Es un híbrido entre una interface y una clase.
- Una clase abstracta puede tener declaraciones de métodos sin cuerpos (como una interface).
- También puede tener métodos con sus respectivos cuerpos (como una clase).
- Al igual que las interfaces, las clases abstractas no se pueden instanciar.
- Están diseñadas para ser extendidas por otras clases.
- Pueden tener constructores y métodos.

Sintaxis

```
// Clase abstracta
public abstract class nombre {
    // Método abstracto (las subclases deben implementarlo)
    public abstract tipo nombre1(parámetros);
    // Método no abstracto
    public tipo nombre2(parámetros) {
        código
    }
}
```

Ejemplo

- Programar clases para dibujar las siguientes figuras: círculos, rectángulos y triángulos.
- Los tres tipos de figuras tienen en común el concepto de área.
- Cada tipo de figura calcula el área de manera distinta.
- Cada tipo de figura puede tener un color diferente y se dibuja de forma distinta.
- Una solución es definir una clase abstracta Figure que defina campos para guardar el área y el color, así como un método para regresar el área y un método abstracto para dibujarse.

Ejemplo

- Cada subclase hereda las variables correspondiente al área y al color e implementa el método para dibujarse.
- La clase Circle se define mediante un punto que representa las coordenadas del centro y su radio.
- La clase Rectangle se define mediante un punto que representa las coordenadas de su esquina superior izquierda, su ancho y su alto.
- La clase Triangle se define mediante tres puntos que representan las coordenadas de sus vértices.
- Para dibujar se usa la clase externa DrawingPanel, disponible en la página del curso.

Clase Figure

```
public abstract class Figure {
    protected double area;
    protected final Color color;

    public Figure(Color color)        // Constructor
    {
        this.color = color;
    }

    public abstract void draw(Graphics2D g);        // Método abstracto
```

Clase Figure

```
    public double getArea()          // Método implementado
    {
        return area;
    }
}
```

Clase Circle

```
public class Circle extends Figure {
    private final int radius;
    private final Point center;

    public Circle(int x, int y, int radius, Color color)
    {
        super(color);   // Llama al constructor de Figure
        this.area = Math.PI * radius * radius;
        this.center = new Point(x, y);
        this.radius = radius;
    }
```

Clase Circle

```
    @Override
    public void draw(Graphics2D g)                    // Implementa draw
    {
        g.setPaint(color);
        g.fillOval(center.x - radius, center.y - radius, 2 * radius, 2 * radius);
    }
}
```

Clase Rectangle

```
public class Rectangle extends Figure {
    private final int width, height;
    private final Point corner;

    public Rectangle(int x, int y, int width, int height, Color color)
    {
        super(color);              // Llama al constructor de Figure
        this.area = width * height;
        this.width = width;
        this.height = height;
        this.corner = new Point(x, y);
    }
```

Clase Rectangle

```
    @Override
    public void draw(Graphics2D g)
    {
        g.setPaint(color);
        g.fillRect(corner.x, corner.y, width, height);
    }
}
```

Clase Triangle

```
public class Triangle extends Figure {
    private static int[] x, y;

    public Triangle(int x1, int y1, int x2, int y2, int x3, int y3, Color color)
    {
        super(color);
        x = new int[] {x1, x2, x3}; y = new int[] {y1, y2, y3};
        double a = Math.sqrt((x[1] - x[0]) * (x[1] - x[0]) + (y[1] - y[0]) * (y[1] - y[0]));
        double b = Math.sqrt((x[2] - x[1]) * (x[2] - x[1]) + (y[2] - y[1]) * (y[2] - y[1]));
        double c = Math.sqrt((x[2] - x[0]) * (x[2] - x[0]) + (y[2] - y[0]) * (y[2] - y[0]));
        double s = (a + b + c) / 2.0;
        this.area = Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }
```

Clase Triangle

```
    @Override
    public void draw(Graphics2D g)
    {
        g.setPaint(color);
        g.fillPolygon(x, y, x.length);
    }
}
```

Clase principal

```
DrawingPanel panel = new DrawingPanel(600, 400);
Graphics2D g = panel.getGraphics();

Figure c1 = new Circle(70, 90, 50, Color.red);
c1.draw(g);
printFigure(c1);

Figure r1 = new Rectangle(200, 50, 170, 100, Color.blue);
r1.draw(g);
printFigure(r1);
```

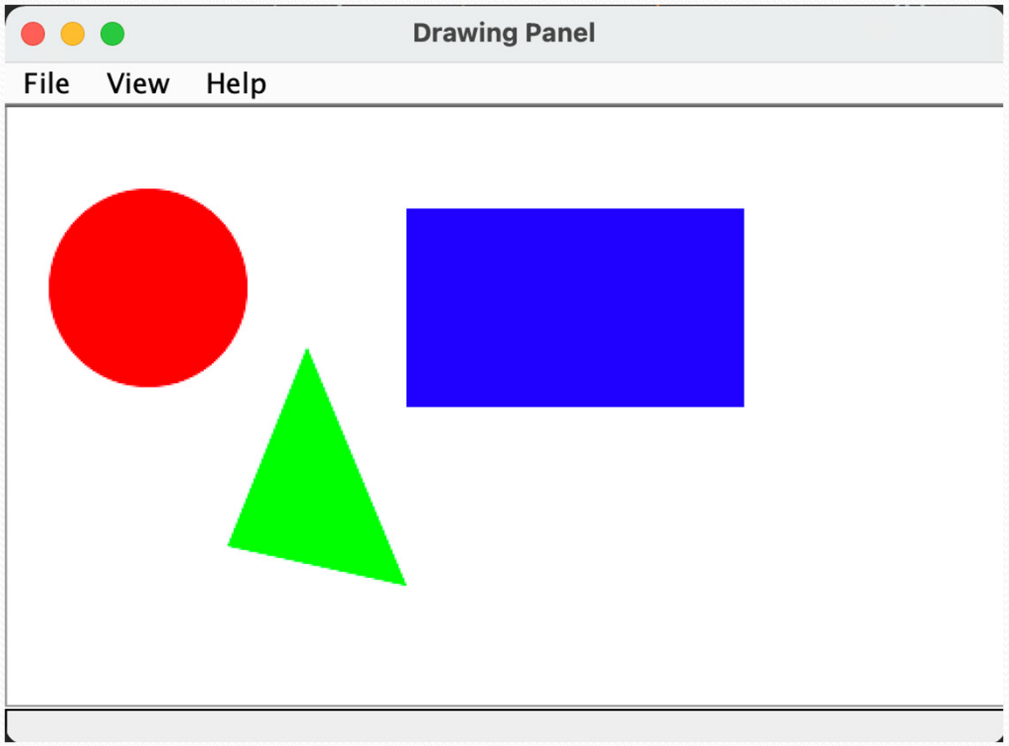
Clase principal

```
Figure t1 = new Triangle(150, 120, 200, 240, 110, 220, Color.green);
t1.draw(g);
printFigure(t1);
```

Método printFigure

```
public static void printFigura(Figure f)
{
    System.out.println("Clase: " + f.getClass());
    System.out.println("Area: " + f.getArea());
    System.out.println("Color: " + f.color);
    System.out.println();
}
```

Resultado



Clases abstractas e interfaces

- ¿Por qué existen tanto interfaces como clases abstractas en Java?
- Una clase abstracta puede hacer todo lo que puede hacer una interface y más.
- Entonces, ¿por qué alguien usaría una interface?
- Respuesta: Java tiene herencia única.
- Una clase puede extender solo una superclase.
- Una clase puede implementar muchas interfaces.
- Tener interfaces permite que una clase sea parte de una jerarquía (polimorfismo) sin agotar su relación de herencia.