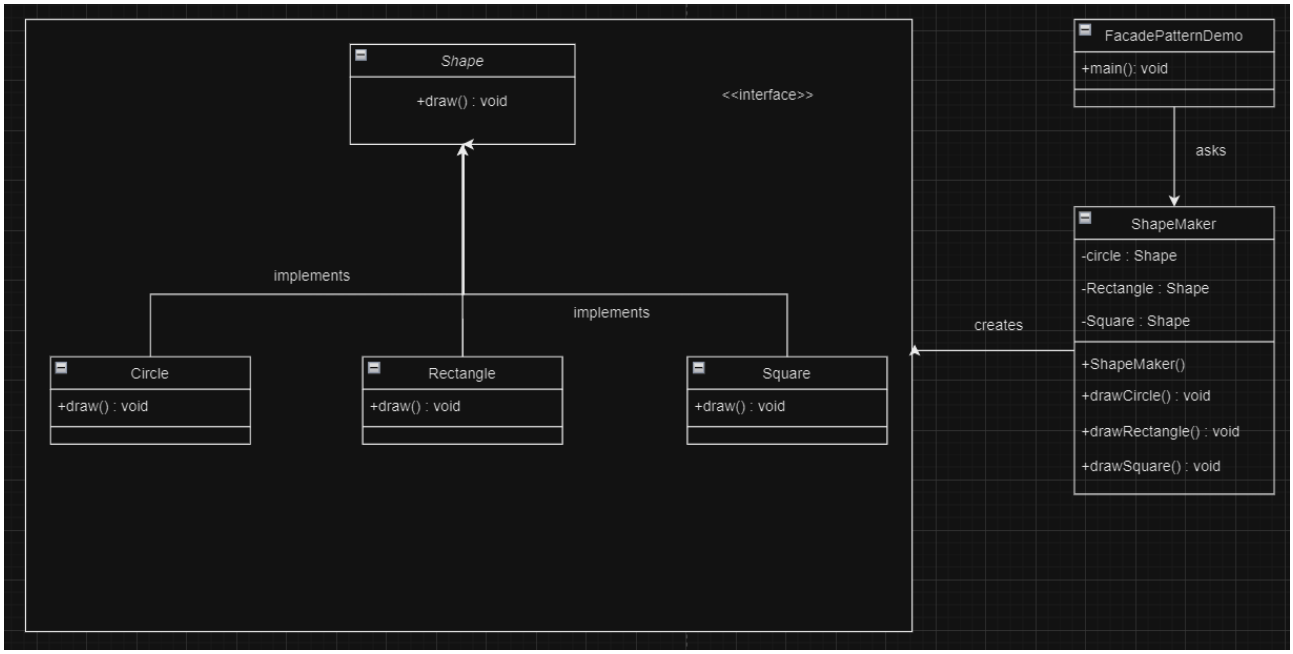
Decorator Design Pattern

Description:

The decorator design pattern is a structural pattern that allows behaviour to be added to individual objects, either statically or dynamically, without affecting the behaviour of other objects from the same class. It involves attaching new responsibilities to objects by wrapping them in one or more decorator objects. These decorators possess the same interface as the original object they decorate, enabling them to seamlessly extend or modify the behaviour of the object they decorate. Through composition, decorators provide a flexible alternative to subclassing for extending functionality, promoting code reusability and maintainability. This pattern is particularly useful in scenarios where the behaviour of objects needs to be modified or extended at runtime, such as adding additional features to graphical components or applying different filters to images.

Class diagram:



Implementation:

1. Shape.java

```
package Decorator;

public interface Shape {
	void draw();
}
```

2. Rectangle.java

```
package Decorator;

public class Rectangle implements Shape {
	@Override
```

```
	public void draw() {
		System.out.println("Shape: Rectangle");
	}
}
```

3. Circle.java

```
package Decorator;

public class Circle implements Shape {
	@Override
	public void draw() {
		System.out.println("Shape: Circle");
	}
}
```

4. ShapeDecorator.java

```
package Decorator;

public abstract class ShapeDecorator implements Shape
{
	protected Shape decoratedShape;
	public ShapeDecorator(Shape decoratedShape)
	{
		this.decoratedShape = decoratedShape;
	}
	public void draw()
	{
		decoratedShape.draw();
	}
}
```

5. RedShapeDecorator.java

```
package Decorator;

public class RedShapeDecorator extends ShapeDecorator
{
	public RedShapeDecorator(Shape decoratedShape)
	{
		super(decoratedShape);
	}
	@Override
	public void draw(){
		decoratedShape.draw();
		setRedBorder(decoratedShape);
	}
	private void setRedBorder(Shape decoratedShape){
		System.out.println("Border Color: Red");
	}
}
```

6. DecoratorPatternDemo.java

```
package Decorator;

public class DecoratorPatternDemo {
	public static void main(String[] args) {
		Shape circle = new Circle();
		Shape redCircle = new RedShapeDecorator(new Circle());
		Shape redRectangle = new RedShapeDecorator(new Rectangle());
		System.out.println("Circle with normal border");
		circle.draw();
		System.out.println("\nCircle of red border");
		redCircle.draw();
		System.out.println("\nRectangle of red border");
		redRectangle.draw();
	}
}
```

Output:

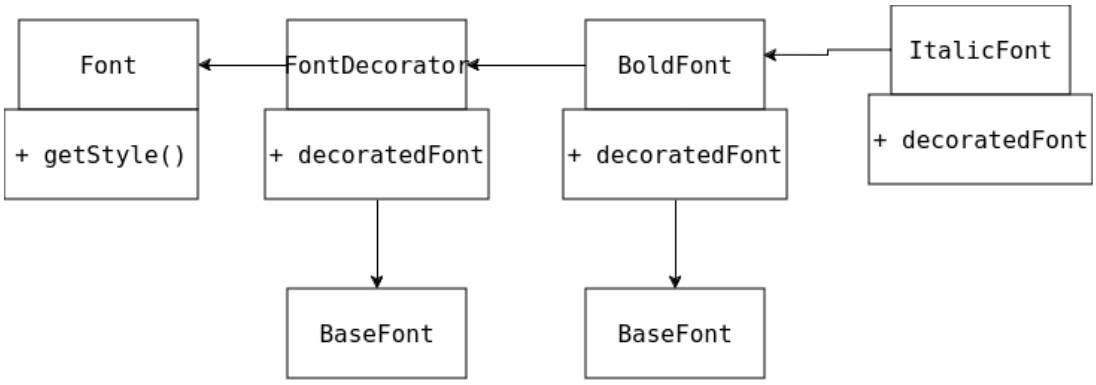
```
Circle with normal border
Shape: Circle

Circle of red border
Shape: Circle
Border Color: Red

Rectangle of red border
Shape: Rectangle
Border Color: Red
```

Decorator Design Pattern for Font System

Class diagram:



Implementation:

1. FontSystem.java

```
// Component Interface
interface Font {
    String getStyle();
}

// Concrete Component
class BaseFont implements Font {
    @Override
    public String getStyle() {
        return "Normal";
    }
}

// Decorator
abstract class FontDecorator implements Font {
    protected final Font decoratedFont;

    public FontDecorator(Font font) {
        this.decoratedFont = font;
    }

    @Override
    public String getStyle() {
        return decoratedFont.getStyle();
    }
}

// Concrete Decorators
class BoldFont extends FontDecorator {
    public BoldFont(Font font) {
        super(font);
```

```
    }

    @Override
    public String getStyle() {
        return "Bold, " + super.getStyle();
    }
}

class ItalicFont extends FontDecorator {
    public ItalicFont(Font font) {
        super(font);
    }

    @Override
    public String getStyle() {
        return "Italic, " + super.getStyle();
    }
}

class UnderlineFont extends FontDecorator {
    public UnderlineFont(Font font) {
        super(font);
    }

    @Override
    public String getStyle() {
        return "Underline, " + super.getStyle();
    }
}

// Client code
public class FontSystem {
    public static void main(String[] args) {
        // Create base font
        Font baseFont = new BaseFont();
        System.out.println("Base Font Style: " + baseFont.getStyle());

        // Apply decorators
        Font boldItalicFont = new BoldFont(new ItalicFont(baseFont));
        System.out.println("Bold Italic Font Style: " + boldItalicFont.getStyle());

        Font underlineFont = new UnderlineFont(baseFont);
        System.out.println("Underline Font Style: " + underlineFont.getStyle());

        Font boldUnderlineFont = new BoldFont(new UnderlineFont(baseFont));
        System.out.println("Bold Underline Font Style: " + boldUnderlineFont.getStyle());
    }
}
```

Output:

```
zoro-d-code@wado-ichimonji:~/Roger/College/Design_lab_Codes/Flyweight_design/2$ cd "/home/zoro-d-code/Roger/College/Design_lab_C
odes/Decorator_design/2/" && javac FontSystem.java && java FontSystem
Base Font Style: Normal
Bold Italic Font Style: Bold, Italic, Normal
Underline Font Style: Underline, Normal
Bold Underline Font Style: Bold, Underline, Normal
```