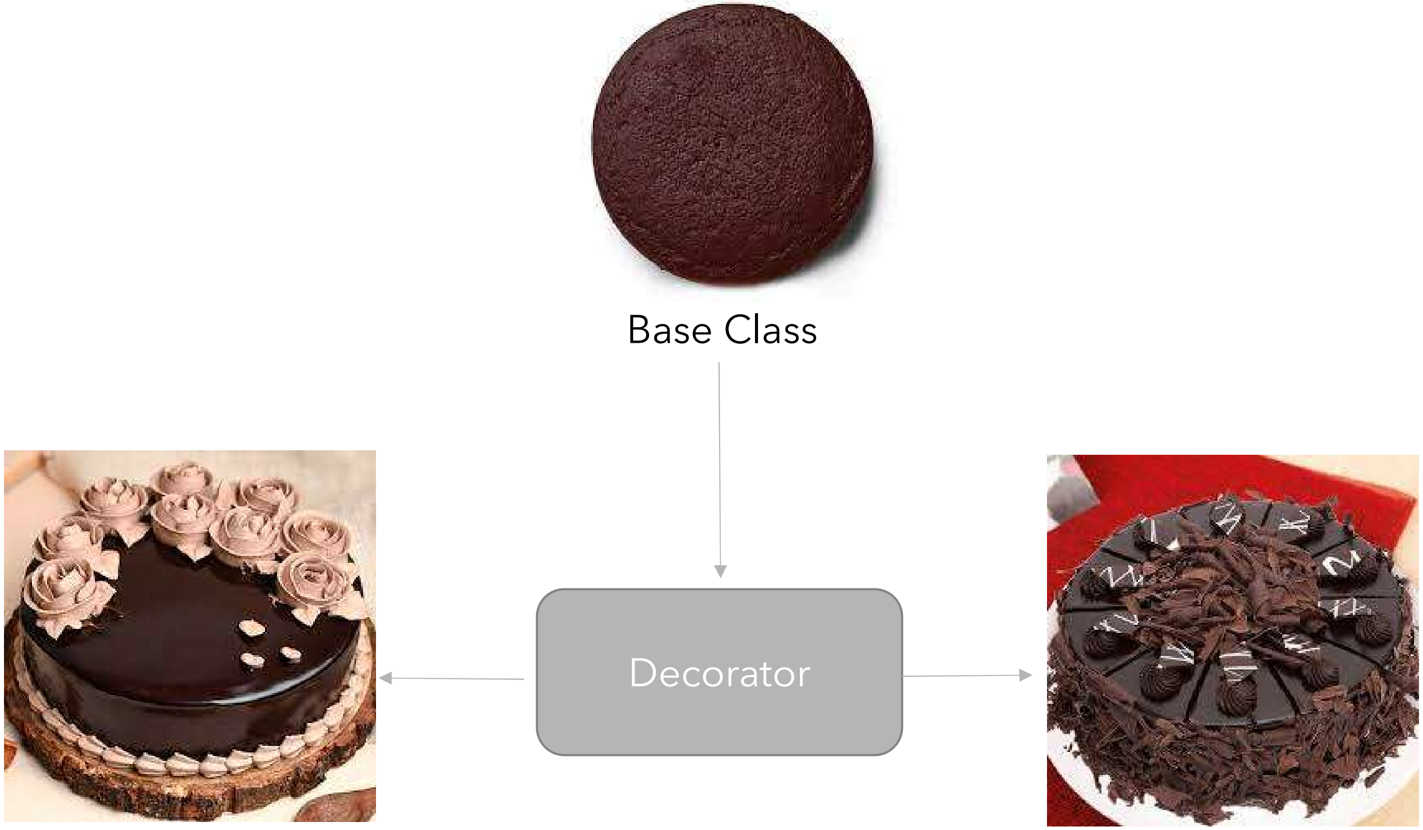
DECORATOR DESIGN PATTERN

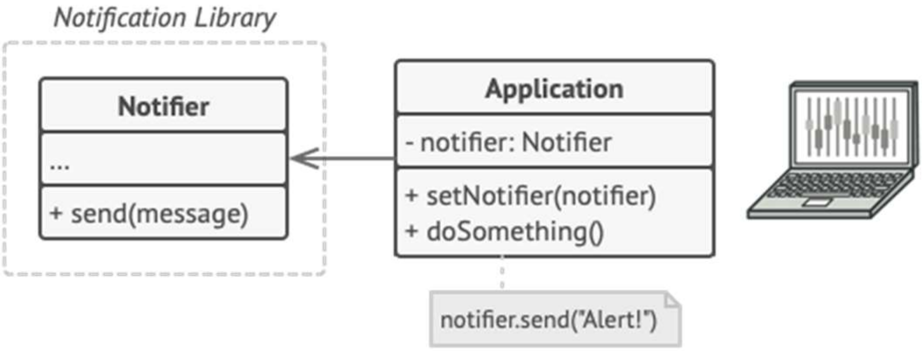


Base Class

Decorator

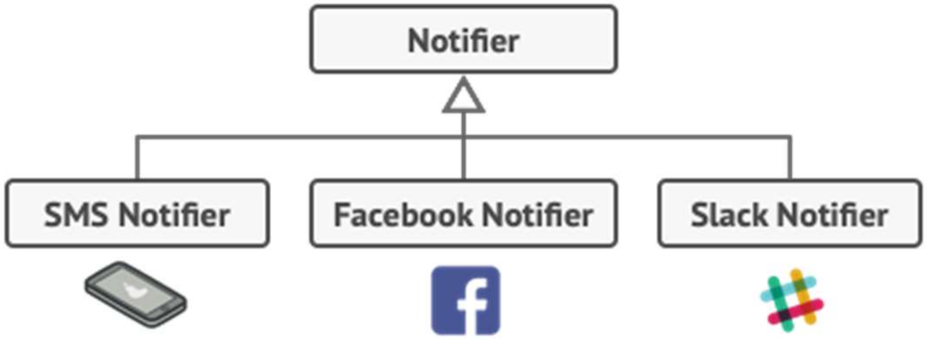
NOTIFICATION

- You created a Notifier class and has send method



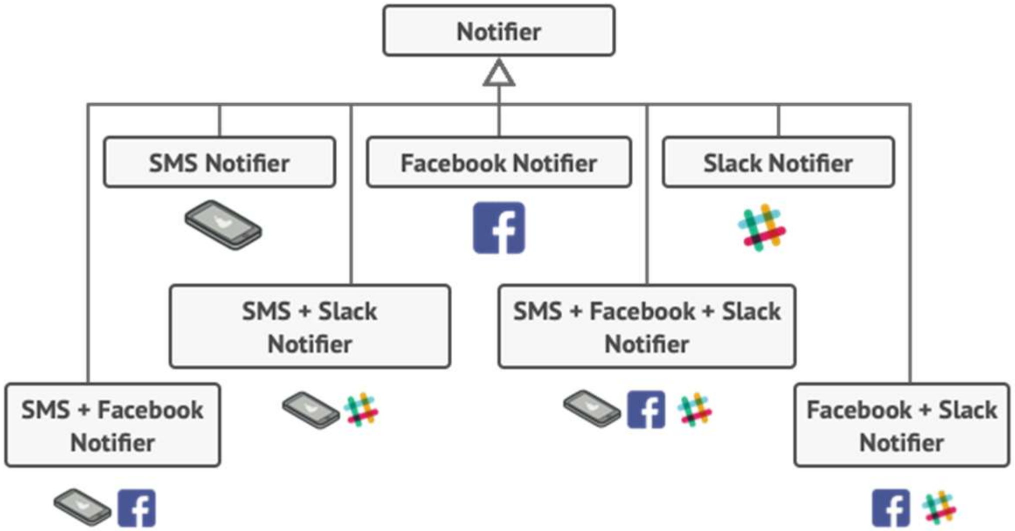
USER REQUIREMENT CHANGES

- User want to send notification SMS, Facebook, Slack
- You extend easily



NOTIFICATION BASED ON CONDITION

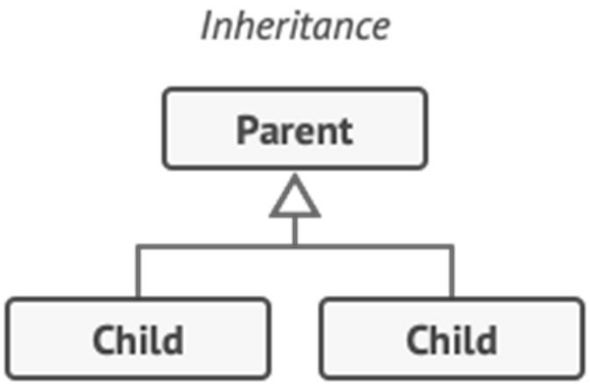
- Pay bill- SMS
- New launch – Facebook + slack
- Emergency = all available channel

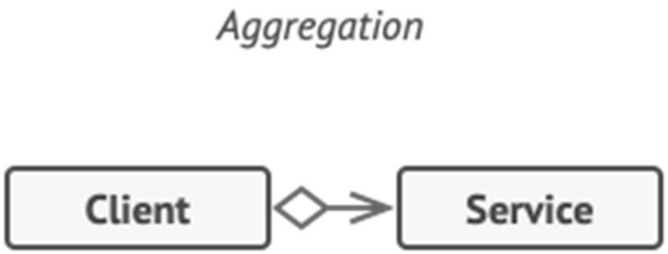


- Problem- You tried to address that problem by creating special subclasses which combined several notification methods within one class. However, it quickly became apparent that this approach would bloat the code immensely, not only the library code but the client code as well.

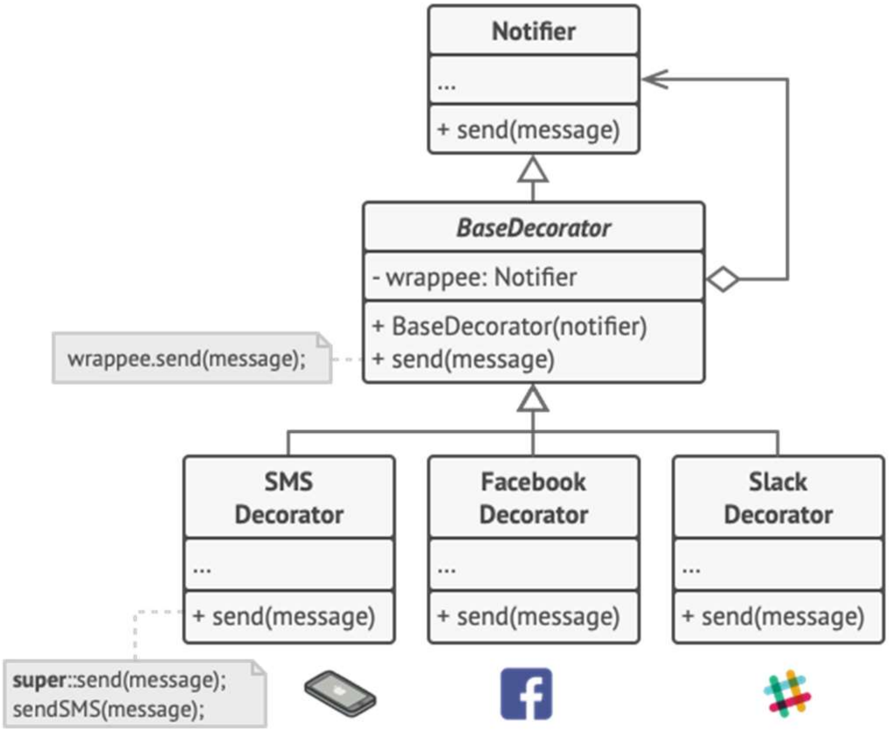
SOLUTION

- One of the ways to overcome these caveats is by using *Aggregation* or *Composition* instead of *Inheritance*.

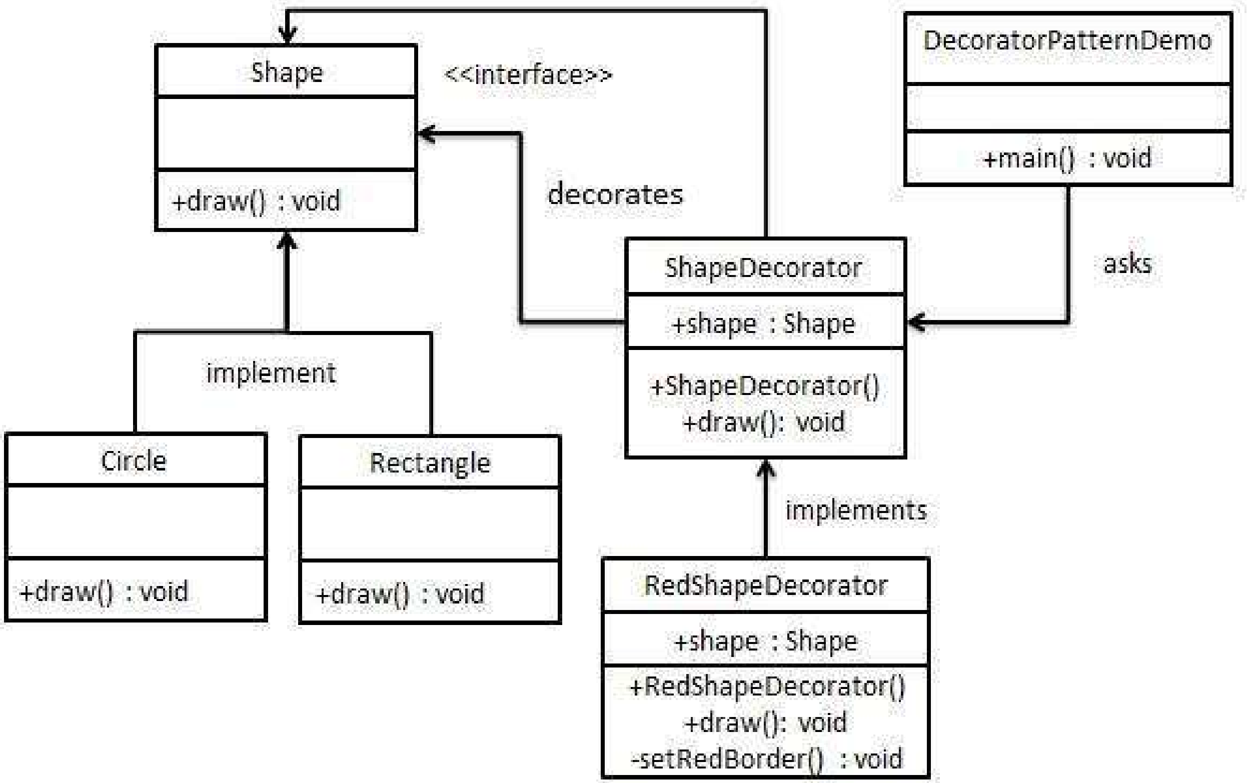




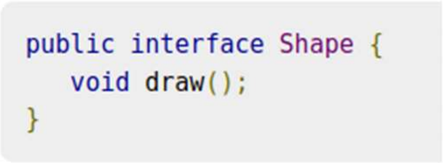
CLASS DIAGRAM



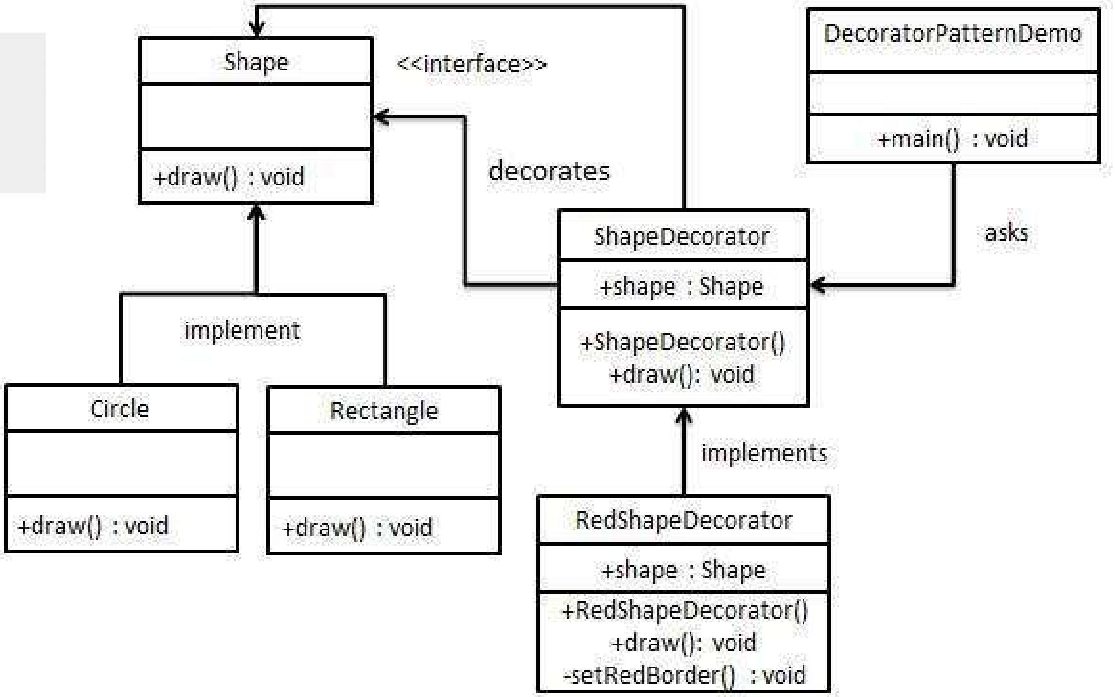
EXAMPLE WITH CODE



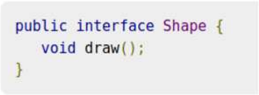
Shape.java



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

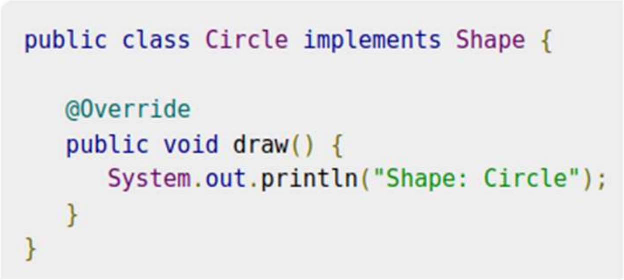


Shape.java

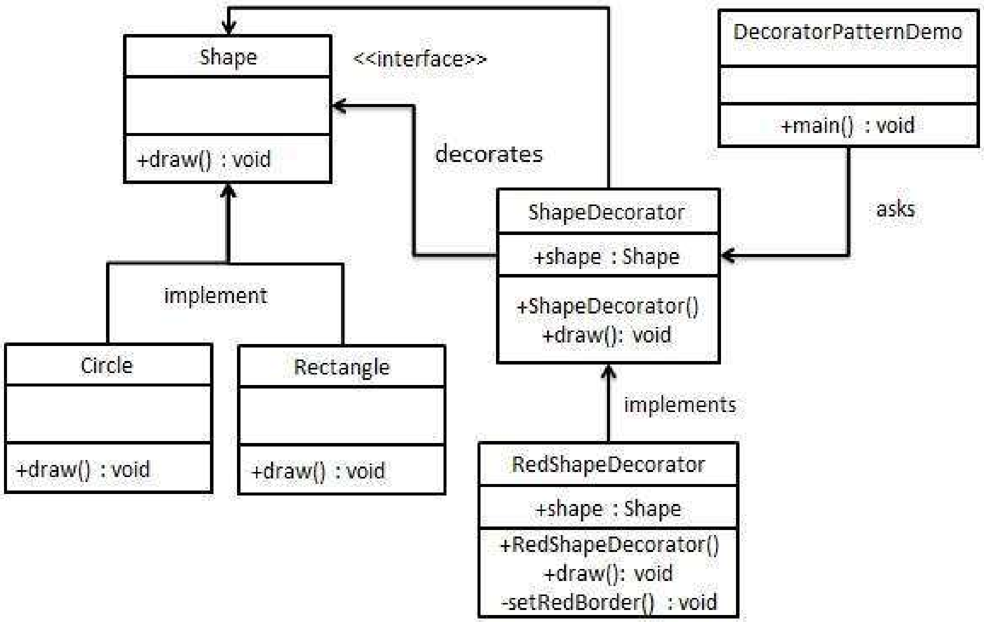


```
public interface Shape {
   void draw();
}
```

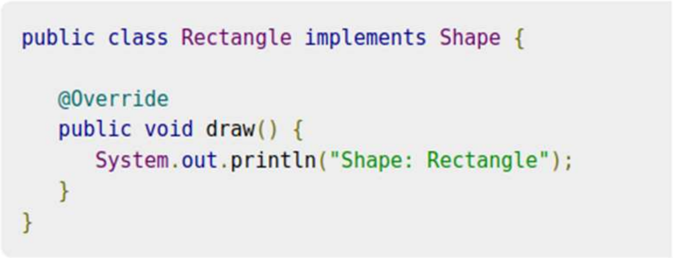
Circle.java



```
public class Circle implements Shape {

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



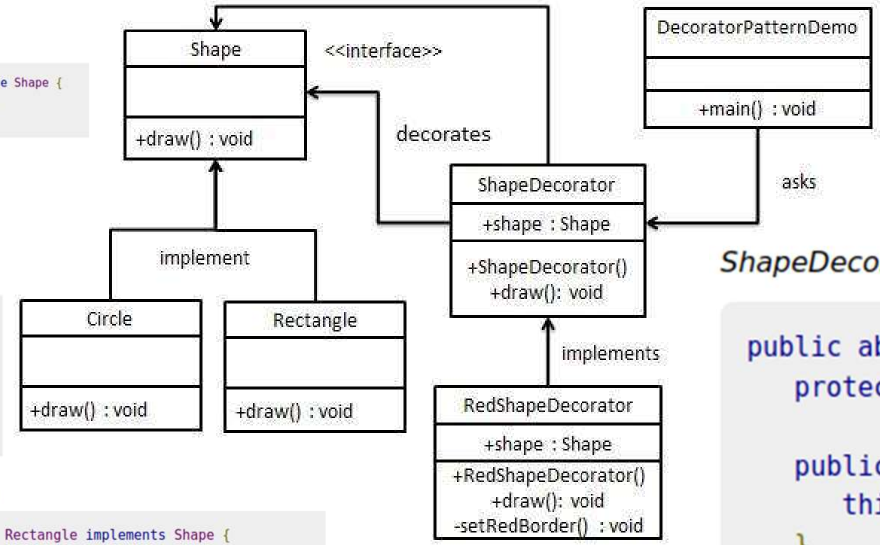
Rectangle.java



```
public class Rectangle implements Shape {

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

Shape.java

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



Circle.java

```java
public class Circle implements Shape {

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

Rectangle.java

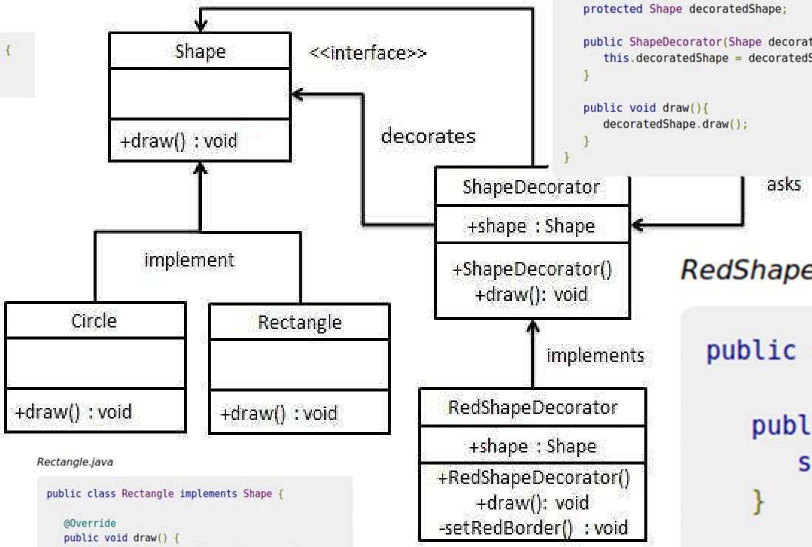
```java
public class Rectangle implements Shape {

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

ShapeDecorator.java

```java
public abstract class ShapeDecorator implements Shape {
   protected Shape decoratedShape;

   public ShapeDecorator(Shape decoratedShape){
      this.decoratedShape = decoratedShape;
   }

   public void draw(){
      decoratedShape.draw();
   }
}
```

Shape.java

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



ShapeDecorator.java

```java
public abstract class ShapeDecorator implements Shape {
   protected Shape decoratedShape;

   public ShapeDecorator(Shape decoratedShape){
      this.decoratedShape = decoratedShape;
   }

   public void draw(){
      decoratedShape.draw();
   }
}
```

Circle.java

```java
public class Circle implements Shape {

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

Rectangle.java

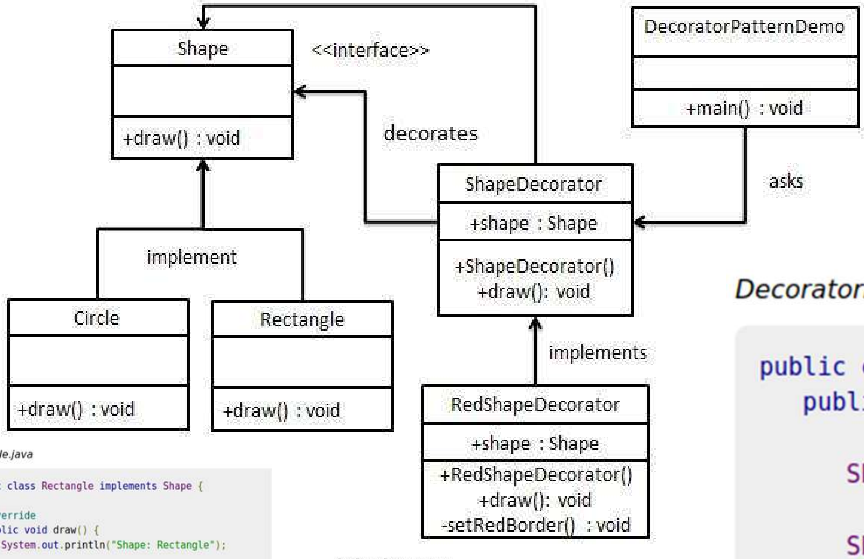
```java
public class Rectangle implements Shape {

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

RedShapeDecorator.java

```java
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");
   }
}
```

Shape.java

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



Circle.java

```java
public class Circle implements Shape {

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

Rectangle.java

```java
public class Rectangle implements Shape {

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

RedShapeDecorator.java

```java
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");
   }
}
```

DecoratorPatternDemo.java

```java
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();
   }
}
```