Decorator Design Pattern

Ian Chelaru

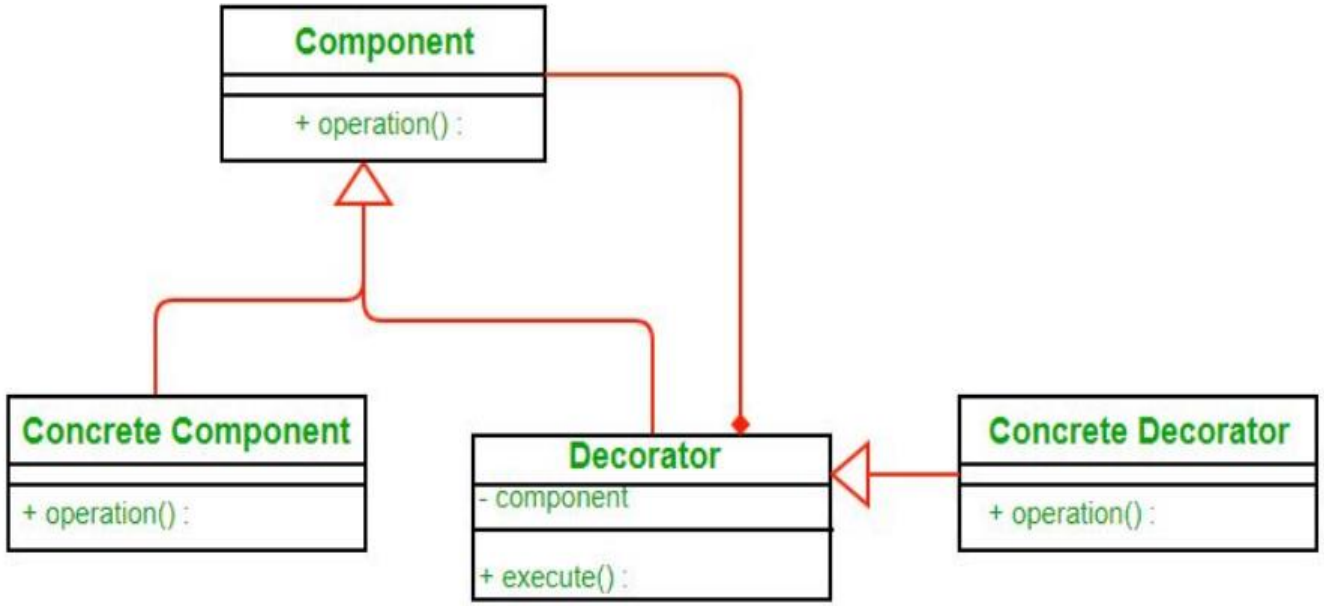
30432

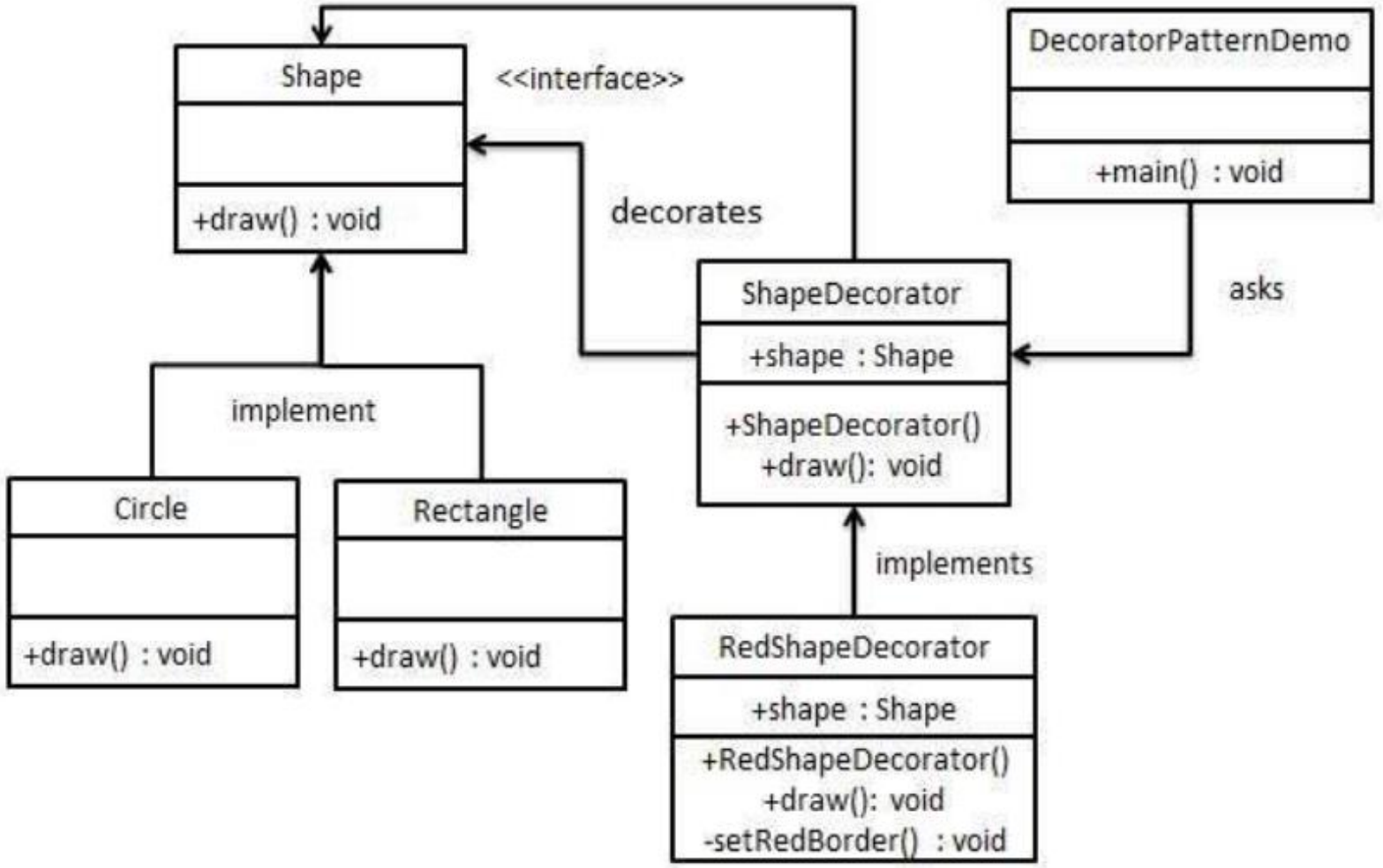
Introduction

- The decorator design pattern allows a user to add new functionality to an existing object without altering its structure.
- Acts as a wrapper to an existing class.
- It is a structural design pattern.

Class Diagram

- Each component can be used on its own or may be wrapped by a decorator.
- Each decorator has an instance variable that holds the reference to component it decorates.
- The ConcreteComponent is the object we are going to dynamically decorate.




Shape
<<interface>>
+draw() : void
decorates
DecoratorPatternDemo
+main() : void
asks
ShapeDecorator
+shape : Shape
+ShapeDecorator()
+draw(): void
implement
Circle
+draw() : void
Rectangle
+draw() : void
implements
RedShapeDecorator
+shape : Shape
+RedShapeDecorator()
+draw(): void
-setRedBorder() : void

Example

Implementation

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

```
public class Rectangle implements Shape {

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

```
public class Circle implements Shape {

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

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

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

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

```
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");
   }
}
```

```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();
   }
}
```

```
Circle with normal border
Shape: Circle

Circle of red border
Shape: Circle
Border Color: Red

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

Output

Advantages

- Used to extend (decorate) the functionality of a certain object at runtime.
- Is an alternative to subclassing.
- Offers a pay-as-you-go approach to adding responsibilities.

Disadvantages

Too many layers of the decorator chain starts to push the decorator pattern beyond its true intent.

Can complicate the process of instantiating the component.

It can be complicated to have decorators keep track of other decorators.

References

https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm

https://www.geeksforgeeks.org/the-decorator-pattern-set-2-introduction-and-design/