Motifs de conception - Motif de décorateur

Le modèle de décorateur permet à un utilisateur d'ajouter de nouvelles fonctionnalités à un objet existant sans altérer sa structure. Ce type de modèle de conception relève du modèle structurel car ce modèle agit comme un wrapper pour la classe existante.

Ce modèle crée une classe décoratrice qui enveloppe la classe d'origine et fournit des fonctionnalités supplémentaires en conservant la signature des méthodes de classe.

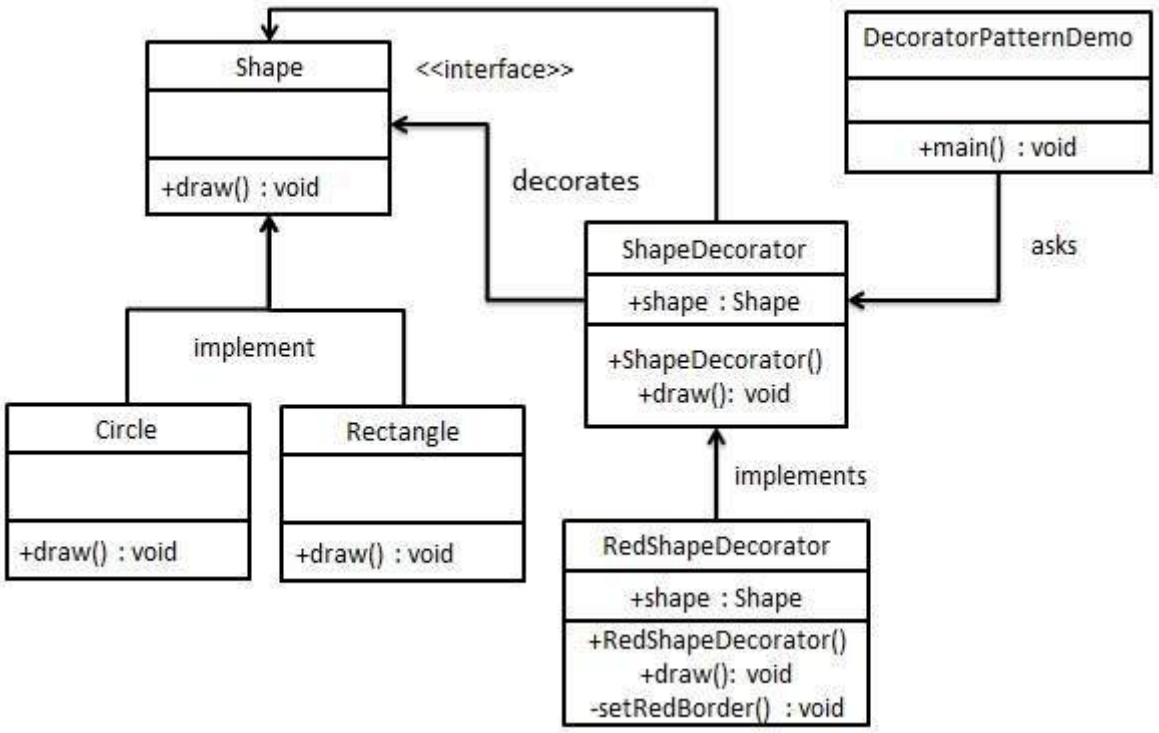
Nous montrons l'utilisation du motif décorateur via l'exemple suivant dans lequel nous allons décorer une forme avec une certaine couleur sans modifier la classe de forme.

la mise en oeuvre

Nous allons créer une interface *Shape* et des classes concrètes implémentant l' interface *Shape* . Nous allons ensuite créer une classe décoratrice abstraite *ShapeDecorator* implémentant l' interface *Shape* et ayant l' objet *Shape* comme variable d'instance.

RedShapeDecorator est une classe concrète implémentant *ShapeDecorator* .

DecoratorPatternDemo , notre classe de démonstration utilisera *RedShapeDecorator* pour décorer des objets *Shape* .



Étape 1

Créez une interface.

Shape.java

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

Étape 2

Créez des classes concrètes implémentant la même interface.

Rectangle.java

```
public class Rectangle implements Shape {

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

Circle.java

```
public class Circle implements Shape {

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

Étape 3

Créez une classe décorative abstraite implémentant l' interface *Shape* .

ShapeDecorator.java

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

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

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

Étape 4

Créez une classe de décorateur en béton étendant la classe *ShapeDecorator* .

RedShapeDecorator.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");
   }
}
```

Étape 5

Utilisez le *RedShapeDecorator* pour décorer des objets *Shape* .

DecoratorPatternDemo.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();
   }
}
```

Étape 6

Vérifiez la sortie.

```
Circle with normal border
Shape: Circle

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

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