Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 9 Padrões Estruturais - Decorator

Intenção

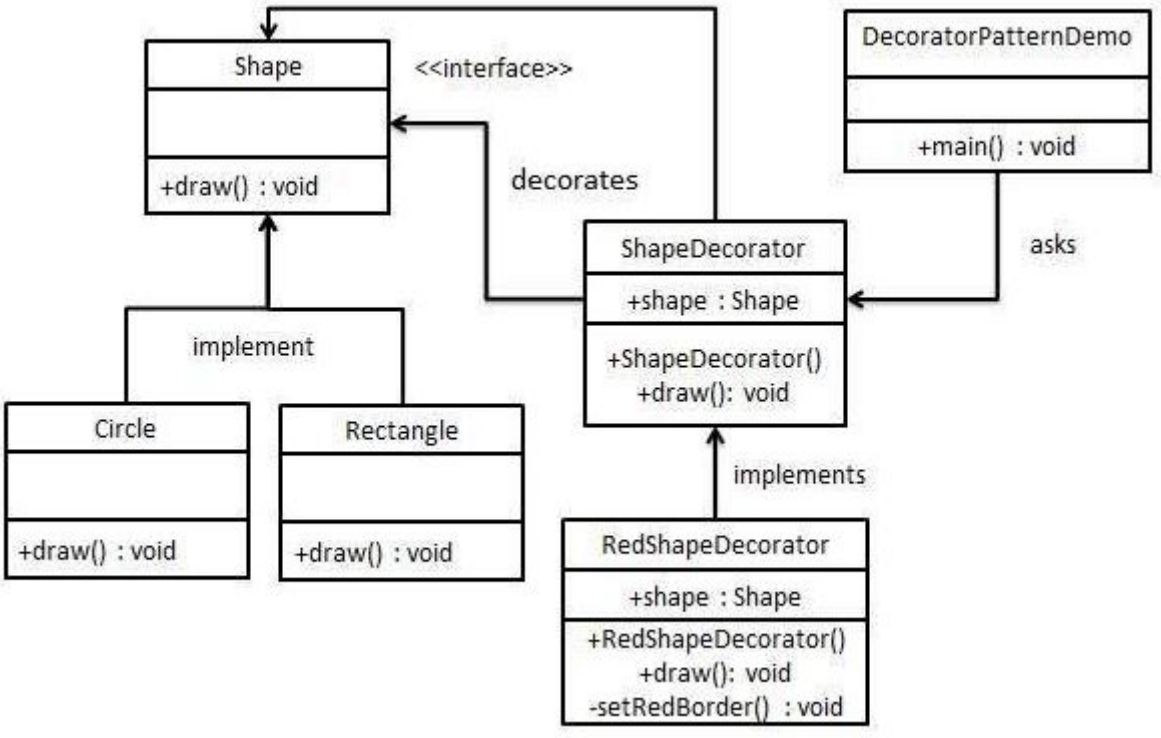
Anexar funcionalidades adicionais a um objeto dinamicamente. Provê uma alternativa flexível à herança como mecanismo de extensão.

Usar este padrão quando…

- Quiser adicionar funcionalidade dinamicamente e transparentemente;
- Quiser adicionar funcionalidade que pode depois ser desativada;
- Extensão por herança é impraticável (não disponível ou produziria uma explosão de subclasses, como no exemplo).

Vantagens e desvantagens

- Mais flexibilidade do que herança:
  - Podem ser adicionadas/removidas em tempo de execução;
  - Pode adicionar duas vezes a mesma funcionalidade.
- O decorador é diferente do componente:
  - A identidade do objeto não pode ser usada de forma confiável.
- Muitos objetos pequenos:
  - Um projeto que utiliza Decorator pode vir a ter muitos objetos pequenos e parecidos.



Passo 1
Crie uma interface.

Shape.java

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

Passo 2
Crie classes concretas implementando a mesma 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");
   }
}
```

Passo 3
Crie uma classe de decorador abstrata implementando a 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();
   }
}
```

Passo 4
Crie uma classe de decorador de concreto estendendo a 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");
   }
}
```

Passo 5
Use o RedShapeDecorator para decorar objetos 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();
   }
}
```

Passo 6

Teste sua implementação!