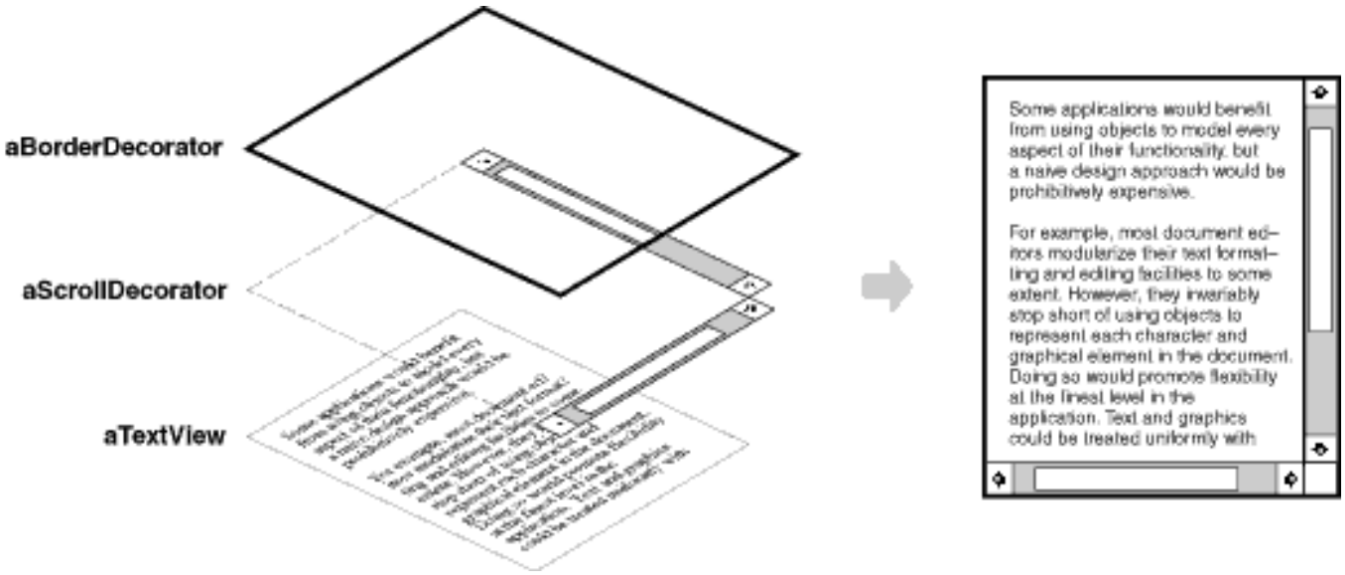
Decorator

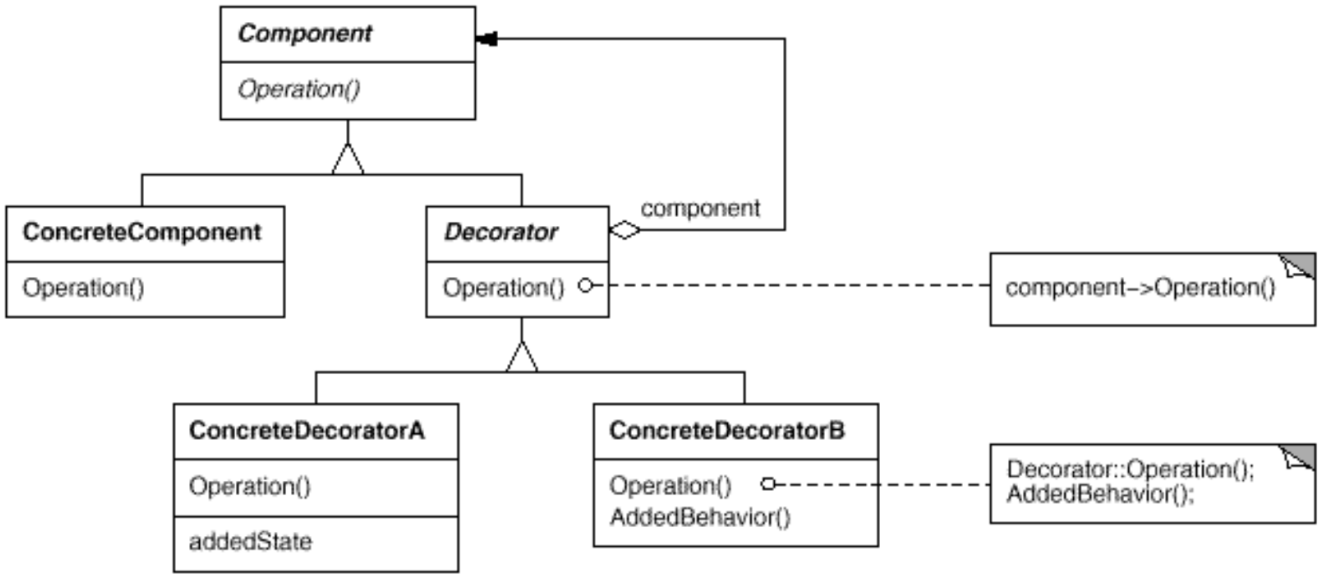
↓ INTENT ↓

```
Associare responsabilità aggiuntive a un oggetto in modo dinamico. I
decorator forniscono un'alternativa flessibile all'ereditarieta
```

↓ MOTIVATION ↓



↓ STRUCTURE ↓



↓ IMPLEMENTATION ↓

1. Omissione della classe di decoratore astratto.

Non c'è bisogno di definire una classe di decoratore astratto quando è necessario aggiungere solo una responsabilità.Questo è spesso il caso quando hai a che fare con una gerarchia di classi esistente piuttosto che progettarne una nuova.

Puoi unire la responsabilità del decoratore per inoltrare le richieste al componente in ConcreteDecorator.

2. Mantenere le classi dei componenti leggere.

Per garantire un'interfaccia conforme, componenti e decoratori devono discendere da una classe Component comune.

È importante:

- mantenere leggera questa classe comune, facendola concentrare sull'assomigliare ad un'interfaccia, non a memorizzare dati.
- la rappresentazione dei dati dovrebbe essere rinviata alle sottoclassi; altrimenti la complessità potrebbe rendere i decoratori troppo pesanti da usare in quantità.

↓ EXAMPLE ↓

Shape ↓

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

Circle ↓

```
public class Circle implements Shape{

    @Override
    public void draw() {
        System.out.println("CERCHIO");
    }

}
```

Rectangle ↓

```
public class Rectangle implements Shape {

    @Override
```

```
    public void draw() {
        System.out.println("RETTANGOLO");
    }

}
```

ShapeDecorator ↓

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

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

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

RedShapeDecorator ↓

```
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("BORDO ROSSO");
    }
}
```

Main ↓

```
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("Cerchio normale:");
        circle.draw();

        System.out.println("Cerchio rosso:");
        redCircle.draw();

        System.out.println("Rettangolo rosso:");
        redRectangle.draw();
    }
```