UNEB
Universidade do
Estado da Bahia

Decorator ou Wrapper

Fagner Pimentel
Adriano Veiga
Igor Eloi

Universidade Estadual da Bahia

11 de Fevereiro de 2011

Outline

Introdução

Propósito

- Agregar responsabilidades adicionais a um objeto dinamicamente.
- Classes decoradoras oferecem uma alternativa flexível ao uso de herança para estender uma funcionalidade.

Motivação

- Adicionar responsabilidades a um objeto, mas não à sua classe.
- Acontece, por exemplo, com criação de interfaces gráficas, quando se deseja acrescentar uma borda a um componente qualquer ou uma barra de rolagem a uma área de texto.
- Uma abordagem mais flexível é inserir o componente em outro objeto que adiciona a borda, um Decorator.

Aplicabilidade

- Adicionar responsabilidades a objetos individuais de forma dinâmica e transparente.
- Quando existem muitas variações de um objeto.
- Redução do número de subclasses.

Conseqüências

Conseqüências

- Fornece uma flexibilidade maior do que a herança estática.
- Evita a necessidade de colocar classes sobrecarregadas de recursos em uma posição mais alta da hierarquia.
- Simplifica a codificação permitindo que você desenvolva uma série de classes com funcionalidades específicas, em vez de codificar todo o comportamento no objeto.
- Aprimora a extensibilidade do objeto, pois as alterações são feitas codificando novas classes.

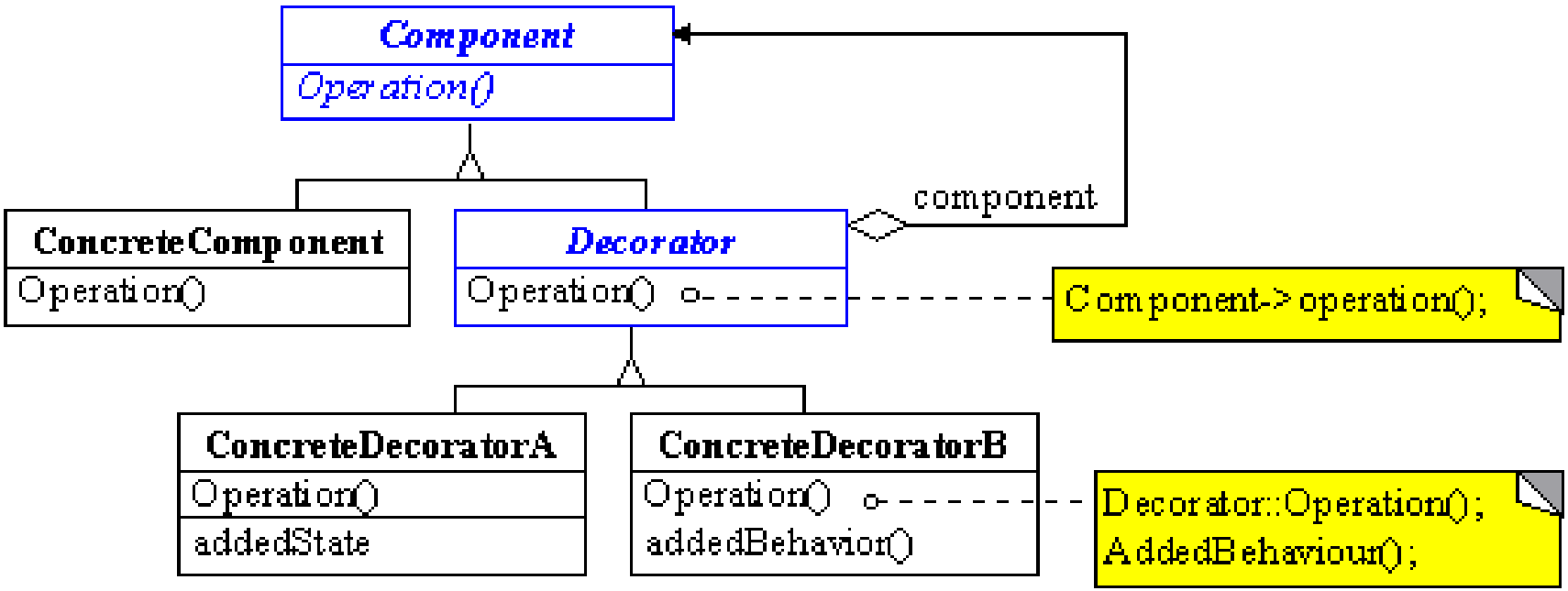
Outline

Estrutura

Estrutura

- Cada objeto Decorator contem outro objeto Decorator.
- Um decorator é como um pequeno composite cujos elementos possuem cada qual um filho único.
- Diferentemente do padrão Composite, cujo propósito é compor objetos agregados, o propósito do Decorator é compor comportamentos.

Estrutura


Component
Operation()
component
ConcreteComponent
Operation()
Decorator
Operation()
Component->operation();
ConcreteDecoratorA
Operation()
addedState
ConcreteDecoratorB
Operation()
addedBehavior()
Decorator::Operation();
AddedBehaviour();

Participantes

Participantes

Componente: define a interface para objetos que podem ter responsabilidades acrescentadas a eles dinamicamente.

ComponenteConcreto: define um objeto para o qual responsabilidades adicionais podem ser atribuídas

Decorator: mantém uma referência para um objeto Componente. Define uma interface que segue a interface de Componente

DecoratorConcretoA e DecoratorConcretoB: acrescenta responsabilidades ao componente

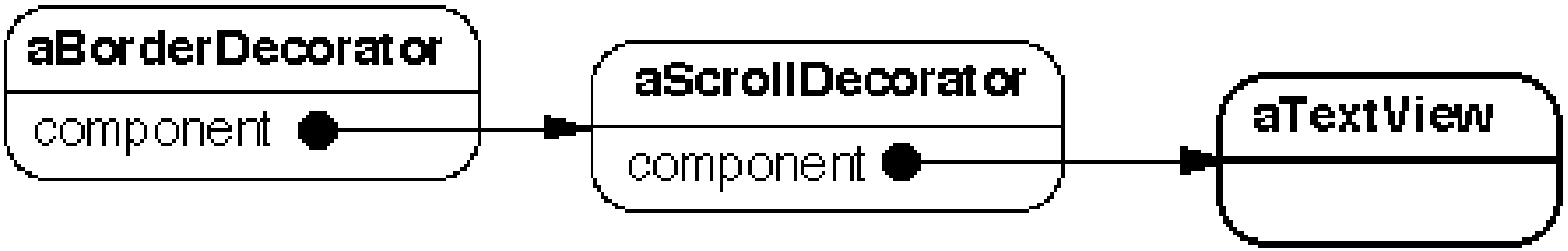
Outline

Implementação

Implementação

- Fornece uma flexibilidade maior do que a herança estática.
- Evita a necessidade de colocar classes sobrecarregadas de recursos em uma posição mais alta da hierarquia.
- Simplifica a codificação permitindo que você desenvolva uma série de classes com funcionalidades específicas, em vez de codificar todo o comportamento no objeto.
- Aprimora a extensibilidade do objeto, pois as alterações são feitas codificando novas classes.

Implementação




```
Window decoratedWindow = new BorderDecorator(
                         new HorizontalScrollBarDecorator(
                         new VerticalScrollBarDecorator(
                         new SimpleWindow())));
```

Componente

```
interface Window {
    public void draw(); // draws the Window
    public String getDescription(); // returns a description of the Window

}
```

ComponenteConcreto

```java
import javax.swing.JFrame;


// implementation of a simple Window without any scrollbars
class SimpleWindow extends JFrame implements Window{

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public SimpleWindow(){
        super("Decorator");
    }
    public void draw() {
        setSize(600, 600);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    public String getDescription() {
        return "simple window";
    }
}
```

Decorator

```
// abstract decorator class - note that it implements Window
abstract class WindowDecorator extends JFrame implements Window {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    protected Window decoratedWindow; // the Window being decorated

    public WindowDecorator (Window decoratedWindow) {
        this.decoratedWindow = decoratedWindow;
    }
   public void draw() {
        decoratedWindow.draw();
    }

}
```

DecoratorConcreto

```
// the first concrete decorator which adds Horizontal scrollbar functionality
class HorizontalScrollBarDecorator extends WindowDecorator {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public HorizontalScrollBarDecorator(Window decoratedWindow) {
        super(decoratedWindow);
    }

    public void draw() {
        drawHorizontalScrollBar();
        decoratedWindow.draw();
    }

    private void drawHorizontalScrollBar() {

        JScrollPane scroll = new JScrollPane();
        scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        getInstance().getContentPane().add(scroll);

    }

    public String getDescription() {
        return decoratedWindow.getDescription() + ", including Horizontal scrollbars";
    }

}
```

Outline

Referência

Referência

- http://www.pg.cefetpr.br/coinf/simone/patterns/decorator.php
- http://pages.cpsc.ucalgary.ca/ jada-low/seng60904/decorator_paper.html
- http://albertoleal.eti.br/2009/06/design-pattern-implementando-o-decorator/