Patrons de Conception
-
Decorator

Simon Urli
urli@i3s.unice.fr

Master 1 MIAGE
2014-2015

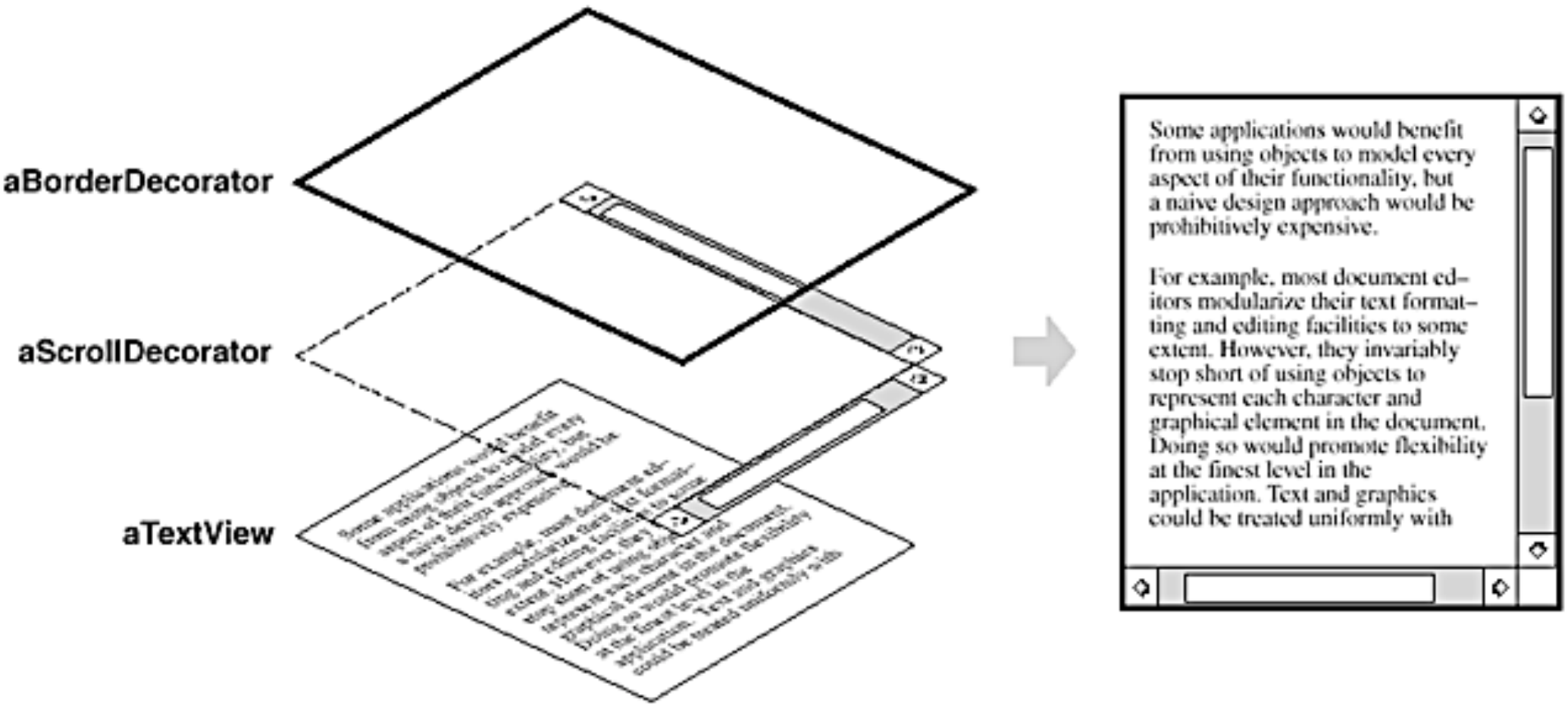
Objectifs

“ Attacher dynamiquement des capacités additionnelles à un objet et fournir ainsi une alternative flexible à l’héritage pour étendre des fonctionnalités. ”

Classification : Patron de structure

Synonyme : wrapper (!!!)

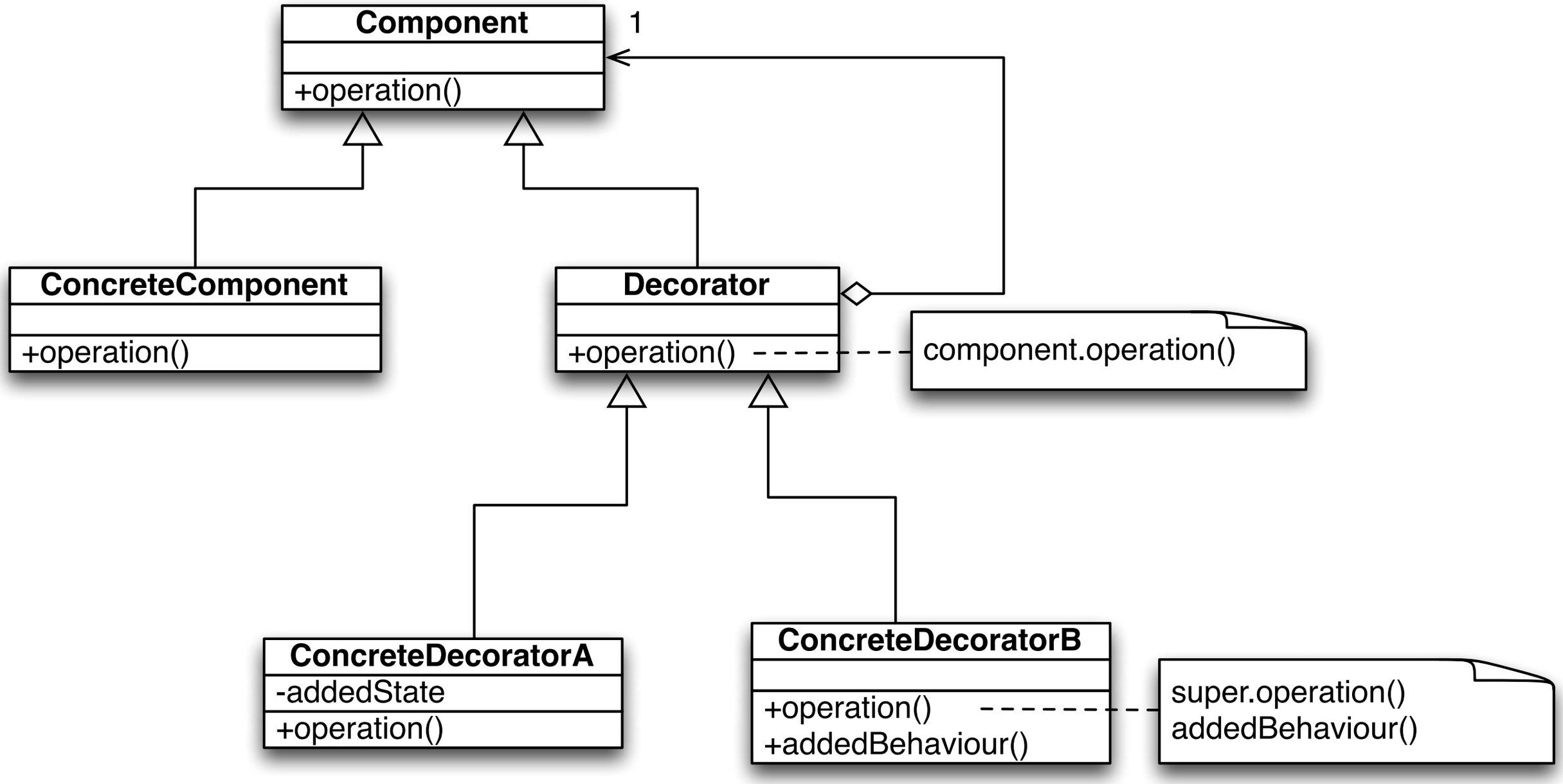
Exemple


aBorderDecorator
aScrollDecorator
aTextView
Some applications would benefit from using objects to model every aspect of their functionality, but a naive design approach would be prohibitively expensive.
For example, most document editors modularize their text formatting and editing facilities to some extent. However, they invariably stop short of using objects to represent each character and graphical element in the document. Doing so would promote flexibility at the finest level in the application. Text and graphics could be treated uniformly with

Applications

- Ajouter des capacités de manière transparente et dynamique
- Pouvoir retirer des capacités
- Gestion de nombreuses variantes possibles là où l'héritage aurait mené à une explosion de combinaisons

Structure


Component
1
+operation()
ConcreteComponent
+operation()
Decorator
+operation()
component.operation()
ConcreteDecoratorA
-addedState
+operation()
ConcreteDecoratorB
+operation()
+addedBehaviour()
super.operation()
addedBehaviour()

Implémentation

- Component : décrit l'interface des objets que l'on peut décorer
- ConcreteComponent : définit une implémentation de l'objet décorable
- Decorator : contient une référence à un objet de type Component et définit une interface conforme à Component
- ConcreteDecorator : ajoute une capacité à component

Exemple

```
public interface ITextView {
      public void setText(String text);
      public String getText();
      public void paint();
  }
  public TextView implements ITextView {
      private String text;
      public TextView(String text) {
         this.text = text;
      }
      public String getText() { return this.text; }
      public void setText(String text) { this.text = text; }
      public void paint() {
         this.displayText(this.getText());
      }
      ...
}
```

Exemple

```
public abstract TextDecorator implements ITextView {
      private ITextView textView;
      public TextDecorator(ITextView itv) {
          this.textView = itv;
      }
      public String getText() { return this.itv.getText(); }
      public void setText(String text) { this.itv.setText(String text); }
      public void paint() { this.itv.paint(); }
  }
  public ScrollDecorator extends TextDecorator {
      private ScrollBar scroll;
      public ScrollDecorator(ITextView itv) {
          super(itv);
          this.scroll = new ScrollBar(super.getText());
      }
      public String getText() {
          return super.getText().substring(this.scroll.minOffset(), this.scroll.maxOffset());
      }
      public void paint() {
          this.itv.displayText(this.getText());
          this.scroll.paint();
      }
}
```

Exemple

```
public BorderDecorator extends TextDecorator {
    public BorderDecorator(ITextView itv) {
        super(itv);
    }
    public void paint() {
        this.itv.paint();
        this.paintBorder();
    }
    ...
}
public class Main {
    public static void main(String[] args) {
        TextView tv = new TextView(‘‘une chaine quelconque...’’);
        ScrollerDecorator scroll = new ScrollDecorator(tv);
        BorderDecorator border = new BorderDecorator(scroll);
        border.paint();
    }
}
```

Cas d'utilisation

- Bibliothèque java.io (InputStream, BufferedInputStream, FilterInputStream, ...)

Exercice*

- Un cocktail est constitué d'une base (alcool ou jus de fruit) et est enrichi en intégrant d'autres éléments (autre(s) boisson(s), sirop, morceaux de fruits, décorations, flamme, etc). Modéliser un cocktail.

**Attention, cet exercice est à consommer avec modération.*