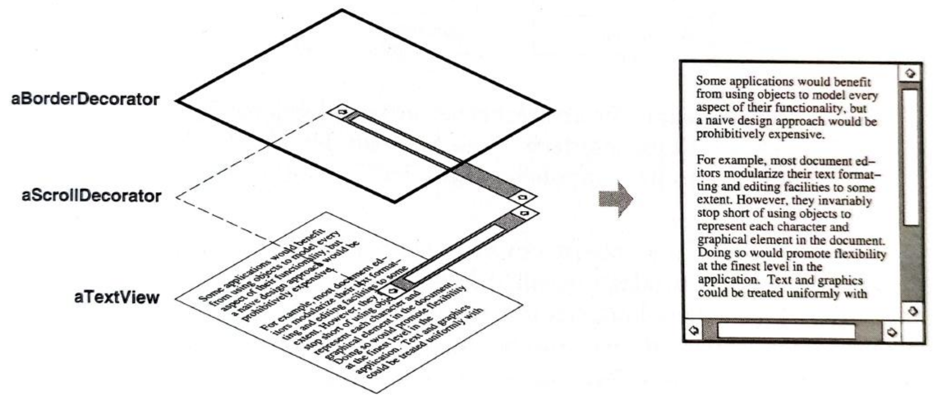
Decorator

An object-structural pattern

Learning goals

1. Learn the idea, structure, and Java implementation of the Decorator design pattern.
2. Learn to apply the Decorator DP in your own programming.

Idea of Decorator



Some applications would benefit from using objects to model every aspect of their functionality, but a naive design approach would be prohibitively expensive.

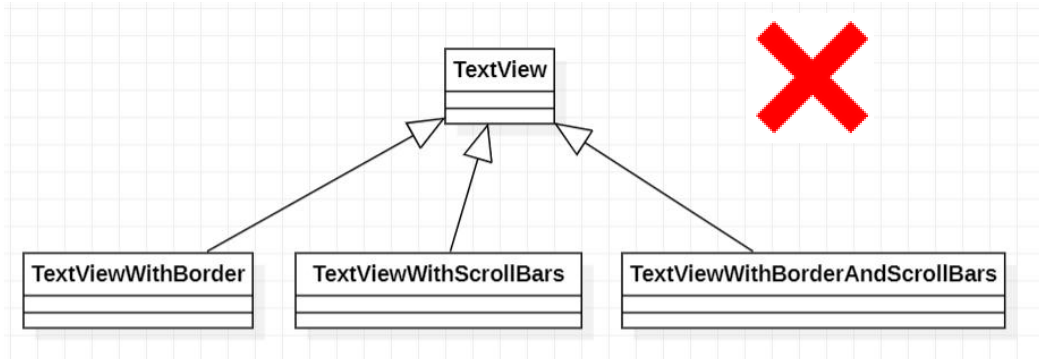
For example, most document ed–itors modularize their text format–ting and editing facilities to some extent. However, they invariably stop short of using objects to represent each character and graphical element in the document. Doing so would promote flexibility at the finest level in the application. Text and graphics could be treated uniformly with

- In class design, the following approach is usually taken:
  - The base class defines the basic functionality.
  - The derived subclasses extend this functionality.
- **Problem**: What if the object should have extended decorative functionality of two or more subclasses simultaneously?

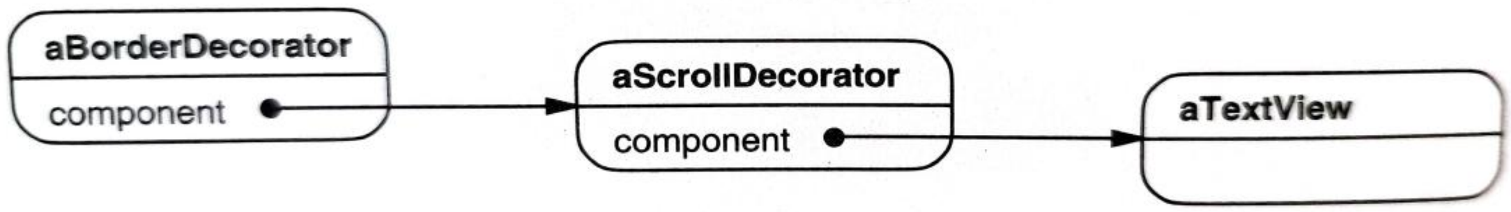
Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 175

Idea of Decorator

- Generally, it is not feasibly to create a subclass for each combination of extra features.
  - Combinatory explosion in the number of subclasses
    - For *n* additional features, 2^n-1 subclasses
  - Complicated, repetitive code, bad extendability, bad testability



Idea of Decorator



- In the Decorator DP, the reference to an object to be decorated is place inside other object, a **decorator**.
- Then, a reference to this decorated object can be placed inside another decorator etc., making a chain.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 176

Idea of Decorator

- The base object as well as the decorated objects share the same superclass.
- The object’s core functionality is declared in that shared superclass.
  - As a consequence, base objects and decorated objects can be used in the similar way.
  - An object may have any number – and any combination – of decorators attached.
- The base object is completely agnostic of its decorations.

Example

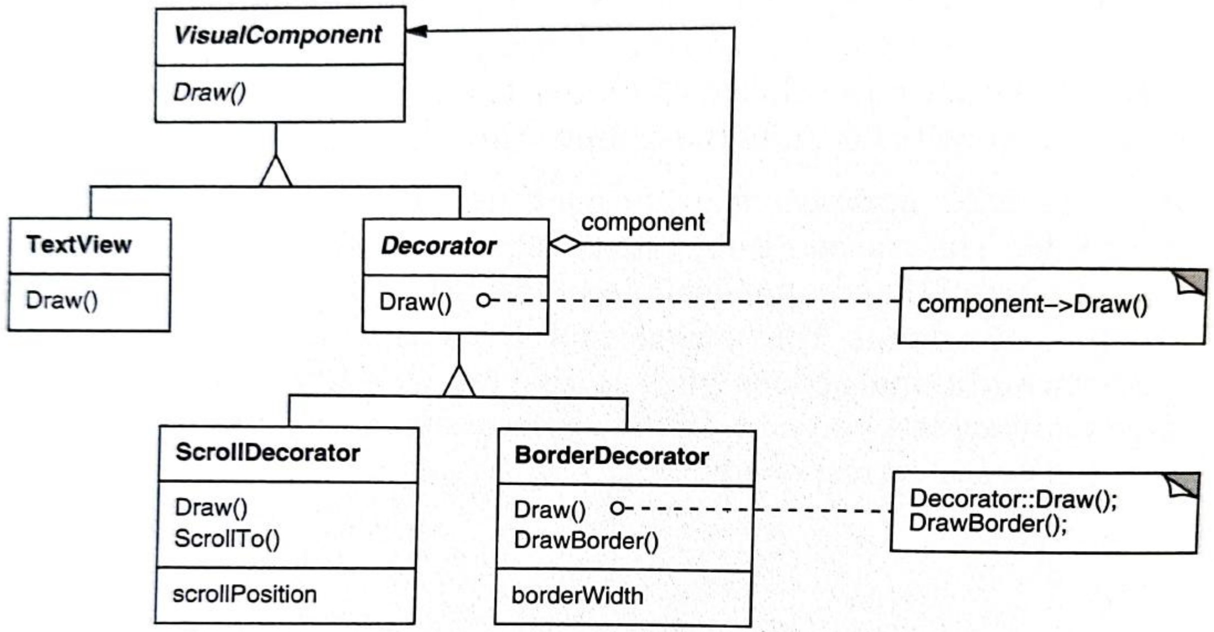
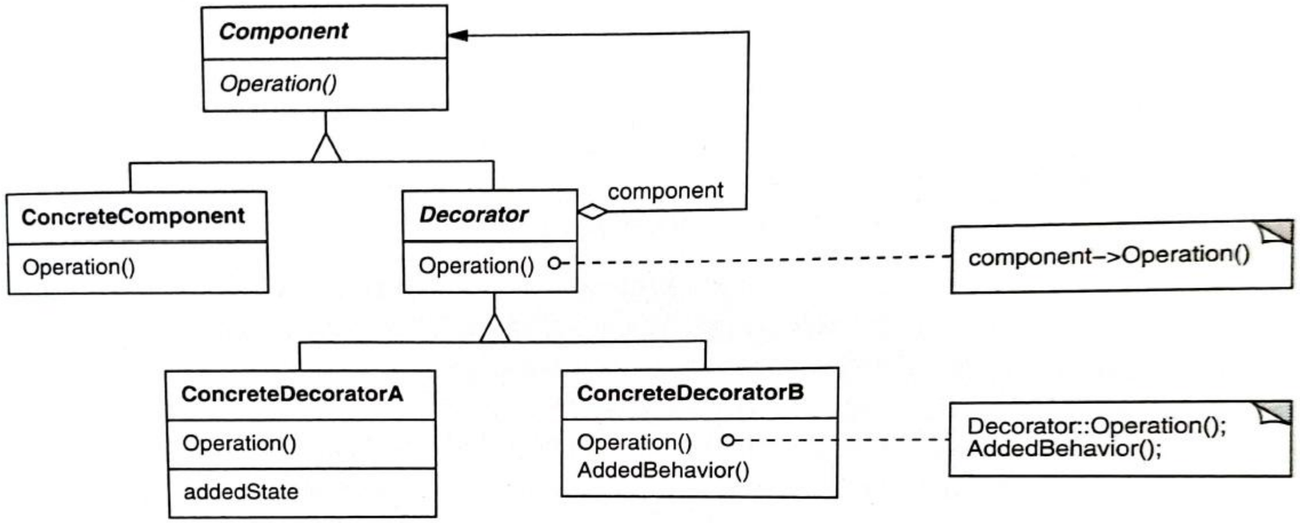


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 176

Example

- At runtime, we can have e.g.:
  - A ScrollDecorator that contains a…
  - … BorderDecorator that contains a …
  - … TextView.
- That is, decorators can be chained – or nested – in an arbitrary way.
- They all share the VisualComponent superclass.
  - Thus, the draw() method can be called for any component – decorated or not.

General structure



- Note that each Decorator object has a reference to a Component
  - It contains the object that is being decorated by this Decorator.
  - That object is either a Decorator or a ConcreteComponent.

Roles

- **Component**: Interface for objects that may have additional decoration
- **Concrete Component:** Concrete object to be optionally decorated by one or more Decorators
- **Decorator**: Contains a reference to a component, and implements the Component interface.
- **Concrete Decorator**: Adds functionality to a component, before or after forwarding the control to the contained object's method.

Practical issues

- Decorators are used for modifying the presentation of an object’s state, not modifying the internals.
- The Component interface should be kept as lightweight as possible to keep the Decorators lightweight, too.
- Decorations can easily be added and withdrawn at runtime.
- At runtime, decorators forward requests to the component referenced
  - This can be either another decorated component, or a base component.
  - The decoration can be added before or after this forwarding.
- As the name implies, the DP is designed to modify the presentation, not internals of an object.
  - For modifying the internals, Strategy DP is used.
- As a drawback, a lot of small classes is generated.
  - This can make the source code harder to understand.