# Decorator Design Pattern

## Structural Design Patterns

Structural patterns are concerned with how classes and objects are composed to form larger structures.

| | |
|---|---|
| Adapter | structural *class* pattern |
| Composite | structural *object* pattern |
| ⏩ Decorator | structural *object* pattern |

## Decorator Pattern

Decorator is a structural pattern that composes objects recursively
to allow an open-ended number of additional responsibilities.

It describes how to add responsibilities to objects *dynamically*.

Each Decorator object conforms to the interface of its component, and forwards messages to the underlying component.

The Decorator can do its job either before or after forwarding a message.

## Decorator Pattern

Decorator is a structural pattern that composes objects recursively
to allow an open-ended number of additional responsibilities.

It describes how to add responsibilities to objects *dynamically*.

Each Decorator object conforms to the interface of its component, and forwards messages to the underlying component.

The Decorator can do its job either before or after forwarding a message.

# Decorator Pattern

## Intent

Attach additional responsibilities to an object dynamically.

Decorators provide a flexible alternative to subclassing for extending functionality.

## Also Known As

Wrapper

## Decorator Pattern

### Motivation

Sometimes we want to add responsibilities to individual objects, not to an entire class.

An object being debugged should log its execution trace

- method invocations
- argument values
- return values

## Decorator Pattern

### Motivation

Sometimes we want to add responsibilities to individual objects, not to an entire class.

An object may require thread safety in a new runtime context

  across only a few specific method invocations

## Decorator Pattern

### Applicability

Use Decorator

to add responsibilities to individual objects dynamically and transparently (without affecting any other objects).

to support for responsibilities that can be withdrawn.

*continues...*

## Decorator Pattern

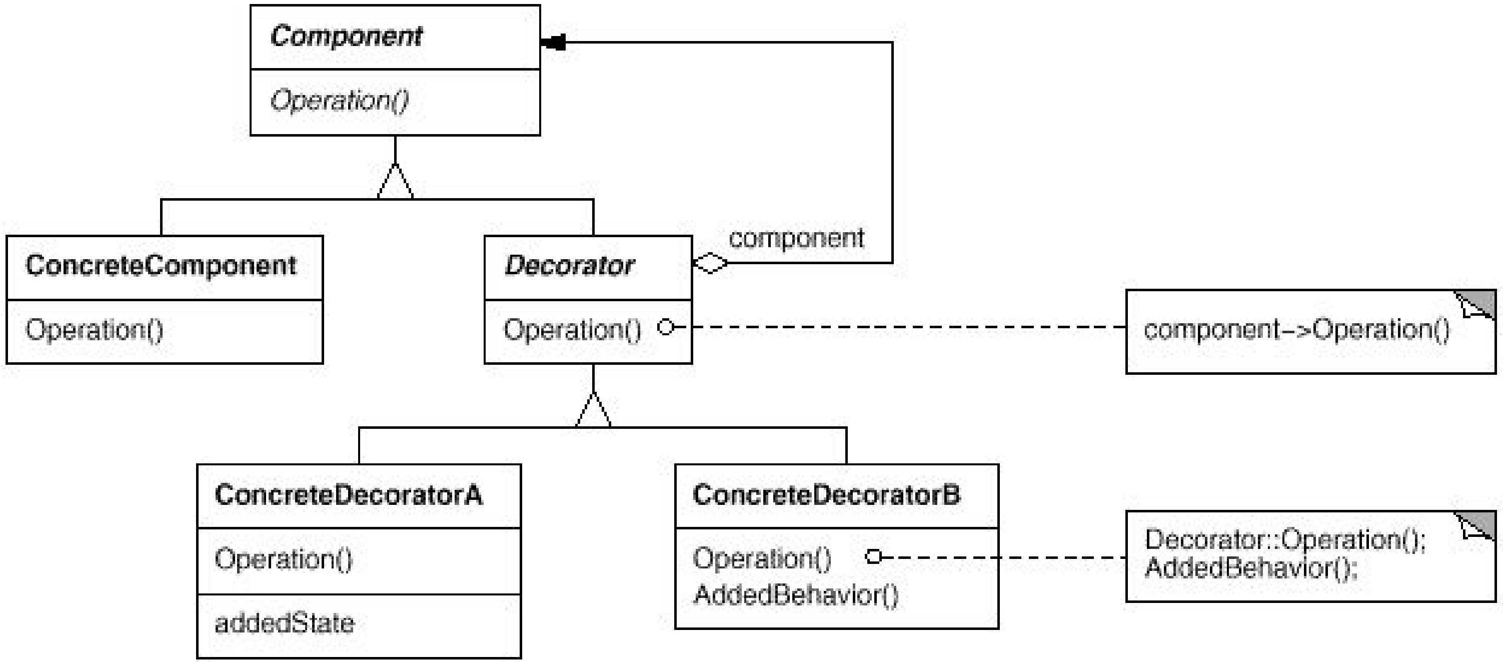
### Applicability

Use Decorator

- to avoid proliferation of subclasses to support every combination.
- to deal with cases where it is not possible to subclass because the class definition may be hidden or unavailable.

# Decorator Pattern

## Structure



## Decorator Pattern

### Participants

Component (GameObject, DataHolder)

defines the interface for objects that can have responsibilities added to them dynamically.

ConcreteComponent (SpeedyBot, KeyValueStore)

defines an object to which additional responsibilities can be attached.

*continues...*

## Decorator Pattern

### Participants

Decorator

> maintains a reference to a Component object and defines an interface that conforms to Component's interface.

ConcreteDecorator (LoggingDecorator, ThreadSafeDecorator)

> adds responsibilities to the component.

## Decorator Pattern

### Collaborations

Decorator forwards requests to its Component object.

It may optionally perform additional operations before and after forwarding the request.

## Decorator Pattern

### Consequences

More flexibility than static inheritance.

A decorator and its component aren't identical.

Lots of little objects.

## Decorator Pattern

### Implementation

Interface conformance.

Omitting the abstract Decorator class.

Changing the skin of an object versus changing its internals

# Decorator Pattern

## Sample Code

Decorator forwards requests to its Component object.

It may optionally perform additional operations before and after forwarding the request.