DECORATOR

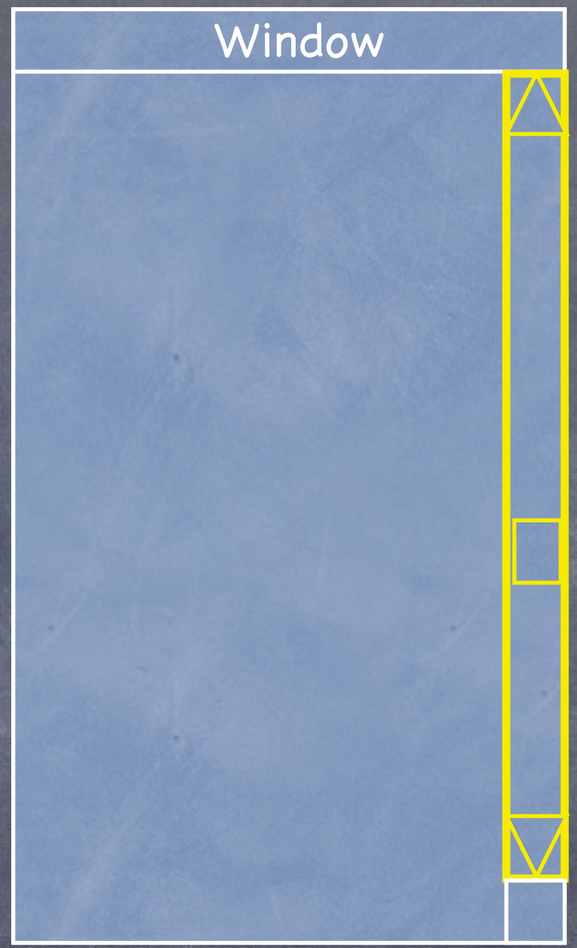
A Structural Pattern

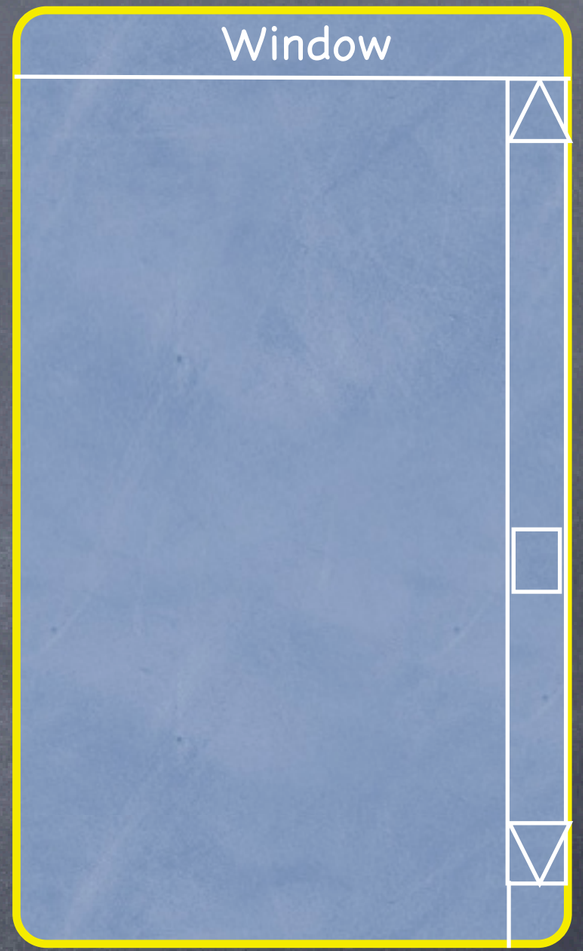
Matt Mayfield, 26 Jan 2010

Decorator: Intent

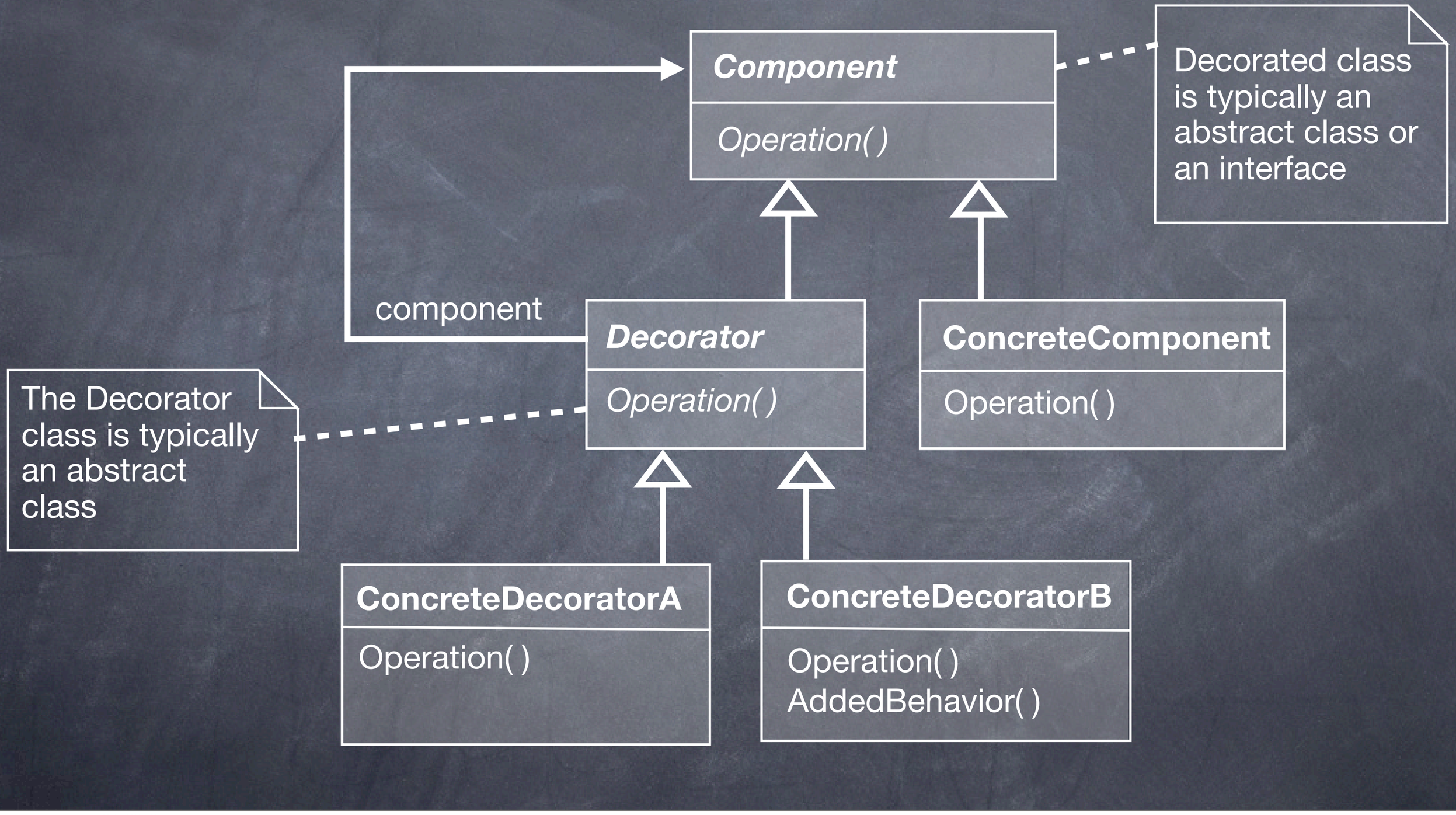
- A way to attach additional responsibilities to an object dynamically at run time.
- A flexible alternative to subclassing for extending functionality of a class
- Also known as a "Wrapper"

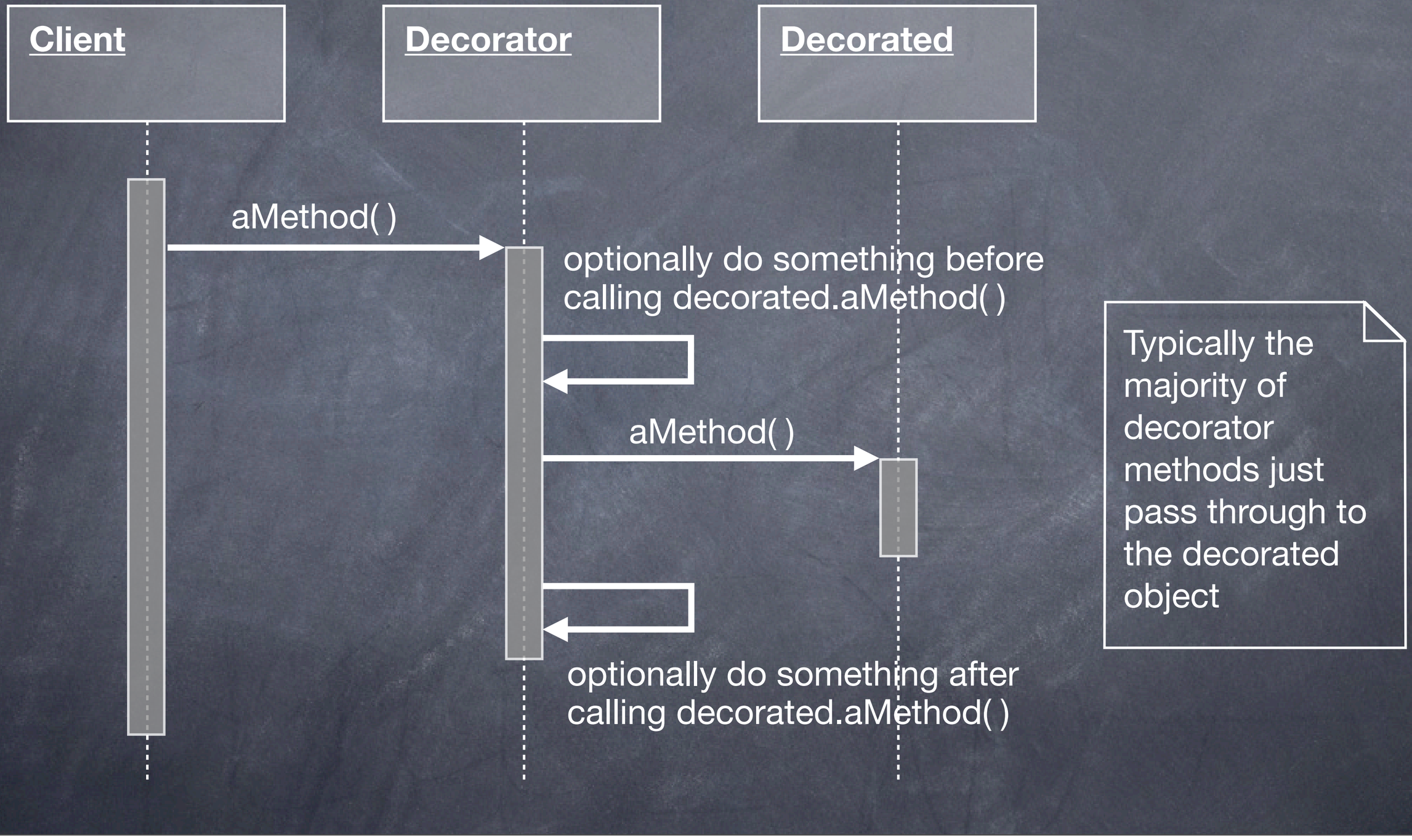
Base Class/Object





Decorated Class/Object


Component
Operation( )
Decorated class is typically an abstract class or an interface
component
Decorator
Operation( )
ConcreteComponent
Operation( )
The Decorator class is typically an abstract class
ConcreteDecoratorA
Operation( )
ConcreteDecoratorB
Operation( )
AddedBehavior( )


Client
Decorator
Decorated
aMethod( )
optionally do something before calling decorated.aMethod( )
aMethod( )
optionally do something after calling decorated.aMethod( )
Typically the majority of decorator methods just pass through to the decorated object

An example

```
//base class
class Window() {
   getPosition();
   setPosition();
   ...
}
```

```
//decorator class
class decorateWindow() {
   getPosition() {
        //pass getPosition()
   }
   setPosition() {
        //pass setPosition()
   }
   uniqueMethod();
   ...
}
```

```
//create decorated
object

decorateWindow X = new
   decorateWindow (
      new Window() );

X.getPosition();
X.uniqueMethod();
```

Another example

```
FileReader frdr = new FileReader(filename);

//decorator
BufferedReader brdr = new BufferedReader(frdr);

//second decorator
LineNumberReader lrdr = new LineNumberReader(brdr);

//call lrdr as you would frdr, now with additional capabilties
```

Decorator: Advantages

- More flexibility that static inheritance. Can add, mix or even remove responsibilities of a class incrementally as needed (at runtime)
- Existing classes do not have to be modified to support extra functionality, as they are not aware that they are being decorated
- You can restrict the use of an object's public methods. Instead of forwarding calls to a public method, a decorator can veto a method by throwing an exception from the wrapper method.

Decorator: Disadvantages

- Decorators are transparent but not identical to the components they decorate
- Decorators are small and can be confusing to debug as their combined collaboration is generally the value not their distinct class or the value of their local variables

Related Patterns

- Adapter: will give an object a completely new interface
- Composite: intended for object aggregation
- Strategy: lets you change the guts of an object as opposed to the skin (decorator)