

## CS4233: Class 15

### The Observer pattern

The Observer pattern is a commonly used pattern that appears frequently in building user interfaces.

The purpose of the Observer pattern is to establish an association (dependency) between objects so that when an object changes, the change is communicated to its observers. This pattern is a key part of the Model-View-Controller (MVC) pattern.

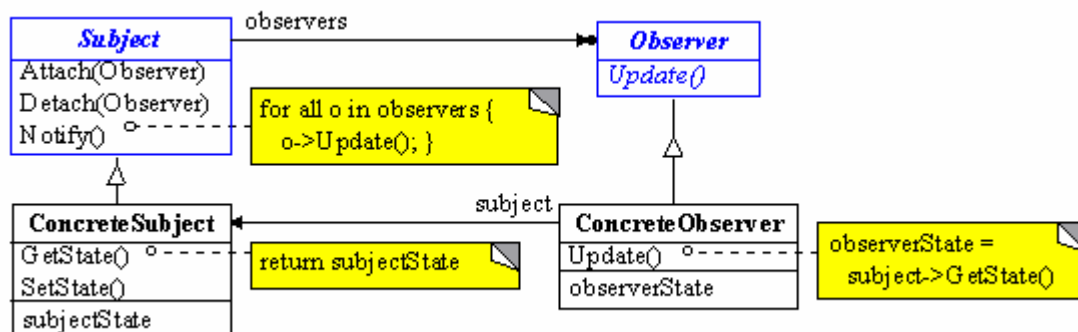
#### Context

There are a set of objects that need to be notified when an event occurs.

#### Solution

Observers delegate the responsibility for reporting the event to another object, the Subject. The Subject has the responsibility for keeping track of all of the Observers and, when the event occurs, it notifies the observer by calling a specific method.

The following UML diagram shows the basic structure of the Observer pattern:



Taken from *Design Patterns Explained* by Shalloway and Trott

#### Example

Consider the design of a weather station that has sensors that will read current temperature, humidity, barometric pressure, and other weather readings. How might you implement this without using an Observer? What about using an Observer pattern -- what are different ways you could implement this?

#### Consequences

Positive consequences:

- Abstract coupling can be extended and reused.
- Dynamic relationships can be resolved at run time.
- Supports broadcast events.
- There can be unexpected / unnecessary event notifications.

## Loose coupling

The Observer is an example of *loose coupling*. Objects are loosely coupled if they are able to interact, but have very little knowledge of each other. In fact, the less knowledge they have of each other, the more loosely coupled they are.

## The Decorator pattern

The decorator pattern is another pattern that is used with graphical user interfaces. However, it is more than a UI pattern. It provides a way of adding functionality to an object without knowing that it's really doing so.

The standard example of a decorator is the scrollbar in a pane. The pane that is being scrolled should not be responsible for doing the scrolling. Of course, there is an association between the scrollbars and the pane.

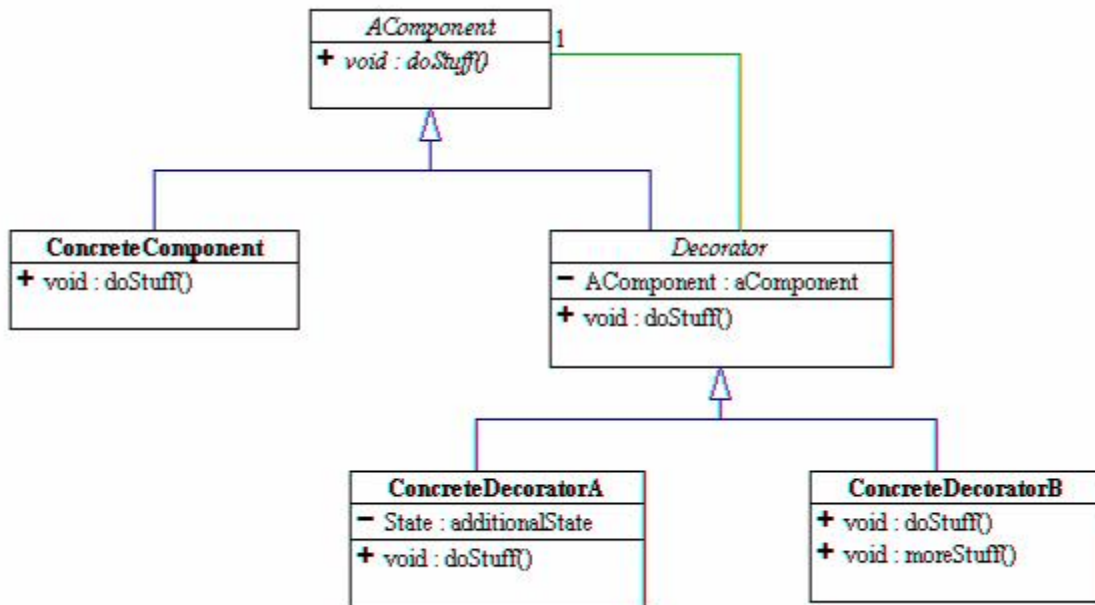
## Context

The Decorator pattern is one to consider when you are faced with a class that can have several variants in behavior. When you begin to look at an exploding class hierarchy, you might be wise to think of the Decorator. It is a good example of delegation. The only difference is that the decorated object is not aware of the delegated behavior.

## Solution

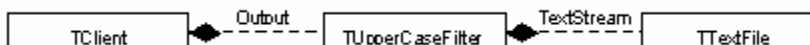
You create a decorator class that implements the same methods as the original class and then adds to the behavior. It delegates the original behavior to the original class (usually) and then adds its behavior to the original results.

A UML diagram of the solution would look like the following:



## Chains of decorators

Decorators may be used in a chain. For example, you can think of the pipe and filter pattern, or the way streams and readers are composed in Java.



## **Example**

Think of the way that some eating establishments prepare sandwiches. They have a base cost for the sandwich, and add charges for additional items, like cheeses, peppers, and so on. How can you calculate the price of a sandwich easily?

One way is to have the sandwich know what goes on it. This could be done in several ways, some more efficient than others. We could have separate fields to store information about each type of additional piece of the sandwich, or we could have a collection of such pieces that all satisfy the same interface. This second solution would work okay until you decide that sandwiches can be part of a larger order, like a smiley meal where the smiley meal has a sandwich, one side dish, and a drink. Now you have to do some calculations where the smiley meals are priced to take \$1 off the total cost of the component pieces.

Using the decorator pattern you would make an abstract class, call it a MealComponent and it has a method called calculateCost. Now you start with the SmileyMeal and "decorate" it with a sandwich. Then decorate that with swiss cheese. Continue this until you have all of the pieces. You take the last one, and tel it to calculateCost. If it has decorated any other component, it asks that to calculateCost before adding in its own cost. This continues down to the smiley meal which has a cost of -1 dollar.

## **Reading**

Chapter 5, sections 5.1-5.6.