

CS 349

Design Patterns



Design Patterns

- *Design Patterns*, Gamma, Helm, Johnson, & Vlissides (1994)
 - Gang of Four / “GoF”
- A set of common approaches for solving recurring problems in software design
- A shared language for describing problems
- Emphases:
 - Reusability
 - Distinguishing between features of a problem which vary and those which are constant
 - Factoring out the elements that vary into separate classes
 - Delegation over inheritance



Design Patterns for User Interfaces

- Observer
- Strategy
- Flyweight
- Composite
- Factory Method
- Command
- Chain of Responsibility
- Memento
- Decorator
- Singleton

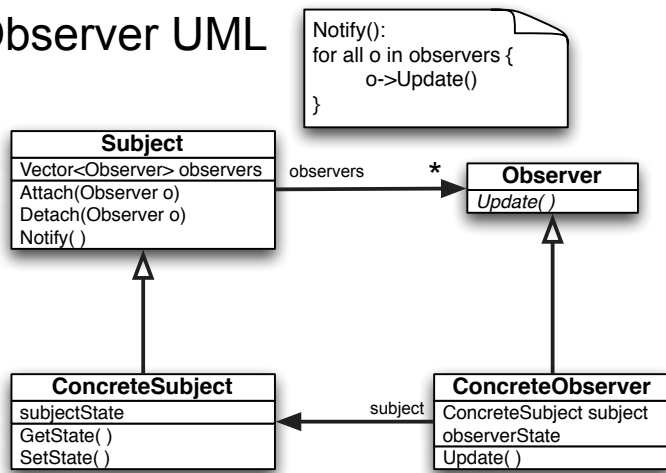


Observer

- Provides a well-defined mechanism that allows objects to communicate without knowing each others' specific types
 - Promotes loose coupling
- AKA “listener” and “publish-subscribe”
- Examples in Java
 - ActionListener
 - PropertyChangeListener
 - WindowListener...
 - Integral part in Model-View-Controller
- Delegates in C#



Observer UML



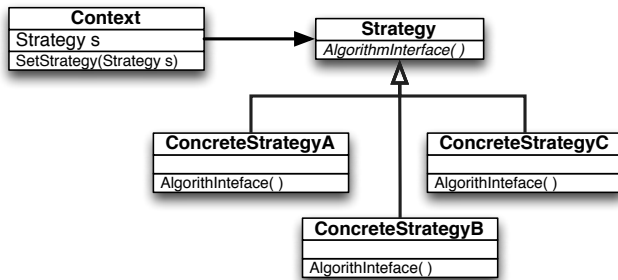
Strategy

- Factors out an algorithm into separate object, allowing a client to dynamically switch algorithms
- Really simple example:
 - quicksort's compare function for sorting any data set
- Other examples?



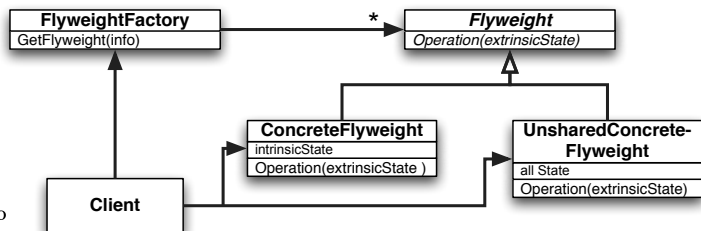
Strategy

- Example: Layout managers
 - Layout manager defines an algorithm for arranging objects in space
 - Examples in Java: FlowLayout, BorderLayout, GridLayout, ...



Flyweight

- A shared object that can be used in multiple contexts
 - Same object represents multiple, finer-grained objects
 - *Reduces resource needs*
- Examples in Java:
 - Renderers and editors in components such as JTable, JTree



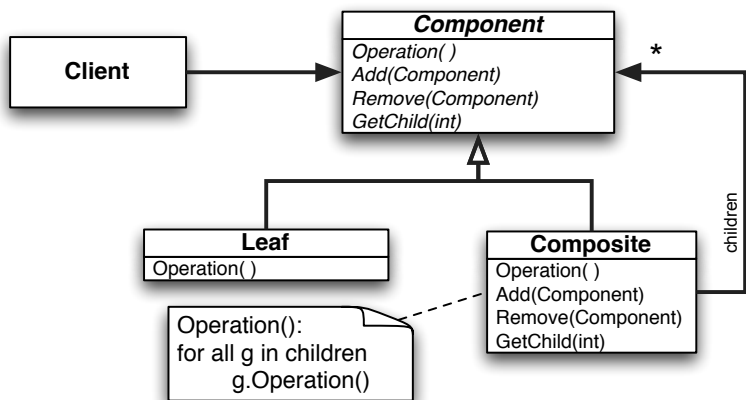
Composite

- Composite defines an interface, plus capability to add other composite objects to an object
- Allows a set of similar objects to be grouped together and treated as one
- Examples?

Composite

- Example
 - Set of objects in vector drawing program
 - Select all of them and allow user to manipulate one to manipulate them all
- Example
 - JPanel: is a component, groups components allowing them to be treated as one

Composite Pattern UML



Factory Method

- Define interface for creating objects implementing a specific interface
- Register the factory with a component that will request new objects at runtime
- Example
 - In drawing application, method of browsing stencils and adding them to canvas is constant
 - What varies is the *type* of stencil
 - How can we make it easy for third parties to create new stencils?
 - Factories provide one method of solving this problem

Factory Example

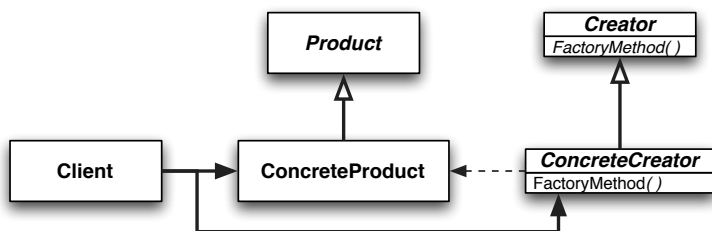
- Define a factory class for producing new stencils:

```
public interface StencilFactory {  
    public Stencil createStencil();  
    public Image getStencilThumbnail();  
}
```

- Third parties implement StencilFactory interface to provide new types of stencils to application
- Register interface with component that calls it whenever it wants an object of that type



Factory Pattern UML



Command

- Represent an operation/command as an object
 - Creates a consistent way of managing commands in interface
 - Command object can be invoked to perform operation
 - Enables interface logging, undo

- Typical interface:

```
public interface Command {  
    public void doIt();  
}
```

- Examples in Java:

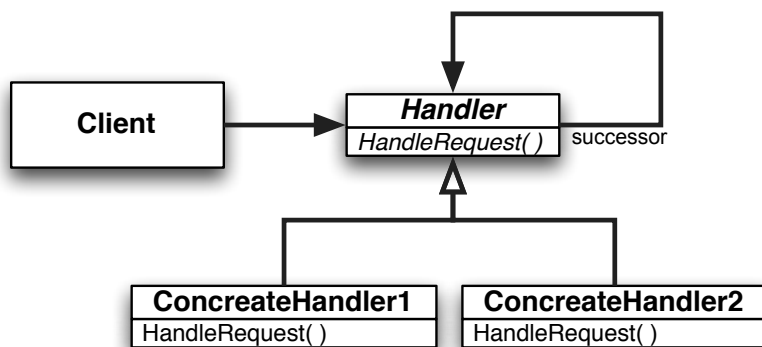
- UndoableEdit
- Runnable



Chain of Responsibility

- Create a chain of loosely coupled objects, each of which may respond to event or pass it to next object in the chain
- Examples
 - Event dispatch (eg, bottom-up vs. top-down dispatch) If event not relevant, can pass it to parent, child, or other object that may wish to act on the event.
 - Context-sensitive help: Have help information attached to each component in the GUI. If it can handle the request, do so. If not, pass it up to the parent component in the GUI.

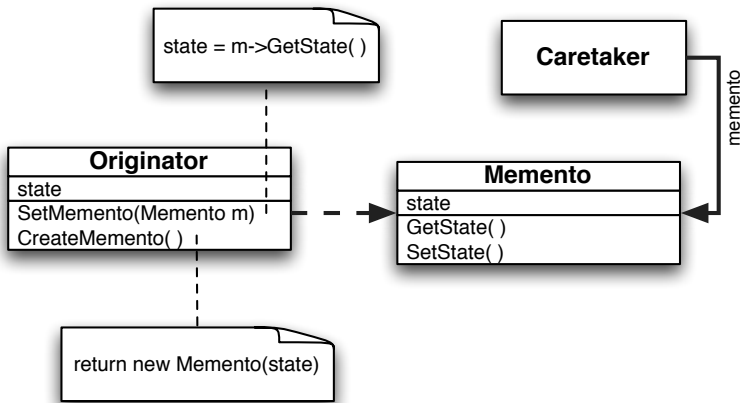
Chain of Responsibility UML



Memento

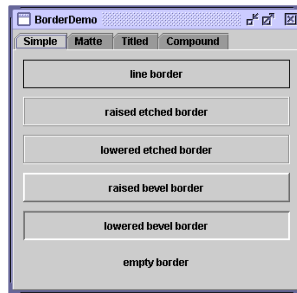
- Capture and store an object's internal state so it can later be restored
 - `getState()`, `setState(...)`
 - Do it without exposing the internal state of the object.
 - How? Object prepares an object with its internal state; only it knows how to read the state in the object and how to reset itself.
- Examples
 - Used in supporting undo (store state), branching histories
 - State of document or objects (save)

Memento UML



Decorator

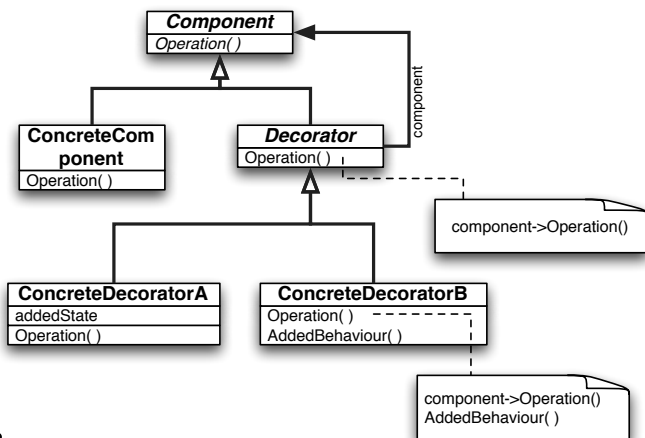
- Layer additional functionality on an object without changing interface of object
- Reduces need to make “top-heavy” classes with lots of functionality
- GoF Examples (but not Decorators in Java)
 - Scroll panes
 - Borders
- Java: IO Streams



From the Java Tutorial



Decorator UML



Singleton

- Class that can have only *one instance* of itself
- Global access to that instance
- Used for things that will last the entire session
 - Application
 - Plug-in managers
 - Factories
- To make a singleton
 - Create a private constructor
 - Create a public, static method to get its instance
 - If instance hasn't been created, create object

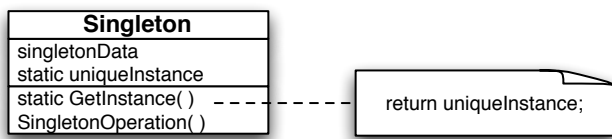


Singleton Example

```
public class Application {  
    private static Application s_Instance = null;  
  
    private Application() {  
        // init object  
    }  
  
    public synchronized static  
        Application getInstance()  
    {  
        if (s_Instance == null) {  
            s_Instance = new Application();  
        }  
        return s_Instance;  
    }  
}
```



Singleton UML



Wrap-up

- User interfaces make use of many design patterns.
- It pays to be familiar with them!
- Most important:
 - Observer
 - Strategy
 - Command (coming)