Structural Patterns

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

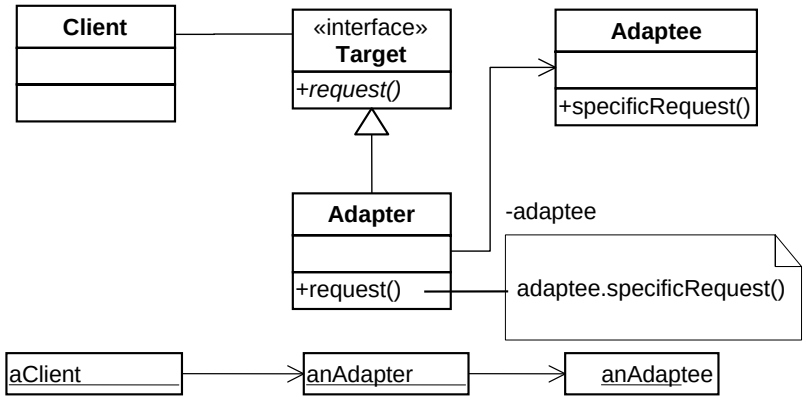
- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

General Principles (inform all design patterns)

- Program to an interface, not an implementation.
- Favour object composition over class inheritance.

Object Adapter

Converts the interface of a class into another interface that clients expect.



Use of Adapters in Java API

A class which implements WindowListener must provide an implementation for each method, even it only needs to implement one e.g. windowClosing.

```
public interface WindowListener extends EventListener
{
  public void windowOpened(WindowEvent e);
  public void windowClosing(WindowEvent e);
  public void windowClosed(WindowEvent e);
  public void windowIconified(WindowEvent e);
  public void windowDeiconified(WindowEvent e);
  public void windowActivated(WindowEvent e);
  public void windowDeactivated(WindowEvent e);
}
```

Window Adapter

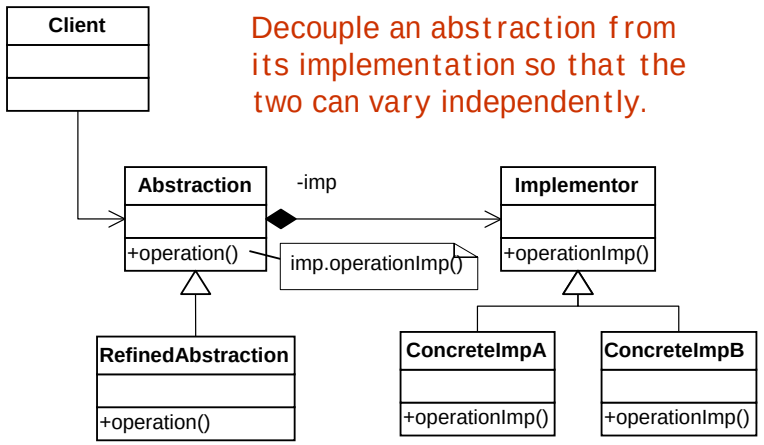
Provides default implementation for WindowListener methods.

```
public abstract class WindowAdapter implements
WindowListener {
  public void windowOpened(WindowEvent e) {}
  public void windowClosing(WindowEvent e) {}
  public void windowClosed(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
}
```


Usage - anonymous class

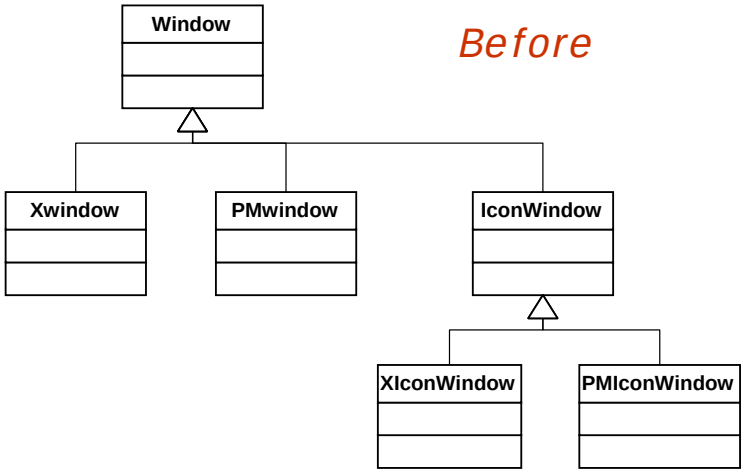
```
public class Closer extends Frame {

  public Closer() {
    /* stuff */
    addWindowListener(
      new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      }
    );
   /* the rest */
  }
```


Bridge

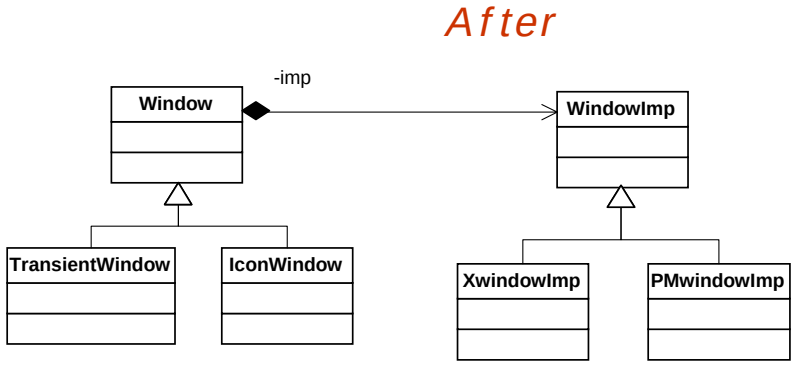


Decouple an abstraction from its implementation so that the two can vary independently.

Bridge - example



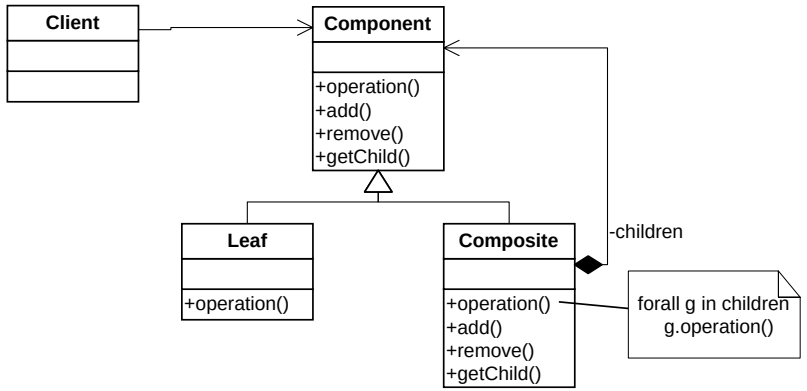
Bridge - example



Bridge - usage

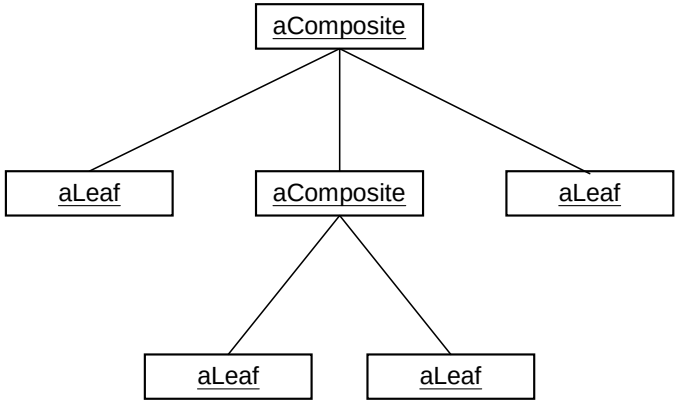
- to avoid permanent binding between an abstraction and its implementation e.g. select or switch at run-time
- both abstraction and implementation must be extensible by subclassing
- changes in implementation of abstraction do not affect client code

Composite



Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions uniformly.

Composite Object example



Composites in Java

- Swing components are organised as a composite.
- Which components are leaf components and which are composites?
- Which methods are used to navigate the composite tree?

Decorator

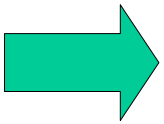
aBorderDecorator

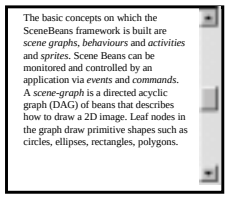
aScrollDecorator

aTextView

The basic concepts on which the SceneBeans framework is built are *scene graphs*, *behaviours* and *activities* and *sprites*. Scene Beans can be monitored and controlled by an application via *events* and *commands*. A *scene-graph* is a directed acyclic graph (DAG) of beans that describes how to draw a 2D image. Leaf nodes in the graph draw primitive shapes such as circles, ellipses, rectangles, polygons.

Attach additional responsibilities to an object dynamically. Provide a flexible alternative to subclassing for extending functionality.





Java example

Decorated Buttons

Decorator Class

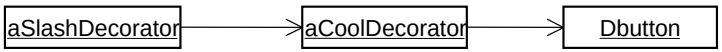
```
public class Decorator extends JComponent
{
    public Decorator(JComponent c) {
        setLayout(new BorderLayout());
        add("Center", c);
    }
}
```


Concrete Decorator

```
public class SlashDecorator extends Decorator {
    int x1, y1, w1, h1;
    public SlashDecorator(JComponent c) {
        super(c);
    }
    public void setBounds(int x, int y, int w, int h) {
        x1 = x; y1= y;
        w1 = w; h1 = h;
        super.setBounds(x, y, w, h);
    }
    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
        g.drawLine(0, 0, w1, h1);
    }
}
```

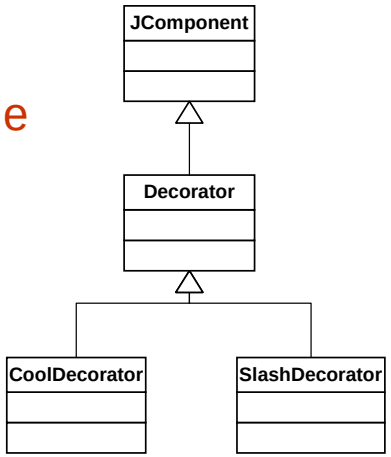

Using the Decorator

```
JPanel jp = new JPanel();
getContentPane().add(jp);
jp.add( new CoolDecorator (
         new JButton("Cbutton")));
jp.add( new SlashDecorator(
         new CoolDecorator(
          new JButton("Dbutton"))));
```



Decorator Example

Structure



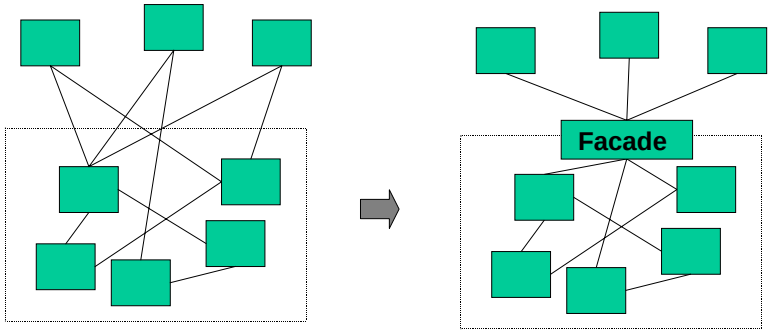
Question

In what way is a **decorator** different from an **adapter**?

Adapters change the interface of a class for a client.
Decorators add methods to particular instances of classes rather than to all of them.

Facade

Provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.



Flyweight

Use sharing to support large numbers of fine-grained objects efficiently

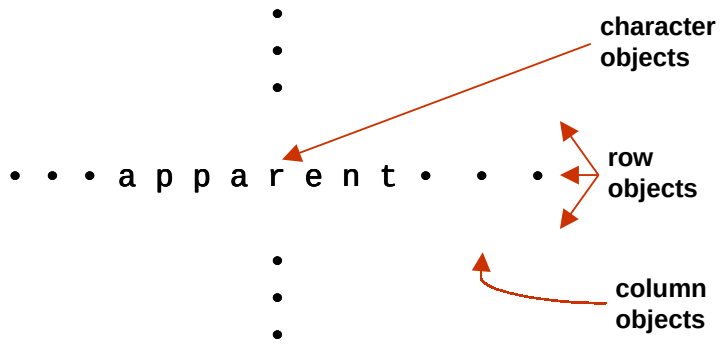
Key Concepts:

Intrinsic state:
stored in flyweight and independent of context, shareable

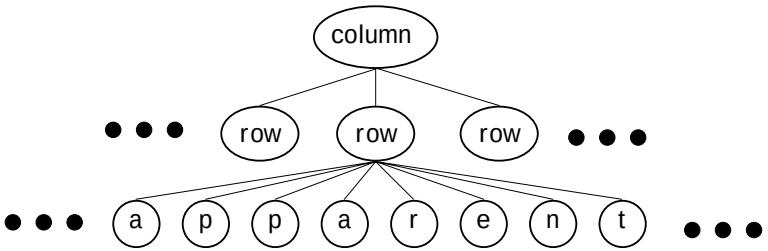
Extrinsic state:
dependent on context, passed to flyweight by client

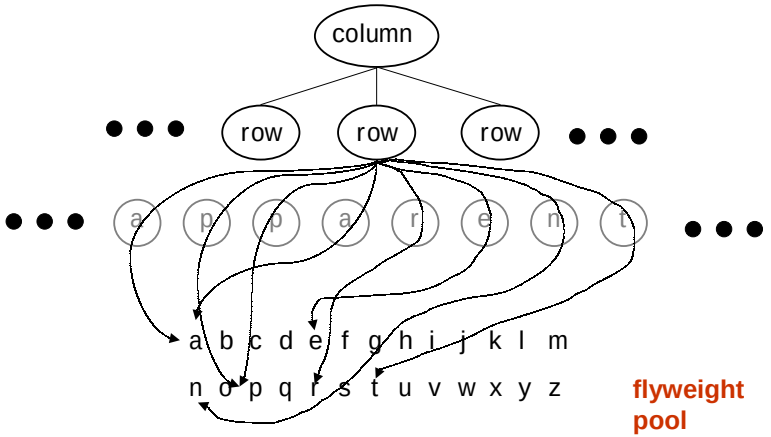
Flyweight - example

Consider how a text editor stores characters:

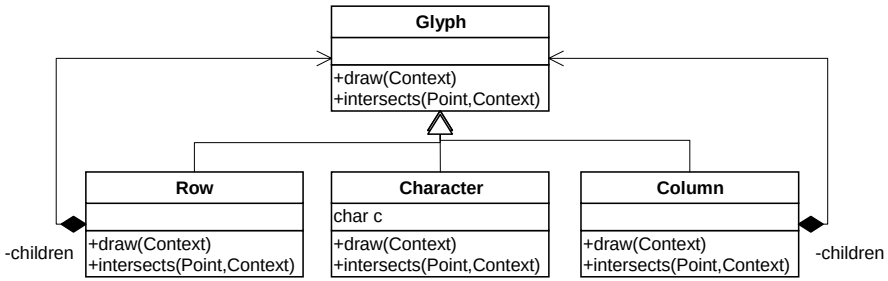


Logically, there is an object for every occurrence of a given character in the document.





Flyweight - structure



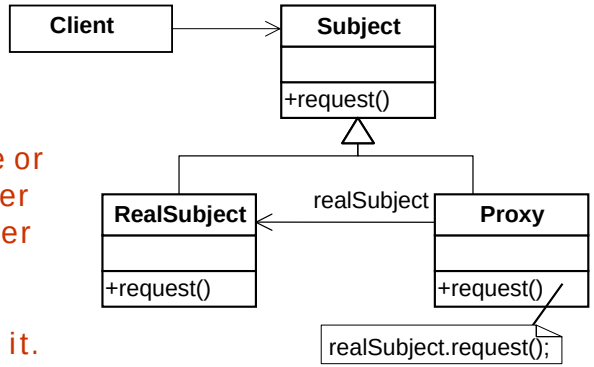
A flyweight representing the letter “a” only stores the corresponding character code; it does not store its location or font. Clients supply the context information that the flyweight needs to draw itself.

Flyweights - consequences

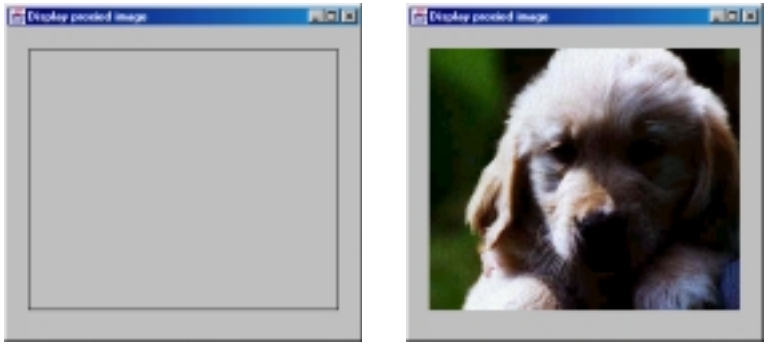
- May introduce additional run-time cost.
- Usually offset by saving in space
  - Dependent on the amount of intrinsic state per object
  - and the amount of intrinsic state per object
  - and whether extrinsic state is computed or stored.

Proxy

Provide a surrogate or placeholder for another object to control access to it.

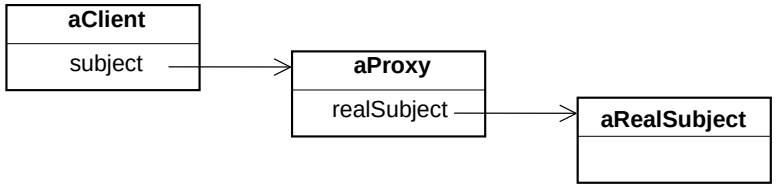


Example



Proxy image displayed until real image loads.

Proxy - example instance structure



Question

Discuss the use of proxies in Java RMI.