Composite Design Pattern

Rick Mercer
Recurring problem:

- Often complex structures are built with *container* and *primitive objects*. Container objects can contain other objects.
- How can code that uses these classes treat all the objects in the structure identically sometimes, yet differently when it matters?

Solution:

- Define an abstract class that represents primitives *and* containers.

*Composite* was used in the View class of Smalltalk MVC as well as most other GUI toolkits.
General Form of Composite

- Client uses Component
- Component has * children
- Leaf
  - + Operation1()
  - + Operation2()
- Composite
  - + Operation1()
  - + Operation2()
  - + addComponent()
  - + removeComponent()
  - + getComponent()
  - + getComponentCount()
Participants

Component
- Declares the interface for all objects in the composition
- Implements default behavior, as appropriate
- Declares an algorithm interface (set of methods) for accessing and managing child components

Leaf: Has no children: it is a primitive

Composite: Defines behavior for components having children
- Also implements child-related operations of Component
Participants

- Component has operations that apply to all
  - The component can be a Composite or a Leaf

- Composite adds methods indicating a collection: add(), and remove()

- In each method, a Component is passed
  - Can add either a Child or a Component

- Component should not add itself

- Should not add a Component to a leaf
Usage Example

```java
ArrayList<Object> a = new ArrayList<Object>();
a.add("abc");
a.add("cde");

ArrayList<Object> b = new ArrayList<Object>();
b.add(1.11);
b.add(2.22);

System.out.println("a: "+ a);
System.out.println("b: "+ b);
b.add(a);
b.add(b);
// a.add(b); Stack Overflow

System.out.println("a: "+ a);
System.out.println("b: "+ b);
```

What types are the Leafs here?
___________________

What type is the Composite?
___________________

What type is the Component?
___________________

Output?
________________________________
________________________________
Java Swing has four major pieces:

- Events and EventListeners
- Layouts
- Drawing
- Graphical Components

- The root of all is also named Component

Component utilizes the Composite pattern in several ways, here's one you'll need for the new project
JMenus in Java Swing

Java menus use the Composite Design Pattern

- **JMenuBar** is a composite extending **JComponent**
- **JMenuBar** is a composite extending **JComponent**
  - Can add others like **JLabel**, **JTextField**
  - Can also add **JMenuItem** to **JMenuItem**

- **JMenuItem** has three subclasses
  - **JMenu**
  - **JRadioButtonMenuItem**
  - **JCheckboxMenuItem**
JMenuItem menu = new JMenu("Composite");
menu.setMnemonic('C'); // Open with alt-C
// Create two leafs
JLabel label = new JLabel("Label");
JTextField textField = new JTextField("text field");
menu.add(label);
menu.add(textField);
// Add a Composite
JMenuItem menuItem = new JMenuItem("menu item");
menu.add(menuItem);
// Add two Composites to a Composite
JMenuItem jmi1Nest = new JMenu("Nest 1");
menu.add(jmi1Nest);
JMenuItem jmiNested1 = new JMenuItem("Nested in 1");
jmi1Nest.add(jmiNested1);
JMenuItem jmiNested2 = new JMenuItem("Nested in 1 also");
jmi1Nest.add(jmiNested2);
// Add two more Composites
JMenuItem checkBox = new JCheckBoxMenuItem("Human", false);
JMenuItem radioButton = new JRadioButtonMenuItem("Computer", true);
menu.add(checkBox);
menu.add(radioButton);
// Add two more Composites
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
menuBar.add(menu);

Run JMenuItemDemoComposite.java

See code demo page