Design Patterns

Engr. Abdul-Rahman Mahmood

DPM, MCP, QMR(ISO9001:2000)

armahmood786@yahoo.com
alphapeeler.sf.net/pubkeys/pkey.htm
pk.linkedin.com/in/armahmood
www.twitter.com/alphapeeler
www.facebook.com/alphapeeler
abdulmahmood-sss alphasecure
armahmood786@hotmail.com
http://alphapeeler.sf.net/me

alphasecure@gmail.com
http://alphapeeler.sourceforge.net
http://alphapeeler.tumblr.com
armahmood786@jabber.org
alphapeeler@aim.com
mahmood_cubix 48660186
alphapeeler@icloud.com
http://alphapeeler.sf.net/acms/

Composite Design Pattern

What is Composite pattern?

Composite is one of the 23 Design Patterns which were selected by the GoF (Gang of Four).

		Purpose		
		Creation	Structure	Behavior
Scope	Class	Factory Method		Interpreter Template
	Objects	Abstract Factory Builder Prototype Singleton	Adapter Bridge **Composite** Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Intent

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

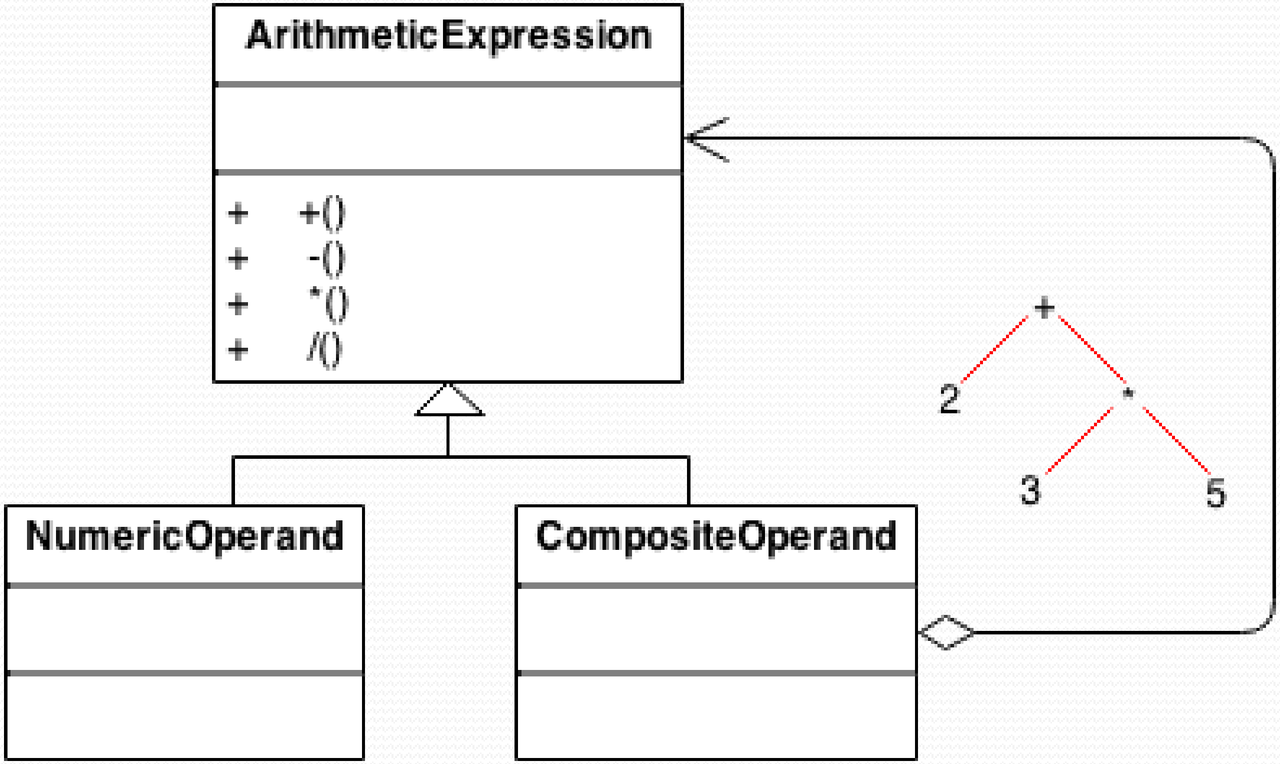
- Recursive composition

- "Directories contain entries, each of which could be a directory."

- 1-to-many "has a" up the "is a" hierarchy

Example

- arithmetic expressions are Composites. An arithmetic expression consists of an operand, an operator (+ - * /), and another operand. The operand can be a number, or another arithmetic expression. Thus, 2 + 3 and (2 + 3) + (4 * 6) are both valid expressions.



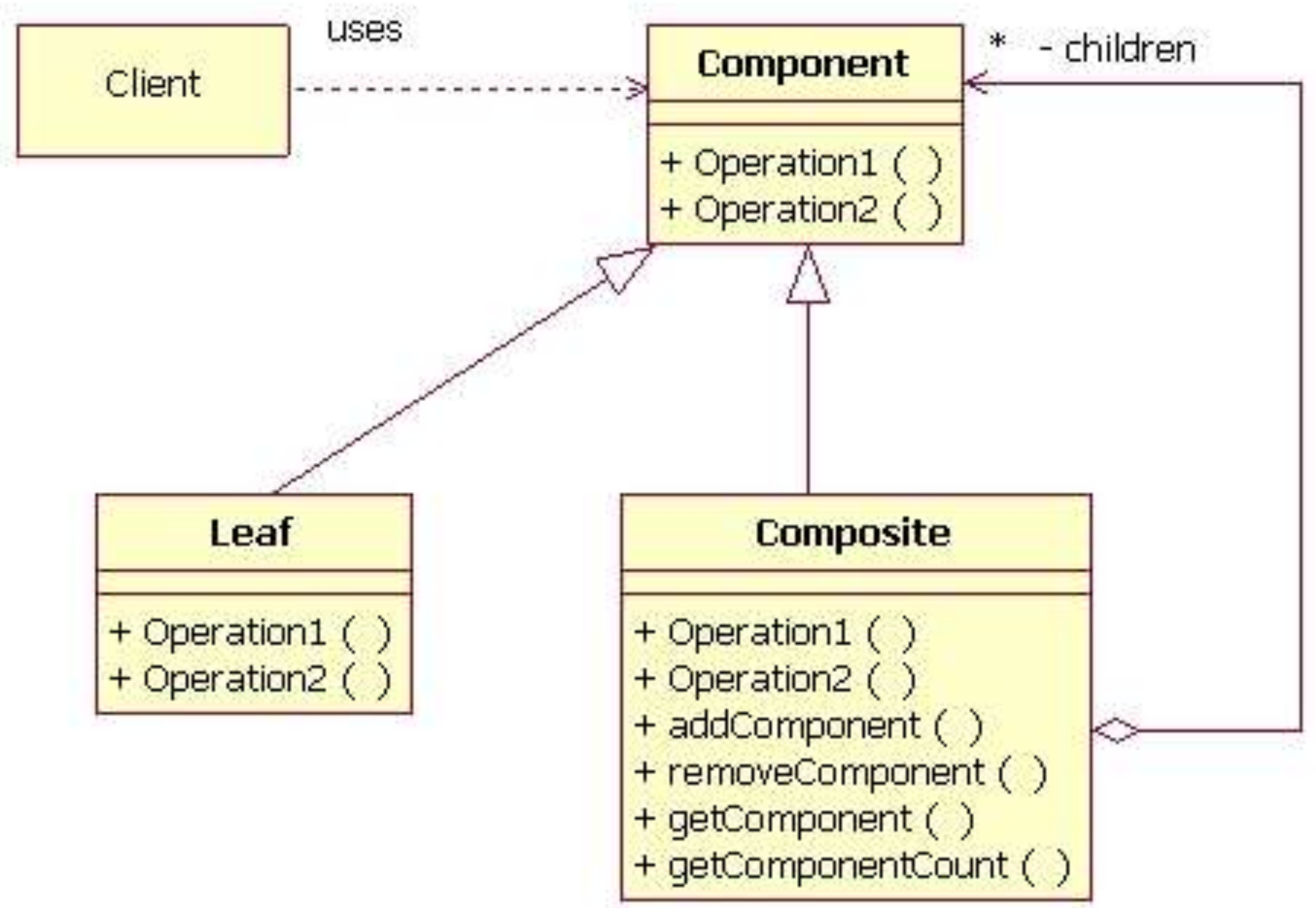
Examples

- Menus that contain menu items, each of which could be a menu.
- Row-column GUI layout managers that contain widgets, each of which could be a row-column GUI layout manager.
- Directories that contain files, each of which could be a directory.
- Containers that contain Elements, each of which could be a Container.

Composite Pattern

- Recurring problem:
  - Application needs to manipulate a hierarchical collection of "primitive" and "composite" objects.
  - Processing of a primitive object is handled one way, and of a composite object is handled differently.
  - Having to query the "type" of each object before attempting to process it is not feasible.
- 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



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

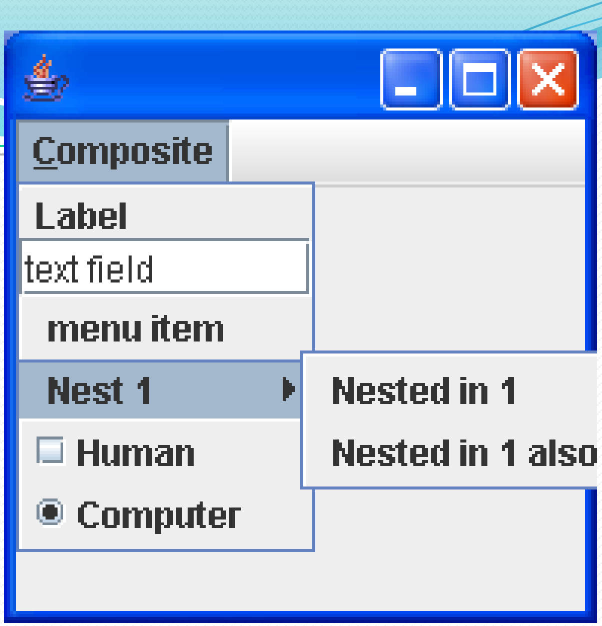
Use Example: Java Swing

- 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
  - One you may find useful or need for your final project

JMenus in Java Swing

- Java menus use the Composite Design Pattern
- `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`

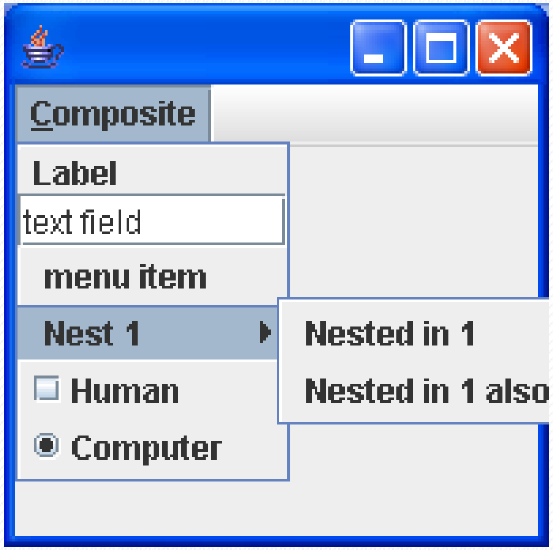
JMenus in Java Swing



```
JMenuItem menu = new JMenu("Composite");
menu.setMnemonic('C');//Open with alt-C
// Create two leafs
JLabel label = new JLabel("Label");
JTextField textF = new JTextField("text field");
menu.add(label);
menu.add(textF);
// 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);
```

JMenuItemDemoComposite

```
// 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`

Check List

- Ensure that your problem is about representing "whole-part" hierarchical relationships.

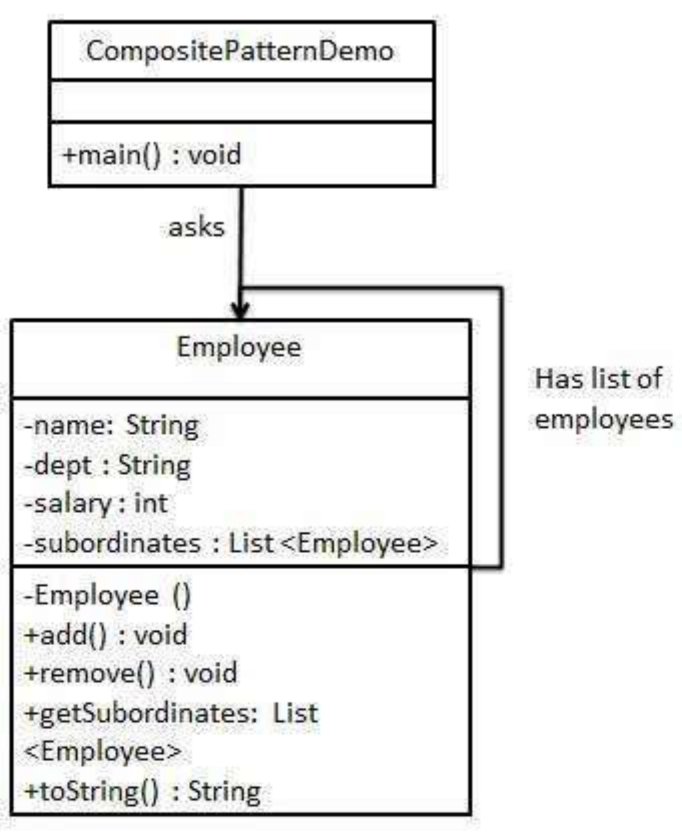
- Consider the rule, "Containers that contain containees, each of which could be a container." For example, "Assemblies that contain components, each of which could be an assembly.“

Rule of Thumb

- Composite and Decorator have similar structure diagrams, reflecting the fact that both rely on recursive composition to organize an open-ended number of objects.
    - **Composite** gives an unified interface to a leaf and composite.
    - **Decorator** gives additional feature to leaf, while giving unified interface.
- Composite can be traversed with Iterator.
- Composite could use Chain of Responsibility to let components access global properties through their parent.
- Composite can let you compose a Mediator out of smaller pieces through recursive composition.
- Flyweight is often combined with Composite to implement shared leaf nodes.

Example: Composite

- We have a class *Employee* which acts as composite pattern actor class. *CompositePatternDemo*, our demo class will use *Employee* class to add department level hierarchy and print all employees.



Example: Composite

- Step1: Create *Employee* class having list of *Employee* objects. *Employee.java*

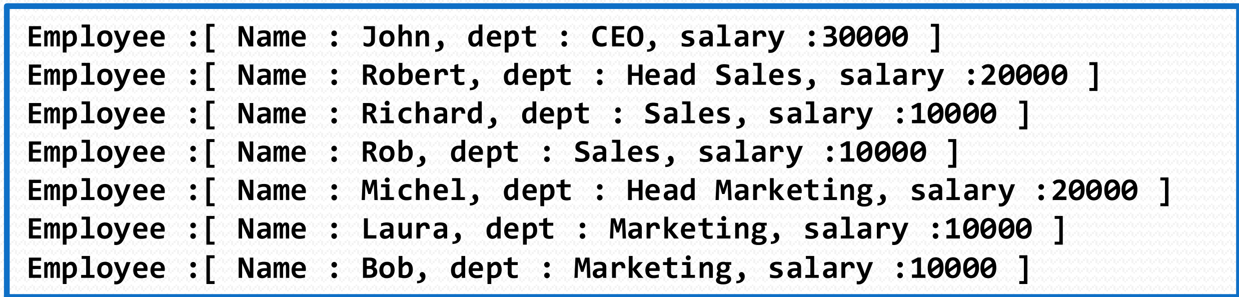
```
import java.util.ArrayList;
import java.util.List;
public class Employee {
private String name;
   private String dept;
   private int salary;
   private List<Employee> subordinates;
   public Employee(String name,String dept, int sal) {
      this.name = name;
      this.dept = dept;
      this.salary = sal;
      subordinates = new ArrayList<Employee>();
   }
   public void add(Employee e) {
      subordinates.add(e);
   }
   public void remove(Employee e) {
      subordinates.remove(e);
   }
   public List<Employee> getSubordinates(){
     return subordinates;
   }
   public String toString(){
      return ("Employee :[ Name : "+name+", dept : "+dept+", salary :"+salary+" ]");
   }
}
```

Example: Composite

- Step2: Use *Employee* class to create & print employee hierarchy. *CompositePatternDemo.java*

```
public class CompositePatternDemo {
public static void main(String[] args) {
Employee CEO = new Employee("John","CEO", 30000);
Employee headSales = new Employee("Robert","Head Sales", 20000);
Employee headMarketing = new Employee("Michel","Head Marketing", 20000);
Employee clerk1 = new Employee("Laura","Marketing", 10000);
Employee clerk2 = new Employee("Bob","Marketing", 10000);
Employee salesExecutive1 = new Employee("Richard","Sales", 10000);
Employee salesExecutive2 = new Employee("Rob","Sales", 10000);
CEO.add(headSales);
CEO.add(headMarketing);
headSales.add(salesExecutive1);
headSales.add(salesExecutive2);
headMarketing.add(clerk1);
headMarketing.add(clerk2);
System.out.println(CEO);
for (Employee headEmployee : CEO.getSubordinates()) {
   System.out.println(headEmployee);
   for (Employee employee : headEmployee.getSubordinates()) {
      System.out.println(employee);
   }
}
}
}
```

Output:



```
Employee :[ Name : John, dept : CEO, salary :30000 ]
Employee :[ Name : Robert, dept : Head Sales, salary :20000 ]
Employee :[ Name : Richard, dept : Sales, salary :10000 ]
Employee :[ Name : Rob, dept : Sales, salary :10000 ]
Employee :[ Name : Michel, dept : Head Marketing, salary :20000 ]
Employee :[ Name : Laura, dept : Marketing, salary :10000 ]
Employee :[ Name : Bob, dept : Marketing, salary :10000 ]
```