**Design Patterns - Composite Pattern**

Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern comes under structural pattern as this pattern creates a tree structure of group of objects.

This pattern creates a class that contains group of its own objects. This class provides ways to modify its group of same objects.

We are demonstrating use of composite pattern via following example in which we will show employees hierarchy of an organization.

Composite design pattern treats each node in two ways:
1) **Composite** – Composite means it can have other objects below it.
2) **leaf** – leaf means it has no objects below it.

**Tree structure:**


The above figure shows a typical Composite object structure. As you can see, there can be many children to a single parent i.e. Composite, but only one parent per child.

Let's see the 4 elements of composite pattern.

##### 1) Component

* Declares interface for objects in composition.
* Implements default behavior for the interface common to all classes as appropriate.
* Declares an interface for accessing and managing its child components.

##### 2) Leaf

* Represents leaf objects in composition. A leaf has no children.
* Defines behavior for primitive objects in the composition.

##### 3) Composite

* Defines behavior for components having children.
* Stores child component.
* Implements child related operations in the component interface.

##### 4) Client

* Manipulates objects in the composition through the component interface.



## Implementation

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.



## Step 1

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;

 // constructor

 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+" ]");

 }

}

## Step 2

Use the *Employee* class to create and 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);

 //print all employees of the organization

 System.out.println(CEO);

 for (Employee headEmployee : CEO.getSubordinates()) {

 System.out.println(headEmployee);

 for (Employee employee : headEmployee.getSubordinates()) {

 System.out.println(employee);

 }

 }

 }

}

## Step 3

Verify the 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 ]

#### UML for Composite Pattern