**Composite Design Pattern**

**Assignment - 1**

Name - Heet Dobariya Roll No. - 22BCP177 Group - G5

* **Composite Design :**

The Composite Design Pattern is a structural design pattern that allows us to compose objects into tree-like structures to represent part-whole hierarchies. This pattern creates a unified interface for individual objects and compositions of objects, allowing clients to treat both in a uniform manner.

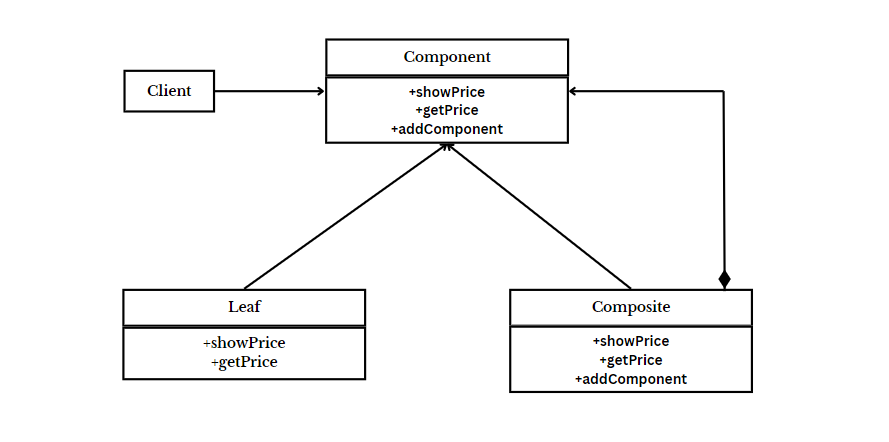
In the Composite pattern, there are three main components:

1. Component: This is the common interface or abstract class that represents both individual objects and compositions. It declares the common operations that can be performed on both types of objects.
2. Leaf: This represents individual objects in the composition. Leaf objects are the end nodes of the tree structure and do not have any children.
3. Composite: This represents the composite objects that contain leaf objects or other composite objects as children. Composite objects implement the operations defined in the component interface, delegating to their children as necessary.

The key idea behind the Composite pattern is that clients can interact with individual objects or compositions of objects uniformly through the component interface. This simplifies client code, as it does not need to distinguish between different types of objects in the hierarchy.

The Composite pattern is commonly used in scenarios where we have tree-like structures and want to treat individual objects and compositions of objects in a uniform manner. Examples include graphical user interfaces, file systems, organization hierarchies, and any other hierarchical structures where you need to represent part-whole relationships.

* **Program :** Implement composite design pattern for the computer as composite component.
* **UML Diagram :**



* **Code :**

public interface Component

{

void showPrice();

int getPrice();

}

public class Leaf implements Component

{

int price;

String name;

Leaf(String name, int price)

{

this.price = price;

this.name = name;

}

@Override

public void showPrice()

{

System.out.println("Leaf -> "+name+" : "+price);

}

@Override

public int getPrice()

{

return price;

}

}

import java.util.\*;

public class Composite implements Component

{

String name;

List<Component> components = new ArrayList();

public Composite(String name)

{

super();

this.name = name;

}

public void addComponent(Component com)

{

components.add(com);

}

@Override

public int getPrice()

{

int p=0;

for(Component c:components)

{

p += c.getPrice();

}

return p;

}

@Override

public void showPrice()

{

System.out.println("Composite -> "+name+" : Price -> "+getPrice());

System.out.println("Leaf of " +name);

for(Component c:components)

{

c.showPrice();

}

}

}

public class CompositeTest

{

public static void main(String[] args)

{

Component hd = new Leaf("Hard Drive", 4000);

Component mouse = new Leaf("Mouse", 500);

Component monitor = new Leaf("Monitor", 8000);

Component ram = new Leaf("RAM", 2000);

Component cpu = new Leaf("CPU", 10000);

Composite ph = new Composite("Peripheral");

Composite cabinet = new Composite("Cabinet");

Composite mb = new Composite("MotherBoard");

Composite computer = new Composite("Computer");

mb.addComponent(cpu);

mb.addComponent(ram);

ph.addComponent(mouse);

ph.addComponent(monitor);

cabinet.addComponent(hd);

cabinet.addComponent(mb);

computer.addComponent(ph);

computer.addComponent(cabinet);

ram.showPrice();

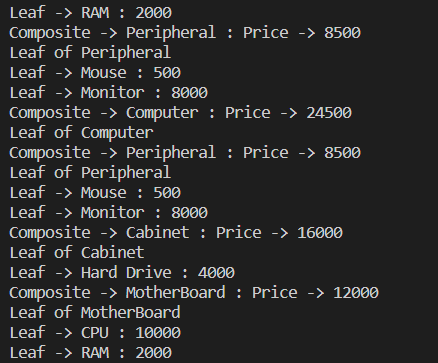
ph.showPrice();

computer.showPrice();

}

}

* **Output:**

****