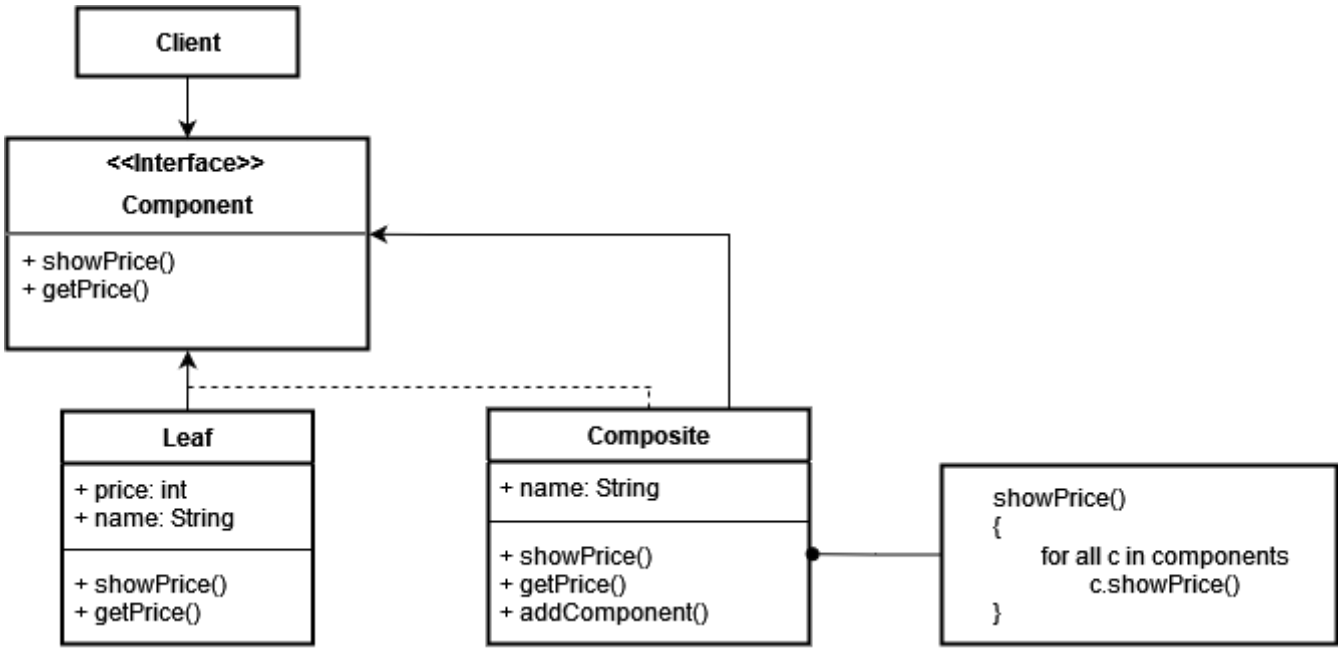
Assignment 6: Composite Design Pattern

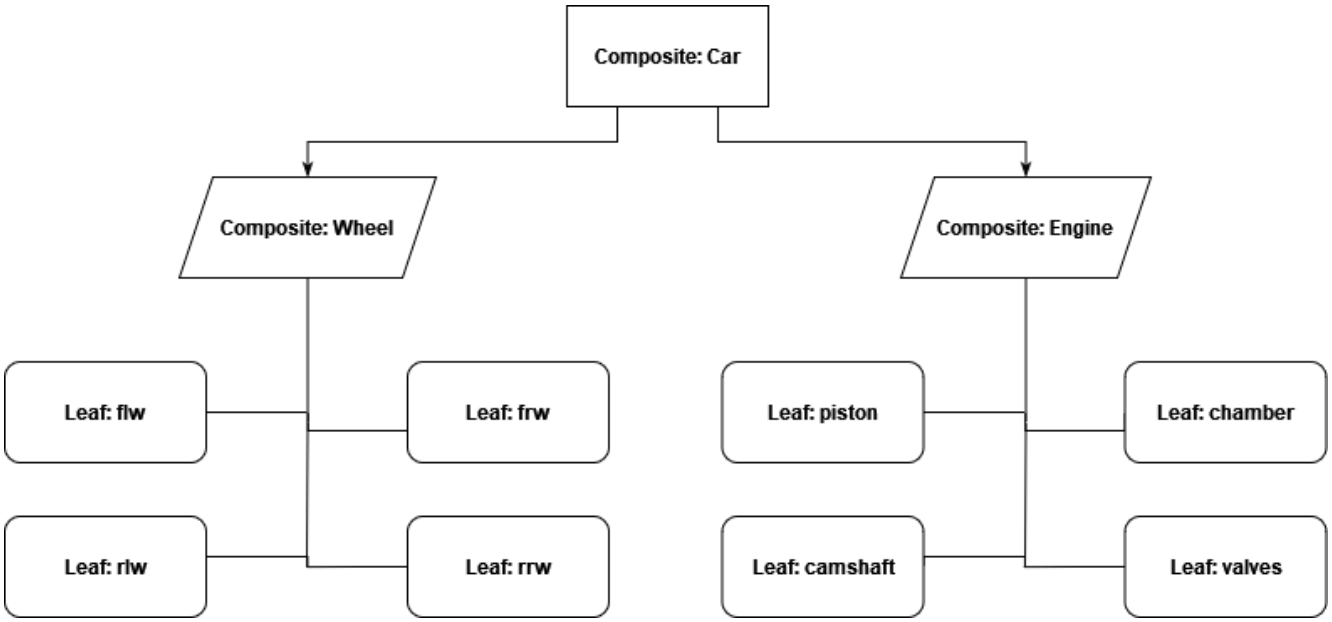
What is Composite Design Pattern?

Composite is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects.

Class Diagram



Object Diagram



Implementation (Code)

- **Component.java**

```
interface Component {
   void showPrice();
   int getPrice();
}
class Leaf implements Component {
   int price;
   String name;
   Leaf(String name, int price){
      this.name = name;
      this.price = price;
   }
   public void showPrice(){
      System.out.println("Leaf -> " + name + " : " + price);
   }
   public int getPrice() {  return price;  }
}
```

- **Composite.java**

```
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);
   }
   public int getPrice(){
      int p=0;
      for (Component c : components){
         p += c.getPrice();
      }
      return p;
   }
   public void showPrice(){
      System.out.println("\nComposite -> " + name + " : Price " + getPrice());
      System.out.println("Leaf of " + name + "\n[");
      for (Component c : components){
         c.showPrice();
      }
      System.out.println("]");
   }
}
```

- **CompositePattern.java**

```
public class CompositePattern {
   public static void main(String[] args) {
      Component flw = new Leaf("Front Left Wheel", 4000);
      Component frw = new Leaf("Front Right Wheel", 4500);
      Component rlw = new Leaf("Rear Left Wheel", 5000);
      Component rrw = new Leaf("Rear Right Wheel", 4000);
      Component piston = new Leaf("Piston", 9000);
      Component chamber = new Leaf("Combustion Chamber", 8000);
      Component camshaft = new Leaf("Cam Shaft", 10000);
      Component valves = new Leaf("Valves", 8500);

      Composite wheel = new Composite("Wheel");
      Composite engine = new Composite("Engine");
      Composite car = new Composite("Car");

      wheel.addComponent(flw);
      wheel.addComponent(frw);
      wheel.addComponent(rlw);
      wheel.addComponent(rrw);
      engine.addComponent(piston);
      engine.addComponent(chamber);
      engine.addComponent(camshaft);
      engine.addComponent(valves);

      car.addComponent(wheel);
      car.addComponent(engine);

      car.showPrice();

   }
}
```

Output:

```
Composite -> Car : Price 53000
Leaf of Car
[

Composite -> Wheel : Price 17500
Leaf of Wheel
[
Leaf -> Front Left Wheel : 4000
Leaf -> Front Right Wheel : 4500
Leaf -> Rear Left Wheel : 5000
Leaf -> Rear Right Wheel : 4000
]
```

```
Composite -> Engine : Price 35500
Leaf of Engine
[
Leaf -> Piston : 9000
Leaf -> Combustion Chamber : 8000
Leaf -> Cam Shaft : 10000
Leaf -> Valves : 8500
]
]
```

Applicability

1. **Use the Composite pattern when you have to implement a tree-like object structure.**
   The Composite pattern provides you with two basic element types that share a common interface: simple leaves and complex containers. A container can be composed of both leaves and other containers. This lets you construct a nested recursive object structure that resembles a tree.
2. **Use the pattern when you want the client code to treat both simple and complex elements uniformly.**
   All elements defined by the Composite pattern share a common interface. Using this interface, the client doesn't have to worry about the concrete class of the objects it works with.