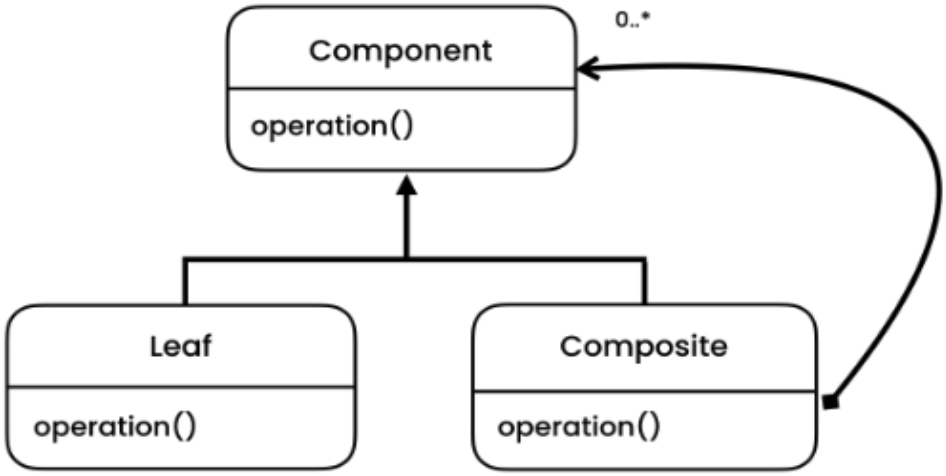
Composite Pattern:

Definition: Represents object hierarchies where individual objects and compositions of objects are treated the same way.

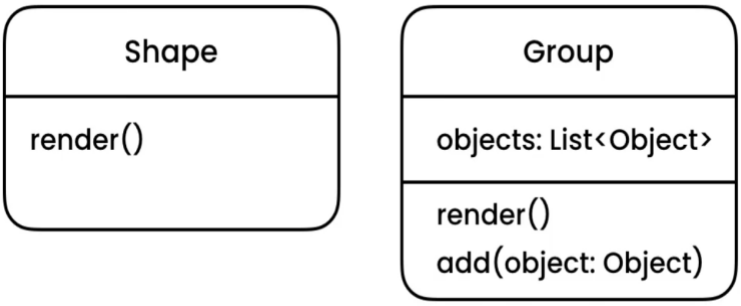


Scenario: We have a group of different shapes e.g. circle, square. If we want to resize 2 squares at a time, we will make its group, same with circle. Now, if we want to resize the whole group of shape in one go, we need to resize the group instead of every individual shape.

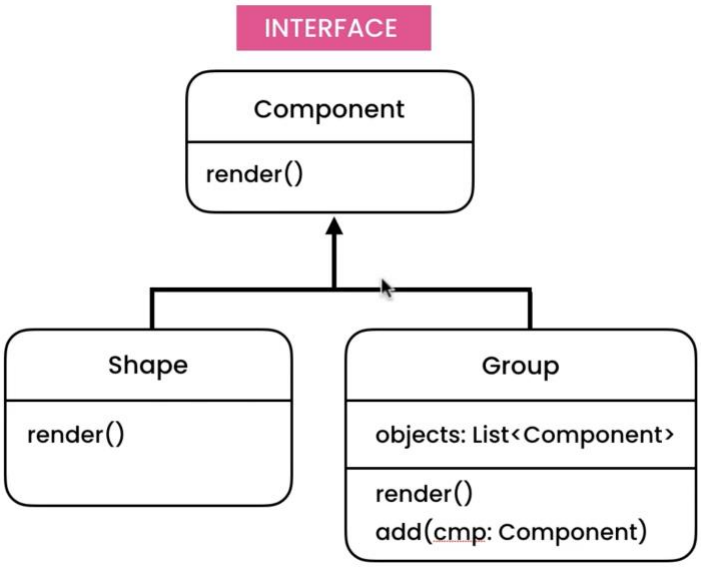
Implementation:

```
public class Group {
   private List<Object> objects = new ArrayList<>();
   public void add(Object shape){
      objects.add(shape);
   }
   public void render(){
      for (var object : objects){
         if(object instanceof Shape)
            ((Shape) object).render();
         else
            ((Group)object).render();
      }
   }
}
```

Problem: We have to cast every time, it must be generic.



Solution: Introduce component interface. Bind Groups with interface (with all common methods) instead of concrete classes and let polymorphism play its role so that we don't need to explicitly cast every time.




```
public interface Component {
    void render();
}
public class Group implements Component{
    private List<Component> components = new ArrayList<>();
    public void add(Component shape){
        components.add(shape);
    }
    public void render(){
        for (var component : components)
            component.render();
    }
}
```