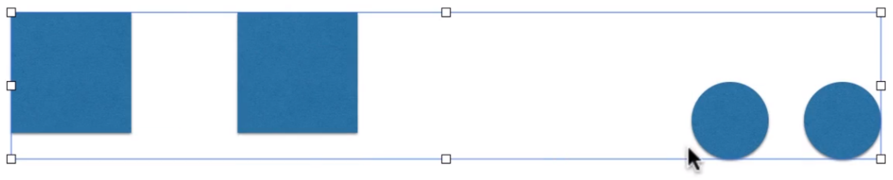
Composite Pattern

If you have a hierarchy and you want to treat the objects in this heirarchy the same way you should use the Composite Design Pattern.

Example Problem - you wish to represent objects in a hieracrchy e.g. in Microsoft Powerpoint you have drawn four shapes two circles which you then group together and two squares which you group togteher you then group both groups together creating a Main group containing two sub groups and four object which can all be resized and moved together. This is quite similar to how the file system on your PC works.



Represents in words this hierarchy becomes :

Group

Group1

Square

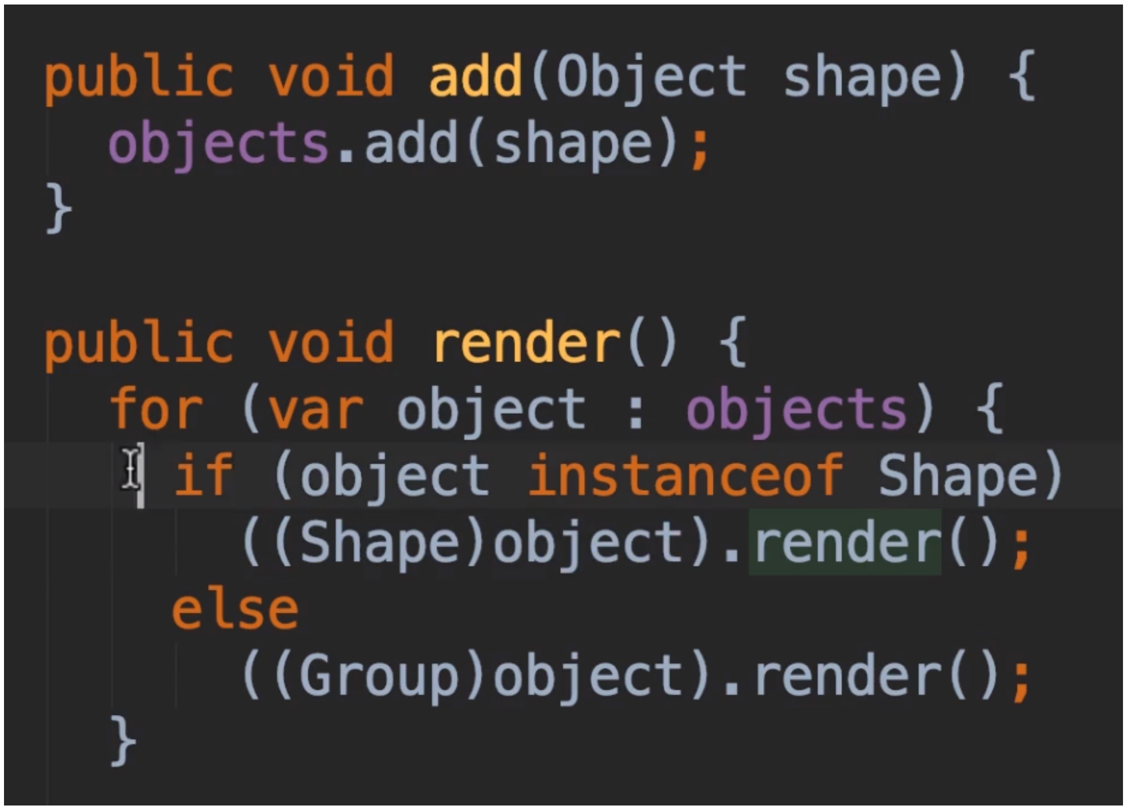
Square

Group2

Circle

Circle

One implemenation may be to have the add method of Group class take an object rather than a shape which will allow it to accept Groups as well as shapes in the Hirarchy. the downside of this is taht in the render method you will have to provide logic to decde on what to do for both shapes and groups which will have to be cast. this same logic will have t o be repeated in every method of the group class such as resize or remove :



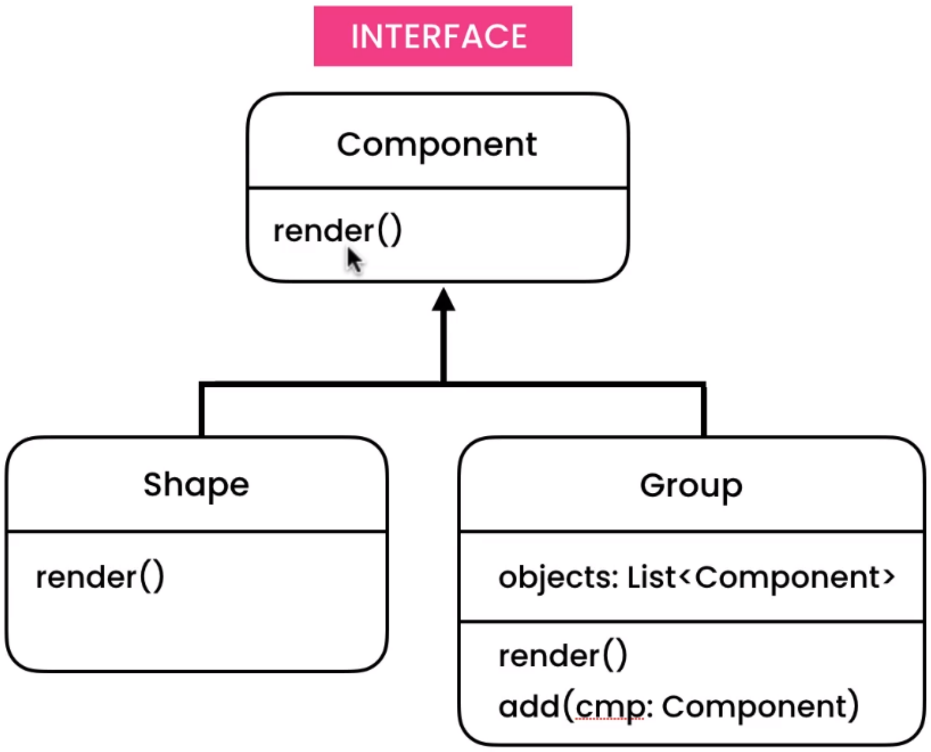
```
public void add(Object shape) {
  objects.add(shape);
}

public void render() {
  for (var object : objects) {
    if (object instanceof Shape)
      ((Shape)object).render();
    else
      ((Group)object).render();
  }
```

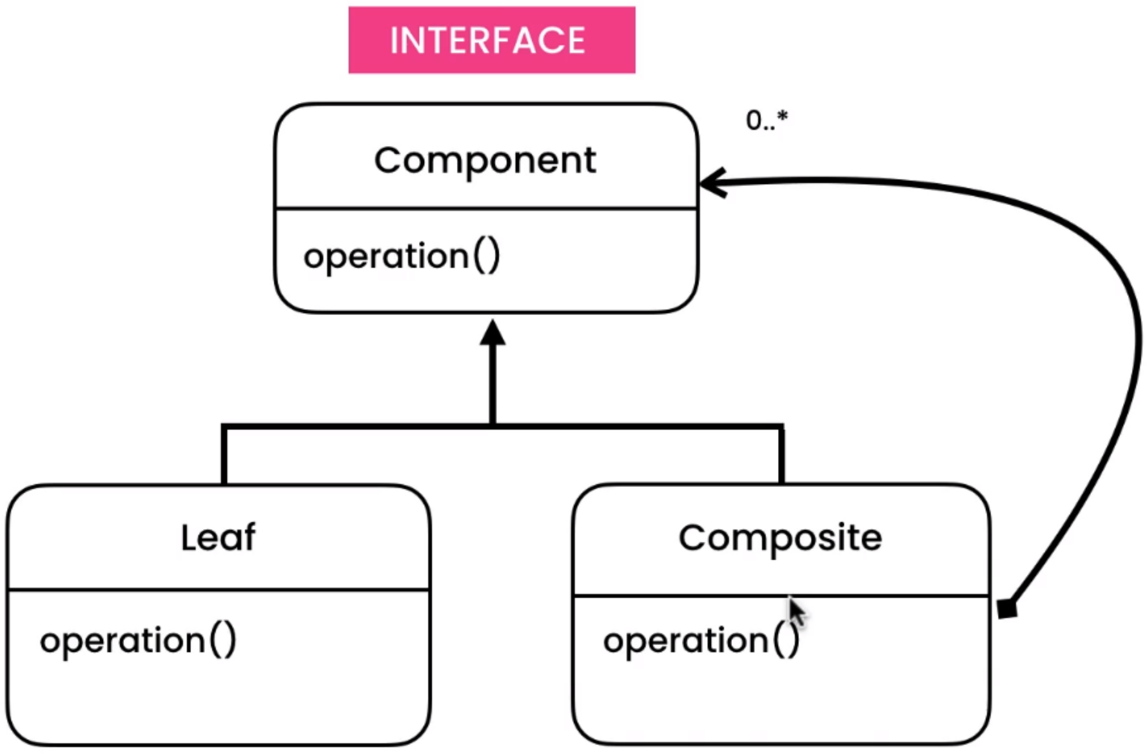
The composite design pattern allows us to treat group and shape objects in the same way :

Solution ;

Extract the common actions to a parent class and make use oof inheritanve and polymorphism to enable child clases to nbe treated as one in the hierarchy



The render methods is available to both parts and components (shapes and groups in our case) so it is extracted to an interface, e.g. in your file system the file is a part and the foler in a component, if you delte a foler you want all the files in that folder to eb deleted recursively. As per the GOF book each Composite (Group) class can be constructed of 0 or more Components (groups or shapes/files) :



public interface
Component {
  void render();
 void move();
}
	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();
	 }
	
	  @Override
	  public void move() {
	    for (var component :
	components)
	      component.move();
	 }
	}
		public class Shape impl
		ements Component {
		  @Override
		  public void render()
		{
		    System.out.println(
		"Render Shape");
		 }
		
		  @Override
		  public void move() {
		    System.out.println(
		"Move Shape");
		 }
		}
			public class Main {
			  @Override
			  public static void main
			() {
			
			 var group1 = new
			Group();
			
			group1.add(new Shape()); //square
			
			group1.add(new Shape()); //square
			
			var group2 = new Group();
			
			group2.add(new Shape()); //circle
			
			group2.add(new Shape()); //circle
			
			var group = new Group();
			
			group.add(group1);
			
			group.add(group2);
			
			group.render();
			
			//outputs Render shape 4x
			
			 }
			
			}