**Composite Design Pattern**

[](https://cdn.journaldev.com/wp-content/uploads/2013/07/composite-design-pattern.jpg)

When we need to create a structure in a way that the objects in the structure has to be treated the same way, we can apply composite design pattern.

Lets understand it with a real life example – A diagram is a structure that consists of Objects such as Circle, Lines, Triangle etc. When we fill the drawing with color (say Red), the same color also gets applied to the Objects in the drawing. Here drawing is made up of different parts and they all have same operations.

Composite Pattern consists of following objects.

1. **Base Component** – Base component is the interface for all objects in the composition, client program uses base component to work with the objects in the composition. It can be an interface or an [**abstract class**](https://www.journaldev.com/1582/abstract-class-in-java) with some methods common to all the objects.
2. **Leaf** – Defines the behaviour for the elements in the composition. It is the building block for the composition and implements base component. It doesn’t have references to other Components.
3. **Composite** – It consists of leaf elements and implements the operations in base component.

Here I am applying composite design pattern for the drawing scenario.

**Composite Pattern Base Component**

Composite pattern base component defines the common methods for leaf and composites. We can create a class Shape with a method draw(String fillColor) to draw the shape with given color.

Shape.java

public interface Shape {

public void draw(String fillColor);

}

**Composite Design Pattern Leaf Objects**

Composite design pattern leaf implements base component and these are the building block for the composite. We can create multiple leaf objects such as Triangle, Circle etc.

Triangle.java

public class Triangle implements Shape {

@Override

public void draw(String fillColor) {

System.out.println("Drawing Triangle with color "+fillColor);

}

}

Circle.java

public class Circle implements Shape {

@Override

public void draw(String fillColor) {

System.out.println("Drawing Circle with color "+fillColor);

}

}

**Composite object**

A composite object contains group of leaf objects and we should provide some helper methods to add or delete leafs from the group. We can also provide a method to remove all the elements from the group.

Drawing.java

import java.util.ArrayList;

import java.util.List;

public class Drawing implements Shape{

//collection of Shapes

private List<Shape> shapes = new ArrayList<Shape>();

@Override

public void draw(String fillColor) {

for(Shape sh : shapes)

{

sh.draw(fillColor);

}

}

//adding shape to drawing

public void add(Shape s){

this.shapes.add(s);

}

//removing shape from drawing

public void remove(Shape s){

shapes.remove(s);

}

//removing all the shapes

public void clear(){

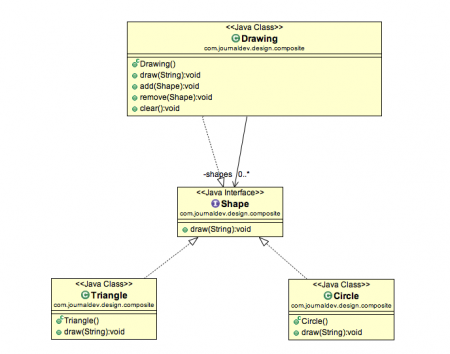
System.out.println("Clearing all the shapes from drawing");

this.shapes.clear();

}

}

Notice that composite also implements component and behaves similar to leaf except that it can contain group of leaf elements.

[](https://cdn.journaldev.com/wp-content/uploads/2013/07/Composite-Pattern-java.png)

Our composite pattern implementation is ready and we can test it with a client program.

**Composite Design Pattern Client Program**

TestCompositePattern.java

public class TestCompositePattern {

public static void main(String[] args) {

Shape tri = new Triangle();

Shape tri1 = new Triangle();

Shape cir = new Circle();

Drawing drawing = new Drawing();

drawing.add(tri1);

drawing.add(tri1);

drawing.add(cir);

drawing.draw("Red");

drawing.clear();

drawing.add(tri);

drawing.add(cir);

drawing.draw("Green");

}

}

Output of the above composite pattern client program is:

Drawing Triangle with color Red

Drawing Triangle with color Red

Drawing Circle with color Red

Clearing all the shapes from drawing

Drawing Triangle with color Green

Drawing Circle with color Green

**Composite Pattern Important Points**

* Composite pattern should be applied only when the group of objects should behave as the single object.
* Composite design pattern can be used to create a tree like structure.

java.awt.Container#add(Component) is a great example of Composite pattern in java and used a lot in Swing.