Composite Design Pattern

When we want to represent part-whole hierarchy, use tree structure and compose objects. We know tree structure what a tree structure is and some of us don't know what a part-whole hierarchy is. A system consists of subsystems or components. Components can further be divided into smaller components. Further smaller components can be divided into smaller elements. This is a part-whole hierarchy.

Everything around us can be a candidate for part-whole hierarchy. Human body, a car, a computer, lego structure, etc. A car is made up of engine, tyre, … Engine is made up of electrical components, valves, … Electrical components is made up of chips, transistor, … Like this a component is part of a whole system. This hierarchy can be represented as a tree structure using composite design pattern.

"Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly." is the intent by GoF.

Real World Example

In this article, let us take a real world example of part-whole hierarchy and use composite design pattern using java. As a kid, I have spent huge amount of time with lego building blocks. Last week I bought my son an assorted kit lego and we spent the whole weekend together building structures.
Let us consider the game of building blocks to practice composite pattern. Assume that our kit has only three unique pieces (1, 2 and 4 blocks) and let us call these as primitive blocks as they will be the end nodes in the tree structure. Objective is to build a house and it will be a step by step process. First using primitive blocks, we should construct multiple windows, doors, walls, floor and let us call these structures. Then use all these structure to create a house.

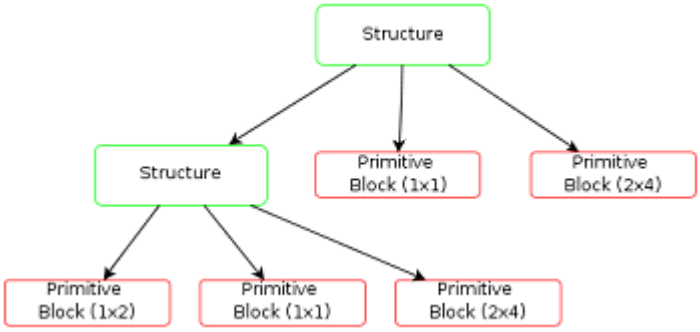
Primitive blocks combined together gives a structure. Multiple structures assembled together gives a house.

Important Points

- Importance of composite pattern is, the group of objects should be treated similarly as a single object.
- Manipulating a single object should be as similar to manipulating a group of objects. In sync with our example, we join primitive blocks to create structures and similarly join structures to create house.
- Recursive formation and tree structure for composite should be noted.
- Clients access the whole hierarchy through the components and they are not aware about if they are dealing with leaf or composites.

Tree for Composite

When we get a recursive structure the obvious choice for implementation is a tree. In composite design pattern, the part-whole hierarchy can be represented as a tree. Leaves (end nodes) of a tree being the primitive elements and the tree being the composite structure.



Uml Design for Composite Pattern

Component: (structure)

1. Component is at the top of hierarchy. It is an abstraction for the composite.
2. It declares the interface for objects in composition.
3. (optional) defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.

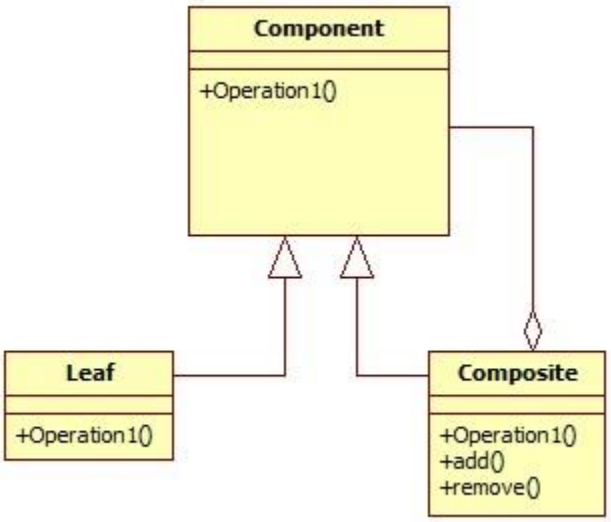
Leaf: (primitive blocks)

1. The end nodes of the tree and will not have any child.

2. Defines the behaviour for single objects in the composition

Composite: (group)

1. Consists of child components and defines behaviour for them
2. Implements the child related operations.



Composite Pattern Implementation

```
package com.javapapers.designpattern.composite;

public class Block implements Group {

    public void assemble() {
        System.out.println("Block");
    }
}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

package com.javapapers.designpattern.composite;

public interface Group {
    public void assemble();
}
package com.javapapers.designpattern.composite;

import java.util.ArrayList;
import java.util.List;

public class Structure implements Group {
          // Collection of child groups.
          private List groups = new ArrayList();

          public void assemble() {
                    for (Group group : groups) {
                              group.assemble();
                    }
          }

          // Adds the group to the structure.
          public void add(Group group) {
                    groups.add(group);
          }

          // Removes the group from the structure.
          public void remove(Group group) {
                    groups.remove(group);
          }
}

package com.javapapers.designpattern.composite;
```

```
public class ImplementComposite {
          public static void main(String[] args) {
                //Initialize three blocks
                Block block1 = new Block();
                Block block2 = new Block();
                Block block3 = new Block();

                //Initialize three structure
                Structure structure = new Structure();
                Structure structure1 = new Structure();
                Structure structure2 = new Structure();

                //Composes the groups
                structure1.add(block1);
                structure1.add(block2);

                structure2.add(block3);

                structure.add(structure1);
                structure.add(structure2);

                structure.assemble();
            }
}
```