**Composite Pattern**

## Definition

Build a complex object out of elemental objects and itself like a tree structure.

Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern comes under structural pattern as this pattern creates a tree structure of group of objects.

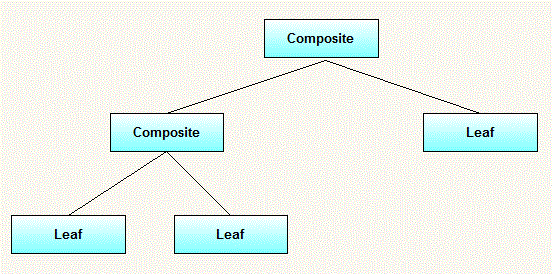
This pattern creates a class contains group of its own objects. This class provides ways to modify its group of same objects.

As described by Gof:

"Compose objects into tree structure to represent part-whole hierarchies. Composite lets client treat individual objects and compositions of objects uniformly".

Composite design pattern treats each node in two ways-Composite or leaf. Composite means it can have other objects below it.leaf means it has no objects below it.

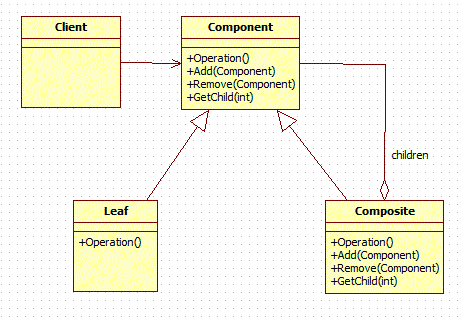
**Tree Structure:**



### When to use it:

* You want to represent part-whole hierarchies of objects.
* You want client to be able to ignore difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

### UML Diagram for Composite design pattern:



### Elements:

* **Component**
  + declares interface for objects in composition.
  + implements default behavior for the interface common to all classes as appropriate.
  + declares an interface for accessing and managing its child components.
* **Leaf**
  + represents leaf objects in the composition. A leaf has no children.
  + defines behaviour for primitive objects in the composition.
* **Composite**
  + defines behaviour for components having children.
  + stores child components.
  + implements child related operations in the component interface.
* **Client**
  + manipulates objects in the composition through the component interface.

### Workflow:

Client use the component class interface to interact with objects in the composition structure. If recipient is a leaf then request is handled directly. If recipient is a composite, then it usually forwards request to its child components, possibly performing additional operations before and after forwarding.

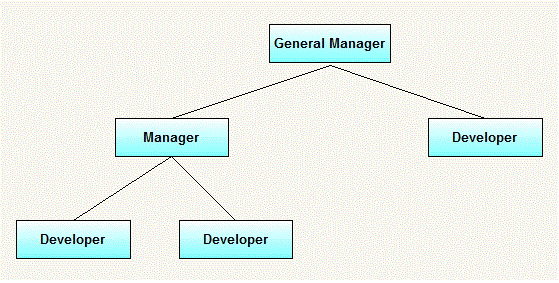
### Recursion:

What makes the Composite pattern one of the most beautiful is the power of recursion. I can explain this with the same organization example. You want to find the total salary paid to all employees of the organization. It is nothing but the salary of CEO + the salary paid to all the departments. What is the salary of a department? It is the salary of the department head + the salary of all projects. What is the total salary of a project? It is the salary of the project manager + the salary of all the project members. In short, the salary of anything is the salary of self + the salary of all its sub groups.

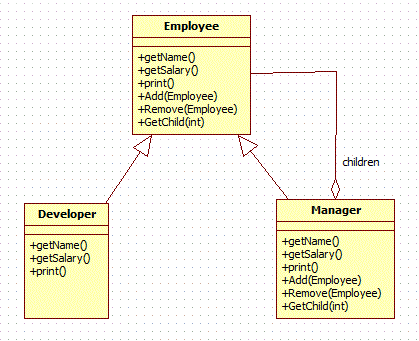
### Example:

In a small organization, there are 5 employees. At top position, there is 1 general manager. Under general manager, there are two employees, one is manager and other is developer and further manager has two developers working under him. We want to print name and salary of all employees from top to bottom.

**Tree structure for example:**

[](http://3.bp.blogspot.com/-OVvv9oMZI7k/UGMw1UvumOI/AAAAAAAAAak/EsBioEmjKSs/s1600/CompositeDesignPatternExampleHierarchy.gif)

**UML diagram for above example:**

[](http://4.bp.blogspot.com/-hxzn_ztVv5g/UGNJlLm1HPI/AAAAAAAAAbI/gkr2g8P-oP0/s1600/CompositeDesignPatternExample.gif)

Comparing from above generic elements. Our example consists of following elements.

* Manager(Composite)
* Developer(Leaf)
* Employee(Component)

**Where to use & benefits**

* Want to represent a part-whole relationship like tree folder system
* Group components to form larger components, which in turn can be grouped to form still larger components.
* Related patterns include

* + [Decorator](http://www.javacamp.org/designPattern/decorator.html), which is often used with composite pattern and with the same parent class.
  + [Flyweight](http://www.javacamp.org/designPattern/flyweight.html), which is used with composite pattern to share components.
  + [Iterator](http://www.javacamp.org/designPattern/iterator.html), which is used to traverse the composites.
  + [Visitor](http://www.javacamp.org/designPattern/visitor.html), which localizes operations across composite and leaf classes.

**Disadvantages:**

* Once tree structure is defined, composite design makes tree overly general.
* Leaf classes have to create some methods which has to empty.