**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.

**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.

## Example

A component has many elements and itself which has many elements and itself, etc. A file system is a typical example. Directory is a composite pattern. When you deal with Directory object, if isFile() returns true, work on file, if isDirectory() returns true, work on Directory object.

class Directory {

Directory dir;

File[] f;

...

boolean isDirectory() {

return f == null;

}

boolean isFile() {

return f != null;

}

File getFile(int i) {

if (isFile())

return f[i];

return null'

}

Directory getDirectory() {

if (isDirectory())

return dir;

return null;

}

....

}