Decorator

Pizza
getDesc();

PlainPizza
getDesc
return "plain"

ToppingDeco
constructor : this.pizza=pizza
getDesc
return pizza.getDesc()

Mozzerela
Constructor(Pizza pizza)
super(pizza)
getDesc
return this.pizza.getDesc() + "Mozzerela"

Tomato
Constructor(Pizza pizza)
super(pizza)
getDesc
return this.pizza.getDesc() + "Tomato"

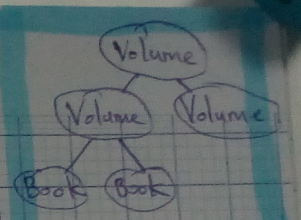
```
Pizza aPizza = new Tomato(
                   new Mozzerela(
                   new PlainPizza
                   )
                   )

aPizza.getDesc()
```

Composition is better than Inheritance
With inheritance, if top class changes, everything below need to be refactored.

Composite

Node
+ void add (Node node)
+ void remove (Node node)
+ Node getNode(int id) ← for node
+ String getName() ← for leaf
+ void displayInfo()



node **Volume**
- Node nodes
- add(Node node)
  nodes.add(node)
- remove(Node node)
  nodes.remove(node)
- getNode(int id)
  return nodes.get(id)
- displayInfo()
  foreach nodes as node
  node.displayInfo()

leaf **Book**
- name
- getName()
  return name
- displayInfo()
  echo name

```
Node vol1 = new Volume
vol1.add(new Volume)
vol1.add(new Book)
....
```

They usually go hand in and hand. In that using the composite pattern often leads to also using the decorator pattern.

The composite pattern allows you to build a hierarchical structure (such as a tree of elements) in a way that allows your external code to view the entire structure as a single entity. So the interface to a leaf entity is exactly the same as the entity for a compound entity. So the essence is that all elements in your composite structure have the same interface even though some are leaf nodes and others are entire structures. User interfaces often use this approach to allow easy composability.

http://en.wikipedia.org/wiki/Composite_pattern

The decorator pattern allows an entity to completely contain another entity so that using the decorator looks identical to the contained entity. This allows the decorator to modify the behaviour and/or content of whatever it is encapsulating without changing the outward appearance of the entity. For example, you might use a decorator to add logging output on the usage of the contained element without changing any behaviour of the contained element.

http://en.wikipedia.org/wiki/Decorator_pattern