**Decorator Design Pattern**

Decorator design pattern is used to add additional responsibilities to an object dynamically.

**Intent**

* Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

The intent is to provide flexible alternative to subclassing by adding responsibilities to an object dynamically.

**Problem**

* Need to add additional functionality to object. Subclassing makes it inflexible and complex.

We may require to add some additional functionality to the object. This may be done by using inheritance but that will make it static and inflexible and client will not be able to add functionality as it is required.

**Solution**

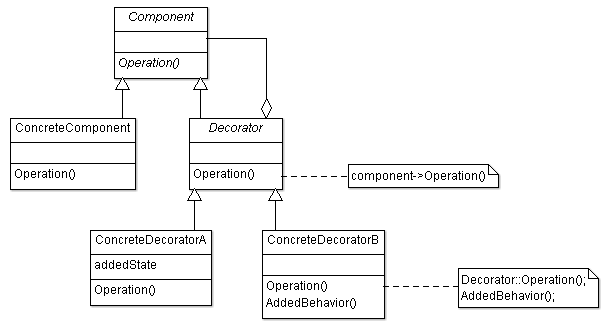
* Give the responsibility of adding additional functionality to another object without using subclassing.

So additional functionality request will be forwarded to this object which will be responsible for adding the required functionality.

**Where it is applicable?**

* Adding additional responsibility to an object dynamically.
* To avoid subclassing for extending functionality.

**Structure**



**Participant classes**

* **Component** class provides the interface to client for adding functionality dynamically.
* **ConcreteComponent** class represents the object where additional functionality will be added.
* **Decorator** class is derived from Component and has the responsibility for adding additional functionality. It has instance of class Component.
* The concrete classes **ConcreteDecoratorA** and **ConcreteDecoratorB** are used for adding functionality to the Component object.

**How they work together?**

* Decorator does the Operation on ConcreteComponent as it has instance of class Component. Extra functionality will be added by method Operation() of ConcreteDecoratorA and ConcreteDecoratorB.
* This looks better than the subclassing, functionalities are added dynamically and provide more flexibility. But this increases the number of objects as the functionality increases and will make it difficult to manage. So inheritance may work better if we have very less number of functionality or functionality need not require to be added dynamically.