Adding behavior or Enhancing without altering the class itself

Want to augment an object with additional functionality

Do not want to rewrite or alter existing code (**OCP**)

Want to keep new functionality separate (**SRP**)

Need to be able to interact with existing structures

Two Options:

* Inherit from the required object if possible as some classes are final
* Build a Decorator, which simply references the decorated object(s)

So the Decorator facilitates the addition of behaviors to individual objects without inheriting from them. What problems can it solve?

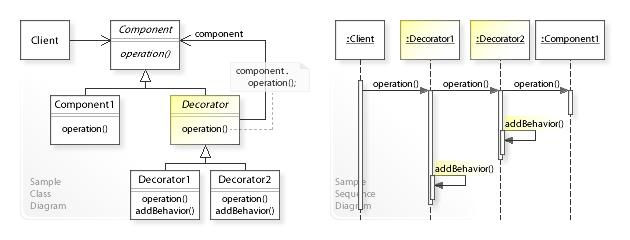
* Responsibilities should be added to (and removed from) an object dynamically at run-time.
* A flexible alternative to subclassing for extending functionality should be provided.

**Decorator design pattern** is used to modify the functionality of an object at runtime. At the same time other instances of the same class will not be affected by this, so individual object gets the modified behavior.

**GOF Definition**

Allows for the dynamic wrapping of objects in order to modify their existing responsibilities and behaviors

**Class Diagram**



Following are the participants of the Decorator Design pattern:

* **Component** – this is the wrapper which can have additional responsibilities associated with it at runtime.
* **Concrete component**– is the original object to which the additional responsibilities are added in program.
* **Decorator**-this is an abstract class which contains a reference to the component object and also implements the component interface.
* **Concrete decorator**-they extend the decorator and builds additional functionality on top of the Component class.

**Decorator vs Visitor**

Well, they are actually as different as they can be!

You use *Decorator* when you want to enhance existing object with some new, more-or-less transparent functionality like validation or caching. See example here: [Should I extend ArrayList to add attributes that isn't null?](https://stackoverflow.com/questions/9358821)

*Visitor* on the other hand is used when you have a hierarchy of classes and want to run different method based on concrete type but avoiding instanceof or typeof operators. See real-life example: [Is This Use of the "instanceof" Operator Considered Bad Design?](https://stackoverflow.com/questions/8841577)

Decorator works on an object, Visitor works on composite structure,

*Visitor* works on an inheritance hierarchy, *Composite* is a different GoF design pattern.

Decorator is Structural design pattern, visitor is Behavioral design pattern.

They serve for absolutely different purposes:

1. you will use decorator when you want to dynamically enrich the functionality of objects by providing single elements that decorate other objects so that they do actually add some behavior to them (in fact it is a structural pattern in the sense that it alters the structure of the objects you are working with)
2. You will use visitor when you want to separate an algorithm from the objects it is used with. What you do is that you have this visitor which is passed to a multitude of different objects, usually a hierarchy, (they are said to accept the visitor), and this visitor does specific operations according to the type of objects it is visiting in the specific moment. In this way you can have your visitor do whatever it wants with the specific objects without the need to specify these operations in the objects itself (that's why it is ***behavioral***). It's a sort of having abstract methods that are not defined in the object itself.