By Hamid Gholizadeh (27 Jan 2014)
www.gholizadeh.net

Decorator Patter

Decorator pattern is one of the extension patterns. That means by this pattern, we might provide the possibility of extending a Class functionality. It seems we can do that possibly by inheriting the base class and extending its functionality. It is true! So why we need decorator pattern?!

This pattern comes to the scenario when there is possibility of adding features to (decorating) the normal behavior of the class with many options and their combination. Suppose we have class A as a base; this class can be extended to have addition features f1,f2, and f3, giving rise to classes Af1,Af2,Af3, respectively, all inherited from A. if we require to have combination of these features (say, f1 and f2,) we need to create another classes called Af1f2 in our hierarchy structure. Again for another combination of features we would require to create another class (e.g. Af1f3) and so on. As you see for covering all possible combinations, we would require to create 3!=6 classes. The number of required classes in hierarchy would dramatically increase if the number of features increase (for 6 feature we would require 6!=720 class in the inheritance hierarchy!). This is the place where decorator pattern takes the role and provides a structure that reduce the required subclasses to only the number of feature we require to have. So the possibility of having combination of the features without the necessity of having the corresponding subclasses explicitly in the hierarchy would be the bonus of the pattern usage; e.g. we would only need 3 subclasses in first example and 6 subclasses for the second example.

The name “decoration” is analogues to the way we use this pattern in practice; e.g if we want the behavior of the Class A with two features f1 and f2, we would write something like Class a=new Af2(new Af1(new A))) in our code, resembling that class A is decorated with f1 and f2 features.

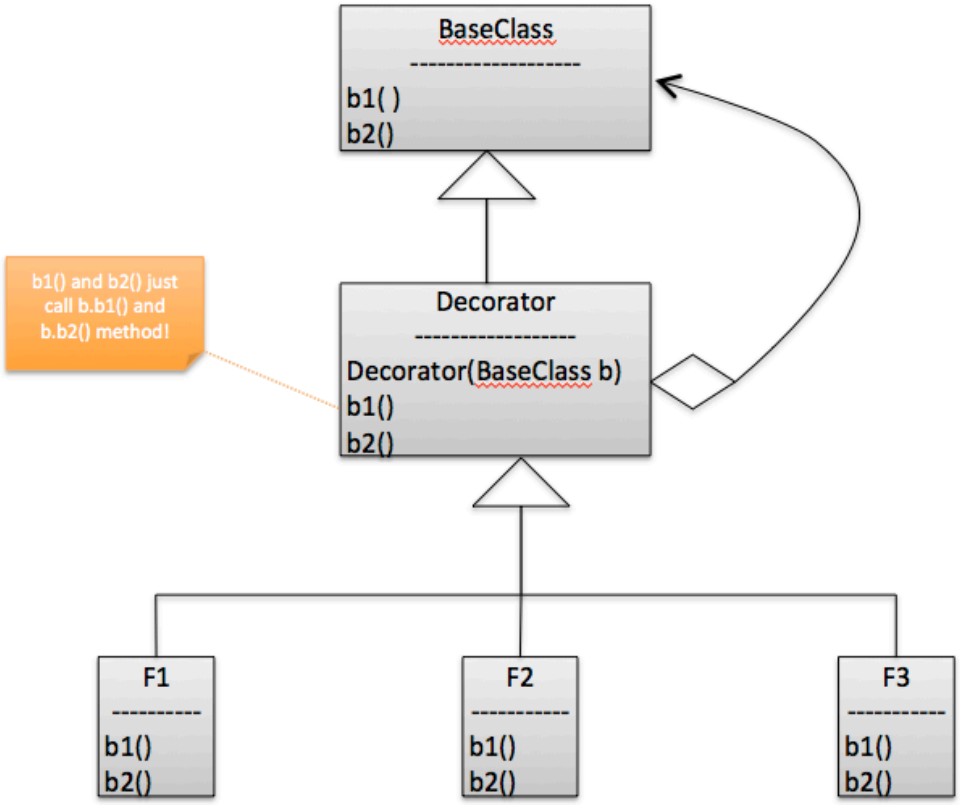

Fig1. Decorator Pattern

Figure above shows the UML diagram for this pattern. The decorator class gets an instance of the base class in its constructor (i.e., b). It calls b's corresponding methods in the body of the respective methods; e.g., in b1() it calls b.b1(). If we would like the invocation like A a=new F1(new F2(new F3)).b1( ) work fine, all inherited classes require to call super.b1() first in there b1 implementation to let the other feature functionalities is also applied!!!