**Strategy**

Define a family of algorithms, encapsulate each one, and make them interchangeable. This allows the algorithm to vary independently from clients that use it.

**Observer**

A *One to Many* relationship; when one object (the “Subject”) changes states, all of the dependent objects (the “Observers”) are notified.

**Decorator**

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

Do this by creating an interface or abstract class, and then implementing/extending it as a concrete component. Also, implement a decorator class from the same interface, and then extend the decorator class with all of the concrete decorators that you need.

The key is that each decorator contains a reference to the base class, which means you can keep wrapping decorators on the outside every time you need to add in additional functionality. The decorator’s behavior consists of performing its own action, and then calling the object which its contains to perform it’s action.