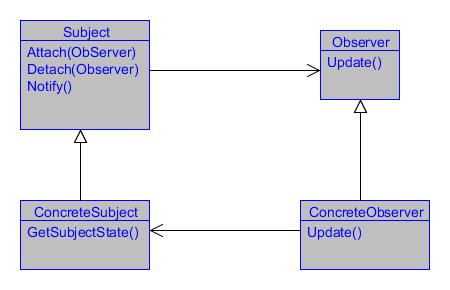
Many a times, we need one part of our application updated with the status of some other part of the application. One way to do this is to have the receiver part repeatedly check the sender for updates, but this approach has two main problems. First, it takes up a lot of CPU time to check the new status and second, depending on the interval we are checking for change we might not get the updates "immediately".

This problem has one easy solution i.e. Observer Pattern. Here is the Class diagram for the Observer pattern (Reference: GoF Design Patterns)

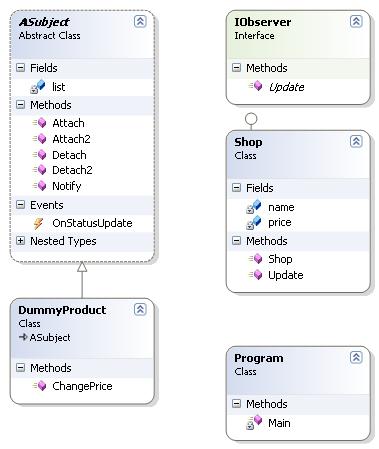


Using the Code

Let us now discuss all the classes one by one:

* Subject: This class keeps a track of all the observers and provides the facility to add or remove the observers. Also it is the class that is responsible for updating the observers when any change occurs. In our solution, we have ASubject implemented for the same purpose.
* ConcreteSubject: This class is the real class that implements the Subject. This class is the entity whose change will affect other objects. We have DummyProject class implemented for the same.
* Observer: This represents an interface that defines the method that should be called whenever there is change. We have implemented this as IObserver.
* ConcreteObserver: This is the class which needs to keep itself updated with the change. This class just needs to implement the Observer and register itself with the ConcreteSubject and it is all set to receive the updates. We have Shop class in our application serving the same purpose.

So let me just have the same diagram I've shown above for my implementation here:



Before we go ahead and see the code, there is one more thing I want to say here. In .NET, we have delegates which are actually a very good example of Observer pattern. Also, we have implemented the delegate's way of having an observer pattern working too. So let's now look at the code.

Source: CodeProject