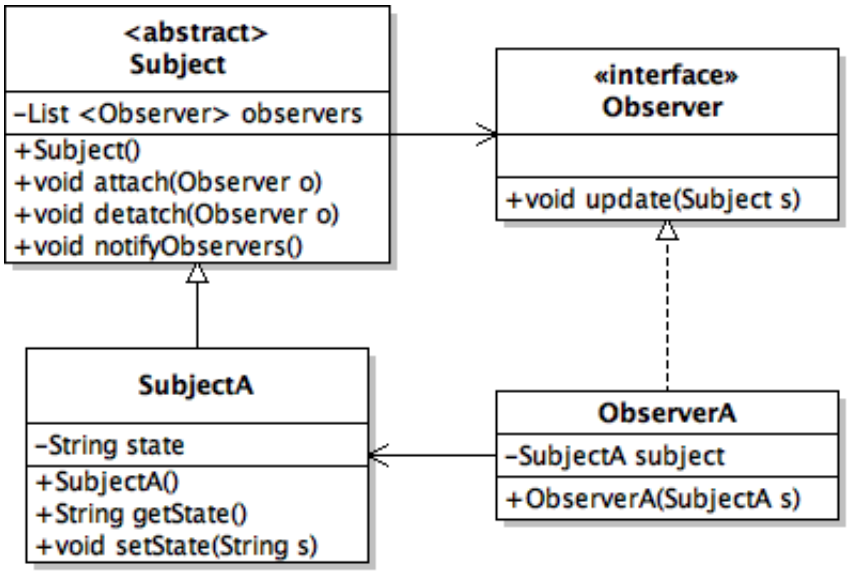
CECS 277 – Lecture 25 – Design Patterns

Observer

The Observer pattern is a Behavioral design pattern that allows an object to notify other objects whenever its state has changed. This type of interaction is also known as publish-subscribe, a subject object publishes notifications about its state change and automatically sends them out to each subscribing object. It is not always necessary to fully implement the Observer pattern since java.util has interface Observer and class Observable built in.

Observer UML – The Observer pattern has four main classes: The Subject is an abstract class that keeps a list of observers that it will notify whenever its state changes. Concrete Subject classes keep track of the state the object is in and notifies observers if the state changes. The Observer interface (or abstract class) has an update() method that is called by Subject when its state changes. Concrete observers contain an instance of the Subject that they are observing so that when the notification comes in that the state has changed, they may call the Subject's methods to find out more about the Subject's updated state.



Observer Template Code:

```
public interface Observer {
   public void update( Subject s );
}
public class ObserverA implements Observer {
   private SubjectA subject;
   public ObserverA( SubjectA s ) {
      subject = s;
      subject.attatch( this );
   }
   @Override public void update( Subject s ) {
      if( subject == s ) {
         display( );
      }
   }
```

```
   public void display( ) {
      System.out.println("OA: SubA:" + subject.getState());
   }
}
public class ObserverB implements Observer {
   private SubjectA subject;
   public ObserverB( SubjectA s ) {
      subject = s;
      subject.attatch( this );
   }
   @Override
   public void update( Subject s ) {
      if( subject == s ) {
         display( );
      }
   }
   public void display( ) {
      System.out.println("OB: SubA:" + subject.getState());
   }
}
public abstract class Subject {
   private List<Observer> observers = new ArrayList<Observer>();
   public void attach( Observer o ) {
      observers.add( o );
   }
   public void detatch( Observer o ) {
      observers.remove( o );
   }
   public void notifyObservers( ) {
      for( Observer o : observers ) {
         o.update( this );
      }
   }
}
public class SubjectA extends Subject {
   private String state;
   public SubjectA( ) {
      state = "A";
   }
   public String getState( ) {
      return state;
   }
   public void setState( String s ) {
      state = s;
      notifyObservers( );
   }
}
```

```
public class Main {
   public static void main( String [] args ) {
      SubjectA subA = new SubjectA( );
      ObserverA obsA = new ObserverA( subA );
      ObserverB obsB = new ObserverB( subA );
      subA.setState( “B” );
      subA.detatch( obsB );
      subA.setState( “C” );
   }
}
/* Output:
   OA: SubA: B
   OB: SubA: B
   OA: SubA: C
*/
```