Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

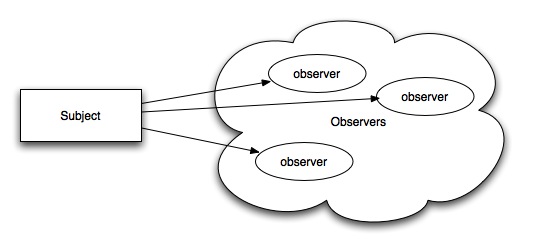
CECS 277

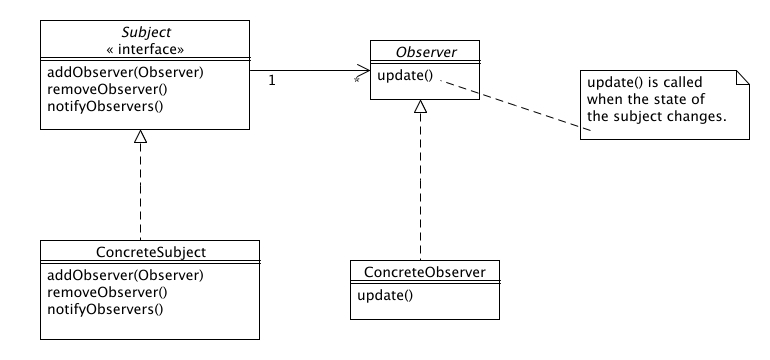
Lab Design Patterns: Observer

**Code located at:** <http://web.csulb.edu/~mopkins/cecs277/code/labObservercode.zip>

The **Observer Pattern** is a  commonly-appearing pattern and has a wide variety of uses. The pattern allows objects to be notified when events occur.

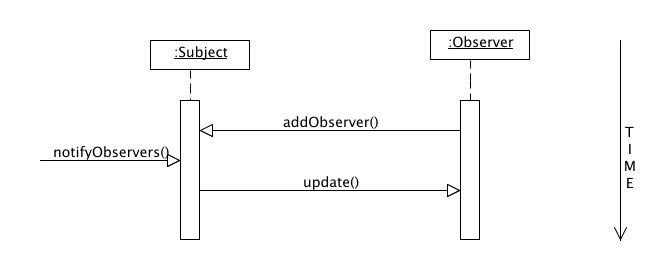
The Observer pattern follows the *Subscriber* model whereby **observers** (the subscribers) can register to be notified when another object (the **subject**) changes state. The notification mechanism provides a *loose coupling* between the observers and the subject objects they register with. This allows a one-to-many relationship between the subject and the observers:





The interfaces are:

- [Subject.java](http://www.people.westminstercollege.edu/faculty/ggagne/may2012/lab2/Subject.java)  
- [Observer.java](http://www.people.westminstercollege.edu/faculty/ggagne/may2012/lab2/Observer.java)

An observer must first register itself with the subject via the addObserver() method. When the state of the subject changes, the subject calls the update() method in the observer.  
  
We can illustrate this using UML **sequence diagrams**:

Look at the code for the observer lab. It contains a complete example of the Observer pattern. The application is for an auction house where bidders place bids on items (think ebay).

1. Create a java project from the code and generate a UML class diagram from the code. Which class is the subject? Which class is the observer?
2. Using the auction code as an example create a zoo system that allows a client to register and unregister listeners that will be notified when an animal is added to the zoo. In addition, create a concrete listener that will print the name of the animal that is added to the zoo.
3. Generate the UML class diagram for the zoo. Does it follow the pattern correctly?

.  
Details:  
You need to create five classes, plus a main class that will drive the application:

1. **A ZooSubject** class that acts as the Subject interface
2. A **Zoo** concrete subject class that will act as the subject, responsible for storing a list of the animals in the zoo and notifying a collection of registered listeners each time a new animal is added to the zoo
3. An **Animal** class that represents an animal with a name
4. An **AnimalAddedListener** class that acts as the Observer interface
5. A **PrintNameAnimalAddedListener** concrete observer class that prints the name of the animal that is added to the zoo

The main class might look like this:

public class Main {

public static void main (String[] args) {

// Create the zoo to store animals

Zoo zoo = new Zoo();

// Register a listener to be notified when an animal is added

zoo.registerAnimalAddedListener(new PrintNameAnimalAddedListener());

// Add an animal notify the registered listeners

zoo.addAnimal(new Animal("Tiger"));

}

}

The resulting output is:

Added a new animal with name 'Tiger'