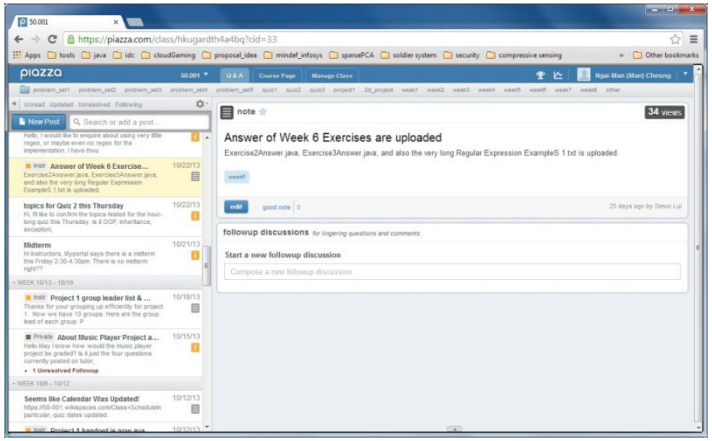
Introduction to Information Systems and Programming

Design Patterns

How to design the software?

- Example 1: To ensure there is only one instance of an object, available to a number of other classes. E.g., A LogFile class for a pool of applications
- Example 2: A message board: whenever a new message is posted to the topic, all the registered observers will be notified



What are design patterns?

- How people have organized / designed their program in the past
- Best software design solutions that others have encountered
  - Improve performance
  - Make it easier to modify the code
- Can apply the same principle to your own software design
  - Solutions to common recurring problems in software design
- One of the most valuable tools for developers / software designers

What are design patterns?

- Comprehensive OO design pattern discussion:
  - Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (Gang of Four, GoF), “Design Patterns: Elements of Reusable Object-Oriented Software” Addison-Wesley 1994.
- We will cover:
  - Singleton
  - Observer
  - Visitor

Singleton

- To ensure there is only one instance of an object, available to a number of other classes
- E.g., Log file for a pool of applications
- GoF: “Ensure a class has only one instance and provide a global point of access to it.”
- A creational pattern: it is used to construct objects

Singleton

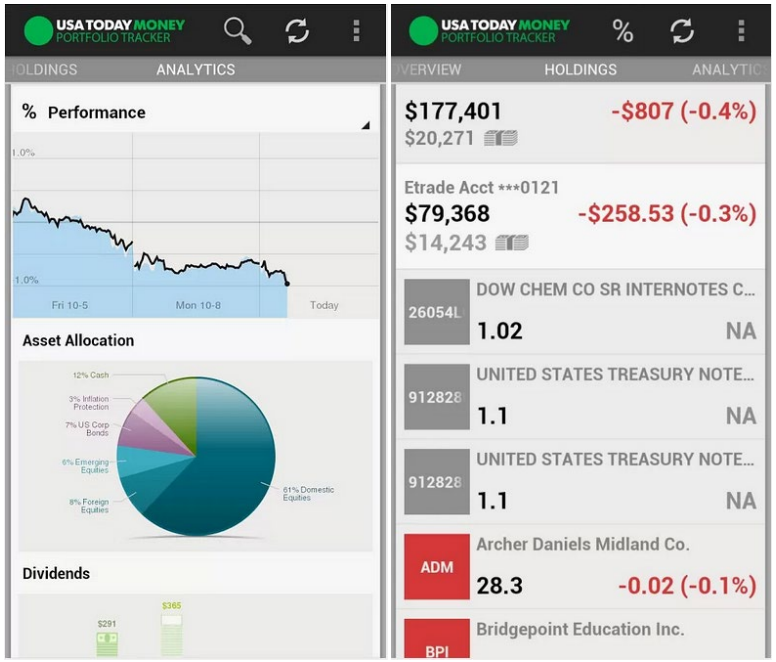
```
public class Singleton {
  private static Singleton instance= null;

  private Singleton(){
  }

  public static Singleton getInstance(){
    if (instance == null)
      instance = new Singleton();
    return instance;
  }
}
```

-Private constructor

-Lazy instantiation: create the instance only when it is used

Observer

- Useful when you are interested in the state of an object and want to get notified whenever there is any change
- Subject (Publisher) maintains a list of its dependents (observers / subscribers)
- Subject notifies them automatically of any state changes

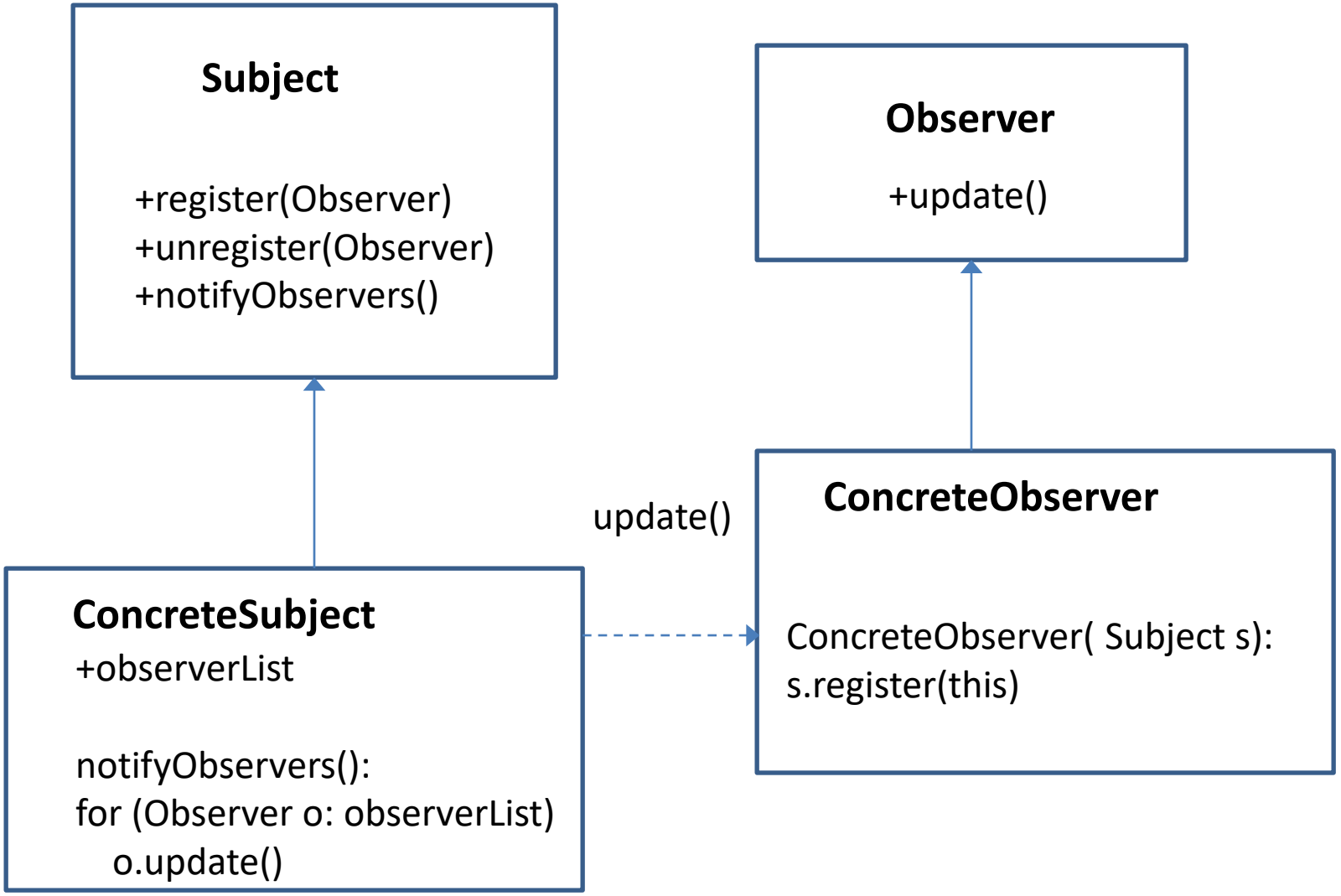


Observer

- GoF: “Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”

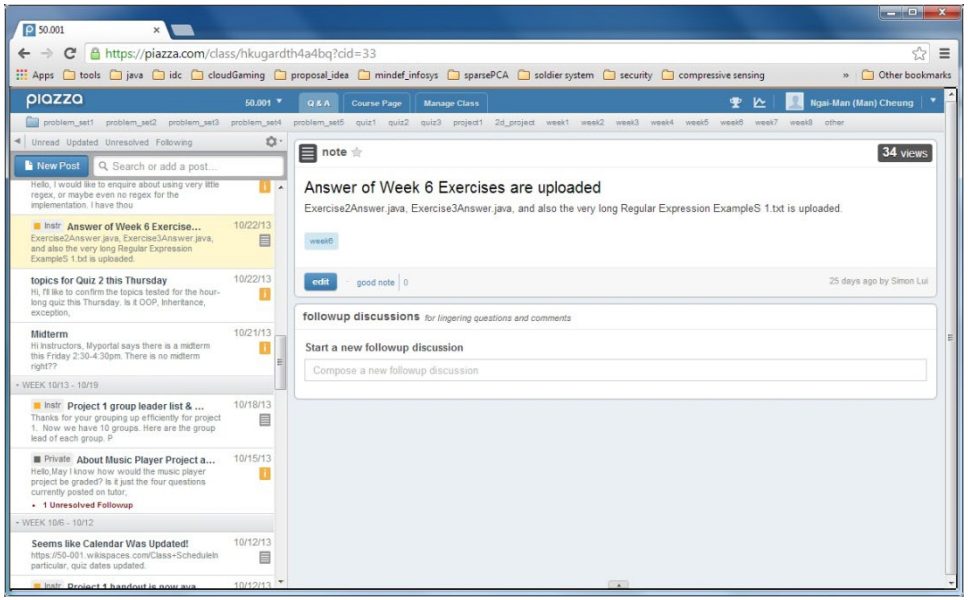
- A behavioral pattern: form relationships between objects during execution at runtime

Observer Design Pattern



Example: Message board

- Subject: a topic
- Observers can register to this topic
- Whenever a new message is posted to the topic, all the registered observers will be notified



Subject.java

```
public interface Subject {
  void register(Observer o);
  void unregister(Observer o);
  void notifyObservers();

}
```

register: call to add a new observer
unregister: call to remover the observer
notifyObservers: inform all registered observers when state change occurs

Observer.java

```
public interface Observer {
  void update(String msg);

}
```

update: call by Subject to inform Observer a change in state. Different classes of Observer can implement this different to respond to the change in state.

Topic.java

```
public class Topic implements Subject {
    private String message;  // the status
    private ArrayList<Observer> observers;

    public Topic (){
      observers = new ArrayList<Observer>();
    }
    public void notifyObservers() {
      for (Observer o: observers)
        o.update(this.message);
    }
    public void register(Observer o) {
      observers.add(o);
    }
    public void unregister(Observer o) {
      observers.remove(o);
    }

    // other features particular to Topic
}
```

Subscriber.java

```
public class Subscriber implements Observer {

    private String message; // received message
    private Subject subject;

    public Subscriber(Subject subject){
      this.subject = subject;
      // register itself to the subject
      this.subject.register(this);
    }

    @Override
    public void update(String msg) {
      // get update from Subject
      this.message  = msg;
      // do something according to the update
      ...
    }
}
```

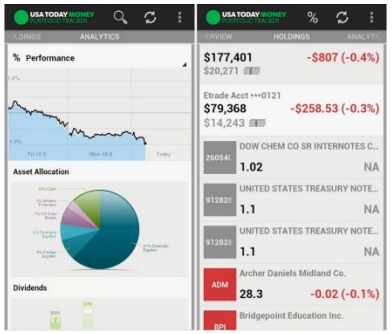
Test.java

```
public class Test {

    public static void main(String[] args) {
      Topic topic50001 = new Topic();
      Subscriber scott = new Subscriber(topic50001);
      Subscriber roger = new Subscriber(topic50001);

      topic50001.postMessage("quiz tomorrow !!!");

      Subscriber man = new Subscriber(topic50001);
      topic50001.postMessage("hw due this wed!!");
      topic50001.unregister(man);
      topic50001.postMessage("great! man has gone!");

    }
}
```

Why Observer?

- Reduce coupling
- An object (Subject) needs to share its state with other objects, without knowing who those objects are

Activity



- Using the Observer design pattern, develop a stock alert system that sends alerts to subscribers for any update of the stock prices
  - Develop StockGrabber class that keeps the list of subscribers for several stock prices, and notifies them for update
  - Develop StockObserver class that monitors changes in the stock prices
  - Develop a Test class to simulate the system