There are two subclasses of abstract class Politician: Conservative and Liberal.

```java
public interface SpeechBehavior {
    public String getQuote();
}

public class ConservativeSpeech implements SpeechBehavior {
    public String getQuote() { return "Liberals just tax and spend"; }
}

public class LiberalSpeech implements SpeechBehavior {
    public String getQuote() { return "We will save the environment"; }
}
```

1) (30%) Write Politician, Conservative and Liberal using the Strategy Pattern for getQuote which is a concrete (i.e., not abstract) method in Politician.

abstract public class Politician extends Observable {

    public void makeMediaEvent() { setChanged(); notifyObservers(); }

}

public class Conservative

public class Liberal
Reporters observe Politicians. When A Politician is told to makeMediaEvent, then the reporters are supposed to be notified and to then ask the Politician for a String from getQuote() which they then publish by System.out.println(). Here is example driver code that shows that bob is added as an observer of john, and then john is told (maybe by his campaign manager to create some attention for himself):

```java
Reporter bob = new Reporter();
Politician john = new Conservative();
john.addObserver(bob);
john.makeMediaEvent();
```

which will result in bob being notified and printing whatever string he gets from john (by calling john’s getQuote method).

Finish writing the Reporter class using the Java API for the Observer Pattern.

```java
public class Reporter
```
What would be the necessary changes to have Politicians also observe each other (in addition to Reporters)? The idea would be if one creates a media event, then the other, who is observing, would also create a media event? Here is the driver code needed:

```java
Politician john = new Conservative();
Politician jane = new Liberal();
john.addObserver(jane);
jane.addObserver(john);
john.makeMediaEvent();
```

Note that john’s media event prompts jane to make her own media event.

Provide the additional code to be added to Politician to case this chain reaction.

Extra credit: are there any unexpected runtime events that this chain reaction triggers, be specific.
To break stereotypes we will make some new types of Politician (to combine with the basic Liberal and Conservative), such as a FearMaker, BibleThumper, or a ChangePromiser. First of all they have their own SpeechBehaviors:

```java
public class FearMakerSpeech implements SpeechBehavior {
    public String getQuote() {
        return "Terrorists will get you if you don't vote for me.";
    }
}

public class BibleThumperSpeech implements SpeechBehavior {
    public String getQuote() {
        return "A vote for me is a vote to stop evolution.";
    }
}

public class ChangePromiserSpeech implements SpeechBehavior {
    public String getQuote() {
        return "It's time for real change, vote for me.";
    }
}
```

Using the Decorator pattern we can make some new kinds of Politician, such as a BibleThumper Conservative or a ChangePromiser Liberal. For instance, you can make a ChangePromiser Liberal by:

```java
Politician p = new Liberal();
Politician cp = new ChangePromiser(p);
```

Note that when you getQuote from a cp, you get the String which is the concatenation of their two quotes (one being a Change Promiser, the other just being a Liberal):

"It's time for real change, vote for me. We will save the environment."

Note that you can have a ChangePromiser Liberal or a ChangePromiser Conservative. It can get kinda weird to then make a bible-thumping, change-promising fear-making liberal, but it could also be done.

First make Decorator. On the back (or another page)

3a) [15%] Create a UML class diagram showing your design.
3b) [25%] Finish coding (only) Decorator and ChangePromiser.