1. You now are to make additions to the code such that Reporters observe Politicians, and when Politicians are given new talking points (by calls to the method setQuote) all Reporters are notified immediately and visit that Politician.

Recall that the abstract class Politician has two concrete subclasses, Senator and President. All Politicians implement Corruptible:

```java
interface Corruptible {
    void acceptMoney(int amount); // bribe money
    void setQuote(String s);      // quotes the Politician paid to say
    String getQuote();           // given to Reporters
}
```

All Politicians accept Visitors, including subclasses: Lobbyist and Reporter.

When a Lobbyist visits a Senator, the Senator accepts bribe money and is told what to say to subsequent Reporters. When a Lobbyist visits a President the two are caught on camera (see methods next page for details).

A Reporter visiting either a Senator or President just gets a quote which is subsequently printed.

For instance:

Politician ted = new Senator("Ted");
Politician george = new President("George");
Lobbyist jack = new Lobbyist("Jack");
Reporter bob = new Reporter("Bob");

```java
ted.addObserver(bob);       // so bob will visit ted later when notified
jack.setBribe(10000);       // give lobbyist money to pass to politician
jack.setQuote("we must drill in Anwar"); // tell politicians what to say
ted.acceptVisitor(jack);    // send the lobbyist to Senator Ted
// ted.acceptVisitor(bob);    // AUTOMATIC NOW (triggered by setQuote method)
bob.printQuote();           // Ted’s words then get printed, and
george.acceptVisitor(jack); // George caught photographed with lobbyist
```

Note that Reporter bob will be notified by Senator tom when a new quote has been set, and that causes the Reporter to immediately and automatically ask the Senator to accept the Reporter as a visitor, resulting in (as before):

An unnamed source said we must drill in Anwar
George photographed with Jack, Click!
Abstract class Politician extends Observable implements Corruptible {

    private String quote;
    private int funds;
    private String name;
    Politician(String name) { this.name = name; }
    public String getName() { return name; }
    public void bePhotographedWith(Visitor v) {
        System.err.println(name + " photographed with " + v.getName() + ",
                Click!");
    }
    public void acceptMoney(int amount) { funds += amount; }
    public void setQuote(String s) {
        quote = s;
        setChanged();
        notifyObservers();
    }
    public String getQuote() { return quote; }
    abstract void acceptVisitor(Visitor v);
}

class Senator extends Politician {
    Senator(String name) { super(name); }
    void acceptVisitor(Visitor v) { v.visit(this); }
}

class President extends Politician {
    President(String name) { super(name); }
    void acceptVisitor(Visitor v) { v.visit(this); }
}
abstract class Visitor {
    private String name;

    Visitor(String name) { this.name = name; }
    public String getName() { return name; }
    abstract void visit(Senator s);
    abstract void visit(President p);
}

class Lobbyist extends Visitor {
    private int bribe = 0;
    private String quote = "";

    Lobbyist(String name) { super(name); }

    void setBribe(int x) { bribe = x; }
    void setQuote(String s) { quote = s; }

    void visit(Senator s) {
        s.acceptMoney(bribe);
        s.setQuote(quote);
    }
    void visit(President p) { p.bePhotographedWith(this); }
}

class Reporter extends Visitor implements Observer {
    private String quote = "";

    Reporter(String name) { super(name); }

    public void update(Observable obs, Object o) {
        ((Politician)obs).acceptVisitor(this);
    }

    void printQuote() {
        System.err.println("An unnamed source said "+quote);
    }
    void visit(Senator s) { quote = s.getQuote(); }
    void visit(President p) { quote = p.getQuote(); }
}
2) [25%] Draw a UML sequence diagram to show the complete flow of control for the following (from main). Assume jack and ted are valid instances of Lobbyist and Senator. Also assume Reporter Bob has been added as an observer of Senator ted.

```java
jack.setQuote("we must drill in Anwar"); // tell politicians what to say
ted.acceptVisitor(jack); // send the lobbyist to Senator Ted
bob.printQuote(); // Ted’s words then get printed
```
3) [25%] Senator and President extend Politician. How would you make only
the Senators (effectively) observable (as if a President’s actions are not).
Hint: remember the Adaptor Pattern. Sketch and provide sufficient code to
get the idea across. Note that you need to have the code in main work
exactly as before.

It starts easy enough. The Adaptor Pattern is used to hide within Senator a
private Observable. Also, make an ObservableI to help specify what methods a
Senator needs to provide to “simulate” extending Observer:

```java
class Senator extends Politician implements ObservableI {
    public Senator(...) {
        super(name);
        observable = new Observable();
    }

    public void addObserver(ObserverI o) { observable.addObserver(o); }
    public void setChanged() { observable.setChanged(); }

    // ...
}
```

Now the tricky part is how to deal with notifyObservers() because if you
write:

```java
public void notifyObservers() { observable.notifyObservers(); }
```

then the observers, such as Reporters, will indeed be notified, but with
their update method called with a reference to the Senator’s private instance
variable observable, not, unfortunately, the parent instance of Senator. It
is as if the Reporters are observing the Senators’ private variables! So
something has to be modified. You could pass the parent class (the instance
of Senator) as the second argument, or make accommodation in ObserverI to
have a notifyObservers method that passes the parent class instead of the
private Observable.

API Notes (you only need the following methods for this quiz):

Observable is a class, which provides for you the methods:

```java
public void addObserver(Observable o);
public void setChanged();
public void notifyObservers(); // use either this version or
public void notifyObservers(Object o); // use this version, your choice
```

and Observer is an interface:

```java
public interface Observer {
    public void update(Observable obs, Object o);
    

    // ...
}
```