1) (10%) Every Politician implements Talker:

```
public interface Talker { public String talk(); }
```

and Politicians sometimes don’t know what to say so they delegate to some other Talker to supply the string of words for them. For instance, Talker tom is the spokesperson for Politician pat below. Note that we don’t know what class tom is an instance of, only that he can effectively interface as a Talker for pat.

```
Politician pat = new Politician();
pat.setSpokesperson(tom); // have tom speak for pat
System.out.println(pat.talk());
```

Using the Adaptor Pattern straightforwardly, finish writing Politician below. Be careful to fill in any missing Java code fragments below.

```
public class Politician extends PublicServant implements Talker {
    private Talker spokesperson;

    public void setSpokesperson(Talker t) { spokesPerson = t; }

    public String talk() { return spokesperson.talk(); }
}
```

2) (20%) Parrot also implements Talker. The problem is, a Parrot is in one of two moods: Cooperative or Uncooperative. Usually a Parrot is cooperative and talk() returns a nice String of words. But if it is in an Uncooperative mood, when asked to talk() it throws a Fit (a kind of Exception to normal behavior, if you know what I mean).

In the following you will rewrite Politician’s public String talk() method to deal with Parrots talking for them.

2a) (5%) First, complete the following Java definition of a Fit. Note that it doesn’t have to supply any new methods; just make it simple:

```
public class Fit extends Exception {}
```
2b) (5%), now write a new version of the interface Talker to deal with those Talkers that might throw a Fit. Remember there is only one public String method talk() to include in the interface:

```java
public interface Talker {
    public String talk() throws Fit;
}
```

2c) (10%) Since Politician pat’s spokesPerson is now under federal indictment, a Parrot was recruited from the hills of San Francisco to fill in. Change Politician’s public talk method so that if it’s delegated Talker unexpectedly throws a Fit when asked to talk, the Politician can catch it (so it stays contained and unknown publicly). Have the Politician simply say “ah, I have no response to that” if the spokesPerson throws a Fit.

```java
public String talk() {
    try {
        return spokesPerson.talk();
    }
    catch (Fit f) {
        return "ah, I have no response to that";
    }
}
```
3) (40%) Now, write the Parrot class. It has two mood-dependent methods: tickle and talk. Public void tickle() causes its mood to change (as will be specified later) while public talk() can throw a Fit, remember.

3a) (10%) First finish the following bits and pieces (look carefully)

```java
public class Parrot implements Talker {
    Mood currentMood;
    Cooperative cooperative;
    Uncooperative uncooperative;

    public Parrot() {
        cooperative = new Cooperative(this);
        uncooperative = new Uncooperative(this);
        setCooperative();
    }

    // set the current mood to cooperative
    public void setCooperative() {
        currentMood = cooperative;
    }

    // set the current mood to uncooperative
    public void setUncooperative() {
        currentMood = uncooperative;
    }

    // Use the straightforward State Pattern for tickle. The Parrot’s
    // mood will be changed inside the Mood classes, not here.

    public void tickle() {
        currentMood.tickle();
    }

    public String talk() throws Fit {
        return currentMood.talk();
    }
}
```
3b) (10%) Next, write the base class for the parrot’s Mood. The rules for changing the parent Parrot’s mood will involve count and LIMIT, and explained as we go. If you tickle the parrot when uncooperative, it not only resets count to zero but it also sets the parrot’s mood to cooperative.

```java
public abstract class Mood {
    protected int count;
    protected int LIMIT = 10;
    protected Parrot parent;

    public Mood(Parrot p) {
        parent = p;
        count = 0;
    }

    abstract public String tickle();
    abstract public String talk() throws Fit;
}
```

3c) (10%) Now write Cooperative. The int count in Mood keeps track of the number of times the Parrot is asked to talk(). If count reaches LIMIT, the parent Parrot’s mood is set to Uncooperative. But if you tickle the parrot when it is cooperative, the count is reset to zero.

```java
public class Cooperative extends Mood {
    private String s = "Vote for me"; // always return this same string

    public Cooperative(Parrot p) { super(p); }

    public void tickle() { count = 0; }

    public String talk() {
        count++;
        if (count == LIMIT)
            parent.setUncooperative();
        return s;
    }
}
```
3d) (10%) Now write Uncooperative. If it is asked to talk() it throws a Fit. If you tickle the parrot when it is uncooperative, the count is reset to zero and the parent Parrot is set to be cooperative again.

```java
public class Uncooperative extends Mood {

    public Uncooperative(Parrot p) { super(p); }

    public void tickle() {
        count = 0;
        parent.SetCooperative();
    }

    public String talk() throws Fit {
        throw new Fit();
        return null; // not sure this is needed
    }
}
```
4a) (5%) Write an additional method for your Parrot class so it can be visited by instances of Visitor:

```java
public void accept(Visitor v) { v.visit(this); }
```

4b) (5%) Write a base class Visitor that visits with either a Politician or a Parrot.

```java
abstract public class Visitor {
    abstract public void visit(Politician p);
    abstract public void visit(Parrot p);
}
```

4c) (5%) Write a subclass of Visitor called SnackV which, when it visits with a Politician, causes it to talk(), and if it visits a Parrot, it causes it to be cooperative (using setCooperative).

```java
public class SnackV extends Visitor {
    public void visit(Politician p) { p.talk(); } // doesn’t call S.O.P.
    public void visit(Parrot p) { p.setCooperative(); }
}
```
5) (15%) The Parrot, from the time it is born, is always observing itself, and every time it becomes uncooperative, it prints out “Squaak!”. Use the straightforward (Java-based) Observer Pattern to accomplish this. Make all changes necessary by annotating existing code elsewhere in this exam. So that Parrot is complete below, copy the changes you made in answering 3a, above).

```java
public class Parrot extends Observable implements Talker, Observer {
    Mood currentMood;
    Cooperative cooperative;
    Uncooperative uncooperative;

    public Parrot() {
        cooperative = new Cooperative(this);
        uncooperative = new Uncooperative(this);
        setCooperative();
        addObserver(this);
    }

    // set the current mood to cooperative
    public void setCooperative() {
        currentMood = cooperative;
    }

    // set the current mood to uncooperative
    public void setUncooperative() {
        currentMood = uncooperative;
        setChanged();
        notifyObservers();
    }

    public void update(Observable obs, Object o) {
        System.err.println("Squaak!");
    }
}
```