Q. What is the problem being solved in the Observer Pattern?

Given two different objects, one object called the Observer wants to know when some event occurs to the other

Q. What are the three elements of a good solution to this problem?

1. Subject code should be unaware of specific observers
2. Multiple observers should be able to observe the same object
3. An observer should be able to observe multiple subjects

Q. Why is polling not a good solution?

When the subject code is aware of the specific observer it is like hardcoding a constant variable. Anytime it is referenced or changed it will have to be changed in all other classes as well. This is not good because it is not a modular design. In addition if there are multiple observers then there will be multiple dependencies on multiple classes.

Q. Describe the solution used in the observer pattern.

The observer pattern uses an observer as an interface between the two objects. The subject being observed has a list of observed and when some change occurs a certain method is called. This method will go over the entire list of observers and will decide which method to call to handle the correct one. This way the subject never interacts with the specific individual observers.

Q. Given an Account object and a Person object. When account is debited, the

person object is to be notified (and it prints the amount being debited). Write code segments/snippets using java.util.Observable and java.util.Observer to implement this.

Also, given the following information

java.util.Observable is an abstract class with following methods

- setChanged()
- clearChanged()
- notifyObservers (Object n)
- addObserver(Observer o)
- deleteObserver(Observer o)

java.util.Observer is an interface with following abstract method

- void update (Observable o, Object n)

```
Public class Person implements Observer{

    private Account account= New account();

    public Person(Account account){
        this.account = account;
    }

    Observable observer = new Observable();

    this.account.addObserver(observer);
    observer.update(observer, acc);

   void update(Observable o, Object n) {
        this.account = n;
        setChanged();
        notifyObservers(this);
        System.out.println(this.account.getDebit());
    }

}
```