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

There is a subject and there is an observer. The observer wants to do something whenever some event happens in the subject. As an example, the subject is a table. The observer is a line graph. Whenever a table entry is changed, the line graph gets redrawn.

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

1. The subject should not know about the specific observer. Observer should not know about the specific subject (but would know about the event details).

2. Subject can have any number of observers.

3. Observer can observe any number of subjects.

Q. Why is polling not a good solution?

1. Waste of CPU cycles. Polling is basically a loop in observer code to check if something changed on the subject.

2. Missing events. If the polling frequency is too less, some events may be missed.

3. Makes the observer code complex (One implementation would be to have a separate thread that does the check and then sleeps. Another implementation would be to intersperse the polling code in regular code).

Q. Describe the solution used in the observer pattern.

The subject maintains a list of observers. The Observer is an interface. Implementations of observers will use the subject's addObserver (or similar) method to add themselves to this list and removeObserver (or similar) method to remove themselves from this list. The Observers will need to implement a handleEvent (or notify or actionPerformed etc) method. When some event happens on the subject, it will go over it's list of observers and invoke each of the corresponding handleEvent method.

Note that the Subject does not know anything about the observers (as it only knows

the interface Observer). Also, it can have any number of observers. Similarly, observers can be added to any subject (and to multiple subjects).

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)

```
class Account extends Observable {
        boolean debit( int amt) {
           // on success do the following
                setChanged();
                notifyObservers(amt);
                clearChanged();
        }
}

class Person implements Observer {
        public void update (Observable o, Object n) {
                System.out.println( (int) n + " was debited");
        }
}

some other class:
        Account account = new Account();
        Person person = new Person();
        account. addObserver(person);
        account.debit(100);
```