Alice has decided to implement her own version of FaceBook called FBook. She realizes that most likely this maps to a graph ADT and recons that FBook will have to support at least creating and deleting profiles, adding and deleting friendships, and a number of other operations she has only some vague intuition about so far.

Here is the signature she has in mind so far:

```java
public class FGraph {
    public void addProfile (Profile p);
    public void deleteProfile (Profile p)
        throws InexistentProfileException;
    public void addFriendship (Profile p, Profile q);
    public void deleteFriendship (Profile p, Profile q)
        throws InexistentFriendshipException;
}
```

**Question 1 [20p]:** Write specifications for deleteProfile and deleteFriendship

```java
public void deleteProfile (Profile p) throws InexistentProfileException;
//REQUIRES:
//MODIFIES: this
//EFFECTS: if p is null or p is not a profile in the graph throws InexistentProfileException
// otherwise deletes the profile and all corresponding friendship links form the graph.
// For example if the graph has the profiles A, B and C and all possible friendships between
// these profiles, deleting profile A will result in a graph composed of profiles B and C
// and one friendship B-C
```

(for this one you might even want to improve the header)

```java
public void deleteFriendship (Profile p, Profile q)
    throws InexistentFriendshipException, InexistentProfileException;
//REQUIRES:
//MODIFIES: this
//EFFECTS: if p or q are null or p or q are not profiles in the graph throws InexistentProfileException
// if p-q is not a friendship in the graph then throws InexistentFriendshipException
// otherwise deletes the friendship between p and q in the graph.
// For example if the graph has the profiles A, B and C and all possible friendships between
// these profiles, deleting friendship A-B will result in a graph composed of the same
// profiles and friendships only between A-C and B-C
```

**Question 2 [20p]:** Should Alice add more methods you would add to the class signature? If no why? If yes which ones and why (present the method header and a short description)?

Yes. See textbook – generally a class will have three of the following categories of methods: getters, setters, constructors and producers.

**Question 3 [20p]:** Now that Alice has the specs ready she has decided to represent the graph as an adjacency list as follows.

```java
public class FGraph {
    private Vector<Profile> pf;
    //methods to manipulate the whole graph including
    //add any method you like in addition to the ones below
    public void addProfile (Profile p);
    public void deleteProfile (Profile p)
```
throws InexistentProfileException;
public void addFriendship (Profile p, Profile q);
public void deleteFriendship (Profile p, Profile q)
        throws InexistentFriendshipException;
}

public class Profile {
    private String name;
    private Vector<Profile> friends; //list of friends for a profile

    //methods to manipulate individual profiles
    //add any method you like
}

Write the representation invariant for this representation.

\[
RI(g) = \text{g.pf is not null AND} \\
\text{g.pf does not contain duplicates AND} \\
(\text{forall i, 0 <= i < g.pf.size}) \\
\text{let P = g.pf[i];} \\
\text{P.name is not null AND} \\
\text{P.name is unique AND} \\
\text{P.friends is not null AND} \\
\text{P.friends does not have duplicates AND} \\
\text{P is not in P.friends AND} \\
(\text{forall j, 0 <= j < P.friends.size}) \\
\text{let Q = f.pf[i].friends[j]} \\
\text{Q is member of g.pf AND} \\
\text{P is within Q.friends AND} \\
\text{P is not friend with itself AND} \\
\text{Q is in adjacency list of P} \\
\text{and P is in Q's adjacency list}
\]

\*

Question 4 [20p]. Sketch a Java implementation for deleteProfile that will match your own specification.

public void deleteProfile (Profile p) throws InexistentProfileException;
    //REQUIRES:
    //MODIFIES: this
    //EFFECTS: if p is null or p is not a profile in the graph throws InexistentProfileException
    //otherwise deletes the profile and all corresponding friendship links form the graph.
    //For example if the graph has the profiles A, B and C and all possible friendships between
    //these profiles, deleting profile A will result in a graph composed of profiles B and C
    //and one friendship B-C

    if (p == NULL)
        throw new InexistentProfileException ("Null profile");
    if (this.pf.remove(p)) { // p is in pf
        for (int i=0; i<pf.size; i++) {
            Profile k = this.pf.elementAt(i);
            k.friends.remove(p);
        }
    }
    else {
        throw new InexistentProfileException ("No profile:" + p);
    }
**Question 5 [20p].** Sketch an argument to prove that your own implementation for deleteProfile above correct.

Basically you have to provide a two step argument:
- Prove that if the rep invariant holds before you enter deleteProfile then it will also hold after
- You implement the spec.
(you got max points here even if you had minimal sketches of this argument)

**Question 6.** Alice thinks she needs a way to iterate over all friendship links in the graph. Help her achieve this: most likely she needs a method to create a generator (let’s call it `getAllFriendships`) and a generator class (let’s call it `FriendshipsIterator`).

A. [10p] Present an implementation for the `getAllFriendships` method

```java
// this is a method of FGraph
public iterator getAllFriendships () {
    return new FriendshipsIterator (this);
}
```

B. [20p] Present an implementation for the generator class

```java
private static FriendshipsIterator implements Iterator {
    // I'll assume there is a class Friendship defined somewhere else that
    // simply stores a pair of profiles, that is, it stores a friendship
    Vector<Friendship> f;

    // the constructor collects all edges in the friendship graph and avoids duplicates
    FriendshipsIterator (FGraph g) {
        f = new Vector<Friendship>;
        for (int i=0; i<g.pf.size; i++) {
            for (int j=0; i<g.pf[i].friends.size; j++) {
                Friendship toAdd = new Friendship (g.pf[i], g.pf[i].friends[j])
                if notAlreadyInserted (f, toAdd)
                    f.add(toAdd)
            }
        }

        public boolean hasNext () {
            return (f.size() > 0);
        }

        public Object next () {
            Friendship last = f.lastElement();
            f.remove(last);
            return last;
        }
    }
}
```

C. [20p] Present a representation invariant and abstraction function for the generator class

```java
RI(c) = c.f != null AND c.f does not contain duplicates
AF(c) = the set of friendships in c.f (in any order)
```


```
Static inner class
```
Question 7. Bob notices a potential problem with Alice’s representation: His intuition is that there is useful historical information that should be maintained in the system (e.g., to detect spammers or malicious activity) and, with Alice’s representation, there is no easy way to keep track of history. For example, he argues that the system should record the interaction history between profiles (e.g., the fact that Jennifer has been friend with Tom but latter un-friend him, then she accepted his friendship request again, then she un-friend him again).

- [20p] Update the FBook representation to be able to record this type of information (basically to record the history of a friendship).
  
  [The solution below is a big hammer solution – not elegant but good enough for this quiz. Ideally you maintain edges as separate objects with their own history.]

  ```java
  public class FGraph {
    private Vector<Profile> pf;
    private Vector<Action> log;
    // methods to manipulate the whole graph including
    // add any method you like in addition to the ones below
    public void addProfile (Profile p);
    public void deleteProfile (Profile p)
      throws InexistentProfileException;
    public void addFriendship (Profile p, Profile q);
    public void deleteFriendship (Profile p, Profile q)
      throws InexistentFriendshipException;
  }
  
  public class Profile {
    private String name;
    private Vector<Profile> friends; // list of friends for a profile
    // methods to manipulate individual profiles
    // add any method you like
  }
  
  public class LoggedAction {
    Time loggedTime;
    // has some other private members to keep track of the action
    // in this case I only want to keep track of adding and deleting
    // friendships
    Profile p1;
    Profile p2;
    String action;
  }
  
  - [10p] Update your specification and header for deleteProfile (if needed).

  It depends on the representation you choose and the semantics you want. For the representation above, most likely you’ll want add in the EFFECTS that you need to clear the log for all data that becomes meaningless after a profile removal.

  - [10p] Update your specification and header for deleteFriendship (if needed).

  Add in EFFECTS how the log is updated.

  - [20p] Update your FB representation invariant.

  Add constraints on the log to guarantee consistency: profiles must exist, not self-links, no two actions at the same time, links are first added then deleted, no two successive deletes, etc ..