INF100 H17 Assignment 7

Due on Friday, November 24, 2017

Albin Severinson, Dag Haugland

Abstract

I realize now that implementing Pokemon may have been a slightly too ambitious project. Instead we will implement Facebook. In fact, Facebook reportedly made $10.3 billion only in the last quarter. This is much more money than the paltry $569 made by Nintendo. Specifically, we will write a class to represent a user. Furthermore, we will allow connecting users to each other. As usual the implementation is split over several sub problems, where each step builds upon the previous. We will only grade (pass/fail) the program for the final problem of the assignment. If you have questions specific to this assignment you should send them to me at `albin.severinson@uib.no`.

Submission

1. Add your program file to a `.zip` archive named `ex7_firstname_lastname.zip`. You do this by right clicking on the file ending with `.java` that contains your program code and choosing compress or add to archive (exactly what it is called varies by operating system and version). You should only add your final program, i.e., the one for problem 3. Remember that the file name must match the class name and that you therefore can not rename the `.java` file! Ask your group leader if you are having issues with this. Note that handing in the wrong file may cause you to fail the assignment!
2. If you complete the voluntary getting ahead part of the assignment, it should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 5 and the `.java` file for the getting ahead part). Put both files into the same `.zip` archive.
3. Log in to `https://mitt.uib.no` and press the box marked with "INF100" to enter the page for this course.
4. Press "Oppgåver" on the left. Next, select "Assignment 7".
5. Press "Lever oppgåve" and upload the `.zip` file you created in step 1.
6. We grade assignments in one week and semester assignments in two weeks.

General Guidelines

These are general guidelines that will (hopefully) help you maintain your sanity through your programming endeavours.

- Have your computer indent your code for you. You do this in DrJava by selecting all of your code, right clicking, and selecting `indent lines`.
- You should make a copy of all your code after completing each problem so that you can roll back to a previous version when you inadvertently break something. Even the developers at Google sometimes have to roll back.
- Set aside time to clean up your code. The number that gets bandied around on the internet is that the best programmers spend roughly half their time cleaning up and rewriting their code, i.e., you are halfway done when you get it working.
- You save time by making sure you properly understand your current program before moving on.
- You should probably be spending a significant part of your time on stackoverflow.
- Google is your friend when you want to figure out how a standard class or method works (`Scanner`, say). For example, searching for "java Scanner" gives you its Java docs page as the first result.
- When in doubt, look at the Google Java style guide to figure out how to write something.

Problem 1

Start by creating a class `User`. This class should have fields

```
/* name of this user */
private String name;

/* friends of this user */
private ArrayList<User> friends;

/* all registered users */
private static ArrayList<User> users = new ArrayList<User>();
```

Note the `static` keyword. In this case it means that `ArrayList<User> users` is common to all objects created from this class. More generally, something marked with `static` is part of the class and not to a specific object instance.

The class constructor should look like this:

```
public User(String name) {
    this.name = name;
    this.friends = new ArrayList<User>();
    users.add(this);
}
```

Furthermore, the `toString()` method should return a string with the following format

```
name : {friend1, friend2, friend3, ...}
```

For example, a user "Alice" with friends "Bob" and "Charlie" would print like this

```
Alice : {Bob, Charlie, }
```

Problem 2

For this problem we will implement friend connections. First, implement a method that adds another user to the `friends` of this user.

```java
public void addFriend(User friend) {

}
```

However, being friends is a symmetric relationship. For example, if “Alice” befriends “Bob”, we would like “Alice” to be added to “Bob’s” lists of friends, and “Bob” to “Alice’s” list of friends. Write a `static` method that implements this. This method should call the `addFriend()` method of both `user1` and `user2`.

```java
public static void connect(User user1, User user2) {

}
```

Next, create a static method for returning the list of registered users.

```java
public static ArrayList<User> getUsers() {
    // return ArrayList of registered users...
}
```

Create a new file `Main.java` with a `main` method to try out the features added so far. The `main` method should look like this:

```java
public static void main(String args[]) {
    // create users
    User bob = new User("Bob");
    User alice = new User("Alice");
    User charlie = new User("Charlie");

    // setup friend connections
    User.connect(alice, bob);
    User.connect(alice, charlie);

    // print all registered users
    System.out.println(User.getUsers());
}
```

It should produce the following output

```
[Bob : {Alice, }, Alice : {Bob, Charlie, }, Charlie : {Alice, }]
```

Problem 3

Next we will implement a feature not yet offered by Facebook: We will allow sorting all users by the number of friends they have! We will implement this using the `Comparable` interface. If `User` implements `Comparable` we can use `Collections.sort` to sort the users. You can read up on this interface here. First, change your class declaration to `public class User implements Comparable<User> {`.

The `User` class needs a `compareTo` method to implement the `Comparable` interface. This method should return a negative integer if this user has fewer friends than `otherUser`, a positive integer if this user has more friends, and 0 if both users have an equal number of friends.

```java
public int compareTo(User otherUser) {

}
```

Update your `main` method to look like this:

```java
public static void main(String args[]) {
    // create users
    User bob = new User("Bob");
    User alice = new User("Alice");
    User charlie = new User("Charlie");

    // setup friend connections
    User.connect(alice, bob);
    User.connect(alice, charlie);

    // sort users by their number of friends
    Collections.sort(User.getUsers());

    // print all registered users
    System.out.println(User.getUsers());
}
```

It should produce the following output. Remember that you need to import `Collections`. Note that the list of users is sorted.

```
[Bob : {Alice, }, Charlie : {Alice, }, Alice : {Bob, Charlie, }]
```

Getting Ahead

It is a good start. Even Facebook has humble beginnings. Since this is the last assignment of the course we have two tracks for the getting ahead part. The first is it implement a few of the features you think our social network needs. I recommend you to think of the parts of the course you found most challenging and to use those parts to add more features. Suggestions include

- Users posting messages to a shared board.
- Unfriending users.

The second track is to move beyond the scope of this course. Two very useful programming tools not covered by this course are hash maps and recursion. You can learn about hash maps here. A good introduction to recursion can be found here.

This part is meant as preparation for the coming course material and will not be graded. However, we will still look at it and comment it! Furthermore, you never have to catch up if you are always getting ahead. This part should be handed in separately. If you complete this part you thus need to hand in two sets of files (the `.java` files for problem 3 and the `.java` files for this part). Put both files into the same `.zip` archive.