CSci 1933 Assignment 2

This assignment is due on **Friday, October 30, 2015** at **11:55 PM**

You are supposed to work in group of 2 (unless exempted by the instructor), otherwise the homework will not be graded. You have the options 1) to work with the previous teammate or 2) to find a new teammate. If you have difficulties finding a teammate, please post to Student Forum on Moodle site as soon as possible.

1. Overview

1.1. You will complete the implementation of a simple social network back-end which wraps Twitter. It will track user relationships, add separate group functionality, and be able recommend new people to follow.

1.2. We've provided functionality for reading in the dataset and much of the data management. Your job will be to implement the IGroup and IUser interfaces.

> **Note** | Your implementing classes should be called User and Group.

1.3. This assignment consists of three parts:

1.3.1. In Part One, you must correctly implement both the IUser and IGroup interfaces.

1.3.2. In Part Two, you must complete the method recommendUsersToFollow() in SocialNetwork.java. The B level algorithm will be fairly simple, based on the idea that you're likely to want to follow users who are followed by users whom you already follow. That is, if you like what user X has to say, there's a chance you'll like what the users whom X follows have to say.

1.3.3. In Part Three, you must expand the recommendUsersToFollow() method to take group membership into account. In addition, you will add functionality to monitor the number of calls made to the compareTo() and equals() methods of User and Group, so we can see how our social network implementation will or will not scale.

2. Getting Started

2.1. Start a new IntelliJ project and import the following files into the project according to the instructions. Hint: follow these instructions closely to avoid the file path issues you may have had in HW1!

- `SocialNetwork.java` (Contains main() as well as some data handling methods; add this into your `src/` folder)

> **Note** | You'll find code in above file that we won't use for this assignment, but that we'll use for future assignments.

- `DataReader.java` (add this into your `src/` folder; this will be used in future homework. It's included for the sake of completeness.)
- `TweetReader.java` (add this into your `src/` folder; this will be used in future homework. It's included for the sake of completeness.)
- `IUser.java` (User class interface; add this into your `src/` folder)
- `IGroup.java` (Group class interface; add this into your `src/` folder)
- `Identifiable.java` (This is a parent interface of both `IGroup`, and `IUser`; add this into your `src/` folder)
- `NetworkTest.java` (To check your work; add this into your `src/` folder).

2.2. Follow these instructions to add JUnit to your project:

- Select "Project structure" from the "File" menu.
- Go to the "Libraries", click on the "+" sign, click "Java"
- Import "junit.jar" under <IntelliJ IDEA installation directory>\lib. (On Mac, this is Application/IntelliJ IDEA 14 CE.app/Contents/lib, for more information, please read https://www.jetbrains.com/idea/help/configuring-testing-libraries.html

3. Part 1

3.1. To receive full credit on this part, you must implement the User and Group classes fully. In addition to implementing the IUser and IGroup interfaces, they also must override the equals() and toString() methods inherited from the basic Java class Object.

3.2. You will not need to (and should not) modify any of the provided files for Part One. SocialNetwork.java, DataReader.java, and NetworkTest.java will have errors until you've implemented IUser and IGroup.

3.3. To implement interfaces in Java, do the following in your class declarations:

```
public class User implements IUser {
…
}
```

3.4. See the comments in IUser.java, and IGroup.java for details on what each method should do.

3.5. Things you will need to keep in mind while implementing these classes:

3.5.1. Constructors: Both `User` and `Group` should have constructors that take zero arguments. The unit tests (in NetworkTest.java) and other provided code require this.

3.5.2. Lists: Both `IUser` and `IGroup` have methods that require you to return `List`s. `List` is an interface and cannot be instantiated, so you will need to choose an appropriate implementation to use (don't change `IUser` or `IGroup`). See the Java documentation on the `List` interface for options. Example:

```
List<IUser> users = new ArrayList<IUser>();
```

3.5.3. Interface Inheritance: Interfaces, like classes can have parents and children. `IUser` and `IGroup` are both children of `Identifiable`, which in turn is a child of `Comparable`. Methods described in parent interfaces are cumulative. This means that for `IUser` and `IGroup`, you must implement `getID()`, `setID()`, and `compareTo()`. See `Identifiable.java` for a description what these methods should do.

Recall that if you are using IntelliJ and you specify that your class is implementing (for example) `IUser`, IntelliJ can automatically insert "stubs" for all the methods you need to implement using its "Generate" functionality.

3.5.4. Method Overrides: You also must override `toString()` and `equals()` in both your `User` and `Group` classes. To override their implementations in `Object`, they will need to have the following signatures:

```
public boolean equals(Object o) {
…
}
public String toString() {
…
}
```

3.5.5. The `equals()` method should return `true` only when the `Object` passed in is the same type (`User` or `Group`) and has the same ID as this `Object`. See `Shape.java` from Lab 5 for hints on how to check type (e.g. the "instanceof" keyword).

3.5.6. `toString()` should return the `User`'s or `Group`'s ID.

3.5.7. When you add an object to a group, remember to check for its existence in the group beforehand.

4. Part 2

4.1. For this part of the assignment, you will complete the `recommendUsersToFollow()` method in `SocialNetwork.java`. Currently the method looks like this:

```
public List<IUser> recommendUsersToFollow(IUser user, double
minCoefficient) {
return null;
}
```

4.2. Below we reference `user`, `f`, and `fof`. They are defined as :

- `user`: The user for whom we are recommending users to follow; passed into the method `recommendUsersToFollow()`.
- `f`: The users `user` follows will be referred to individually as `f`.

- `fof`: The users `f` follows will be referred to individually as `fof`.

In other words, `fof` are people who the followers of `user` follow. See below for a visual explanation of these relationships.

4.3. You will implement a simple method for recommending users to follow, namely recommending to a `user` that he/she follow people who are being followed by people whom `user` already follows.

4.4. Here is the basic algorithm to recommend new users for a user `user` to follow:

4.4.1. Set up your data structures:

- Create a list `potentialUsersToRecommend`, which initially is empty. This will hold a list of users (identified by `fof` below) that will be considered for recommending user `user` to follow.

- Create a list `recommendedUsers`, which initially is empty. This list will hold a list of users recommended for `user` to follow. This is the list that should be returned by `recommendUsersToFollow()`.

4.4.2. For each user `f` that `user` follows:

- If `user` is not equal to `fof` and `user` does not already follow `fof`, then if `fof` already is in the list `potentialUsersToRecommend`, increment the count of how many times `fof` has been encountered.
- Else add `fof` to the list `potentialUsersToRecommend`, and record the fact that `fof` has been encountered once.

- At the end of this step, you should have list `potentialUsersToRecommend` containing the users `fof` and a corresponding count for the number of times `fof` has been encountered.

4.4.3. For each user `fof` in `potentialUsersToRecommend`:

- If `fof` has been encountered "often enough", which will be true if `F > minCoefficient` (see below for details of how to compute the factor `F`), then `fof` should be added to `recommendedUsers`.

```
F = number of users that user follows that follow fof /total
number of users that user follows.
```

Note Remember to cast integers to double/float while computing above expression.

- At the end of this step, you should have a list `recommendedUsers` of users for whom `F` (see calculation below), is greater than `minCoefficient`.

4.5. Finally, the JUnit test in `NetworkTest.java`, `testRecommendUsersToFollow()`, should pass if you've implemented `recommendUsersToFollow()` correctly.

Hints

- There are a number of ways for you to keep track of how many times each user `fof` has been "encountered" (as described in the algorithm above). We discussed several such methods in class, which you can implement using data structure we already have covered. A few ideas: a wrapper class for User that is Comparable, parallel lists.
- If you're having trouble, I recommend using System.out.printf() or .println() to "log" what is happening as the loops are progressing. For instance, these lines of code may be useful:
  - `System.out.printf("Incrementing count for %s to %d\n",potentialUsersToRecommend.get(index),newCount);`
  - `System.out.println("Adding user: " + fof.getID());`

4.6. Now, you are ready to run JUnit test to verify that your implementation achieves the minimal correctness (Just passing JUnit tests does not necessarily guarantee 100 points on this homework). To run JUnit test

4.6.1. Click "Run" in the menu
4.6.2. Click "Edit Configurations"
4.6.3. Click "+" at the top left corner and select "JUnit"
4.6.4. Fill out the configurations, most importantly, choose "NetworkTest" in the "Class" field
4.6.5. Click "OK" to close the configuration window.
4.6.6. Click "Run" in the menu and click "Run {testname_you_defined}" to start the JUnit testing.

5. Part 3

5.1. For Part 3, we want a more advanced way to recommend users to follow. Basing it solely on who follows whom is a good start, but we can do better: specifically, you will modify your algorithm to take into account the groups users belong to.

Again, we reference `user`, `f`, and `fof` from last part.

5.2. Because Twitter relationships are one way, we care more about what groups the users we follow are in than what groups we are in. We want to try and identify

users whose behavior is similar to that of the users we already follow, rather than similar to our own. For Part Three, we will compare what groups the users `f` we follow are in with the users `fof` that they follow.

5.3. Inside of the `recommendUsersToFollow()`, do the following:

5.3.1. In step 1 of Part Two, add a second counter (a group counter) for each user in the `potentialUsersToRecommend` list. Again, as mentioned above, there are a variety of ways to do this.

5.3.2. After step 2 of Part Two, add the following:

- For each user `fof` in the list `potentialUsersToRecommend`:
  - For each group `g` that the user `fof` is in:
    - Add the number of users `f` that `user` follows that are in `g` to the group counter for `fof` created above (Hint:).

> **Hint** Remember the method `getFollowedUsersInGroup()` that you implemented for `IUser`?

At the end of this step, you should have counts for all of the users `fof` in `potentialUsersToRecommend`, of users `f` that `user` follows in groups with `fof`.

5.3.3. In step 3 of Part Two, modify the follow coefficient `F` equation to account for the group factor `G`. The group factor `G` is the value of the group counter multiplied by five and divided by the total number of users that `user` follows.

```
G = ( value of the group counter * 5 ) /total number of users
that user follows.
```

5.4. The modified version of the follow coefficient `F` equation includes the group factor G as specified next:

```
F = ( number of users that user follows that follow fof + G )/total
number of users that user follows.
```

5.5. You will also need modify the JUnit test for recommendUsersToFollow(), testRecommendUsersToFollow(), to work with this new algorithm. Currently, we make sure `u13` isn't a recommended user to follow.

```
...
List rf = sn.recommendUsersToFollow(u4, 0.3);
assertFalse(rf.contains(u10));
assertFalse(rf.contains(u13));
assertFalse(rf.contains(u2));
```

5.6. Because of the new equation for determining the follow coefficient, `u13` should now be recommended to be followed by `u4`. Modify the test such that if `u13` isn't recommended as a user to follow, it will fail. Then, if you modified the JUnit test correctly, and modified the user follow recommendation algorithm correctly, the JUnit test should pass.

5.7. After modifying the JUnit test, you can run JUnit test again using the Configuration you set up in **4.6**

6. Submission

Before you submit your solution, **you must create a text file called group.txt in your src/ directory**. In this file, put the names and x.500 ID's of the members of your group.

Put `build.gradle` file in the project root and run command `gradle tar`. This will create a tar.gz file for submission. Please make sure the `src/` files are in the created tar.gz file.

Submit the tar.gz to the **Lecture Moodle Site.**

---End of Assignment 2---