CS 367 Links
Main
- Home
- Syllabus

Course Work
- Grading
- Exams
- Programs
- Homeworks

Resources
- Piazza
- CS Computer Labs
- TA Consulting
- Remote Access
- New to Linux?

- Learn@UW
- CSL
- UW CS Department

Guides
- CS 302 Style
- CS 302 Commenting

Eclipse
- Eclipse Home
- CS 302 Tutorial
- CS 302 Debugging Lab
- CS 302 Download Info

Java 8
- Java SE API Specs
- CS 302 Download Info

CS 367: Introduction to Data Structures

CS 367 Programming Assignment 1

DUE by 10:00 PM on Sunday, February 14th

Announcements | Overview | Specifications | Steps | Submission

Announcements

Corrections, clarifications, and other announcements regarding this programming assignment will be found below.

- **1/26/2016:** NOTE: Program 1 must be done individually. Pair programming is only allowed on programs 2 - 5.
- **1/26/2016:** Handin directories have NOT been created for program 1. Watch here for an announcement.
- **1/26/2016:** Here are some problems seen in the first programming assignment that we want to warn you about:
  - not following the commenting and style guidelines
  - not following the program output format exactly as shown in the sample-run file
  - using multiple Scanners for console (e.g., System.in) input
  - having additional output other than what is specified (especially debug related)
  - having poorly-written code, even if it works (e.g., redundancy, convoluted functions, poor modularity)
- **1/26/2016:** Progam 1 assigned.
- **2/4/2016:** Extremely minor changes to User.java, CommandProcessor.java and the sample output have been done. Look for // CORRECTED in the source files. Verify the new sample output also.

Overview

Goals

The goals of this assignment are to:

- Practice compiling and running Java programs in the development environment of your choice (and Linux to verify what you submit).
- Become familiar with Java's `List` interface by using Java's `ArrayList` class.
- Gain experience using iterators (from Java's `Iterator` and `Iterable` interfaces) to traverse lists.
- Get practice throwing exceptions.
- Use command-line arguments.
- Review basic file and console I/O.

Description

For this assignment you will write a program that will simulate the working of the popular instant messaging service WhatsApp. Your simulation of this service will cover the most basic parts of it like one to one messaging and broadcasts. The run of this simulation will be console based. The model will have users and messages. Each user has her identification in the form of name, password and a unique nickname. Each message has a 'from user', 'to user', the message, sent time, etc. Every user maintains a list of her friends whom she can contact. She also can have a list of broadcast lists. Each broadcast list enlists a set of friends whom she can broadcast messages to. If you have noticed, the list of broadcast lists is a nested list structure. We will also implement the facility for the user to add more friends to her contact list from the global contact list. She can remove friends too. Also, she should be able to create new broadcast lists and add friends to the new list. She will be able to modify existing lists by adding/removing friends. She can delete entire lists too! All this will be accomplished from the

console using commands. We will pre-populate the global contact list. Also, a few chat conversations will be populated. A few users will have broadcast lists too. This is to help you get started fast.

For this programming assignment you won't be building an abstract data type (ADT) from scratch. Instead, you'll use Java's `ArrayList` class to implement the classes as specified below.

Specifications

Pre-population Input File

Text file storing user profile information and messages will be read by your program.

A sample is provided here.

Sample output

A sample output is given here. This assumes that you have used the above sample prepopulation file as the input to the program when you start your program

A pointwise explanation of each type of line is as follows.

- **user** - lines beginning with "user" describe users. The first line says the last name of the user is reed, first name is patricia, unique nickname is preed and the password is welcomePREED. The set of all user lines forms the global contact list.
- **flist** - lines beginning with "flist" describe the friends list of a user. The eighth line says that the user whose nickname is preed has rdixon, lday and jhartc as friends whom she can contact. Each user has at most one of these lines.
- **bcast** - lines beginning with "bcast" describe a single broadcast list owned by a user. The tenth line describes a broadcast list owned by user rdixon. The descriptive name of the list is "best friends". The unique nickname of the list is rdixbf. Users with unique nicknames preed and kelliott are member of this list. rdixon sends a single message to the list and both preed and kelliott get the message. rdixon has another list called rdixwp. So, in essence, rdixon has two broadcast lists. The list of these two lists is a nested list. Clear now?
- **message** - lines beginning with "message" describe a message. The one on line 13 describes a message from rdixon to lday sent at "01/19/2016 23:34:52" which says "Lets meet today". This message has not been read by lday. The one on line 14 describes a message from rdixon to one of his broadcast lists rdixbf. This message has been read by preed while it is yet to be read by kelliott.

*Note: for names, differences in capitalization should be ignored. After reading them from the input files or the console, convert them to lowercase before taking any further action. E.g. "`BirdsWithArms`" becomes "`birdswitharms`".

Information about Java I/O is available in both a summary version and a detailed version. Examples of file input are linked to in those documents.

Simulation Flow

The following points describe the flow of the simulation.

- When the program is invoked, it reads the pre-population file and sets up the state. It then prompts for the nickname of the user and validates the password for a successful login. The loop continues till a successful login is done.
- The program then stores the nickname in a special Session class which we will describe later. The program subsequently enters a read-execute loop on the console. In simpler words, it reads commands from the logged in user and executes them till the user logs out.
- After logging out a user the login prompt is shown again.

Command syntax

The following is a brief description of the set of commands you need to support and implement. The exact error messages that should be displayed are part of the javadocs of the

CommandProcessor class. Look at the method descriptions of that class for more details. More than one of the following commands map to a single method in the CommandProcessor. For example, both the search fn: and the search ln: commands are taken care of by the search method. When we say display INVALID_COMMAND or anything like than in all caps, you need to display that constant from the StringConstants class.

- **send message: nickname,"message in quotes"** - this sends the message to the nickname. This can be a nick name of a broadcast list also.
- **read messages unread from: nickname** - this displays the unread messages from nickname. This must also internally mark those messages as read. Please note that in this command the nickname cannot be a broadcast list nickname. Broadcast lists are for one way broadcasts. The people in the broadcast see the messages coming from the owner of the broadcast. The receivers have no idea if the message came from a broadcast.
- **read messages all from: nickname** - similar as above, but displays all messages including the ones sent by this user to that nickname. This must also include the sent messages from any broadcast lists that the nickname is a member of.
- **read messages unread:** - this displays all unread messages from all nicknames known to the logged in user sorted by the time of sending earlier first
- **read messages all:** - same as above, but displays all messages from all nicknames known to the user. This must also include all sent messages including the ones from all broadcast lists.
- **search fn: single-word** - searches both the global contact list and the friends list for people whose first names contain the single-word after doing a case insensitive search.
- **search ln: single-word** - searches both the global contact list and the friends list for people whose last names contain the single-word after doing a case insensitive search.
- **add friend: nickname** - adds nickname to the friends list
- **remove friend: nickname** - removes nickname from the friends list. This must additionally remove this friend from all broadcast lists owned by this user, if nickname was a part of them. However, old messages are not deleted.
- **add to bcast: nickname0,nickname1** - adds friend nickname0 to the broadcast list nickname1 owned by the user
- **remove from bcast: nickname0,nickname1** - removes frinend nickname0 from the broadcast list nickname1 owned by the user
- **remove bcast: nickname** - deletes the broadcast list nickname owned by the user
- **logout:** - logs out the user and displays the login prompt again.
- **exit:** - exits the program. This should only work when some user is logged in. It should not work from the login prompt.

All Classes

The javadocs for all classes provided to you is here. You are expected to complete those sections of it marked as "//TODO".

You may ***not*** add any other public methods than those listed in the provided files.

You may ***not*** modify any class in any way except where noted as "//TODO".

Console Input/Output

For purposes of thie project this is how you would be accepting user input and showing your messages on the console. You MAY NOT use any other mechanism to do input/output in this project. You MUST use the objects provided in the Session class.

For Input

String nickname = Config.getInstance().getConsoleInput().nextLine();

For Output

For simpler constants, use

Config.getInstance().getConsoleOutput().printf(StringConstants.ENTER_COMMAND);

For positional strings, which require substitution at appropriate places, use

Config.getInstance().getConsoleOutput().printf(StringConstants.USER_DISPLAY_FOR_SEARCH,

firstName, lastName, nickName, "yes");

Command-line Arguments

Recall that in Java, when you run a program it is a `main` method that runs and that `main` method always has the following header:

```
public static void main(String[] args)
```

The array-of-`Strings` parameter, `args`, contains the *command-line arguments* that are specified when the program is run. If the program is run from a Linux prompt (i.e., not from a programming environment like Eclipse), the command-line arguments are simply typed after the name of the class whose main method is to be executed. For example:

```
java YY a.txt b.txt c.txt
```

runs the Java interpreter on the `main` method of the `YY` class, passing the strings "a.txt", "b.txt" and "c.txt" as the command-line argument.

To use command-line arguments when you run a Java program using Eclipse:

1. Right click on the source file that contains the main class you want to run in the "Package Explorer" window.
2. Select "Run As" from the pop-up menu.
3. Select "Run ..." from the pop-up menu, which brings up the "Run" window.
4. Click on the "(x)= Arguments" tab.
5. Enter the arguments in the "Program arguments:" text box.
6. Click either the "Run" button or the "Apply" and "Run" buttons.

Steps

After you have read this program page and given thought to the problem we suggest the following steps:

1. Pair Programming is not allowed on program p1.
2. Review these style and commenting standards that are used to evaluate your program.
3. You may use the Java development environment of your choice in CS 367. ***However, all programs must compile and run on the lab computers for grading.*** If you are going to use the CS lab computers, we recommend that you use Eclipse. You may want to review the Eclipse tutorial to learn the basics. Note that on the Linux lab computers, you should enter "`eclipse&`" at the prompt instead of what is described in the tutorial.
4. Download the files `here` to a programming assignment p1 folder that you make.
5. Complete (by filling //TODOs) and thoroughly test all the given classes.
6. If you are not using the lab computers to develop your program, make sure you compile and run your program to ensure that it works on the Linux lab computers. You can compile your Java source using `javac` in a terminal window as in this example:

   ```
   javac *.java
   ```

   and the run your program using `java` as in:

   ```
   java WhatsApp path-to-pre-population-file
   ```

7. Submit your work for grading.

Submitting Your Work

Make sure your code follows the style and commenting standards used in CS 302 and CS 367.

All submitted classes should belong to the default package.

Submit the following files by the due date and time (or refer to the late policy) using the 367 Forms:

- BroadcastList.java

- CommandProcessor.java
- Helper.java
- Message.java
- User.java

Last updated: 2/8/2016