CSE 114 Fall 2022: Assignment 8

Due: November 17, 2022 at 11:59 PM [KST]

Read This Right Away

For the due date of this assignment, don't go by the due date that you see on Brightspace because it is in EST. Go by the one given in this handout.

Directions

- At the top of every file you submit, include the following information in a comment
  - Your first and last name
  - Your Stony Brook email address
- Please read carefully and follow the directions exactly for each problem.
- Your files, Java classes, and Java methods must be named as stated in the directions (including capitalization). Work that does not meet the specifications may not be graded.
- Your source code is expected to compile and run without errors. Source code that does not compile will have a heavy deduction (i.e. at least 50% off).
- You should create your program using a text editor (e.g. emacs, vim, atom, notepad++, etc).
- You may use the command-line interface to compile and run your programs or you can now use IntelliJ IDEA.
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied ***at the time the assignment is posted***. If you have a question about features you may use, please ask!

What to Submit

Combine all your .java files from the problems below into a single zip file named as indicated in the **Submission Instructions** section at the end of this assignment. Upload this file to **Brightspace** as described in that section.

Multiple submissions are allowed before the due date. Only the last submission will be graded.

Please do **not** submit `.class` files or any that I did not ask for.

Naming Conventions In Java And Programming Style In General

Please use these conventions as you establish your *programming style.* Programming professionals use these, too.

- **Names:** Choose informative names,
    - e.g. `hourlyRate` rather than `hr` for a variable name.
    - `CheckingAccount` rather than `CA` for a Java class name.
- **Class names:** We start a class name with an uppercase letter as I have in my examples: Hello not hello or `HELLO`. For multi-word names you should capitalize the first letter of each word,
    - e.g. `UndergraduateStudent, GraduateStudent, CheckingAccount`
- **Variable names:** We start a variable name with a lowercase letter and capitalize each 'word' after that. This is called 'Camel case':
    - e.g. `count, hourlyRate, averageMonthlyCost`
- **Method names:** Naming convention for method names is the same as that for variable names.
- **Use of white space:** Adopt a good indentation style using white spaces and be consistent throughout to make your program more readable. Most text editors (like emacs and vim) should do the indentation automatically for you. If this is not the case, see me and I can help you configure your setup.

I will not repeat these directions in each assignment, but you should follow this style throughout the semester.

Problem 1 (25 Points)

Goal

The goal of this problem is build on what you learned in the last assignment. This will exercise your knowledge of building and utilizing arrays of objects as well as passing objects and arrays to methods and returning objects and arrays.

Start with your files from your Assignment7 submission. If there were any bugs that you identified after submission, fix those bugs first!

You will be building two new objects.

- **EmailHost** – This class acts like a mail delivery system and holds an array of messages to be delivered to users. Each object represents 1 hostname (so "sunykorea.ac.kr", "gmail.com", etc) and it handles email only for that machine. Email that would be sent there has a *to* address of <someuser@hostname> (i.e. 'antonino.mione@sunykorea.ac.kr' would go to the EmailHost object handling mail for sunykorea.ac.kr.) Methods in this class are detailed below.
- **Controller** – This class is used to managse EmailHosts. It will keep a list of registered machines that handle email. (See the provided UseController.java). Controller will provide methods to
    - Register an EmailHost object (puts it in a list of objects)
    - Find the host to which an email should be sent given the email address. This will return the EmailHost object that handles mail to that address.

EmailHost

This class will have the following members:

capacity – This is a constant integer ('final') which indicates the maximum amount of messages that can be held by the host (you can set this around 100 or so).

messages – This is an array of Message objects that hold the messages delivered.

nextMessage – This is an integer that tracks the next available slot in the message array.

hostname – This is a String with the name of the host.

Methods you should implement include:

A getter method for hostname. For the most part, none of the other members need getters.

send(Message msg) – This takes the message and adds it to the list of messages being held for delivery by the host. It should check the 'to' email address in the message and assure it is destined for this machine. If not, print an error (i.e. 'Bad Host') and do not add the message. Just return.

getMessagesForUser(String userEmail) – This will return an array of Message objects where the message's 'to' address matches the userEmail String passed to the method. You should return an array of the exact size to hold the returned objects. This means you may have to loop through the message array and count how many messages go to the user, then create the return array and fill it. Also, you can create a large array and copy messages into that 'temporary' array. Then, check how many messages were actually added to it, create a new return array and copy the message objects into that new array to be returned to the caller.

Controller

This class provides the following members:

- **eHosts** – This is an array of EmailHost objects. Make the number of slots in the array configurable with an integer passed to the constructor.
- **nextHost** – This should be the next open slot in the eHosts array. Everytime you add (register) a node, it should be incremented.

Regarding methods, it should provide:

- A constructor taking 1 integer as an argument which is the capacity of the EmailHost array (number of hosts it can manage).
- **boolean registerHost(String hostName, EmailHost eHost)** – This adds the host to the array of EmailHosts. It returns true if the operation succeeded and false otherwise. Reasons the operation may fail include the host already being in the EmilHost array or the array exceeding its capacity.
- **EmailHost findHost(String hostname)** – This looks up the EmailHost by name in its array. It returns the actual EmailHost object.
- **EmailHost getHostFor(String address)** – This finds a host by email address. This means it will have to split the string and check only the hostname to find the EmailHost object needed. Like findHost(), It returns the EmailHost object. This is needed since after looking up the host, UseController will call the .send() method on that host object to set up mail delivery.

Implementation notes

Read through the attached UseController.java module. This code will test your implementation (it contains some of what you wrote for the original UseMessage.java code from last assignment but it adds calls to test Controller and EmailHost functionality.) Make sure you thoroughly understand how these calls work and compare them with the output provided. This will be your guide as to how to develop the methods given above for each class.

Note that you are allowed to add additional 'helper' methods in both Controller and EmailHost if you feel they would be useful to modularize the code.

You may add tests to UseController but add them at the end of the existing code. Do not change any of the provided calls. You code should be able to produce results like those shown in this assignment (See output.txt attached to the assignment.)

Hand in `Message.java, EmailHost.java, Controller.java,` and `UseController.java` (if you add more tests to it).

Submission Instructions

Please follow this procedure for submission:

0. Place the deliverable source files into a folder by themselves. The folder's name should be CSE114_HW8_<yourname>_<yourid>. So if your name is Alice Kim and your id is 12345678, the folder should be named 'CSE114_HW8_AliceKim_12345678'

1. Compress the folder and submit the zip file. ````
   a. On Windows, do this by clicking the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace.
   b. On Mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace.
2. Navigate to the course Brightspace site. Click **Assignments** in the top navbar menu. Look under the category 'Assignments'. Click **Assignment8**.
   a. Scroll down and under **Submit Assignment**, click the **Add a File** button**.**
   b. Click **My Computer** (first item in list).
   c. Find the zip file and drag it to the 'Upload' area of the presented dialog box.
   d. Click the **Add** button in the dialog.
   e. You may write comments in the comment box at the bottom.
   f. Click **Submit**. 🡐 Be sure to do this so I or the grading TA can retrieve the submission!