Homework 06 - Forum Site

Authors: Andrew Chafos, Vince Li, Jack Kelly, Hussain Miyaziwala, Shishir Bhat

Problem Description

It is another hot, sunny morning at Georgia Tech. As you groggily walk to your CS 1331 Lecture, you notice a bizarre, hazy mist in front of your path. Curious, but unfazed, you decide to keep walking, straight into the mist. However, once you step through the other side, the scene entirely changes! Van Leer looks much newer, an entirely different set of students walk around you, and it is now a rainy day. The CULC has mysteriously disappeared from sight. Before long, you realize what has happened: you have been transported to the year 1998! The police would never believe you were from the future, and even if they did, you may be seized and quarantined by the government or scientific community to protect the timeline. No, instead you head over to the nearest library, which fortunately has public computers available for use, and decide to develop your own forum site, to see if any other strangers across the internet have experienced the same bizarre time traveling phenomenon. Later you may try to contact relatives, but right now, this is your top priority. First things first, you must develop a secure backend for your site, and the cornerstone of this backend will be the User class. Equipped with the JDK version 1.1 documented in a textbook from the library and your knowledge of Java from CS 1331, you start working!

Solution Description

For this assignment create a file named `User.java`. In this file, you will be creating a number of fields and methods. Based on the description given for each field and method, you will have to decide whether or not the field/method should be static, and whether it should be private or public.

Variables:

- `String username`
  - `username` represents the String name of a given User object. If you create multiple User objects, each should be able to have its own unique username (but note that 2 User objects *could* have the same username; your code should simply enable 2 User objects to have different usernames if desired). This variable should not be able to be modified outside of the User class.
- `int password`
  - `password` represents the password of a given User object. This variable should be able to be different for different User objects, and it should not be able to be modified outside of the User class (it may be modified within the User class, however).
- `int numUsers`
  - `numUsers` represents the total number of User objects that have been instantiated. This value should be consistent across the User class, only changing when a new User is created. It should not be able to be modified outside of the User class.
- `User newestUser`
  - `newestUser` represents the most recently created User object. It, too, should be consistent across the User class, only being changed when a new User is created. It should not be able to be modified outside of the User class.
  - Since there will be no User instances when the program begins, this variable can be set to `null` by default.
- `boolean displayNewest`
  - `displayNewest` determines the behavior of the method `getWelcomeMessage()`. It should be the same value across all User objects. It should not be able to be modified outside of the User class,

but it will be in its setter `setDisplayNewest(boolean displayNewest)`, which is found within the User class.
  - This variable should be set to `true` by default!

The Constructor:

The constructor should only take in values for `username` and `password` and set each of them respectively. `numUsers` should be incremented, and `newestUser` should be set to the newest User object created (Hint: what keyword in Java would let us refer to the current User object?).

The Methods:

All of the following should be able to be called from outside of the User class. However, some are static, and some are non-static. You must choose the option that makes the most sense.

- `void setDisplayNewest(boolean displayNewest)`
  - This method takes in `displayNewest` and sets the User variable `displayNewest` to it.
- `int getNumUsers()`
  - This method takes no parameters and returns the number of User instances (what variable would give us that?).
- `String getUsername()`
  - This method takes no parameters and returns the username of this particular User instance
- `String getWelcomeMessage()`
  - This method takes no parameters and returns a `String`. This method will return a different String message depending on the states of some variables found in the User class.
  - If there are currently no User instances, the returned string should match this: `"This site doesn't have any users yet!"`
  - If there is at least one User instance, and `displayNewest` is false: `"Welcome to our site! We have [numUsers] user(s) and counting!"`
  - If there is at least one User instance, and `displayNewest` is true: `[newestUsername] just joined our site! Please give them a warm welcome!`, where `newestUsername` is the String username of the newest User.
  - Example: `"Bob just joined our site! Please give them a warm welcome!"`
- `void changePassword(String usernameInput, int passwordInput, int newPassword)`
  - This method takes in `usernameInput`, `passwordInput`, and `newPassword`. If `usernameInput` matches the username of the current User instance, and `passwordInput` matches the password of the current User instance, then set the current User instance's `password` variable equal to `newPassword`.
- `boolean validLogin(String usernameInput, int passwordInput)`
  - This method takes in `usernameInput` and `passwordInput`. If `usernameInput` matches the username of the current User instance, and `passwordInput` matches the password of the current User instance, return `true`. Otherwise, return `false`.

Note: Each method requires proper JavaDocs (see below for more information)

Testing your code

If you want to test out your own code before submitting we recommend creating a separate java file and implementing a main method there. You can then create new User objects and test your methods. Below is some sample testing code. **These tests are not comprehensive!**

```
User.getWelcomeMessage()
-> This site doesn't have any users yet!
```

```
User user1 = new User("Bob", 12345);
User.getWelcomeMessage()
-> Bob just joined our site!  Please give them a warm welcome!
user1.getUsername()
-> Bob
user1.validLogin("Bob", 1273)
-> false
User user2 = new User("Karen", 456);
User.setDisplayNewest(false);
User.getWelcomeMessage()
-> Welcome to our site!  We have 2 user(s) and counting!
```

Feel free to create your own tests in the `main` method! Try to write tests for the remaining methods in the file!

Rubric

- [10] User variables
  - [5] Correct usage of static/non-static for each
  - [5] Correct visibility for each
- [20] Constructor
  - [10] Correct setting of `username` and `password`
  - [5] Increments `numUsers`
  - [5] Updates `newestUser`
- [10] `setDisplayNewest`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Works as expected
- [10] `getNumUsers`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Works as expected
- [10] `getUsername`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Works as expected
- [15] `getWelcomeMessage`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Works when there are no User instances
  - [5] Correctly changes depending on `displayNewest`
- [15] `changePassword`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Correctly evaluates the input
  - [5] Correctly updates the password
- [10] `validLogin`
  - [5] Correct usage of static/non-static and correct access modifier
  - [5] Works as expected

Allowed Imports

To prevent trivialization of the assignment, you are *not* allowed to import any classes or packages.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- var (the reserved keyword)
- System.exit

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 1.0
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b) {
    ...
    }

}
```

A more thorough tutorial for Javadocs can be found here.

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.

3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.

To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **25 points**. This means that up to 25 points can be lost from Checkstyle errors.

For this homework we will not count off for the checkstyle issue of `"'username' hides a field"` or `"'password' hides a field"`

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `User.java`

Make sure you see the message stating "HW06 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications