Homework 01 – Intro to Variables and Operations

Problem Description

You have now been introduced to the basics of programming in Java. This means you can write simple, but useful programs! We are going to practice using those skills by creating and manipulating several variables and of different types. For this homework, we will give you broad strokes of what to do. You will then implement the directions in Java code within two classes – one called PrimitiveOperations, and the other called StringOperations.

Refer to HW00 and Module 1 for a refresher on creating classes in Java. All of your code should execute inside the main method of each class – remember that a properly written main method in Java has a very specific method header.

Make sure you read the entire document! There are some things even at the very end that you will not want to skip for the current and future assignments.

Solution Description

This homework will be split into four sections. The first two sections involve working with the various primitive types in Java and then using methods that come with the String class. The third section involves finally basic debugging on erroneous code that is provided. Finally, the fourth section is a short task that will help the TAs get to know you better.

Section 1: Primitive Operations

All instructions in this section should be carried out in the class `PrimitiveOperations`, in the main method, and must be in the respective order

- Declare and initialize two variables, an integer type (byte, short, int, or long) and a floating-point value (float or double).
    - The names and values of the variables can be your choice, both for this step and all other steps.
    - Make sure that the numbers you choose can be stored within the respective primitive type you choose.
    - Print each of these values out on their own line using `System.out.println()`.
- Multiple these two variables together and assign the product to a new variable, ensuring that no data is lost (i.e., when multiplying 5 and 3.5, the answer should be 17.5).
    - Print out this new value.
- Use casting to convert the integer from the first step to a floating-point value and store that in another new variable.
    - Print out the value.
- Use casting to convert the floating-point value from the first step to an integer type and store that in a new variable.
    - Print out the value.
- Shifting focus, declare a char variable and assign an uppercase letter to it.
    - Print out this value.
- Using a *numerical operation*, change the letter to the same letter, but in lowercase.
    - Use a numerical operation – do not simply reassign the variable.
    - Hint: You may want to review a table of ASCII values for this part, and you will likely have to use casting to get this to work.

    - Print out the new char value.

Section 2: String Methods

All instructions in this section should be carried out in the class `StringOperations`, in the main method, and must be in the respective order

- Create a new String object and assign it your name.
    - Print it out.
- Replace the first letter of the String with 'A'. Next, replace the last letter of the String with 'Z'.
    - Recall: In Java, Strings are immutable, meaning you cannot change a String in-place.
    - Do NOT just hard-code a new String with the first and last letters changed.
    - Print out the result.
- Lastly, let's work with some URLs. Declare a new String and give it the value of a web address.
    - The web address should be in the form `www.name.tld`, such as `www.gatech.edu` or `www.stackoverflow.com`
    - Print out this address.
- This last operation could be a little tricky. Create a substring of the variable that only consists of the "name" part of the URL, then concatenate the integer "1331" to the end.
    - For example, `www.gatech.edu` would become `gatech1331`.
    - Again, do not hard-code the new String value.
    - Hint: The String class has a `.length()` method which you will likely find useful here but is not necessary.
    - Print out this result.

Section 3: Debugging

For this homework, we have provided three Java programs: Bad1.java, Bad2.java, and Bad3.java. Each program contains a single error.

- Attempt to compile and run each program and observe the resulting error message.
- Create a file named `errors.txt`
- Within this text file, state whether each error occurs at compile time or runtime.
- Write a single sentence for each bad program which describes the reason for that error.
- Now fix whatever is wrong with the programs so that you are able to compile and run them.
    - Some fixes may be open-ended. For those, any fix is allowed, as long as the program successfully runs.
- You will submit `errors.txt` and the fixed versions of `Bad1.java`, `Bad2.java`, and `Bad3.java`
- Note: The error is in the files' content. **Do not change the filenames for any file**

Section 4: Tell us who you are!

As TAs, we are here to help you be successful in this course. A good first step is for us to learn a little more about you and put names to the faces in our section.

- Along with your submission, please attach a picture of yourself (optional) a text file named `me.txt`
- In your `me.txt` file, write a few sentences about yourself (e.g., your major, how long you have been at Georgia Tech, a personal interest, etc.), as well as why you have chosen to take the online asynchronous version of CS1331. If you have a preferred name that's different from the one in Canvas and you would like to share it, let us know it here

Rubric

[25] `PrimitiveOperations.java`

- [5] Program is able to be run (main method)
- [5] Correct variables
- [15] Correct output

[25] `StringOperationsjava`

- [5] Program is able to be run (main method)
- [5] Correct variables
- [15] Correct output

[36, 12 per file] `Bad.java files and errors.txt`

- The rubric items below apply for each Bad.java file
- [7 per file] Describes each error within errors.txt
  - [3 per file] Correctly identifies whether each error occurs at compile time or runtime
  - [4 per file] Writes a sentence explaining each error
- [5 per file] Corrects each java file

[14] `me.txt`

- [14] Includes a few sentences to introduce yourself!

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

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

Feature Restriction

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`

Collaboration

Collaboration Statement

To ensure that you acknowledge a 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 assignment alone, using only course-provided 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].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

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

- PrimitiveOperations.java
- StringOperations.java
- Bad1.java
- Bad2.java
- Bad3.java
- errors.txt
- me.txt
- me.png (optional)

Make sure you see the message stating "HW01 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:

- Prevent upload mistakes (e.g. non-compiling code)
- 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.

For future assignments, other portions of your assignment are also autograded once the submission deadline has passed. The autograders used are often dependent on specific output formats, so make sure your code passes the visible tests that check whether specific classes/methods exist (correct spelling, etc.).

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Make sure that all class names and method names are spelled correctly
- 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 a note containing all official clarifications