The University of Melbourne
School of Computing and Information Systems
COMP90041
Programming and Software Development
Second Semester, 2019

First Assessed Exercise (lab1)

Submission due Friday, 16 August 2019, 5:00PM

These exercises are to be assessed, and so **must be done by you alone**. Sophisticated similarity checking software will be used to look for students whose submissions are similar to one another.

See below for instructions on using command line arguments in Java.

1. Write a program `Welcome.java` that expects two string command line arguments, a first name and a second name. The program prints out on one line "`Hello` *first second*`.`", and on the next line, "`Is that` *first SECOND* `or` *FIRST second*`?`" where *first* and *second* are the first and second command line arguments, and *FIRST* and *SECOND* are the first and second arguments printed in all upper case letters. Do not print any extra spaces, even at the beginning or ends of the lines. Do not worry about too many or too few command line arguments. For example, if the command line arguments are "`Joe`" and "`Blow`", the program should print out:

   ```
   Hello Joe Blow.
   Is that Joe BLOW or JOE Blow?
   ```

2. Write a program `Circle.java` that prints out the area of a circle whose *diameter* is specified as the only command line argument. The result should be formatted with four digits after the decimal point, and no extra characters or spaces before the decimal point. Don't include any extra characters in the output. The command line argument may be written as an integer or as a floating point number. Do not worry about too many or too few command line arguments, or the command line argument not being a number. For example, if the command line argument is specified as 2, the output should look like:

   ```
   3.1416
   ```

   Hint: The area of a circle of *radius* r is πr^2.

   Hint: You can use `Double.parseDouble(`*s*`)`, where *s* is a string, to turn a string representing a floating point number into the double it represents.

Using Command Line Arguments

The command line of a Java program is whatever is placed on the command line after `java` *programname*. Each word or number placed there is made a separate command line argument when the program is run. If you want multiple words, or text with whitespace or special characters, to be treated as a single argument, you can surround it with single quote characters ('). So for example, if your command line is:

```
java Program 42 hello there 'this is a test'
```

when the program is run, it will receive 4 command line arguments: "42", "hello", "there", and "this is a test".

Inside a Java `main` method

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

the variable `args` holds all the arguments passed to the Java program on the command line as strings. You can access the first command line argument as `args[0]`, the second as `args[1]`, and so on. We will learn why this works in a few weeks, but for now, just use that syntax to access command line arguments.

Submission and Verification

You must submit your project from any one of the student unix servers. Make sure the version of your program source files you wish to submit is on these machines (your files are shared between all of them, so any one will do), then `cd` to the directory holding your source code and issue the command:

```
submit COMP90041 lab1 Welcome.java Circle.java
```

Important: you must wait a minute or two (or more if the servers are busy) after submitting, and then issue the command

```
verify COMP90041 lab1 | less
```

This will show you the test results and the marks from your submission, as well as the file(s) you submitted.

If your output is different from the expected (correct) output, when you verify your submission you will see the differences between your output and what was expected. This will be shown as some number of lines beginning with a minus sign (`-`) indicating the expected output and some number of lines beginning with a plus sign (`+`) presenting your actual output. There may also be some lines beginning with a single space showing lines you produced that were as expected. Carefully compare the expected and actual lines, and you should be able to find the error in your output. The actual and expected outputs will be aligned, making it easier to find the differences. Some differences are hard to spot visually, however, such as the difference between a capital `O` and a zero (`0`) or the difference between a small `l` and a capital `I` and a one (`1`). This depends on the font you are using. If the only difference between actual and expected output are in whitespace or capitalisation, you will receive partial credit; this is shown in your verification feedback.

Also note that the differences shown only reflect program *output*, not input. Therefore, if your program outputs a prompt, waits for input, and then outputs something else, the differences shown will not include the input, or even the newline the user types to end the input. In that case, the prompt would be shown immediately followed by your program's next output (which may be another prompt), on the same line. This is as expected.

If your program compiles on your computer but the verification output reports that your program does not compile on the server, you may have failed to submit all your files, or you may have named them incorrectly. It is also possible that your program contains a `package` declaration. This would appear near the top of your .java file. If you have such a declaration, your program will probably not compile, so you should delete any such declaration before submitting.

If the verification results show any problems, you may correct them and submit again, as often as you like; only your final submission will be assessed. If your submission involves multiple files, you must submit *all* the files every time you submit.

If you wish to (re-)submit after the project deadline, you may do so by adding "`.late`" to the end of the project name (*i.e.,* `lab1.late`) in the `submit` and `verify` commands. But note that a penalty, described below, will apply to late submissions, so you should weigh the points you will lose for a late submission against the points you expect to gain by revising your program and submitting again. **It is your responsibility to verify your submission.**

Late Penalties

Late submissions will incur a penalty of 1% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is a little more than 2 days late will lose half the marks. These lab exercises are frequent and of low point value, and your lowest lab mark will be dropped. Except in unusual circumstances, I will not grant extensions for lab submissions.

Academic Honesty

This lab submission is part of your final assessment, so cheating is not acceptable. Any form of material exchange between students, whether written, electronic or any other medium, is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties.