The University of Melbourne
School of Computing and Information Systems

COMP90041
Programming and Software Development
Second Semester, 2019

Second Assessed Exercise (lab2)

Submission due Friday, 30 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.

1. Write a program `Efficiency.java` that computes the fuel efficiency, in litres per hundred kilometres, of a vehicle. It should print out the prompt `Enter␣vehicle␣make:␣` (where spaces are shown as ␣) and read in a vehicle make (which may be more than a single word) as a string. Then it should print out the prompt `Enter␣vehicle␣model:␣` and read in a vehicle model (which may also be more than a single word) as a string. Next it should print out the prompt `Enter␣kilometres␣travelled:␣` and read in the number kilometers traveled (use a `double`). Next it should print out the prompt `Enter␣litres␣of␣fuel␣used:␣` and read in the number litres of fuel consumed (use a `double`). Note that all of these prompts should end with a single space, and no newline, so that the value to be entered is typed on the same line as the prompt. For input of both numbers, users should be permitted to type in numbers with or without a decimal point. Finally the program should print `Fuel␣efficiency␣for␣a␣` followed by the make and model, separated by a space, and then `:␣` followed by the fuel efficiency in litres per hundred kilometers, shown to two decimal places, followed by `␣litres/100␣km`. For example, a run of this program might look like this, with user input underlined:

```
Enter␣vehicle␣make:␣Ferrari
Enter␣vehicle␣model:␣458 Spider
Enter␣kilometres␣travelled:␣593
Enter␣litres␣of␣fuel␣used:␣86.13
Fuel␣efficiency␣for␣a␣Ferrari␣458␣Spider:␣14.52␣litres/100␣km
```

   Important: If you use the `Scanner` class for this question, or any other you submit for assessment, **you must not** create more than one instance of the `Scanner` class. You may use the same instance of `Scanner` as many times as you like.

2. Write a Java program called `Factors.java` that takes one integer argument on the command line, and prints out all of the factors of that number from 1 up to that number on a single line, separated by spaces. For example, if given the command line argument `60`, the program should print out the single line:

1␣2␣3␣4␣5␣6␣10␣12␣15␣20␣30␣60

Once again, spaces in the output are shown as ␣. Note there is no extra space at the beginning or end of the line, and the line ends with a newline character immediately after the `60`.

You do not need to handle integers larger than 100,000,000 or smaller than 1, so an `int` is large enough. Your program must run in under 10 seconds for each test or it will time out and you will fail that test. Try implementing it the simplest way you can think of, and I think you will find that 10 seconds is plenty.

Hint: You can check if an integer `a` is divisible by an integer `b` by checking if `a % b` is 0. If it is, then `a` is divisible by `b`.

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 lab2 Efficiency.java Factors.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 lab2 | 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.,* `lab2.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.