tags: UCSD CSE11 SS1-2022, PA1

CSE 11 SS1-2022 PA1 - Covid Transmission

Due date: Tuesday, July 5 @ 11:59PM PST

No late submission except for legitimate and approved special cases

Provided Files

None.

Files to Submit

- CovidTransmission.java

Goal:

Programming Assignment 1 is an introduction to Java programming. In this PA, you will get exposure to primitive data types, variables, keyboard input, console output, and if-else logic.

Please read the **entire write-up** before getting started.

Some General Notes

- Most of these notes are necessary for autograding purposes. If any of these does not make sense, you probably aren't doing anything that it applies to. However, you should make sure that when you do learn about it later on in this quarter that you follow these notes. We cannot be lenient regarding these things because everything is autograded to ensure fairness.
- Make sure to read the autograder output after you submit to Gradescope (wait until the autograder is finished running).
- Match the specifications that we provide *exactly*, otherwise we cannot ensure that the autograder will function correctly. This includes file names, class names, method signatures, extending, throwing, etc.
- "Constants" should be defined as `private static final` variables.
- Do not use *any* static variables (other than for constants) or instance variables that are not specified in this writeup. We cannot ensure that these do not get clobbered during grading. Any extra variables used should be local only.
- Unless otherwise specified, do not add any extra `import`s or use any packages not from `java.lang` other than those implied or explicitly named by this writeup.
- Do not specify a package for your files. This will cause them to fail to compile with the autograder.
- Do not add any extra classes to your files and do not write code in files that are not specified.
- Do not call helper methods except from the class where they are implemented, as we will be using our own version of classes during grading (which will only have the instance variables and methods specified in this writeup). You shouldn't be able to call them anyway if you have declared them as `private`.
- If a behavior (say, on some specific input) is not specified, you may handle that case however you want. However, if you implement a specific behavior for some special case of a required method, do not rely on that specific behavior when using that method as a helper method (only assume that the method works as specified in the writeup).
- For the AI form, make sure you are filling them out and specifying your email address as the one linked with your Gradescope account. If you fill them out after submitting, you can either resubmit to update your score immediately or wait for us to rerun the autograder for everyone after the deadline.

Part 1: Class Logistics

Academic Integrity

We take Academic Integrity (AI) very seriously at UCSD. Before beginning the assignment, please complete the AI form:

https://forms.gle/PPqXhzj88dEGY5gf7

You must fill this out in order to receive autograder points for any assignment.

Part 2: Style

In this summer session, styles will not be counted for points. You will not lose points for inappropriate styling, but it's good to develop a style habit that will benefit you in the future.

You can keep these guidelines in mind as you write your code or go back and fix your code at the end.

Coding style is an important part of ensuring readability and maintainability of your code. We will grade your code style in all submitted code files according to the style guidelines. Namely, there are a few things you must have in each file/class/method:

1. File headers
2. Class headers
3. Method headers
4. Inline comments
5. Proper indentation (do not intermingle spaces and tabs for indentation)
6. Descriptive variable names
7. No magic numbers
8. Reasonably short methods (if you have implemented each method according to specification in this write-up, you're fine)
9. Lines shorter than 80 characters (note, tabs will be counted as 4 characters toward this limit. It is a good idea to set your tab width/size to be 4. A good way to check is using the command `grep -n '.\{81,\}' *.java` in the directory with your files, but note that this won't necessarily take tabulation into account appropriately.)
10. Javadoc conventions (@param, @return tags, /** header comments */, etc.)

A full style guide can be found here. If you need any clarifications, feel free to ask on Edstem.

Part 3: Setup

Java

You will need Java installed on your machine in order to do this assignment:

Java Environment Setup: (instructions, video)

If you need help on Setup, feel free to ask for help either in lectures, discussions, and office/tutor hours.

Shell

Make sure you have access to bash/Z shell/some shell that you can use to run the `javac` and `java` commands (or use an IDE: what is IDE):

- Windows: https://gitforwindows.org/
- Mac: Terminal
- Unix: Terminal

Feel free to familiarize yourself with some commands (optional):

Essential Commands: https://www.hongkiat.com/blog/web-designers-essential-command-lines/

Unix reference sheet: https://files.fosswire.com/2007/08/fwunixref.pdf

Part 4: CovidTransmission.java

Disclaimer: These are sample statistics that do not accurately reflect COVID-19 transmission. Please follow the CDC guidelines to stay up to date on the coronavirus pandemic: https://www.cdc.gov/

The Centers for Disease Information (CDI) has recently found a way to compute risk for contracting COVID from someone who has tested positive. The risk is computed based on the amount of time the two individuals spent together. However, all the programmers that worked for the CDI left to work at Apple for much more money (and free snacks), so the CDI now needs your help to compute COVID risks.

Task

```
public class CovidTransmission {
    ...
    public static void main(String[] args);
    ...
}
```

In the `main()` method of the `CovidTransmission` class in the `CovidTransmission.java` file, implement the following logic that outputs the number of minutes that a person A and a person B are in contact with each other and the risk level of the virus transmitting. Note that when we write "`someMethod()`" in the write-up, we are referring to the method with the name `someMethod` but it does not necessarily have no arguments, such as in the case of `main()`.

Input

The array `args` always contains the arguments passed in from the command line. For example, if the program were run with the command `java CovidTransmission 1 Hello 20`, then `args` array would be `["1", "Hello", "20"]`. However, for this `main()` method, we will not use whatever is in the `args` array.

Instead, we will read user input from `System.in` using a `Scanner` (remember to import it). When a user runs `java CovidTransmission`, they will then type the input to our program. For this assignment, you can safely assume that this input will come as a single line of 6 whitespace-separated integers. We will henceforth denote these integers, in the order that they are given, as D1 H1 M1 D2 H2 M2. These numbers specify when person A was in contact with person B. D1 H1 M1 will represent the time (date, hour, minute) they began contact and D2 H2 M2 will represent when (date, hour, minute) the contact ended.

Output

After checking the numbers given as input, we will print some output to `System.out`. This printout will always be two items, separated by a single space, followed by a single newline character. The `System.out.println()` method will print whatever String is passed in followed by a newline, so if you use it, you do not need to explicitly add a newline character. If you use `System.out.print()`, you will need to add the newline character yourself. In the case of invalid input (more on this below), the printout should be `-1 -1`. In all other cases, the first item should be the number of minutes that the two people were in contact and the second item should be a word (case-sensitive) representing the level of risk for transmission.

- If the risk level is low, the word is `low`.
- If the risk level is medium, the word is `medium`.
- If the risk level is high, the word is `high`.
- If the risk level is extremely high, the word is `HIGH`.

Implementation

Valid Inputs

If any of the following conditions are not satisfied, then the input is invalid (and therefore the output should be `-1 -1`).

- D1 and D2 are both in the range [1, 31].
- H1 and H2 are both in the range [0, 23].
- M1 and M2 are both in the range [0, 59].

- D2 H2 M2 does not represent an earlier time than D1 H1 M1 (note that if they are the same time, then we are treating it as valid and a contact time of 0). We do not have months/years so we will always treat lower dates as being earlier than higher dates.

Formula to Determine Risk Level

We will determine the level of risk based on the number of minutes that the two people are in contact. Treat 1 10 10 - 1 10 10 as being in contact for 0 minutes and 1 10 10 - 1 10 11 as being in contact for 1 minute (i.e., contact time is subtraction). There are 24 hours in a day and 60 minutes in an hour.

- 0 minutes <= contact time <= 60 minutes: the risk level is low.
- 60 minutes < contact time <= 180 minutes: the risk level is medium.
- 180 minutes < contact time <= 360 minutes: the risk level is high.
- contact time > 360 minutes: the risk level is extremely high.

Sample Test Cases

- Case 1
  - **Input**: `10 10 9 11 6 23` (person A comes into contact with person B on the 10th of the month at 10:09am and terminates contact with person B on the 11th of the month at 6:23am)
  - **Output**: `1214 HIGH` (the two people spent 1214 minutes together meaning that their contact time was > 360 mins, putting them in the group of extremely high risk)
- Case 2
  - **Input**: `10 10 9 10 10 10`
  - **Output**: `1 low`
- Case 3
  - **Input**: `10 10 9 10 11 10`
  - **Output**: `61 medium`
- Case 4
  - **Input**: `10 7 9 10 10 10`
  - **Output**: `181 high`
- Case 5
  - **Input**: `99 9 9 9 10 10`
  - **Output**: `-1 -1` (D1 is out of range)
- Case 6
  - **Input**: `10 7 9 9 10 10`
  - **Output**: `-1 -1` (D2 H2 M2 represents an earlier time than D1 H1 M1)

Getting Started:

If you followed the set up instructions you should have access to the command line/shell and have Java downloaded.

Create the file `CovidTransmission.java` in the directory that you prefer.

After you create your file, you will need to remember where it is so you can access it. Here are some tips for finding files via command line:

- "directory" essentially means "folder"
- the `ls` command lists all files/directories in the current directory
- the `cd FolderName` command goes into the directory with the path "FolderName"
- the `cd ..` command goes to the parent directory (the parent directory is always the directory called `..`)

Command Line example

Mia had too much fun this summer and doesn't know where her school work is on her computer anymore.

She first types `cd ~` (and presses enter/return) to go to her home directory. She then types `ls` to see the following directories: `Desktop Work Memes Trash`

Thinking her school work is in Work she does `cd Work` but then types `ls` to see the directories: `FunInternship Research`

So she goes back a directory by doing `cd ..`

Then she goes into her desktop `cd Desktop` and sees all the directories by typing `ls` and seeing: `School Resumes HopesAndDreams`

She goes into the school folder by typing `cd School`

You can use the example as a guide or read the bash resources listed in the Setup section at the top of the assignment. Locate your file `CovidTransmission.java` via bash/command line and follow the instructions below.

Compiling and Running Code

- Open a command prompt window and go to the directory where you saved the Java file (here, called `File.java` containing only the public class `File`).
    - if the file is in a folder on my Desktop called School I would write `cd ~/Desktop/School` in my command line
- Type `javac File.java` and press enter to compile your code. Make sure that there are no errors in the compilation. This will create the class file called `File.class`.
- Now, type `java File` and press enter to run the program.

You need to recompile your file each time you make (and save) a change because the class file is only updated when compiling.

Testing

Try the sample inputs described above. Do you get the same results as their corresponding outputs? Now try some of your own inputs, do you get the results you would expect? If you do, submit to the autograder on Gradescope to see how many tests you pass. If you're failing, is your output in the correct format? Is your code doing what is specified?

You can submit as many times as you want on Gradescope, and you will see some autograder output after each submission.

Note: Only for this PA, you are able to see all the test cases and results when you submit your assignment to Gradescope. It is your responsibility to test your program comprehensively.

Submission:

Turning in your code

Submit all of the following files to Gradescope by **Tuesday, July 5 @ 11:59PM PST**

- CovidTransmission.java

When submitting, please wait until the autograder finishes running and read the output. **Your code must compile in the autograder in order to receive proper partial credit. Make sure that you have followed the General Notes to ensure that the autograder can run.**