"Motivated" Practice Exercises [Part 4]

Start Assignment

Due Oct 17 by 11:59pm **Points** 42 **Submitting** a website url

Overview

This assignment is intended to give you further practice with writing production and test code in the Java language. You will also receive practice in constructing basic UML diagrams and breaking your programs up into classes.

To help with the time management of this assignment, I suggest you try and complete at least 1 problem per day.

Directions

You **shall** create solutions for Exercises 41 though 46, inclusive, as listed on the Programming Exercises page.

You **shall** create one GitHub repository for *all* exercises and ensure that your solution code has been pushed to this repository prior to the submission of this assignment.

- Your GitHub repository **shall** follow the naming convention of `<lastname>-a04`

You **shall** create one baseline solution for each exercise which incorporates *all* associated constraints

- Each baseline solution **shall** be created as an IntelliJ project (using Gradle with JDK 16)
- Each baseline project **shall** be named `exercise<nn>`
- Each baseline project **shall** contain a class named `Solution<nn>` that serves as the application entry point.
  - This class **shall** be contained within a package called `baseline`
  - As an example, for exercise 42 I would create a GitHub repo named `hollander-a04`, a project named `exercise42` and an application class `baseline.Solution42`

You **may** create one additional solution for each challenge component.

- Each challenge solution **shall** be in the same repository as the associated exercise, but within its own project.
- Each challenge solution **shall** be created as an IntelliJ project (using Gradle with JDK 16)
- Each challenge project **shall** be named `exercise<nn>-challenge<uu>`, where `uu` is replaced by the challenge number with a leading 0.
- Each challenge project **shall** contain a class named `Challenge<nn>` that serves as the application entry point.
  - This class **shall** be contained within a package called `challenges`
  - As an example, for exercise 41, challenge 1, within the GitHub repo named `hollander-a04`, I would create a project named `exercise41-challenge01` and an application class called `challenges.Challenge01`

You **shall** push at least 6 commits to your GitHub repository with at least 1 commit pushed per day over a 3 day period.

- You **shall** make at least two commits followed by a corresponding push for *each* exercise
  - Your first commit **shall** be of your program skeleton with *only* pseudocode (only class, field, and method definitions, no logic)
  - Subsequent commits **shall** contain changes to your pseudocode, tests, and the inclusion of programming logic
- You **shall** push each completed exercise as its own separate commit.
  - *Example*: As soon as you complete exercise 41 (the code is written, your unit tests pass, and you have manually run your program to verify its behavior), you commit your solution to your local git repository, then you push your changes up to GitHub. Only after your code appears on GitHub do you start the next exercise.
- You **shall** push your files to GitHub using an appropriate git client.
  - The "Add file" button on the GitHub website is **not** an appropriate client.
  - Attempting to manually add files to a repository will most likely result in de-synchronizing your local and remote repos and cause you unnecessary headache.
- This requirement can only be satisfied if you spend at least 3 days working on this assignment.

You **shall** create a **.gitignore** file to ensure that your **build** folder and your **.gradle** folder are not stored within git.

You **shall** include the following comment at the top of *each* of your *.java files (with the appropriate name substituted in place):

```
/*
 *  UCF COP3330 Fall 2021 Assignment 4 Solutions
 *  Copyright 2021 first_name last_name
 */
```

You **shall** write your solution in both pseudocode *and* java

- Your pseudocode **shall** be provided within each corresponding .java file
- Your pseudocode **shall** be interspersed with your production code such that it is clear which statements correspond to which comments
  - Be aware that since your initial commit of each solution must contain only a program skeleton, your pseudocode for that commit will be blocked within each method that you have defined. As your make additional commits, your pseudocode will start to spread out between the program logic.

You **shall** follow good programming practices when solving these exercises.

- You **shall** use "clean" identifiers (refer to **Module 04 - The Anatomy of a Class**)
- You **shall** decompose your solution into multiple "clean" methods (refer to **Module 04 - The Anatomy of a Class**)
  - Points will be deducted if your entire solution exists within a `main` method
- You **shall** follow one of the following Java Style Guides with only minor deviations when it makes sense:
  - **https://google.github.io/styleguide/javaguide.html (https://google.github.io/styleguide/javaguide.html)**
  - **https://www.oracle.com/java/technologies/javase/codeconventions-contents.html (https://www.oracle.com/java/technologies/javase/codeconventions-contents.html)**
- You **shall** install the *SonarLint* plugin for IntelliJ and modify your code to remove any major warnings or style issues
  - SonarLint rules **shall** override your chosen style guide

    - Some exceptions are allowable: e.g. you do not need to replace print statements with a logger; you may have classes with numbers in their identifiers

You **shall** create Class Diagrams for each exercise using PlantUML. These diagrams must be pushed to your Git repository along with your java code.

- Each class within your diagram must list both the attributes and the methods defined within that class.
- You **shall** store these diagrams within a `docs` folder. You must create this folder in your project root directory (at the same level as .idea and src).

You **shall** use JUnit 5 to incorporate automated unit testing into your solutions.

- You are expected to create a unit test for each method within your solution that either modifies the state of an object or returns a value
- You do not need to create unit tests for methods that *only* print, *only* scan input, or *only* print a prompt and then scan input
- I **strongly** suggest you write your test code before you write your production code, as this will enable you to clearly specify the behavior of your methods

Because this particular assignment makes heavy use of files, the following additional requirements must also be satisfied:

- Input and output files used by an exercise **shall** be stored in a folder called `data`. This folder must be created in your project root directory.

Submission

You will submit a link to your GitHub page (e.g. https://github.com/drhollander/hollander-a04). Your repositories should be private prior to the assignment due date, but **must** be public within 24 hours of the assignment due date.

If we cannot access your repositories, or if you provide an invalid link, you will receive a 0 - *please double check your submission once it has been made*.

Late submissions will receive a 0 as per the syllabus.

Grading Criteria

You will receive 7 points per **working** solution that is present in your GitHub repository (1 point for your UML diagram, 1 point for having pseudocode, 3 points for well written production code that matches your UML diagram, 2 point for comprehensive test code).

- If your exercise code fails to compile or generates an incorrect answer (minor formatting differences aside), you will not receive credit for that solution.

- The graders will deduct 1 point for **each requirement** (shall statement) that you fail to satisfy on **each problem**, up to a maximum of 50% of the total score.