ICL 1

Due No Due Date **Points** None **Available** after Sep 22 at 6pm

** This assumes you've gained access to the class repo in Github. **

Background

The in-class work for this week focuses on building big software systems. The concept of divide and conquer is essential and is accomplished using modularity. In addition, object-oriented class design allows us to achieve modularity because the module encapsulates both data and functionality (procedures).

During this in-class work, we will focus on figuring out how to (a) define module boundaries, and (b) characterize what a module does. This needs special consideration as a system gets bigger.

First Scenario (v1)

Submitted to Canvas 9/23/21. Per Slack, v2 and v3 are not to be uploaded to Slack but completed on our own.

Start with a traditional Hello World program.

Version 1: Write the traditional Hello World program.

Submission:

1. Determine your "su_username". It'll likely be the first letter of your first name, followed by your last name. For example, mine is "sriley".
2. Make a new Java project (using IntelliJ) under https://github.com/stephen-riley/cpsc-5011-fall-2021-projects/tree/main/week_01/ICL (https://github.com/stephen-riley/cpsc-5011-fall-2021-projects/tree/main/week_01/ICL) in your fork. Call it `<su_username>_icl_01`.
3. Create a class called `Driver.java` and write the code to print "Hello, world!".
4. Under the `[su_username]_ice_01` package, create a class called `Driver.java` and write the code for the **First Scenario (v1)**.
5. Push your changes to your repo. Verify you can see them on Github.

Second Scenario (v2)

Due to popular demand, we want to update the program to a general-purpose "greeting" system that will display a customized greeting to each user under a variety of contexts.

The current implementation both "computes" the greeting and prints it. This violates the Single Responsibility Principle (SRP), where every object should have a single, encapsulated responsibility; thus, there can be only one reason to modify a class.

Version 2: Break the program into a greeting generator (that returns a string) and the main program that prints it. This implementation will require at least two classes.

Third Scenario (v3)

We will now customize the greeting. The first question is, "How do we handle customizations"? One option is to create a dictionary that will give us some flexibility as to where runtime information will come from. For now, let the driver be responsible for the dictionary; therefore, the driver is responsible to the greeter generator for all customization information.

How would we handle more than one argument input? Let's think of the idea of a "template", something like `"Hello $name - that's a nice $color shirt."` where template variables (preceded by the $ symbol) are translated to corresponding values. This is called variable interpolation in other languages (i.e. String.format does something like this, but the list of variables is pre-compiled).

Version 3: Update your driver to include a dictionary. Pass that dictionary to the greeter generator.

Create a template that will translate the template variables to the corresponding values, stored in the dictionary. Be sure to accommodate translation of more than one template variable in your greeting.

Fourth Scenario (v4)

Your customer would like to have the greeting customized with the time of day. For example, instead of saying "Hello Taylor" it might say "Good afternoon Taylor"

Assume a "daypart" is defined as follows:

- MORNING is after 5AM but at or before 12 noon
- AFTERNOON is after noon but at or before 5PM
- EVENING is after 5PM but at or before 10PM
- NIGHT is after 10PM but at or before 5AM

Version 4: Extend the template language to include a variable `$daypart`, and have the system compute the appropriate string to describe that time of day. Base the time of day on the system clock. You can use LocalTime() for this.

Extension: the module that computes the daypart should can also be given a string suitable for LocalTime.parse(String) and computes the day part using that time. You need not modify the driver or template language, just give the daypart module that extra functionality.

Submitting it all

When you're all done:

1. Make sure to push it all to your fork.
2. Verify you can see your current code on Github.
3. Issue a Pull Request (PR) against the class repo.
4. When I see the PR, I'll accept it.
5. At that point, you should see your code in the **main repo (https://github.com/stephen-riley/cpsc-5011-fall-2021-projects)**.