Assignment 2: Pair Programming

Assignment Handout

Due date: Sunday, March 24th by **11:59pm**
This assignment is done **in pairs.** Note that you should have created your pairing before starting the assignment.
If you haven't done so by the deadline in our announcement, you will lose 5%.
Please email us **ASAP** if you still don't have a pairing.

Overview

In this assignment, you will focus on some core elements of the **process** of producing software. By completing this assignment, you will work on:

1. Pair programming
2. Testing
3. Clean Coding

Starter Files/GitHub Repo

You will be submitting your assignment entirely on GitHub.
Private GitHub repos for your pair (**one repo per pair**) will be created in our course GitHub Org, and you will be responsible for ensuring that your repo is up to date (with your submission on the **master** branch) at the time the assignment is due. You will be responsible for any delays introduced if you did not submit a request for pairing or resolved any issues with your pair.

The GitHub repo you get will look like this:
https://github.com/csc301-winter-2019/a2-starter

Pair Programming

As we talked about in lecture, part of the prescriptions of some of the software processes we looked at included Pair Programming.
In this assignment, you will perform **a prescribed version** of pair programming for **two features** of the Java program. You only have to pair program as prescribed for these two features - for the rest of the features it is up to you how you want to finish them with your partner. Here's how we'd like you to do pair programming:

- Decide roles (driver/navigator) and the feature to be developed (they should be big enough to spend 2 to 3 hours thinking about implementation and then coding) . Together break down the feature to multiple "checkpoints" and design the solution on paper. Estimate how long it will take you to finish this feature with the proposed design. This design will be the guide in coding your solution. Once you are happy with your design, the driver (using their own GitHub account) starts coding the solution while the navigator watches and helps whenever needed (no checking phones or going to get coffee). After the allocated time is done (could be 30-90 minutes, you can take a break and switch roles). Repeat this enough times until the feature is done. Experiment with different variations to see what works best for you. You can then move on to the second feature and **switch starting roles.**
    - This includes the **tests** for the features you chose - make sure they are pair programmed as well.
- In your **`README.md`**
    - Should explain which features were pair programmed
    - Should explain who the driver and navigator was for different parts of the features
    - Should give a **reflection** on how it went, and what you liked and disliked about this process
- The commits should reflect who the driver was for each feature
    - **Any** work done on a particular feature should **only** be done by the driver for this feature - we will check!

Specification: Pizza Parlour

For this assignment, you will be creating an API for a virtual pizza parlour, based on some **client requirements.**
Unlike A1, there is not much starter code to start with - you will have to take the specifications of the app, and use the concepts from class to write a **Java program** to make it work.
You will also write a **command-line interface** to demonstrate the functionality of your app

Going off of our theme that Software Development is not just coding, you will notice that the API itself is not that difficult. We know that you can probably hack something together quickly that more or less works most of the time.
However, that is not what professionals do. You will have to think about and reflect on why you chose certain elements, including:

- How to represent objects
- The relationships between objects (coupling, cohesion)
- Design of functions
- Using **design patterns effectively**
    - You can use the ones we mentioned in class, or others
- Clean coding practises

The Starter Code

The stater code is meant to be set up as a **Maven** project in your IDE, using the `pom.xml` file. For example, in Eclipse, you can select `Import -> Maven -> Existing Maven Project`, and select your repo folder. This should work in a similar way for other IDE's.

There is only one java file in the starter code, `PizzaParlour.java`, which will have the main method that will start the program. Right now, there is a simple start to the command line interface for a pizza customer to start ordering. You will have to add to it to fulfill the requirements of the parlour below (you can replace the contents of the main method entirely as well). We will run your code from this main method, so do not get rid of it.
The rest of the design is up to you!
You should make effective use of **design patterns** that we talked about in class, or others - you will be graded on program design!
Continue reading for more information on what you have to do.

Client requirements

Our customers (pizza parlour patrons) need to be able to do the following **while the app is running** (we do not require external persistence in files apart from what's listed below)

1. Submit a new order
    - With the following possible components (any number of them):
        - Pizza
            - Size
            - Type
            - Toppings
        - Drinks
    - Order number
    - Pizza type can be (pepperoni, margherita, vegetarian, Neapolitan) and each one of them will have a specific method of preparation (you can make your own assumptions as to what goes on each type of pizza). Your program needs to support adding new types of pizza as well.
    - Toppings can include (olives, tomatoes, mushrooms, jalapenos, chicken, beef, pepperoni)
    - Drinks can be (Coke, Diet Coke, Coke Zero, Pepsi, Diet Pepsi, Dr. Pepper, Water, Juice)
    - Please note that the total price will be calculated based on the order
        - Prices can be dynamically changed for any items
            - You can store the prices for items in a file (persistence)
2. Update existing order:
    - Changing specific elements of a Pizza order, drinks, etc.
3. Cancel order
4. Ask for pickup or delivery
    - Pickup from store
    - Three ways to send delivery to a specific address
        - Pizza parlour's in-house delivery person
        - Uber Eats: Accepts delivery details using the following information in **JSON**:
            Address:
            Order Details:
            Order Number:
        - Foodora: Accepts delivery details using the following information in **csv**
            Address:
            Order Details:
            Order Number:
5. Ask for the menu using one of the following:
    - The full menu
    - Specific item where the user passes in the item name and the price is returned.

Testing

You will design tests (JUnit) to test your code functionality.
It is up to you how you test your code, but in order to receive full marks, you should test with **at least 80% line coverage** of the repo with **meaningful tests and test names.** You can learn more about code coverage here.

Code Craftsmanship

You must have good programming and formatting style in your code. Review our lectures on this to ensure you are creating high-quality, clean code.
You are free to use use tools like Linters, IDE tools, etc. to help you. Mention in your README.md which tools you used to help you create clean code.

Handing in your work

We will grade the most recent version (and look at the commit history) of your submission on the **master branch** of your pair's repo.
Make sure everything is updated there, including your README.md

Grading

You will be graded on the following criteria.

- **Pair Programming** (15 marks):
  - Your process explained in README.md
  - Your commit history reflecting your process explanation
  - Your reflection about your process, including positives and negatives
- **Program design** (20 marks)
  - The design patterns you chose to use
    - The description of why you chose to use them for your implementation
  - Relationships between objects: code cohesion, coupling
  - Function design
- **Functionality** (25 marks): We will test it and look at your code to see your implementation of the required features.
- **Tests** (20 marks)
  - Thorough testing of the required features
  - At least 80% line coverage of the repo with meaningful tests and test names.
- **Code Craftsmanship** (10 marks)
  - Programming and formatting style is good
  - Let us know what tools you used (Linters, IDE tools, etc.)
- **README.md explanation and organization** (10 marks): Your README should be well formatted and organized. The TA should be able to use your app easily based on the instructions you provide.