Sprint Week One - Fall2022

*Sprint One*

Welcome to Sprint Week One everyone! In this sprint you will be tasked with creating a Java program. The program will be a command line client (a simple text-based console application) which will allow a user to run the commands as outlined below.

* The project must be pushed to GitHub. Using a PR workflow is required for working with your team. Your team must create a repo (this is the repo that will be submitted at the sprint end) and follow the branch, PR workflow as taught in class. Following this will be part of the overall grading of the sprint.
* When designing your unit tests for the app be sure to use mocking as we covered in class.
* The data can be hardcoded or better yet read from a file, bonus points if you implement the ability to add new records from the command line app.
* You should also create and run a setup of documented test cases to cover all the use cases you would expect a user to encounter with your app.

See below for a project default in the case that you don’t want to/can’t come up with any of your own ideas to pitch. With that we wish you all good luck on the sprint!

# Default Project

A local Country Club, The Golf Club, is tired of their hodgepodge system of files for tracking data on memberships and tournament participation. This system of sharing local spreadsheets and text docs is giving the staff headaches which is throwing their own games off! The owner of The Golf Club reaches out to your team to help them create a solution to the problem and save them from a life of hand cramps and headaches.

The owner of the establishment, Ms. Lorelai McIlroy (you can call her Rory), **first** asks that you create a service that allows the employees to retrieve the following information:

* For Members:
  + Member Name
  + Member Address
  + Member Email Address
  + Member Phone Number
  + Start Date of membership
  + Duration of membership
  + Membership type (normal/trial/special offer/family plan/other) (If family plan – indicate which other members in the database, if any, are connected on their plan. If other – you should include the monthly membership cost.)
  + Current tournaments they are actively participating in.
  + Past tournaments they have previously participated in.
  + Upcoming tournaments they are registered to participate in.
* For Tournaments:
  + Start date.
  + End date.
  + Location
  + Entry Fee
  + Cash Prize Amount
  + Participating Members
  + Final Standings (If the tournament is over already.)

This application can have a simple text-based console interface (like what you guys use to write back in Semester 1 in those early IPO problems).

You do not want the headache of having to repeatedly go back to fix bugs for The Golf Club and they certainly do not need it either, so your code should be tested thoroughly before shipping. In fact, Rory has requested a full suite of unit tests to give as high a level of coverage for your application as is reasonably possible to ensure its relative robustness, and to allow for refactoring down the line if necessary for future contractors.

**Submission Requirements:**

Each team member must submit a document containing the URL to the teams GitHub and a summary of how the sprint went from your prospective. Please include things like your role in the team, how each team member participated from your perspective and any particular areas of challenge you and your team had over the course of the week. If you reached outside of the team for help, what was that help, who was it from and what topics did you need to get help with.

Each team should also include a short link to a demo video of your sprint application working. An easy way to do this is for all team members to attend a Teams meeting and record it. Add this link to the submission document as well and don’t forget to give permission for your instructors to view it. In the video you should be sure to cover the app functioning, a quick review of the test cases, a quick code walk through and show the PR trail your team used in GitHub.

**Grade Meaning**

1 Incomplete. Student shows severe lack of understanding of the material – solution is heavily incomplete, non-functional, or completely off base of what the assignment was asking for.

2 Partially Complete. Students show some understanding of the material. Solution may be non-functional or partially functional, but the approach is correct, albeit with some major bugs or missing features.

3 Mostly Complete. Student demonstrates understanding of the major ideas of the assignment. Solution is mostly working, albeit with a few small bugs or significant edge cases which were not considered. Shows a good understanding of the correct approach, and is either nearly a feature-complete solution, or is a feature-complete solution with some bugs.

4 Complete (Equivalent to: Pass.) Student shows complete understanding of assigned work and implemented all necessary features. Any bugs that are present are insignificant (for example aesthetic bugs when testing the functionality of code) and do not impact the core functionality in a significant way. All necessary objectives for the assignment are completed, and the student has delivered something roughly equivalent to the canonical solution in terms of features and approach.

5 Complete with Distinction (Equivalent to: Pass Outstanding) The student demonstrates a clear mastery of the subject matter tested by the QAP. The solution goes above and beyond in some way, makes improvements on the canonical solution, or otherwise demonstrates the student’s mastery of the subject matter in some way. A solution in this category would consider all reasonable edge cases and implement more than the necessary functionality required by the assignment.