CPSC 2151

Lab 3

Due: At the end of the lab period

In this lab you will be developing functional and non-functional requirements for a system that already exists and you are familiar with: iROAR. You will also practice writing some contracts for some example classes. You are required to work in groups of 3 or 4 students, and only need to submit one file per group. Make sure to include everyone's name on the assignment.

Part A

Instructions:

Work with your group to think of as many requirements as you can for Clemson's iROAR portal. Organize these requirements into a Word document as best as you can, trying to keep similar requirements together. Remember to include both Functional and Non-Functional requirements in your list. Refer to the slides for the video about Requirements for more help with identifying requirements (available in the Lab Canvas page under Modules -> Module 5: Requirements and Project Reports).

Functional requirements must be written as a user story. User stories include three parts: The user role, the action, and the benefit. Remember that user stories are simple sentences, as each action should be its own user story. Do not combine related user stories into one complicated requirement. Non-functional requirements are not written as user stories, but should still be fairly simple sentences (i.e. the system must be written in Java). You should be able to think of quite a few user stories and non-functional requirements.

TIPS and additional Requirements

- It will probably be easier to think of requirements from the perspective of the role of a student using iROAR, since that is your role. What other roles exist in iROAR? What requirements would you imagine they would have? Your user stories must include multiple roles.
- Look through the categories in the FURPS+ model to help identify non-functional requirements.
- Make sure your documents is well organized to make it easier for the grader to check to see that you meet the requirements in the rubric.
- You must write at least 30 user stories and 5 non-functional requirements.
- If you are having trouble coming up with user stories, see if any of your current user stories are too complicated and can be broken up into smaller simpler stories.

Part B

For this part of the lab, you will practice writing contracts for classes. You should write these contracts in the same document from Part A (or in a separate document initially and combine it with Part A before submitting) and make sure each part is well organized.

`Clock.java`

This class is designed to simulate a 24-hour clock. It allows a clock and an alarm to be set. It also has features to move the clock time from the current time instead of just setting. Alarm time should be always after the current clock time. It has a few additional features as well to say whether it is AM or PM, for example.

The following constructors and methods must be publicly available. Note that they must have the exact names, parameters and return types listed. You must include all contracts including any **invariants**, **preconditions** and **postconditions**.

- Constructor: `public Clock(int hour, int minutes)`: sets the alarm to a minute past the current time hour.
- `public int advanceHour(int by)`: increases the hour by the specified hour by. If the total exceeds 24, it is taken to be a roll over. Returns the new hour. If needed, advances the alarm to a minute past, to maintain invariant.
- `public int advanceMinutes(int by)`: increases the minutes by the specified minutes by. If the total exceeds 60, it is taken to be a roll over. Returns the new minutes. If needed, advances the alarm to a minute past, to maintain invariant.
- `public int minutesFromMidnight()`: returns the number of minutes from midnight.
- `public boolean isAM()`: returns true iff clock time is AM.
- `public void setAlarm(int hour, int minutes)`: sets the alarm to be hour and minutes from the current time.
- `public int minutesForAlarm()`: returns the number of minutes from the current time for the alarm.

`RunnerPlanner.java`

This class is designed to simulate a runner's plan. It allows a runner to set a goal and it tracks how far they are from achieving that goal. The current distance must be always below the goal.

The following constructors and methods must be publicly available. Note that they must have the exact names, parameters and return types listed. You must include all contracts including any **invariants**, **preconditions** and **postconditions**.

- Constructor: `public RunnerPlanner(double distance)`: sets the goal distance for the planned run. Initial runner distance, speed, and minutes run are set to zero.
- `public double currentLegSpeedandDuration(double speed, double duration)`: returns the distance that would be traveled in the next leg.
- `public double adjustGoalDistance(double by)`: returns the new goal distance. Adjusts goal distance by the amount by.
- `public double currentLegSpeed()`: returns speed.
- `public double currentLegDuration()`: returns duration.
- `public double goalDistance()`: returns current goal distance.
- `public double distanceToGoal()`: returns how much more distance remains to be run.

Contracts and Formatting

You need to create contracts for each class (invariants) and every method in the class (preconditions and postconditions). Contracts should be formally stated when possible. We have not covered much formal notation, but if a parameter named row needs to be greater than 0, then the precondition should say "x > 0" not "[x must be greater than 0]."

Everything that has been mentioned as a "best practice" in our lectures or in a video should be followed, or you risk losing points on this portion of the lab assignment.

Submission Document

Please make sure contracts for both classes are well labeled and formatted. Include the name of all group members in the document

Tips and other requirements

- Carefully read through the description before you start writing contracts. You don't want to miss any relevant information.
- Remember our "best practices" that we've discussed in class. Use good variables, avoid magic numbers, etc.

Groups

You are required to work in a group for this assignment. Groups must contain either 3 or 4 students. Our goal for lab is for everyone to have a chance to practice the concepts we are teaching in class (or from the videos you have been watching), so remember to work collaboratively and do not try divide-and-conquer. Only one person should be typing them into the document. If the TA sees that you are working separately and then copying and pasting into one document, you will not earn credit for this lab. Your group must work the entire time to receive full credit. If you divide-and-conquer, rush through and finish early, you will not get full credit.

Submitting your file

You will submit your file using Canvas in the lab section you are enrolled in. Make sure to convert your file to a PDF to ensure it can be opened on the grader's machine. The submission will only accept PDFs. Only one member per group needs to submit the assignment, but they should include all group members names on the submission.