Phase 0 Progress Report - Group 043

Specification Summary

We are creating a scheduling app for University of Toronto students where they can create an account, upload their weekly schedule, and add friends. The main feature of the app is to display the user's timetable, and after filling in your weekly classes and commitments, you can connect with friends, select a friend in order to determine when you both have free time, or have events such as a class in common.

CRC Model Summary

Our CRC model is subdivided into the layers of clean architecture.

The primary layer consists of our entity classes: Person, Event and Schedule. The Person class stores a name, password, schedule, and a list of friends who are also instances of the Person class. The Schedule class stores Events corresponding to times of the day and day of the week. It carries the methods getEvent and toString to output information from the schedule. The Event class stores an event name, start time and end time and is used in the Schedule class.

The next layer consists of our use case classes: ScheduleEditor and ScheduleComparer. ScheduleEditor edits a person's schedule by adding, editing or removing events. ScheduleComparer consists of the compareSchedule class which can access a person's friend's schedule to compare them with the user's.

The next layer consists of our controller class ScheduleManager. This class stores all the users and their corresponding schedules for ease of access. It is used to call on ScheduleEditor and scheduleComparer to carry out their methods.

Finally our interface, called CalendarPage, currently displays a person's schedule on the screen and is responsible for receiving an input for who the user would like to share their schedule with.

Scenario Walkthrough Summary

Our scenario walkthrough considers the situation where a user shares their schedule with another user. Using it, we were able to clean up our CRC model somewhat, noticing places where a responsibility was represented (or, as happened with CalendarPage, where a responsibility should actually have been moved elsewhere), and double check that our idea for the shape of the program could achieve its primary task of sharing a schedule between two users and helping them check for times they were both free. Following the sequence of responsibilities that would allow the program to complete this task, we traced through the CalendarPage, ScheduleManager, ScheduleComparer, and Person and Schedule classes.

Skeleton Program Summary

In the Main class of our skeleton program, we give the user options to view their schedule, add and remove events from their schedule, and compare their schedule with another user. Viewing the schedule will output the contents of their schedule as a string. Choosing to add or remove an event will prompt the user to enter details of their event, before adding it to their schedule, or removing it from their schedule. If the user chooses to compare schedules, the program outputs ScheduleComparer's return as a string. Entering 'quit' will end the process.

ScheduleManager represents and manages the entire system of users and their schedules. It stores a private HashMap with Person keys and Schedule values, which is initialized empty. As a controller class ScheduleManager calls on the use case classes ScheduleComparer and ScheduleEditor to compare and edit the user's schedule respectively, when instructed to do so by the user.

ScheduleComparer's compare method is called from ScheduleManager. This method takes in two users to compare their schedules and returns a schedule to show times where two given users are both free and when at least one of them is busy. It does so by implementing a compare method that takes in two given Person users (user1 and user2), creating a new schedule, getting the schedules of both users and comparing each hour of each day of the week between the two schedules of the users.

ScheduleEditor's addEvent and removeEvent methods are called by ScheduleManager. The addEvent method takes in a user-given event and user as parameters and accesses the information from the event to add the name, day, start time and end time of the event to the schedule. As for removeEvent, it takes in the event's name as a String, the event day as a String and the user as a Person to look for and set the event in the user's schedule back to null.

The entities of the program as mentioned are Schedule, Person and Event. Schedule stores HashMap of the day as a string (key) and an inner HashMap (value) of start hour keys to string values of the event name. Schedule also includes a toString method that presents the mentioned information as a string. The Person class holds the username, password, schedule and list of friends of a user. The Event class stores the string name, string day, integer start time and integer end time of an event.

Open Questions:

- How to format our user interface class to make it most presentable and user friendly — What does a user-friendly design look like and how might we implement that?
- How to face privacy questions (e.g. who can view a user's schedule) — What is the process for considering ethical issues in relation to a program, typically? Is there a particular way of doing this that people have found particularly helps facilitate discussion and effectively consider the ethical pros and cons of one choice over another?
- How to more effectively communicate and meet with group members in order to divide tasks more efficiently — Other than the advice of 'communicate more' or 'speak to an instructor', how do we manage arranging meetings between six people?

- How do we fix style warnings in IntelliJ that seem to force us to completely rewrite our code? IntelliJ sometimes gives style warnings with suggested fixes that break other parts of our code, but we don't know how to fix those problems because the style warnings aren't very specific / have missed the actual problem. Is there a suggested way for us to work those out?

What Has Worked Well So Far With Your Design

We are confident in how the program responsibilities are split up between classes. We have also found that our primary functionalities work well in the entity classes we set up, and feel that it is a good foundation to the rest of the program and to any changes we might make.

Brief Summary of Each Group Member's Work and Future Plans

We have all met together a number of times to discuss ideas and implementation before we split up work to implement on our own time. Rachel and Emma completed the specification, while Emma, Sunehra, and Priyanka completed the CRC model, though we all met to provide input on the final result. Rachel completed the scenario walk through while we all split up the skeleton program and collaborated on it together. Finally, Sunehra and Rachel worked on the progress report. However, for most parts of our project we did it in real time and collaborated while one person edited documents and code.

We are planning to work next on our login system, add an editEvent option that would allow users to edit aspects of event like its name, write unit tests, double check our methods to see if we are following our CRC model as desired (and if not, why), and make them more specific according to our program. We intend to determine how exactly we will divide up the tasks when we regroup after this phase of the project.