Charles Prior, Louise Li,

Kenan Colak, Qianyi Xue

03.26.24

Evolution 1 Design Decisions

# Overall Design

Overall, we broke our project up using the Model, View, Controller (MVC) design pattern. This was chosen to allow us to create a text-based UI while remaining flexible for any future UI upgrades requested by the client.

# Model

## Storage of State

For the storage of information and data, we began by creating classes to represent the professor, students, courses, and lectures, and record the attendance of students. The main implementation for these classes came through how they inherited one another with the choice being to have the Professor class act as the “base” class and have an n-amount of courses. As such, we designed Professor to contain an ArrayList of Course called courses so that the Professor could access each of the different classes they taught. As each of the professors, courses, lectures, and students need identifiers, they were all given a form of identification, commonly using IDs. The Professor class contained variables for the name, ID, email and courses taught by the Professor with methods corresponding to getting and setting these variables as well as adding and removing courses from the Professor’s catalog.

Since the professor will need an n-amount of courses to manage, we created a Course class next. In this class, we had the identifier variables such as the courseID, courseName, the professor who taught it, and the times the course took place. Most importantly, the class had two ArrayList variables that contain the students within the course as well as a list of lectures for the Course class. The normal getter-setter methods were implemented as before for the Course class. As the Course has an n-amount of students and lectures, we had to implement those classes next. The Lecture class contains the variables for identification as before, including the lectureID, courseName, professor who teaches it, the date of the lecture, another list of students for that lecture for attendance purposes. Lecture class also includes an attendanceSession variable of class AttendanceSession for the purpose of storing information regarding the attendance of students, discussed further in the *Manipulation of State* section. The Student class simply had the normal getter-setter methods as well as the identifier variables of legalName, studentID, email and displayName to be able to identify and distinguish each of the students.

As all these classes contain each other within one another, this allows for us to have very well made has-a relationships within our classes as a professor has-a course which has-a lecture which has-a student and so on. Through these classes, we were able to properly implement this portion of the assignment and store the proper variables within the proper locations.

## Manipulation of State

To be able to store the Attendance of each student, we divided our attendance classes into two separate ones named AttendanceSession and AttendanceRecord. AttendanceSession was made to record and store the attendance in a specific lecture that is going on at that moment whereas AttendanceRecord keeps track of all previous attendances through the dates, the student associated with the attendance, their status and what lecture that student’s attendance is being checked for. One is for modifying and the other is for storage.

To implement the requirement to email students upon change of attendance we used the Observer design pattern. The application (App; the Publisher in the design pattern) has an EventManager to keep track of the listeners to its events. Listeners implement the EventListener interface. In this evolution, we only have one type of Listener, the EmailAlertsListener, which takes a student and attendance status and notifies via the students email that their attendance status has changed.

# View

For the UI and viewing portion of the assignment, a class called TextUserView was created that managed all the printing necessities to output the proper display outputs to the user. This was done so by creating methods to print the courses, students, lectures for starters and then having methods for a start screen, a header as well as the status of students to see their attendance status for the current class. By putting all of those methods together, a proper UI was formed to be called within the App.java class for the fully put-together code functionality.

# Controller

For the controller portion of our MVC, we created a class to handle the entirety of this portion on its own called TextUserController. This class took care of such methods as removing students from a course, showing attendance from a course, changing students’ display names, updating students’ attendance records and many more. These methods were used to build the overall program’s ability to manipulate the state of the students, classroom, lectures, course and professor through its joint UI managed through the aptly named TextUserView class. By having all of our methods for the controller in one class, we were able to streamline the modification of the different classes by the user and simply call the methods necessary when the manipulation was demanded.