Group Homework 8 : Student Management System

(Deadline as per Canvas)

For HW8, you must find a group and sign up on Ed Discussion. **All details related to partners, rules for partnering, and partner sign up will be on Ed Discussion. Please read all the most up to date information on our post before proceeding with this homework.**

This is a required part of HW8, individual submissions will not be allowed.

We'll learn more about Version Control & Git later in the course, but if you want to use GitHub (or some similar cloud-based code management system) to set up a remote code repository for code collaboration, you can look ahead to "Version Control & Git" in Module 14. We will be setting up each group with a shared space in Codio you can use instead for version control.

If you use GitHub (or some similar cloud-based code management system) to set up a remote code repository for code collaboration, YOU ARE REQUIRED TO KEEP THAT REPOSITORY PRIVATE AND ONLY SHARE WITH YOUR OTHER GROUP/TEAM MEMBER. DO NOT CREATE PUBLICLY ACCESSIBLE REPOSITORIES TO HOST ANY ASSIGNMENTS IN THIS COURSE.

If you are caught posting code to a publicly accessible location, it will be considered cheating. We will notify The Office of Student Conduct (OSC) and you will fail the course.

Like the previous assignment, this homework is also very detailed, so please start as early as you can on it.

It deals with the following topics:

- Object-Oriented Programming Design
- Inheritance
- File I/O
- Regex

Introduction

In this homework, you will implement a console-based student management

system. The objective of this assignment is to design a system for students to manage their courses. There will be three main user roles in the application: Admin, Student, and Professor.

In the student management system, a) A student can log in to their account, view/add/drop courses, check their course schedule, and view grades. b) A professor can view course information they have, and view the student lists for these courses. c) An admin can view course/student/professor lists, and add/delete courses/students/professors. The course information will be in the *courseInfo.txt* file. There will also be *three files* containing student/professor/admin information. The student management system will read and parse all of the files. Once all information has been loaded into the system, you'll be able to log in as a(n) student/professor/administrator to test the system.

You are expected to design several classes and to implement methods to build the application. For example, you may create a class **FileInfoReader** to load the files. You can create an abstract class **User** and a **Professor** class, **Student** class, and **Admin** class which all extend and implement the **User** class. You can have a **Course** class that represents a single course and a **Controller** class to control the main logic of the entire system.

Below are explanations (and samples) of the pieces of information in each of the four provided data files.

courseInfo.txt - Courses information file that contains: course ID; course name; lecturer; days; start time; end time; capacity

```
CIT590; Programming Languages and Techniques; Brandon L Krakowsky; MW; 16:30; 18:00; 110
```

studentInfo.txt - Student information file that contains: student ID; student name; student username; password; course ID: course grade (could be multiple)

```
001; StudentName1; testStudent01; password590; CIS191: A, CIS320: A
```

profInfo.txt - Professor information file that contains: prof name; prof ID; prof username; password

```
Clayton Greenberg; 001; Greenberg; password590
```

adminInfo.txt - Admin information file that contains: admin ID; admin name; admin username; password

```
001; admin; admin01; password590
```

Below are examples of what the system could look like. Feel free to make your program do exactly this or make it look even fancier.

When entering the system, one can select to log in as a(n) student/professor/admin, or quit the system.

```
-------------------------
Students Management System
-------------------------
 1 -- Login as a student
 2 -- Login as a professor
 3 -- Login as an admin
 4 -- Quit the system

Please enter your option, eg. '1'.
```

Student Login

When logging in as a student, one can view course information, add/drop courses, view grades, or return to the previous menu.

```
Please enter your username, or type 'q' to quit.
testStudent01
Please enter your password, or type 'q' to quit.
password590
-------------------------------
      Welcome, StudentName1
-------------------------------
 1 -- View all courses
 2 -- Add courses to your list
 3 -- View enrolled courses
 4 -- Drop courses in your list
 5 -- View grades
 6 -- Return to previous menu

Please enter your option, eg. '1'.
```

View all courses: All courses are displayed in the console.

```
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 0, lecturer: Professor Brandon L Krakowsky
CIT591|Introduction to Software Development, 12:00-13:30 on MW, with course capacity: 120, students: 0, lecturer: Professor Arvind Bhusnurmath
CIT592|Mathematical Foundations of Computer Science, 10:00-11:00 on TR, with course capacity: 72, students: 0, lecturer: Professor Clayton Greenberg
CIT593|Introduction to Computer Systems, 11:00-12:00 on MWF, with course capacity: 72, students: 0, lecturer: Professor Thomas Joseph Farmer
CIT594|Data Structures and Sofware Design, 10:30-12:00 on TR, with course capacity: 72, students: 0, lecturer: Professor Eric Noel Fouh Mbindi
CIT595|Computer Systems Programming, 15:00-16:30 on TR, with course capacity: 72, students: 0, lecturer: Professor Insup Lee
CIT596|Algorithms and Computation, 10:30-12:00 on MW, with course capacity: 120, students: 0, lecturer: Professor Arvind Bhusnurmath
CIS105|Computational Data Exploration, 10:00-11:00 on MWF, with course capacity: 60, students: 0, lecturer: Professor Clayton Greenberg
```

After adding course CIT590 to the student's schedule.

```
The courses in your list:
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 1, lecturer: Professor Brandon L Krakowsky
```

If one tries to add a course which has already been added to their schedule, the system should prompt a message towards this. In addition, if one tries to add a course which doesn't exist in the system, the operation will not succeed.

```
Please select the course ID you want to add to your list, eg. 'CIT590'.
Or enter 'q' to return to the previous menu.
CIT590
Course added successfully

Please select the course ID you want to add to your list, eg. 'CIT590'.
Or enter 'q' to return to the previous menu.
CIT590
The course you selected is already in your list
Please select the course ID you want to add to your list, eg. 'CIT590'.
Or enter 'q' to return to the previous menu.
```

One cannot add a course which has a time conflict with another added course.

```
Please select the course ID you want to add to your list, eg. 'CIT590'.
Or enter 'q' to return to the previous menu.
CIS620
The course you selected has time conflict with CIT590 Programming Languages and Techniques.
Please select the course ID you want to add to your list, eg. 'CIT590'.
Or enter 'q' to return to the previous menu.
```

Drop a course: If one tries to drop a course which is not on his/her schedule, the operation will not succeed.

```
--------------------------------
      Welcome, StudentName1
--------------------------------
 1 -- View all courses
 2 -- Add courses to your list
 3 -- View selected courses
 4 -- Drop courses in your list
 5 -- View grades
 6 -- Return to previous menu

Please enter your option, eg. '1'.
4

The courses in your list:
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 1, lecturer: Professor Brandon L Kr

Please enter the ID of the course which you want to drop, eg. 'CIT591'.
Or enter 'q' to return to the previous menu
CIT591
The course isn't in your schedule.

The courses in your list:
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 1, lecturer: Professor Brandon L Kr

Please enter the ID of the course which you want to drop, eg. 'CIT591'.
Or enter 'q' to return to the previous menu
CIT590
Course dropped successfully

The courses in your list:

Please enter the ID of the course which you want to drop, eg. 'CIT591'.
Or enter 'q' to return to the previous menu
q
--------------------------------
```

View grades: Grades for courses taken are displayed in the console.

```
5
Here are the courses you already taken, with your grade in a letter format
Grade of CIS320 Introduction to Algorithms: A
Grade of CIS191 Linux/Unix Skills: A
```

Professor Login

When logging in as a professor, one can view the information for courses they teach, view students list, or return to the previous menu.

```
-------------------------------------
      Welcome, Brandon L Krakowsky
-------------------------------------
1 -- View given courses
2 -- View student list of the given course
3 -- Return to the previous menu

Please enter your option, eg. '1'.
1
------------The course list------------
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 0, lecturer: Professor Brandc
```

After student **StudentName2** with ID **002** has added CIT590, the lecturer can see the student's basic information.

```
Please enter your option, eg. '1'.
2
------------The course list------------
CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 110, students: 1, lecturer: Professor Brandon L Krakowsky

Please enter the course ID, eg. 'CIS519.
Or type 'q' to quit.
CIT590
Students in your course CIT590 Programming Languages and Techniques:
002 StudentName2
```

Administrator Login

When choosing to login as an administrator, one can add/delete/edit/view a course/professor/student information, or return to the previous menu.

```
Please enter your option, eg. '1'.
3
Please enter your username, or type 'q' to quit.
admin03
Please enter your password, or type 'q' to quit.
password590
--------------------------------
      Welcome, admin
--------------------------------
1 -- View all courses
2 -- Add new courses
3 -- Delete courses
4 -- Add new professor
5 -- Delete professor
6 -- Add new student
7 -- Delete student
8 -- Return to previous menu

Please enter your option, eg. '1'.
2
Please enter the course ID, or type 'q' to end.
CIT900
Please enter the course name, or type 'q' to end.
Java
Please enter the course start time, or type 'q' to end. eg. '19:00'
19:00
Please enter the course end time, or type 'q' to end. eg. '20:00'
20:00
Please enter the course date, or type 'q' to end. eg. 'MW'
MF
Please enter the course capacity, or type 'q' to end. eg. '72'
20
Please enter the course lecturer's id, or type 'q' to end. eg. '001'
015
Successfully added the course: CIT900|Java, 19:00-20:00 on MF, with course capacity: 20, students: 0, lecturer: Professor Joseph L Devi
--------------------------------
```

If an admin wants to add a course that already exists in the system, the program should prompt with a message.

```
-------------------------------
      Welcome, admin
-------------------------------
1 -- View all courses
2 -- Add new courses
3 -- Delete courses
4 -- Add new professor
5 -- Delete professor
6 -- Add new student
7 -- Delete student
8 -- Return to previous menu

Please enter your option, eg. '1'.
2
Please enter the course ID, or type 'q' to end.
CIS550
The course already exist
Please enter the course ID, or type 'q' to end.
'
```

If the lecturer of the course we want to add does not exist in the system, we also need to add the new professor to the system first, otherwise, this operation will not succeed.

```
Please enter your option, eg. '1'.
2
Please enter the course ID, or type 'q' to end.
CIT789
Please enter the course name, or type 'q' to end.
Python
Please enter the course start time, or type 'q' to end. eg. '19:00'
9:00
Please enter the course end time, or type 'q' to end. eg. '20:00'
10:00
Please enter the course date, or type 'q' to end. eg. 'MW'
MF
Please enter the course capacity, or type 'q' to end. eg. '72'
90
Please enter the course lecturer's id, or type 'q' to end. eg. '001'
100
The professor isn't in the system, please add this professor first
Please enter the professor's ID, or type 'q' to quit
100
Please enter professor's name, or type'q' to end
Olivia
Please enter a username
prof1
Please enter a password
123
Successfully added the new professor: 100 Olivia
Successfully added the course: CIT789|Python, 9:00-10:00 on MF, with course capacity: 90, students: 0, lecturer: Professor Olivia
```

When adding a new course given by a lecturer, the program needs to check if the course has a time conflict with all of the lecturer's other courses.

NA

```
2 -- Add new courses
3 -- Delete courses
4 -- Add new professor
5 -- Delete professor
6 -- Add new student
7 -- Delete student
8 -- Return to previous menu

Please enter your option, eg. '1'.
2
Please enter the course ID, or type 'q' to end.
CIT100
Please enter the course name, or type 'q' to end.
Python
Please enter the course start time, or type 'q' to end. eg. '19:00'
16:30
Please enter the course end time, or type 'q' to end. eg. '20:00'
17:00
Please enter the course date, or type 'q' to end. eg. 'MW'
M
Please enter the course capacity, or type 'q' to end. eg. '72'
10
Please enter the course lecturer's id, or type 'q' to end. eg. '001'
029
The new added course has time conflict with course: CIT590|Programming Languages and Techniques, 16:30-18:00 on MW, with course capacity: 
Unable to add new course: CIT100|Python, 16:30-17:00 on M, with course capacity: 10, students: 0, lecturer: Professor Brandon L Krakowsky
-------------------------------
```

After adding a new course, we can see the newly added course CIT900 in the system.

```
CIS660|Advanced Topics in Computer Graphics and Animation, 10:30-12:00 on MW, with course capacity: 32, students: 0, lecturer: Professor Stephen Lane
CIS670|Advanced Topics in Programming Languages, 13:00-15:00 on MW, with course capacity: 32, students: 0, lecturer: Professor Stephan A. Zdancewic
CIT900|Java, 19:00-20:00 on MF, with course capacity: 20, students: 0, lecturer: Professor Joseph L Devietti
```

An admin can add a new student/professor to the system.

```
6 -- Add new student
7 -- Delete student
8 -- Return to previous menu

Please enter your option, eg. '1'.
6
Please enter the student's ID, or type 'q' to quit
007
Please enter student's name, or type'q' to end
Bob
Please enter a username
test
Please enter a password
123
Please enter ID of a course which this student already took, one in a time
Type 'q' to quit, type 'n' to stop adding.
CIT590
Please enter the grade, eg, 'A'
A
Please enter ID of a course which this student already took, one in a time
Type 'q' to quit, type 'n' to stop adding.
n
Successfully added the new student: 007 Bob
```

When adding a new student/professor, the program needs to check if the ID and username already exists in the system.

```
4 -- Add new professor
5 -- Delete professor
6 -- Add new student
7 -- Delete student
8 -- Return to upper menu

Please enter your option, eg. '1'.
4
Please enter the professor's ID, or type 'q' to quit
015
The ID already exists
Please enter the professor's ID, or type 'q' to quit
099
Please enter professor's name, or type'q' to end
Bob
Please enter a username
Lee
The username you entered is not available.
Please enter a username
'
```

When deleting courses/professors/students/, the program needs to check if the courses/professors/students/ in fact exist in the system.

You may assume all input is valid.

Your Tasks:

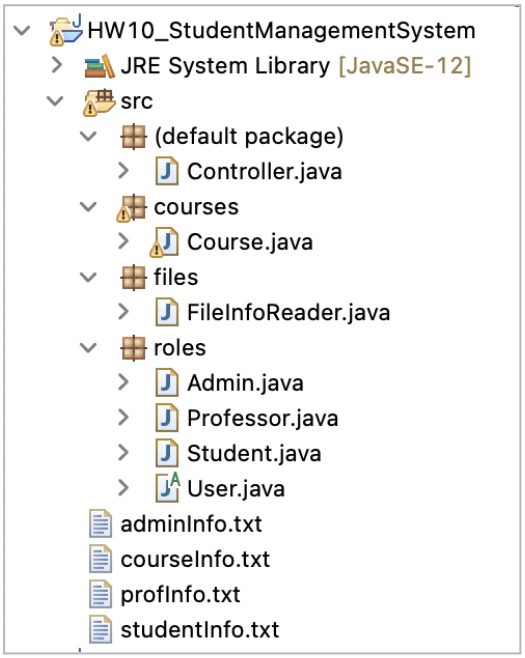
1. **Read and parse the provided files in Java, cleaning them up if/when needed.** You may assume the information in the files is valid.
    a. Courses information file – *courseInfo.txt*. This file contains information for a number of courses. The information for each course is on a separate line. The pieces of information for each course are separated by semicolons (";"). We want you to read the file in and load the information.
    b. Admin information file – *adminInfo.txt*. This file contains basic information for administrators including username, password, name, and ID. You need to read the file and load the information.
    c. Student information file – *studentInfo.txt*. This file contains basic information for students.
    d. Professor information file – *profInfo.txt*. This file contains basic information for professors.

e. You may assume all of the information in the files is valid. For example, all information for professors in the *courseInfo.txt* file is given in the *proInfo.txt* file. We suggest that you load the list of professors before loading the list of courses.

2. **Design the student management system**
   We are not going to provide you with a specific design for this homework. Feel free to design your own student management system. We do, however, require you to have the following:
   a. **At the very least, you MUST have a Controller class that launches and runs the management system.** This class displays the menu, helps with user login, accomplishes the different user operations and continuously takes in user input. The Controller class must have the main() method.. **Note, the Controller.java file SHOULD NOT be in a package. (See screenshot on the next page.)**

   Here are some other suggestions:
   a. You need a **FileInfoReader** class that reads and parses the files.
   b. You need a class that defines a **Course**. The **Course** class is expected to have instance variables which store all of the information for the course. It also needs to provide functionality, for example, checking if one course has a time conflict with another course, adding/removing students from the course, and other helper methods you may need.
   c. You need classes that define a **Student**, **Professor**, and **Administrator**. And they are expected to share some common attributes and to share as much code as possible. We want you to think hard about how you can accomplish this. This answer lies within the scope of the object-oriented concepts we have covered in this course. For example, you might have a **User** class with a subclass **Student** for the functionality of student operations like adding/dropping/viewing courses, a subclass to represent **Professor**, and a subclass **Admin** to add/delete/view information for courses, students and professors.

   Here is an overview of the suggested class (and package) structure described above.



Testing

Regardless of your design, we expect you to write unit tests for every public method. The only public methods that you can leave untested are the following: methods that get input from the user or simply display information to the user, and simple getters and setters.

You do not need to test helper methods that are declared private.

A method that reads and parses files could still be tested. Also, if a method does some computation and prints to the console, then the computation part of the method should still be tested.

write

You can create any number of test files and place them in any package, but **you MUST name the test files in the format "<YourClass>Test.java"**. For example, if you were to create a test file for your *FileInfoReader* class, you MUST name it *FileInfoReaderTest.java*.

Javadocs

Add Javadocs for all classes, methods, and variables.

What to Submit

Please submit all packages and classes in your entire Java project. Make sure it includes everything in your "src" folder.

If working in a group, both team members must submit. Include the names of the members of your team as part of the @author tag in the Javadocs for all of your classes.

Evaluation

You will be graded out of 40 points:

- Code writing, functionality, and design (20 pts)
  - At the very least, did you write a Controller.java that launches and runs the program?
  - Did you make sure Controller.java is not in a package?
  - Does the system work as expected?
  - Did you make good design decisions about code reuse?
    - How much code is being shared between Student, Professor and Admin?
    - How is this code sharing being achieved in your design?
  - Does the code take too long to run? (10 seconds would be too long!)
- Loading, parsing, and navigating files (10 pts)
  - Did you keep file I/O in as few methods as possible?
  - Did you correctly load valid data from every file?
  - What data structure(s) did you use?
- Unit testing (5 pts)
  - Are your test filenames in the format <YourClass>Test.java? For example, a test file for a class named *FileInfoReader* must be named *FileInfoReaderTest.java*.
  - Did you write unit tests for every public method (excluding simple getters and setters)?
- Style & Javadocs (5 pts)
  - Adding Javadocs to all methods and variables, and comments to all non-trivial code
  - Properly naming variables, using proper indentation, etc.