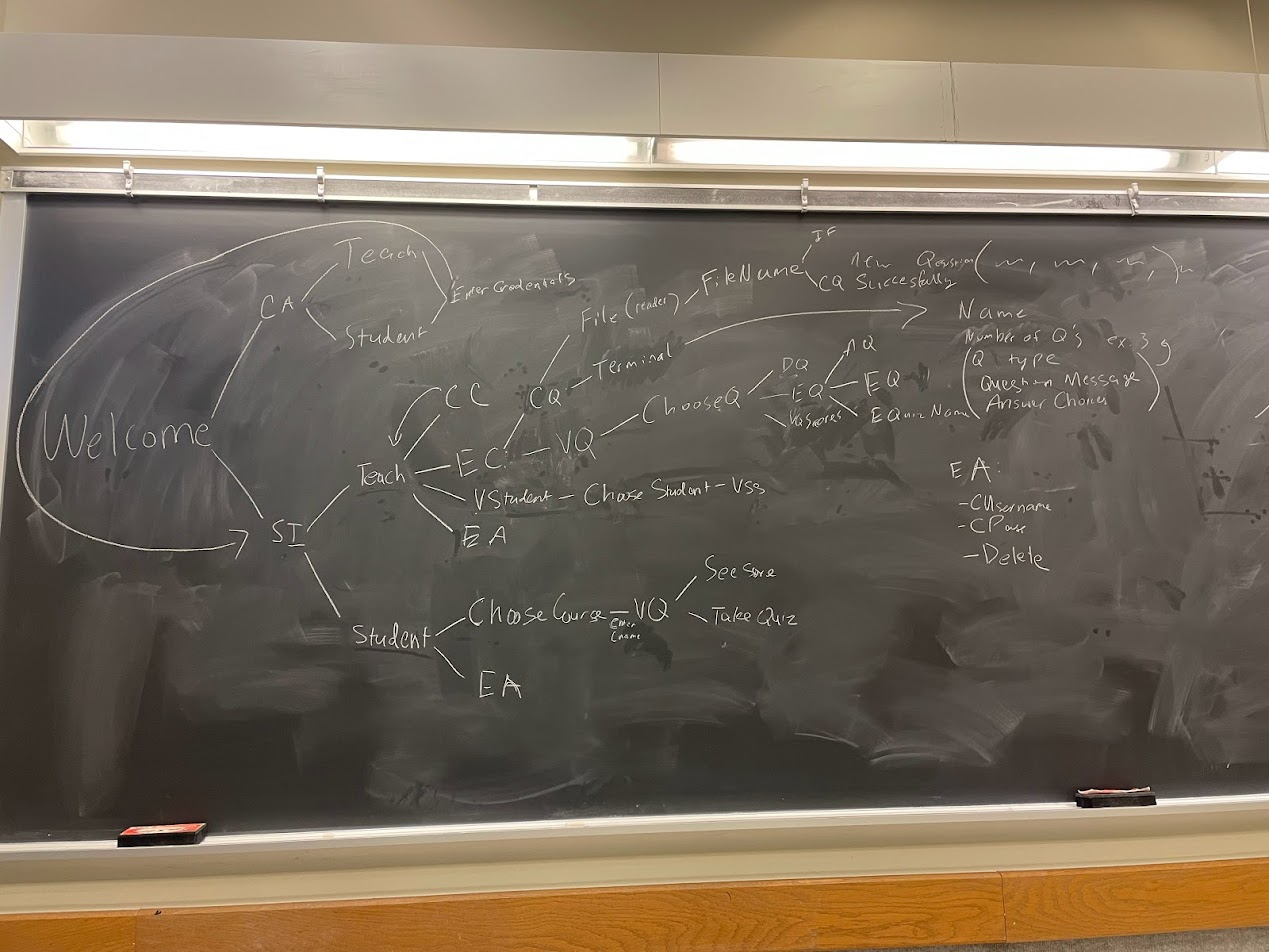
**Arul Chauhan, Ryan Chen, Zachary Garcia, Jebran Syed, Andrew Wu**

**CS18000 Project 4**

**Part One**

Our group selected the quiz option for our project. Since all the user communication was done through a terminal, we optimized our system around receiving input from the terminal and performing actions based on that. The first step is to welcome the user into the program. Once done, they can either select to login to an existing account or create a new account. If the user chooses to create a new account, they can either create a teacher or student account by simply entering their desired username and password. If their username is already taken, the program will prompt them to enter a different username. Once an account has been created, the user can login with that account and gain access to the main program. The access gained depends on whether the user logged in with a teacher or student account. As a student, the user can either edit their account (which includes deleting the account), or they can select a course to work in. From there, the student can choose which quiz they want to work with, and from there they can either view their score if one was given by a teacher, or take the quiz if they haven’t before. On the other hand, as a teacher the user has 4 options. They can edit their account (the same actions as a student account), create a course, edit a course, or view a specific student’s data. If the teacher selects to create a course, they will be prompted with a name for the course. Once entered, the course will be created. If the teacher decides to view a student, they will be prompted to select a specific student and from there all the student’s scores on quizzes will be shown. Finally, if the teacher decides to edit a course, they can either create a quiz in the course or view an already existing quiz. If they choose to create a quiz, they can do so through either a file input or through the terminal. If they choose to input the quiz through a file, they will be prompted for the file name, to include file format and the quiz will be created off of that. However, if the file is non-existent, can’t be accessed, or is in the wrong format the program will tell the user the file couldn’t be read. If the quiz is successfully created, the program will print a success message. If the teacher decides to create a quiz through the terminal, they will be asked to enter the name of the quiz, followed by questions in a certain format. If the teacher selects to view a quiz, they will have to enter the name of the quiz they are trying to view. From here, they can delete the quiz, edit the quiz, or view scores. If they choose to edit the quiz further, they can add questions, edit questions, or edit the quiz name. The original version of our application flow is shown below. Note that some minor details may not be perfectly aligned with the implementation; this is just a reference to our design process.



**Part Two**

**Andrew Wu Individual Portion:**

I handled the menu portion of the project. I worked on the logic portion that calls all the methods that the other members of my group worked on. The menu file starts off with printing a welcome statement, then asking the user if they would like to create an account or sign in if they have an account. If the user creates an account, the menu prompts the user if they are a teacher or a student and then gets their username and their password for their account. If the user wants to sign in, they input their credentials and depending on their account type (teacher or student), they are prompted with different options. If you are a teacher, you can view the quizzes already associated with your course, edit your course, which involves editing or creating a quiz, viewing a student’s grade for a particular quiz, or edit their account, where they can edit their username or password, or delete their account. If the account type is a student, they can edit their account like how a teacher can and choose a course where they can select a quiz and view their grade or take the quiz. At any point within the program, the user is able to quit the program, allowing them to stop accessing the program at any point.

**Zach Garcia Individual Portion:**

I created the Course, Quiz, and Question classes, apart from some methods that are involved in the project selections. I also created the CourseTest class, which tests Course, and subsequently tests Quiz and Question. Additionally, I am responsible for Menu.init(), which re-instantiates each Course, Quiz, and Question object by reading from the main.txt file. I designed the file flow completely. The file flow is essentially how everything gets stored into main.txt to be parsed on the next boot up of the program. My Course methods include the following: constructor, setQuizList(), getQuizList(), addQuiz(), editQuiz(), deleteQuiz(), getCourseName(), setCourseName(), toString(); my Quiz methods include the following: non empty constructor, addQuestion(), editQuestion(), deleteQuestion(), getQuizName(), setQuizName(), getQuestionList(), toString(); my Question methods include the following: constructor, addChoice(), editChoice(), deleteChoice(), getQuestionMessage(), setQuestionMessage(), toString(), getChoiceList(), setChoiceList(); Most of the system output is handled in the Menu class, but some of my methods include system output also, as to make it easier to incorporate into the Menu class.

The flow that a user takes requires the use of numerous methods involving the creation, editing, and deletion of different courses, quizzes, questions, and choices (choices subset of questions, questions subsets of quizzes, quizzes subsets of courses, while courses are stored in teacher files, which are stored in main.txt [I use the notion subset here; that is NOT to say subclasses]).

If I had another opportunity for this project, I would hope to more carefully think about the application as a whole instead of just my portion. This is because a few of us caught a huge error somewhat later on: not everything was being saved to be kept for later uses after a user logs off. This epiphany led to my design of the file flow described earlier. For project 5, my team and I have already discussed starting much earlier and planning out what kind of work we’ll be doing. Although I think that the requirement to start communicating with our team should have been a week earlier than April 1st, as the deadline for the entire project is April 11th.

**Ryan Chen Individual Portion:**

I was responsible for the selections: File Imports and Randomization. Additionally, I created/assisted the implementation of the preexisting control flow in the Menu.java class, the core features involving Quizzes,Teachers and Students. I created the file import/attach prompts for teacher/student including the quiz.txt format. Additionally, I created the importQuiz() method, which takes in the file from the according file path and creates a local file with the correct signature. I also helped create the takeQuiz() method which allows the student to take the quiz. The outputs of this method create a file with the responses of the student which include answers to different question types (T/F, multiple choice, fill in the blank) and also allows the student to attach filenames. This username\_quizname\_attempt#.sub.txt file stores student answers to the quiz and is time-stamped and added to the quizname\_attempts.txt master file for the associated quiz. To actually take the quiz, I designed the readQuiz() method which was able to re-instantiate the chosen quiz object by parsing strings from the file into their respective Quiz.java and Question.java fields. This readQuiz() method returns a Quiz object with the correct fields instantiated and is passed to a shuffleQuiz() method, which I designed as part of the randomization selection. The shuffleQuiz() method takes in the re-instantiated Quiz object and randomizes both the questions and also the choices associated with the question. I implemented the methods readQuiz(), shuffleQuiz(), and takeQuiz() methods, added required prompts and added additional control flow to Menu.java to allow the student to take the quiz. I also worked extensively with Zach within the Quiz/Question class. I modified/tweaked the Quiz methods [ addQuestion(), editQuestion(), deleteQuestion() ,toString()] and Question methods [toString()] to implement the control flow/prompts that allowed the Teacher to create quizzes (add questions/add choices), edit the quizzes (edit questions and their corresponding choices), and deleting questions and associated choices. To make sure these changes were implemented correctly, I also added an exportQuizToFile() method that essentially overwrites the quizname.txt file in the correct, edited format. Lastly, I aided in creating test cases for these sections listed.

If I had another attempt at this project, I would plan to meet earlier with the group to hash out the overall program flow, potential errors/cases, classes and methods needed, and other details prior to splitting the work. Additionally, I would have changed the way work was originally divided so that the work was originally split into more individual/self-sufficient modules which allowed for full autonomy in design, implementation, and testing before combining the modules together at the end. For project 5, it would be wise to front-load and have a more detailed design planning period with multiple required meetings at the start so that members could start actually coding and finding errors/potential error cases while doing so with more spaced out meetings for the remainder of the project.

**Jebran’s Individual Portion:**

I was primarily responsible for the Account class. This encompassed the process of creating accounts, login validation, account editing, account deletion. Additionally I was responsible for integrating all of these methods in the Menu.java class which is responsible for the flow of the entire program. The main parts of my work revolved around 4 methods: createAccount(), login(), deleteAccount(), editAccount().

The createAccount method prompts the user for their designation i.e Student or Teacher, followed by which they are prompted for a username which can only be a string containing no spaces and then a password which can also only be a string that does not contain any spaces. Based on these details, a new file is created for the user which contains their credentials and the username is added to a usernames file. The login method prompts the user for a username, checks if the username exists or not by iterating through the usernames file and then prompts the user for a password. If the credentials are validated by going through the respective username’s file to check for the password, the user gains access to the application. The deleteAccount method is executed when an account is to be deleted. This method ensures that the file created for the user is deleted and that the user can no longer use their previous credentials to gain access to the program by deleting the username from the usernames file. The editAccount method essentially allows a user to edit their credentials i.e username and password. Users are given two options when this method is called. They can either change their username or change their password. This changes the credentials in the respective user’s file. After this process they can login with their updated credentials while still having access to their previous data. I was also responsible for creating AccountTest.java which contains a main method that tests the functionality of the four methods in Account.

The things I would have done differently for this project would be the way usernames and passwords are to be retrieved from files instead of storing them in one line and limiting users due to a regex. This would provide users more flexibility in terms of the type of password and usernames they could have. Furthermore, I would have tried to figure out a more efficient way to store the user details instead of creating new files for each student or teacher. I believe all of these issues that I have pointed out can be addressed and made more efficient in Project 5 by leveraging the Server-Client architecture.

**Arul’s Individual Portion**

I was responsible for the Teacher class, which included a field of courses which the teacher has created. The class was centered around the following methods: createCourse, checkCourseExistance, printSubmissions, parseSubmissions, and checkStudentSubmissions This included all the courses the teacher has ever made, and is stored as an ArrayList since the teacher should be able to create multiple courses at a time. Create course simply creates a course given an Account, as the account should be a teacher when passed in. It won’t create a course if the name already exists and puts a $ modifier in front of the file. CheckCourseExistance checks if a course exists in the list, print submissions prints all the submissions on the file, parsesubmissions adds the submissions to an arraylist, and checkstudentsubmissions checks a specific student’s submissions.

**Part Three**

For project 5, we plan to meet and plan every aspect of not only the application flow, but also everything involving roles and work distribution. This way, we can catch large errors earlier than we had to for project 4. We are playing with the idea of splitting the group up into pairs so that roles that work more closely together may do so without inhibiting the rest of the group. For example, if one person is managing a server class and another person is managing a client class, they may work together closely while the rest of the team manages the GUI and concurrency aspects of project 5. We would prefer to meet twice or three times per week until the project is completed. This is what we did for project 4. We have already talked about which team members are stronger on concurrency, which team members are stronger on GUI, and which team members are stronger on server side and client side applications. This way, we already have an idea of how to split the work. In hindsight, our collaboration for project 4 was superb and we hope to replicate that experience for the next project. With these plans, the next project should be completed without errors.

4/11/2022 - Meet to discuss project 5

4/13/2022 - Design workflow, assess desired work distribution,

For this last project, we had no conflicts. We expect to have the same experience for the next project. Should conflict actually arise, we will discuss what we think should be done in relation to the problem(s). If it is a work issue, it will be easy to revolve since we worked so well together last time. For other disagreements, we will simply have more meetings to resolve simple or complex issues.