Rollins Club Creator

CMS 270

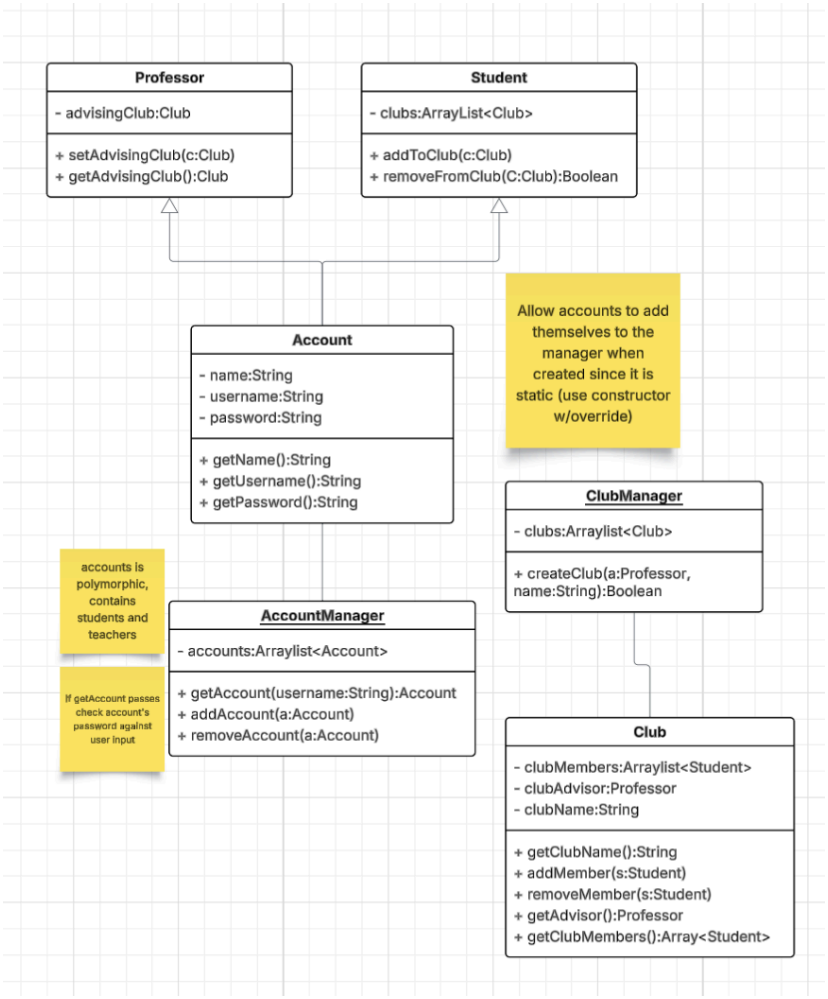
Jonathan: Backend
Robert: Frontend
Tyler: Documentation and Presentation

Project Overview:

The Rollins Club Creator is a Java-based program designed to increase efficiency, simplify, and streamline the student club creation process. Students often need a way to propose new club ideas, while professors and administrators need a way to review and approve them. Our system digitalizes this, allowing students and professors to create accounts, log in, and perform the actions required for their roles, such as submitting club requests for students and accepting or declining them for professors.

We chose this project because it directly connects with something Rollins can definitely improve, and it has the potential to demonstrate that object-oriented programming can model real-world applications. This project also allows us to implement key Java concepts, including encapsulation, inheritance, abstraction, UI integration, and object relationships.

UML Diagram:



OOP Implementation:

We designed our project to use core object-oriented programming principles, including inheritance, encapsulation, and abstraction.

Inheritance:

Our program uses inheritance to define two user types: students and professors. These classes inherit properties such as name, username, and password from the account class. Both of these subclasses share the same basic properties but also have some unique properties, such as creating club requests for students and accepting/declining club requests for professors.

Encapsulation:

All classes except the manager classes are fully encapsulated. All of their instance variables are private and must be accessed/modified through getters and setters. This prevents changes that shouldn't occur in the program and keeps the internal logic hidden from the rest of the program.

Abstraction:

Abstraction was used in the design of the account system. The account class acts as a representation for all users (both students and professors) without specifying details about each role. This allows the program to treat all users the same and have general features such as logging in, while allowing each role to have different behaviors.

Manager Classes:

We used two manager classes, accountManager and clubManager, as controllers in our program. The accountManager class handles logging in, account creation, and user look-up. The clubManager class handles updating request statuses for proposed clubs.

Challenges and Solutions:

During the development of our club creator system, our team encountered a couple of challenges that helped us better understand Java and UI design.

1. Creating Dynamic Buttons for Request Viewer

One challenge we faced was dynamically generating buttons for the request viewer based on the number of submitted requests. The main issue was maintaining a proper reference to the specific class instance each button corresponded to. Each button triggered the wrong action or failed to make the corresponding request.

We figured out the issue by changing how the buttons were created and assigned, ensuring that each button stored the proper request instance or used the adequate variable in the ActionListener. This ensured the UI correctly correlated the button being interacted with with the intended action.

2. Implementing GUI functions before learning about ActionListeners

Another challenge we faced was that at the start of the project, we attempted to build functional GUIs before completely understanding ActionListeners. This led to confusion about how user buttons, events, and interactions should occur. This made it challenging and complex to connect the UI to the program's logic at the beginning of the project.

Once we went over and fully understood ActionListeners in class, we revisited and redid most of the GUI code. With an adequate understanding of event handling, we were able to implement button behavior correctly, switch between screens, and effectively communicate between UI elements and backend logic.

3. Reformatting to MCV

The initial prototype connected the UI directly to the code with static references, making it hard to test, debug, and expand as more complex features were required.

We reformatted the code to MCV, with the controller serving as an intermediate between the backend and frontend and holding non-static references to manager classes.

Contribution Summary:

Jonathan:

- Wrote initial UI and starter code
- Refactored the program to MVC

- Rebuilt major parts of the UI and controller logic
- Integrated backend and front end using CardLayout
- Did multiple extensive overhauls to improve structure and functionality

Robert:

- Expanded GUI with new screens and layout improvements
- Reworked and updated UI after refactor to align with MVC structure
- Performed multiple UI refinements near the end of the project

Tyler:

- Wrote documentation, including the project report
- Created and designed the PowerPoint presentation
- Organized OOP explanations, challenges, and project summary