# CSC207: Phase 0 Progress Report

**Group Members:** Aliyan, Bernard, Ruiting, Tiffany, Miguel, Linh, Anita, Leya

**Specification:**

The domain for our project is a Banking app with a built in financial planner. The app allows the user to create an account (or login to an existing account) which is linked to the their chequing account, savings account, and total balance. Using these linked accounts, the user is able to perform a variety of actions when banking. For example, the user is able to deposit, withdraw, or transfer money. They are also able to view transaction history which displays the previous transactions made on the accounts.

Additionally, the users financial information is used for financial planning specifically curated to each individual. For instance, they are able to view projected growth of their investments using their investment performance history as well as the interest they accumulate over a certain period of times.

**CRC Model:**

We have a total of 22 CRC cards, dividing them into 5 different categories. The plan is have each category represent a package and whichever card was placed in that category would be put in that package for the project.
To look at all CRC cards, refer to github project folder here:
https://github.com/CSC207-UofT/course-project-marble/tree/main/CRC%20Cards

Here is a list of all the class names generated from our CRC model, divided into the category they belong in:

- Server: OwnerRepository and Server
- ClientUI: ClientUserInterface and CallActions
- Entity: Account, Chequing, Owner, Savings and Transaction
- Actions: Actions, Transfer, Withdrawal, CheckLogin, GetTransactionHistory, Deposit, ViewAccounts and CheckBalance
- Action_request_response: ActionRequest and LoginRequest

The general plan in sorting these categories is to based off the layers in clean architecture. By basing our packages on where each CRC card categories, we draw a clear line on what classes can be accessed by what other classes based on which package ("layer") they belong in.

**Scenario Walk-Through:**

The scenario walk through specifies 2 different scenarios and how the banking app + financial planner will be used to accomplish these tasks. The first scenario is when Person 1 (new user) wants to use the app to send money to Person 2 (existing user). In this scenario

Person 1 must download the app and create a user account (Owner). Making a new user account will also give them access to making chequing and savings accounts under their name. Person 1 will now be prompted with actions he is able to do with his account. In this scenario, they will select "transfer". Person 1 will fill in Person 2 information and the money will be sent, automatically deducting the same amount from Person 1's account balance while adding the amount to person 2's account balance.

The second scenario is when an existing user wants to track investments, assets and loans. The user will login to his account and will create a new portfolio. To track his stocks the user may enter what stocks he holds in the current holdings field. To track his loans the user may enter his student debt information in the liability field. To track assets the user may enter his watch value in the assets field. Using this information, the users net worth is displayed on the home page.

**Skeleton Program:**

Our skeleton program consists of the server, clientUI, entity and actions folder:

- Within the server folder we have implemented the OwnerRepository class which creates an owner object and a Server class which is responsible for receiving input from the user and allocating it to the appropriate class.
- In the clientUI folder we implement ClientUserInterface class which is responsible for creating a user account, logging them in, and gives the users prompts on using the banking app. It also implements the CallActions class which takes an ActionRequest object and calls the corresponding action that is used to process the request.
- The entity folder contains the varying entities for our program. It consists of the Account class (with Chequing and Saving subclasses), Owner class, and Transaction class.
- The actions folder contains various actions that a user is able to do with the app. This includes CheckBalance, CheckLogin, Deposit, GetTransactionHistory, Transfer, ViewAccounts, and Withdrawal. All of these classes extend an abstract class called Actions.

Note that the unit tests written for phase 0 can be found in the CheckLoginTest.java, OwnerRepositoryTest.java, ClientUserInterfaceTest.java, and LoginRequestTest.java. The tests for the other classes are currently pending and will be added in Phase 1.

**Questions:**

- What would be the best method for data storage based on our specific project and implementation?

**Design Pros:**

Our current implementation and the design of our program works well so far because there is a clear flow and dependency between adjacent layers. We have mimicked the clean architecture method for the design of our program which allows for an organized, readable program. We also focused on the SOLID design principle. A particular example of this is using the Single Responsibility Principle in which we made sure that every class has a single responsibility to limit the complexity of each class.

**Current Contributions and Future Plans for Each Member:**

| **Name** | **Plan** |
|---|---|
| Aliyan | Contributions: Implemented the Server and ClientUserInterface<br><br>Future Plans: Make the Server able to handle multiple users at once and be able to save OwnerRepository for later use. |
| Anita | Contributions: Implemented the Deposit class<br><br>Future Plans:<br>Implementing Transfer class<br>Unit testing for actions (Deposit and Transfer)<br>Collaborating on additional features in Actions package |
| Bernard | Contributions:<br>- Implemented the Entity package, in collaboration with Tiffany.<br>- Wrote up the docstrings for most methods.<br><br>Future Plans:<br>- Collaborate and coordinate with the people that work on the Action and help them utilize the Entity package smoothly.<br>- Implement more entity classes if the team is going to expand functionalities. |

| | |
|---|---|
| Leya | Contributions:<br>- Implemented the CheckBalance and ViewAccounts class.<br>- Worked on CRC cards<br>- Worked on Progress Report<br>Future Plans:<br>- Implement spending tracker<br>- UI<br>- Unit testing for actions (CheckBalance and ViewAccounts)<br>- Fix documentation |
| Linh | Contributions:<br>- Implemented the Withdrawal class<br>- Worked on CRC cards<br>- Worked on Progress Report<br>Future Plans:<br>- Implement the TransactionHistory class<br>- Create missing documentation and fix up existing documentation<br>- Create unittests for Withdrawal and TransactionHistory |
| Miguel | Contributions: made testing files except CheckLoginTest, general testing classes worked with Ali to test server working over different IPs<br><br>Future Plans:<br>-GUI<br>-UI<br>-finishing testing classes in particular network testing |
| Ruiting | Contributions:<br>- Implemented ActionRequest, LoginRequest, CheckLogin, worked with Aliyan to implement login() in ClientUserInterface and Server<br><br>Future Plans:<br>- Implement other subclasses of ActionRequest for data transfer<br>- Keep working on Server and ClientUserInterface class |

| | |
|---|---|
| Tiffany | Contributions:<br>- Implementing the entity packages and their methods: Account, Checking, Savings, Transaction, Portfolio...<br>- Collaborated with Bernard on all those classes.<br><br>Future Plans:<br>- Working with peers in Actions and fulfilling additional features<br>- Expanding entities |