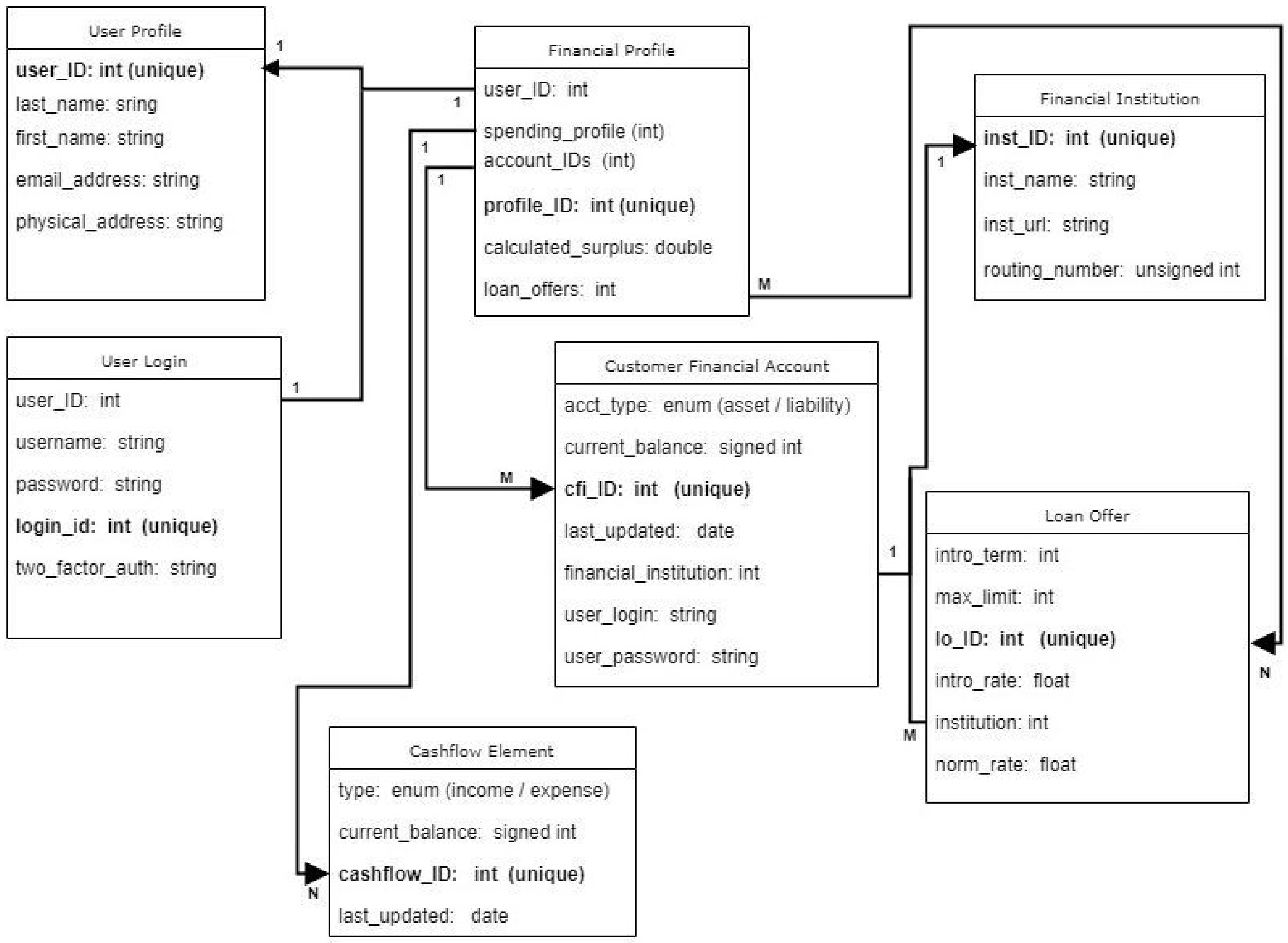
Homework 4

Group 10
Class: CS-361

I. UML class diagram:



Key objects will include:

- User Profile Object: describing fundamentals of each user
- User Login Object: linked to User Profile - login, password, UserID
- Financial Profile: collection of financial data for individual user
- Financial Account: a single financial account (checking, savings, credit card, mortgage, loan, etc, etc.)
- Cashflow Element: describes both an income element (salary, interest, dividends, etc.) and expected expenses (rent, utilities, tuition, etc, etc.)

- Financial Institution: describes known financial institutions, and store key data (routing number, etc.)
- Loan Offer / Opportunity: describes each known loan offer available to customers (including interest rates, introductory terms, limits, etc.)

II. Packaging of these entities & implementations

We began with several object oriented classes; User Profile, User Login, and Financial Profile. All three of these classes could fall under one module and that module could be named User. These classes provides a Coupling method to each other, we would realize that Coupling could make for a better security to the data and software. We also would think that Coupling will reduces the maintainability when it comes to software development and that it goes against the notion of all the quality attributes of great software development. It's a trade off that we are willing to make in order to make our software more robust in term of security and user's data integrity. Because of the security reason the classes depends on each other in some form or fashion or another. The possibility of a piece of data is change in one class could affect another is very likely, hence the arise of the levels of difficulties can be realized when maintaining the code between classes. When a user changes their password or profile information, sometimes that piece of data will also have to change or copy over to another class. There are no one single point of failure or breakdown in one particular class without another class realizing it. This makes it harder to maintain the code, but it also makes it harder for security failure or hacking because of all the classes have to be in agreement of each other about the data.

The Customer Financial Account class and the Financial Profile class are also tightly coupled. A user's financial profile can contain many types of financial accounts. So when a new piece of data in the Customer Financial Account class is updated or changed, this will also need to be reflected back to the user's Financial Profile class. The Financial Profile and Customer Financial Account classes would have to depend on the incoming data from outside sources. Especially when we look at the Customer Financial Account class, the application would have to talk to an outside source, most likely the user's banks or credit card provider to get the necessary data.

The Cashflow Element class are not as tightly coupled to the user's financial profile class as the customer financial account was. We could almost say that its falls somewhere between half on the Cohesion method and Coupling method. The users simply input their spending information and all the data is sitting in that class with very little gets to transfer to other classes. Although the piece of data about the user current balance falls under the Coupling method. As because the cashflow element class will have to reach out to outside external source, i.e. user's banks or credit card provider to get the updated data and bring that back into the application database.

As far as the Loan Offer class, we would most likely classify this as a cohesion class. Even though the Loan Offer class data will flow to other classes, but it does not rely on outside classes or entity to update its data. The financial institution makes and set the data, it is a functional or informational type of cohesion in that it works together with the financial institution class to set up data for one purpose. This Loan Offer class is a foundation to other classes, other classes relies on the data coming out from this class to run their operations. Some of the other classes that taps into the Loan Offer class that would in turn make it into the Coupling method are; both the Financial Profile and the Customer Financial Account class.

So in moving forward with modules, we can group the Financial Institution and Loan Offer class into a single module, we could called it External module or some sort. Other modules or classes would be considered coupling to the External module as because they depends on the External's module data.

III. Assessment of incremental vs. iterative development

Our design will support iterative development. The app is designed to collaborate amongst various financial institution website and retrieve financial data from all the sources and store it in a similar format in the database.

There should be consistency in the financial data stored from all the sources since our app will plot spending charts, tracking financial transactions and upcoming bills from the data.

So, during testing when we will be trying to access information from different sources there can be some differences as to how the app is able to access the data. Some website might be encrypting their data some might not allow third party access.So, while testing we need to develop our app and modify it in cycles to accommodate the variation in websites of different financial institutions.

.In the iterative method we are focused to create a working prototype first and then adding the desired functionality in incremental steps. As per our prototype the section dealing with the goals for the user is another good example of iterative development. In this segment as user is allowed to build custom goals as well. This section will use a similar algorithm each time and get interest rate, expected time period to build a fund and then app will compute based on user's income or spending if they would be able to attain these goals or they need more time to achieve their target.

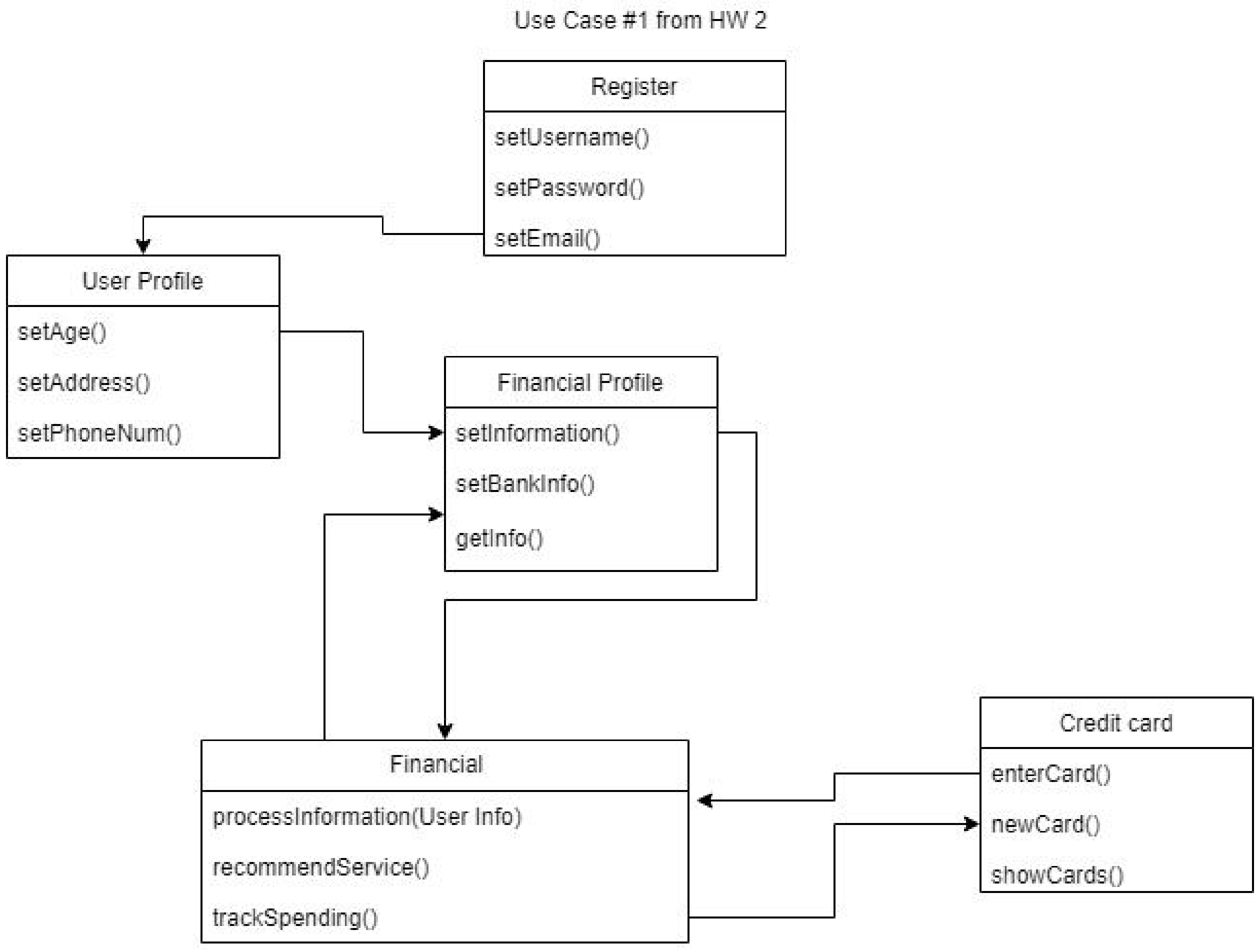
IV. Useful Design Patterns

The design pattern that are present in our system is a combination of Facade, Observer, and Mediator. Once a user creates an account or logs in, the financial profile is displayed and is the main object in which the user will access. This will contain all the information that is relevant to the user in the program. Essentially, when the user makes any changes to their profile, the program will use this new information to recalculate and update the Customer Financial Account in the background. This is shown with the other objects that are linked; Customer Financial Account, Loan Offer, and Cashflow Element. The profile information is processed through these objects. Since the user is the ultimate in the decisions regarding information, the program will adjust accordingly and run with those requests.

Here, we are using the also using a Mediator and Observer structure when the program sends a request to the Financial institution and Loan Offer for information. The program acts as the connection between the user and institution(s) while also using the information given. For example, there will be a list of loan offers that the user can choose from - a list of offers from loan institutions that will give options to the user. On the other hand, the program also requests financial information from bank accounts and credits card services that the it will use to calculate and update information for the user profile. These elements in combination with the facade and observer structure becomes the bulk of the design.

Moving forward, the design we have may change to contain more a Strategy design pattern. By elaborating more on what kind of direction the user wants to take; the time period, the type of loans/institutions, etc., we can determine which strategy the user can use to achieve their goals. This design pattern will likely contain multiple standard algorithms that can be modified based on the user settings. For example, if the user wants to be able to pay off their debt using strictly loans, we can use one of the algorithms that utilizes only the loans.

V. Use-case sequence diagram



Following the Use case from Homework #2 The user starts by creating an account using the set functions to initialize the username, password, .etc. They then go into their profile and add their information along with their financial information. That data is processed by the system using functions to calculate the best credit card for the user. The financial and credit card classes work together with the financial profile as it changes to update the best options.

VI. Interfaces

Many of the interfaces we need require contracts that check if the necessary data is there. For example, in the financial class we require that the user has entered data before the functions can be used. This is done between the database and the class. Another interface that directly interacts with the user is the user login interface. The user must enter the information before proceeding onwards to the rest of the system. That information must be accurate as well. Unique to that person's login info. This will be handled by an XML request as we will be able to pass data from the database to the interface.

There will be an interface between application and financial account server. The application will send a request to fetch financial data. The data for the financial transaction needs to be passed securely and saved in the application database. This step involves information to be transferred securely.Also, if the app remains inactive for long time or if the server for the financial institution is unreachable the application needs to securely logout the user.

When receiving data from the user the interface needs to be secure. This system is handling people's private financial information and needs to have several functions that will protect it. If the financial information is unable to be accessed the function needs to know that and return a value to the interface that can be displayed to the user.

Displaying user data will be done with XML requests so we can extract and display their information. What is important is that the functions being used to get access to that data must be secure.

Also one of the interface that we think is a key component in application is the web crawler function. Basically this small application with in the larger system plays an important role in scrapping the internet for data. In a sense of contracts, the internet already provides us the data that we need. The way we see it, is that the internet will always honor its contract in providing us the data. What we must do next with the data that is gathered is also equally important. Once we have the data from the web crawler interface, internally we have to sort out the pertinent data and save it back in to our application database. The Loan Offer class then is contractually bound to have those data ready for other classes to tap in. The Loan Offer class also acts as an interface between the processed data coming from the web crawler to any classes inside the application architecture.

VII. Likely Exceptions and Handling

Some of the exception that can occur are:

1. Once the user has completed their profile. After using the app for some time. If they close one of their financial accounts or the account gets frozen due to inactivity. The exception handler for the app in this case will notify the user to either delete the account if they have closed the account or inform them about the inactivity in the account. The user can delete the account or can bring the account into good standing.
2. In case user has some foreign financial accounts. If the app cannot access those accounts due to financial rules and regulations of foreign country the app can notify user to either not include the financial account at all or if the user wants to incorporate that financial data they can manually enter data using form based input.
3. Encryption of financial data by financial institutions. If the financial data is encrypted for some reasons and the app is unable to retrieve the data and store in it’s database.The user can be prompted to input the data using form based input or not include the account if they user doesn’t use the account regularly.
4.

VIII. Team Member Contributions:

- Aseem Prashar: Incremental vs. Iterative development and exception handling.
- Forrest Allen: Use-case sequence diagram
- Thach Vo: Package the implementations
- Khoa Phan: Design patterns
- Michael Volz: ULM Class Diagram