Project Phase 1 -- Specification for the ATM Machine

You will be writing a program that runs an ATM (Automated Teller Machine), also called a "cash machine". The program will display options on the screen and the user will select an option by typing on the keyboard. In real life, the user would press a touch screen, instead. We will assume that the ATM is the only interaction that the user has with their bank account. There is no physical bank at which your customer can make changes to their accounts, outside of your program.

Your program will allow users to interact with their accounts by:

- viewing their account balance(s)
- transfer money between accounts that they own
- withdraw money from an account (This will decrease their balance.)
- transfer money from their account to another user's account (This will also decrease their balance.)
- pay a bill by transferring money out to a non-user's account (This can be stored in an **outgoing.txt** file that is outside of the program. It also decreases their balance.)
- deposit money into their account by entering a cheque or cash into the machine (This will be simulated by individual lines in an input file called **deposits.txt**. You can decide the format of the file. This will increase their balance.)
- requesting the creation of an account from the bank manager

Each user can have more than one account. But no account can be co-owned by multiple users. Instead of using a card and numeric passcode to interact with their account, the user will have to use a login and password. Only a bank manager can create and set the initial password for a user. But the user can change their password, later.

Accounts

There are two main types of accounts: Debt and Asset. Debt accounts include:

1. Credit Cards Accounts

These are accounts that display a positive balance when the user owes money and a negative balance when the user overpays. It is not possible to transfer money out of a credit card account. But it **is** possible to transfer money in.

2. Line of Credit Accounts

A line of credit account allows you to transfer money in or out. But it also displays a positive balance when the user owes money and a negative balance when the user overpays.

Asset accounts include:

1. Chequing Accounts

If a user has only one checking account, it will be the default destination for any deposits. If the user has more than one chequing account, one of the chequing accounts should be selected as a "primary" account. This will be the default destination for deposits. Chequing accounts store a positive balance when money is stored in the account. The balance is allowed to decrease to -$100 if the user withdraws money when there is a positive balance, but the withdrawal is more than the balance. A negative balance beyond -$100 will not be allowed. A withdrawal from a chequing account with a negative balance will not be allowed.

2. Savings Accounts

A savings account stores a $0 or positive balance at all times. Before closing, on the 1st of every month, all savings account balances increase by a factor of 0.1%. In other words, a savings account with $100 in it on January 31, will have a balance of $100.10 on February 1st.

Time

The bank manager should only have to set the date once. You can assume that the program is shut down and restarted every night at midnight. When the user asks to see their account balance, it should be up-to-date as of the previous day. If you can make it completely up-to-date, that is even better! (but optional). If you want to include time stamps, you can do this any way you want: through the input file(s), using the system clock, or any other way.

Handling Cash

The bank machine should know how many $5, $10, $20, and $50 bills it has. When the amount of any denomination goes below 20, your program should send an alert to a file called **alerts.txt** that the real-life manager would read and handle by restocking the machine. When a user requests a withdrawal, your program will have to decide which bills to give to the user and decrease the total of those denominations accordingly.

The Bank Manager

The bank manager is the only person who should be able to create a login and set the initial password for a user. Also, they can increase the number of $5, $10, $20, and/or $50 bills in the machine to simulate restocking the machine. Lastly, the manager has the ability to undo the most recent transaction on any asset or debt account, except for paying bills.

User Info

A user should have the option to see on the screen:

- a summary of all of their account balances
- the most recent transaction on any account
- the date of creation of one of their accounts
- their net total (The total of their debt account balances subtracted from the total of their asset account balances.)

System Boundaries

Input into your program should come from the keyboard, the **deposits.txt** file, and any configuration files that you create to store information such as the current date, or user login information. Your file should output to the **outgoing.txt** file, the **alerts.txt** file, those same configuration files, or any other files that you want to use to store information from your program.

What you hand in

DO NOT clone the ProjectGroups repository. We will create an assignment on MarkUs called **phase1**. Please clone the phase1 repository. All of your project files, packages, and uml diagrams should be inside the phase1 folder. Make sure each of your team members uses git to add, commit, and push their work frequently to minimize the number of conflicts. Be sure to **git pull** every time you sit down to work. This ensure that you are working on the most current version of your team's files.

At the end of phase1, your repository should include:

1. All of your code
2. Examples of any files we need, in order to run your project. For example, include **deposits.txt** with a few lines of text in it, so we can easily edit it to simulate depositing money.
3. A file called **README.txt** that contains enough instructions so that we can run your program and try out all of the functionality. This includes anything we must do to start up the program initially.
4. A uml diagram that demonstrates the inheritance relationships between the classes that you wrote and implementation of interfaces that you wrote (if any). It should also include the names of all methods, the types of all variables, and the accessibility modifiers for both. If your entire uml diagram is in one file, call it **design.pdf**. If you have split your diagram into multiple files, call them **design1.pdf**, **design2.pdf**, etc.

Teamwork, Git, Comments, and Tests

Use git frequently, with useful commit messages. Only commit code that compiles. Be kind and considerate of your teammates. If you find that your group dynamic is not working, feel free to consult a TA or your instructor. There are many ways to resolve common issues that arise without the TA or instructor "taking sides". It is important that the team figure out a way to work well together.

We will look at your git log and will consider everyone's contribution when deciding on your mark. We can also see the contents of each commit, so please do not cut-and-paste the same code back into a file without making changes or pretending that superficial changes are more important than they are.

It is helpful to the marker and also to your fellow team members to include proper Javadoc. If a part of your code is particularly difficult to understand, you can also leave a quick in-line comment to help the reader. Please do not include more comments than is helpful. If you do, it will take much longer to read through your code.

You are not required to write tests for your code. However, you may want to do that anyway. In particular, if there is a class that many people are working on at the same time, it may be helpful to decide on a set of minimal tests that the code must pass.

Preview of Phase 2

There will be some other features required in Phase 2 involving new types of accounts, allowing the user more options, and other types of bank workers who will have some of the same access as the bank manager, but not all of it.

You will also be asked to implement features of your choice. For example, your ATM could handle multiple currencies, investments such as buying/selling stocks, etc. Or you could design features that are not yet currently handled by banks, such as creating labels and paying for courier delivery service, handling in-game wealth/points for massive multiplayer online games, or other features that you think would be useful.

Your program has to run and satisfy the Phase 1 specification in order to get a passing grade. Do NOT add extra features until your Phase 1 works! However, you are welcome to design in a way that makes Phase 2 easier. **Most of your Phase 1 marks come from your use of encapsulation, the absence of code smells, and adherence to the SOLID principles of design. If it is easy to work with your code (read, understand, add new things), then you will get a good mark.** At the end of the semester, we will replace your Phase 1 mark with the maximum of your Phase 1 and Phase 2 marks.