Michael Wise
Professor Arias
CMPT 220 Software Development I
14 May 2020

ATM Machine - Final Project

Abstract

For this project, I have created a program that simulates a functioning ATM machine. Some of the classes I have implemented include ATM, Account, Bank, and ErrorHandling. There are some class hierarchies - for example, the Account class is a superclass of the Savings and Checkings class. The ATM has a simple to use GUI with login by user ID. The ATM simulates the withdrawal and depositing of money in an account. Keeps users protected by only letting you interact with your balance if you have the proper security information.

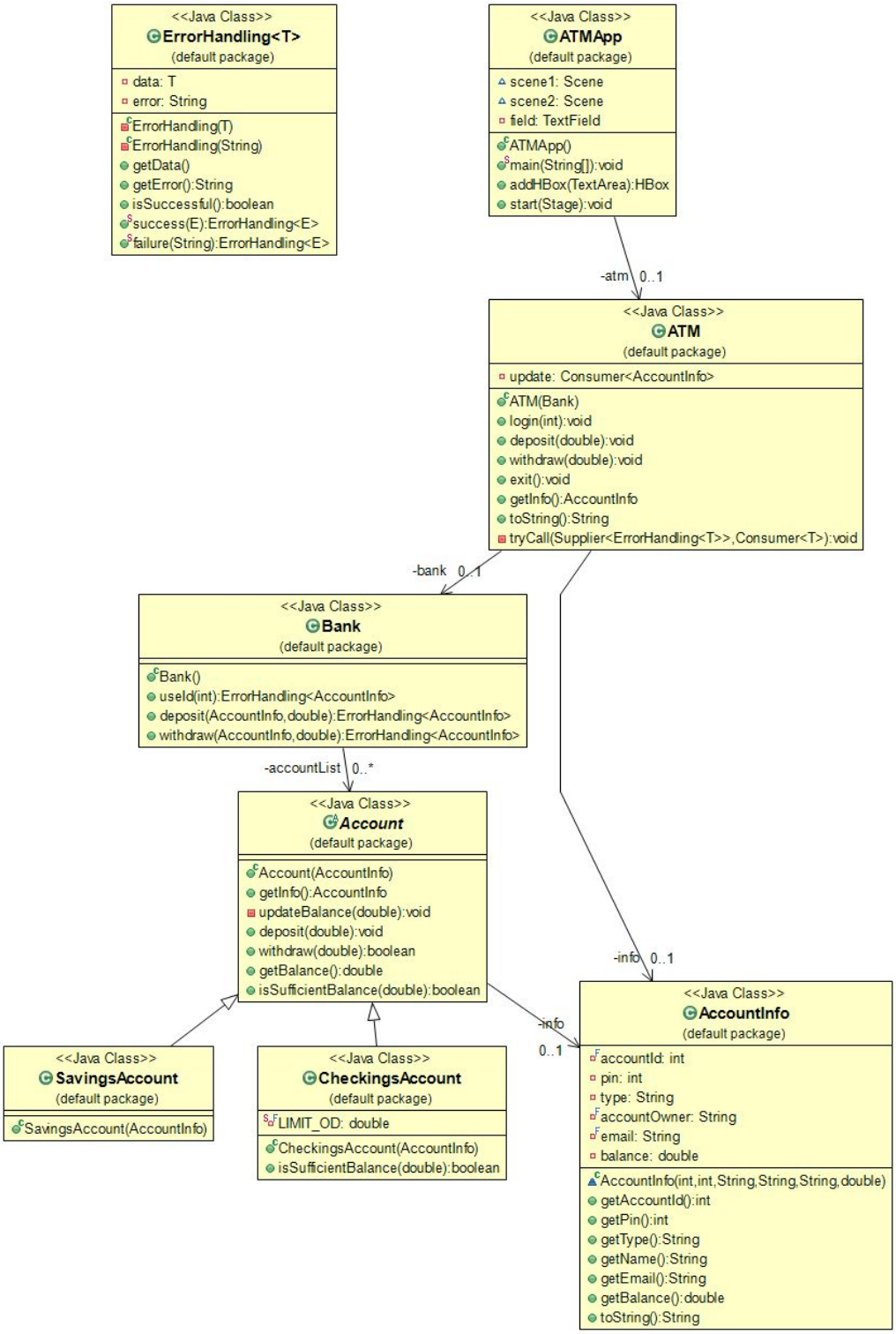
Introduction

The main reason I wanted to do this was to take a simple programming task and really improve my skills by making it as complex and efficient as possible. As a data science major, I really have no plans using Java in any future jobs, but I think it is crucially important to understand the software development process regardless as well as having a wide spectrum of programming knowledge in various languages. I wanted to learn how to use GUIs in Java along with learning how to effectively integrate classes into a big program. First, we will cover a description of all the classes and how they interact with one another. Then we will describe the problem that this system addresses, outline other similar work, and describe how to use the system.

Detailed System Description

The ATM will allow users to perform standard actions you would expect to get at an ATM. The main class, ATMApp, creates the GUI with JavaFX. There are two different scenes: the main login scene, and the actual ATM interface. The ATMApp class interacts with the ATM class when clicking buttons and entering information. The ATM class interacts with the Bank class that maps all of the user information to the correct account ID. The bank stores the information from the Account class, which is a superclass of the two types of account:

CheckingsAccount and SavingsAccount. They deal with user info and have different perks (i.e. checkings account allows you an overdraft fee). There is also ErrorHandling which handles exceptions with transactions. The UML diagram for all the classes is shown here:



Requirements

The system is addressing the problem of being able to transfer money from cash to electronic currency efficiently and securely. ATMs are designed for the purpose of allowing users to withdraw and deposit money in convenient locations without having to go directly through a bank or other service. They save a lot of hassle when you are in desperate need for physical cash (assuming that the location has an ATM, which is a totally separate problem). This simulates the software that someone would install into an ATM machine. The code is then able to tell the physical machinery to perform actions like deposit and withdraw the money. Of course, I don't own an ATM to install this onto, so this project focused on only the software side of it. It also addresses security problems like entering an ID and PIN number. The system does not let you access any information unless you have an existing account in the bank.

Literature Survey

There are hundreds of ATM manufacturers around the world, all having different features. In an era where digitalization is getting bigger and bigger, there have been features that allow users to not require cards or cash, such as Apple Pay. People can access/interact with their balances without the need of a physical credit/debit card or cash. We are already advancing towards a society where even ATM machines are becoming obsolete with software like Venmo and Paypal. Virtual currency is quickly becoming the new norm, with physical currency becoming more and more antiquated. Almost all ATMs nowadays run on a certain software with a GUI - it just makes it so much easier for users to make transactions with.

User Manual

The ATM features a GUI that will make transactions seem effortless. The ATM first takes your name and your account ID (5 digits long). If it doesn't detect a valid ID, it will not let you log in. (IDs that work by default are 12345, 24680, 12321, and 30001 just for the sake of simulation. Once logged in, the menu gives options to view balance, name, email, account type, and the ability to deposit and withdraw funds. It will be touch-based, so all you have to do is click on the option you want (rather than some machines that have numbered analog buttons you

press to choose an option). It also simulates your keyboard as a physical keypad (all you need to do is type in amounts). Once you have the amount you desire, click withdraw or deposit to accordingly transfer that amount. Withdrawals will not go through unless you have enough in the account balance, or own an account with overdraft protection.

Conclusion

The general goal of this system is to make the transaction of currency easy and accessible to anyone regardless of technological competency. The software encapsulates the functionality of the hardware parts of the physical ATM. There are multiple ways to access an account balance and the ATM effectively authenticates users by interacting with the bank's list of accounts (which I could implement further into a database if I had more time). It definitely was challenging at first trying to configure the GUI with JavaFX. However, once I finally started to get the hang of it, I was able to make a pretty decent looking interface. Getting the buttons and amounts to work required a lot of use of the setOnAction() method. It seemed to be super tedious and repetitive. Then I discovered you can make a class that handles errors and actions with the combination of Java lambda expressions introduced in Java 8. Now I could express instances of classes much more compactly, and that saved a lot of lines of code. Honestly, it was hard learning how to incorporate multiple classes and using them with a GUI that has exception handling. But now that I know that it is something I can do. While the ATM is relatively simple, it still has taught me that I have the ability to do bigger, greater projects in the future, even if I don't necessarily plan to use Java. Overall, this class and this project did a great job at really exploring the ups and downs of the software development process. It is not easy to do this stuff - it takes quite a lot of time and effort. With that, I hope to take these skills I learned and apply them to future classes at Marist and future jobs in the workplace.

References/Bibliography

JavaFX Tutorial

- https://www.tutorialspoint.com/javafx/index.htm

UML created with ObjectAid UML Explorer

- https://www.objectaid.com/home

Exception Handling Interface based on:

- https://docs.oracle.com/middleware/12212/lcm/LCMAJ/oracle/fmwplatform/actionframework/api/v2/ActionResult.html

JavaFX Tooling and Runtime for Eclipse and OSGi:

- https://www.eclipse.org/efxclipse/install.html

Thank you Professor Arias for a great semester!