AUTOMATED TELLER MACHINE (ATM)

OSCAR VILLASANA

BENJAMIN WODHAMS

RYAN WOLFE

CONTENTS

PROBLEM STATEMENT

ATM's are very important to today's way of living because they allow people to retrieve money they have in their bank account without the need of a teller. This greatly reduces the amount of time it takes to get to get money. ATM's are located all over modern cities making it the fastest way to get money without the need of going to the bank.

Our project was simulating an Automated Teller Machine (ATM). The ATM will include accounts of users, a feature to create accounts, and the amount of money it holds. Each User has at least one bank account, a name, username, password, address, age, birthday and a user ID. There were 2 types of accounts: Checking account and savings account. We decided to go with these two types of bank accounts because they are the most common. Each account has a balance and a pin number for security. Each account will have its unique features. An account will have an account name, withdraw amount and a deposit amount. These features depend on the type of account. For example, a checking account will have a maximum daily withdraw amount of 300 dollars where a Business account will have no limit. Savings and credit accounts are typically monitored so you don't withdraw cash and we will take that into account. Some users will also be considered VIP which allow them to withdraw and deposit a larger amount of money than a non-VIP member.

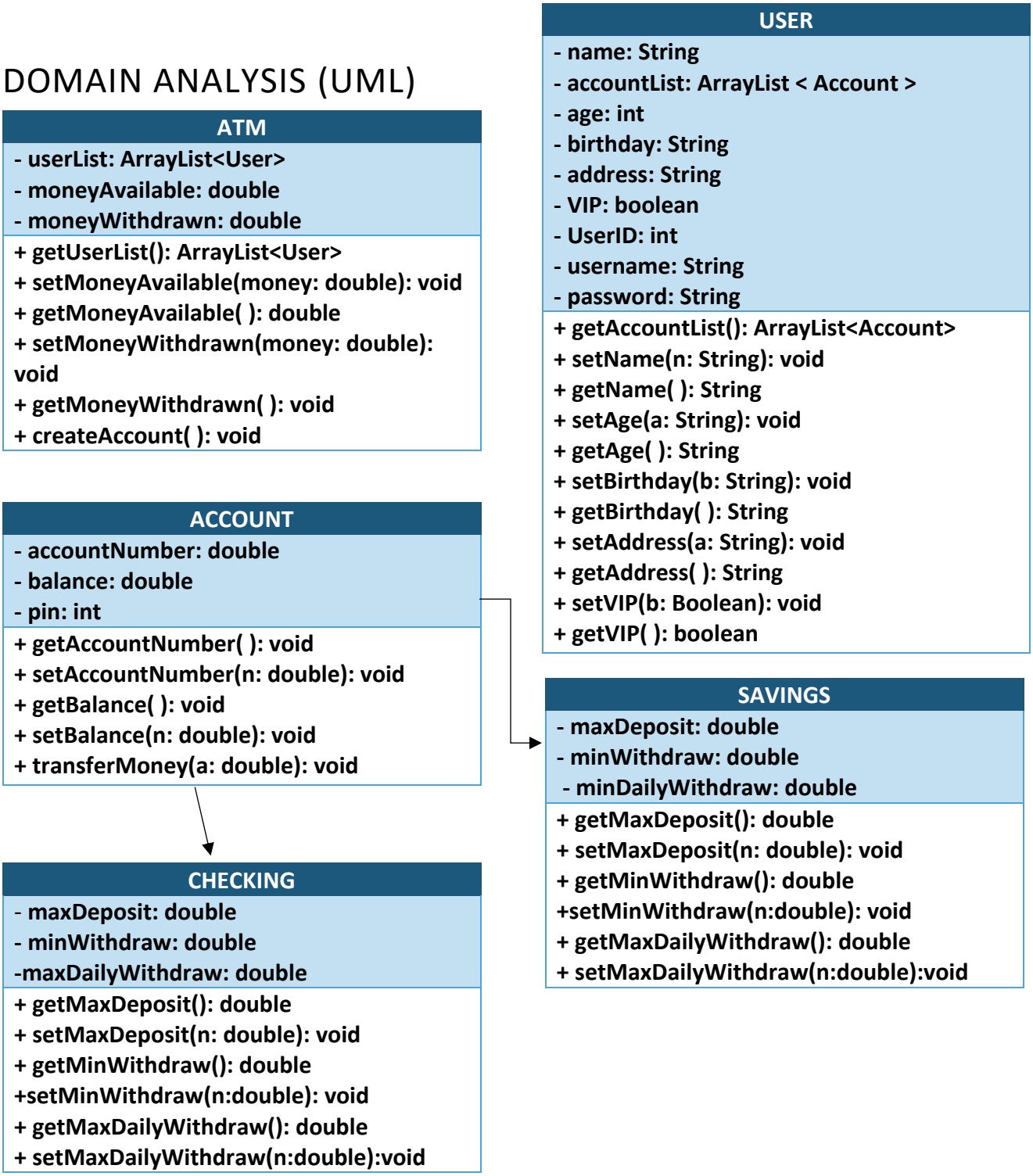
The users are able to login to their account or create and account if they are a new user. Once they have done this, they can see their current accounts and they can deposit money or withdraw money. If a new user doesn't have any accounts, a new GUI window will popup

where they will enter their information, and they will then be able to login. We will introduce serialization in order to keep each user's information saved even if the application is closed.

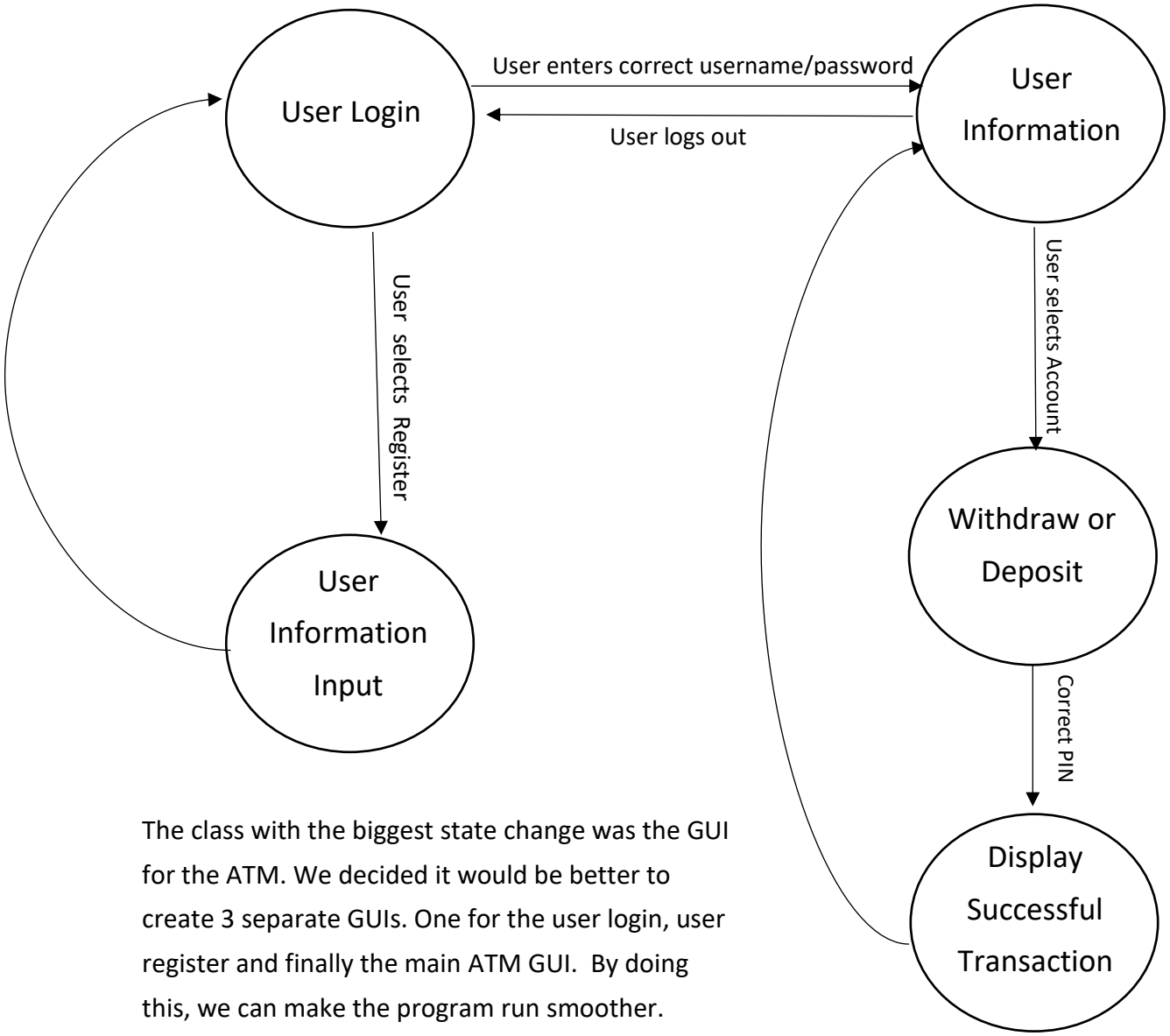
SYSTEM BOUNDARY

The ATM simulation has features for everyone. If there is a new user, the new user will enter their information such as: name, username, password, address, age, phone number and birthday. The ATM will check to see if another user does not have the same username. We excluded password strength as a real ATM/Bank would require you to do as it was a much more difficult thing to do. Once a valid username is chosen, a new account has been created and the new user is ready to login. Once a user logs into their account, they will see their information as well as their bank accounts with their balances. The user will be able to withdraw of deposit money from their account once they enter the correct account pin. We added a security feature that when the wrong pin is entered 4 times, the system shuts down to prevent unauthorized users to access the money. Our main focus was security and that's why we required two layers of protection. The first one was logging in with a username and password. Once they were in, the second security feature asked for the account pin to handle the money. We excluded issues of the ATM being down. We also excluded the issue of only being able to withdraw factors of 20 just to make the program more interactable. In a real ATM, you may only withdraw an amount that is divisible by 20 as the ATM only has 20 dollar bills. We also decided to remove the transfer money to a different user as it would require the user sender to know the user receiver's information. We also decided to create 3 separate GUI in order to maintain the program running soother so there wouldn't be many popups for all the different functions. This makes the program run smoother.

DOMAIN ANALYSIS (UML)



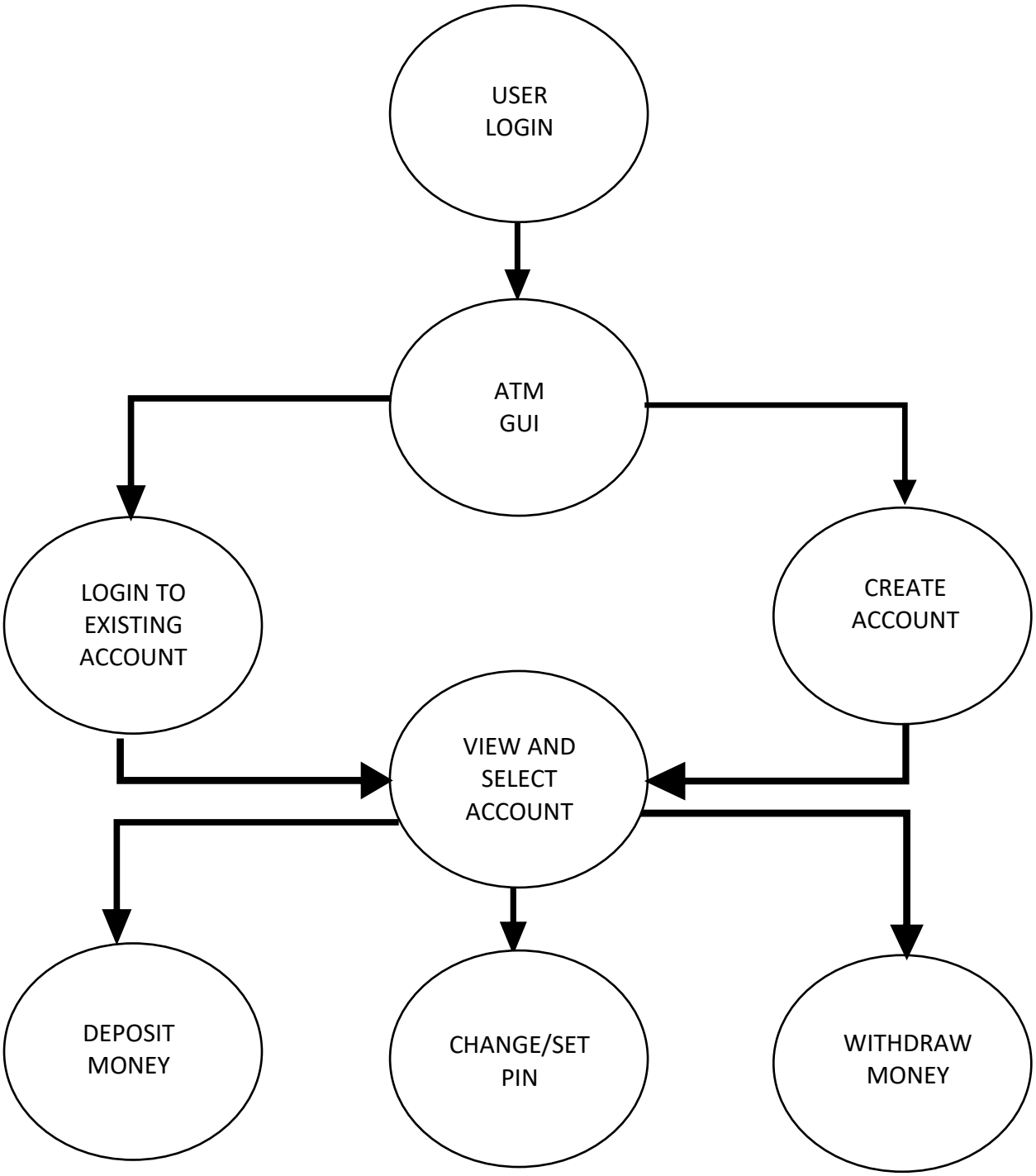
STATE MODEL



The class with the biggest state change was the GUI for the ATM. We decided it would be better to create 3 separate GUIs. One for the user login, user register and finally the main ATM GUI. By doing this, we can make the program run smoother.

Since our program demanded many functions to operate, it would have been somewhat uncomfortable for the user to interact with multiple pop up frames popping out for everything the user did. We created a GUI so the user can login and when they do sign in, there is a separate GUI that handles all the transactions the user desires.

INTERACTION ANALYSIS



IMPLEMENTATION

When writing the code for the ATM machine, there actually was not too many issues that the group ran into. However, the group did decide to cut out a feature with the ATM that we foresaw as being too difficult to implement with the time we had for the project. Originally, each User was intended to have an ArrayList of various accounts that they could have opened with the bank. This would have posed some major difficulties in creating the GUI because the GUI displays all of the User's accounts in a list of checkboxes. The group wasn't sure how to be able to implement the ArrayList of accounts so that the checkbox in the GUI would continuously update and expand or shrink number of checkboxes as the User created or closed accounts with the bank. Furthermore, the group found it troublesome to find a way such that when a new User is initialized, they would have uncreated accounts that should not be displayed in the GUI. To circumvent these foreseen issues, the group decided to design the ATM so that every User simply had two accounts, a checking and saving account. This was found to be optimal because one, most people only have a single checking and/or saving account with a single bank, and two, the group did not have to worry about the process of account creating within the GUI.

One of the aspects of the ATM project that was difficult to implement was having an actual keypad that the User could press to enter their pin numbers. Having an actual keypad appear instead of a text field proved to have a much more polished look for the ATM system as well as being more similar to what a real person might see when they are using an ATM. The difficulty with the keypad was figuring out how to design a JPanel with all of the buttons that looked like a real keypad as figuring out a way to have the keypad appear and accept inputs when the User attempted to deposit money, withdraw money, and set their pins. What ended up working was creating 10 buttons for digits 0-9, and when each button is pressed, the button's value gets added to a string. When it comes time to check the pin, the string that was being populated gets parsed for an integer. Problems arose when testing the keypad because a button listener was created for the keypad, and this listener would end up being initialized more than once during a method call, so that each time a user would hit the keypad, the listener would populate the string multiple times. In order to fix this, an if statement was made

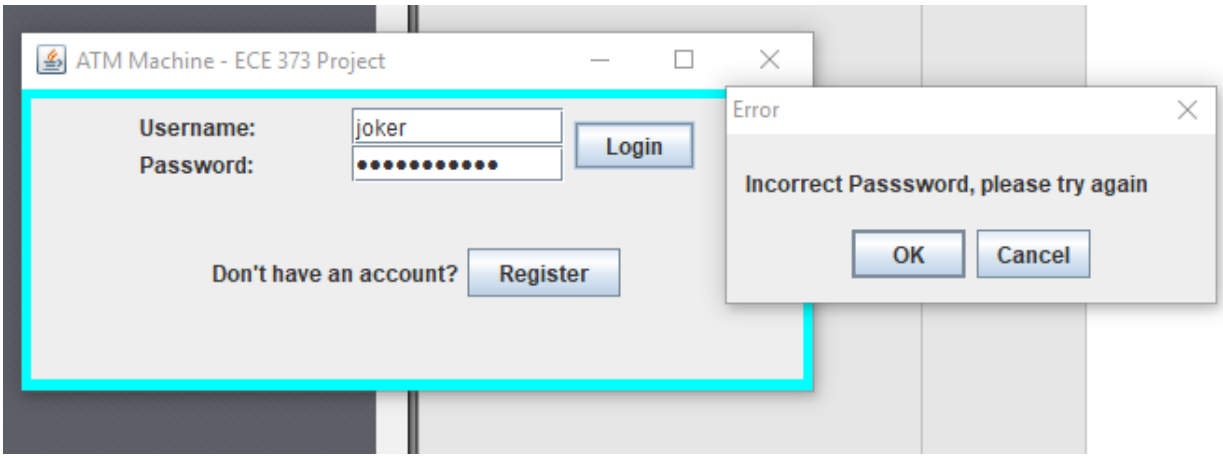
to check if the length of the keypad's button listeners was greater than one, and if it was, to not initialize the listener.

TESTING & RESULTS

To test the ATM system, a driver was created which initialized a number of users and their accounts. Integrated testing was then used to test the ATM's features one at a time. For example, the first feature that was tested was the login screen. The group tested the ATM's login feature to detect any possible bugs and ensure that the login and register features were working as intended.

Example of login testing: Incorrect password

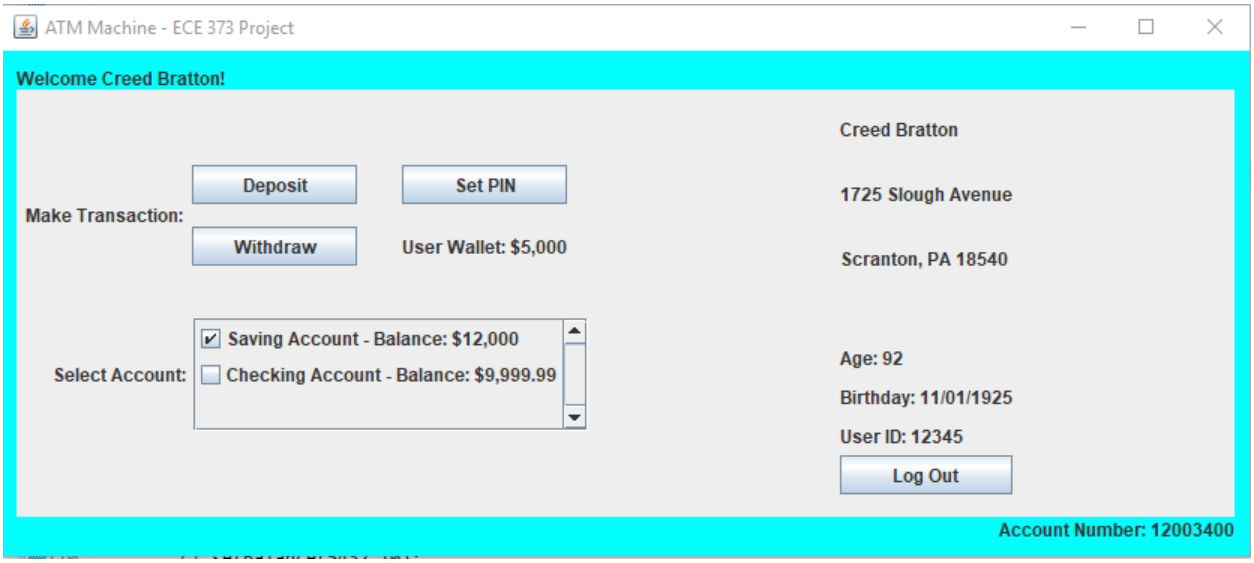
```
User u7 = new User();
u7.setName("Creed Bratton");
u7.setAge(92);
u7.setBirthday("11/01/1925");
u7.setAddress("1725 Slough Avenue");
u7.setAddress2("Scranton, PA 18540");
u7.setVIP(true);
u7.setUsername("joker");
u7.setPassword("1");
u7.setPhone("(800)964-DMPC");
u7.setUserID(12345);
```



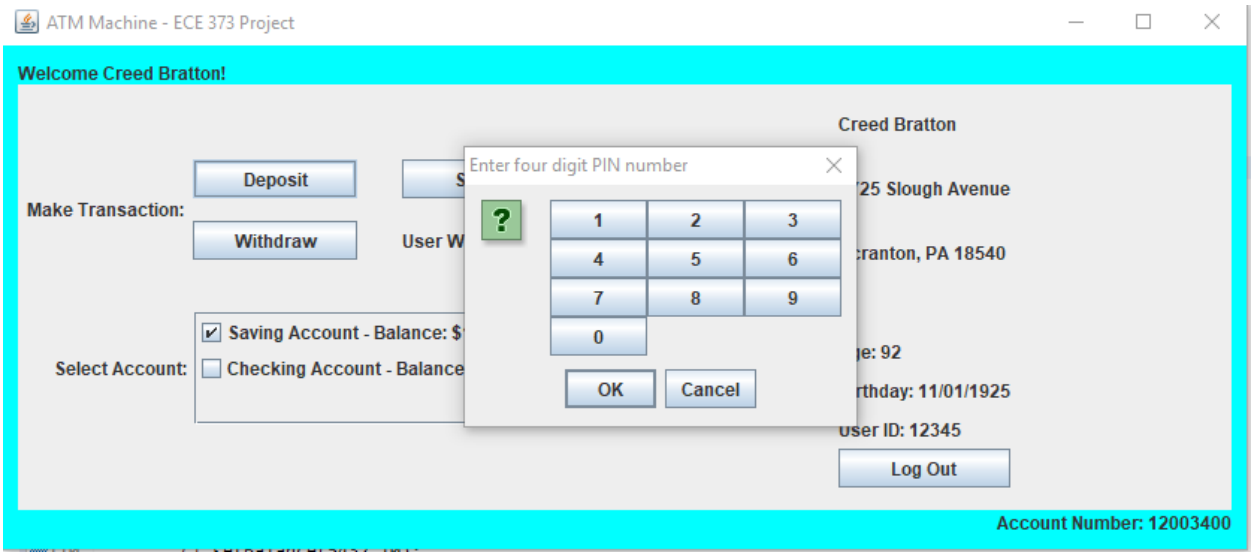
An error appears when entering an incorrect password

After vigorously testing the logins and accounts, the group tested all of the different interactions that were expected with withdrawing and depositing money as well as a user changing their pin number.

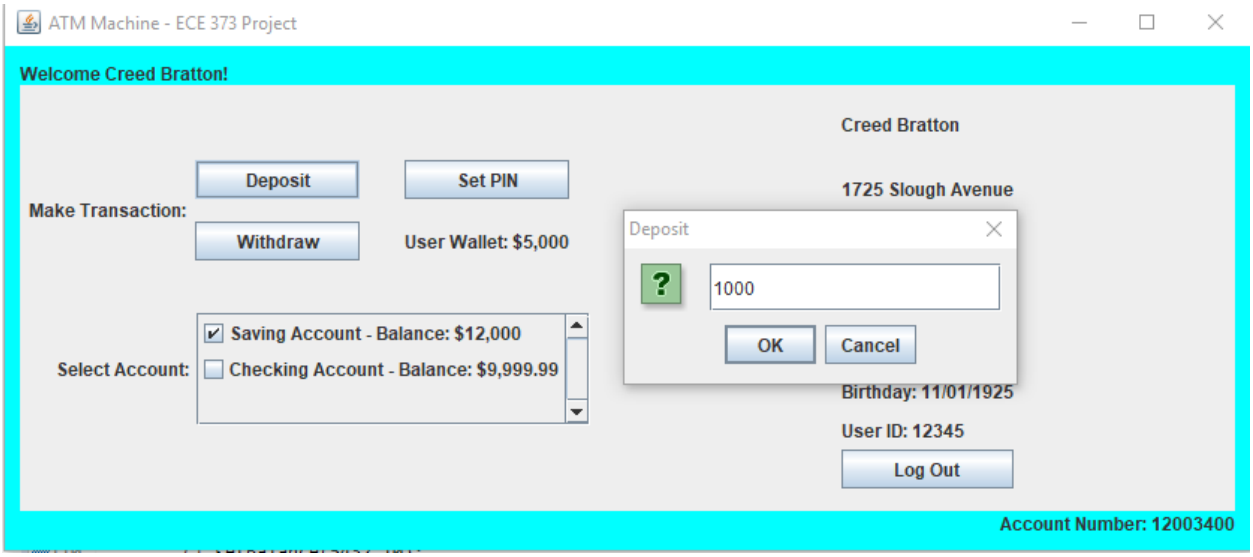
Example of transaction testing: depositing money



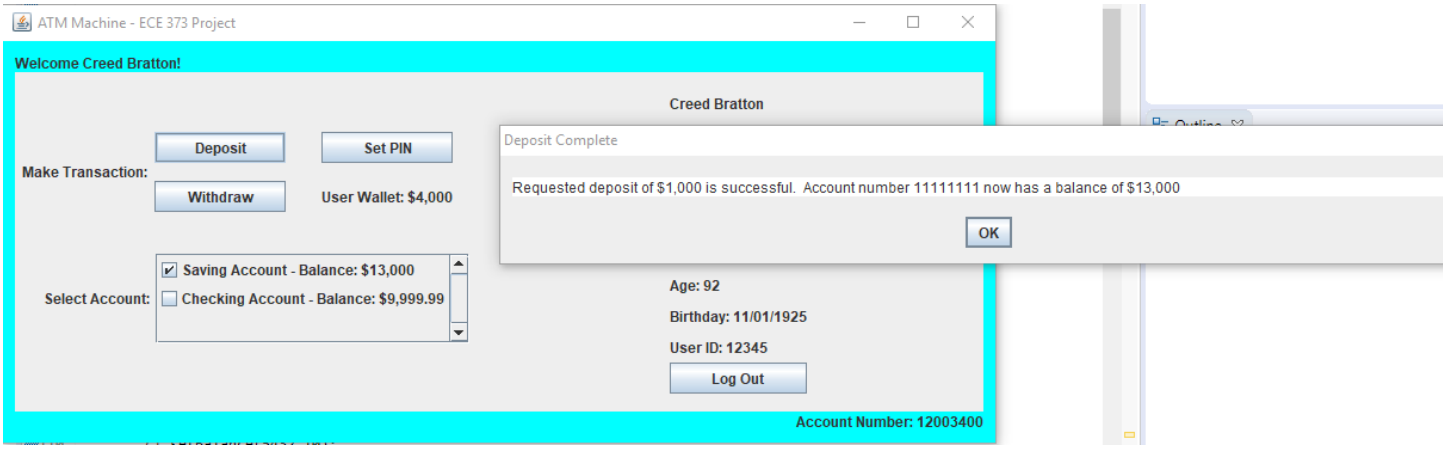
User first must select an account



User must then enter their correct PIN number

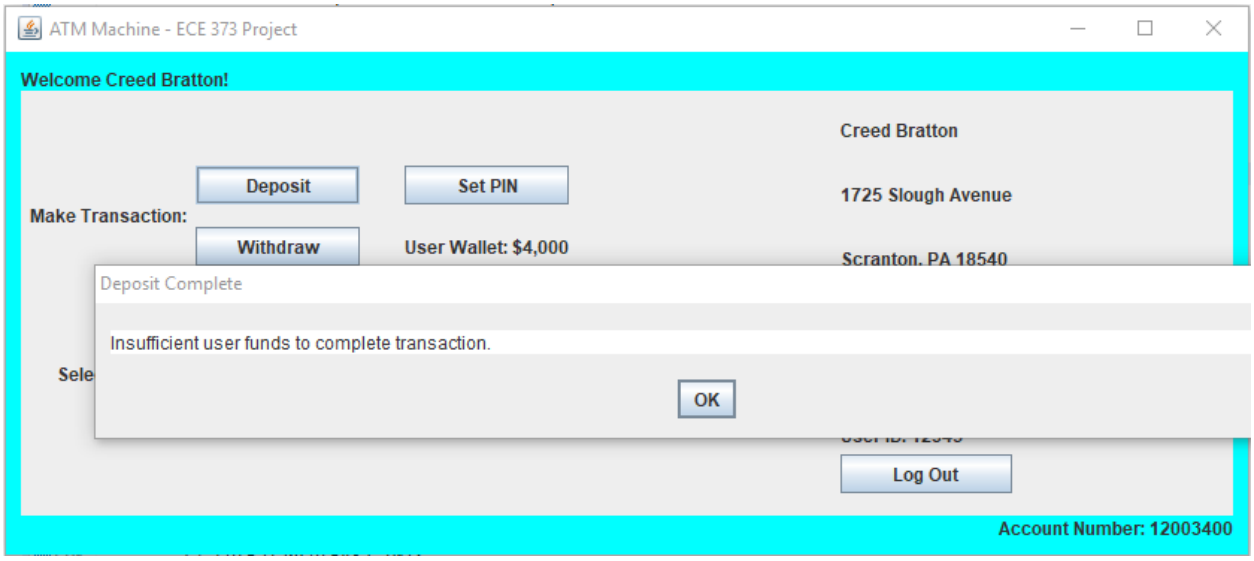


User must then enter a value to deposit that is possible based on their wallet and deposit limits



If these steps are successful, the deposit goes through and the user's wallet and account balance are updated accordingly.

For testing the transactions, the group went through every possible case that a user might try to ensure that no unknown outcomes would occur. For example, the group tested to make sure the user could not make a transaction if they enter an incorrect pin, and if they enter an incorrect pin four consecutive times, the ATM window will exit. The group also tested to make sure that an account was selected before making any transaction or setting their pins. The group would then test what would happen when the user attempted to deposit or withdraw a non-integer value or values that are too high or too low.



When the user attempts to deposit more money than what they have in their wallet.

EXTENSIBILITY

As stated in the implementation section, the ATM system could be expanded by having additional account type classes as well has each user possessing an ArrayList of accounts instead of just a checking and saving accounts. This would have required of course creating new classes additional account types, but also a lot of work with the GUI to create a way for the User to create new accounts, and these new accounts then appearing in the group of checkboxes of accounts in the GUI.

Another potential extension that could possibly be made to the ATM system is having family accounts. This would require additional classes for the different subtypes of User: adults, kids, parents, etc. This could lead to some cool interactions such as a child user having limited capabilities with the ATM, and parent users being able to see and interact with their children's accounts.

A third extension to the ATM system that the group flirted with was making the ATM transactions only able to deposit and withdraw US dollar amounts, i.e: 1, 5, 10, 20, 50, 100. This would not have required additional classes, but simply some additional if-else statements in the methods that handle depositing and withdrawing money. This was however decided against because the ATM behaves more like an online bank where users can transfer any amount of money.