Name: Anthony Jasso
Name: Derek Aguirre
Date: 11/9/2020
Instructor Daniel Mejia
Assignment – Programming Assignment 4

I confirm that the work of this assignment is completely our own. By turning in this assignment, I declare that I did not receive unauthorized assistance. Moreover, all deliverables including, but not limited to the source code, lab report and output files were written and produced by my partner and I, alone.

1. **Program Explanation**

This assignment tested both of our code-reading skills, we had to review each other's code and find ways to implement their methods/classes into our own. Initially, we reviewed the functionality of their own methods and implementation strategy to see if we could merge it properly with our own. We both compared our approaches to solving the previous lab's requirements. In the process of reviewing the other person's code, we were able to get a better understanding which approach of ours worked better for solving the previous lab's tasks. After reviewing each other's code, we started creating a list on a text file of the names of all the classes and methods that both of us implemented. We were able to compare methods and classes in an easier format and it allowed us to easily record which methods or classes we finished reviewing/merging. The listing of all of the contents of our code helped expedite the process of code merging and was one of the key factors that contributed to us finishing this assignment in a timely manner. Most of the beginning of this assignment consisted of reviewing our classes side by side and taking the best of both of our approaches to combine them and in most cases the result was a much more elegant approach to solving the assignment. The most time consuming part of the assignment was the code merging so we prioritized creating one coherent program with the merged code before working on any new implementation.

2. **What did I learn?**

There are different approaches to solving a single task. For example, when implementing a UI for the bank, we decided to go with a do-while loop because every menu prompt is guaranteed to run at least once. The handling of certain accounts that do not exist within the customer object could have been improved. Instead we gave default values, in this case, -1 to the attributes and checked if the accounts matched the values. We could have used different constructors that handle the variety of possible customer accounts instead of initializing a 'non-existent' account with an account number of -1. This approach would save on space when creating an account object because the account that is not requested will not be created and saved. The lab assignment took approximately 28 hours of total collaboration time to complete.

3. **Solution Design**

The design pattern that was used to solve this problem was the singleton design pattern. We created a singleton class called logger that holds all of the log functions for each transaction. We used ArrayLists as our data structure to store the bank customer's data because its dynamic in nature and its elements can easily be accessed. We assumed that the customer knew their ID when logging into the bank user functions.

4. **Testing**

We tested our program by creating a list of possible errors that the user might encounter. We went down the list and made sure that our code could handle all of the potential errors. We used both black-box and white-box testing to test our code. Black-box testing was used mainly when dealing with UI errors, while white-box testing was used with object errors. We tested our program thoroughly and after every modification of the code, but it could have been improved by utilizing the JUnit testing in the IDE.
The cases that we used was the following:

- Input Mismatch
- Negative number inputs for transactions
- Accessing an account that doesn't exist
- Paying a user from an account that doesn't exist
- Generating bank statement for new user

Yes, there were multiple instances where the program would break and fixing it would result in not only solving the issue but free up space within the code.

5. **Test results**

Most of the input mismatch and negative number test cases passed because Derek handled most of them in his PA3 code.

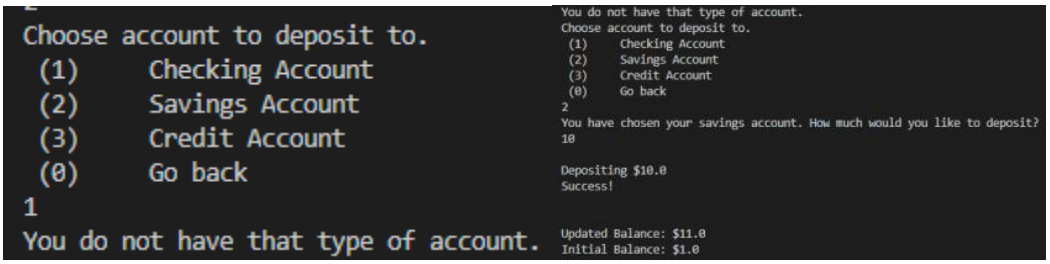
```
Welcome to the Bank of the Miners!
Which type of user are you?
(1) Customer
(2) Bank Manager
(3) Transaction Reader
(0) Exit
a
Please enter an integer.
```

```
Welcome to the Bank of the Miners!
Which type of user are you?
(1) Customer
(2) Bank Manager
(3) Transaction Reader
(0) Exit
4
Please choose a value inside the range.
```

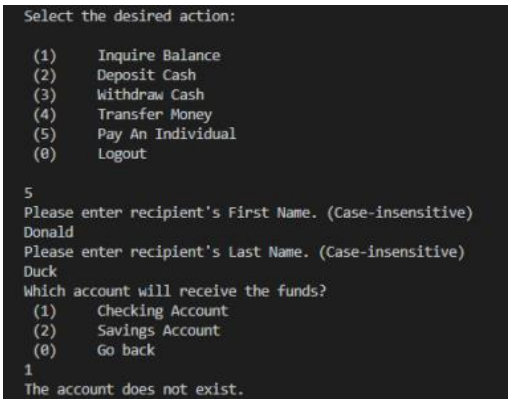
Input Mismatch

```
Choose account to deposit to.
 (1)     Checking Account
 (2)     Savings Account
 (3)     Credit Account
 (0)     Go back
2
You have chosen your savings account. How much would you like to deposit?
-10
Cannot deposit a negative amount.
```

Negative number inputs for transactions



Accessing an account that doesn't exist



Paying a user from an account that doesn't exist

6. Code Review

Person One (Anthony Jasso)

My partner's code was very thorough and long at times, but overall well organized. Their implementation of the bank interface was very clean, appealing to the user's eye and easy to navigate through. Their implementation of the transaction reader could have been shortened. The transaction reader was using nested if statements and nested loops that could have posed an issue to the time complexity of the program. Keeping all of the user's accounts in the customer object is better than having separate ArrayLists for each account type.

Person Two (Derek Aguirre)

I appreciated that there were less files to have to manage, but it meant that his RunBank was difficult to read through sometimes since most of the functionality of the code was relegated to that class. I liked the implementation of the transaction reader. His approach solved the same problem in a fraction of the lines that mine took up. I also liked how he implemented the deposit, withdraw, and transfer functions. I did not agree with his implementation for handling how accounts are saved. All accounts and customers were saved in separate ArrayLists of their respective typings in his RunBank class. They are then all used in a constructor for RunBank which takes all of the ArrayLists. I learned that we tend to be biased with our version of solving problems. Both of us learned how to make some compromises with the implementation and favor the other person's implementation over their own.

7. **Reflection**

We started with comparing our code side by side and reviewing which one worked better given the situation. After we determined which one solved the problem better, we started splitting the methods and combining the ones that we chose. The majority of our code was readable and easy to understand so we did not have a difficult time understanding the code. However, there were some extensive methods that required a line by line review such as the transaction reader. Methods like those took a lot more time to merge because they were more complex and had a lot more variables and functions involved in making them work. We ran into a lot of issues trying to fix the constructors for the account classes and it caused issues with user creation. We were not able to create a user without an account due to these issues, but we still managed to find a workaround that involved setting an account number to -1 if they did not exist.

8. **Demo of another team**

Estevan Ramos demoed us with his PA4. We understood the process of creating a new user from the manager menu that he implemented. He did not provide us with the javadoc however. Yes, we broke their code by creating a new user and trying to login to their account. We also broke the code when trying to write a bank statement to a new user. They did meet the requirements for the current tasks for PA4 which included the interface implementation and the user login with passwords. Even though their group did meet the requirements for the current programming assignment, they did not meet the requirements for the previous programming assignment where a manager creates a bank statement for a new customer. Their functionality for the previous programming assignments did not hold up when checking for exceptions.

9. **Demo for another team**

We demoed with Estevan Ramos, his partner was not able to attend the demo session so we do not know who his partner is. When we ran our code for them to test, we did not provide much extra information regarding the menu itself. The menu design was intuitive enough to be understood without prior knowledge. The choices were a number selection from 1 to however much selections were needed for any given prompt. The selections were titled and did not give Estevan any trouble when navigating through them. The only user-input intensive part of the code was entering information for a new customer. There are 6 steps to creating a new user, however, each step specified the format it preferred to have the data in. Javadoc was not provided to Estevan because we were not sure if it was considered code sharing since it shows what methods are in which classes. Estevan did not break our code but testing their code and listening to him explain the reasoning helped us think of ways on how to ensure the robustness of our code for the future programs. We did meet all functionality requirements of this programming assignment along with the requirements of all the previous ones.

Person One (Anthony Jasso)

I contributed the object classes and the printable interface. The object classes were used to store and modify the customer's data and the printable interface was used to implement some methods that were in the menu and logger files. Helped with the bug fixing portion of merging eachother's code. I was tasked with bug testing methods that gave us a hard time and creating an interface class that required three methods. There would be times where we would swap roles but

for the most part I was bug testing. There are different ways to approach a problem and simpler ways to solve a problem, all it takes to see this is from another person's perspective.

Person Two (Derek Aguirre)

I provided the user interface methods and the general choice making process behind the program. The user interface was what facilitated the usage of the system by multiple users, the program's robustness was built upon the foundation of the user interface. I also provided the logger and the user created exceptions. I helped Anthony integrate his methods into the user interface and made sure to help refactor his classes' constructors to integrate better into the system. Did equal work as him, I was tasked with putting together the methods and he would bug test them and fix them if they needed to be fixed. Sometimes we would switch roles in that regard but our work dynamic did not change too much. Most bugs he fixed would have taken me a lot longer to solve even if they were from my own code. Through working with a partner, I learned that I can be biased to my problem solving approaches and fail to see other options.