Derek Aguirre – Person One
Anthony Jasso – Person Two
November 23, 2020
Daniel Mejia
Programming Assignment 5

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**

In programming assignment #5, we were given the task of creating a UML class, use-case, state transition, and use-case scenario diagrams from our code. We approached this problem by splitting up the work between the two of us and looking at each other's work and making revisions if necessary. We looked at our class notes on how to create the diagrams and integrated whiteboarding into planning sessions. Listed current and past bugs and assigned priority to them. Our assignment consisted of around five phases. The first phase of our assignment was reviewing the code to get a better understanding of how to create the diagrams. The second phase consisted of diagramming the code, we had one person lead the diagramming and the other person would provide information. After the first round of diagramming was completed, we sought to refactor our code in the third phase of our assignment to eliminate unnecessary associations between classes and clean up parameters. We repeated refactoring and diagramming until we were satisfied with the results. The fourth phase consisted of demoing where we took feedback and immediately started phase five which was our final round of refactoring and code touch-ups.

2. **What did I learn?**

We learned how to deal with another layer of input mismatch exception handling inside loops. We also learned more about how to create a correct use case diagram. We were originally mixing up the use case diagram with a type of state transition diagram until we fixed our diagram. The first thing we can improve would be how we check for customers during the login process. We currently prompt the customers to log in with their ID number instead of their first and last name. We went with this approach because ID numbers the chances of two people having the same name in the system are very likely. Ideally, we could implement the name and the ID number instead of just either one by themselves. During our demo with Minh's team, we noticed that they have password saving functionality for their customers, the customers can easily log in again when providing their name. We finished most of the coding early on so most of the time was spent on diagramming, we spent around 18 hours in total working on the program and making sure that the program's functionality matched the diagrams.

3. **Solution Design**

We often switched roles between refactoring and adding new code. One person would lead the refactoring and the other would watch while looking for errors in the code or logic. We also handed off the code to each other for fresh perspectives if one of us got stuck on a specific

part. We used ArrayLists as our data structure to store the bank customer's data because it is dynamic in nature and its elements can easily be accessed.
We assumed that the customer knew their ID and password when trying to access their account during the login phase of the program.

4. **Testing**

We tested our program by listing all of the 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, but it could have been improved by utilizing the JUnit testing in the IDE. The cases that we used was the following:

- Input Mismatch on customer menu
- Long doubles
- Out of range on menu selections

Yes, there were many occasions 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**

```
Welcome Mickey Mouse!
Select the desired action:

 (1)     Inquire Balance
 (2)     Deposit Cash
 (3)     Withdraw Cash
 (4)     Transfer Money
 (5)     Pay An Individual
 (0)     Logout

octopus
Please enter an integer selection.
Welcome Mickey Mouse!
Select the desired action:

 (1)     Inquire Balance
 (2)     Deposit Cash
 (3)     Withdraw Cash
 (4)     Transfer Money
 (5)     Pay An Individual
 (0)     Logout
```

Input Mismatch on customer menu

```
Choose account to deposit to.
 (1)     Checking Account
 (2)     Savings Account
 (3)     Credit Account
 (0)     Go back
1
You have chosen your checking account. How much would you like to deposit?
31.378283

Depositing $31.378283 into Checking Account (1000)
Success!


Updated Balance: $2885.55
Initial Balance: $2854.17
```

Long doubles

```
 Select the desired action:

  (1)     Inquire Balance
  (2)     Deposit Cash
  (3)     Withdraw Cash
  (4)     Transfer Money
  (5)     Pay An Individual
  (0)     Logout

 20
 Please choose between the range of selections.
 Welcome Mickey Mouse!
 Select the desired action:

  (1)     Inquire Balance
  (2)     Deposit Cash
  (3)     Withdraw Cash
  (4)     Transfer Money
  (5)     Pay An Individual
  (0)     Logout
```

```
Welcome Mickey Mouse!
Select the desired action:

 (1)     Inquire Balance
 (2)     Deposit Cash
 (3)     Withdraw Cash
 (4)     Transfer Money
 (5)     Pay An Individual
 (0)     Logout

-43
Please choose between the range of selections.
Welcome Mickey Mouse!
Select the desired action:

 (1)     Inquire Balance
 (2)     Deposit Cash
 (3)     Withdraw Cash
 (4)     Transfer Money
 (5)     Pay An Individual
 (0)     Logout
```

Out of Range on menu selections

6. **Code Review**

Person One (Derek Aguirre)

I appreciated that there were fewer 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 his Account class and Checking, Savings, and Credit subclasses so we ended up favoring those classes instead of mine. His transaction reader approach also 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 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.

Person Two (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 and appealing to the user's eye. 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.

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**

We demod Minh Dang's team, however, only Minh was present from his team during the demo time. We understood their process for performing tasks. The terminal interface for the customer choices was fairly intuitive and was easy to follow. The customer would login and choose which account they want to make transactions on then the program would list the functions of the given account. They did not provide the Javadoc for their program during the demo. We did not find a way to break their code, however, we noticed that their code would log the customer out if they performed an action regardless of the outcome. It was not an intended functionality of the program and their team was planning on fixing it at the time. Other than the issue previously mentioned, they did meet all of the base functionality requirements.

9. **Demo for another team**

Minh Dang's team demod with us, as previously mentioned, he was the only attendee from his team during the demo.We did provide them with enough information for them to use the program. Even though we provided information about the program, the program itself already provided most of the information when prompting for an action.We did not provide them with a copy of the Javadoc because the demo mainly consisted of checking for functionality. They did

not break our code, but they suggested that we format decimals to only show two decimal places. We did meet all the functionality requirements that were assigned at the time of the demo.

Person One (Derek Aguirre)

I led the State Transition Diagram and the code refactoring for this assignment. I integrated whiteboarding into our problem solving when working with loops to easier illustrate how to trace and follow the logic of the methods that contain the customer choices. We worked equally on most parts, I was tasked with making the changes to the code and sharing my screen to my partner for further review. I learned that when I can't solve a problem I need to let my partner step in and offer another perspective on how to solve it, even if the approach is completely different than what I came up with.

Person Two (Anthony Jasso)

I was tasked with doing the use-case diagram part of the programming assignment and also gave a hand in cleaning up the code from the previous programming assignment. I incorporated all of the program's functionalities onto the use-case diagram to show the process of each function the user does within the bank. I would complete a portion of the diagram then ask for feedback from my partner and adjust it accordingly. I learned that having another perspective on things is important and can have the ability to solve the same programming problems in a different manner.