Name: Derek Aguirre

Date: 9/19/2020

CS 3331 – Advanced Object-Oriented Programming – Fall 2020

Instructor: Daniel Mejia

Assignment: Programming Assignment 3

I confirm that the work of this assignment is completely my 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 me alone.

1. Program Explanation

In this programming assignment the goal is to expand on the first couple of programming assignments and add more functionality to the program. This time around, it was more focused on retaining an automated state so that the program can be used for long periods of time without breaking. This assignment tested my skills in many areas of programming, I was tasked with manipulating objects and classes, implementing a design pattern, and performing functions by reading an input file.   The most time consuming part of the assignment was refactoring my code to make sure that it was more robust, and the logic was easier to build upon if such a situation arose in the future. The part I had a difficult time with for the last assignment was figuring out the logic behind how my classes are supposed to interact has gotten more manageable. However, after learning that my logic for the user choices was flawed and very delicate, I decided to refactor all of the logic behind the exception handling and decision-making that users can make.

This program most likely would not have been completed on time if I did not work on it in pieces. The instructions, while simple and easy to understand, required a lot of care and effort because they were such complex tasks. I often had to break up the sub problems into smaller sub problems. The coding itself was not too bad, however, the debugging really took a lot of time from my assignment's completion.

2. What did I learn?

I learned a lot just from this assignment alone, I previously said that I learned a lot from PA2 but this assignment really tested my logic rather than my programming skills. It was very mentally draining to have to constantly reevaluate my own logic every time there was a mistake. I learned what common mistakes I make, especially when it comes to loops, as well was how to identify them better and rectify them much more efficiently. I feel that I learned a great deal about exception handling too while working on this. I still seek to improve on the aforementioned points in the future because I personally feel that those are at the core of any programming project I have created and will create in the future.

Solution Design

The first thing I did when I was given this assignment was refactor my code and fix a lot of the issues that were outstanding from the last programming assignment. Things like being able to log out and keep the program running, or having caught exceptions keep the code from stopping. Those issues took most of my time when programming.

After I was content with the way I refactored my code, I started working on creating the BankStatement class as well as the functionality for it. My approach for the method was that I would pass in an ArrayList of type Customer to the method. Once inside the method, I asked the manager for the customer’s first and last names and used a for each loop to iterate through the existing users so that I could verify if the user existed. If there was a match on the names then the program would notify in the console that the generation has begun. After the prompt, the program used the current iteration of the for each loop to save the fields to strings that will be formatted in the bank statement later. All the info that was saved was info like the user’s names/dob/id number/address/phone number; those fields were unchanging and could safely be saved for later use. I also saved the current balance and the account number of each account in the same iteration so that I could report a final balance at the end of the bank statement. As for the initial balances, I created a new ArrayList of type Customer and passed it the values from the input file. Once populated, I was able to iterate through the data structure and pick out the starting balance for each user’s account. Now the actions need to be retrieved, so I used a BufferedReader to read the Log file since I will be only retrieving the information from the log during that specific running time of the program. While inside the file reader, the program actively seeks the lines that contain the first and last name of the individual that was specified at the start of the program. The program will ignore any line that does not have the names of the specified user, as well as a few other cases of lines to ignore. The program will also ignore any line that contains the words login, logout, or inquired because those are not necessary for the bank statement. All of the lines that are readable are inside the condition of ignoring the unnecessary lines. The program then creates variables for the date, action, amount, initial balance, updated balance, and also a string to perform lineFormat. The lineFormat strips all information from the lines that is not necessary, then the program goes to checking for readable lines. The program will then actively look for instances of when the current line contains the words Deposited, Withdrew, Transferred, and Paid, if there is a match, the program will add all the lines to an ArrayList of type String to use for formatting. The program does this with all the remaining information such as, Amount, Initial Balance, Ending Balance, and Date. However, some go through more refining before being added. Once all the information is saved, the bank statement can be written to a file using the first name + last name of a user +Statement at the end. The way the program prints the actions is the line to print is put inside a standard for loop that is constrained by the size of the actionList. All lists share the same size, so it did not matter which I used as the constraints. The program writes the same line with the values of each ArrayList iterated by one for each run-through so it prints out the information at i for each of the actions.

I stated working on the user creation functionality for the bank manager next, it works similarly to the statement at the beginning. At first I declare variables for the accounts and initialize integer variables for the max value of all the account numbers and for the id number. The program then prompts the user with a 6-step information form giving information about the new customer. After the information is provided, the user has the option of creating additional accounts for the user and if they choose to create the new accounts, the manager must provide the initial amount of money the customer will have in the accounts. Once all the information is provided, the accounts are ready to be created. At this point in the program, the object for the customer has not been created yet so if the manager cancels the operation then all information is lost. However, there is a confirm button that can be selected to create the object of the customer. The new customer now exists with an ID and account numbers that are one higher than the previous user's. One thing to note is that the user will not be reflected in the updated balance sheet until they login to their account. One the login is complete; the user is now saved in the updated balance sheet. The updated balance sheet is called at every login and logout of any user.

Next, I worked on the design pattern implementation for the program. I ended up choosing the singleton design pattern and implemented it in my Logger class since there will ever be and should ever be one instance of a logger in my program.

Lastly, I completed the transaction reader. This was the most complicated part of the program for me to solve because my logic defaulted to using if statements and while loops. The transaction reader will dynamically find which index each of the columns are saved to and the program will save every column's index to an int variable for later use. A BufferedReader reads through the file and every line is represented as a string. The program will check if the line contains an action word such as pays, transfers, inquires, deposits, or withdraws. Once a line contains an action word the program will verify if the user shown on the transaction list exists as a user in the bank. If a match occurs then the program will check if the index of the same line contains an account type that the customer will be sending funds from. If applicable, the program can check for sending and receiving, only sending, or only receiving. Once an action is read, then it notifies the user if the transaction is complete.

3. Testing

I tested my program thoroughly and I can say with more confidence that my code is closer to being unbreakable than before. I say closer because I am not sure if there is something I am not thinking of that can end my code but for now, nothing I tried ended my code, so my code has become more resilient to user error. Tested my program nonstop, I would say that most of the time went into testing and debugging my program more than anything. I tested my code's inner workings and logic many times so this is considered white-box testing.

I demoed Minh Dang and his code ran solid with the requirements from previous labs. He does have all the requirements for the current lab too, at least the ones I could see. I cannot see if he implemented a design pattern. However, the thing I could see immediately was that his transaction reader worked and would print right at the start of the code. He was also able to generate a bank statement for every user, however the actions that the users made were not written in the file. A user is also able to be created from the console, the user will only be updated in the updated balance sheet when the program is exited. Also, the exception for file not found was not caught when his program was writing to a file that was still open. The user addition was also limited to just creating a checking account, the account number did increment by one, but the id number of the user was chosen in the console. Other than those points mentioned, the requirements that were met worked solidly and worked as intended.