Rigoberto Quiroz

9/09/2019

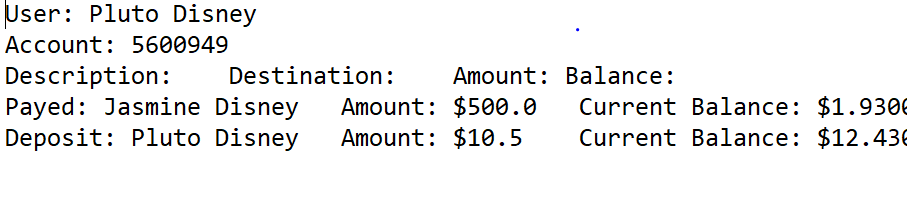
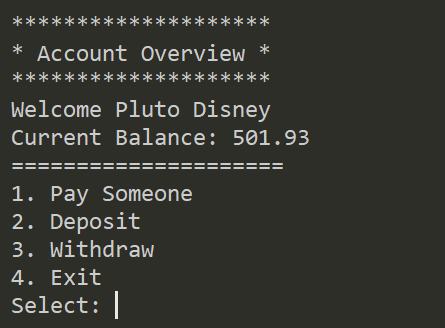
Lab 1 Report

CS3331 3:00 PM – 4:20 PM

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.

**Program Explanation:**

For this lab I created a Checking Class that would simulate some features the Banking system provides. Some of the features are: Balance, deposit, pay someone and withdraw. These features will be true for all the users that are in the data set for this class. Our data is coming from a text file which includes first name, last name, account number, and balance, among other information. With these features implemented as methods, the class should be capable of affecting all the users the same way. Once the user has finished using the features that checking’s provides, a text file will be created with all the history/logs the user made while logged in.



The way I was able to solve this lab was by storing all the data from the text file into a Linked List and whenever a user would do an action, such as deposit, or withdraw, the corresponding methods would get the Linked List and update accordingly. As each action is being made a file called logs (file that keeps record of user actions) is being updated. For example, Pluto Disney logs into his/her account and makes a deposit. At that moment a text file is created with the action he/she made. The information is extracted from the Linked List and it is updated. The text file keeps updating/appending for each action until the user has finished.

Some of the breakdowns I made in my code to simplify the lab was to create separated methods that would accomplish their own task. For example, deposit, withdraw, and pay someone have their methods, which would call the Checking Class methods, extract the user information and update according to their task. Since I’m using a Linked List, I only need to pass the reference of the user I am using, after that I extract the information and update. In the middle of this process another method is implemented that will grab the successful task and update/append the information into the text file. First, I check if the file exists, if it does then I will proceed to create the checking class, otherwise an error will be thrown and stoop the execution of the code. The linked list is created by using a loop, the code captures each row, which is this split into sections, each making up a word, then a new Object of class Checking is created. Since I am using a Linked List I have to create references to the next object in order to simulate a list. After the Linked List has been successfully created, the user will be able to do the task they want.

**What did I Learn?**

What I was able to get out of this lab was a refresher in java syntax. As well as being able to remember and re-learn some of the algorithms used in previous CS courses and see how they can impact this data sets such as the running time. I can improve my solution by making a doubly linked list and some sorting algorithm to make extracting data faster. Rather than creating a new head and having to search all the list. Another way I could improve the solution is by creating a more efficient data structure like hash maps that will get the data faster rather than looking for each instance and comparing data. Overall, I really enjoyed this lab and it approximately took me 3 hours to finish the lab.

**Solution Design:**

What I did in this program was first make sure the file exists, then create a linked list with all the users in the text file. Once that was done, the user will have to input their account number if it does not exist then the process will be repeated until the get in. Otherwise an “Account Overview” menu would display and ask the user what action they want to do, whether it be seeing account info, pay someone, deposit, withdraw, and exit. As stated, before we will extract the data from the LinkedList modify data and re-enter it. Once the user exits, they can see their logs.

I decided I would use a linked list to store my data because I can access user as an instance, traverse the list easily and implement more checking features. I must worry about losing the head, but temp heads can help me with that. I can also store data more efficiently, I do not have to convert everything to a string, then extract it and convert it. I can just extract and modify.

“fileDimensions()” method, I am assuming that the file will always have the same amount of columns. I do not check if the file was written appropriately. Meaning that the file, ‘Bank Users.txt’ must have first name, last name, account number, checking, savings, checking starting balance, and interest rate in that order for the methods such as deposit among others to be able to function. Another assumption that I made was that each user will only be able to log in once, they cannot do actions in one account, then exit their account and jump to another, without having to exit the program. Another assumption would be that “Bank Users.txt” is not updating, meaning that every time a log file is produced, the “Bank Users.txt” is not being updated with the new balance they might have acquired and every time I execute my program a new data structure is created with the info in the text file. Another assumption would be that the Users.txt will always contain the first row of data (first name, last name, account number, etc.), and Users.txt will always have more than 1 user. The last assumption I had was that the “log.txt” file would only record the actions the user made with one account, and not record the changes that might have occurred when methods such as “paySomeone()” are used. For example, Pluto pays Micky, the log file will record that Pluto payed Micky and display Pluto’s balance, but the log file will not display Micky’s balance after the method was performed.

**Testing**

The ways I tested my program was with the following:

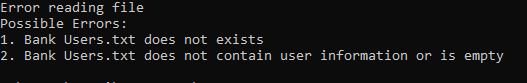
* If user selects an account that is not in the Linked List
* If the user selects an option that is not available to do
* If the user inputs Strings or characters rather than doubles or integers
* All x amount of user actions appears in log.txt file
* If Bank Users.txt does not exits
* If log file does not exist
* If user attempts to use more money than what they have

For these test cases I used both Black and White box testing to make sure that the program was doing the correct thing. For Black box testing, I made sure that the console would output the correct user information and correct errors when the user inputs something wrong, or files are missing. White box testing, I made sure that each method received a reference to the linked list, the account the user is using, the testing of all conditions and loops used, get expected outputs and test try and catch scenarios.

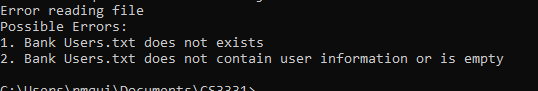
I tested possible scenarios the user may encounter, and everything was functional, but the testing can be improved by allowing other people use the code and see if they might encounter other types of bugs, I did not take account for. The following test cases broke my code, but they were fixed, and they now output a console error when triggered due to the use of try and catch exceptions.

**Test Results:**

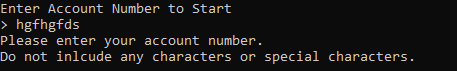
**Test Case**: Bank Users.txt was not found



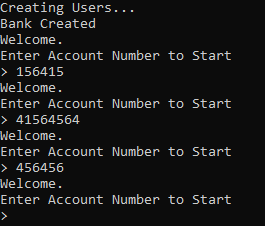
**Test Case**: Bank Users.txt contains no user info



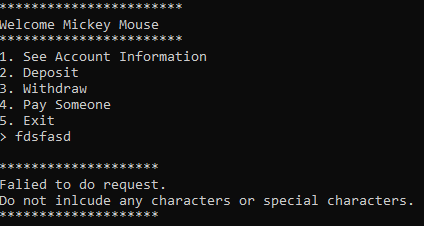
**Test Case**: User inputs special characters



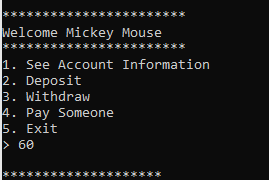
**Test Case**: User enters an account number that does not exists



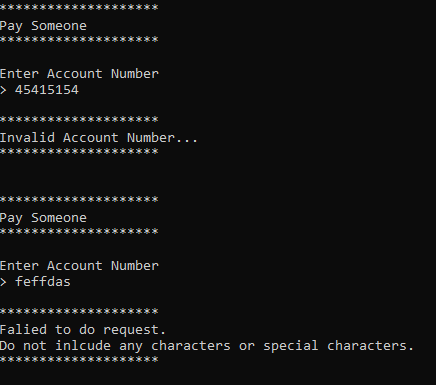
**Test Case**: User input random strings or characters



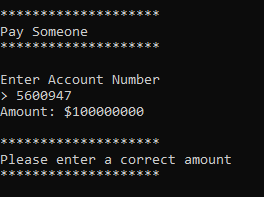
**Test Case**: User inputs a menu choice that does not exists



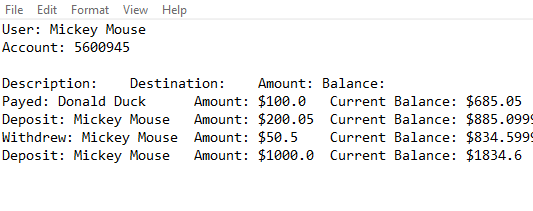
**Test Case**: User inputs random strings or characters and account number does not exist. (Same output in Depositing and Withdrawing error actions)



**Test Case**: User uses more money than what they have (Same output in withdraw error action)



**Test Case**: Log File outputs x amount of actions preformed



Test Case: User inputs negative amount (Also applies for Withdraw and Pay Someone methods)

