Rigoberto Quiroz

9/09/2019

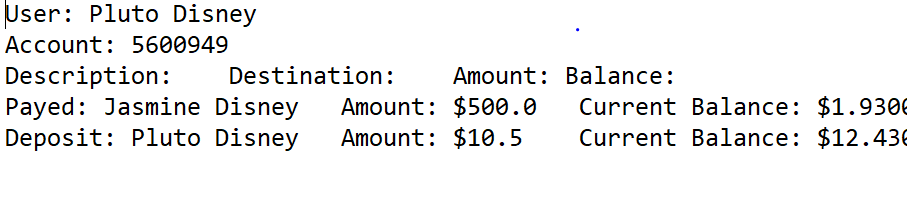
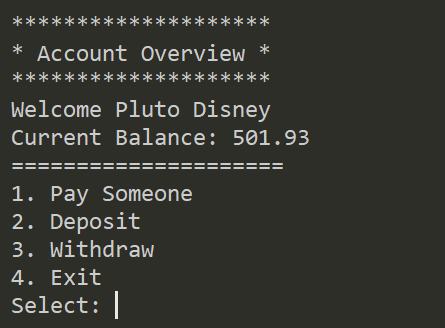
Lab 1 Report

CS3331 3:00 OM – 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 2D array and whenever a user would do an action, such as deposit, or withdraw the corresponding methods would get the 2D array 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 2D array and it is updated. The text file keeps updating/appending for each action until the user has finished. The 2D array is being updated by reference, which explains why most of my methods are void.

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 will get the 2D array, extract the user information and update according to their task. Since the 2D array is made up of Strings, each time I extract a certain piece of information I must convert it into an integer or double, manipulate it, convert it back to a String and set it back to its index in the array. 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. Another break down I made was the creation of the data structure which holds all the user data. First, I check if the file exists, if it does then we will get the dimensions of the file. How many rows and columns does the file have? Once answered I created a 2D array with the specified dimensions and insert each piece of data into their corresponding places. Each row contains one user. Each column represents something about the user information. Each task is separated into its own method to make it more readable and easier to understand. After the 2D 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 list class that can store different data types rather than making everything a String and have to convert each field when needed. Another way I could improve the solution is by created a more efficient data structure such a linked list to make an add user feature and not rely too much in a text file or a hash map that will get the data faster rather than looking for each index 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, get its dimensions (How many rows and Columns?) and create the 2D array. Once that was done a list of accounts would appear and the user would have to choose which account, they want to use. Then an “Account Overview” menu would display and ask the user what action they want to do, whether it be pay someone, deposit, withdraw, and exit. Which choice they make the 2D array and the logs file will be updated. As stated, before we will extract the data from the 2D array convert the field to the appropriate data type, affect it and re-enter it into the 2D array as a string. Once the user exists, they can see their logs.

I decided I would use a 2D array to store my information because it is simple to handle and easy to understand, I do not have to worry about losing to head or come up with an equation that will effectively store data in a hash map. The 2D array can store this data and acquire such information if index is known. Which leads to the following assumptions I had when creating some methods.

“fileDimensions()” method, I am assuming that the file will always have the same amount of columns, even though I check that each row has 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 2D array is created with the info in the text file. 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:

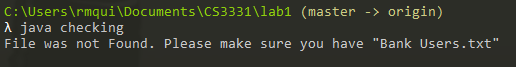
* If user selects an account that is not in the 2D array
* If the user selects an option that is not available to do
* If the user inputs Strings or characters rather than doubles or integers
* All x amount of user actions appear in log.txt file
* If Bank Users.txt does not exits
* If Bank Users.txt is written is in an inconsistent format
* 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 the 2D array, the account the user is using, the testing of all conditions and loos used, get expected outputs (2D array is being updated) and test try and catch scenarios.

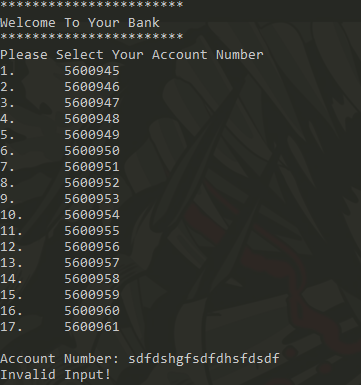
I tested possible scenarios the user may encounter, and everything was functional, but the testing can be improved by allowing other people use the code and see if they might encounter other types of bugs, I did not take account for. The following test cases broke my code, but they were fixed, and they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

**Test Results:**

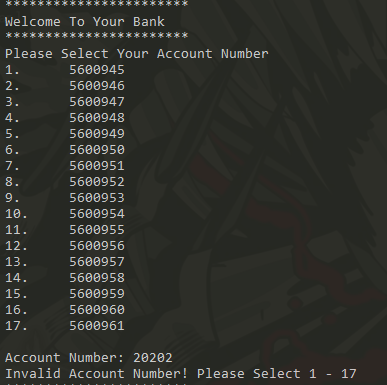
**Test Case**: Bank Users.txt was not found



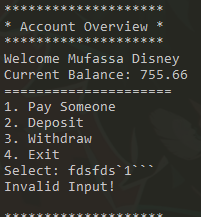
**Test Case**: User inputs invalid account number



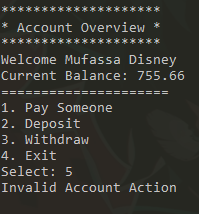
**Test Case**: User enters an account number that does not exists



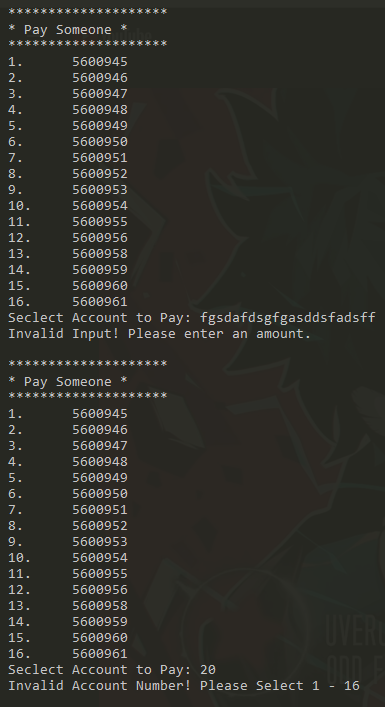
**Test Case**: User input random strings or characters



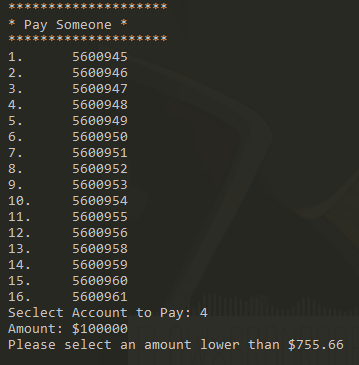
**Test Case**: User inputs a menu choice that does not exists



**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

