Rigoberto Quiroz

11/08/2019

Lab 2 Report

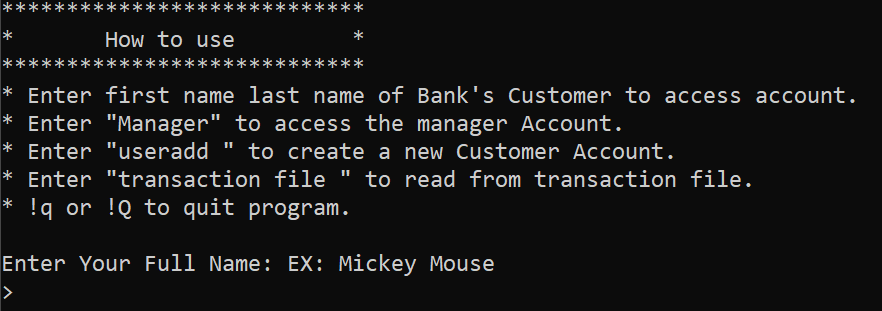
CS3331 3:00PM – 4:20PM

**Honesty Statement:**

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 we had to refactor our previous lab (lab 2) to be able to handle some new features. Some of the new features include adding new users, customer name handling, reading and handling from transaction file, creating a Bank Statement class and adding some new Manager user features. The same features from the previous lab will still function, deposit, withdraw, and transfer as well as creating a log.txt file that will record all the actions did while logged in



The way I was able to solve this lab was by storing all my customer information into a linked list. This linked list will make class to other classes such as the Person, Account and other classes to store and retrieve customer information. The same functionalities from the previous lab still work. Customers can deposit, withdraw, inquire, transfer/pay someone. The only difference this time would be that each customer will now have their own private logs. Another difference would be that I added a feature that we can read from a transaction file, all the transaction written will be executed into the program as if the user was executing it themselves. We will display when a transaction has been completed or failed. If failed, we will display which transaction failed and why. A new feature that the manager account can do is create banking statements from any customer they want. This banking statements will be made by the new Bank Statement class and printed out in a txt file. Another new feature we added was the creation of new users while using the program. New users will have the opportunity to become new customers, they will be required to fill in their first name, last name, date of birth, address, phone number and at least make a savings account. The new users will be placed at the end of the linked list and will get a new id number.

Some of the breakdowns I made in my program were that I made helper methods that will contribute to the reading of the transaction file, the creation of new users, and the logs of each individual user. For example the breakdowns in reading the transaction file was that I will first get all the header labels and store them in a array, after that I will get the information related to each header/label, as I am getting the information I am preforming the task (in other methods) of depositing, withdrawing, or transferring, and displaying if the task was successful or not. A similar process was taken in the breakdown of the other features, the creation of users will use different method to check if the user input is correct, and logs making sure that we have enough space to store all the logs , otherwise resizing for more logs.

**What did I Learn:**

What I learned from this lab was that code structure is very necessary if you want to refactor your code and implement new features. As well as a better understanding of design patterns, you will notice that I used the template design pattern to implement the creation of bank statements. Having other methods set up the printing of the statement and have a main method that will call upon all the help methods to create the statement, works like a recipe. I also got a better understanding of strings, how. When adding new users into my linked list, I had to make sure that the new users where entering the appropriate information. For example, when the user was entering their first or last name, I had to check that the user was not entering whitespaces or special characters. What I ended up doing was checking the ascii code of each character in string and making sure that character was part of a certain range of characters I wanted to user to do.

**Solution Design:**

My solution designs consist of the same as the precious lab (lab2), using the linked list to enter and output certain user request. The difference is that I had to refactor certain parts of create users, deposit, withdraw, transfer and inquire methods. Since my input files will not always have the same format I have to take into account that the headers the file has and store appropriately, the same thing applies for the Transaction file. I had to make a method that will make an array of all the headers and when storing the information into my linked list I know which index that information will be thanks to the array of headers.

I also had to refactor my action features which allowed to user to deposit, withdraw, and transfer from their accounts. Since I was reading from my transaction file and did not want to write new methods that would do the same thing but just handle the transaction file, I added new parameters which would tell me if I was reading from a transaction file or just a customer was preforming these tasks. The same thing applies for the all of the other functions a user might do.

As the user or the transaction file are being used, their logs will be recorded, in my linked list class I have an array that will store all the logs of size 10. If the user reaches the array limit, then the array will increase in size by double. Allowing the user to have more logs. In order to retrieve this information, the user will have to enter index to access to reach specific logs. When the manager wants to print out the banking statement then we will retrieve all the information from the Bank class and iterate through the log array to retrieve all their logs.

For this lab we had to implement some design patterns, the design pattern I used was the template design pattern to perform the Bank Statement. The manager will just call the Bank Statement class statement method, and that method will call in order all other methods needed to create the header, body and footer of the bank statement.

**Testing:**

The ways I tested my program was with the following:

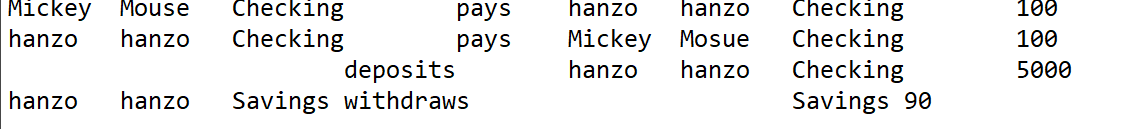
* In transaction file, if we use an account that is not in our link list
* Invalid transactions in file
* Entering incorrect information when adding new users
* Making sure that the new user info is being stored and has a new id
* New user transaction statement is correct
* Transaction file does not exist
* Different patterns of headers
* All transactions appear

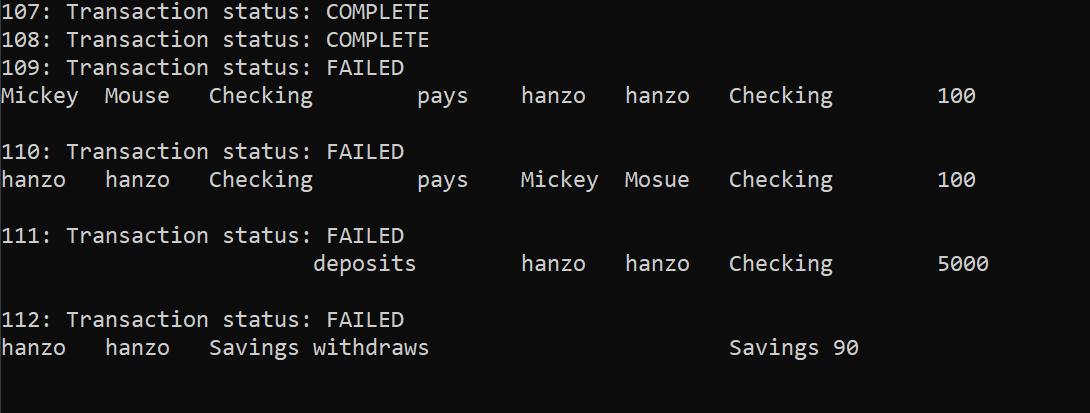
For these test cases I used both Black and White box testing to make sure that the program was doing the correct thing. For Black box testing, I made sure that the console would output the correct user information and correct errors when the user inputs something wrong, or files are missing. White box 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.

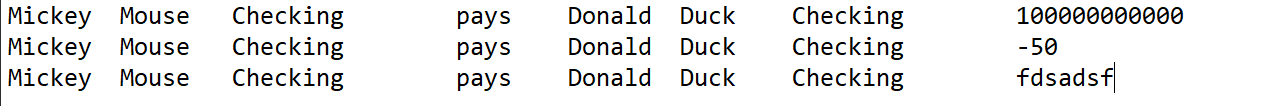
**Test Results:**

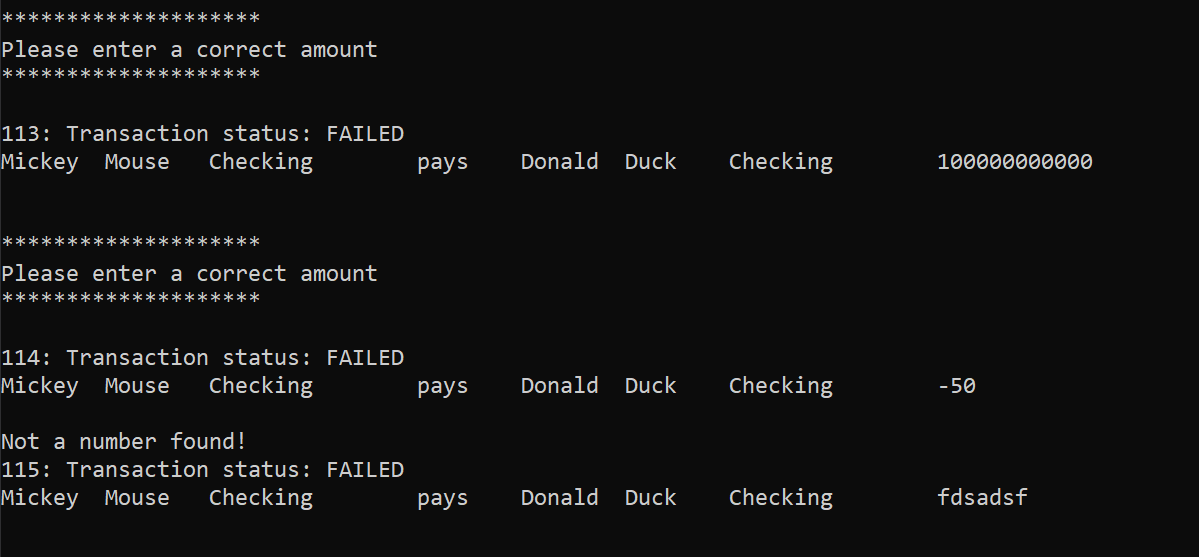
**Test Case**: Transaction file using an account that does not exists.



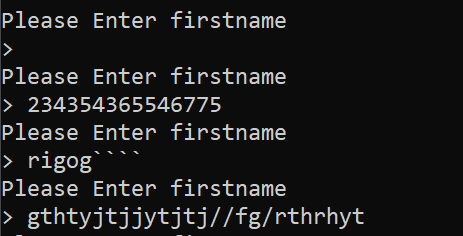


**Test Case**: Invalid transaction in file

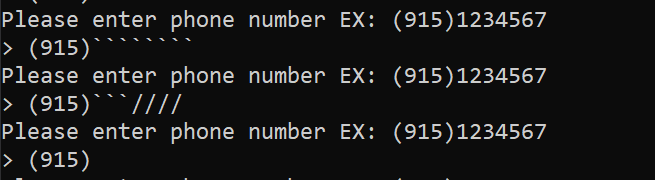


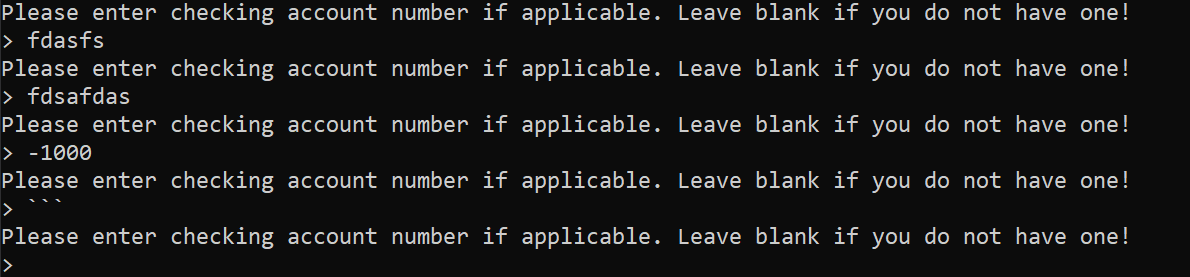


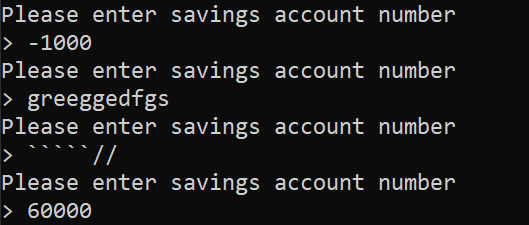
**Test Case**: adding users

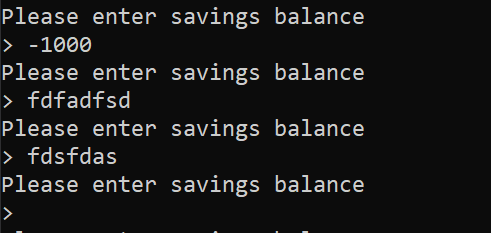


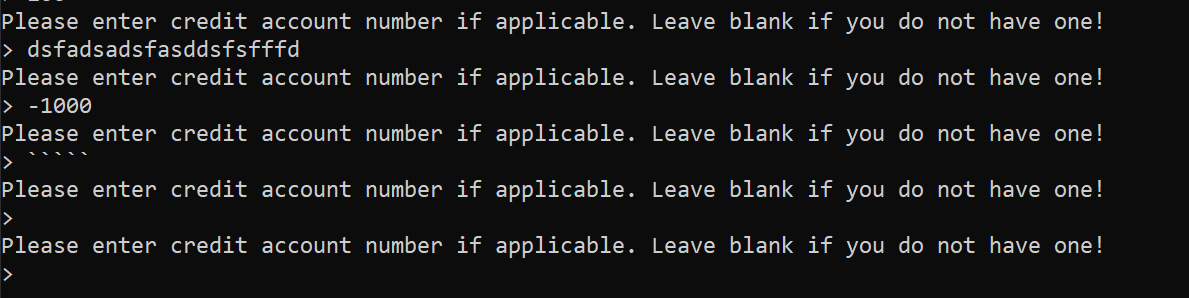
Same applies for the entering last name

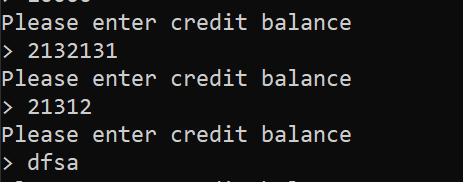


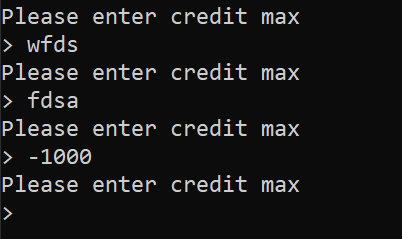




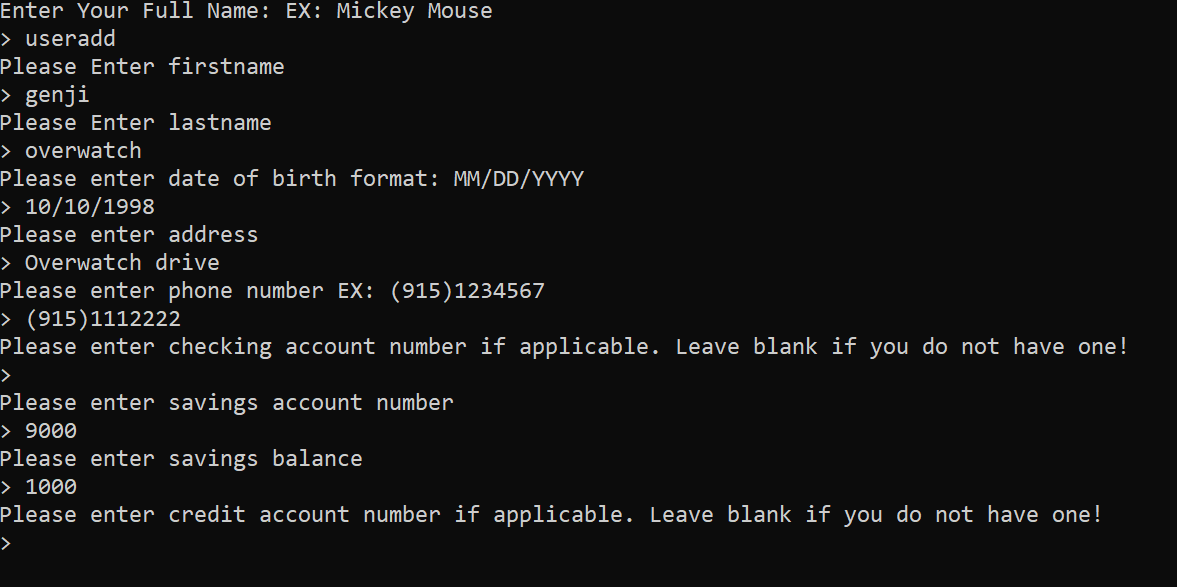


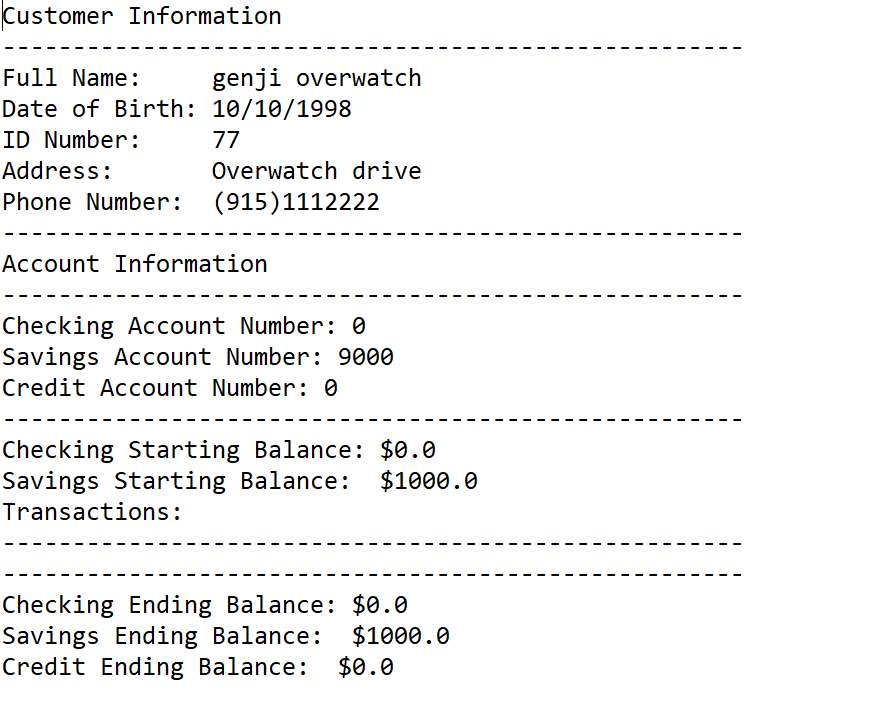


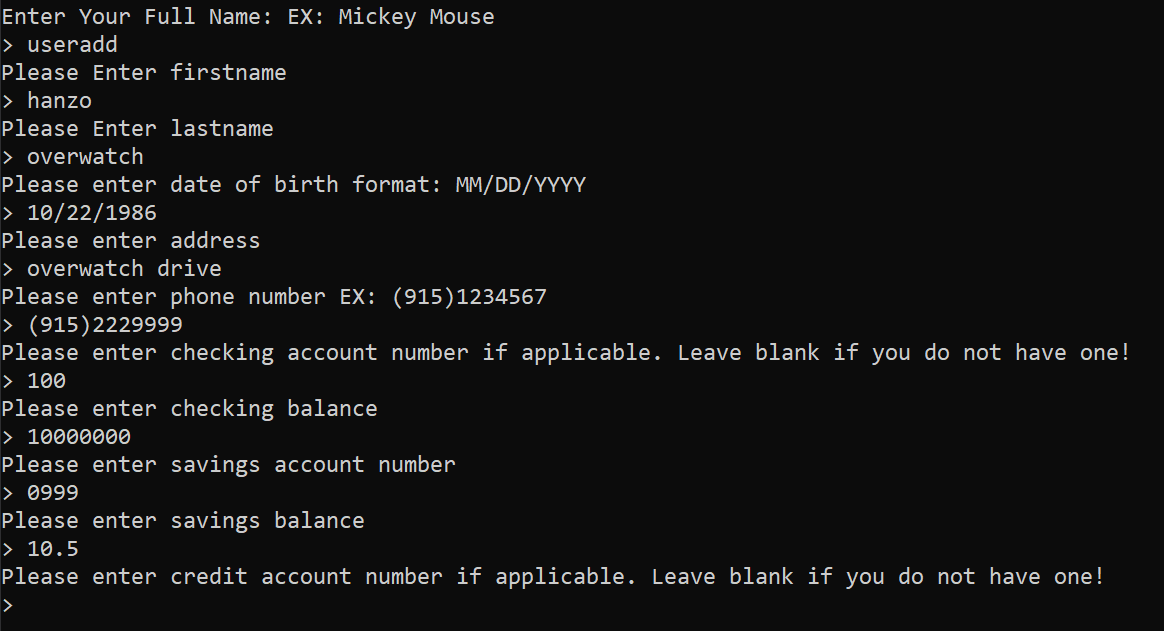


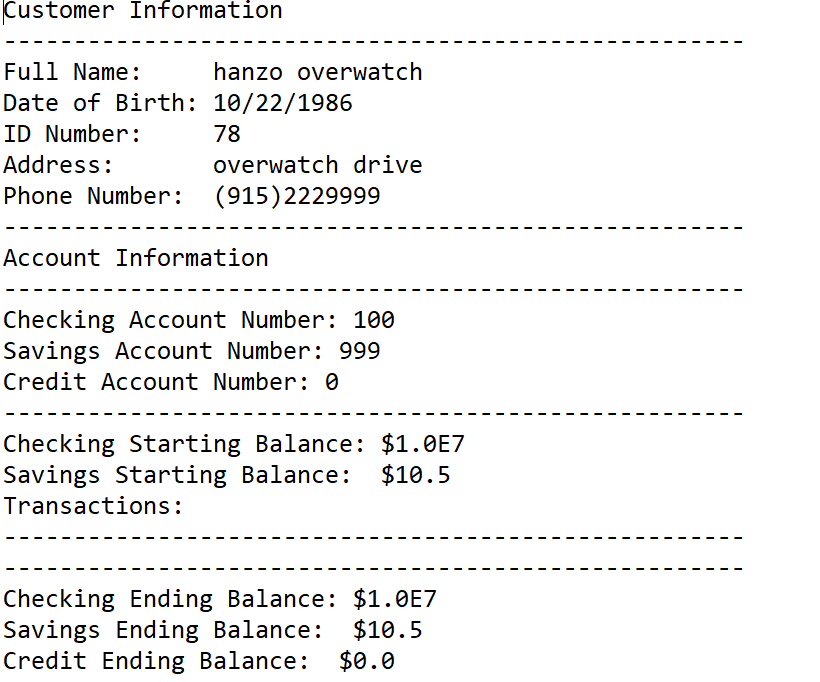


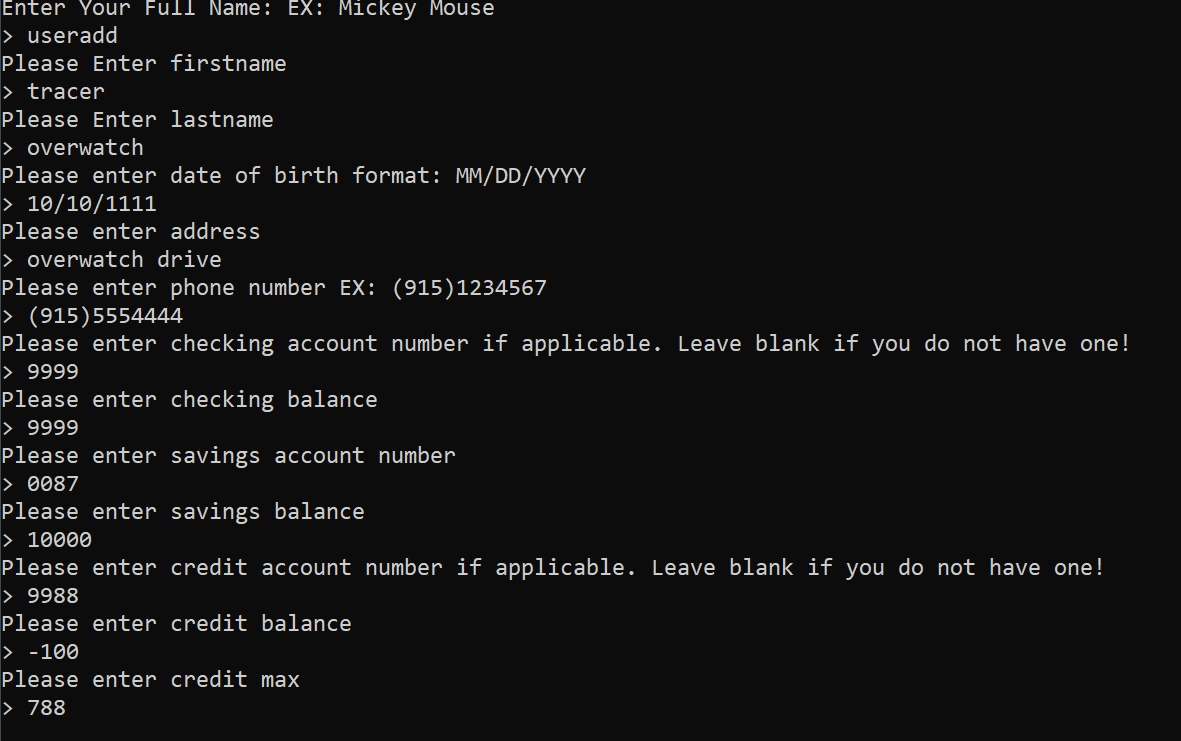
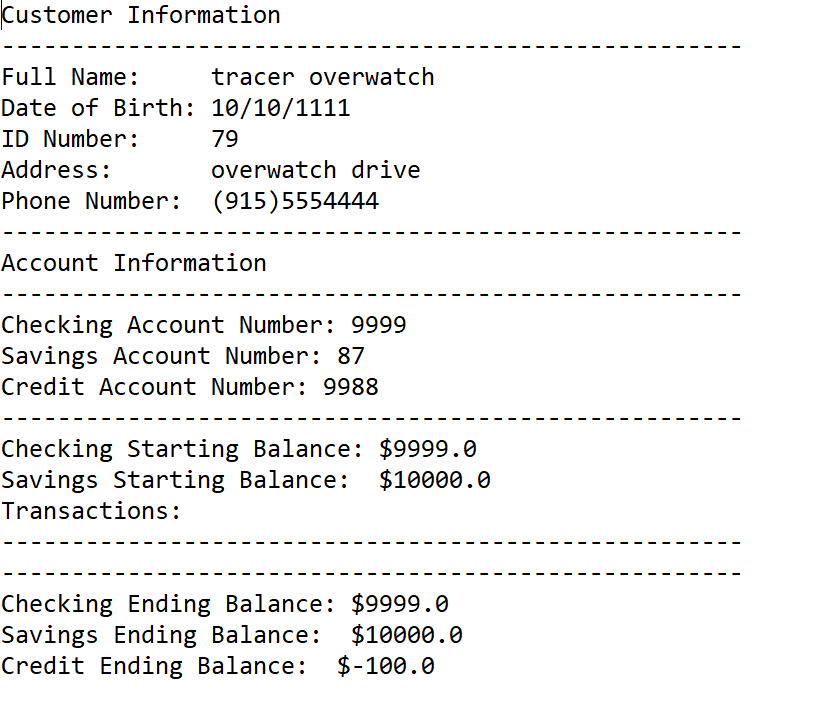
**Test Case**: user adding and printing statement



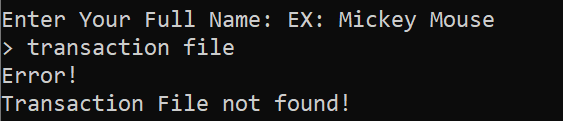




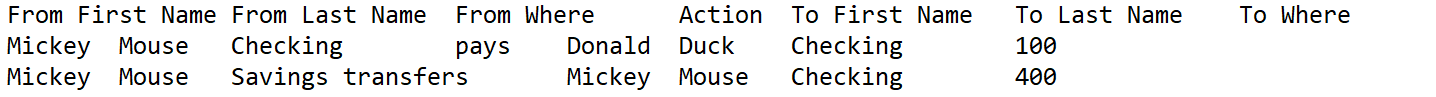


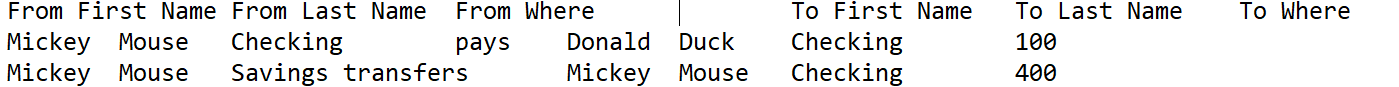
 

**Test Case**: transaction file not found



**Test Case**: Different header labels





**Test Case**: All transaction appears

