Program Explanation:

The primary objective of this project was to design a functional simulation of a banking system that could facilitate common banking transactions such as deposits, withdrawals, transfers, and balance inquiries for a client. To actualize this, I created seven interconnected classes, namely Bank, Account, Checking, Savings, Credit, Customer, and Transaction. These classes encapsulated the different aspects and functions of the banking system. The robust use of polymorphism was key in managing the diversity of accounts and transactions, employing techniques like method overloading and overriding in the bank client interface. I tackled the task systematically by breaking it down into smaller, more digestible parts. For instance, I created distinct methods to handle different parts of the program, such as the main menu, the csv file reading, and the manager menu.

What did I learn?

This assignment has improved my understanding of object-oriented programming, particularly in the domain of polymorphism. The experience of implementing polymorphism in the form of method overloading and overriding has been instructive. Despite this, I see room for improvement in my solution, including the enhancement of code conciseness, readability, and reusability. A broader and deeper implementation of polymorphism could potentially have made the code more robust, but my current level of expertise made that difficult. The project, in its entirety, took a week to complete.

Solution Design:

The design of the program commenced with the construction of essential classes, namely Bank, Account, Checking, Savings, Credit, Customer, and Transaction, each representing distinct entities within the banking system. Upon defining these classes, I implemented a method in the main class to read client data from a CSV file, subsequently creating instances of bank clients and feeding their data into the corresponding classes - Customer, Credit, Savings, and Checking. List was the primary data structure utilized, facilitating efficient storage of client data and transaction management. Transactional operations such as deposits, withdrawals, and transfers modified the initial account balances established by the CSV reader based on the nature of the transaction: deposits increased the balance, withdrawals reduced it, and transfers adjusted the balances of two accounts simultaneously. All these changes were tracked and instantaneously updated, ensuring the account balances accurately reflected the current status. The program design was predicated on the assumption that clients performing transactions were already registered within the bank's system, a necessary condition for associating transactions with valid accounts.

Testing:

I used white box testing to verify the program, as the code's internal structure was known to the developers. During the testing process, there were instances where the code broke due to unanticipated user inputs or data orientation issues. However, these occurrences were turned into opportunities to improve the code. For example, an initial problem arose with case sensitivity in user inputs in a switch case. This was addressed by converting user inputs into lower case to match the switch case. Additionally, checking and savings balances were initially reversed, an issue identified and solved due to testing.

Test results:

The testing phase confirmed that the system could accurately execute all banking transactions. Transactions were properly recorded in a log, viewable by the bank manager and exported to a text file upon system closure. These results demonstrated the successful functionality and reliability of the system in handling various banking operations. The process of building, testing, and refining this banking system simulation served not only to provide a practical banking solution but also to deepen my understanding and mastery of object-oriented programming and its various principles.

------------------------------------------------------------------------------------------------------------------------------

Code displayed when manager access log after client performs transaction:   
>----Customer Menu----

What would you like to do today?

1. Inquiry about a balance

2. Deposit money to an account

3. Withdraw money from an account

4. Transfer money between accounts

5. Switch Bank Roles

--Type 'EXIT' to Close--

5

Back to Role Selection

are you a Bank Client or Manager?

A. Bank Client

B. Bank Manager

b

>Enter manager password:

>manager

----Manager Menu----

Would you like to inquire by name or by type/number

A. Inquire account by name.

B. Inquire account by type/number

>a

What is the name of the customer>

minnie mouse

Costumer Found!

Which account would you like to inquire?

1. Checkings

2. Savings

3. Credit

4. Transaction Log

>4

Log:

Customer Minnie Mouse with account type: Checkings made a transfer to Savings. Current balance: 399.0

Customer Minnie Mouse with account type: a made a balance inquiry. Current balance: 726.99

Customer Minnie Mouse with account type: B made a deposit. Current balance: 10000.0

------------------------------------------------------------------------------------------------------------------------------

Code Review:

In conducting the code review, I followed a systematic and thorough process, examining each function independently to ensure its correct operation and optimal performance. Using a comprehensive code review checklist, I inspected each part of the code to verify adherence to coding standards and best practices. This checklist covered a broad array of considerations, such as variable naming conventions, function modularity, code readability, and documentation. This detailed examination enabled me to identify and rectify any potential issues, hence improving the overall quality and reliability of the final codebase. Therefore, the code review process was an important step towards refining the code and ensuring its readiness for use in a real-world context.