Banking Management System

R Nishanth Datta [BL.EN.U4AID23043]

S Sai Preran[BL.EN.U4AID23045]

Neha Girish Mantur[BL.EN.U4AID23034]

M Jayvardhan Reddy[BL.EN.U4AID23033]

Amrita School of Artificial Intelligence

Amrita Vishwa Vidyapeetham, Artificial Intelligence and Data Science,

Amrita Nagar, Choodasandra, Junnasandra, Bengaluru, Karnataka, India

#1. [Bl.en.u4aid23043@bl.students.amrita.edu](mailto:Bl.en.u4aid23043@bl.students.amrita.edu)

#2. [Bl.en.u4aid23045@bl.students.amrita.edu](mailto:Bl.en.u4aid23045@bl.students.amrita.edu)

#3. [Bl.en.u4aid23034@bl.students.amrita.edu](mailto:Bl.en.u4aid23034@bl.students.amrita.edu)

#4. [Bl.en.u4aid23033@blstudents.amrita.edu](mailto:Bl.en.u4aid23033@blstudents.amrita.edu)

*Abstract* —

This Java code creates a basic financial system with a graphical user interface (GUI). Users can create two different kinds of accounts using it: current and savings accounts. Users can create an account, use their usernames and passwords to log in, and carry out standard operations including fund transfers between accounts, withdrawals, deposits, and balance checks. However, the system does not have persistent data storage; instead, account information is stored in arrays, which are destroyed when a programme terminates. Security elements and account number generation are also streamlined. Using Java's Swing framework for GUI interaction, the code illustrates fundamental banking functions; however, it requires enhancements for practical use..

Introduction

The goal of this Java code is to illustrate fundamental ideas of object-oriented programming and GUI development using the Java programming language through a streamlined implementation of a financial system. With the help of dialogue boxes and buttons, users can interact with the bank through an intuitive graphical user interface created by the system using the Swing framework

Basic banking functions including opening an account, logging in, making withdrawals and deposits, checking one's balance, and transferring money across accounts are all simulated by the code.  
There are two basic account kinds available: savings and current accounts, each with their own set of regulations. When enrolling, users can select the sort of account they want, provide personal data such name, address, and birthdate, and create a username and password for their login. Additionally, the system includes fundamental security features to verify

It is imperative to acknowledge the limits of this code, despite its ability to demonstrate a financial system in a practical manner. Notably, arrays are used by the system to store account information in place of persistent data storage. This implies that when the programme ends, all data is lost. Furthermore, for demonstration purposes, account number generation and password security have been simplified; for real-world use, these features would need to be significantly improved. Although the code is a useful teaching tool for demonstrating the essential ideas behind developing interactive software systems in Java, it would require significant improvement before being implemented in a real-world banking setting.

The login process acts as the initial gateway for users to access the platform services. When the user is navigated to login page users are asked to enter their credentials such as username, email address and password. This mechanism ensures privacy of user’s account details and only registered users can proceed with renting or managing vehicles.

**Methodologies**

Code demonstrates a basic Banking Management system built using Java AWT and Swing. It lacks a clear adherence to specific software development methodologies, but analysis reveals elements of various approaches.

a)Observed Methodologies and Techniques:

• Code executes sequentially through predefined functions

b)Event-Driven Programming:

• User interactions (button clicks, etc.) trigger events.

• Events are handled using the ActionListener interface, especially for the 'Checkout' button.

c)Modular Design:

• Functional separation into methods

• Promotes reusability and maintainability.

• GUI Development using AWT and Swing:

• User interface created using Java AWT (layouts) and Swing components (labels, text fields, etc.).

• Missing Methodologies (for Potential Improvement)

**System Design and Implementation**

**1)**Accountable Interface:

This defines a standard set of actions that need to be supported by all account types. This guarantees that dealing with various account kinds (Savings, Current, etc.) will be consistent. The Accountable interface serves as the system's standard operating procedure or contract for all bank account kinds. It requires that any class that implements this interface give the methods defined inside it actual implementations: Deposit(), Withdraw(), and Details().

A)Withdraw():

The Withdraw() method is in charge of outlining the reasoning behind taking money out of a bank account.  
Despite being stated in the interface, the real implementation will change based on the particular kind of account. For example, a savings account and a current account may have differing withdrawal restrictions and costs.

B)Deposit():

This function establishes the reasoning behind a bank account deposit.As with Withdraw(), different account types may have different implementations of the deposit capability.

C)Details():

2) Abstract Account Class:

This creates a standard structure for every kind of tangible account. It specifies a common procedure for balancing checks and stores common data fields like

* AccountHolders\_name
* Sex
* AccountNumber
* AccountType
* PhoneNo
* Address
* Dob
* balance
* IFSC
* Branch\_name

**Object-Oriented Design**

1)Abstraction:

This helps in putting a simpler interface in front of complicated implementation details. This frees you from having to understand an object's internal workings in order to interact with it at a higher level. Through the abstraction of each account's unique implementation details, the Accountable interface defines common actions for all account types.

2)Encapsulation:

This referes to combining data and the operations performed on it into a single entity (the object). This safeguards the information and guarantees controlled access to and modification of it. The methods to interact with account information (Check\_Balance(), Withdraw(), Deposit(), etc.) and account information (name, balance, etc.) are encapsulated in the Account class.

3)Inheritance:

The process of building new classes (subclasses) that inherit the characteristics and behaviours of existing classes (superclasses). It also builds a hierarchy of similar classes and encourages code reuse. SavingsAccount and CurrentAccount are derived from the Account class, sharing common methods and attributes and setting their own set of rules (interest rate, withdrawal limits).

4)Polymorphism:

A fundamental component of object-oriented programming, polymorphism literally translates to "many forms." Programming terminology for this feature describes how objects of various classes can react differently to the same method call.By offering their own implementations for the methods specified in the interface, SavingsAccount and CurrentAccount both carry out the Accountable interface. The subclasses override Withdraw, Deposit, and Details to give each kind of account unique behaviour.This function defines how a bank account's associated information is to be shown to the user.Details like the name of the account holder, account number, amount, account type, and any other pertinent data may be displayed.

**Class Hierarchy**

Account is the main class; it acts as an abstract model for many kinds of accounts. It includes information on the account holder's name, sex, account number, address, phone number, balance, IFSC code, and branch name, among other details that are shared by all accounts. The Accountable interface, which specifies procedures for carrying out necessary banking operations—withdrawal, deposit, and details—is implemented by Account.The Account class is the ancestor of two concrete classes, SavingsAccount and CurrentAccount. These classes are for particular account kinds that have unique features, like different interest rates and withdrawal restrictions. By overriding the Accountable interface's functions, both classes customise the implementation to meet their own requirements.The Customer and Customer2 classes extend SavingsAccount and CurrentAccount, respectively, farther down the hierarchy. These classes include features unique to dealing with customers, such as signing up, entering

**Functionality**

The following are the main features that the system offers:  
1. New User Registration:

This feature enables users to open new current or savings accounts. The system asks users to create a unique username and password, choose their branch, and enter personal information throughout the registration process.  
2. Old User Login:

Allows current users to log in with their login information. After successfully logging in, users are shown a menu with options to complete different types of transactions.  
3. Transaction Handling:

Basic banking transactions are supported by the system.  
4.Withdraw:

Permits users to take money out of their accounts while abiding by the particular withdrawal restrictions associated with each kind of account.  
5.Deposit:

Permits customers to add money to their accounts.  
Transfer: Makes it easier to move money between accounts in the system.  
6.Check Balance:

Shows the user's account balance as of right now.

## **PROBLEM STATEMENT**

**Banking System Simulation using Object-Oriented Programming in Java**

The BankingSystem code attempts to solve the following issue: creating and executing a streamlined banking system that  
models different kinds of accounts: Different account kinds, such savings accounts and current accounts, each with their own set of regulations and restrictions (like interest rates and withdrawal limits), should be supported by the system.The Main Goal is to Maintains client data.The system should keep track of and handle customer data, such as account information, login passwords, and personal information.Thishandles standard banking transactions: Essential functions such fund transfers between accounts, deposits, withdrawals, and balance inquiries should be available to users.  
Feature-rich interface: With a straightforward graphical user interface (GUI), users should be able to interact with the system to register, log in, and manage their accounts.

**Future Scope**

A fundamental framework for account management and basic transaction processing is provided by the ongoing Banking Management System project. It may, however, be greatly improved to offer a more thorough banking experience. The project's future goals include adding functionality such as periodic interest computation, statement creation, transaction history monitoring, and support for account types other than current and savings accounts. Furthermore, adding online banking functions like bill payment, cell top-ups, and external financial transfers would improve user usability and convenienc significantly. Enhancing security, user experience, and data management should also come first. A strong and user-friendly system would include the implementation of persistent storage via a database, strong data validation, search and filtering capabilities, and a more advanced GUI framework. Furthermore, enhancing authentication protocols, encrypting data, and conducting security audits will guarantee the dependability and safety of the system. Other directions for future development include investigating the creation of a companion mobile application and integrating third-party APIs for services like SMS notifications.

**Goals**

The project's main goal is to strike a balance between functional simulation and instructional value. It does this by providing a concrete illustration of OOP's capabilities and establishing the foundation for a possibly more advanced banking system.

1. Educational Demonstration:

The main objective is to demonstrate how object-oriented programming ideas—such as inheritance, polymorphism, and encapsulation—can be used in real-world situations.. The project acts as a guide for comprehending the ways in which OOP may be utilised to manage intricate interactions between various entities, write reusable components, and organise code.  
2. Useful Modelling:  
 The project's goal is to build a condensed version of a banking environment that users may interact with to access fundamental banking functions.Users should be able to create accounts, manage their accounts, carry out transactions (withdrawals, deposits, and transfers), and safely handle login and logout.Another key objective is to generate revenue and profitability for corporate tenants through effective inventory management and pricing strategies by accurately tracking vehicle usage reducing downtime, and implementing dynamic pricing policies, the system helps rental companies maximize profile profits from investment while maintaining competitive rates.

3. Lay the Groundwork for Growth:

Create a Base System The project offers a foundational structure that can be built upon and enhanced to include more intricate features and functionalities.The application of OOP concepts facilitates the future addition of new account kinds, features, and integrations.

**Results**

When the BankingSystem.java programme runs, a graphical application that mimics a condensed banking environment is displayed. The outcomes you may see are broken out as follows:  
User Interface:  
A welcome message is displayed in the "Banking System" window that opens when the programme runs.  
"New User Registration," "Old User Login," and "Exit" are the three buttons that are displayed.  
New User Registration:

Selecting "New User Registration" invites the user to open a current account or a savings account.After that, the user is guided by the system to enter personal information such as name, birthdate, address, and phone number, choose a branch, establish an initial balance, and create a username and password.  
For the new account, an exclusive account number is generated immediately.

Old User Login:

The user is prompted to input their password and username upon clicking "Old User Login".  
The user sees a menu with the options "Withdraw", "Deposit", "Transfer", "Check Balance", "Details", "Change Password", "Close Account" and "Logout" if the login credentials match those of an existing account.  
Transaction Handling:

When a user chooses a transaction option, a dialogue box will appear and ask for pertinent details. For example, the amount to be withdrawn or deposited, the recipient username for transfers, etc.  
The requested transaction is completed by the system, which also outputs dialogue messages (such as "Insufficient Balance" or "Transfer Successful") to indicate success or failure.  
Account Management:

The user's personal and account-specific information is displayed via the "Details" option.  
The option to "Change Password" lets the user create a new password for their

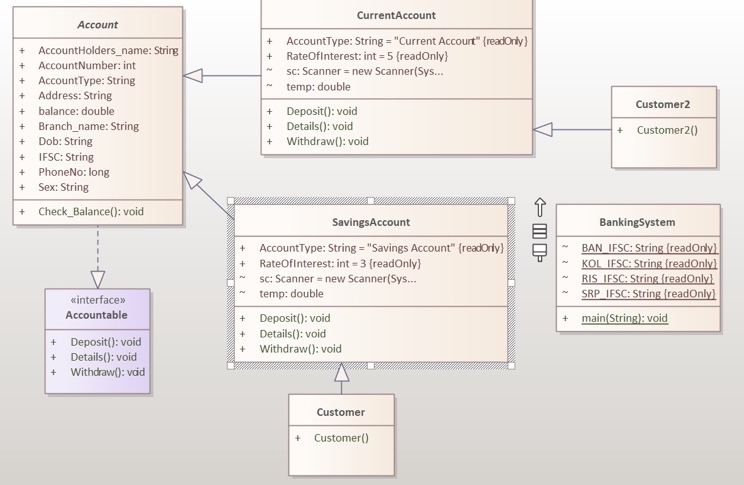
Old User Login:

The user is prompted to input their password and username upon clicking "Old User Login".  
The user is logged in and shown a choice of options if the credentials match an already-existing account:   
To remove a user's account from the system, select "Close Account".  
Log Off and Leave:  
The user is brought back to the original login screen by selecting "Logout".  
The application can be closed by clicking "Exit".

**Conclusion**

A simple but functional example of a financial system built with fundamental Java object-oriented programming concepts may be found in the BankingSystem.java file. Despite having few capabilities, the system effectively illustrates how inheritance, polymorphism, and encapsulation may be used to manage client data, represent various account types, and perform standard banking tasks. Users can interact with the system in an easy and engaging way thanks to the Java Swing-built user interface.  
The existing implementation establishes a strong basis for future growth, even if its primary purpose is to serve as a teaching example. The project can develop into a more complete and useful banking system by adding features like database integration for persistent data storage, improved UI design, stronger security measures, and a larger variety of functionality. This illustrates that is rich in application The Java-based project Banking Management System. which displays a basic banking system, has been examined in this paper. The implementation successfully illustrates the potential of object-oriented programming concepts like inheritance, polymorphism, and encapsulation despite its simplicity. These concepts are used to manage customer data, model different kinds of accounts, and perform necessary banking tasks. Through an easy-to-use graphical interface, users may interact with these functions thanks to the system's user interface, which was created using Java Swing.  
 Although the project's main purpose is to demonstrate OOP ideas, it also offers a strong framework for further development. This project can be expanded into a more complete and useful banking system by adding more sophisticated features such persistent data storage through databases, improved security measures, a richer user interface, and new financial functionalities.

**UML Diagram**

****

.