***Project Title:*** BANK MANAGEMENT SYSTEM.

***Course Name:*** Programming in Java.

***Subject Code:*** CSE2006.

***Submitted By:*** Rahul Kumar Behera.

***Submitted To:*** VIT BHOPAL University.

***Date of Submission:*** 25-11-2025

***INTRODUCTION:*** This report details the design, implementation, and testing of the Bank Management system, this project mostly uses Concept of Object-Oriented Programming, specifically Multilevel Inheritance, to a real-world problem domain. This system provides a functional simulation of standard banking operation, offering a modular, secure, and user-friendly experience.

***Problem Statement:*** Traditional banking, though now mostly digital, requires strong and scalable software systems to support operation. The core problem to be solved herein is the design and implementation of a simplified, console-based banking system that automates core operations of banking. This should be able to manage customer accounts efficiently, enable financial transactions-deposits and withdrawals-along with value-added services such as interest calculation, with data integrity and clarity in user experience.

## *Functional Requirements:*

1. **Account Management Module**: This system allows the user to create a new bank account by entering details such as Account Number, Name, Age, Gender, Account Type, and Initial Balance.
    - ***Input :*** User data via console.
    - ***Output:*** A new BankAccount object is created and stored in memory.

2. ***Transaction Processing Module:*** This system allows users to deposit and withdraw money. Withdrawals shall only be permitted if sufficient funds are available.
    - ***Input :*** Transaction amount.
    - ***Output:*** Updated account balance or an error message.

3. ***Information and Calculation Module:*** This system displays all account details, the current balance, and calculate the annual interest based on the account type.
    - ***Input:*** User menu selection.
    - ***Output:*** Formatted account information, balance, or interest calculation results.

### *Non-Functional Requirements:*

1. ***Usability:*** The system have features like intuitive, numbered console menu that helps users to navigate the system with basic computer literacy.

2. ***Reliability:*** The system handles invalid user inputs (e.g., negative amounts, non-numeric values) gracefully without crashing, providing informative error messages.

3. ***Performance:*** In this system operations (deposit, withdraw, interest calculation) executes properly and provide feedback to the user in real-time.

4. ***Maintainability:*** The code is modular due to multilevel inheritance structure, with clear comments and meaningful variable names to facilitate future enhancements.

## *System Architecture:*

The system follows a Layered Architecture built through Multilevel Inheritance.

- **Presentation Layer:** The bankingMenu() method in the BankWithInterest class handles all user interaction via the console.
- **Business Logic Layer:** The BankingOperations and BankWithInterest classes contain the core logic for transactions and interest calculations.
- **Data Layer:** The BankAccount base class acts as the data model, holding the state of the bank account.

## *Design Diagrams:*

https://drive.google.com/file/d/1176gSrphsjOSp3HtTQzOjTv-b-hNKCJQ/view?usp=sharing

## *Design Decisions & Rationale:*

- **Multilevel Inheritance:** Chosen to model a natural "is-a" relationship. A BankWithInterest *is-a* BankingOperations which in turn *is-a* BankAccount. This promotes code reusability.
- **Encapsulation:** All instance variables are declared as protected to allow access to subclasses while restricting access from unrelated external classes.
- **Scanner Object:** Placed in the base class BankAccount so that all derived classes can inherit and use the same Scanner object for input, avoiding resource duplication.

## *Implementation Details:*

The system is implemented in a single .java file for simplicity, but the classes are logically separated. Key implementation aspects include:

- The BankWithInterest constructor dynamically sets the interest rate, demonstrating polymorphism in initialization.
- Input validation in deposit() and withdraw() methods ensures data integrity.
- The bankingMenu() method uses a do-while loop to provide a continuous user experience.

***Screenshots / Results:***

```
PS D:\JAVA>  & 'C:\Program Files\Java\jdk-25\bin\java.exe
e\User\workspaceStorage\94384c319114001cf83b609bef89f49d\
=== BANK MANAGEMENT SYSTEM ===
Enter Account Number: 9345782312
Enter Name: Ram Kumar
Enter Age: 31
Enter Gender (M/F/O): M
Enter Account Type (Savings/Current): Saving
Enter Initial Balance: 50000

=== BANKING MENU ===
1. Display Account Details
2. Display Balance
3. Deposit Money
4. Withdraw Money
5. Calculate Annual Interest
6. Exit
Enter your choice: 1

=== ACCOUNT DETAILS ===
Account Number: 9345782312
Name: Ram Kumar
Age: 31
Gender: M
Account Type: Saving
Balance: ?50000.0

=== BANKING MENU ===
1. Display Account Details
2. Display Balance
3. Deposit Money
4. Withdraw Money
5. Calculate Annual Interest
6. Exit
Enter your choice: 3
```

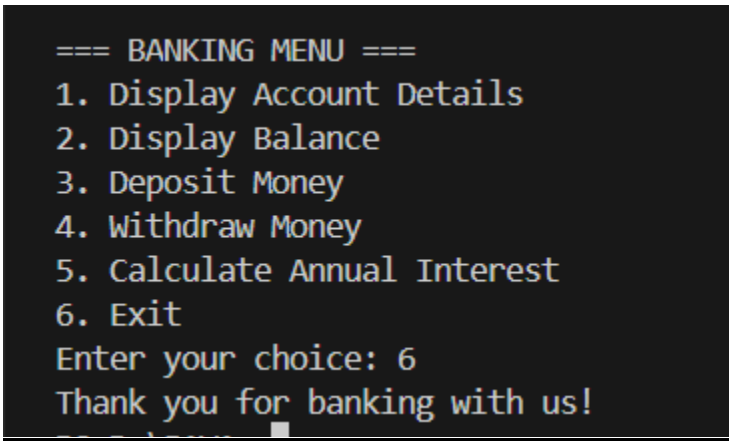
```
Enter amount to deposit: -100
Invalid amount! Please enter a positive value.

=== BANKING MENU ===
1. Display Account Details
2. Display Balance
3. Deposit Money
4. Withdraw Money
5. Calculate Annual Interest
6. Exit
Enter your choice: 4

Enter amount to withdraw: 55000
Insufficient balance!
Current Balance: 50000.0

=== BANKING MENU ===
1. Display Account Details
2. Display Balance
3. Deposit Money
4. Withdraw Money
5. Calculate Annual Interest
6. Exit
Enter your choice: 5

=== ANNUAL INTEREST CALCULATION ===
Account Type: Saving
Annual Interest Rate: 2.5%
Current Balance: ?50000.0
Annual Interest: ?1250.00
Total after 1 year: ?51250.00
```




## ***Testing Approach:***

manual black-box testing approach was used, focusing on the functional requirements.

- Test Case 1 (Account Creation): Verified that all entered details were stored and displayed correctly.
- Test Case 2 (Invalid Deposit): Entered -100. Result: System displayed "Invalid amount!" as expected.
- Test Case 3 (Over-Withdrawal): Tried to withdraw 55000 when balance was 50000. Result: System correctly showed "Insufficient balance!".
- Test Case 4 (Interest Rate): Created a Savings account with 550000 balance. Result: Calculated interest was ₹1250.00, confirming the 2.5% rate

## ***Challenges Faced:***

- Challenge 1: Managing the Scanner object to avoid InputMismatchException due to newline characters.
    - Solution: Used scanner.nextLine() after reading numeric values to consume the leftover newline character.
- Challenge 2: Designing the class hierarchy to logically separate concerns.
    - Solution: Decided to put transaction logic in BankingOperations and interest-specific logic in BankWithInterest, which felt natural.

## ***Learnings & Key Takeaways:***

- Gained a practical understanding of how multilevel inheritance works in a real-world project.

- Reinforced concepts of method overriding and the use of protected access modifiers.
- Improved skills in handling user input in console applications and implementing basic input validation and error handling.

## ***Future Enhancements:***

- Implement a collection (like an ArrayList) to manage multiple bank accounts instead of just one.
- Add a password-based authentication system for security.
- Replace the console interface with a Graphical User Interface (GUI) using Swing or JavaFX.
- Integrate with a database (e.g., MySQL) for persistent storage of account data.
- Implement fund transfer functionality between two accounts.

## ***References:***


- Oracle Java
  Documentation: https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html
- GeeksforGeeks - Multilevel Inheritance in Java.