ATM MANAGEMENT SYSTEM IN JAVA

PROJECT REPORT

Submitted by

Joshua D.	231001079
Karthipan G	231001084
Kavitamizkumaran A	231001086

DEPARTMENT OF IT

In partial fulfilment of the requirement for the award of

B.TECH IN INFORMATION TECHNOLOGY

NOVEMBER 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

RAJALAKSHMI ENGINEERING COLLEGE

(An AUTONOMOUS Institution Affiliated to ANNA UNIVERSITY

CHENNAI - 602105

This is to certify that the Project report entitled

ATM MANAGEMENT SYSTEM IN JAVA

has been submitted by

Joshua D 231001079
Karthipan G 231001084
Kavitamizkumaran 231001086

In partial fulfilment for the award of

DEPARTMENT OF INFORMATION TECHNOLOGY

during the academic year 2024-2025

Faculty guide   Head of the Department

Certified that the candidate was examined by us in the Mini Project work viva-voce examination held on…………………

Internal Examiner   External Examiner

ATM MANAGEMENT SYSTEM IN JAVA

Overview

An ATM Management System is a simulation of an ATM machine that allows users to interact with an ATM interface to perform common banking transactions like checking account balance, withdrawing funds, depositing money, changing PIN, and more. The system must manage customer data, validate credentials, and handle various banking operations.

This project focuses on implementing an ATM Management System in Java, using object-oriented principles. We will simulate the ATM machine with options for both customers and administrators. Below is a breakdown of the key features, system components, and implementation details.

Objective:

The goal of the ATM Management System is to model the operations of a real-world ATM machine. The system provides users with the ability to authenticate themselves, interact with their bank account, and perform financial transactions. It also enables administrators to manage user accounts and view account details. The system is designed to be secure, ensuring that only authorized users can access sensitive data or perform transactions.

Key Features:

1. User Authentication:
    - The system allows customers to authenticate themselves using a unique account number and PIN. If the credentials are valid, they are granted access to their account and can proceed with banking operations. This authentication is a crucial feature for security and preventing unauthorized access.
2. Account Management:
    - Customers can check their current balance, which is stored and updated in the system whenever a transaction (deposit/withdrawal) occurs.

- The system allows customers to withdraw money, ensuring that the amount requested does not exceed the available balance.
- It also supports deposits, where customers can add funds to their account.
- Customers can change their PIN after logging in, enhancing security by allowing the user to update their credentials periodically.

3. Admin Management:
    - The admin has the ability to manage customer accounts. The system allows an administrator to create new accounts, remove accounts, and view account details for troubleshooting or oversight purposes.
    - Admin functionality is crucial for account maintenance and ensuring that the ATM system stays up-to-date with customer data.
4. Transaction History:
    - The system logs each transaction made by the user (withdrawals, deposits, balance checks). This transaction history can be helpful for auditing and troubleshooting purposes.

5. Security:

   - Security is a top priority in the ATM Management System. The system ensures that all user interactions are authenticated, and sensitive information like PINs is handled securely.

   - The system can simulate basic security checks, such as limiting the number of failed login attempts before the account is temporarily locked or flagged.

Design Considerations:

1. Class Structure:

   - Account Class: Manages user account details (account number, PIN, balance). It contains methods for validating the PIN, checking balance, withdrawing, and depositing funds.

   - Customer Class: Manages customer actions, including login, balance checking, withdrawals, deposits, and PIN changes. It communicates directly with the Account class to perform these tasks.

- Admin Class: Manages the overall user accounts in the system. It can add, remove, and view customer accounts. It maintains a collection (e.g., HashMap or ArrayList) of all accounts.
- ATMSystem Class: This is the main interface that simulates the ATM machine. It handles user input, displays menus, and delegates tasks to the Customer and Admin classes.

2. Data Storage:
    - While the ATM system in this case study uses in-memory data storage (e.g., using HashMap or ArrayList), a real-world ATM system would typically use a database to store customer information and transaction history.
    - In this project, account data is stored in a HashMap where the account number is the key, and the Account object is the value. This allows for quick lookup and manipulation of account details.
3. Error Handling and Validation:

- The system includes checks for invalid inputs, such as incorrect PINs or insufficient balance during withdrawals. Error messages guide the user to retry or correct their actions.
- For example, when a customer tries to withdraw more money than they have, the system will display an error message and prevent the transaction.

4. Extensibility:
    - The design allows for future expansion. For instance, additional features like transferring funds between accounts, viewing transaction history, or adding multi-language support can be incorporated easily by extending the classes and adding new methods.
    - The system could also be connected to an actual database in a production environment to persist data, making it more realistic and functional for large-scale usage.
5. User Interface:
    - The user interface is command-line-based (CLI) in this project. It prompts the user to enter their account number, PIN, and select options from the menu to perform various actions.

    - For a real-world project, this could be expanded to a graphical user interface (GUI) using frameworks like JavaFX or Swing, or even developed as a web application using Java with frameworks like Spring Boot.

6. Security Mechanisms:
    - PIN encryption: In a more advanced version of this project, the system could store PINs in an encrypted format (using hashing algorithms like SHA-256) to improve security.
    - Session management: For multi-session or multi-user environments, implementing a session timeout or automatic logout after inactivity would be an important security feature.
    - Failed login attempts: The system could be enhanced to track failed login attempts and temporarily lock the account or enforce CAPTCHA-like mechanisms after a certain number of failed attempts.

Example Workflow:

1. The customer enters their account number and PIN to log in.

2. If the credentials are valid, the system provides a menu with options like checking balance, withdrawing money, depositing funds, etc.
3. The customer selects an option, and the corresponding method is called (e.g., withdraw funds).
4. The system processes the transaction (e.g., checking if the balance is sufficient for withdrawal) and displays the result.
5. After completing their transaction, the customer can choose to log out or perform another operation.
6. The admin can log in separately to manage customer accounts, such as adding new accounts, viewing details, or deleting accounts.

The ATM Management System project in Java is designed to simulate the operations of an automated teller machine (ATM) for banking transactions. The system allows customers to authenticate themselves using their account number and PIN, and then perform various banking activities such as checking balances, withdrawing money, depositing funds, and changing their PIN. The system is structured around multiple classes: Account for managing user account details (like account number, balance, and PIN), Customer for handling customer-related operations such as login and transactions, and Admin for managing customer accounts, including adding, removing, and viewing account details. The main class, ATMSystem, coordinates the entire process, providing a user-friendly interface that interacts with both customers and administrators. It allows customers to carry out operations based on their account status, while administrators can maintain the database of accounts and monitor customer activities. Through objectoriented design, the project demonstrates key concepts like data encapsulation, inheritance, and interaction between objects. The ATM Management System is an ideal case study to understand how object-oriented programming can be used to simulate real-world systems and manage complex user interactions in Java.

Features:

1. User Authentication:
   - Login via Account Number and PIN.
   - Validation of account number and PIN.
2. Customer Transactions:

   - Check Balance: Display the current account balance.
   - Withdraw Money: Deduct the amount from the balance after checking for sufficient funds.
   - Deposit Money: Add a deposit to the balance.
   - Change PIN: Allow the customer to change their PIN after authentication.
3. Admin Management:
   - Add new customer accounts.
   - View customer account details.
   - Remove customer accounts.
4. Transaction History: Record each action performed by the customer (withdraw, deposit, balance check).

Design:

Classes Involved:

1. ATMSystem — Main class that controls user input and interaction with the system.

2. Account — Represents a user's bank account, including attributes like account number, PIN, and balance.
3. Customer — Represents customer information and their banking activities (deposit, withdraw, etc.).
4. Admin — Responsible for managing customers (add, remove, view).
5. Transaction — Represents transaction details such as withdrawal, deposit, etc.

Conclusion:

The ATM Management System case study in Java illustrates how to build a basic yet functional banking system using object-oriented principles. It covers a wide range of essential concepts like class design, inheritance, polymorphism, and data handling. The project not only demonstrates the practical application of Java programming but also provides a blueprint for creating more sophisticated financial systems in the future.