University Institute of Computing (UIC)

Object oriented programming (oops)

Project on:

Simple ATM Simulation

Student name: Adarsh singh

UID: 24BCA10542

Section/Group: 24BCA-3A

Subject Code: 24CAH-201

Submitted to: Ms. Jyoti Rani

Designation : Assistant Professor

1. Abstract

This project titled “Simple ATM Simulation using Object-Oriented Programming in C++” aims to replicate the basic operations of an Automated Teller Machine (ATM) through a console-based application. It allows users to securely log in using an account number and PIN, and perform essential banking functions such as checking balance, depositing money, and withdrawing funds.

The project is developed using key Object-Oriented Programming (OOP) concepts like encapsulation, inheritance, and polymorphism, ensuring code modularity and clarity. To make the system more realistic and persistent, file handling is used for storing and updating user account details. This ensures that user data remains intact even after the program is closed.

Overall, the project demonstrates how OOP principles can be applied to solve real-world problems effectively while also improving understanding of structured program design and file management in C++.

2. Introduction

Automated Teller Machines (ATMs) are widely used banking terminals that allow customers to perform financial transactions without visiting a bank branch. This mini project simulates a basic ATM system using C++ and Object-Oriented Programming principles.

2.1 Project Overview

- The system is designed to simulate the basic operations of an Automated Teller Machine (ATM).
- It allows users to log in using an account number and PIN, check balance, deposit money, withdraw funds, and exit the system.
- User account details are stored in a file to maintain data permanently.
- The project is built using C++ and applies Object-Oriented Programming (OOP) concepts such as classes, objects, and encapsulation.
- File handling is used to manage and update account information securely.
- The system provides a simple and interactive menu-driven interface for easy use.

3. Objective and Scope

The main objective of this project is to design and develop a Simple ATM Simulation System using Object-Oriented Programming (OOP) in C++. The project aims to provide a practical understanding of how OOP principles can be applied to build real-world applications that are structured, efficient, and easy to maintain.

The system focuses on implementing essential ATM functionalities such as secure user login, balance inquiry, cash deposit, withdrawal, and exit operations. It demonstrates the effective use of classes, objects, inheritance, encapsulation, and polymorphism, ensuring modular design and code reusability.

The scope of the project includes:

Implementing user authentication using account number and PIN verification.

Managing user balance and transaction details efficiently.

Performing deposit and withdrawal operations with real-time balance updates.

Maintaining user account data permanently through file handling.

Providing a simple, menu-driven interface for smooth user interaction.

Ensuring data accuracy, security, and reliability within the system.

Overall, this project serves as a practical example of applying OOP and file management techniques to develop a functional, real-world banking simulation.

4. Working of the System

1. The program starts with a welcome message.
2. The user is prompted to enter their account number and PIN.
3. The system verifies credentials using stored file data.
4. Once logged in, users can choose to:
   a. Check Balance
   b. Deposit Money
   c. Withdraw Money
   d. Exit
5. Transactions update the account balance, which is saved

back to the file.
6. The user can exit safely after completing transactions.

5. Code

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Account {
private:
   string accNo, name, pin;
   double balance;

public:
   Account() : accNo(""), name(""), pin(""), balance(0.0) {}

   void createAccount() {
      cout << "\n=== Create New Account ===\n";
      cout << "Enter Account Number: ";
      cin >> accNo;
```

```
    cout << "Enter Name: ";
    cin.ignore();
    getline(cin, name);
    cout << "Set 4-digit PIN: ";
    cin >> pin;
    balance = 0.0;
    cout << "✅ Account created successfully!\n";
  }

  bool login(string acc, string p) {
    return (accNo == acc && pin == p);
  }

  void menu() {
    int choice;
    do {
      cout << "\n=== ATM MENU (" << name << ") ===";
      cout << "\n1. Check Balance\n2. Deposit\n3. Withdraw\n4. Logout\nEnter choice: ";
      cin >> choice;
```

```
        switch (choice) {
            case 1:
                cout << "Your Balance: ₹" << fixed << setprecision(2)
<< balance << endl;
                break;
            case 2:
                deposit();
                break;
            case 3:
                withdraw();
                break;
            case 4:
                cout << "Logging out...\n";
                break;
            default:
                cout << "Invalid choice! Try again.\n";
        }
    } while (choice != 4);
}

void deposit() {
```

```
    double amt;
    cout << "Enter amount to deposit: ";
    cin >> amt;
    if (amt <= 0)
        cout << "Amount must be positive!\n";
    else {
        balance += amt;
        cout << "Deposited ₹" << amt << " successfully!\n";
    }
}

void withdraw() {
    double amt;
    cout << "Enter amount to withdraw: ";
    cin >> amt;
    if (amt > balance)
        cout << "Insufficient balance!\n";
    else if (amt <= 0)
        cout << "Invalid amount!\n";
    else {
```

```
            balance -= amt;
            cout << "Withdrawal successful!\n";
        }
    }

    string getAccNo() { return accNo; }
    string getName() { return name; }
};

int main() {
    Account accounts[5];
    int accountCount = 0;

    cout << "===== WELCOME TO SIMPLE ATM SYSTEM =====\n";

    while (true) {
        int choice;
        cout << "\n1. Create Account\n2. Login\n3. Exit\nEnter choice: ";
        cin >> choice;
```

```
    if (choice == 1) {
        if (accountCount < 5) {
            accounts[accountCount].createAccount();
            accountCount++;
        } else {
            cout << " ❌ Account limit reached (max 5 users)!\n";
        }
    }
    else if (choice == 2) {
        string acc, pin;
        cout << "Enter Account Number: ";
        cin >> acc;
        cout << "Enter PIN: ";
        cin >> pin;

        bool found = false;
        for (int i = 0; i < accountCount; i++) {
            if (accounts[i].login(acc, pin)) {
                cout << "\n✅ Login successful! Welcome, " <<
accounts[i].getName() << "!\n";
```

```
            accounts[i].menu();
            found = true;
            break;
        }
    }
    if (!found)
        cout << " ✗ Invalid login details!\n";
  }
  else if (choice == 3) {
    cout << "Thank you for using ATM System!\n";
    break;
  }
  else {
    cout << "Invalid choice! Try again.\n";
  }
 }

 return 0;
}
```

OUTPUTS:

Output

```
===== WELCOME TO SIMPLE ATM SYSTEM =====

1. Create Account
2. Login
3. Exit
Enter choice: 1

=== Create New Account ===
Enter Account Number: 101
Enter Name: adarsh
Set 4-digit PIN: 1234
✅ Account created successfully!
```

```
1. Create Account
2. Login
3. Exit
Enter choice: 2
Enter Account Number: 101
Enter PIN: 1234

✅ Login successful! Welcome, adarsh!
```

```
=== ATM MENU (adarsh) ===
1. Check Balance
2. Deposit
3. Withdraw
4. Logout
Enter choice: 2
Enter amount to deposit: 5000
Deposited ₹5000 successfully!
```

```
=== ATM MENU (adarsh) ===
1. Check Balance
2. Deposit
3. Withdraw
4. Logout
Enter choice: 3
Enter amount to withdraw: 2000
Withdrawal successful!
```

```
=== ATM MENU (adarsh) ===
1. Check Balance
2. Deposit
3. Withdraw
4. Logout
Enter choice: 1
Your Balance: ₹3000.00
```

7. Challenges Faced

1. Managing file data consistency while updating balances:

Ensuring that user account details such as balance and PIN remained accurate after each transaction was a challenge. Special care was required to prevent data duplication or loss while reading and writing to the file during multiple operations.

2. Handling user input errors and PIN validation securely:

Designing the system to handle incorrect inputs gracefully and verifying the PIN securely was crucial. The challenge was to make the program user-friendly while maintaining security and preventing unauthorized access.

3. Implementing object-oriented features while maintaining simplicity:

Applying OOP concepts such as inheritance and polymorphism without making the program overly complex required careful planning. The goal was to keep the structure simple yet demonstrate clear use of OOP principles.

4. Ensuring data persistence using file handling without database support:

Since the project does not use an external database, all account data had to be managed using text files. Implementing reliable file handling to store, retrieve, and update user details accurately was a significant challenge.

5. Designing a clear and user-friendly interface:

Creating a smooth, menu-driven interface that guides the user through different operations without confusion required thoughtful planning and testing.

6. Testing and debugging transaction operations:
Ensuring that deposit, withdrawal, and balance inquiry functions worked correctly in all possible cases required repeated testing and debugging to eliminate logical and runtime errors.

8. Conclusion

The **Simple ATM Simulation** project demonstrates how **Object-Oriented Programming (OOP)** in **C++** can be effectively used to design a functional and interactive banking system. The use of OOP principles such as **encapsulation, inheritance, and polymorphism** ensures that the code is modular, reusable, and easy to understand.

By incorporating **file handling**, the project achieves data persistence, allowing user account details to be stored and retrieved even after the program is closed. This makes the system more realistic and closer to how actual banking applications operate.

Overall, the project provides a strong foundation for understanding how software systems are structured and how OOP concepts can be applied to solve real-world problems. It also enhances logical thinking, problem-solving, and programming skills, which are essential for developing more advanced applications in the future.