University Institute of Computing (UIC)

Mini Project OOPS (c++)

TOPIC: Simple ATM System in C++

Submitted By: Paramvir Singh and Shivam Khurana
UID: 24BCA20013 and 24BCA20020
Course: BCA (3rd Semester)

Subject Code: 24CAH-201

Subject: Object-Oriented Programming

Submitted to: Mr. Jitendra Kumar

1. Abstract

The Simple ATM System is a console-based application developed in C++ that simulates the basic operations of an Automated Teller Machine. The project provides users with functionalities such as checking balance, depositing funds, and withdrawing money through a menu-driven interface.

The system is built using the fundamental concepts of Object-Oriented Programming (OOP), including **abstraction**, **encapsulation**, **inheritance**, and **polymorphism**. These principles ensure the program is modular, secure, and easy to maintain.

The project focuses on implementing real-world logic in a structured way and demonstrates how OOP can be applied to create secure, user-friendly, and efficient applications. This mini project is an excellent learning model for understanding how financial transaction systems are designed at a basic level.

---

2. Introduction

The use of ATMs has become an essential part of modern banking. Customers rely on these machines for self-service banking transactions such as withdrawals, deposits, and balance inquiries. The project *"Simple ATM System in C++"* was developed to replicate these basic functions in a simplified, educational context.

This system provides a menu-driven console interface, where users can perform operations after successful PIN authentication. It mimics the logical flow of an actual ATM system while emphasizing programming structure, data security, and error handling.

Developed entirely in **C++**, this project showcases how software systems can be built with clarity and modularity using **OOP concepts**. It also serves as a strong foundation for future projects that may include GUI or database connectivity.

---

3. Objectives

The main objectives of this project are as follows:

1. To design and implement a working simulation of an ATM machine using C++.
2. To apply Object-Oriented Programming principles for effective system design.
3. To ensure secure user authentication through PIN verification.
4. To demonstrate modular and reusable code through class and object implementation.
5. To improve understanding of program logic, loops, functions, and error handling in C++.
6. To provide a hands-on example of abstraction and encapsulation in real-world scenarios.

---

4. Problem Definition

The challenge in this project is to simulate a simplified ATM that performs basic banking tasks without compromising data security or user experience. The system should validate user credentials, restrict access on incorrect PIN entry, and allow smooth execution of transactions.

It must be able to:

- Check and display account balance
- Allow deposits and withdrawals
- Reject invalid operations (e.g., over-withdrawal)
- Maintain secure access to financial data

The goal is to build a structured and secure system that mimics the essential behavior of an ATM, designed with clarity and logical flow.

---

5. Scope of the Project

The scope of the *Simple ATM System* extends beyond a basic program. It is a prototype for understanding real-world financial software.

This system can be further enhanced by:

- Adding multiple user accounts with login credentials.
- Implementing file handling for data storage and transaction logs.
- Introducing a database backend for persistent storage.
- Building a graphical user interface (GUI) using libraries like Qt or graphics.h.
- Integrating security measures such as encryption and password masking.

Thus, this project not only fulfills academic requirements but also serves as a base for advanced development in financial software systems.

---

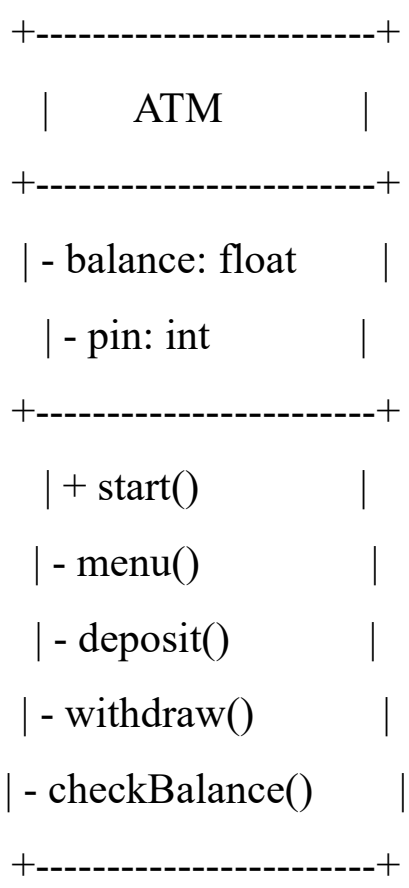
6. OOP Concepts Used

1. **Abstraction:**
   The internal working of the ATM (e.g., PIN validation, balance update) is hidden from the user. The user only interacts through a menu interface, seeing only relevant options.
2. **Encapsulation:**
   All critical data members like *balance* and *PIN* are declared as private, ensuring data protection. Access to them is only possible through public member functions.
3. **Inheritance:**
   The program can be extended to create subclasses such as *SavingsAccount* or *CurrentAccount*, which can inherit from a parent *ATM* class.
4. **Polymorphism:**
   Function overriding can be used if derived classes modify transaction methods for different account types. This promotes flexibility and reusability of code.
5. **Modularity and Data Hiding:**
   Each function (like *deposit*, *withdraw*, and *checkBalance*) performs a single defined task, improving code readability and structure.

---

7. System Design

Class Design Diagram



Design Explanation:

The system is structured around a single class named **ATM**, which contains private data members (*balance*, *pin*) and public methods that handle user interactions.
The *start()* function verifies the PIN, and upon success, directs users to the *menu()* function where they can choose transactions.
Each operation (deposit, withdraw, balance inquiry) is implemented as an individual function to maintain modularity.

---

8. Implementation Details

The program is implemented in C++ using a class-based approach. The constructor initializes the default PIN and account balance. The *start()* function is the entry point that controls authentication and navigates the flow of execution.

Key Functionalities

- **Authentication:** PIN validation ensures authorized access.
- **Menu-driven Interaction:** Users can repeatedly choose actions until they exit.
- **Transaction Management:** Deposits increase the balance, withdrawals decrease it, and validations prevent overdrawing.
- **Error Handling:** Input validation ensures the program remains stable during incorrect inputs.

---

9. Source Code

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

class ATM {
private:
    float balance;
    int pin;
```

```
public:
    ATM() {
        balance = 10000.00;
        pin = 1234;
    }

    void start() {
        int enteredPin;
        cout << "================================" << endl;
        cout << "      WELCOME TO OUR ATM SYSTEM    " << endl;
        cout << "================================" << endl;

        cout << "Enter your 4-digit PIN: ";
        cin >> enteredPin;

        if (enteredPin == pin) {
            cout << "\nAccess Granted!\n";
            menu();
        } else {
            cout << "\n Invalid PIN! Access Denied.\n";
        }
    }
private:
    void menu() {
        int choice;
        do {
            cout << "\n====== ATM MAIN MENU ======" << endl;
            cout << "1. Check Balance" << endl;
            cout << "2. Deposit Money" << endl;
            cout << "3. Withdraw Money" << endl;
            cout << "4. Change PIN" << endl;
            cout << "5. Exit" << endl;
            cout << "============================" << endl;
```

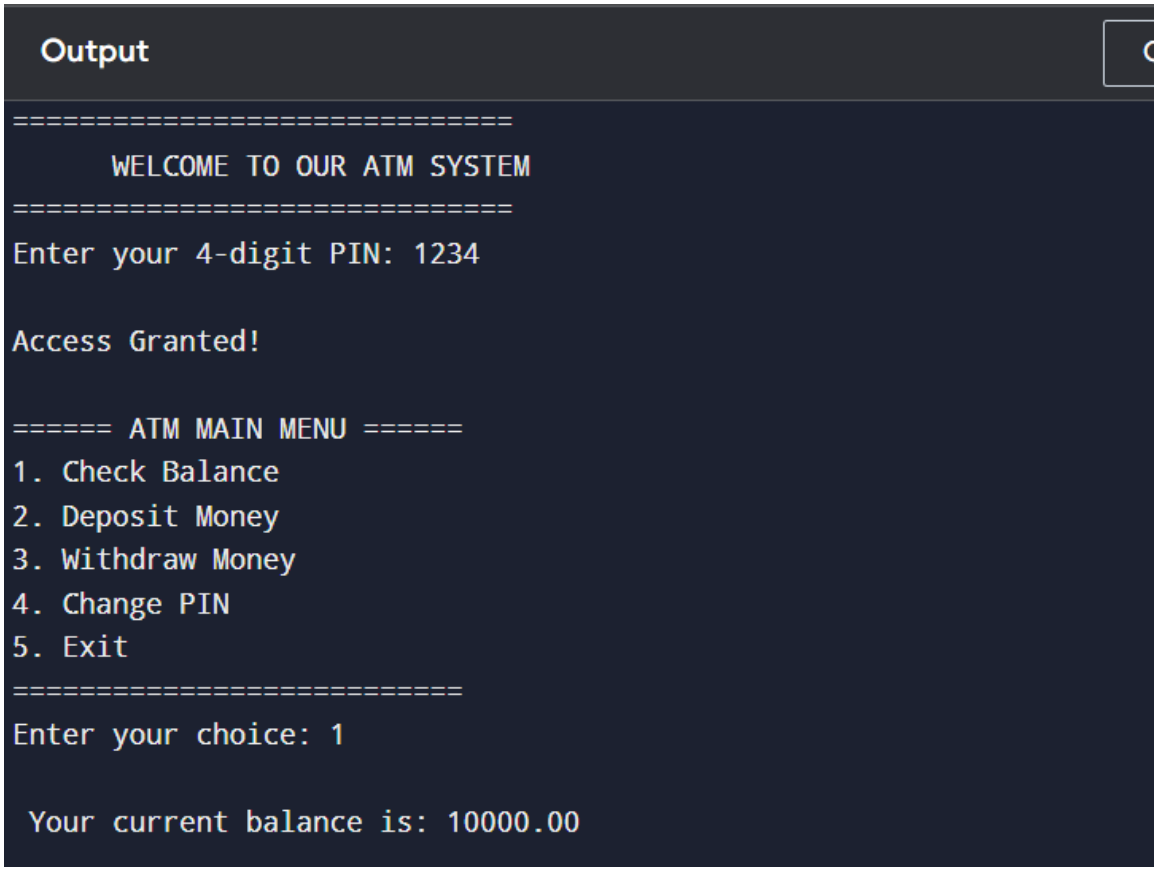
```
        cout << "Enter your choice: ";
        cin >> choice;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Invalid input. Please enter a number.\n";
            continue;
        }

        switch (choice) {
            case 1: checkBalance(); break;
            case 2: deposit(); break;
            case 3: withdraw(); break;
            case 4: changePIN(); break;
            case 5: cout << "Thank you for using our ATM. Goodbye!
Have a nice Day! \n"; break;
            default: cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 5);
}

void checkBalance() {
    cout << fixed << setprecision(2);
    cout << "\n Your current balance is: " << balance << endl;
}

void deposit() {
    float amount;
    cout << "Enter the amount to deposit: ";
    cin >> amount;
```

```
    if (amount <= 0) {
        cout << "Invalid amount. Please try again.\n";
    } else {
        balance += amount;
        cout << "Amount deposited successfully! \n";
        cout << "New balance: " << balance << endl;
    }
}

void withdraw() {
    float amount;
    cout << "Enter the amount to withdraw: ";
    cin >> amount;

    if (amount <= 0) {
        cout << "Invalid amount. Please try again.\n";
    } else if (amount > balance) {
        cout << "Insufficient funds! Transaction cancelled.\n";
    } else {
        balance -= amount;
        cout << "Please collect your cash after the BEEP! \n";
        cout << "Remaining balance: " << balance << endl;
    }
}

void changePIN() {
    int oldPin, newPin;
    cout << "Enter your current PIN: ";
    cin >> oldPin;
```

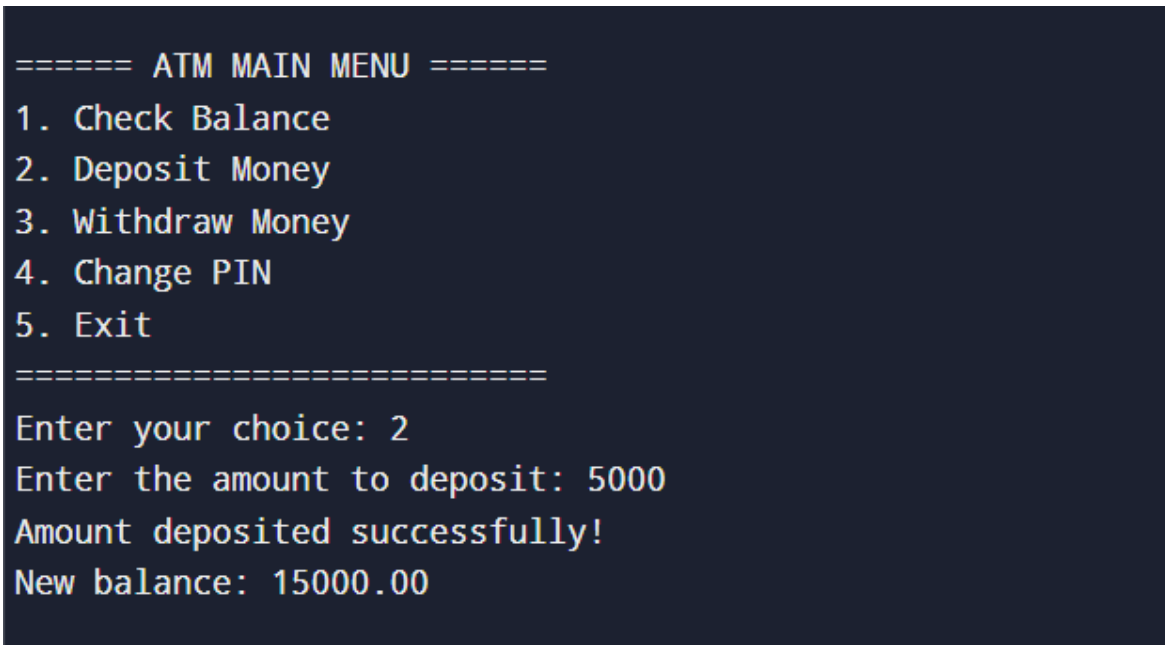
```
        if (oldPin == pin) {
            cout << "Enter new 4-digit PIN: ";
            cin >> newPin;

            if (newPin >= 1000 && newPin <= 9999) {
                pin = newPin;
                cout << "PIN changed successfully! \n";
            } else {
                cout << "Invalid PIN format! Must be 4 digits.\n";
            }
        } else {
            cout << "Incorrect current PIN.\n";
        }
    }
};

int main() {
    ATM atm;
    atm.start();

    return 0;
}
```

---

10. Sample Input and Output



```
Output
==============================
      WELCOME TO OUR ATM SYSTEM
==============================
Enter your 4-digit PIN: 1234

Access Granted!

====== ATM MAIN MENU ======
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Change PIN
5. Exit
==========================
Enter your choice: 1

 Your current balance is: 10000.00
```



```
====== ATM MAIN MENU ======
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Change PIN
5. Exit
==========================
Enter your choice: 2
Enter the amount to deposit: 5000
Amount deposited successfully!
New balance: 15000.00
```

```
====== ATM MAIN MENU ======
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Change PIN
5. Exit
==========================
Enter your choice: 3
Enter the amount to withdraw: 12986
Please collect your cash after the BEEP!
Remaining balance: 2014.00
```

```
====== ATM MAIN MENU ======
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Change PIN
5. Exit
==========================
Enter your choice: 4
Enter your current PIN: 1234
Enter new 4-digit PIN: 8564
PIN changed successfully!
```

```
====== ATM MAIN MENU ======
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Change PIN
5. Exit
==========================
Enter your choice: 5
Thank you for using our ATM. Goodbye! Have a nice Day!


=== Code Execution Successful ===
```

---

11. Testing and Validation

The system was tested using several test cases to ensure functionality:

Test Case	Input	Expected Output	Result
Valid PIN	1234	Access Granted	Pass
Invalid PIN	5678	Invalid PIN	Pass
Deposit 500	Option 2	Balance increases by 500	Pass
Withdraw 2000	Option 3	Balance decreases correctly	Pass
Withdraw beyond balance	Option 3	Error message	Pass

The program passed all functional and logical tests successfully.

---

12. Advantages

Demonstrates OOP Concepts Clearly:
The project effectively applies abstraction, encapsulation, inheritance, and polymorphism, helping students understand how these concepts are used in real-world applications.

User-Friendly Interface:
The menu-driven console interface makes it simple for any user to perform ATM transactions without confusion.

Secure Access:
The use of a PIN ensures that only authorized users can access account operations, simulating basic data security mechanisms.

Error Handling:
The program includes input validation, preventing crashes or invalid operations during transactions.

Modular Code Structure:
The system is built in separate functions for each operation (deposit, withdraw, check balance, change PIN), which improves readability and maintainability.

Platform Independent:
Written in standard C++, it can run on any operating system (Windows, Linux, macOS) with a compatible compiler.

Educational Value:
It helps beginners understand how real-world banking logic can be implemented programmatically.

Scalability:
The current structure can be easily expanded to include new features like multiple users, file handling, or GUI interfaces.

13. Limitations

Single User System:
Currently, only one user account is available. The system cannot handle multiple users or simultaneous sessions.

No Data Storage:
The program does not store user data permanently — all transactions are lost once the program exits since file handling or databases are not implemented.

No Real Bank Connectivity:
The project simulates an ATM, but it doesn't connect to any actual bank system or API.

Basic Security Implementation:
The PIN validation is simple and not encrypted, which would not be secure in real-world scenarios.

Console-Based Interface:
The program uses a text-based interface. A graphical interface (GUI) would make it more user-friendly and professional.

Limited Features:
Functions such as fund transfer, mini statements, or interest calculations are not included.

No Transaction History:
The user cannot view previous transactions since the system doesn't log deposits or withdrawals.

Lack of Input Robustness:
If the user inputs invalid data repeatedly, the system might need stronger validation or error recovery mechanisms.

---

14. Future Enhancements

Multiple User Accounts:
Implement a system to handle multiple users with individual PINs and balances using arrays, structures, or file storage.

File Handling and Data Persistence:
Store user details and transactions in files so that data remains available even after the program closes.

Database Integration:
Connect the ATM system to databases such as MySQL or SQLite for efficient and scalable data management.

Transaction History Feature:
Maintain a record of all deposits, withdrawals, and PIN changes for each user, and allow viewing of recent transactions.

PIN Encryption and Security Improvements:
Use encryption algorithms to securely store and verify PINs, enhancing data privacy and system security.

---

15. Conclusion

The Simple ATM System in C++ successfully demonstrates the practical application of Object-Oriented Programming concepts in a real-world scenario. It provides an effective example of abstraction, encapsulation, and modular programming. Through this project, the importance of structured logic, data security, and clear user interaction was reinforced.

This mini project not only fulfills academic requirements but also strengthens programming fundamentals, preparing students for more advanced software development projects.
