# RUSTAMJI INSTITUTE OF TECHNOLOGY

**BSF ACADEMY, TEKANPUR**

**Practical File for**

**CS305 (OOPM)**

**Submitted by**

*Shani Bharadwaj (0902CS231105)*

B.Tech. Computer Science & Engineering 3rd Semester

(2023-2027 batch)

**File Checked by**

Mr. Vivek Gupta

# Self-Declaration Certificate

I, **Shani Bharadwaj**, hereby declare that I have completed the lab work of CS305 (OOPM) at my own effort and understanding.

I affirm that the work submitted is my own, and I take full responsibility for its authenticity and originality.

Date:

Shani Bharadwaj

0902CS231105

## ENVORIONMENT USED

|  |  |  |
|---|---|---|
| **Hardware Configuration** | : | Device name - DESKTOP-B8U8S9F |

Processor    Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz   1.70 GHz

Installed RAM - 4.00 GB

Device ID - 0E57302F-F83F-4891-AC96-616980C9C67D

Product ID - 00331-10000-00001-AA627

System type - 64-bit operating system, x64-based processor

|  |  |  |
|---|---|---|
| **C Compiler** | : | GCC Compiler |
| **User Interface** | : | VS Code |

## GROUP MEMBERS

|  |  |  |
|---|---|---|
| **Member-1** | : | Shani Bharadwaj 0902CS231105 |
|  |  | GitHub Repository URL |
| **Member-2** | : | Riya Joshi 0902CS231084 |
|  |  | GitHub Repository URL |
| **Member-3** | : | Sakshi Bhadoriya 0902CS231091 |
|  |  | GitHub Repository URL |

# ATM MANAGEMENT SYSTEM

## What is an ATM?

ATM stands **for Automated Teller Machine**. It is an electronic device that allows customers to perform basic banking transactions without the need for a bank teller or human assistance. ATMs are typically used for withdrawing cash, but they also provide other services like depositing money, transferring funds between accounts, and checking account balances.

---

## How Does an ATM Work?

The basic functionality of an ATM revolves around the interaction between the user (customer), the machine, and the bank's central system. Here's a breakdown of the operations:

**1. User Authentication**

- Card Insertion: The user inserts a bank card (Debit/Credit/ATM card) into the machine.

- PIN Entry: After the card is inserted, the user is prompted to enter a Personal Identification Number (PIN). This PIN is verified against the information stored in the bank's database. If the PIN is correct, the user is authenticated and can access their account.

**2. Connection to the Bank's System**

- The ATM connects to the bank's central computer system (the core banking system) through a secure communication network, usually the ATM network. This network can be proprietary (e.g., owned by a particular bank) or part of a global network (e.g., Visa, MasterCard, or a third-party processor).

**3. Transaction Request**

- After authentication, the user selects the desired service (e.g., withdrawal, balance inquiry , fund transfer, etc.). The ATM then communicates the request to the bank's central system, passing along relevant account details (without revealing the PIN or other sensitive data).

**4. Transaction Processing**

- Withdrawal: For cash withdrawal, the ATM communicates with the central system to verify that the user's account has enough balance to cover the transaction. If the balance is sufficient, the ATM dispenses the cash.

- Deposit: For deposits, the user inserts cash or checks (depending on the machine's functionality), and the ATM records the transaction after verifying the deposit.

- Balance Inquiry: For balance inquiries, the ATM checks the user's account balance and displays it on the screen.

- Fund Transfer: If the user selects to transfer funds, the ATM sends a request to the central system, which verifies the transfer details (e.g., amount, destination account) and processes the transfer.

**5. Transaction Confirmation**

- After the transaction is processed, the ATM prints a receipt (optional) showing the transaction details, or displays a confirmation message on the screen.

- The machine then ejects the card, and the user can take their cash (if applicable).

**6. Disconnection**

- Once the transaction is completed, the ATM disconnects from the bank's system and returns to the idle state, awaiting the next user.

---

## Types of ATM Transactions

- Cash Withdrawals: The most common function of an ATM. It allows users to withdraw cash from their bank accounts.

- Balance Inquiry: The ATM provides the current balance in the user's account.

- Fund Transfers: Users can transfer money between accounts (either within the same bank or to a different bank, depending on the network).

- Deposits: Some ATMs allow users to deposit cash or checks directly into their account.

- Bill Payments: Some advanced ATMs allow users to pay bills such as utilities or loans.

- Mini-Statements: Users can request a mini statement showing recent transactions.

---

## ATM Components and Operations

1. Card Reader: This is the device that reads the user's card (magnetic stripe or chip). It captures card data and forwards it for verification.

2. Keypad: The user inputs their PIN or other instructions using the ATM's keypad. The input is securely transmitted to the bank's system for processing.

3. Display Screen: The screen displays instructions, prompts for input, and transaction details. It's used for communication between the ATM and the user.

4. Cash Dispenser: This is the machine’s mechanism for dispensing physical currency (notes) when a withdrawal request is made. The dispenser ensures that the correct denomination and quantity of cash is provided.

5. Receipt Printer: After a transaction, the ATM can print a receipt containing details of the transaction (date, amount, account balance, etc.).

6. Deposit Slot (if applicable): Some ATMs allow users to deposit cash or checks directly into their account.

7. Security Features: To prevent unauthorized access and fraud, ATMs include various security measures, including cameras, encryption, anti-skimming devices, and locks.

---

## ATM Security Measures

ATMs are designed with multiple security features to protect users and banks from fraud. Some of these include:

- Encryption: All data transmitted between the ATM and the bank’s system is encrypted to prevent interception.

- PIN Protection: The user’s PIN is not stored in the ATM but is securely transmitted to the bank’s server for verification.

- Anti-Skimming Devices: Many ATMs are equipped with anti-skimming technology to prevent criminals from reading the information on the ATM card.

- Surveillance Cameras: Most ATMs have cameras to monitor activity and prevent theft or fraud.

- Security Alarms: ATMs often have alarms that trigger if the machine is tampered with or vandalized.

- Two-Factor Authentication (in some cases): Advanced ATMs may ask for additional verification like a one-time password (OTP) sent to the user’s phone.

---

## ATM Networks

ATMs can be part of different networks, such as:

- Bank-specific networks: Many banks operate their own ATM networks. For example, an ATM might only serve customers of that particular bank or its partners.

- Shared networks: Some ATMs are part of shared networks that allow customers of various banks to access ATMs beyond their own bank's machines. Examples of shared ATM networks include **Cirrus, Plus, Interac, Pulse, and NYCE.**

- International networks: Major international networks like **Visa** and **MasterCard** offer cross-border ATM access, enabling users to withdraw cash in foreign countries.

## <u>ATM Maintenance and Replenishment</u>

ATMs require regular maintenance to ensure they remain functional and secure:

- Cash Replenishment: Banks or ATM operators replenish the machine with cash as it is withdrawn. This is typically done regularly or when the ATM begins to run low on cash.

- Software Updates: ATMs receive periodic software updates to improve functionality, security, and compliance with new banking regulations.

- Repairs and Diagnostics: If an ATM malfunctions (e.g., if a dispenser jams or a keypad stops working), it must be repaired promptly to minimize downtime.

---

## <u>Benefits of ATMs</u>

- Convenience: ATMs are available 24/7, allowing customers to access banking services outside of normal business hours.

- Accessibility: ATMs provide easy access to cash, especially in remote or underserved areas.

- Speed: Transactions are generally faster than waiting in line for a bank teller.

- Cost-effective: For banks, ATMs reduce the need for human staff to perform simple transactions.

---

## <u>Limitations of ATMs</u>

- Limited Services: While ATMs offer many basic banking services, they can't handle complex requests such as loans or personalized financial advice.

- Security Risks: Although ATMs have security features, they are still susceptible to fraud, card skimming, and other criminal activities.

- Cash Limitations: ATMs usually have a withdrawal limit, meaning you can’t withdraw large sums of money in a single transaction.

---

**Overall, ATMs are an essential part of modern banking infrastructure, offering convenience, security, and access to banking services for millions of customers worldwide.**

## Source Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;
class Account {
private:
string accountNumber;
string accountHolderName;
double balance;
string pin;
public:
Account(string accNum, string accHolder, double initialBalance, string accountPin) {
accountNumber = accNum;
accountHolderName = accHolder;
balance = initialBalance;
pin = accountPin;
}
string getAccountNumber() {
return accountNumber;
}
string getAccountHolderName() {
return accountHolderName;
}
double getBalance() {
return balance;
}
void deposit(double amount) {
balance += amount;
cout << "Deposited: " << amount << endl;
```

```
}
bool withdraw(double amount) {
if (amount > balance) {
cout << "Insufficient balance!" << endl;
return false;
}
balance -= amount;
cout << "Withdrawn: " << amount << endl;
return true;
}
void changePin(string newPin) {
pin = newPin;
cout << "PIN changed successfully!" << endl;
}
bool validatePin(string inputPin) {
return pin == inputPin;
}
void displayAccountDetails() {
cout << "Account Number: " << accountNumber << endl;
cout << "Account Holder Name: " << accountHolderName << endl;
cout << "Current Balance: " << balance << endl;
}
};
class Transaction {
private:
string accountNumber;
string transactionType;
double amount;
public:
Transaction(string accNum, string type, double amt) {
accountNumber = accNum;
```

```
transactionType = type;
amount = amt;
}
void logTransaction() {
ofstream outFile("transactions.txt", ios::app);
if (outFile.is_open()) {
outFile << "Account Number: " << accountNumber << ", "
<< "Transaction Type: " << transactionType << ", "
<< "Amount: " << amount << endl;
outFile.close();
} else {
cout << "Unable to open file!" << endl;
}
}
};
class ATM {
private:
vector<Account> accounts;
bool isAccountCreated = false;
public:
void createAccount(string accNum, string accHolder, double initialBalance, string
accountPin) {
Account newAccount(accNum, accHolder, initialBalance, accountPin);
accounts.push_back(newAccount);
isAccountCreated = true;
cout << "Account created successfully!" << endl;
}
Account* findAccount(string accNum) {
for (auto& account : accounts) {
if (account.getAccountNumber() == accNum) {
return &account;
}
```

```
}
return nullptr;
}
bool checkAccountCreated() {
return isAccountCreated;
}
void deposit(string accNum, double amount) {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
Account* account = findAccount(accNum);
if (account) {
account->deposit(amount);
Transaction transaction(accNum, "Deposit", amount);
transaction.logTransaction();
} else {
cout << "Account not found!" << endl;
}
}
void withdraw(string accNum, double amount) {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
Account* account = findAccount(accNum);
if (account) {
if (account->withdraw(amount)) {
Transaction transaction(accNum, "Withdraw", amount);
transaction.logTransaction();
}
```

```
} else {
cout << "Account not found!" << endl;
}
}
void checkBalance(string accNum) {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
Account* account = findAccount(accNum);
if (account) {
cout << "Current Balance: " << account->getBalance() << endl;
} else {
cout << "Account not found!" << endl;
}
}
void viewTransactionHistory() {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
ifstream inFile("transactions.txt");
string line;
if (inFile.is_open()) {
while (getline(inFile, line)) {
cout << line << endl;
}
inFile.close();
} else {
cout << "Unable to open file!" << endl;
}
```

```
}
void transferFunds(string fromAccNum, string toAccNum, double amount) {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
Account* fromAccount = findAccount(fromAccNum);
Account* toAccount = findAccount(toAccNum);
if (fromAccount && toAccount) {
if (fromAccount->withdraw(amount)) {
toAccount->deposit(amount);
Transaction transaction(fromAccNum, "Transfer to " + toAccNum, amount);
transaction.logTransaction();
cout << "Transfer successful!" << endl;
}
} else {
cout << "One or both accounts not found!" << endl;
}
}
void deleteAccount(string accNum) {
if (!checkAccountCreated()) {
cout << "Please create an account first!" << endl;
return;
}
for (auto it = accounts.begin(); it != accounts.end(); ++it) {
if (it->getAccountNumber() == accNum) {
accounts.erase(it);
cout << "Account deleted successfully!" << endl;
return;
}
}
```

```
cout << "Account not found!" << endl;
}
};
int main() {
ATM atm;
int choice;
string accNum, accHolder, accountPin, toAccNum;
double amount;
while (true) {
cout << "ATM Management System" << endl;
cout << "1. Create Account" << endl;
cout << "2. Deposit" << endl;
cout << "3. Withdraw" << endl;
cout << "4. Check Balance" << endl;
cout << "5. View Transaction History" << endl;
cout << "6. Transfer Funds" << endl;
cout << "7. Change PIN" << endl;
cout << "8. Delete Account" << endl;
cout << "9. View Account Details" << endl;
cout << "10. Exit" << endl;
cout << "Enter your choice: ";
cin >> choice;
switch (choice) {
case 1: {
cout << "Enter Account Number (10 digits): ";
cin >> accNum;
cout << "Enter Account Holder Name: ";
cin >> accHolder;
cout << "Enter Initial Balance: ";
cin >> amount;
cout << "Enter Account PIN (4 digits): ";
cin >> accountPin;
atm.createAccount(accNum, accHolder, amount, accountPin);
break;
}
case 2: {
cout << "Enter Account Number: ";
cin >> accNum;
cout << "Enter Amount to Deposit: ";
cin >> amount;
atm.deposit(accNum, amount);
break;
}
case 3: {
cout << "Enter Account Number: ";
cin >> accNum;
cout << "Enter Amount to Withdraw: ";
cin >> amount;
atm.withdraw(accNum, amount);
break;
}
case 4: {
cout << "Enter Account Number: ";
cin >> accNum;
atm.checkBalance(accNum);
break;
}
case 5: {
atm.viewTransactionHistory();
break;
}
```

```
case 6: {
cout << "Enter Your Account Number: ";
cin >> accNum;
cout << "Enter Recipient Account Number: ";
cin >> toAccNum;
cout << "Enter Amount to Transfer: ";
cin >> amount;
atm.transferFunds(accNum, toAccNum, amount);
break;
}
case 7: {
cout << "Enter Account Number: ";
cin >> accNum;
cout << "Enter New PIN: ";
cin >> accountPin;
Account* account = atm.findAccount(accNum);
if (account) {
account->changePin(accountPin);
} else {
cout << "Account not found!" << endl;
}
break;
}
case 8: {
cout << "Enter Account Number: ";
cin >> accNum;
atm.deleteAccount(accNum);
break;
}
```

```
case 9: {
cout << "Enter Account Number: ";
cin >> accNum;
Account* account = atm.findAccount(accNum);
if (account) {
account->displayAccountDetails();
} else {
cout << "Account not found!" << endl;
}
break;
}
case 10: {
cout << "Exiting..." << endl;
return 0;
}
default: {
cout << "Invalid choice! Please try again." << endl;
break;
                }
            }
      }
}
```

## OUTPUT:

```
ATM Management System
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
5. View Transaction History
6. Transfer Funds
7. Change PIN
8. Delete Account
9. View Account Details
10. Exit
Enter your choice: 2
Please create an account first!
```

## EXPLANATION

The program simulates an ATM system that allows users to create an account, deposit and withdraw money, transfer funds, view transaction history, and manage account details. It uses object-oriented principles like classes, constructors, and member functions to encapsulate the behaviour of the ATM, accounts, and transactions. The ATM class coordinates the operations and maintains the accounts.

- **Account Class**

```
class Account {
private:
   string accountNumber;
   string accountHolderName;
   double balance;
   string pin;

public:
   Account(string accNum, string accHolder, double initialBalance, string
accountPin) {
      accountNumber = accNum;
      accountHolderName = accHolder;
      balance = initialBalance;
      pin = accountPin;
   }
```

### Data Members:

accountNumber: Stores the account number (string).

accountHolderName: Stores the name of the account holder.

balance: Stores the current balance of the account (double).

pin: Stores the account PIN (string).

Constructor: Initializes the account with account number, holder name, initial balance, and PIN.

```
    string getAccountNumber() { return accountNumber; }
    string getAccountHolderName() { return accountHolderName; }
    double getBalance() { return balance; }
```

Getter Methods: These methods return the values of account number, account holder name, and balance. These are useful to access these details from outside the class.

```
 void deposit(double amount) {
        balance += amount;
        cout << "Deposited: " << amount << endl;
    }
```

Deposit Function: This function adds the specified amount to the account balance.

```
 bool withdraw(double amount) {
        if (amount > balance) {
            cout << "Insufficient balance!" << endl;
            return false;
        }
        balance -= amount;
        cout << "Withdrawn: " << amount << endl;
        return true;
    }
```

Withdraw Function: This function checks if the withdrawal amount is greater than the available balance. If so, it prints "Insufficient balance!" and returns false. If the balance is sufficient, the amount is subtracted from the balance and the transaction is successful.

```
    void changePin(string newPin) {
        pin = newPin;
        cout << "PIN changed successfully!" << endl;
    }
```

Change PIN Function: This function changes the account PIN to a new one.

```
bool validatePin(string inputPin) {
      return pin == inputPin;
  }
```

Validate PIN Function: This function checks if the entered PIN matches the stored PIN. It returns true if they match, otherwise false.

```
  void displayAccountDetails() {
      cout << "Account Number: " << accountNumber << endl;
      cout << "Account Holder Name: " << accountHolderName << endl;
      cout << "Current Balance: " << balance << endl;
  }
```

Display Account Details: This function prints the account's details, including the account number, holder name, and balance.

```
class Transaction {
private:
  string accountNumber;
  string transactionType;
  double amount;


public:
  Transaction(string accNum, string type, double amt) {
      accountNumber = accNum;
      transactionType = type;
      amount = amt;
  }
```

Data Members:

accountNumber: The account number involved in the transaction.

transactionType: The type of the transaction (e.g., deposit, withdraw).

amount: The amount involved in the transaction.

Constructor: Initializes the transaction with the account number, transaction type (deposit/withdraw), and the transaction amount.

```
  void logTransaction() {
    ofstream outFile("transactions.txt", ios::app);
    if (outFile.is_open()) {
      outFile << "Account Number: " << accountNumber << ", "
          << "Transaction Type: " << transactionType << ", "
          << "Amount: " << amount << endl;
      outFile.close();
    } else {
      cout << "Unable to open file!" << endl;
    }
  }
```

Log Transaction Function: This function logs the transaction details to the transactions.txt file. It appends the transaction to the file. If the file cannot be opened, it prints an error message.

```
class ATM {
private:
  vector<Account> accounts;
  bool isAccountCreated = false;
public:
  void createAccount(string accNum, string accHolder, double initialBalance,
string accountPin) {
    Account newAccount(accNum, accHolder, initialBalance, accountPin);
    accounts.push_back(newAccount);
    isAccountCreated = true;
    cout << "Account created successfully!" << endl;
  }
```

accounts: A vector to store the accounts. Each account is an instance of the Account class.

isAccountCreated: A boolean flag that tracks if any account has been created. Initially, it's set to false.

Create Account Function: This function creates a new Account and adds it to the accounts vector. It also sets the isAccountCreated flag to true.

```
Account* findAccount(string accNum) {
    for (auto& account : accounts) {
        if (account.getAccountNumber() == accNum) {
            return &account;
        }
    }
    return nullptr;
}
```

Find Account Function: This function takes an account number and searches for it in the accounts vector. If the account is found, it returns a pointer to the Account object; otherwise, it returns nullptr.

```
bool checkAccountCreated() {
    return isAccountCreated;
}
```

Check if Account is Created: This function checks if an account has been created using the isAccountCreated flag. It returns true if an account is created, otherwise false.

## ➢ Operations

**Deposit**: The deposit function checks if an account is created first. If no account is created, it prompts the user to create an account. Otherwise, it finds the account and deposits the specified amount.

**Withdraw**: Similar to the deposit, the withdraw function ensures that an account exists before attempting the withdrawal.

**Check Balance**: This function checks the balance of the account, but only if an account has been created.

**Transfer Funds**: Transfers funds between two accounts by first checking if both accounts exist and if the source account has sufficient balance.

**View Transaction History**: This function reads and displays the transaction history from the transactions.txt file.

**Delete Account**: Deletes the specified account from the accounts vector if it exists.

## ➢ Main Function

```
while (true) {
    cout << "ATM Management System" << endl;
    cout << "1. Create Account" << endl;
    cout << "2. Deposit" << endl;
    cout << "3. Withdraw" << endl;
    cout << "4. Check Balance" << endl;
    cout << "5. View Transaction History" << endl;
    cout << "6. Transfer Funds" << endl;
    cout << "7. Change PIN" << endl;
    cout << "8. Delete Account" << endl;
    cout << "9. View Account Details" << endl;
    cout << "10. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;
```

This is the main menu where the user selects an operation by entering a number.

```
    switch (choice) {
        case 1: {
            // Creating account
            cout << "Enter Account Number (10 digits): ";
            cin >> accNum;
            cout << "Enter Account Holder Name: ";
            cin >> accHolder;
            cout << "Enter Initial Balance: ";
            cin >> amount;
            cout << "Enter Account PIN (4 digits): ";
            cin >> accountPin;
            atm.createAccount(accNum, accHolder, amount, accountPin);
            break;}
```

Case 1: Creates a new account and stores the account details entered by the user.

```
case 2: {
            // Deposit
            cout << "Enter Account Number: ";
            cin >> accNum;
            cout << "Enter Amount to Deposit: ";
            cin >> amount;
            atm.deposit(accNum, amount);
            break;
        }
```

Case 2: Allows the user to deposit money into their account. It prompts for the account number and amount, then performs the deposit.

The other cases (3 to 9) perform similar tasks for withdrawal, checking balance, viewing transaction history, transferring funds, changing PIN, deleting an account, and displaying account details.

```
        case 10: {
            cout << "Exiting..." << endl;
            return 0;
        }
```

Case 10: Exits the program.