Bahria University, Islamabad

Department of Software Engineering

Software construction Lab (Fall-2025)

Instructor: Engr. Saad

Student: Haseeb Irfan

Enrollment: 01-131232-029

Lab Journal: open ended

Date: 10/15/25

Task No:	Task Wise Marks		Documentation Marks		Total Marks (20)
	Assigned	Obtained	Assigned	Obtained	
1					
2					
3					
4					
5					

Comments:

Signature

Open Ended Task: ATM

ATM Class:

```
import java.util.Scanner;

public class ATM {
    private Bank bank;
    private Scanner scanner;
    private Customer currentCustomer;
    private BankAccount currentAccount;

    public ATM(Bank bank) {
        this.bank = bank;
        this.scanner = new Scanner(System.in);
    }

    public void start() {
        System.out.println("=== WELCOME TO THE ATM SYSTEM ===");

        while (true) {
            System.out.println("\n1. Customer Login");
            System.out.println("2. Register New Account");
            System.out.println("3. Administrator Login");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");

            int choice = getIntInput();

            switch (choice) {
                case 1:
                    customerLogin();
                    break;
                case 2:
                    registerNewAccount();
                    break;
                case 3:
                    adminLogin();
                    break;
                case 4:
                    System.out.println("Thank you for using our ATM.
Goodbye!");
                    return;
                default:
                    System.out.println("Invalid choice. Please try
again.");
            }
        }
    }

    private void registerNewAccount() {
        System.out.println("\n=== REGISTER NEW ACCOUNT ===");

        // Get customer information
        System.out.print("Enter your full name: ");
        String name = scanner.nextLine().trim();

        if (name.isEmpty()) {
            System.out.println("Name cannot be empty.");
            return;
        }

        // Generate unique customer ID
        String customerId = generateCustomerId();
        System.out.println("Your generated Customer ID: " +
customerId);

        // Get PIN
        String pin;
        while (true) {
            System.out.print("Create a 4-digit PIN: ");
            pin = scanner.nextLine().trim();
            if (pin.matches("\\d{4}")) {
                break;
            } else {
                System.out.println("PIN must be exactly 4 digits.");
            }
        }

        // Confirm PIN
        System.out.print("Confirm your PIN: ");
        String confirmPin = scanner.nextLine().trim();
        if (!pin.equals(confirmPin)) {
            System.out.println("PINs do not match. Registration
cancelled.");
            return;
        }

        // Select account type
        System.out.println("\nSelect account type:");
        System.out.println("1. Savings Account (Minimum balance:
$100)");
        System.out.println("2. Checking Account (Overdraft limit:
$500)");
        System.out.print("Choose account type (1-2): ");

        int accountTypeChoice = getIntInput();
        String accountType;
        double minInitialBalance;

        if (accountTypeChoice == 1) {
            accountType = "SAVINGS";
            minInitialBalance = 100.0;
        } else if (accountTypeChoice == 2) {
            accountType = "CHECKING";
            minInitialBalance = 0.0;
        } else {
            System.out.println("Invalid choice. Registration
cancelled.");
            return;
        }

        // Get initial deposit
        double initialDeposit;
        while (true) {
            System.out.printf("Enter initial deposit (minimum $%.2f):
$", minInitialBalance);
            initialDeposit = getDoubleInput();
            if (initialDeposit >= minInitialBalance) {
                break;
            } else {
                System.out.printf("Initial deposit must be at least
$%.2f\n", minInitialBalance);
            }
        }

        // Generate account number
        String accountNumber = generateAccountNumber(accountType);

        // Create customer and account
        try {
            Customer newCustomer = new Customer(customerId, name,
pin);
            bank.addCustomer(newCustomer);

            BankAccount newAccount = bank.createAccount(accountType,
accountNumber, customerId, initialDeposit);

            if (newAccount != null) {
                System.out.println("\n✓ REGISTRATION SUCCESSFUL!");
                System.out.println("Customer ID: " + customerId);
                System.out.println("Account Number: " +
accountNumber);
                System.out.println("Account Type: " + accountType);
                System.out.println("Initial Balance: $" +
String.format("%.2f", initialDeposit));
                System.out.println("\nPlease remember your Customer
ID and PIN for future logins.");

                // Ask if user wants to login immediately
                System.out.print("\nWould you like to login now?
(yes/no): ");
                String loginChoice =
scanner.nextLine().trim().toLowerCase();
                if (loginChoice.equals("yes") ||
loginChoice.equals("y")) {
                    currentCustomer = newCustomer;
                    currentAccount = newAccount;
                    showCustomerMenu();
                }
            } else {
                System.out.println("Registration failed. Please try
again.");
            }
        } catch (Exception e) {
            System.out.println("Error during registration: " +
e.getMessage());
        }
    }

    private void customerLogin() {
        System.out.print("Enter Customer ID: ");
        String customerId = scanner.nextLine();
        System.out.print("Enter PIN: ");
        String pin = scanner.nextLine();

        Customer customer = bank.getCustomer(customerId);
        if (customer != null && customer.verifyPin(pin)) {
            currentCustomer = customer;
            System.out.println("Login successful! Welcome, " +
```

```java
customer.getName());
            showCustomerMenu();
        } else {
            System.out.println("Login failed. Please check your 
credentials.");
        }
    }

    private void adminLogin() {
        System.out.print("Enter Admin Username: ");
        String username = scanner.nextLine();
        System.out.print("Enter Admin Password: ");
        String password = scanner.nextLine();

        if ("admin".equals(username) && "password".equals(password)) 
{
            System.out.println("Admin login successful!");
            showAdminMenu();
        } else {
            System.out.println("Invalid admin credentials.");
        }
    }

    private void showAdminMenu() {
        while (true) {
            System.out.println("\n--- Bank Administrator Menu ---");
            System.out.println("1. View All Customers");
            System.out.println("2. View All Accounts");
            System.out.println("3. Create New Account");
            System.out.println("4. Unblock Customer ATM Access");
            System.out.println("5. Back to Main Menu");
            System.out.print("Choose an option: ");

            int choice = getIntInput();

            switch (choice) {
                case 1:
                    viewAllCustomers();
                    break;
                case 2:
                    viewAllAccounts();
                    break;
                case 3:
                    createNewAccount();
                    break;
                case 4:
                    unblockCustomer();
                    break;
                case 5:
                    return;
                default:
                    System.out.println("Invalid choice. Please try 
again.");
            }
        }
    }

    private void viewAllCustomers() {
        System.out.println("\n--- All Customers ---");
        for (Customer customer : bank.getAllCustomers()) {
            System.out.println("ID: " + customer.getCustomerId() +
                    " | Name: " + customer.getName() +
                    " | ATM Blocked: " +
customer.isAtmAccessBlocked() +
                    " | Failed Attempts: " +
customer.getFailedLoginAttempts());
            customer.displayAccounts();
            System.out.println();
        }
    }

    private void viewAllAccounts() {
        System.out.println("\n--- All Accounts ---");
        for (BankAccount account : bank.getAllAccounts()) {
            System.out.println("Account: " +
account.getAccountNumber() +
                    " | Type: " + account.getAccountType() +
                    " | Customer: " + account.getCustomerId() +
                    " | Balance: $" + String.format("%.2f",
account.getBalance()) +
                    " | Status: " + account.getStatus());
        }
    }

    private void createNewAccount() {
        System.out.print("Enter Customer ID: ");
        String customerId = scanner.nextLine();
        System.out.print("Enter Account Type (SAVINGS/CHECKING): ");
        String accountType = scanner.nextLine();
        System.out.print("Enter New Account Number: ");
        String accountNumber = scanner.nextLine();
        System.out.print("Enter Initial Balance: $");
        double initialBalance = getDoubleInput();

        bank.createAccount(accountType, accountNumber, customerId,
initialBalance);
    }

    private void unblockCustomer() {
        System.out.print("Enter Customer ID to unblock: ");
        String customerId = scanner.nextLine();

        Customer customer = bank.getCustomer(customerId);
        if (customer != null) {
            if (customer.isAtmAccessBlocked()) {
                customer.unblockATMAccess();
            } else {
                System.out.println("Customer is not blocked.");
            }
        } else {
            System.out.println("Customer not found.");
        }
    }

    private void showCustomerMenu() {
        while (currentCustomer != null) {
            selectAccount();
            if (currentAccount == null) {
                break;
            }

            System.out.println("\n--- ATM Menu (" +
currentAccount.getAccountType() + ") ---");
            System.out.println("1. Check Balance");
            System.out.println("2. Deposit Funds");
            System.out.println("3. Withdraw Funds");
            System.out.println("4. View Transaction History");
            System.out.println("5. Transfer Funds");
            System.out.println("6. Select Different Account");
            System.out.println("7. Logout");
            System.out.print("Choose an option: ");

            int choice = getIntInput();

            switch (choice) {
                case 1:
                    checkBalance();
                    break;
                case 2:
                    depositFunds();
                    break;
                case 3:
                    withdrawFunds();
                    break;
                case 4:
                    viewTransactionHistory();
                    break;
                case 5:
                    transferFunds();
                    break;
                case 6:
                    currentAccount = null;
                    break;
                case 7:
                    currentCustomer = null;
                    currentAccount = null;
                    System.out.println("Logged out successfully.");
                    break;
                default:
                    System.out.println("Invalid choice. Please try
again.");
            }
        }
    }

    private void selectAccount() {
        if (currentAccount != null) return;

        java.util.ArrayList<BankAccount> customerAccounts =
currentCustomer.getAllAccounts();
        if (customerAccounts.isEmpty()) {
            System.out.println("No accounts found for this
customer.");
            currentCustomer = null;
            return;
        }

        System.out.println("\n--- Select Account ---");
        for (int i = 0; i < customerAccounts.size(); i++) {
            BankAccount account = customerAccounts.get(i);
            System.out.println((i + 1) + ". " +
account.getAccountNumber() + " - " +
                    account.getAccountType() + " - Balance: $" +
```

```java
                    String.format("%.2f", account.getBalance()));
        }

        System.out.print("Select account (1-" +
customerAccounts.size() + "): ");
        int accountChoice = getIntInput();

        if (accountChoice >= 1 && accountChoice <=
customerAccounts.size()) {
            currentAccount = customerAccounts.get(accountChoice - 1);
            System.out.println("Selected account: " +
currentAccount.getAccountNumber());
        } else {
            System.out.println("Invalid selection.");
        }
    }

    private void checkBalance() {
        System.out.println("Current Balance: $" +
String.format("%.2f", currentAccount.getBalance()));
        printReceipt("BALANCE_INQUIRY", 0,
currentAccount.getBalance());
    }

    private void depositFunds() {
        System.out.print("Enter deposit amount: $");
        double amount = getDoubleInput();

        if (amount > 0) {
            currentAccount.deposit(amount);
            printReceipt("DEPOSIT", amount,
currentAccount.getBalance());
        } else {
            System.out.println("Invalid amount.");
        }
    }

    private void withdrawFunds() {
        System.out.print("Enter withdrawal amount: $");
        double amount = getDoubleInput();

        if (amount > 0) {
            currentAccount.withdraw(amount);
            printReceipt("WITHDRAWAL", amount,
currentAccount.getBalance());
        } else {
            System.out.println("Invalid amount.");
        }
    }

    private void viewTransactionHistory() {
        currentAccount.displayTransactionHistory();
    }

    private void transferFunds() {
        System.out.println("\n--- Transfer Funds ---");
        System.out.println("1. Transfer to my other account");
        System.out.println("2. Transfer to another customer");
        System.out.print("Choose transfer type: ");

        int transferType = getIntInput();
        if (transferType == 1) {
            transferToOwnAccount();
        } else if (transferType == 2) {
            transferToOtherCustomer();
        } else {
            System.out.println("Invalid choice.");
        }
    }

    private void transferToOwnAccount() {
        java.util.ArrayList<BankAccount> customerAccounts =
currentCustomer.getAllAccounts();
        if (customerAccounts.size() < 2) {
            System.out.println("You need at least 2 accounts to
transfer between your own accounts.");
            return;
        }

        System.out.println("Select destination account:");
        for (int i = 0; i < customerAccounts.size(); i++) {
            BankAccount account = customerAccounts.get(i);
            if
(!account.getAccountNumber().equals(currentAccount.getAccountNumber()
)) {
                System.out.println((i + 1) + ". " +
account.getAccountNumber() + " - " +
                        account.getAccountType() + " - Balance: $" +
                        String.format("%.2f", account.getBalance()));
            }
        }

        System.out.print("Select account: ");
        int destChoice = getIntInput();
        System.out.print("Enter transfer amount: $");
        double amount = getDoubleInput();

        if (destChoice >= 1 && destChoice <= customerAccounts.size())
{
            BankAccount destAccount = customerAccounts.get(destChoice
- 1);
            if
(!destAccount.getAccountNumber().equals(currentAccount.getAccountNumb
er())) {
                if
(bank.transferFunds(currentAccount.getAccountNumber(),
destAccount.getAccountNumber(), amount)) {
                    printReceipt("TRANSFER", amount,
currentAccount.getBalance());
                }
            } else {
                System.out.println("Cannot transfer to the same
account.");
            }
        } else {
            System.out.println("Invalid selection.");
        }
    }

    private void transferToOtherCustomer() {
        System.out.print("Enter destination account number: ");
```

```java
        String destAccountNumber = scanner.nextLine();
        System.out.print("Enter transfer amount: $");
        double amount = getDoubleInput();

        if (bank.transferFunds(currentAccount.getAccountNumber(),
destAccountNumber, amount)) {
            printReceipt("TRANSFER", amount,
currentAccount.getBalance());
        }
    }

    private void printReceipt(String transactionType, double amount,
double balance) {
        System.out.println("\n=== TRANSACTION RECEIPT ===");
        System.out.println("Account: " +
currentAccount.getAccountNumber());
        System.out.println("Type: " + transactionType);
        if (amount > 0) {
            System.out.println("Amount: $" + String.format("%.2f",
amount));
        }
        System.out.println("Balance: $" + String.format("%.2f",
balance));
        System.out.println("Thank you for banking with us!");
        System.out.println("============================\n");
    }

    private String generateCustomerId() {
        int nextId = bank.getAllCustomers().size() + 1;
        return "C" + String.format("%03d", nextId);
    }

    private String generateAccountNumber(String accountType) {
        int nextAccount = bank.getAllAccounts().size() + 1;
        if (accountType.equals("SAVINGS")) {
            return "SA" + String.format("%03d", nextAccount);
        } else {
            return "CA" + String.format("%03d", nextAccount);
        }
    }

    private int getIntInput() {
        while (true) {
            try {
                return Integer.parseInt(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.print("Please enter a valid number: ");
            }
        }
    }

    private double getDoubleInput() {
        while (true) {
            try {
                return Double.parseDouble(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.print("Please enter a valid amount: ");
            }
        }
    }
}
```

Bank Class:

```java
import java.util.HashMap;
import java.util.ArrayList;

public class Bank {
    private HashMap<String, Customer> customers;
    private HashMap<String, BankAccount> accounts;
    private String bankName;

    public Bank(String bankName) {
        this.bankName = bankName;
        this.customers = new HashMap<>();
        this.accounts = new HashMap<>();
        initializeSampleData();
    }

    private void initializeSampleData() {
        // Sample data remains the same
        Customer customer1 = new Customer("C001", "Alice Johnson",
"1234");
        Customer customer2 = new Customer("C002", "Bob Smith",
"5678");
        Customer customer3 = new Customer("C003", "Carol Davis",
"9012");

        SavingsAccount savings1 = new SavingsAccount("SA001", "C001",
1000.0);
        CheckingAccount checking1 = new CheckingAccount("CA001",
"C001", 500.0);

        SavingsAccount savings2 = new SavingsAccount("SA002", "C002",
1500.0);
        CheckingAccount checking2 = new CheckingAccount("CA002",
"C002", 800.0);

        SavingsAccount savings3 = new SavingsAccount("SA003", "C003",
2000.0);

        customer1.addAccount(savings1);
        customer1.addAccount(checking1);
        customer2.addAccount(savings2);
        customer2.addAccount(checking2);
        customer3.addAccount(savings3);

        addCustomer(customer1);
        addCustomer(customer2);
        addCustomer(customer3);

        addAccount(savings1);
        addAccount(checking1);
        addAccount(savings2);
        addAccount(checking2);
        addAccount(savings3);

        System.out.println("Sample data initialized with 3 customers
and 5 accounts.");
    }

    // ADD VALIDATION TO PREVENT DUPLICATE CUSTOMER IDs
    public boolean addCustomer(Customer customer) {
        if (customers.containsKey(customer.getCustomerId())) {
            System.out.println("Customer ID already exists: " +
customer.getCustomerId());
            return false;
        }
        customers.put(customer.getCustomerId(), customer);
        return true;
    }

    // ADD VALIDATION TO PREVENT DUPLICATE ACCOUNT NUMBERS
    public void addAccount(BankAccount account) {
        if (!accounts.containsKey(account.getAccountNumber())) {
            accounts.put(account.getAccountNumber(), account);
        } else {
            System.out.println("Account number already exists: " +
account.getAccountNumber());
        }
    }

    public Customer getCustomer(String customerId) {
        return customers.get(customerId);
    }

    public BankAccount getAccount(String accountNumber) {
        return accounts.get(accountNumber);
    }

    public ArrayList<Customer> getAllCustomers() {
        return new ArrayList<>(customers.values());
    }

    public ArrayList<BankAccount> getAllAccounts() {
        return new ArrayList<>(accounts.values());
    }

    public boolean transferFunds(String fromAccountNumber, String
toAccountNumber, double amount) {
        BankAccount fromAccount = getAccount(fromAccountNumber);
        BankAccount toAccount = getAccount(toAccountNumber);

        if (fromAccount == null) {
            System.out.println("Source account not found.");
            return false;
        }

        if (toAccount == null) {
            System.out.println("Destination account not found.");
            return false;
        }

        if (fromAccount.getStatus() != AccountStatus.ACTIVE) {
            System.out.println("Source account is not active.");
            return false;
        }

        if (toAccount.getStatus() != AccountStatus.ACTIVE) {
            System.out.println("Destination account is not active.");
            return false;
        }

        if (fromAccount.withdraw(amount)) {
            if (toAccount.deposit(amount)) {
                fromAccount.addTransaction(new
Transaction(TransactionType.TRANSFER, amount, "SUCCESS",
                        "Transfer to " + toAccountNumber));
                toAccount.addTransaction(new
Transaction(TransactionType.TRANSFER, amount, "SUCCESS",
                        "Transfer from " + fromAccountNumber));
                System.out.println("Transfer successful!");
                return true;
            } else {
                fromAccount.deposit(amount);
                System.out.println("Transfer failed: Could not
deposit to destination account.");
                return false;
            }
        } else {
            System.out.println("Transfer failed: Insufficient funds
or withdrawal not allowed.");
            return false;
        }
    }

    public BankAccount createAccount(String accountType, String
accountNumber, String customerId, double initialBalance) {
        Customer customer = getCustomer(customerId);
        if (customer == null) {
            System.out.println("Customer not found.");
            return null;
        }

        // Check if account number already exists
        if (accounts.containsKey(accountNumber)) {
            System.out.println("Account number already exists: " +
accountNumber);
            return null;
        }

        BankAccount newAccount;
        try {
            switch (accountType.toUpperCase()) {
                case "SAVINGS":
                    newAccount = new SavingsAccount(accountNumber,
customerId, initialBalance);
                    break;
                case "CHECKING":
                    newAccount = new CheckingAccount(accountNumber,
customerId, initialBalance);
                    break;
                default:
                    System.out.println("Invalid account type.");
                    return null;
            }

            addAccount(newAccount);
            customer.addAccount(newAccount);
            System.out.println("New " + accountType + " account
created: " + accountNumber);
            return newAccount;
        } catch (IllegalArgumentException e) {
            System.out.println("Error creating account: " +
e.getMessage());
```

```
            return null;
        }
    }

    // NEW METHOD: Check if customer ID is available
    public boolean isCustomerIdAvailable(String customerId) {
        return !customers.containsKey(customerId);
    }

    // NEW METHOD: Check if account number is available
    public boolean isAccountNumberAvailable(String accountNumber) {
        return !accounts.containsKey(accountNumber);
    }
}
```

Bank Account Class:

```
import java.util.ArrayList;

public abstract class BankAccount {
    protected String accountNumber;
    protected double balance;
    protected String customerId;
    protected AccountStatus status;
    protected ArrayList<Transaction> transactionHistory;

    public BankAccount(String accountNumber, String customerId,
double initialBalance) {
        this.accountNumber = accountNumber;
        this.customerId = customerId;
        this.balance = initialBalance;
        this.status = AccountStatus.ACTIVE;
        this.transactionHistory = new ArrayList<>();
    }

    // Abstract methods to be implemented by subclasses
    public abstract boolean withdraw(double amount);
    public abstract boolean deposit(double amount);
    public abstract String getAccountType();

    // Common methods
    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getCustomerId() {
        return customerId;
    }

    public AccountStatus getStatus() {
        return status;
    }

    public void setStatus(AccountStatus status) {
        this.status = status;
    }

    public ArrayList<Transaction> getTransactionHistory() {
        return new ArrayList<>(transactionHistory);
    }

    public void addTransaction(Transaction transaction) {
        transactionHistory.add(transaction);
    }

    public void displayTransactionHistory() {
        System.out.println("\n--- Transaction History for Account: "
+ accountNumber + " ---");
        if (transactionHistory.isEmpty()) {
            System.out.println("No transactions found.");
        } else {
            for (Transaction transaction : transactionHistory) {
                System.out.println(transaction);
            }
        }
    }
}
```

Checking Account:

```
public class CheckingAccount extends BankAccount {
    private static final double OVERDRAFT_LIMIT = 500.0;
    private static final double TRANSACTION_FEE = 1.50;

    public CheckingAccount(String accountNumber, String customerId,
double initialBalance) {
        super(accountNumber, customerId, initialBalance);
    }

    @Override
    public boolean withdraw(double amount) {
        if (status != AccountStatus.ACTIVE) {
            System.out.println("Cannot withdraw from inactive
account.");
            return false;
        }

        if (amount <= 0) {
            System.out.println("Withdrawal amount must be
positive.");
            return false;
        }

        double totalAmount = amount;
        if (balance < 1000) {
            totalAmount += TRANSACTION_FEE;
        }

        if (balance - totalAmount < -OVERDRAFT_LIMIT) {
            String message = "Withdrawal failed: Overdraft limit of
$" + OVERDRAFT_LIMIT + " would be exceeded.";
            addTransaction(new
Transaction(TransactionType.WITHDRAWAL, amount, "FAILED_OVERDRAFT",
message));
            System.out.println(message);
            return false;
        }

        balance -= totalAmount;
        String message = "Withdrawal successful.";
        if (totalAmount > amount) {
            message += " Transaction fee: $" + TRANSACTION_FEE + "
applied.";
        }
        message += " New balance: $" + String.format("%.2f",
balance);

        addTransaction(new Transaction(TransactionType.WITHDRAWAL,
amount, "SUCCESS", message));
        System.out.println(message);
        return true;
    }

    @Override
    public boolean deposit(double amount) {
        if (status != AccountStatus.ACTIVE) {
            System.out.println("Cannot deposit to inactive
account.");
            return false;
        }

        if (amount <= 0) {
            System.out.println("Deposit amount must be positive.");
            return false;
        }

        balance += amount;
        String message = "Deposit successful. New balance: $" +
String.format("%.2f", balance);
        addTransaction(new Transaction(TransactionType.DEPOSIT,
amount, "SUCCESS", message));
        System.out.println(message);
        return true;
    }

    @Override
    public String getAccountType() {
        return "Checking Account";
    }

    public double getOverdraftLimit() {
        return OVERDRAFT_LIMIT;
    }
}
```

Customer class:

```
import java.util.ArrayList;
import java.util.HashMap;

public class Customer {
    private String customerId;
    private String name;
    private String pin;
    private HashMap<String, BankAccount> accounts;
```

```java
    private int failedLoginAttempts;
    private boolean atmAccessBlocked;

    public Customer(String customerId, String name, String pin) {
        this.customerId = customerId;
        this.name = name;
        this.pin = pin;
        this.accounts = new HashMap<>();
        this.failedLoginAttempts = 0;
        this.atmAccessBlocked = false;
    }

    public boolean verifyPin(String inputPin) {
        if (atmAccessBlocked) {
            System.out.println("ATM access is blocked. Please contact
administrator.");
            return false;
        }

        if (this.pin.equals(inputPin)) {
            failedLoginAttempts = 0;
            return true;
        } else {
            failedLoginAttempts++;
            if (failedLoginAttempts >= 3) {
                atmAccessBlocked = true;
                System.out.println("Too many failed attempts. ATM
access blocked.");
            } else {
                System.out.println("Invalid PIN. " + (3 -
failedLoginAttempts) + " attempts remaining.");
            }
            return false;
        }
    }

    public void unblockATMAccess() {
        this.atmAccessBlocked = false;
        this.failedLoginAttempts = 0;
        System.out.println("ATM access unblocked for customer: " +
name);
    }

    public void addAccount(BankAccount account) {
        accounts.put(account.getAccountNumber(), account);
    }

    public BankAccount getAccount(String accountNumber) {
        return accounts.get(accountNumber);
    }

    public ArrayList<BankAccount> getAllAccounts() {
        return new ArrayList<>(accounts.values());
    }

    public String getCustomerId() { return customerId; }
    public String getName() { return name; }
    public boolean isAtmAccessBlocked() { return atmAccessBlocked; }
    public int getFailedLoginAttempts() { return failedLoginAttempts;
}
```

```
    public void displayAccounts() {
        System.out.println("\n--- Accounts for " + name + " ---");
        if (accounts.isEmpty()) {
            System.out.println("No accounts found.");
        } else {
            for (BankAccount account : accounts.values()) {
                System.out.println(account.getAccountNumber() + " - "
+
                        account.getAccountType() + " - Balance: $" +
                        String.format("%.2f", account.getBalance()) +
                        " - Status: " + account.getStatus());
            }
        }
    }
}
```

Saving Account:

```
public class SavingsAccount extends BankAccount {
    private static final double MINIMUM_BALANCE = 100.0;
    private static final double INTEREST_RATE = 0.02;

    public SavingsAccount(String accountNumber, String customerId,
double initialBalance) {
        super(accountNumber, customerId, initialBalance);
        if (initialBalance < MINIMUM_BALANCE) {
            throw new IllegalArgumentException("Initial balance must
be at least $" + MINIMUM_BALANCE);
        }
    }

    @Override
    public boolean withdraw(double amount) {
        if (status != AccountStatus.ACTIVE) {
            System.out.println("Cannot withdraw from inactive
account.");
            return false;
        }

        if (amount <= 0) {
            System.out.println("Withdrawal amount must be
positive.");
            return false;
        }

        if (balance - amount < MINIMUM_BALANCE) {
            String message = "Withdrawal failed: Minimum balance
requirement of $" + MINIMUM_BALANCE + " would be violated.";
            addTransaction(new
Transaction(TransactionType.WITHDRAWAL, amount, "FAILED_MIN_BALANCE",
message));
            System.out.println(message);
            return false;
        }

        balance -= amount;
        String message = "Withdrawal successful. New balance: $" +
String.format("%.2f", balance);
        addTransaction(new Transaction(TransactionType.WITHDRAWAL,
amount, "SUCCESS", message));
        System.out.println(message);
        return true;
    }

    @Override
    public boolean deposit(double amount) {
        if (status != AccountStatus.ACTIVE) {
            System.out.println("Cannot deposit to inactive
account.");
            return false;
        }

        if (amount <= 0) {
            System.out.println("Deposit amount must be positive.");
            return false;
        }

        balance += amount;
        String message = "Deposit successful. New balance: $" +
String.format("%.2f", balance);
        addTransaction(new Transaction(TransactionType.DEPOSIT,
amount, "SUCCESS", message));
        System.out.println(message);
        return true;
    }

    @Override
    public String getAccountType() {
        return "Savings Account";
    }

    public void applyInterest() {
        double interest = balance * INTEREST_RATE;
        balance += interest;
        addTransaction(new Transaction(TransactionType.INTEREST,
interest, "SUCCESS",
                "Interest applied: $" + String.format("%.2f",
interest)));
    }

    public double getMinimumBalance() {
        return MINIMUM_BALANCE;
    }
}
```

Transaction Class:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Transaction {
    private static int nextTransactionId = 1;

    private String transactionId;
    private TransactionType type;
    private double amount;
    private LocalDateTime timestamp;
    private String status;
    private String description;

    public Transaction(TransactionType type, double amount, String
```

```java
status, String description) {
        this.transactionId = "TXN" + String.format("%06d",
nextTransactionId++);
        this.type = type;
        this.amount = amount;
        this.timestamp = LocalDateTime.now();
        this.status = status;
        this.description = description;
    }

    @Override
    public String toString() {
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        return String.format("%s | %s | %s | $%.2f | %s | %s",
                transactionId, timestamp.format(formatter), type,
amount, status, description);
    }

    public String getTransactionId() { return transactionId; }
    public TransactionType getType() { return type; }
    public double getAmount() { return amount; }
    public LocalDateTime getTimestamp() { return timestamp; }
    public String getStatus() { return status; }
    public String getDescription() { return description; }
}
```

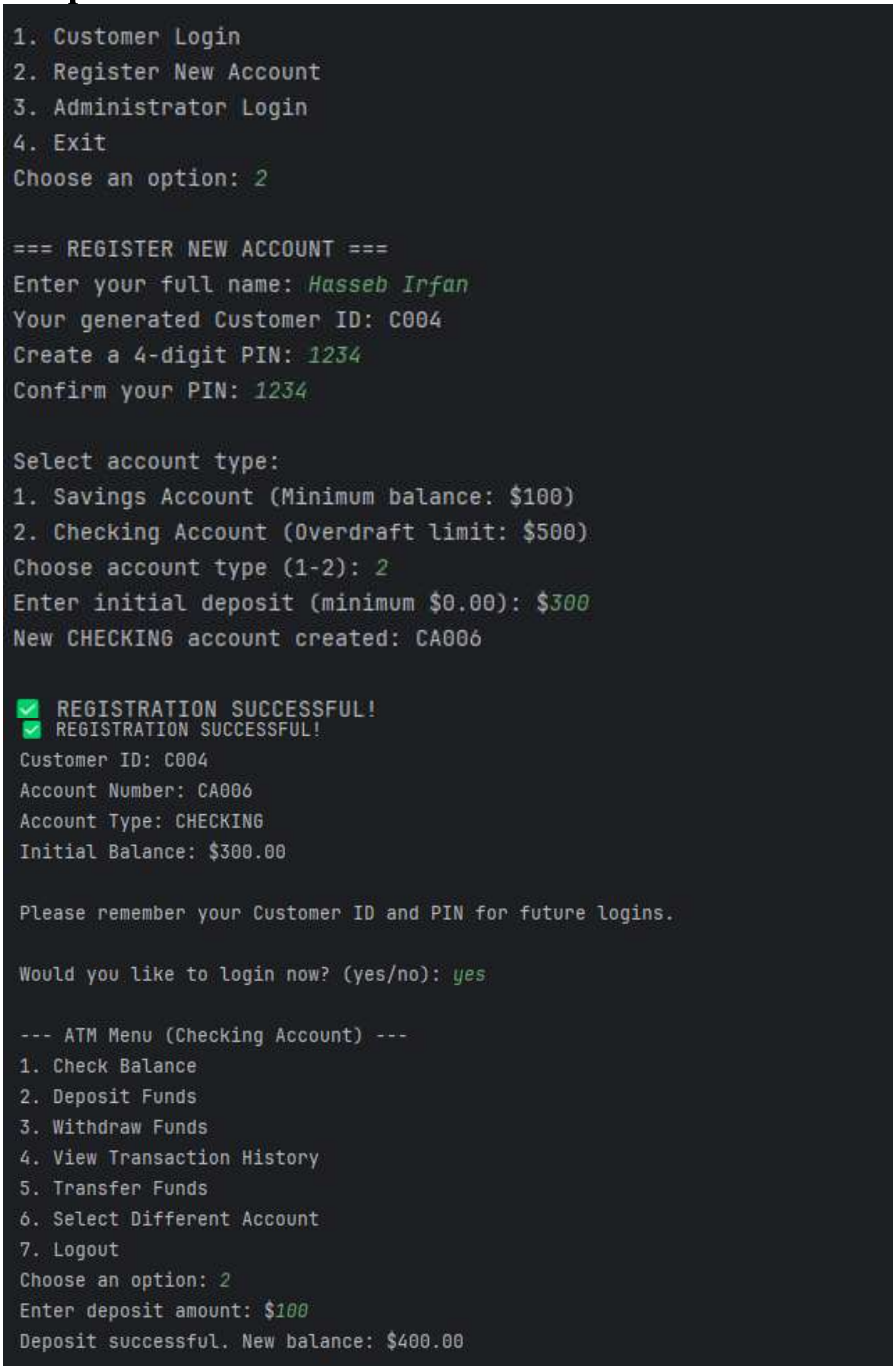
TransactionType Class:

```java
public enum TransactionType {
    DEPOSIT,
    WITHDRAWAL,
    TRANSFER,
    INTEREST,
    FEE
}
```

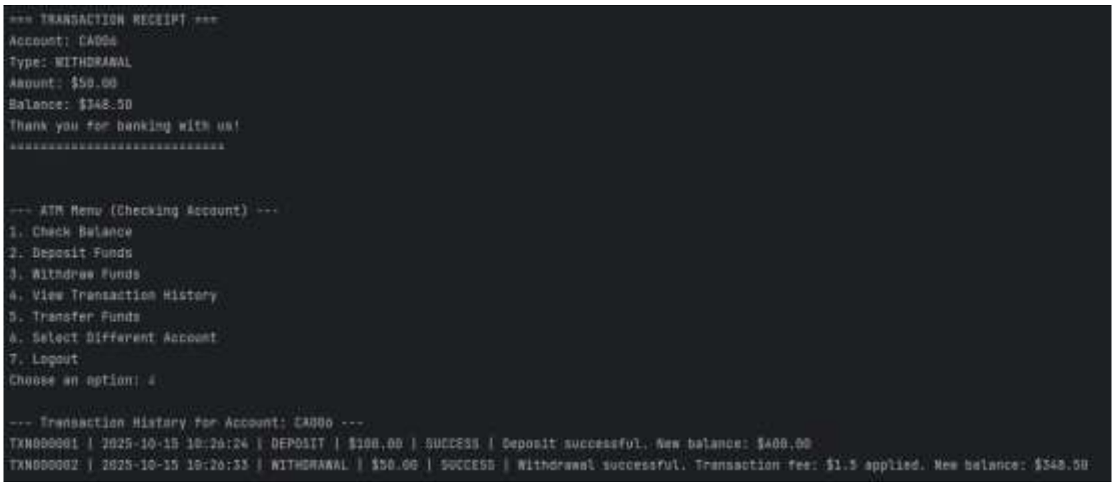
Main Class:

```java
public class Main {
    public static void main(String[] args) {
        Bank bank = new Bank("Java Banking System");
        ATM atm = new ATM(bank);
        atm.start();
    }
}
```

Output:



```
1. Customer Login
2. Register New Account
3. Administrator Login
4. Exit
Choose an option: 2

=== REGISTER NEW ACCOUNT ===
Enter your full name: Hasseb Irfan
Your generated Customer ID: C004
Create a 4-digit PIN: 1234
Confirm your PIN: 1234

Select account type:
1. Savings Account (Minimum balance: $100)
2. Checking Account (Overdraft limit: $500)
Choose account type (1-2): 2
Enter initial deposit (minimum $0.00): $300
New CHECKING account created: CA006

✅ REGISTRATION SUCCESSFUL!
✅ REGISTRATION SUCCESSFUL!
Customer ID: C004
Account Number: CA006
Account Type: CHECKING
Initial Balance: $300.00

Please remember your Customer ID and PIN for future logins.

Would you like to login now? (yes/no): yes

--- ATM Menu (Checking Account) ---
1. Check Balance
2. Deposit Funds
3. Withdraw Funds
4. View Transaction History
5. Transfer Funds
6. Select Different Account
7. Logout
Choose an option: 2
Enter deposit amount: $100
Deposit successful. New balance: $400.00
```



```
=== TRANSACTION RECEIPT ===
Account: CA006
Type: WITHDRAWAL
Amount: $50.00
Balance: $348.50
Thank you for banking with us!
============================


--- ATM Menu (Checking Account) ---
1. Check Balance
2. Deposit Funds
3. Withdraw Funds
4. View Transaction History
5. Transfer Funds
6. Select Different Account
7. Logout
Choose an option: 4

--- Transaction History for Account: CA006 ---
TXN000001 | 2025-10-15 10:26:24 | DEPOSIT | $100.00 | SUCCESS | Deposit successful. New balance: $400.00
TXN000002 | 2025-10-15 10:26:33 | WITHDRAWAL | $50.00 | SUCCESS | Withdrawal successful. Transaction fee: $1.5 applied. New balance: $348.50
```