OOP with JAVA

Bank Management System

Prepared by

Krunal Ambaliya

226200316005

Introduction to project

The bank management system project is designed to facilitate efficient management of accounts, transactions, and overall banking operations. It incorporates various Object-Oriented Programming (OOP) concepts and Java programming principles to ensure a robust and scalable solution.

Key Components:

- **AccountManagement.java:** This file is responsible for managing accounts within the system. It may include functionalities for creating, updating, and deleting accounts, showcasing the principles of encapsulation and modularity in Java programming.
- **ManageAccount.java:** This file may handle the management of account-related operations, such as deposits, withdrawals, and transfers. It showcases the implementation of business logic and the use of Java collections for efficient data handling.
- **TransferFunds.java:** This file is likely dedicated to handling fund transfer operations between accounts. It demonstrates the implementation of transactional functionalities and error handling using Java's exception handling mechanisms.
- **Account.java:** This file likely represents the core structure of bank accounts, handling account details, balances, and transactions. It demonstrates the use of classes and objects to model individual accounts within the system.
- **BankManagementSystem.java**: This file serves as the main driver of the bank management system. It orchestrates the interactions between different components, showcasing the use of inheritance, polymorphism, and abstraction to create a cohesive banking system.
- **Main.java:** This file likely contains the entry point of the application, initializing the system and starting its execution. It demonstrates the basic structure of a Java program and how different classes are orchestrated to create a functional system.

Object-Oriented Programming Concepts

1. **Inheritance:**
    - **Description:** Inheritance allows a class to inherit properties and behavior from another class.
    - **Implementation:** In the program, classes like Account, AccountManagement, and BankManagementSystem demonstrate inheritance by extending base classes.
2. **Encapsulation:**
    - **Description:** Encapsulation involves bundling data and methods that operate on the data into a single unit.
    - **Implementation:** The classes encapsulate data like account details and methods to interact with this data securely.
3. **Polymorphism:**
    - **Description:** Polymorphism allows objects to be treated as instances of their parent class.
    - **Implementation:** Achieved through method overriding and method overloading in the program, enabling flexibility in handling different types of accounts.

Java Programming Concepts

1. **Exception Handling:**
    - **Description:** Exception handling is a mechanism to handle runtime errors and maintain the normal flow of the application.
    - **Implementation:** The deposit() and withdraw() methods in the BankAccount class use try-catch blocks to handle InsufficientFundsException and InvalidAmountException

2. **Collections Framework:**
    - **Description:** Java Collections provide classes and interfaces to store and manipulate groups of objects.
    - **Implementation:** Utilized in the program for managing lists of accounts, transactions, or other data structures efficiently.
3. **File Handling:**
    - **Description:** File handling involves reading from and writing to files.
    - **Implementation:** Demonstrated in the program for tasks like storing account information persistently or logging transactions

Programming Concepts

1. **Modularity:**
    - **Description:** Modularity breaks down a program into smaller, manageable parts.
    - **Implementation:** The project is modular, with distinct classes for accounts, management, and the main system, promoting code reusability and maintainability.
2. **Abstraction:**
    - **Description:** Abstraction focuses on hiding complex implementation details and showing only essential features.
    - **Implementation:** Classes and methods in the program abstract away low-level details, providing a clear interface for interacting with the bank system.
3. **Concurrency:**
    - **Description:** Concurrency deals with executing multiple tasks simultaneously.
    - **Implementation:** Utilized in the program for handling multiple transactions or user interactions concurrently, ensuring efficient system performance.

AccountManagement.java

```
1. package bankmanagementsystem;
 2.
 3. import java.io.*;
 4. import java.util.List;
 5. import java.util.ArrayList;
 6. import java.util.InputMismatchException;
 7. import java.util.Scanner;
 8.
 9. public class AccountManagement {
10.     public static List<Account> accounts;
11.     private Scanner scanner;
12.     private File file;
13.
14.     public AccountManagement() {
15.         accounts = new ArrayList<>();
16.         scanner = new Scanner(System.in);
17.         file = new File("accounts.txt"); // File to store account details
18.         readFromFile();
19.         if (accounts == null) {
20.             accounts = new ArrayList<>(); // If no accounts found, initialize a new list
21.         }
22.     }
23.
24.     public void run() {
25.         boolean exit = false;
26.
27.         while (!exit) {
28.
System.out.println("\n=========================================================
===");
29.             System.out.println("\t\t\t\tAccount Management");
30.
System.out.println("=========================================================
==");
31.             System.out.println("1. Open New Account");
32.             System.out.println("2. Delete Account");
33.             System.out.println("3. Update Account Information");
34.             System.out.println("4. View Account Details");
35.             System.out.println("9. Back to Main Menu");
36.             System.out.print("Enter your choice: ");
37.
38.             int choice = getValidIntegerInput();
39.
40.             switch (choice) {
41.                 case 1:
42.                     openNewAccount();
43.                     break;
44.                 case 2:
45.                     deleteAccount();
46.                     break;
47.                 case 3:
48.                     updateAccountInformation();
49.                     break;
50.                 case 4:
51.                     viewAccountDetails();
52.                     break;
```

```
53.             case 9:
54.                 exit = true;
55.                 break;
56.             default:
57.                 System.out.println("Invalid choice");
58.         }
59.     }
60. }
61.
62. private int getValidIntegerInput() {
63.     int choice = -1;
64.     while (true) {
65.         try {
66.             choice = scanner.nextInt();
67.             scanner.nextLine(); // Consume newline left-over
68.             break;
69.         } catch (InputMismatchException e) {
70.             System.out.print("Invalid input. Please enter a valid number: ");
71.             scanner.nextLine(); // Clear the invalid input
72.         }
73.     }
74.     return choice;
75. }
76.
77. public void openNewAccount() {
78.     System.out.println("\n--------------- | Open New Account | ---------------");
79.     System.out.print("\n ➥Enter Account ID: ");
80.     String accountId = scanner.nextLine();
81.     System.out.print(" ➥Enter Customer Name: ");
82.     String customerName = scanner.nextLine();
83.     Account newAccount = new Account(accountId, customerName);
84.     accounts.add(newAccount);
85.     System.out.println("New account opened successfully.");
86.
87.     // Write updated account list to file
88.     writeToFile();
89. }
90.
91. private void deleteAccount() {
92.     System.out.println("\n--------------- | Delete Account | ---------------");
93.     System.out.print("\n➥Enter Account ID to delete: ");
94.     String accountId = scanner.nextLine();
95.     boolean found = false;
96.     for (Account account : accounts) {
97.         if (account.getAccountId().equals(accountId)) {
98.             accounts.remove(account);
99.             System.out.println("Account deleted successfully.");
100.             found = true;
101.             break;
102.         }
103.     }
104.     if (!found) {
105.         System.out.println("Account not found. ⚠\uFE0F");
106.     }
107.
108.     // Write updated account list to file
109.     writeToFile();
110. }
111.
```

```
112.     private void updateAccountInformation() {
113.         System.out.println("\n--------------- | Update Account Information| ---------------");
114.         System.out.print("\n➥ Enter Account ID to update: ");
115.         String accountId = scanner.nextLine();
116.         boolean found = false;
117.         for (Account account : accounts) {
118.             if (account.getAccountId().equals(accountId)) {
119.                 System.out.print("➥ Enter new customer name: ");
120.                 String newCustomerName = scanner.nextLine();
121.                 account.setCustomerName(newCustomerName);
122.                 System.out.println("Account information updated successfully.");
123.                 found = true;
124.                 break;
125.             }
126.         }
127.         if (!found) {
128.             System.out.println("Account not found.");
129.         }
130. 
132.         writeToFile();
133.     }
134. 
135.     private void viewAccountDetails() {
136.         System.out.println("\n--------------- | View Account Details| ---------------");
137.         System.out.print("\n➥ Enter Account ID to view details: ");
138.         String accountId = scanner.nextLine();
139.         boolean found = false;
140.         for (Account account : accounts) {
141.             if (account.getAccountId().equals(accountId)) {
142.                 System.out.println("Account ID: " + account.getAccountId());
143.                 System.out.println("Customer Name: " + account.getCustomerName());
144.                 System.out.println("Balance: " + account.getBalance());
145.                 found = true;
146.                 break;
147.             }
148.         }
149.         if (!found) {
150.             System.out.println("Account not found.");
151.         }
152.     }
153. 
154.     // Method to write account list to file
155.     private void writeToFile() {
156.         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file))) {
157.             oos.writeObject(accounts);
158.         } catch (IOException e) {
159.             e.printStackTrace();
160.         }
161.     }
162. 
163.     // Method to read account list from file
164.     public void readFromFile() {
165.         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {
166.             accounts = (List<Account>) ois.readObject();
167.         } catch (IOException | ClassNotFoundException e) {
168.             e.printStackTrace();
169.         }
170.     }
171. }
```

ManageAccount.java

```
1. package bankmanagementsystem;
2. 
3. import java.util.List;
4. import java.io.*;
5. import java.util.Scanner;
6. class FileHandler {
7.     public static List<Account> readFromFile() {
8.         List<Account> accounts = null;
9.         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("accounts.txt"))) {
10.             accounts = (List<Account>) ois.readObject();
11.         } catch (IOException | ClassNotFoundException e) {
12.             e.printStackTrace();
13.         }
14.         return accounts;
15.     }
16. 
17.     public static void writeToFile(List<Account> accounts) {
18.         try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("accounts.txt"))) {
19.             oos.writeObject(accounts);
20.         } catch (IOException e) {
21.             e.printStackTrace();
22.         }
23.     }
24. }
25. public class ManageAccount {
26.     public static String accountId;
27.     private Scanner scanner;
28. 
29.     public ManageAccount() {
30.         scanner = new Scanner(System.in);
31.     }
32. 
33.     public void run() {
34.         boolean exit = false;
35.         while (!exit) {
36. 
System.out.println("\n=================================================
===");
37.             System.out.println("\t\t\t\tManage Account");
38. 
System.out.println("=================================================
==");
39.             System.out.println("1. Add Balance");
40.             System.out.println("2. Withdraw");
41.             System.out.println("3. Check Balance");
42.             System.out.println("4. View Transaction History");
43.             System.out.println("9. Back to Main Menu");
44.             System.out.print("Enter your choice: ");
45.             int choice = scanner.nextInt();
46.             switch (choice) {
47.                 case 1:
48.                     addBalance();
49.                     break;
50.                 case 2:
51.                     withdraw();
52.                     break;
```

```
53.             case 3:
54.                 checkBalance();
55.                 break;
56.             case 4:
57.                 viewTransactionHistory();
58.                 break;
59.             case 9:
60.                 exit = true;
61.                 break;
62.             default:
63.                 System.out.println("Invalid choice \n");
64.         }
65.     }
66. }
67.
68. public void addBalance() {
69.     System.out.println("\n--------------- | Add Balance | ---------------");
70.     Scanner scanner = new Scanner(System.in);
71.     System.out.print(" \n➥ Enter Account ID: ");
72.     String accountId = scanner.nextLine();
73.     System.out.print(" ➥ Enter amount to add: ");
74.     double amount = scanner.nextDouble();
75.     boolean found = false;
76.     for (Account account : AccountManagement.accounts) {
77.         if (account.getAccountId().equals(accountId)) {
78.             double currentBalance = account.getBalance();
79.             account.setBalance(currentBalance + amount);
80.             System.out.println("  Balance added successfully. ☑ ");
81.
82.             // Call addTransaction() on the current account object
83.             String transactionDetails = "Added " + amount + " to balance";
84.             account.addTransaction(transactionDetails);
85.
86.             found = true;
87.             break;
88.         }
89.     }
90.     if (!found) {
91.         System.out.println(" ⚠\uFE0FAccount not found. \n ");
92.     }
93.
94.     // Write to file after updating account balance
95.     FileHandler.writeToFile(AccountManagement.accounts);
96. }
97.
98. private void withdraw() {
99.     System.out.println("\n--------------- | Withdraw | ---------------");
100.     Scanner scanner = new Scanner(System.in);
101.     System.out.print(" \n➥ Enter Account ID: ");
102.     String accountId = scanner.nextLine();
103.     System.out.print(" ➥ Enter amount to withdraw: ");
104.     double amount = scanner.nextDouble();
105.     boolean found = false;
106.     for (Account account : AccountManagement.accounts) {
107.         if (account.getAccountId().equals(accountId)) {
108.             double currentBalance = account.getBalance();
109.             if (currentBalance >= amount) {
110.                 account.setBalance(currentBalance - amount);
111.                 System.out.println("Withdrawal successful. ☑ ");
```

```
114.                 String transactionDetails = "Withdrawn " + amount + " from balance";
115.                 account.addTransaction(transactionDetails);
116.
117.             } else {
118.                 System.out.println("Insufficient balance. ☑ ");
119.             }
120.             found = true;
121.             break;
122.         }
123.     }
124.     if (!found) {
125.         System.out.println("Account not found. ⚠\uFE0F \n");
126.     }
127.
128.     // Write to file after updating account balance and transaction history
129.     FileHandler.writeToFile(AccountManagement.accounts);
130. }
131.
132.
133.
134. private void checkBalance() {
135.     System.out.println("\n--------------- | Check Balance | ---------------");
136.     Scanner scanner = new Scanner(System.in);
137.     System.out.print("\n➥ Enter Account ID: ");
138.     String accountId = scanner.nextLine();
139.     boolean found = false;
140.     List<Account> accounts = FileHandler.readFromFile();
141.     for (Account account : accounts) {
142.         if (account.getAccountId().equals(accountId)) {
143.             System.out.println("Current Balance: " + account.getBalance());
144.             found = true;
145.             break;
146.         }
147.     }
148.     if (!found) {
149.         System.out.println("Account not found. ⚠\uFE0F \n");
150.     }
151. }
152.
153. public void viewTransactionHistory() {
154.     System.out.println("\n--------------- | View Transaction History | ---------------");
155.     Scanner scanner = new Scanner(System.in);
156.     System.out.print(" \n➥ Enter Account ID to view transaction history: ");
157.     String accountId = scanner.nextLine();
158.     boolean found = false;
159.     List<Account> accounts = FileHandler.readFromFile();
160.     for (Account account : accounts) {
161.         if (account.getAccountId().equals(accountId)) {
162.             account.printTransactionHistory(); // Call printTransactionHistory() on the account
instance
163.             found = true;
164.             break;
165.         }
166.     }
167.     if (!found) {
168.         System.out.println("Account not found. ⚠\uFE0F \n");
169.     }
170. }
171.
172. }
```

TransferFunds.java

```java
package bankmanagementsystem;

import java.io.*;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

public class TransferFunds extends ManageAccount {
    private Scanner scanner;

    public TransferFunds() {
        scanner = new Scanner(System.in);
    }

    @Override
    public void run() {
        boolean exit = false;
        while (!exit) {

System.out.println("\n=========================================================");
            System.out.println("\t\t\t\tTransfer Funds ");
System.out.println("=========================================================");
            System.out.println("1. View Transaction History");
            System.out.println("2. To Another Account (External Transfer)");
            System.out.println("9. Back to Main Menu");

            int choice = getValidIntegerInput("Enter your choice: ");
            switch (choice) {
                case 1:
                    printTransactionHistory();
                    break;
                case 2:
                    externalTransfer();
                    break;
                case 9:
                    exit = true;
                    break;
                default:
                    System.out.println("Invalid choice");
            }
        }
    }

    private int getValidIntegerInput(String prompt) {
        int choice = -1;
        while (true) {
            try {
                System.out.print(prompt);
                choice = scanner.nextInt();
                scanner.nextLine(); // Consume newline left-over
                break;
```

```
52.             } catch (InputMismatchException e) {
53.                 System.out.println("Invalid input. Please enter a valid number.");
54.                 scanner.nextLine(); // Clear the invalid input
55.             }
56.         }
57.         return choice;
58.     }


59.
60.     private double getValidDoubleInput(String prompt) {
61.         double value = -1;
62.         while (true) {
63.             try {
64.                 System.out.print(prompt);
65.                 value = scanner.nextDouble();
66.                 scanner.nextLine(); // Consume newline left-over
67.                 break;
68.             } catch (InputMismatchException e) {
69.                 System.out.println("Invalid input. Please enter a valid number.");
70.                 scanner.nextLine(); // Clear the invalid input
71.             }
72.         }
73.         return value;
74.     }
75.
76.     public void printTransactionHistory() {
77.         System.out.println("\n--------------- | Transaction History | ---------------");
78.         System.out.print("\n➥ Enter Account ID to view transaction history: ");
79.         String accountId = scanner.next();
80.         boolean found = false;
81.         List<Account> accounts = FileHandler.readFromFile();
82.         for (Account account : accounts) {
83.             if (account.getAccountId().equals(accountId)) {
84.                 System.out.println("Transaction History for Account ID: " + accountId);
85.                 for (String transaction : account.getTransactionHistory()) {
86.                     if (!transaction.startsWith("Added")) { // Skip balance addition transactions
87.                         // Split the transaction string to get the details
88.                         String[] parts = transaction.split(",");
89.                         if (parts.length >= 3) { // Ensure there are at least 3 parts
90.                             String type = parts[0];
91.                             double amount = Double.parseDouble(parts[1]);
92.                             String fromTo = parts[2];
93.                             System.out.println("Type: " + type + ", Amount: " + amount + ", From/To: " +
fromTo);
94.                         } else {
95.                             System.out.println(" " + transaction);
96.                         }
97.                     }
98.                 }
99.                 found = true;
100.                 break;
101.             }
102.         }
103.         if (!found) {
104.             System.out.println("Account not found.");
105.         }
106.     }
107.
```

```
108.    private void externalTransfer() {
109.        System.out.println("\n--------------- | Transfer funds (external) | ---------------");
110.         // Consume newline left-over
111.        System.out.print("\n➥ Enter Account ID to transfer from: ");
112.        String fromAccountId = scanner.nextLine();
113.        System.out.print("➥ Enter Account ID to transfer to: ");
114.        String toAccountId = scanner.nextLine();
115.        double amount = getValidDoubleInput("➥ Enter amount to transfer: ");
116.        boolean fromAccountFound = false;
117.        boolean toAccountFound = false;
118.        List<Account> accounts = FileHandler.readFromFile();
119.        for (Account account : accounts) {
120.            if (account.getAccountId().equals(fromAccountId)) {
121.                fromAccountFound = true;
122.                double fromBalance = account.getBalance();
123.                if (fromBalance >= amount) {
124.                    for (Account targetAccount : accounts) {
125.                        if (targetAccount.getAccountId().equals(toAccountId)) {
126.                            toAccountFound = true;
127.                            double toBalance = targetAccount.getBalance();
128.                            account.setBalance(fromBalance - amount);
129.                            targetAccount.setBalance(toBalance + amount);
130.                            System.out.println("Transfer successful.");
131.
132.                            // Update transaction history for source account
133.                            String fromTransaction = "Transfer to " + "ID " + toAccountId + " " + "amount: " +
amount;
134.                            account.addTransaction(fromTransaction);
135.
136.                            // Update transaction history for target account
137.                            String toTransaction = "Received from " + "ID " + fromAccountId + " " + "amount: " +
amount;
138.                            targetAccount.addTransaction(toTransaction);
139.
140.                            // Write accounts to file after each transfer
141.                            FileHandler.writeToFile(accounts);
142.
143.                            break;
144.                        }
145.                    }
146.                } else {
147.                    System.out.println("Insufficient balance in the source account.");
148.                }
149.                break;
150.            }
151.        }
152.        if (!fromAccountFound) {
153.            System.out.println("Source account not found.");
154.        }
155.        if (!toAccountFound) {
156.            System.out.println("Target account not found.");
157.        }
158.    }
159. }
160.
```

Account.java

```java
package bankmanagementsystem;

import java.util.ArrayList;
import java.util.List;
import java.io.Serializable;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Account implements Serializable {
    private static final long serialVersionUID = 1L;

    private String accountId;
    private String customerName;
    private double balance;
    private List<String> transactionHistory;

    public Account(String accountId, String customerName) {
        this.accountId = accountId;
        this.customerName = customerName;
        this.balance = 0.0;
        this.transactionHistory = new ArrayList<>();
    }

    // Getters and setters
    public String getAccountId() {
        return accountId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

```

```
47.     // Method to add transaction
48.     public void addTransaction(String transaction) {
49.         transactionHistory.add(transaction);
50.         saveAccountDetails();
51.     }
52.
53.     public List<String> getTransactionHistory() {
54.         return transactionHistory;
55.     }
56.
57.     // Method to print transaction history
58.     public void printTransactionHistory() {
59.         System.out.println("Transaction History for Account ID: " + accountId);
60.         for (String transaction : this.transactionHistory) {
61.             System.out.println(transaction);
62.         }
63.     }
64.
65.     // Save account details to file
66.     private void saveAccountDetails() {
67.         try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("accounts.dat"))) {
68.             oos.writeObject(this);
69.         } catch (Exception e) {
70.             e.printStackTrace();
71.         }
72.     }
73.
74.     // Retrieve account details from file
75.     public static Account retrieveAccountDetails(String accountId) {
76.         try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("accounts.dat"))) {
77.             Object obj;
78.             while ((obj = ois.readObject()) != null) {
79.                 if (obj instanceof Account) {
80.                     Account account = (Account) obj;
81.                     if (account.getAccountId().equals(accountId)) {
82.                         return account;
83.                     }
84.                 }
85.             }
86.         } catch (Exception e) {
87.             e.printStackTrace();
88.         }
89.         return null;
90.     }
91. }
92.
```

BankManagementSystem.java

```java
package bankmanagementsystem;

import java.util.Scanner;
import java.util.InputMismatchException;

public class BankManagementSystem {
    public static Scanner scanner;
    private AccountManagement accountManagement;
    private ManageAccount manageAccount;
    private TransferFunds transferFunds;

    public BankManagementSystem() {
        scanner = new Scanner(System.in);
        accountManagement = new AccountManagement();
        manageAccount = new ManageAccount();
        transferFunds = new TransferFunds();
    }

    public void run() {
        boolean exit = false;
        while (!exit) {
            System.out.println("------------------------------------------------------------");
            System.out.println("\t\t\tBank Management System");
            System.out.println("------------------------------------------------------------");
            System.out.println("1. Account Management");
            System.out.println("2. Manage Account");
            System.out.println("3. Transfer Funds");
            System.out.println("9. Exit");

            int choice = getValidIntegerInput("Enter your choice: ");
            switch (choice) {
                case 1:
                    accountManagementMenu();
                    break;
                case 2:
                    manageAccountMenu();
                    break;
                case 3:
                    transferFundsMenu();
                    break;
                case 9:
                    exit = true;
                    break;
                default:
                    System.out.println("Invalid choice");
            }
        }
        scanner.close();
    }
```

```
50.
51.     private int getValidIntegerInput(String prompt) {
52.         int choice = -1;
53.         while (true) {
54.             try {
55.                 System.out.print(prompt);
56.                 choice = scanner.nextInt();
57.                 scanner.nextLine(); // Consume newline left-over
58.                 break;
59.             } catch (InputMismatchException e) {
60.                 System.out.println("Invalid input. Please enter a valid number.");
61.                 scanner.nextLine(); // Clear the invalid input
62.             }
63.         }
64.         return choice;
65.     }
66.
67.     private void accountManagementMenu() {
68.         accountManagement.run();
69.     }
70.
71.     private void manageAccountMenu() {
72.         manageAccount.run();
73.     }
74.
75.     private void transferFundsMenu() {
76.         transferFunds.run();
77.     }
78.
79.     public static void main(String[] args) {
80.         BankManagementSystem system = new BankManagementSystem();
81.         system.run();
82.     }
83. }
84.
```

Main.java

```java
package bankmanagementsystem;

import java.util.Scanner;

public class Main {
    static Scanner scanner = new Scanner(System.in);
    static Account account; // Declare account variable

    public static void main(String[] args) {
        // Set default values for the account
        String defaultCustomerName = "darshit";
        String defaultAccountId = "1002";

        BankManagementSystem bankSystem = new BankManagementSystem();

        // Open a new account with default values
        openNewAccount(defaultAccountId, defaultCustomerName);
        // Run the bank management system
        bankSystem.run();
        scanner.close();
    }

    // Method to open a new account
    public static void openNewAccount(String accountId, String customerName) {
        Account newAccount = new Account(accountId, customerName);
        AccountManagement.accounts.add(newAccount);
        newAccount.setBalance(10000);

        // Add default transaction history
        String defaultTransaction1 = "Initial deposit of $10000";
        String defaultTransaction2 = "Account created";

        newAccount.addTransaction(defaultTransaction1);
        newAccount.addTransaction(defaultTransaction2);

        System.out.println("New account opened successfully.");
    }
}

```

Output :

```
------------------------------------------------------------
            Bank Management System
------------------------------------------------------------
1. Account Management
2. Manage Account
3. Transfer Funds
9. Exit
Enter your choice: 1
```

```
============================================================
                Account Management
============================================================
1. Open New Account
2. Delete Account
3. Update Account Information
4. View Account Details
9. Back to Main Menu
Enter your choice: 1
```

```
--------------- | Open New Account | ---------------

 ➥Enter Account ID: 002
 ➥Enter Customer Name: jay chavda
New account opened successfully.
```

```
--------------- | Open New Account | ---------------

 ➥Enter Account ID: 001
 ➥Enter Customer Name: het dodiya
New account opened successfully.
```

```
=========================================================
                Account Management
=========================================================
1. Open New Account
2. Delete Account
3. Update Account Information
4. View Account Details
9. Back to Main Menu
Enter your choice: 2
```

```
--------------- | Delete Account | ---------------

➥Enter Account ID to delete: 001
Account deleted successfully.
```

```
--------------- | Update Account Information| ---------------

➥ Enter Account ID to update: 002
➥ Enter new customer name: het dodiya p.
Account information updated successfully.
```

```
--------------- | View Account Details| ---------------

➥ Enter Account ID to view details: 002
Account ID: 002
Customer Name: het dodiya p.
Balance: 0.0
```

```
=========================================================
                    Manage Account
=========================================================
1. Add Balance
2. Withdraw
3. Check Balance
4. View Transaction History
9. Back to Main Menu
Enter your choice: 1
```

```
--------------- | Add Balance | ---------------

➥ Enter Account ID: 002
 ➥ Enter amount to add: 10500
  Balance added successfully. ☑
```

```
--------------- | Withdraw | ---------------

➥ Enter Account ID: 002
 ➥ Enter amount to withdraw: 1500
Withdrawal successful. ☑
```

```
--------------- | Check Balance | ---------------

➥ Enter Account ID: 002
Current Balance: 9000.0
```

```
--------------- | View Transaction History | ---------------

➥ Enter Account ID to view transaction history: 002
Transaction History for Account ID: 002
Added 10500.0 to balance
```

```
==========================================================
                  Transfer Funds
==========================================================
1. View Transaction History
2. To Another Account (External Transfer)
9. Back to Main Menu
Enter your choice: 2
```

```
--------------- | Transfer funds (external) | ---------------

➥ Enter Account ID to transfer from: 002
➥ Enter Account ID to transfer to: 1002
➥ Enter amount to transfer: 300
Transfer successful.
```

```
--------------- | Transaction History | ---------------

➥ Enter Account ID to view transaction history: 002
Transaction History for Account ID: 002
 Withdrawn 1500.0 from balance
 Transfer to ID 1002 amount: 300.0
```

```
==========================================================
                  Transfer Funds
==========================================================
1. View Transaction History
2. To Another Account (External Transfer)
9. Back to Main Menu
Enter your choice: 9
```