

Micro-Project

Subject :

OBJECT ORIENTED PROGRAMMING IN JAVA

Topic of Micro Project:

BANKING MANAGEMENT SYSTEM FOR TRANSACTION SERVICES

Created by:

SUDHIRKUMAR J KUCHARA (216170316009)

HET PATEL (216170316030)

RAHI PATEL (216170316024)

Project Description

Introduction

What is Bank Management:

- **Bank management** is characterized by the specific object of management – financial relations connected with banking activities and other relations, also connected with implementing management functions in banking.

What is Transaction:

- **A transaction** is any kind of action involved in conducting business, or an interaction between people. When you go to the bank, fill out a form, and deposit your paycheck, you make a transaction.

Project Overview:

In this project we created offline **Banking application system** which provides services for user. There we use different packages and methods to make it easy to usable and easier using **Command Line Interface (CLI)**. Where we can create account and perform some Transactional related activities.

Imported packages:

Here we imported 5 packages with different works of them. Let's discuss about each of them,

1. `import java.io.FileOutputStream;`

- This package used for Write Details of Members which includes Account no, Account Holder name, Balance. After selecting create new account choice this package will be use there. After feeding details in Application program then all Details will be written in new data(.dat) file explicitly.

2. `import java.io.IOException`

- This package will use to throw exception if any Input-Output exception found, then it will raise exception.

3. `import java.util.ArrayList`

- ArrayList package is mainly used for create ArrayList Object. Here we used it for store account _no. for some transactional related services.

4. `import java.util.List`

- List in Java provides the facility to maintain the ordered collection. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list. The List interface is found in the java.util package and inherits the Collection interface.

5. `import java.util.Scanner`

- Scanner is most useful package imported in this Application program where we can take any input from user and after that we can do any activities. Without Scanner package make this program would bit complex in input view.

All services provided by this Application program:

There are different total 5 services we tried to perform, which includes:

1. Creating Account

- Create New account for transact services.

2. Deposit Account

- Deposit amount in specific account with considering them account number. Where we add amount in account balance.

3. Withdraw Account

- Withdraw amount from specified account with considering them account number. Where we subtract amount in account balance.

4. Check balance

- Here, we can check balance via account number.

5. Transfer Funds

- This is main feature which is used in real-world example where we can deposit amount from any account and then withdraw or transfer it to another account with account number consideration.

Code in Java for Banking Management Application System

Input:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class BankingApplication {
    private static List<Account> accounts = new ArrayList<>();

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        boolean exit = false;
        while (!exit) {
            System.out.println("=== Banking Application ===");
            System.out.println("1. Create an account");
            System.out.println("2. Deposit money");
            System.out.println("3. Withdraw money");
            System.out.println("4. Check Balance");
            System.out.println("5. Transfer Funds");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Fetching choice

            switch (choice) {
                case 1:
                    createAccount(scanner);
                    break;
                case 2:
                    depositMoney(scanner);
                    break;
                case 3:
                    withdrawMoney(scanner);
                    break;
                case 4:
                    checkBalance(scanner);
                    break;
```

```
        case 5:
            transferFunds(scanner);
            break;
        case 6:
            exit = true;
            break;
        default:
            System.out.println("Invalid choice. Please try again.");
    }
    System.out.println();
}

scanner.close();
}

private static void createAccount(Scanner scanner) {
    System.out.print("Enter account number: ");
    String accountNumber = scanner.nextLine();

    System.out.print("Enter account holder name: ");
    String accountHolderName = scanner.nextLine();

    System.out.print("Enter initial balance: ");
    double balance = scanner.nextDouble();
    scanner.nextLine(); // Consume the newline character

    Account account = new Account(accountNumber, accountHolderName, balance);
    accounts.add(account);

    System.out.println("\nAccount created successfully. Data saved in Database.dat file.");

    try (FileOutputStream writer = new
FileOutputStream("C:\\Users\\Admin\\OneDrive\\Desktop\\Java MP\\Database.dat", true)) {
        String accDetails = "\nAccount no: " + accountNumber + "\n";
        String accDetails2 = "Account Holder: " + accountHolderName + "\n";
        String accDetails3 = "Account Balance: " + balance + "\n";
        byte[] message = accDetails.getBytes();
        byte[] message2 = accDetails2.getBytes();
        byte[] message3 = accDetails3.getBytes();
        writer.write(message);
        writer.write(message2);
        writer.write(message3);
    }
```

```
        writer.write("-----".getBytes());
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void depositMoney(Scanner scanner) {
    System.out.print("Enter account number: ");
    String accountNumber = scanner.nextLine();

    Account account = findAccount(accountNumber);
    if (account != null) {
        System.out.print("Enter amount to deposit: ");
        double amount = scanner.nextDouble();
        scanner.nextLine(); // Consume the newline character

        account.deposit(amount);

        addTransaction(account, "Deposit", amount);
    } else {
        System.out.println("Account not found.");
    }
}

private static void withdrawMoney(Scanner scanner) {
    System.out.print("Enter account number: ");
    String accountNumber = scanner.nextLine();

    Account account = findAccount(accountNumber);
    if (account != null) {
        System.out.print("Enter amount to withdraw: ");
        double amount = scanner.nextDouble();
        scanner.nextLine(); // Consume the newline character

        try {
            account.withdraw(amount);

            addTransaction(account, "Withdrawal", -amount);
        } catch (InsufficientBalanceException e) {
            System.out.println("Withdrawal failed: " + e.getMessage());
        }
    }
}
```

```
    } else {
        System.out.println("Account not found.");
    }
}

private static void checkBalance(Scanner scanner) {
    System.out.print("Enter account number: ");
    String accountNumber = scanner.nextLine();

    Account account = findAccount(accountNumber);
    if (account != null) {
        System.out.println("Account balance: " + account.getBalance());
    } else {
        System.out.println("Account not found.");
    }
}

private static void transferFunds(Scanner scanner) {
    System.out.print("Enter sender's account number: ");
    String senderAccountNumber = scanner.nextLine();

    System.out.print("Enter receiver's account number: ");
    String receiverAccountNumber = scanner.nextLine();

    Account senderAccount = findAccount(senderAccountNumber);
    Account receiverAccount = findAccount(receiverAccountNumber);

    if (senderAccount != null && receiverAccount != null) {
        System.out.print("Enter amount to transfer: ");
        double amount = scanner.nextDouble();
        scanner.nextLine(); // Consume the newline character

        try {
            senderAccount.withdraw(amount);
            receiverAccount.deposit(amount);

            addTransaction(senderAccount, "Transfer to " + receiverAccountNumber, -amount);
            addTransaction(receiverAccount, "Transfer from " + senderAccountNumber, amount);

            System.out.println("Transfer successful.");
        } catch (InsufficientBalanceException e) {
            System.out.println("Transfer failed: " + e.getMessage());
        }
    }
}
```



```
        }
    } else {
        System.out.println("One or both accounts not found.");
    }
}

private static void addTransaction(Account account, String type, double amount) {
    Transaction transaction = new Transaction(type, amount);
    account.addTransaction(transaction);
}

private static Account findAccount(String accountNumber) {
    for (Account account : accounts) {
        if (account.getAccountNumber().equals(accountNumber)) {
            return account;
        }
    }
    return null;
}

class Account {
    private String accountNumber;
    private String accountHolderName;
    private double balance;
    private List<Transaction> transactionHistory;

    public Account(String accountNumber, String accountHolderName, double balance) {
        this.accountNumber = accountNumber;
        this.accountHolderName = accountHolderName;
        this.balance = balance;
        this.transactionHistory = new ArrayList<>();
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getAccountHolderName() {
        return accountHolderName;
    }
}
```

```
public double getBalance() {
    return balance;
}

public void deposit(double amount) {
    balance += amount;
    System.out.println("\nDeposit successfully completed. Current balance: " + balance);
}

public void withdraw(double amount) throws InsufficientBalanceException {
    if (amount > balance) {
        throw new InsufficientBalanceException("Insufficient balance. Current balance: " +
balance);
    }
    balance -= amount;
    System.out.println("Withdrawal successful. Current balance: " + balance);
}

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

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

class Transaction {
    private String type;
    private double amount;

    public Transaction(String type, double amount) {
        this.type = type;
        this.amount = amount;
    }

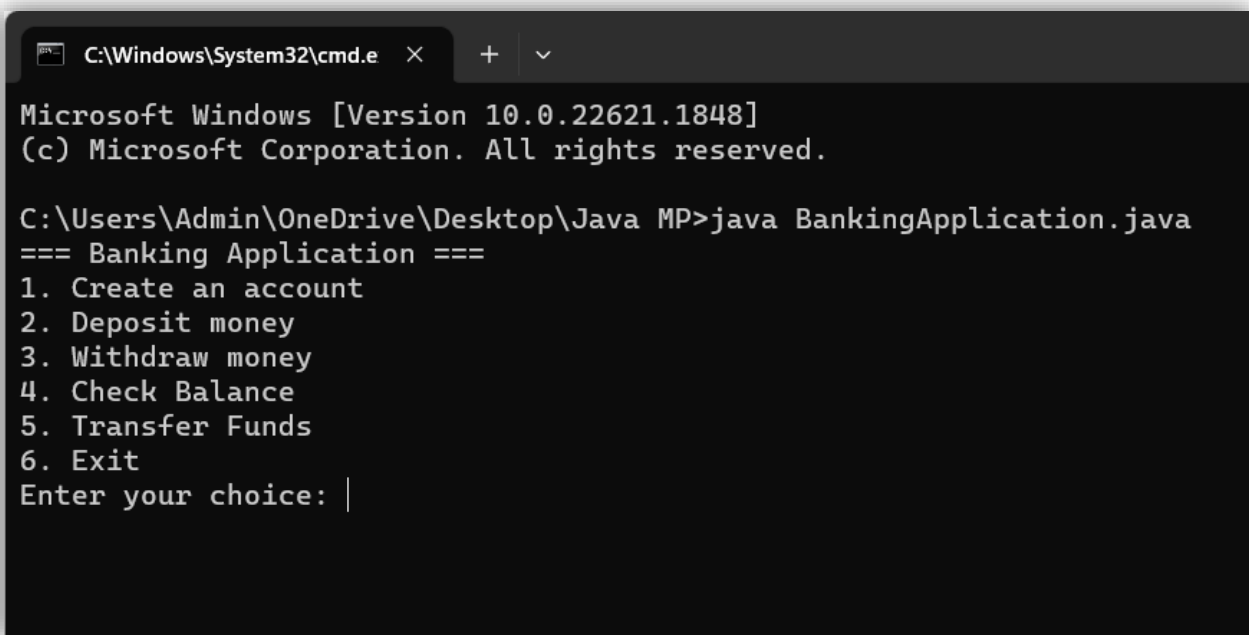
    public String getType() {
        return type;
    }

    public double getAmount() {
        return amount;
    }
}
```

```
}  
  
}  
class InsufficientBalanceException extends Exception {  
    public InsufficientBalanceException(String message) {  
        super(message);  
    }  
}
```

Output:

- Main Page output of program:



```
C:\Windows\System32\cmd.e  X  +  v  
Microsoft Windows [Version 10.0.22621.1848]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\Admin\OneDrive\Desktop\Java MP>java BankingApplication.java  
=== Banking Application ===  
1. Create an account  
2. Deposit money  
3. Withdraw money  
4. Check Balance  
5. Transfer Funds  
6. Exit  
Enter your choice: |
```

1. Creating new Account:

```
=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 1
Enter account number: 1
Enter account holder name: Sudhirkumar Kuchara
Enter initial balance: 2500

Account created successfully. Data saved in Database.dat file.
```

2. Deposit money:

```
C:\Windows\System32\cmd.e  X  +  v

=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 2
Enter account number: 1
Enter amount to deposit: 2250

Deposit successfully completed. Current balance: 4750.0
```

3. Withdraw money:

```
C:\Windows\System32\cmd.e  X  +  v

=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 3
Enter account number: 1
Enter amount to withdraw: 4500
Withdrawal successful. Current balance: 250.0
```

4. Balance checking:

```
C:\Windows\System32\cmd.e  X  +  v

Withdrawal successful. Current balance: 250.0

=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 4
Enter account number: 1
Account balance: 250.0
```

5. Transfer Funds:

- Create another account to able for Transfer.

```
C:\Windows\System32\cmd.e  X  +  v

=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 1
Enter account number: 2
Enter account holder name: Het Patel
Enter initial balance: 5000

Account created successfully. Data saved in Database.dat file.
```

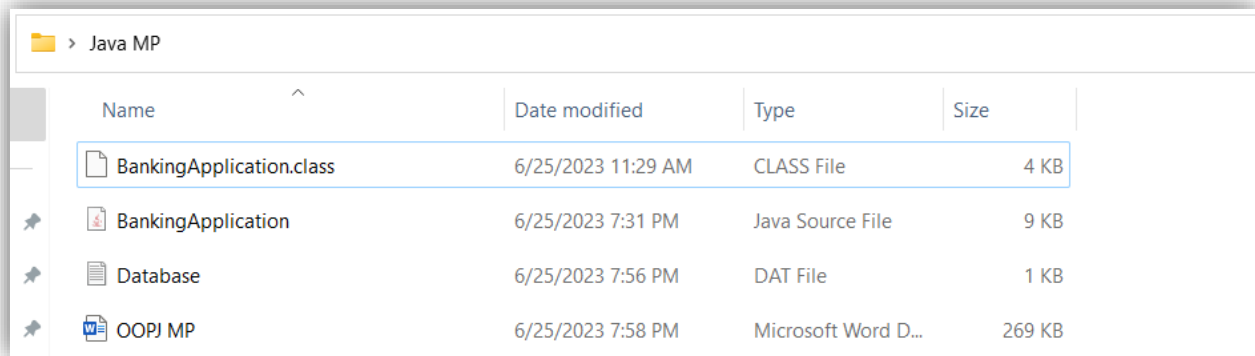
- Transferring fund:

```
C:\Windows\System32\cmd.e  X  +  v

=== Banking Application ===
1. Create an account
2. Deposit money
3. Withdraw money
4. Check Balance
5. Transfer Funds
6. Exit
Enter your choice: 5
Enter sender's account number: 2
Enter receiver's account number: 1
Enter amount to transfer: 2500
Withdrawal successful. Current balance: 2500.0

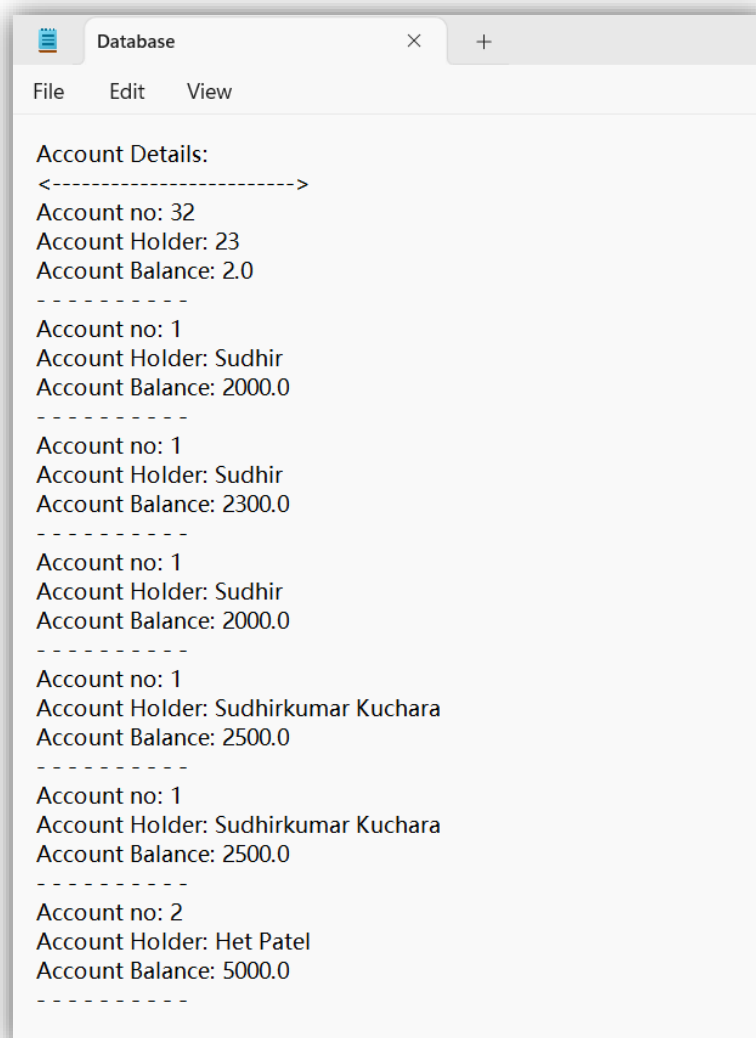
Deposit successfully completed. Current balance: 2750.0
Transfer successful.
```

- Database file in .dat extension where Account Details are written.



Name	Date modified	Type	Size
BankingApplication.class	6/25/2023 11:29 AM	CLASS File	4 KB
BankingApplication	6/25/2023 7:31 PM	Java Source File	9 KB
Database	6/25/2023 7:56 PM	DAT File	1 KB
OOPJ MP	6/25/2023 7:58 PM	Microsoft Word D...	269 KB

Name: Database.dat



```
Account Details:
<----->
Account no: 32
Account Holder: 23
Account Balance: 2.0
-----
Account no: 1
Account Holder: Sudhir
Account Balance: 2000.0
-----
Account no: 1
Account Holder: Sudhir
Account Balance: 2300.0
-----
Account no: 1
Account Holder: Sudhir
Account Balance: 2000.0
-----
Account no: 1
Account Holder: Sudhirkumar Kuchara
Account Balance: 2500.0
-----
Account no: 1
Account Holder: Sudhirkumar Kuchara
Account Balance: 2500.0
-----
Account no: 2
Account Holder: Het Patel
Account Balance: 5000.0
-----
```