INTRODUCTION

In today's digital era, managing personal finances efficiently is crucial. With the advancement of technology, banking systems have evolved to offer various services that simplify financial transactions for users. This report introduces a simple yet functional banking system application developed in Java. The primary objective of this project is to demonstrate the fundamental principles of object-oriented programming (OOP) while providing a practical solution for basic banking operations.

The system encompasses essential functionalities such as creating bank accounts, performing deposits and withdrawals, and checking account balances. It aims to provide a user-friendly interface through a command-line menu, enabling users to interact with the system seamlessly. The project is designed to handle multiple accounts and ensure data integrity through appropriate encapsulation and error handling mechanisms.

This banking system serves as an educational tool, particularly for students and novice programmers, to understand the core concepts of OOP, such as class design, encapsulation, and method implementation. It also lays the foundation for more advanced features and improvements, including persistent storage and graphical user interfaces, making it a versatile and scalable project.

The following sections of the report will delve into the detailed design, implementation, and functionality of the banking system, offering insights into its structure and operation. Through this project, we aim to highlight the practical applications of Java programming in creating efficient and reliable software solutions.

PROJECT REQUIREMENT

This section outlines the requirements necessary for developing the Simple Banking System. The requirements are categorized into functional, non-functional, and optional features to provide a comprehensive understanding of the project scope.

Functional Requirements

1. **Account Creation**
    - The system should allow the creation of new bank accounts.
    - Each account should have a unique account number, holder name, and initial balance.
2. **Deposit Funds**
    - The system should enable users to deposit funds into their accounts.
    - The system should validate that the deposit amount is positive.
3. **Withdraw Funds**
    - The system should allow users to withdraw funds from their accounts.
    - The system should ensure that the withdrawal amount does not exceed the current balance.
    - The system should validate that the withdrawal amount is positive.
4. **Check Balance**
    - The system should provide functionality to check the current balance of any account.
5. **Find Account**
    - The system should allow users to find and retrieve account details using the account number.
6. **User Interface**
    - The system should have a command-line interface that allows users to interact with the banking system.
    - The menu should provide options to create accounts, deposit funds, withdraw funds, check balances, and exit the application.

Non-Functional Requirements

1. **Usability**

- The system should have an easy-to-use command-line interface.
- Clear instructions and messages should be provided for user actions.

2. **Performance**
    - The system should handle up to 10 accounts efficiently.
    - The system should provide quick responses to user inputs.
3. **Reliability**
    - The system should handle invalid inputs gracefully, providing appropriate error messages.
    - The system should ensure the integrity of data through proper validation checks.
4. **Scalability**
    - The design should allow for the easy addition of new features, such as different account types or extended functionality.
5. **Maintainability**
    - The code should be well-documented and modular to facilitate easy maintenance and updates.

Optional Features

1. **Account Types**
    - Introduce different types of accounts, such as Savings and Current accounts, with specific features and behaviors.
2. **Transaction History**
    - Implement functionality to maintain and display a history of transactions for each account.
3. **Graphical User Interface (GUI)**
    - Develop a graphical interface using JavaFX or Swing to enhance user experience.
4. **Persistence**
    - Add the ability to save account data to a file or database to maintain account information between sessions.
5. **Security**
    - Implement basic security measures, such as password protection for accounts.

UML DIAGRAM

Account
-accountNumber : long - holderName : String - balance : double
+ Account(accountNumber: long, holderName: String, initialBalance: double) + getAccountNumber() : long + getHolderName() : String + getBalance() : double + deposit(amount: double) : void + withdraw(amount: double) : void

BankingSystem
- accounts : Account[] - accountCount : int
+BankingSystem(size: int) +createAccount(accountNumber:long, holderName: String, initialBalance: double) : void +findAccount(accountNumber:long):Account + deposit(accountNumber: long, amount: double) : void + withdraw(accountNumber: long, amount: double) : void + checkBalance(accountNumber: long) : void

MainTestCase
+main(String[] args): void

SOURCE CODE

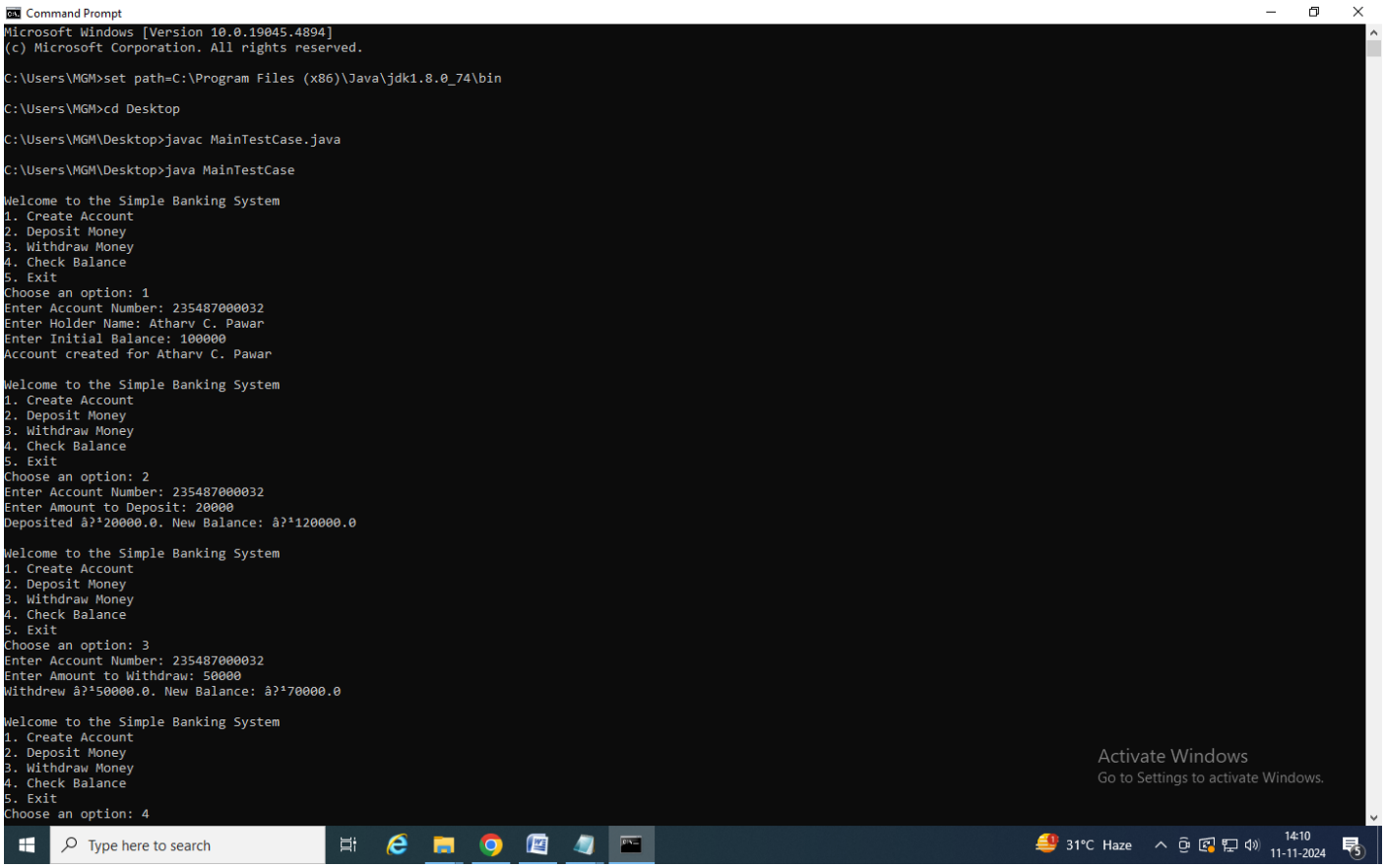
```
public class Account {
    private long accountNumber;
    private String holderName;
    private double balance;

    public Account(long accountNumber, String holderName, double initialBalance) {
        this.accountNumber = accountNumber;
        this.holderName = holderName;
        this.balance = initialBalance;
    }

    public long getAccountNumber() {
        return accountNumber;
    }

    public String getHolderName() {
        return holderName;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited $" + amount + ". New Balance: $" + balance);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
```

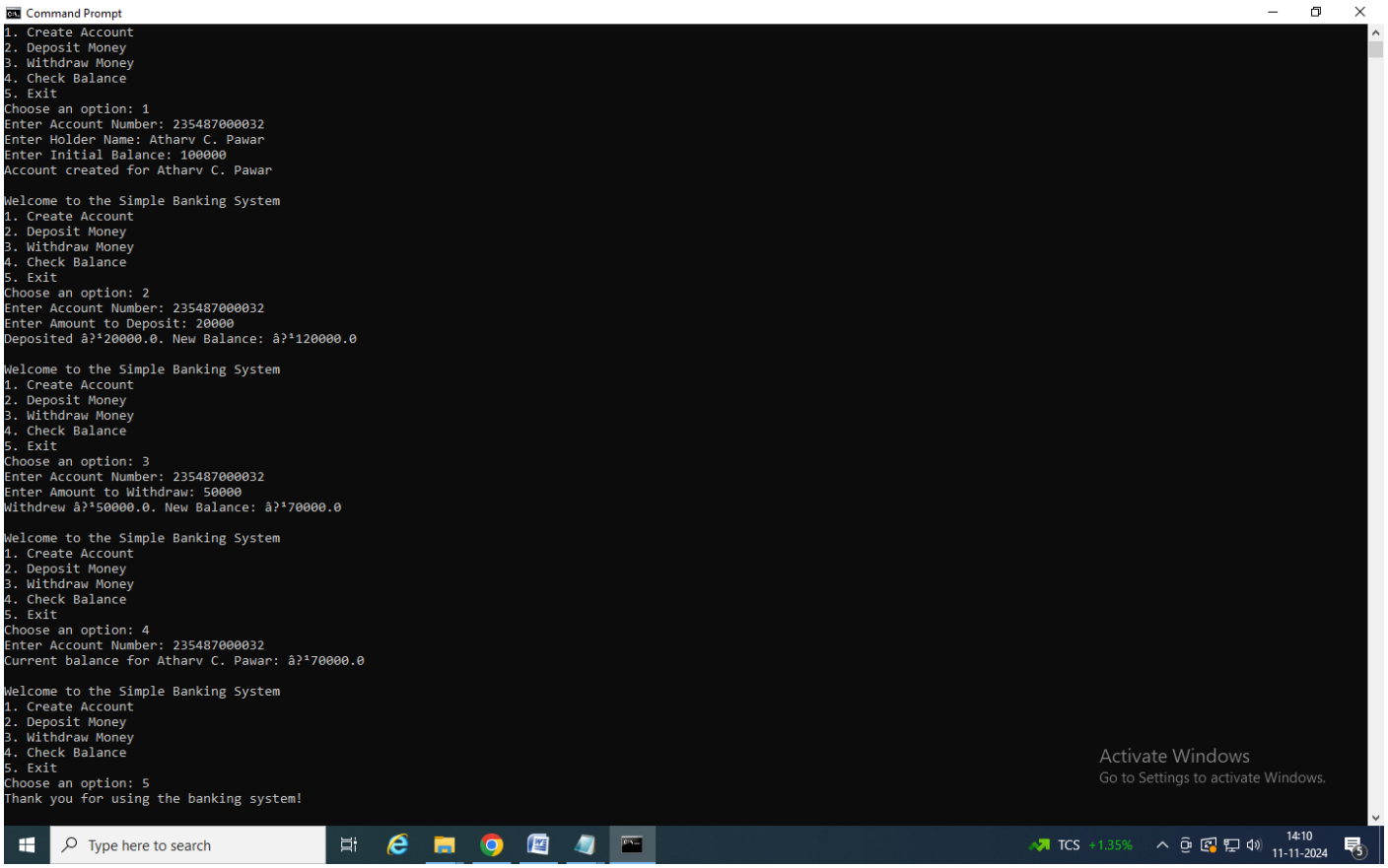
```
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew $" + amount + ". New Balance: $" + balance);
        } else if (amount > balance) {
            System.out.println("Insufficient funds.");
        } else {
            System.out.println("Withdrawal amount must be positive.");
        }
    }
}
import java.util.Scanner;

public class BankingSystem {
    private Account[] accounts;
    private int accountCount;

    public BankingSystem(int size) {
        accounts = new Account[size];
        accountCount = 0;
    }

    public void createAccount(long accountNumber, String holderName, double initialBalance) {
        if (accountCount < accounts.length) {
            accounts[accountCount++] = new Account(accountNumber, holderName, initialBalance);
            System.out.println("Account created for " + holderName);
        } else {
            System.out.println("Maximum account limit reached.");
        }
    }
```

```
    public Account findAccount(long accountNumber) {
        for (int i = 0; i < accountCount; i++) {
            if (accounts[i].getAccountNumber() == accountNumber) {
                return accounts[i];
            }
        }
        return null; // Account not found
    }

    public void deposit(long accountNumber, double amount) {
        Account account = findAccount(accountNumber);
        if (account != null) {
            account.deposit(amount);
        } else {
            System.out.println("Account not found.");
        }
    }

    public void withdraw(long accountNumber, double amount) {
        Account account = findAccount(accountNumber);
        if (account != null) {
            account.withdraw(amount);
        } else {
            System.out.println("Account not found.");
        }
    }

    public void checkBalance(long accountNumber) {
        Account account = findAccount(accountNumber);
        if (account != null) {
            System.out.println("Current balance for " + account.getHolderName() + ": $" +
account.getBalance());
        } else {
```

```
            System.out.println("Account not found.");
        }
    }
}
import java.util.Scanner;
public class MainTestCase {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BankingSystem bank = new BankingSystem(10); // Limit to 10 accounts

        while (true) {
            System.out.println("\nWelcome to the Simple Banking System");
            System.out.println("1. Create Account");
            System.out.println("2. Deposit Money");
            System.out.println("3. Withdraw Money");
            System.out.println("4. Check Balance");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");

            int option = scanner.nextInt();

            switch (option) {
                case 1:
                    System.out.print("Enter Account Number: ");
                    long accNum = scanner.nextLong();
                    scanner.nextLine(); // Consume newline
                    System.out.print("Enter Holder Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Initial Balance: ");
                    double initialBalance = scanner.nextDouble();
                    bank.createAccount(accNum, name, initialBalance);
                    break;
```

```
            case 2:
                System.out.print("Enter Account Number: ");
                accNum = scanner.nextLong();
                System.out.print("Enter Amount to Deposit: ");
                double depositAmount = scanner.nextDouble();
                bank.deposit(accNum, depositAmount);
                break;

            case 3:
                System.out.print("Enter Account Number: ");
                accNum = scanner.nextLong();
                System.out.print("Enter Amount to Withdraw: ");
                double withdrawAmount = scanner.nextDouble();
                bank.withdraw(accNum, withdrawAmount);
                break;

            case 4:
                System.out.print("Enter Account Number: ");
                accNum = scanner.nextLong();
                bank.checkBalance(accNum);
                break;

            case 5:
                System.out.println("Thank you for using the banking system!");
                scanner.close();
                return;

            default:
                System.out.println("Invalid option. Please try again.");
        }
    }
}
}
```


Command Prompt
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.
C:\Users\MGM>set path=C:\Program Files (x86)\Java\jdk1.8.0_74\bin
C:\Users\MGM>cd Desktop
C:\Users\MGM\Desktop>javac MainTestCase.java
C:\Users\MGM\Desktop>java MainTestCase
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 1
Enter Account Number: 235487000032
Enter Holder Name: Atharv C. Pawar
Enter Initial Balance: 100000
Account created for Atharv C. Pawar
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 2
Enter Account Number: 235487000032
Enter Amount to Deposit: 20000
Deposited â?¹20000.0. New Balance: â?¹120000.0
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 3
Enter Account Number: 235487000032
Enter Amount to Withdraw: 50000
Withdrew â?¹50000.0. New Balance: â?¹70000.0
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 4
Activate Windows
Go to Settings to activate Windows.
Type here to search
31°C Haze
14:10
11-11-2024


Command Prompt
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 1
Enter Account Number: 235487000032
Enter Holder Name: Atharv C. Pawar
Enter Initial Balance: 100000
Account created for Atharv C. Pawar
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 2
Enter Account Number: 235487000032
Enter Amount to Deposit: 20000
Deposited â?¹20000.0. New Balance: â?¹120000.0
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 3
Enter Account Number: 235487000032
Enter Amount to Withdraw: 50000
Withdrew â?¹50000.0. New Balance: â?¹70000.0
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 4
Enter Account Number: 235487000032
Current balance for Atharv C. Pawar: â?¹70000.0
Welcome to the Simple Banking System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Exit
Choose an option: 5
Thank you for using the banking system!
Activate Windows
Go to Settings to activate Windows.
Type here to search
TCS +1.35%
14:10
11-11-2024

CONCLUSION

The development of the Simple Banking System in Java provides a comprehensive understanding of core programming concepts, including object-oriented design, encapsulation, and data management. The project successfully meets its primary objectives by facilitating fundamental banking operations such as account creation, deposits, withdrawals, and balance inquiries through a user-friendly command-line interface.

This project demonstrates the practical application of Java programming to create a functional and reliable system, emphasizing the importance of clear user communication and robust error handling. Additionally, the modular and scalable design lays the groundwork for future enhancements, such as the introduction of different account types, transaction history tracking, graphical user interfaces, and data persistence mechanisms.

In essence, the Simple Banking System serves as an educational tool that not only reinforces fundamental programming skills but also highlights the potential for further development and real-world application. The structured approach and adherence to project requirements ensure that the system is both efficient and user-friendly, making it a valuable exercise for novice programmers and a solid foundation for more advanced projects.

By extending and refining this project, developers can explore more sophisticated features and technologies, thereby enhancing their skills and contributing to the creation of more complex and capable software solutions. The Simple Banking System exemplifies how basic programming principles can be leveraged to address practical problems and create meaningful, functional applications.