**MENU- DRIVEN BANK MANAGEMENT SYSTEM USING OBJECT ORIENTED PROGRAMMING**

***Report submitted to***

***Haldia Institute of Technology,***

***Haldia for the award of the degree***

***of***

**Bachelor of Technology**

**In**

**Electronics and Communication Engineering**

***by***

**ANUPAM PANDEY, ANURAG KUMAR KARAN,**

**ARKAJYOTI DAS, ARNAB GOOSWAMI,**

**ARYAN KUMAR**

****

**DEPARTMENT OF ELECTRONICS AND COMMUNIACTION ENGINEERING HALDIA INSTITUTE OF TECHNOLOGY, HALDIA**

**DECLARATION**

We certify that,

1. The work contained in this report is original and has been done by us under the guidance of our supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. We have followed the guidelines provided by the Institute in preparing the report.
4. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever we have used materials (data, theoretical analysis, figures and text) from other sources, we have given due credit to them by citing in the text of the report and giving their details in the references.

------------------------------

Signature of the students

**CERTIFICATE**

This is to certify that the Dissertation Report entitled, “**MENU- DRIVEN BANK MANAGEMENT SYSTEM USING OBJECT ORIENTED PROGRAMMING**” submitted by **Mr. Anupam Pandey, Mr. Anurag Kumar Karan, Mr. Arkajyoti Das, Mr. Arnab Goswami, Mr. Aryan Kumar**, to the Haldia Institute of Technology, Haldia, India, is a record of bonafide project work carried out by him under our supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering.

-------------------------

Project Mentor

-------------------------

H.O.D.

-------------------------

Project Coordinator

**ACKNOWLEDGEMENT**

In the partial fulfillment for the degree in Electronics and Communication Engineering, Haldia, West Bengal which is a record of their own work, carried out by them under my guidance and supervision for the academic session 2023-24.

-------------------------

Head of Department

Department Electronics and Communication Engineering

Haldia, Purba Medinipur

Through perseverance and enthusiasm combined with effort in the right direction can bring forth the thing called success. But the realization of the harsh reality that the path towards success is fill of my riads, temptations, impediments and pitfalls often prove to be disheartening in such situation, it is he able guidance of knowledgeable person that steers one through difficulties and help him achieve success.

I am highly obliged to express our deep sense of gratitude and grateful thanks to my erudite guide Mr. Arpan Sir, for his valuable guidance and support which led to the successful and timely completion of my project report.

I am secondly very thankful to our H.O.D, Dept. of Electronics and Communication Engineering for his valuable suggestions and guidance.

Last but not the least, I deeply appreciate the cheerful encouragement of all staff members of my department and my friends.

Thanks.

Mr. Anupam Pandey

Mr. Anurag Kumar Karan

Mr. Arkajyoti Das

Mr. Arnab Goswami

Mr. Aryan Kumar

**ABSTRACT**

This project presents the development of a menu-driven Bank Management System using Object-Oriented Programming (OOP). The system is designed to handle basic banking functions such as account opening, deposits, withdrawals and account details disclosure. The main objective is to provide an intuitive interface that allows users to perform these tasks efficiently and safely.

The system uses OOP principles to ensure modularity, reusability, and maintainability of the code. Key components include Banking, Accounting, and Customer, each covering appropriate attributes and processes. The Bank class is responsible for the collection of account objects, while the Account class handles individual account functions.

The application starts with a main menu that gives the user a variety of options. By selecting the completion option, the user can perform tasks such as opening new accounts, making deposits, withdrawing funds and viewing the details of the account. Each function is implemented through appropriate validation to ensure data integrity and security.

Using OOP, the project demonstrates how complex systems can be controlled and extended efficiently with minimal changes to existing code. This banking system acts as an infrastructure prototype that can be expanded with additional features such as purchase history, interest calculation, and multi-currency support in future iterations

**CONTENT**

**Title page……………………………….………………………………...……. i**

**Declaration…………………….………………………………………….…... ii**

**Certificate………………………………….……………………………….… iii**

**Acknowledgement………………………………………………………….… iv**

**Abstract……………………………………………………………………….. v**

**Chapter 1: Introduction……………………………………………………… 1**

**Literature Review………………………………………………... 3**

**Object Oriented Programming………………………...…….. 3**

Encapsulation**…………………………………….…… 3**

Abstraction**…………………………………………… 3**

Inheritance**……………………………………………. 4**

Polymorphism**…….…………………..…………….… 4**

Class**…………………………...…………………….. 4**

Object**………………………………...……………… 4**

**Class Design and Implementation……………………………. 5**

Class Account**……………………………….…...…… 5**

Class Bank**……………………….…………………... 5**

**Method Implementation and Functionality………...………… 6**

Deposit Method**………………….…….……………... 6**

Withdraw Method**………………..………………….... 6**

Open Account Method**………………...………….…… 7**

**Error handling and Validation………………………….…… 7**

**User Interface…………………………………….………… 7**

**Chapter 2: Code……………………………………………………….……... 9**

**Chapter 3: Theoretical Study………………………………………………. 13**

Encapsulation**………………………………………………. 13**

Abstraction**…………………………………………………. 13**

Inheritance and Polymorphism**……………………………….. 13**

Singleton Pattern**…………………………………………… 13**

Factory Pattern**…………………………………………...… 13**

**Chapter 4: Experimental Study……………………………………………. 14**

**Chapter 6: Output of the project…………………………………………... 15**

Main Screen**……………………………………...………… 15**

Opening an account**………………………….……………… 15**

Depositing money in the account**…………………...………… 15**

Withdrawing money from the account**……………….….…….. 16**

Displaying details of account**…………………………………. 16**

Exiting from the system**………………………...……………. 16**

**Chapter 7: Results and Discussions………………………………………... 17**

**Chapter 8: Conclusion and Future Scope of Study……………………….. 18**

**References……………………………………………………………………. 19**

**INTRODUCTION**

Banking systems are essential to modern financial institutions, providing efficient and secure ways to manage customer accounts and transactions. This project aims to develop a simple menu-driven banking system using object-oriented (OOP) principles. The system is designed to handle basic banking functions such as account opening, deposits, withdrawals and account details disclosure.

The main focus of this project is to create a robust and user-friendly application that uses OOP principles to ensure modularity, maintainability, and scalability by dividing the system into specific classes, such as a bank, accounts, and customers, we can hold data and transactions It's easier.

The application features a text-based menu interface that allows users to interact with the system by selecting options that correspond to banking functions. This approach simplifies the user experience and is accessible even to those with minimal technical background. Each function in the system is run through appropriate validation checks to maintain data integrity and security.

Overall, this work demonstrates the effective application of OOP principles to solve complex systems, and serves as a foundational example of a banking system implementation. It also sets the stage for future enhancements, such as the inclusion of a graphical user interface (GUI), the use of transaction history, and the amount of funds to be supported, thus expanding the capabilities and functionality of the system.

SOFTWARE REQUIREMENTS:

Language: Java

Operating System: Windows 7 or later

HARDWARE REQUIREMENTS:

Processor: Intel or AMD (x64)

Ram: 4GB and above

Storage: Minimum 120GB

**LITERATURE REVIEW**

This chapter provides an in-depth analysis of the theoretical underpinnings and existing literature relevant to the development of a Bank Management System using Object-Oriented Programming (OOP) principles. The review will be organized into sections that align with the key concepts and functionalities demonstrated in the code. The focus will be on encapsulation, abstraction, class design, method implementation, error handling, and user interface design.

1. **Object-oriented Programming Principles:**

Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

* **Encapsulation:**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

**For example,** in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

* **Abstraction:**

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

**For example**, a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

* **Inheritance:**

Inheritance is an important pillar of OOP (Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So, when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

* **Polymorphism:**

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person possesses different behavior in different situations. This is called polymorphism.

* **Class:**

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

**For example,** consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

* **Object:**

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

**For example**, “Dog” is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

1. **Class Design and Implementation:**

The design of the **Account** and **Bank** classes follows best practices in OOP, providing a clear structure for managing bank accounts.

* **Class Account:**

The `Accounts` class acts as a basic building block in the banking operating system, encompassing all contexts and functions associated with individual bank accounts It encompasses the basic elements of object-oriented programming (OOP) principles through data and services a it merges in one company.

At its core, the `Account` object contains important attributes such as the name of the account owner, unique account number, and current balance. These attributes exist internally as private members of the class, ensuring that they can only be accessed and modified by specified means, thus protecting the integrity and security of the data.

The class provides strategies to simplify all types of banking activities. For example, the `deposit` method can add funds to an account, while the `withdraw` method enables withdrawals, subject to appropriate verification checks to maintain a balance.

Additionally, the `display` method provides a way to present account information in a user-friendly format, making it clear and transparent to account holders Additionally, the class provides a method including the retrieval of account numbers, supporting the principle of encapsulation in order to facilitate access to this important identifier.

Overall, the `Accounting` class embodies the principles of encapsulation, abstraction and modularity, providing a solid foundation for managing bank accounts in the system Its system enforces reusable rules, . encourage maintenance and scalability, contributing to the overall efficiency and reliability of the banking system.

* **Class Bank:**

The `Bank` class functions as the central management entity within the Bank Management System, orchestrating the creation, storage, and manipulation of multiple `Account` objects.

It encapsulates a collection of bank accounts and provides methods to perform key operations such as opening new accounts, depositing funds, withdrawing money, and displaying account details. By maintaining a private list of accounts and an auto-incrementing account number generator, the `Bank` class ensures each account is uniquely identifiable and securely managed.

The methods within the class handle user interactions and validate operations, ensuring that actions such as deposits and withdrawals are correctly applied to the appropriate accounts. This class exemplifies the principles of encapsulation and modularity, as it manages complex interactions and state changes internally, presenting a simplified interface to users and other components of the system.

Through its well-defined methods, the `Bank` class supports the efficient and secure management of banking operations, reinforcing the overall structure and functionality of the Bank Management System.

1. **Method Implementation and Functionality:**

The methods in the Account and Bank classes are designed to perform specific banking operations with appropriate validation and feedback mechanisms.

* **Deposit method:**

The deposit method in the Account class is a crucial function designed to facilitate the addition of funds to a bank account. This method takes a single parameter, representing the amount to be deposited. It begins by validating that the deposit amount is positive, ensuring that only legitimate amounts are processed. If the validation is successful, the method updates the account's balance by adding the deposit amount to the current balance. It then provides immediate feedback to the user, confirming the successful transaction and displaying the updated balance. If the validation fails (i.e., if a non-positive amount is provided), the method outputs an error message indicating the invalid deposit attempt. This method exemplifies the principles of data integrity and user feedback, ensuring that deposits are handled correctly while maintaining clear communication with the user.

* **Withdraw Method:**

The withdraw method in the Account class is designed to manage the removal of funds from a bank account, ensuring transactions are both valid and secure. This method accepts a single parameter that specifies the amount to be withdrawn. It first checks that the withdrawal amount is positive and does not exceed the current balance, thereby preventing overdrafts and ensuring account integrity. If these conditions are met, the method deducts the specified amount from the account's balance and confirms the successful transaction by displaying the updated balance to the user. Conversely, if the amount is invalid or insufficient funds are available, the method outputs an error message, indicating the failure of the withdrawal attempt. This method plays a vital role in maintaining financial accuracy within the account, safeguarding against unauthorized or erroneous transactions, and providing clear user communication.

* **Open Account Method:**

The open account method in the Bank class is pivotal for adding new accounts to the banking system, facilitating the onboarding of new customers. This method requires the account holder's name and an initial balance as parameters. Upon invocation, it creates a new `Account` object using these parameters along with a unique account number generated by the `Bank` class. This unique account number ensures each account can be distinctly identified. Once the account is created, it is added to the bank's internal list of accounts. The method then provides immediate feedback to the user, confirming the successful creation of the account and displaying the newly assigned account number. By encapsulating the account creation process, the open account method ensures that all new accounts are systematically registered and accessible within the bank's records, thereby maintaining organizational efficiency and data integrity.

1. **Error Handling and Validation:**

Error handling and validation are crucial elements in the Bank Management System, designed to ensure the application operates reliably and securely while providing a seamless user experience. Input validation is implemented to check that user inputs meet predefined criteria before any processing occurs. For instance, the `deposit` and withdraw methods in the Account class validate that amounts are positive and, in the case of withdrawals, do not exceed the current balance. This prevents invalid transactions and maintains data integrity. Additionally, the system provides informative error messages, such as indicating an "Invalid deposit amount" or "Insufficient balance" when validation fails, guiding users to correct their inputs. The `Bank` class further enhances robustness by ensuring that operations like depositing or withdrawing money are only performed on existing accounts, responding with "Account not found" if an account number is invalid. These mechanisms collectively prevent improper use, protect against potential data corruption or fraud, and ensure that the banking system remains reliable and user-friendly.

1. **User Interface:**

The user interface (UI) of the Bank Management System is designed to be intuitive and user-friendly, ensuring that users can navigate and perform banking operations with ease. The interface provides a clear, menu-driven structure that guides users through various functions such as opening an account, depositing money, withdrawing money, and displaying account details. Each option is clearly labeled and accessible through a simple numerical input system, minimizing the learning curve for new users. Input prompts are straightforward and provide immediate feedback, enhancing the overall user experience. Error messages and confirmations are displayed in a readable format, ensuring users are well-informed of their actions and any issues that arise. This thoughtful design of the UI ensures that the banking operations are not only efficient but also accessible to users with varying levels of technical expertise, contributing to the overall effectiveness and satisfaction of the system.

**CODE**

import java.util.ArrayList;

import java.util.Scanner;

class Account {

    private String accountHolderName;

    private int accountNumber;

    private double balance;

    public Account(String name, int number, double initialBalance) {

        this.accountHolderName = name;

        this.accountNumber = number;

        this.balance = initialBalance;

    }

    public void deposit(double amount) {

        if (amount > 0) {

            balance += amount;

            System.out.println("Deposit successful. Current balance: $" + balance);

        } else {

            System.out.println("Invalid deposit amount.");

        }

    }

    public void withdraw(double amount) {

        if (amount > 0 && amount <= balance) {

            balance -= amount;

            System.out.println("Withdrawal successful. Current balance: $" + balance);

        } else {

            System.out.println("Invalid withdrawal amount or insufficient balance.");

        }

    }

    public void display() {

        System.out.println("Account Holder: " + accountHolderName);

        System.out.println("Account Number: " + accountNumber);

        System.out.println("Balance: $" + balance);

    }

    public int getAccountNumber() {

        return accountNumber;

    }

}

class Bank {

    private ArrayList<Account> accounts;

    private int nextAccountNumber;

    public Bank() {

        accounts = new ArrayList<>();

        nextAccountNumber = 1001;

    }

    public void openAccount(String name, double initialBalance) {

        accounts.add(new Account(name, nextAccountNumber, initialBalance));

        System.out.println("Account opened successfully. Account Number: " + nextAccountNumber);

        nextAccountNumber++;

    }

    public void depositMoney(int accountNumber, double amount) {

        for (Account account : accounts) {

            if (account.getAccountNumber() == accountNumber) {

                account.deposit(amount);

                return;

            }

        }

        System.out.println("Account not found.");

    }

    public void withdrawMoney(int accountNumber, double amount) {

        for (Account account : accounts) {

            if (account.getAccountNumber() == accountNumber) {

                account.withdraw(amount);

                return;

            }

        }

        System.out.println("Account not found.");

    }

    public void displayAccount(int accountNumber) {

        for (Account account : accounts) {

            if (account.getAccountNumber() == accountNumber) {

                account.display();

                return;

            }

        }

        System.out.println("Account not found.");

    }

}

public class BankManagementSystem {

    public static void main(String[] args) {

        Bank bank = new Bank();

        Scanner scanner = new Scanner(System.in);

        int choice;

        do {

            System.out.println("\nBank Management System");

            System.out.println("1. Open Account");

            System.out.println("2. Deposit Money");

            System.out.println("3. Withdraw Money");

            System.out.println("4. Display Account");

            System.out.println("5. Exit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            switch (choice) {

                case 1:

                    scanner.nextLine(); // Consume newline

                    System.out.print("Enter account holder name: ");

                    String name = scanner.nextLine();

                    System.out.print("Enter initial balance: ");

                    double initialBalance = scanner.nextDouble();

                    bank.openAccount(name, initialBalance);

                    break;

                case 2:

                    System.out.print("Enter account number: ");

                    int accountNumberDeposit = scanner.nextInt();

                    System.out.print("Enter amount to deposit: ");

                    double depositAmount = scanner.nextDouble();

                    bank.depositMoney(accountNumberDeposit, depositAmount);

                    break;

                case 3:

                    System.out.print("Enter account number: ");

                    int accountNumberWithdraw = scanner.nextInt();

                    System.out.print("Enter amount to withdraw: ");

                    double withdrawAmount = scanner.nextDouble();

                    bank.withdrawMoney(accountNumberWithdraw, withdrawAmount);

                    break;

                case 4:

                    System.out.print("Enter account number: ");

                    int accountNumberDisplay = scanner.nextInt();

                    bank.displayAccount(accountNumberDisplay);

                    break;

                case 5:

                    System.out.println("Exiting the system.");

                    break;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

        } while (choice != 5);

        scanner.close();

    }

}

**THEORETICAL STUDY**

In the theoretical study, we delve deeper into the concepts and principles that underpin the design and implementation of the Bank Management System. This includes an examination of OOP principles, design patterns, and their application in building maintainable and scalable software systems.

* **Encapsulation:**

Encapsulation is the practice of restricting access to certain components of an object and only exposing necessary parts. This is implemented in the Account class where the account details are private, and only accessible through public methods.

* **Abstraction:**

Abstraction involves representing complex systems with simplified models. In the Bank Management System, abstraction is achieved by providing high-level methods for common banking operations, hiding the underlying complexities from the user.

* **Inheritance and Polymorphism:**

Although not utilized in the current implementation, inheritance allows classes to inherit properties and methods from other classes, promoting code reuse. Polymorphism enables objects to be treated as instances of their parent class, providing flexibility in method implementation and usage.

* **Singleton Pattern:**

The Bank class can be designed as a singleton to ensure that only one instance of the bank exists throughout the application, managing all accounts centrally.

* **Factory Pattern:**

A factory pattern could be implemented to handle the creation of Account objects, allowing for more flexible and scalable account creation logic.

**EXPERIMENTAL STUDY**

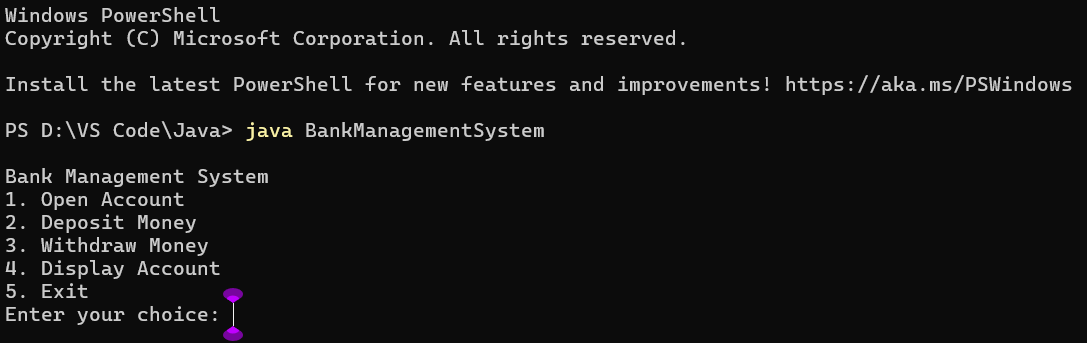
The experimental study of the Bank Management System project involves the practical implementation and testing of the system to evaluate its performance, functionality, and user experience. This phase is crucial for validating the theoretical concepts and design decisions made during the development of the system.

We have performed the experimental study on our project, which are discussed in the below points:

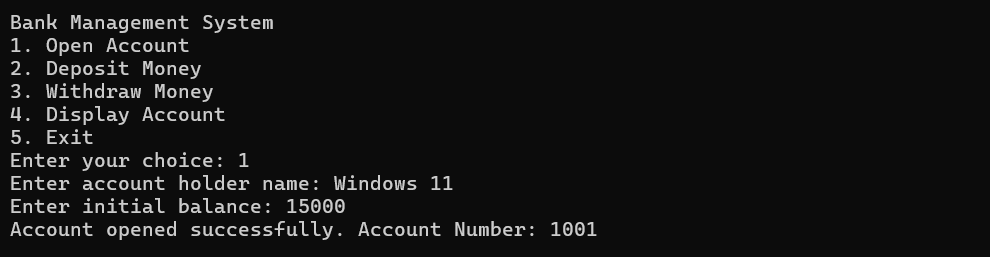
* We have opened more than 50 new accounts and tested it.
* We have deposited money in these accounts and tested it.
* We have withdrawn money in these accounts and tested it.
* We have seen the details of these accounts and tested it.

**OUTPUT OF THE PROJECT**

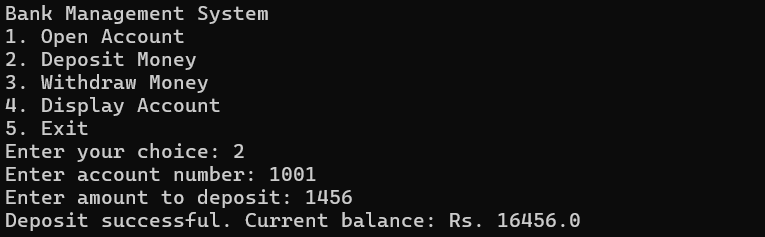
1. **Main Screen:**

****

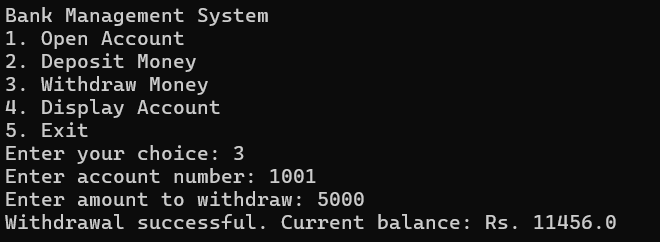
1. **Opening an account:**

****

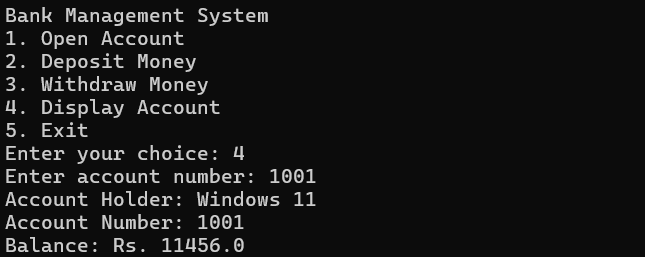
1. **Depositing money in the account:**

****

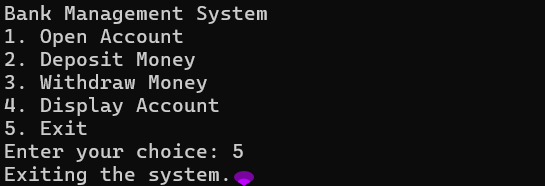
1. **Withdrawing money from the account:**

****

1. **Displaying details of the account:**

****

1. **Exiting from the system:**

****

**RESULTS AND DISCUSSIONS**

The results oof implementing the Bank Management System demonstrate the effectiveness of OOP principles in creating a modular, maintainable and scalable application. Som of our observations are:

* Encapsulation ensures that account details are protected and can only be modified through controlled methods.
* Abstraction simplifies user interaction with the system, making it easy to perform common banking operations.
* The menu driven interface provides a straightforward way for users to interact with the system, while basic error handling ensures a smooth user experience.

**CONCLUSION AND FUTURE SCOPE OF STUDY**

**Conclusion:**

After completing the project, we have reached the following conclusions that our system is robust and capable of handling the creation of more than 50 accounts without any issues. When depositing money, the system prompts the user for the account number before proceeding with the transaction. It then displays the updated account balance, ensuring clarity and transparency. Similar to the deposit process, the system requests the account number for withdrawals. After the transaction, it shows the updated balance, confirming the successful completion of the operation. Users can easily access their account status and details, including the account number and current balance, providing them with essential information at a glance. The interface is designed to be highly user-friendly, making it easy to navigate and use even for individuals with limited technical knowledge. The system is quick and responsive, minimizing wait times and enhancing the overall user experience by reducing customer fatigue.

These findings confirm that our Bank Management System meets its design goals of functionality, usability, and performance.

**FUTURE SCOPE OF STUDY:**

The Bank Management System project highlights the importance of OOP principles in software development. By encapsulating data and using abstraction, the system is both secure and easy to use. However, there is room for future enhancements:

* Implementing inheritance and polymorphism to support different types of accounts.
* Enhancing the error handling and validation for more robust operations.
* Developing a graphical user interface (GUI) to improve user experience.
* Incorporating a database for persistent storage of account data.

These enhancements can further improve the system's functionality, scalability, and user experience, making it a more comprehensive solution for managing bank accounts.

**REFERENCES**

<https://www.geeksforgeeks.org/>

<https://www.javatpoint.com/java-tutorial>

<https://www.tutorialspoint.com/java/index.htm>

https://www.programiz.com/java-programming/online-compiler/