NAME : Pratyush Jha

ROll no. 2401201017

	School of Engineering & Technology	
	Department: SOET	Session: 2025-26
	Programme: B.Tech. , BCA, BSc.	Semester: 3rd
	Course Code: ENCS201, ENCA 203, ENBC205	Number of students:
	Course Name: Java Programming	Faculty: Dr. Manish Kumar

Assignment Number: 01

Instructions:

- Assignment needs to be submitted by given date
- Assignment must be submitted on (https://lms.krmangalam.edu.in/)
- Use of ChatGPT and similar tools is strictly prohibited.
- Assignment must be written on dedicated assignment copy for Java
- Programming subject.
- All assignments must be prepared as per the format shared in classes
- Assignments need to be submitted by each individual.
  Total marks:5 m arks

- This assignment contributes to total 10% of internal evaluation.
- Submission Requirements: Submit individually via **GitHub** and provide the link of submission by **5th September,2025**.
  Assignments will be evaluated on the basis of the following metrics.
  - Originality
  - Correctness
  - Completeness

Sr.no	Assignment Details	COs
1.	**Project Title: Banking Application for Account Management** **Problem Statement:** Design andimplement a banking application that allows users to manage their bank accounts through various operations such as creating accounts, depositing money, withdrawing money, and viewing account details. The application should incorporate the concepts of Java features, control structures, arrays, and strings as per the syllabus. **Project Objectives:** • Apply the basics of Java programming including data types, operators, control structures, and type casting. • Utilize Java's control structures for decision making and looping. • Implement basic I/O operations using the Scanner class. • Handle single and multi-dimensional arrays. • Manipulate strings using Java's String class and methods. **Learning Outcomes:**	CO1

- Develop a foundational understanding of Java programming.
- Gain practical experience with Java control structures and data handling.
- Implement real-world applications using Java arrays and strings.
- Learn to manage and share code using GitHub.

Project Instructions:

1. **Account Class Design:**
   - **Attributes:**
     - accountNumber: Integer, Unique account number.
     - accountHolderName: String, Name of the account holder.
     - balance: Double, Current balance in the account.
     - email: String, Email address of the account holder.
     - phoneNumber: String, Contact number of the account holder.
   - **Methods:**
     - deposit(double amount): Method to deposit money into the account. It should validate the amount to be positive.
     - withdraw(double amount): Method to withdraw money from the account. It should validate the amount to be positive and ensure sufficient balance.
     - displayAccountDetails(): Method to display the current account details.
     - updateContactDetails(String email, String phoneNumber): Method to update the contact details of the account holder.
2. **User InterfaceClass:**
   - **Attributes:**
     - Array to store multiple Account objects.
     - Scanner object for input.
   - **Methods:**
     - createAccount(): Method to create a new account. Collects account details from the user and stores them in the array.
     - performDeposit(): Method to handle deposit operations. Prompts for account number and amount to deposit.
     - performWithdrawal(): Method to handle withdrawal operations. Prompts for account number and amount to withdraw.
     - showAccountDetails(): Method to display account details. Prompts for account number.
     - updateContact(): Method to update contact details. Prompts for account number and new contact details.
     - mainMenu(): Method to display the main menu and handle user choices.
3. **Implementation Steps:**
   - Define the Account class with necessary attributes and methods.

- Create the UserInterface class to interact with users and manage multiple accounts.
- Implement the methods to handle different operations such as account creation, deposit, withdrawal, and display details.
- Utilize Java control structures for decision making and loops to navigate through the menu and handle various user inputs.
- Use arrays to manage multiple accounts dynamically.
- Employ string handling techniques for managing account holder names and contact details.

4. **Sample Interaction:**

Welcome to the Banking Application!
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 1
Enter account holder name: John Doe
Enter initial deposit amount: 1000.0
Enter email address: john.doe@example.com
Enter phone number: 1234567890
Account created successfully with Account Number: 1001

Evaluation Highlight Rubrics (5 points):

Criterion	Points	Description
Correct Implementation of Account Class	1	- All required attributes and methods are correctly implemented. - Proper encapsulation and method functionality (deposit, withdraw, display, update).
Functionality of User Interface Operations	1	- All menu options work as expected (create, deposit, withdraw, view, update). - Handles multiple accounts using arrays.
Use of Java Control Structures	0 .5	- Effective use of loops and decision-making constructs for menu navigation and input handling.

Array and Data Management	0 .5	- Efficient use of arrays to store and manage multiple account objects.
String Handling and Validation	0 .5	- Proper manipulation and validation of strings (names, emails, phone numbers).
Exception Handling and Input Validation	0 .5	- Validates inputs (e.g., positive amounts, sufficient balance). - Graceful handling of invalid inputs.
Code Structure and Modularity	0 .5	- Code is well-organized into classes and methods. - Follows object-oriented principles.
Code Documentation and Readability	0 .5	- Includes meaningful comments and follows naming conventions. - Code is easy to read and maintain.

Code Snippets -

account.class

```java
public class Account {
    private int accountNumber;
    private String accountHolderName;
    private double balance;
    private String email;
    private String phoneNumber;

    public Account(int accountNumber, String accountHolderName, double balance, String email, String phoneNumber) {
        this.accountNumber = accountNumber;
        this.accountHolderName = accountHolderName;
        this.balance = balance;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }

    public int getAccountNumber() {
        return accountNumber;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Amount deposited: " + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            System.out.println("Amount withdrawn: " + amount);
        } else {
            System.out.println("Insufficient balance or invalid amount.");
        }
    }

    public void displayAccountDetails() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Name: " + accountHolderName);
        System.out.println("Balance: " + balance);
        System.out.println("Email: " + email);
        System.out.println("Phone: " + phoneNumber);
    }

    public void updateContactDetails(String email, String phoneNumber) {
        this.email = email;
        this.phoneNumber = phoneNumber;
        System.out.println("Contact details updated.");
    }
}

```

UserInterface.java

```java
import java.util.Scanner;

public class UserInterface {
    private static Account[] accounts = new Account[100];
    private static int accountCount = 0;
    private static int nextAccountNumber = 1001;
    private static Scanner sc = new Scanner(System.in);

    private static Account findAccount(int accountNumber) {
        for (int i = 0; i < accountCount; i++) {
            if (accounts[i].getAccountNumber() == accountNumber) {
                return accounts[i];
            }
        }
        return null;
    }

    private static void createAccount() {
        System.out.print("Enter account holder name: ");
        String name = sc.nextLine();
        System.out.print("Enter initial deposit: ");
        double balance = sc.nextDouble();
        sc.nextLine();
        System.out.print("Enter email: ");
        String email = sc.nextLine();
        System.out.print("Enter phone number: ");
        String phone = sc.nextLine();

        Account newAcc = new Account(nextAccountNumber++, name, balance, email, phone);
        accounts[accountCount++] = newAcc;
        System.out.println("Account created successfully.");
        System.out.println("Your Account Number is: " + newAcc.getAccountNumber());
    }

    private static void performDeposit() {
        System.out.print("Enter account number: ");
        int accNum = sc.nextInt();
        System.out.print("Enter amount to deposit: ");
        double amt = sc.nextDouble();
        sc.nextLine();
        Account acc = findAccount(accNum);
        if (acc != null) acc.deposit(amt);
        else System.out.println("Account not found.");
    }

    private static void performWithdrawal() {
        System.out.print("Enter account number: ");
        int accNum = sc.nextInt();
        System.out.print("Enter amount to withdraw: ");
        double amt = sc.nextDouble();
        sc.nextLine();
        Account acc = findAccount(accNum);
        if (acc != null) acc.withdraw(amt);
        else System.out.println("Account not found.");
    }
```

UserInterface.java

```java

    private static void showAccountDetails() {
        System.out.print("Enter account number: ");
        int accNum = sc.nextInt();
        sc.nextLine();
        Account acc = findAccount(accNum);
        if (acc != null) acc.displayAccountDetails();
        else System.out.println("Account not found.");
    }

    private static void updateContact() {
        System.out.print("Enter account number: ");
        int accNum = sc.nextInt();
        sc.nextLine();
        Account acc = findAccount(accNum);
        if (acc != null) {
            System.out.print("Enter new email: ");
            String email = sc.nextLine();
            System.out.print("Enter new phone: ");
            String phone = sc.nextLine();
            acc.updateContactDetails(email, phone);
        } else {
            System.out.println("Account not found.");
        }
    }

    private static void mainMenu() {
        while (true) {
            System.out.println("\n--- Banking Application ---");
            System.out.println("1. Create a new account");
            System.out.println("2. Deposit money");
            System.out.println("3. Withdraw money");
            System.out.println("4. View account details");
            System.out.println("5. Update contact details");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1 -> createAccount();
                case 2 -> performDeposit();
                case 3 -> performWithdrawal();
                case 4 -> showAccountDetails();
                case 5 -> updateContact();
                case 6 -> {
                    System.out.println("Thank you for using the Banking Application.");
                    return;
                }
                default -> System.out.println("Invalid choice.");
            }
        }
    }

    public static void main(String[] args) {
        mainMenu();
    }
}
```

Explanation :-

Explanation of the Assignment

The given assignment requires the design and implementation of a **Banking Application in Java**. The application must support basic account management operations such as creating new accounts, depositing money, withdrawing money, viewing account details, and updating contact information.

1. Objectives

- To apply the fundamentals of Java programming (data types, operators, control structures, type casting).
- To implement object-oriented programming concepts such as classes, objects, encapsulation, and methods.
- To utilize arrays for managing multiple account objects.
- To perform input/output operations using the Scanner class.
- To manipulate and validate string data such as account holder names, emails, and phone numbers.

2. Class Design

(a) Account Class

- Attributes: `accountNumber`, `accountHolderName`, `balance`, `email`, `phoneNumber`.
- Methods:
  - `deposit(double amount)` → Adds money to the account after validating the input.
  - `withdraw(double amount)` → Deducts money from the account ensuring sufficient balance.
  - `displayAccountDetails()` → Displays all account details.
  - `updateContactDetails(String email, String phoneNumber)` → Updates the account holder's contact information.

(b) UserInterface Class

- Attributes:
  - An array of `Account` objects.
  - A `Scanner` object for user input.
- Methods:
  - `createAccount()` → Creates and stores a new account.
  - `performDeposit()` → Handles deposit operations.
  - `performWithdrawal()` → Handles withdrawal operations.
  - `showAccountDetails()` → Displays details of a specific account.
  - `updateContact()` → Updates account contact details.
  - `mainMenu()` → Displays the menu and manages user choices using loops and control structures.

3. Implementation Highlights

- **Arrays** are used to manage multiple accounts.
- **Control structures** (`if-else`, `switch`, `while`) manage the flow of the program and user navigation.
- **Input validation** ensures deposits are positive, withdrawals do not exceed balance, and account numbers are valid.
- **String handling** is applied to store and update account holder names, emails, and phone numbers.

Sample Interaction : -