Experiment 1.3

Student Name: Shubham Chauhan **UID: 22BCS15879**
Section: 629"A" **Branch: BE-CSE**
Semester: 6th **DOP: 22/01/2025**
Subject: Java **Subject Code: 22CSH-359**

Aim: Create an application to calculate interest for FDs, RDs based on certain conditions using inheritance.

Objective: To calculate interest for Fixed Deposits (FDs) and Recurring Deposits (RDs) using inheritance is to demonstrate how object-oriented programming (OOP) concepts such as inheritance, polymorphism, and encapsulation can be applied to model real-world scenarios in a structured and reusable manner.

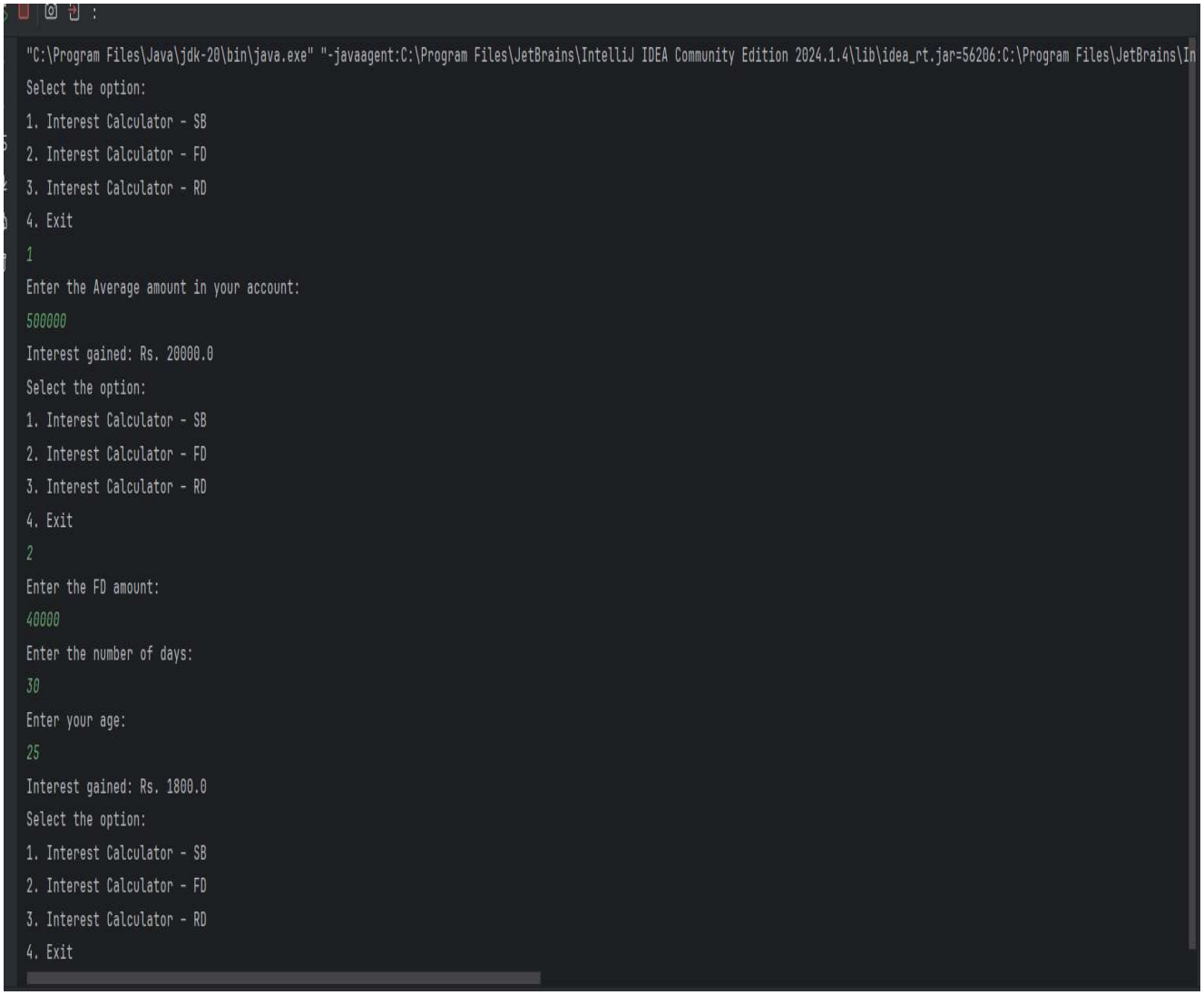
Code:

```
import java.util.Scanner;

abstract class Account {
   protected double interestRate;
   protected double amount;
   public abstract double calculateInterest();
}
class FDAccount extends Account {
   private int noOfDays;
   private int ageOfACHolder;

   public FDAccount(double amount, int noOfDays, int ageOfACHolder) throws InvalidInputException {
      if (amount <= 0 || noOfDays <= 0 || ageOfACHolder <= 0) {
         throw new InvalidInputException("Invalid input. Please enter positive values.");
      }
      this.amount = amount;
      this.noOfDays = noOfDays;
      this.ageOfACHolder = ageOfACHolder;
   }
   @Override
   public double calculateInterest() {
      if (amount < 10000000) {
         if (noOfDays <= 7) interestRate = 0;
         else if (noOfDays <= 45) interestRate = ageOfACHolder >= 60 ? 0.055 : 0.045;
         else if (noOfDays <= 90) interestRate = ageOfACHolder >= 60 ? 0.08 : 0.07;
         else if (noOfDays <= 180) interestRate = ageOfACHolder >= 60 ? 0.085 : 0.075;
         else interestRate = ageOfACHolder >= 60 ? 0.09 : 0.08;
      } else {
         interestRate = 0.1; // Assuming flat rate for amounts >= 1 crore
      }
      return amount * interestRate;
   }
```

```
}
class SBAccount extends Account {
    public SBAccount(double amount) throws InvalidInputException {
        if (amount <= 0) {
            throw new InvalidInputException("Invalid input. Please enter a positive value for amount.");
        }
        this.amount = amount;
        this.interestRate = 0.04; // Assuming a flat rate for SB accounts
    }
    @Override
    public double calculateInterest() {
        return amount * interestRate;
    }
}

class RDAccount extends Account {
    private int noOfMonths;
    private double monthlyAmount;

    public RDAccount(double monthlyAmount, int noOfMonths) throws InvalidInputException {
        if (monthlyAmount <= 0 || noOfMonths <= 0) {
            throw new InvalidInputException("Invalid input. Please enter positive values.");
        }
        this.monthlyAmount = monthlyAmount;
        this.noOfMonths = noOfMonths;
    }

    @Override
    public double calculateInterest() {
        interestRate = 0.06; // Assuming flat interest rate for RD accounts
        double totalAmount = monthlyAmount * noOfMonths;
        return totalAmount * interestRate;
    }
}

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

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

        do {
            System.out.println("Select the option:");
            System.out.println("1. Interest Calculator – SB");
            System.out.println("2. Interest Calculator – FD");
            System.out.println("3. Interest Calculator – RD");
            System.out.println("4. Exit");
            choice = scanner.nextInt();
```

```
            switch (choice) {
                case 1: // SB Account
                    try {
                        System.out.println("Enter the Average amount in your account:");
                        double amount = scanner.nextDouble();
                        SBAccount sbAccount = new SBAccount(amount);
                        System.out.println("Interest gained: Rs. " + sbAccount.calculateInterest());
                    } catch (InvalidInputException e) {
                        System.out.println(e.getMessage());
                    }
                    break;

                case 2: // FD Account
                    try {
                        System.out.println("Enter the FD amount:");
                        double amount = scanner.nextDouble();
                        System.out.println("Enter the number of days:");
                        int noOfDays = scanner.nextInt();
                        System.out.println("Enter your age:");
                        int age = scanner.nextInt();
                        FDAccount fdAccount = new FDAccount(amount, noOfDays, age);
                        System.out.println("Interest gained: Rs. " + fdAccount.calculateInterest());
                    } catch (InvalidInputException e) {
                        System.out.println(e.getMessage());
                    }
                    break;

                case 3: // RD Account
                    try {
                        System.out.println("Enter the monthly deposit amount:");
                        double monthlyAmount = scanner.nextDouble();
                        System.out.println("Enter the number of months:");
                        int noOfMonths = scanner.nextInt();
                        RDAccount rdAccount = new RDAccount(monthlyAmount, noOfMonths);
                        System.out.println("Interest gained: Rs. " + rdAccount.calculateInterest());
                    } catch (InvalidInputException e) {
                        System.out.println(e.getMessage());
                    }
                    break;

                case 4: // Exit
                    System.out.println("Exiting...");
                    break;

                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

Output:



Learning Outcomes:

- **Understanding Object-Oriented Programming (OOP):**
  Gain a deeper understanding of OOP principles such as inheritance, abstraction, and polymorphism through the implementation of the Account abstract class and its derived classes (SBAccount, FDAccount, RDAccount).
  Learn how method overriding works to provide specific implementations for abstract methods in subclasses.
- **Handling Dynamic Business Logic:**

  Understand how to implement conditional logic to calculate interest based on dynamic parameters such as account type, amount, age, and duration.
- **User-Defined Exceptions:**
  Learn how to create and use custom exceptions (InvalidInputException) to validate user inputs and handle errors gracefully.