Experiment - 3

Student Name: Paras **UID: 22BCS13422**
Branch: CSE **Section: IOT-636/B**
Semester: 6th **DOP:23/01/2025**
Subject: Java **Subject Code: 22CSH-359**

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

Objective: Calculate interest based on the type of the account and the status of the account holder. The rates of interest changes according to the amount (greater than or less than 1 crore), age of account holder (General or Senior citizen) and number of days if the type of account is FD or RD.

1. **Define the Abstract Class Account:** Create an abstract class Account with attributes interestRate and amount. Define an abstract method calculateInterest().
2. **Implement SBAccount Class:** Create a class SBAccount that extends Account. Add accountType to check if it's "Normal" or "NRI". Set the interest rate based on the account type. Implement calculateInterest() to calculate interest using the formula: Interest=amount×interestRate100\text{Interest} = \frac{\text{amount} \times \text{interestRate}}{100}Interest=100amount×interestRate
3. **Implement FDAccount Class:** Create a class FDAccount that extends Account. Add attributes for noOfDays (duration) and ageOfACHolder (age of the account holder). Set the interest rate based on:
    a. The amount (< or > 1 crore).
    b. The duration of the deposit (noOfDays).
    c. The age of the account holder.
4. **Implement RDAccount Class:** Create a class RDAccount that extends Account. Add attributes noOfMonths (duration), monthlyAmount (deposit amount), and ageOfACHolder.
5. **Create InvalidInputException:** Define a custom exception class to handle invalid inputs (e.g., negative numbers).
6. **Create the InterestCalculator Main Program:**
7. Display a menu with options:
    a. Calculate interest for SB account.
    b. Calculate interest for FD account.
    c. Calculate interest for RD account.
    d. Exit.
8. **Handle User Input for Each Option: Option 1 (SB):** Get accountType and amount, calculate interest using SBAccount. **Option 2 (FD):** Get amount, noOfDays, and ageOfACHolder, calculate interest using FDAccount. **Option 3 (RD):** Get noOfMonths, monthlyAmount, and ageOfACHolder, calculate interest using RDAccount. **Option 4 (Exit):** End the program.
9. **Validate Inputs:** Use try-catch blocks to handle invalid inputs with InvalidInputException.
10. **Test the Program:**Run the program and test for various cases:
    a. Valid inputs (e.g., positive values).
    b. Invalid inputs (e.g., negative numbers).
    c. Correct interest calculations for all account types.

Code:

```
import java.util.Scanner;
abstract class Account {
    double interestRate;
    double amount;

    abstract double calculateInterest();
}

class InvalidInputException extends Exception {
    public InvalidInputException(String message) {
        super(message);
    }
}
class SBAccount extends Account {
    private String accountType;
    public SBAccount(String accountType, double amount) {
        this.accountType = accountType;
        this.amount = amount;
        if (accountType.equalsIgnoreCase("Normal")) {
            this.interestRate = 4.0;
        } else if (accountType.equalsIgnoreCase("NRI")) {
            this.interestRate = 6.0;
        }
    }
```

```
    @Override
    double calculateInterest() {
        return (amount * interestRate) / 100;
    }
}
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("Negative values are not allowed.");
        }
        this.amount = amount;
        this.noOfDays = noOfDays;
        this.ageOfACHolder = ageOfACHolder;
    }
    @Override
    double calculateInterest() {
        if (amount < 10000000) {
            if (noOfDays >= 7 && noOfDays <= 14) {
                interestRate = ageOfACHolder >= 60 ? 5.0 : 4.5;
            } else if (noOfDays >= 15 && noOfDays <= 29) {
                interestRate = ageOfACHolder >= 60 ? 5.25 : 4.75;
            } else if (noOfDays >= 30 && noOfDays <= 45) {
                interestRate = ageOfACHolder >= 60 ? 6.0 : 5.5;
```

```
            } else if (noOfDays >= 45 && noOfDays <= 60) {
                interestRate = ageOfACHolder >= 60 ? 7.5 : 7.0;
            } else if (noOfDays >= 61 && noOfDays <= 184) {
                interestRate = ageOfACHolder >= 60 ? 8.0 : 7.5;
            } else if (noOfDays >= 185 && noOfDays <= 365) {
                interestRate = ageOfACHolder >= 60 ? 8.5 : 8.0;
            }
        } else {
            if (noOfDays >= 7 && noOfDays <= 14) {
                interestRate = 6.5;
            } else if (noOfDays >= 15 && noOfDays <= 29) {
                interestRate = 6.75;
            } else if (noOfDays >= 30 && noOfDays <= 60) {
                interestRate = 6.75;
            } else if (noOfDays >= 61 && noOfDays <= 184) {
                interestRate = 8.5;
            } else if (noOfDays >= 185 && noOfDays <= 365) {
                interestRate = 10.0;
            }
        }
        return (amount * interestRate) / 100;
    }
}
class RDAccount extends Account {
    private int noOfMonths;
    private double monthlyAmount;
```

```
    private int ageOfACHolder;

    public RDAccount(int noOfMonths, double monthlyAmount, int ageOfACHolder) throws
InvalidInputException {

        if (noOfMonths < 0 || monthlyAmount < 0 || ageOfACHolder < 0) {

            throw new InvalidInputException("Negative values are not allowed.");

        }

        this.noOfMonths = noOfMonths;

        this.monthlyAmount = monthlyAmount;

        this.ageOfACHolder = ageOfACHolder;

    }

    @Override

    double calculateInterest() {

        if (noOfMonths == 6) {

            interestRate = ageOfACHolder >= 60 ? 8.0 : 7.5;

        } else if (noOfMonths == 9) {

            interestRate = ageOfACHolder >= 60 ? 8.25 : 7.75;

        } else if (noOfMonths == 12) {

            interestRate = ageOfACHolder >= 60 ? 8.5 : 8.0;

        } else if (noOfMonths == 15) {

            interestRate = ageOfACHolder >= 60 ? 8.75 : 8.25;

        } else if (noOfMonths == 18) {

            interestRate = ageOfACHolder >= 60 ? 9.0 : 8.5;

        } else if (noOfMonths == 21) {

            interestRate = ageOfACHolder >= 60 ? 9.25 : 8.75;

        }

        double totalAmount = noOfMonths * monthlyAmount;

        return (totalAmount * interestRate) / 100;
```

```
    }
}
public class InterestCalculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            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");
            int choice = sc.nextInt();
            try {
                switch (choice) {
                    case 1:
                        System.out.println("Enter the type of SB Account (Normal/NRI):");
                        String accountType = sc.next();
                        System.out.println("Enter the Average amount in your account:");
                        double amount = sc.nextDouble();
                        if (amount < 0) throw new InvalidInputException("Amount cannot be negative.");
                        SBAccount sbAccount = new SBAccount(accountType, amount);
                        System.out.println("Interest gained: Rs. " + sbAccount.calculateInterest());
                        break;
                    case 2:
                        System.out.println("Enter the FD amount:");
                        double fdAmount = sc.nextDouble();
```

```
                System.out.println("Enter the number of days:");
                int noOfDays = sc.nextInt();
                System.out.println("Enter your age:");
                int age = sc.nextInt();
                FDAccount fdAccount = new FDAccount(fdAmount, noOfDays, age);
                System.out.println("Interest gained is: Rs. " + fdAccount.calculateInterest());
                break;
            case 3:
                System.out.println("Enter the RD monthly amount:");
                double monthlyAmount = sc.nextDouble();
                System.out.println("Enter the number of months:");
                int noOfMonths = sc.nextInt();
                System.out.println("Enter your age:");
                int rdAge = sc.nextInt();
                RDAccount rdAccount = new RDAccount(noOfMonths, monthlyAmount, rdAge);
                System.out.println("Interest gained is: Rs. " + rdAccount.calculateInterest());
                break;
            case 4:
                System.out.println("Exiting...");
                sc.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } catch (InvalidInputException e) {
        System.out.println(e.getMessage());
```

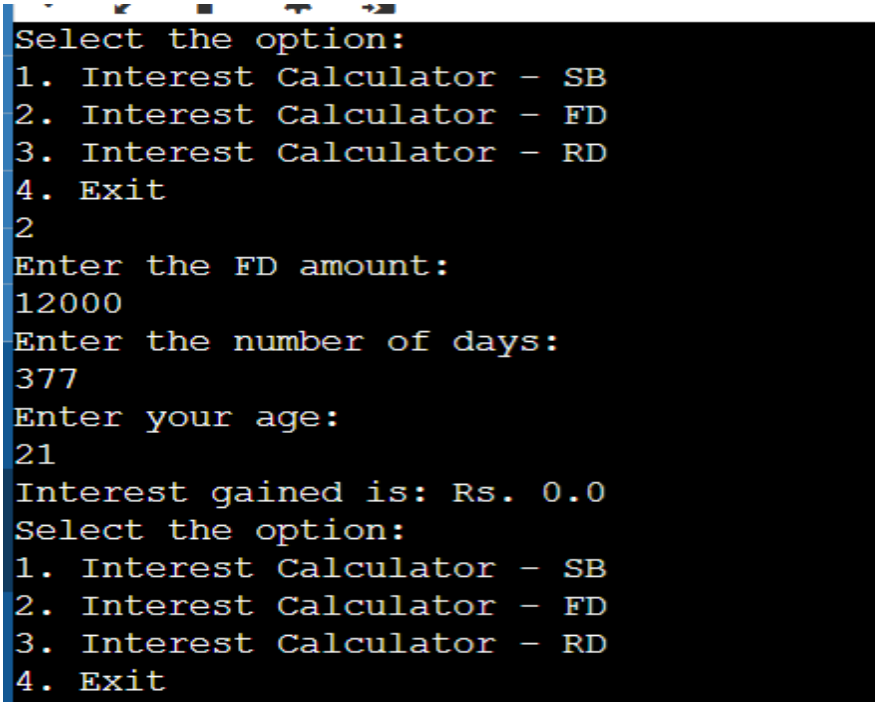
```
            }

        }

    }

}
```

Output:



Q2. .Write a Java program to calculate the square root of a number entered by the user. Use try-catch to handle invalid inputs (e.g., negative numbers or non-numeric values).

Code:

```
import java.util.Scanner;

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

        System.out.print("Enter a number to calculate its square root: ");
```

```
        try {
            String input = scanner.nextLine();
            double number = Double.parseDouble(input);

            if (number < 0) {
                throw new IllegalArgumentException("Cannot calculate the square root of a negative number.");
            }

            double squareRoot = Math.sqrt(number);
            System.out.printf("The square root of %.2f is %.2f%n", number, squareRoot);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input: Please enter a numeric value.");
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

OUTPUT:

```
Enter a number to calculate its square root: 9
The square root of 9.00 is 3.00


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

1. Object-Oriented Programming (OOP): Understand abstraction with the Account abstract class.
2. Implement inheritance using subclasses (SBAccount, FDAccount, RDAccount).
3. Learn polymorphism through method overriding for calculateInterest.
4. Practice encapsulation with private fields and public methods.
5. Exception Handling:Create and use custom exceptions (InvalidInputException).
6. Handle errors gracefully with try-catch blocks.
7. Apply while loops for continuous user interaction.