Experiment-3

Student Name:Priyal sharma **UID:** 22BCS10727
Branch: CSE **Section/Group:** IOT-635(A)
Semester: 6th **Date of Performance:**04/02/25
Subject Name: Project Based Learning in Java **Subject Code:** 22CSH-359

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

2. **Objective:** To calculate interest for Savings Bank (SB), Fixed Deposit (FD), and Recurring Deposit (RD) accounts using inheritance in Java, based on user input and specific conditions.

3. **Algorithm:**

   1. **Create Base Class (Account):**
      - Define common attributes (interestRate, amount) and an abstract method calculateInterest().
   2. **Implement Subclasses:**
      - SBAccount: Calculates interest based on account type (Normal/NRI).
      - FDAccount: Calculates interest based on tenure, deposit amount, and age.
      - RDAccount: Calculates interest based on tenure, monthly deposit, and age.
   3. **Develop Main Class (InterestCalculator):**
      - Display a menu for account types.
      - Accept inputs for the selected account type and calculate interest.
      - Handle invalid inputs using exceptions.
      - Loop until the user chooses to exit.
   4. **Display Results:**
      - Show calculated interest for the selected account.

4. **Implementation/Code:**

```
import java.util.*;


abstract class Account {

    double interestRate;

    double amount;
```

```
    abstract double calculateInterest();
}
class FDAccount extends Account {
    int Days;
    int age;
    FDAccount(double amount, int d, int a) {
        this.amount = amount;
        this.Days = d;
        this.age = a;
    }
    double calculateInterest() {
        if (amount < 0 || Days < 0 || age < 0) {
            throw new IllegalArgumentException("Invalid value entered.");
        }
        if (amount < 10000000) {
            if (Days >= 7 &&Days <= 14) {
                interestRate = (age >= 60) ? 5.00 : 4.50;
            } else if (Days >= 15 && Days <= 29) {
                interestRate = (age >= 60) ? 5.25 : 4.75;
            } else if (Days >= 30 && Days <= 45) {
                interestRate = (age >= 60) ? 6.00 : 5.50;
            } else if (Days >= 45 && Days <= 60) {
                interestRate = (age >= 60) ? 7.50 : 7.00;
```

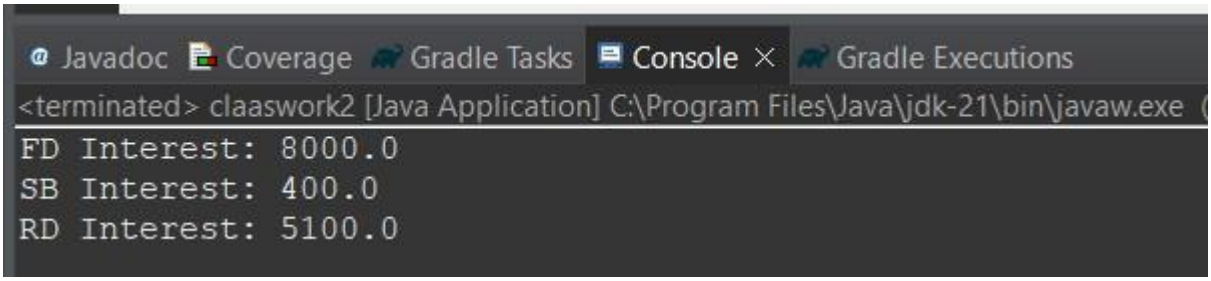
```
            } else if (Days >= 61 && Days <= 184) {
                interestRate = (age >= 60) ? 8.00 : 7.50;
            } else if (Days >= 185 && Days <= 365) {
                interestRate = (age >= 60) ? 8.50 : 8.00;
            }
        } else {
            if (Days >= 7 && Days <= 14) {
                interestRate = 6.50;
            } else if (Days >= 15 && Days <= 29) {
                interestRate = 6.75;
            } else if (Days >= 30 && Days <= 45) {
                interestRate = 6.75;
            } else if (Days >= 45 && Days <= 60) {
                interestRate = 8.00;
            } else if (Days >= 61 && Days <= 184) {
                interestRate = 8.50;
            } else if (Days >= 185 && Days <= 365) {
                interestRate = 10.00;
            }
        }
        return (amount * interestRate) / 100;
    }
}
```

```
class SBAccount extends Account {
    String accountType;

    SBAccount(double amount, String accountType) {
        this.amount = amount;
        this.accountType = accountType;
    }
    @Override
    double calculateInterest() {
        if (amount < 0) {
            throw new IllegalArgumentException("Invalid value entered.");
        }
        interestRate = accountType.equalsIgnoreCase("NRI") ? 6.00 : 4.00;
        return (amount * interestRate) / 100;
    }
}
class RDAccount extends Account {
    int noOfMonths;
    double monthlyAmount;
    int age;
    RDAccount(double monthlyAmount, int noOfMonths, int ageOfACHolder) {
        this.monthlyAmount = monthlyAmount;
        this.noOfMonths = noOfMonths;
        this.age = ageOfACHolder;
```

```
    }        double calculateInterest() {
        if (monthlyAmount < 0 || noOfMonths < 0 || age < 0) {
            throw new IllegalArgumentException("Invalid value entered.");
        }
        if (noOfMonths == 6) {
            interestRate = (age >= 60) ? 8.00 : 7.50;
        } else if (noOfMonths == 9) {
            interestRate = (age >= 60) ? 8.25 : 7.75;
        } else if (noOfMonths == 12) {
            interestRate = (age >= 60) ? 8.50 : 8.00;
        } else if (noOfMonths == 15) {
            interestRate = (age >= 60) ? 8.75 : 8.25;
        } else if (noOfMonths == 18) {
            interestRate = (age >= 60) ? 9.00 : 8.50;
        } else if (noOfMonths == 21) {
            interestRate = (age >= 60) ? 9.25 : 8.75;
        }
        return (monthlyAmount * noOfMonths * interestRate) / 100;
    }
}
public class claaswork2 {
public static void main(String args[]) {


        Account fdAccount = new FDAccount(100000, 91, 65);
```

```
        System.out.println("FD Interest: " + fdAccount.calculateInterest());


        Account sbAccount = new SBAccount(10000, "Normal");

        System.out.println("SB Interest: " + sbAccount.calculateInterest());


        Account rdAccount = new RDAccount(5000, 12, 70);

        System.out.println("RD Interest: " + rdAccount.calculateInterest());

    }

    }
```

5. Output:



6. Complexities

Time Complexity: O (1)
Space complexity: O (1)

7. Learning Outcome:

- **Inheritance**: Code reuse through subclasses.
- **Abstraction**: Enforce specific implementations in derived classes.
- **Exception Handling**: Validate inputs and handle errors.
- **Real-World Application**: Solve practical problems with object-oriented programming.