Experiment No. 4

Title: Create class SavingsAccount. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12this interest should be added to savingsBalance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value.

Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of Rs 2000.00 and Rs 3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.

Objectives: 1. To learn static variable and static method

Theory:

In java, static belongs to class. You can create static variables and static methods. You can call these directly by using class name, without creating instance.

Static Variables

Static variables belong to the class and not to the object. Static variables are not part of object state, means there is only one copy of the values will be served to all instances. If you define a field as static, then there is only one such field per class. In contrast, each object has its own copy of all instance fields.

Static variables are created when the program starts and destroyed when the program stops.

Static variables can be accessed by calling with the class name . *ClassName.VariableName.*

For example, let's suppose we want to assign a unique identification number to each employee. We add an instance field *id* and a static field *nextId* to the Employee class:

```
class Employee
{ ……
private int id;
private static int nextId = 1;
}
```

Now, every employee object has its own id field, but there is only one *nextId* field that is shared among all instances of the class. Let's put it another way. If there are one thousand objects of the Employee class, then there are one thousand instance fields id, one for each object. But there is a single static field *nextId*. Even if there are no employee objects, the static field *nextId* is present. It belongs to the class, not to any individual object. **static methods:**

Static methods are also similar to static variables; you can access them with reference to class name, without creating object. Inside static methods, you cannot access instance variables or instance methods. You can only access static variables or static methods. Because static methods don't operate on objects, you cannot access instance fields from a static method. But static methods can access the static fields in their class. Here is an example of such a static method:

```
public static int getNextId()
{
return nextId; // returns static field
}
```

To call this method, you supply the name of the class:

```
int n = Employee.getNextId();
```

Key concepts: class, object, static variable, static method.

Algorithm:

□□□Create class SavingsAccount having static variable annualInterestRate

□□Declare private instance variable savingsBalance.

□□□Define a method calculateMonthlyInterest to calculate the monthly interest .

□□Define a static method modifyInterestRate to set the annualInterestRate to a new value.

□□Define a class SavingsAccount.

□□Create two objects of class savingsAccount saver1 and saver2 by using parameterized constructor with initial balances of Rs 2000.00 and Rs 3000.00 respectively. □□Set annualInterestRate to 4%, then calculate the monthly interest and print the new Balances for both savers.

8) Set annualInterestRate to 5%, then calculate the monthly interest and print the new Balances for both savers.