**1)**

**Given:**

**public class TaxUtil {**

**double rate = 0.15;**

**public double calculateTax(double amount) {**

**return amount \* rate;**

**}**

**}**

**Would you consider the method calculateTax() a 'pure function'? Why or why not?**

**If you claim the method is NOT a pure function, please suggest a way to make it pure.**

* The method calculateTax() is **not a pure function** because it relies on the **instance variable** rate, which is external to the method.
* A pure function must only depend on its input parameters and have no side effects.
* Change it to accept rate as a parameter:

public double calculateTax(double amount, double rate) {

return amount \* rate;

}

**2)**

**What will be the output for following code?**

**class Super**

**{**

**static void show()**

**{**

**System.out.println("super class show method");**

**}**

**static class StaticMethods**

**{**

**void show()**

**{**

**System.out.println("sub class show method");**

**}**

**}**

**public static void main(String[]args)**

**{**

**Super.show();**

**new Super.StaticMethods().show();**

**}**

**}**

This program has two classes:

* One is the main class Super, which has a static method show() that prints "super class show method".
* Inside it, there's another class StaticMethods, which has its own show() method that prints "sub class show method".

In the main method:

* Super.show(); directly calls the method from the outer class and prints:  
  super class show method
* new Super.StaticMethods().show(); creates an object of the inner class and calls its method, printing:  
  sub class show method

So the output is:

* super class show method
* sub class show method

**3)**

**What will be the output for the following code?**

**class Super**

**{**

**int num=20;**

**public void display()**

**{**

**System.out.println("super class method");**

**}**

**}**

**public class ThisUse extends Super**

**{**

**int num;**

**public ThisUse(int num)**

**{**

**this.num=num;**

**}**

**public void display()**

**{**

**System.out.println("display method");**

**}**

**public void Show()**

**{**

**this.display();**

**display();**

**System.out.println(this.num);**

**System.out.println(num);**

**}**

**public static void main(String[]args)**

**{**

**ThisUse o=new ThisUse(10);**

**o.show();**

**}**

**}**

* The display() method in the ThisUse class overrides the one in Super, so both this.display() and display() call the child class method and print "display method".
* The num variable in ThisUse is set to 10 via the constructor. So both this.num and num print 10.
* The num from Super class is hidden and not accessed in this code.

**Output :**

display method

display method

10

10

**4) What is the singleton design pattern? Explain with a coding example.**

* The Singleton design pattern ensures that only one object of a class is created during the program’s lifetime.
* It provides a global access point to that instance.

**Code :**

class Singleton{

private static Singleton instance;

private Singleton() {

System.out.println("Singleton instance created");

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public void showMessage() {

System.out.println("Hello from Singleton");

}

}

public class Main {

public static void main(String[] args) {

Singleton s1 = Singleton.getInstance();

Singleton s2 = Singleton.getInstance();

s1.showMessage();

System.out.println(s1 == s2);

}

}

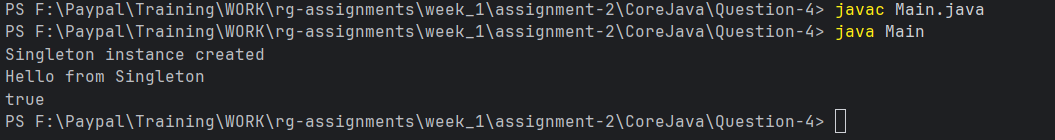
**Output :**

Singleton instance created

Hello from Singleton

True

* Singleton.getInstance() checks if an object exists.
* If not, it creates one; otherwise, it returns the existing one.
* s1 == s2 is true because both variables point to the same object.



**5) How do we make sure a class is encapsulated? Explain with a coding example.**

To make a class **encapsulated :**

* Declare variables as **private**.
* Provide **public getter and setter** methods to access and modify them.

**Code :**

class StudentData {

private String name;

private int age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

public class Encapsulated {

public static void main(String[] args) {

StudentData s = new StudentData();

s.setName("Ashish");

s.setAge(22);

System.out.println(s.getName());

System.out.println(s.getAge());

}

}

**Output :**

Ashish

22

**6)**

**Perform CRUD operation using ArrayList collection in an EmployeeCRUD class for the below Employee**

**class Employee{**

**private int id;**

**private String name;**

**private String department;**

**}**

**Code:**

import java.util.ArrayList;

class Employee {

private int id;

private String name;

private String department;

public Employee(int id, String name, String department) {

this.id = id;

this.name = name;

this.department = department;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public String getDepartment() {

return department;

}

public void setName(String name) {

this.name = name;

}

public void setDepartment(String department) {

this.department = department;

}

public void display() {

System.out.println("ID: " + id + ", Name: " + name + ", Department: " + department);

}

}

public class EmployeeCRUD {

private ArrayList<Employee> employees = new ArrayList<>();

// Create

public void addEmployee(Employee emp) {

employees.add(emp);

System.out.println("Employee added.");

}

// Read

public void displayEmployees() {

if (employees.isEmpty()) {

System.out.println("No employees to display.");

} else {

for (Employee emp : employees) {

emp.display();

}

}

}

// Update

public void updateEmployee(int id, String newName, String newDept) {

for (Employee emp : employees) {

if (emp.getId() == id) {

emp.setName(newName);

emp.setDepartment(newDept);

System.out.println("Employee updated.");

return;

}

}

System.out.println("Employee not found.");

}

// Delete

public void deleteEmployee(int id) {

for (Employee emp : employees) {

if (emp.getId() == id) {

employees.remove(emp);

System.out.println("Employee deleted.");

return;

}

}

System.out.println("Employee not found.");

}

// Main method for testing

public static void main(String[] args) {

EmployeeCRUD crud = new EmployeeCRUD();

// Sample operations

crud.addEmployee(new Employee(1, "Ashish", "IT"));

crud.addEmployee(new Employee(2, "Neha", "HR"));

System.out.println("\nAll Employees:");

crud.displayEmployees();

System.out.println("\nUpdating Employee with ID 1:");

crud.updateEmployee(1, "Ashish Odedra", "DevOps");

System.out.println("\nAll Employees after update:");

crud.displayEmployees();

System.out.println("\nDeleting Employee with ID 2:");

crud.deleteEmployee(2);

System.out.println("\nAll Employees after deletion:");

crud.displayEmployees();

}

}

**Output :**

Employee added.

Employee added.

All Employees:

ID: 1, Name: Ashish, Department: IT

ID: 2, Name: Neha, Department: HR

Updating Employee with ID 1:

Employee updated.

All Employees after update:

ID: 1, Name: Ashish Odedra, Department: DevOps

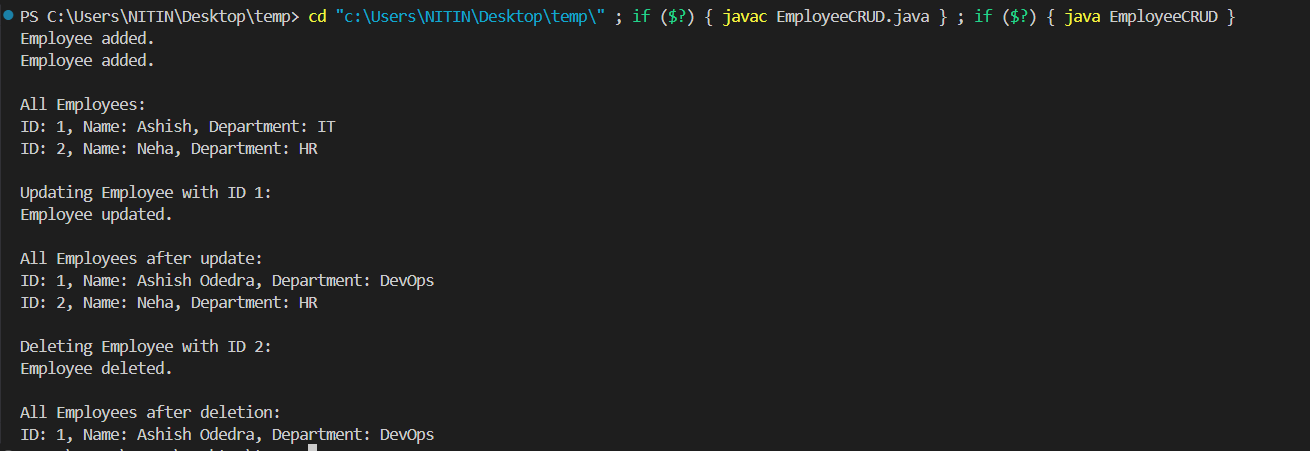
ID: 2, Name: Neha, Department: HR

Deleting Employee with ID 2:

Employee deleted.

All Employees after deletion:

ID: 1, Name: Ashish Odedra, Department: DevOps



**7) Perform CRUD operation using JDBC in an EmployeeJDBC class for the below Employee**

**class Employee{**

**private int id;**

**private String name;**

**private String department;**

**}**