**Please make sure to save/push all your code in the branch feature-java created in the previous week assignment as part of your github repo rg-assignments**

**Please share your output screenshots in the assignment document along with the github link for each question. Provide an explanation wherever possible as part of your response :-)**

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.

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();

}

}

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();

}

}

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

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

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;

}

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;

}

### **1.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q1.java>

### **Not a pure function**, because it uses an instance variable rate (i.e., depends on external state).

**To make it pure:**

public class TaxUtil {

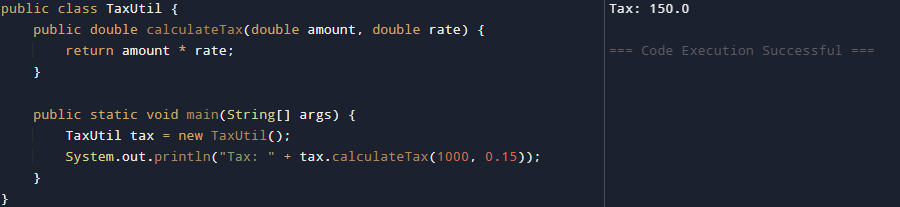
public double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

This is a pure function — it always returns the same result for the same input and has no side effects.



—----------------------------------------------------------------------------------------------------------------------------

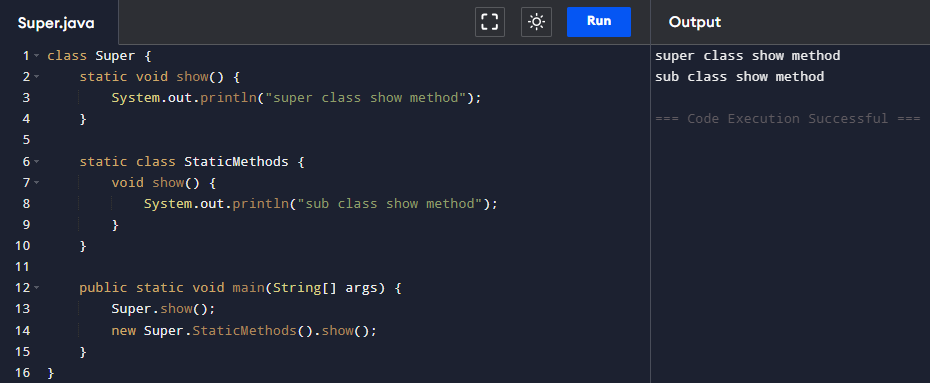
### **2.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q2.java>

**Output:**

super class show method

sub class show method

The outer class Super's static method and inner static class's instance method both get called correctly.



—----------------------------------------------------------------------------------------------------------------------------

### **3.**

[https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q3.java](https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q2.java)

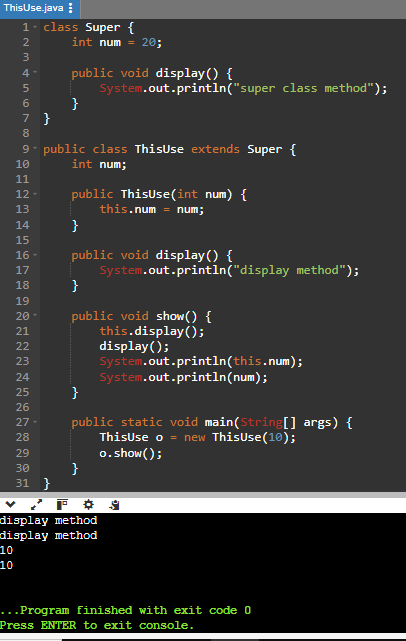
**Output:**

display method

display method

10

10



this.display() and display() call the overridden method in ThisUse. num and this.num refer to the same instance variable (10).

—----------------------------------------------------------------------------------------------------------------------------

### **4.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q4.java>

Ensures only one instance of a class exists and provides a global point of access.

**Example:**

public class Singleton {

private static Singleton instance;

private Singleton() {

// private constructor

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public void showMessage() {

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

}

}

public class Main {

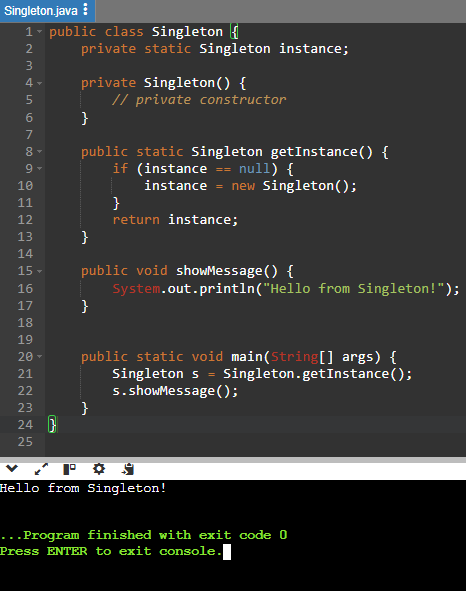
public static void main(String[] args) {

Singleton s = Singleton.getInstance();

s.showMessage();

}

}



—----------------------------------------------------------------------------------------------------------------------------

### **5.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q5.java>

Encapsulation of a class can be ensured by using private fields and public getters & setters.

**Example:**

public class Student {

private String name;

private int age;

public String getName() { return name; }

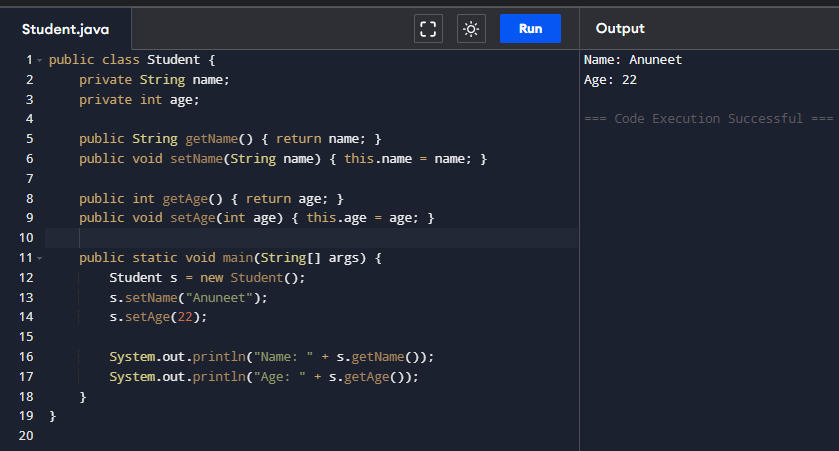
public void setName(String name) { this.name = name; }

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

}

This ensures data is controlled and secure.



—----------------------------------------------------------------------------------------------------------------------------

### **6.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q6.java>

**Employee class:**

public class Employee {

private int id;

private String name;

private String department;

// Constructor

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

this.id = id;

this.name = name;

this.department = department;

}

// Getters

public int getId() { return id; }

public String getName() { return name; }

public String getDepartment() { return department; }

// Setters

public void setName(String name) { this.name = name; }

public void setDepartment(String department) { this.department = department; }

}

**EmployeeCRUD class:**

import java.util.\*;

public class EmployeeCRUD {

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

public void addEmployee(Employee emp) {

employees.add(emp);

}

public void updateEmployee(int id, String name, String department) {

for (Employee e : employees) {

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

e.setName(name);

e.setDepartment(department);

}

}

}

public void deleteEmployee(int id) {

employees.removeIf(e -> e.getId() == id);

}

public void listEmployees() {

for (Employee e : employees) {

System.out.println(e.getId() + " " + e.getName() + " " + e.getDepartment());

}

}

### **Main.java (Testing Class)**

public class Main {

public static void main(String[] args) {

EmployeeCRUD crud = new EmployeeCRUD();

// Adding employees

crud.addEmployee(new Employee(1, "Alice", "HR"));

crud.addEmployee(new Employee(2, "Bob", "IT"));

crud.addEmployee(new Employee(3, "Charlie", "Finance"));

System.out.println("All Employees:");

crud.listEmployees();

// Updating an employee

crud.updateEmployee(2, "Bob Marley", "Security");

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

crud.listEmployees();

// Deleting an employee

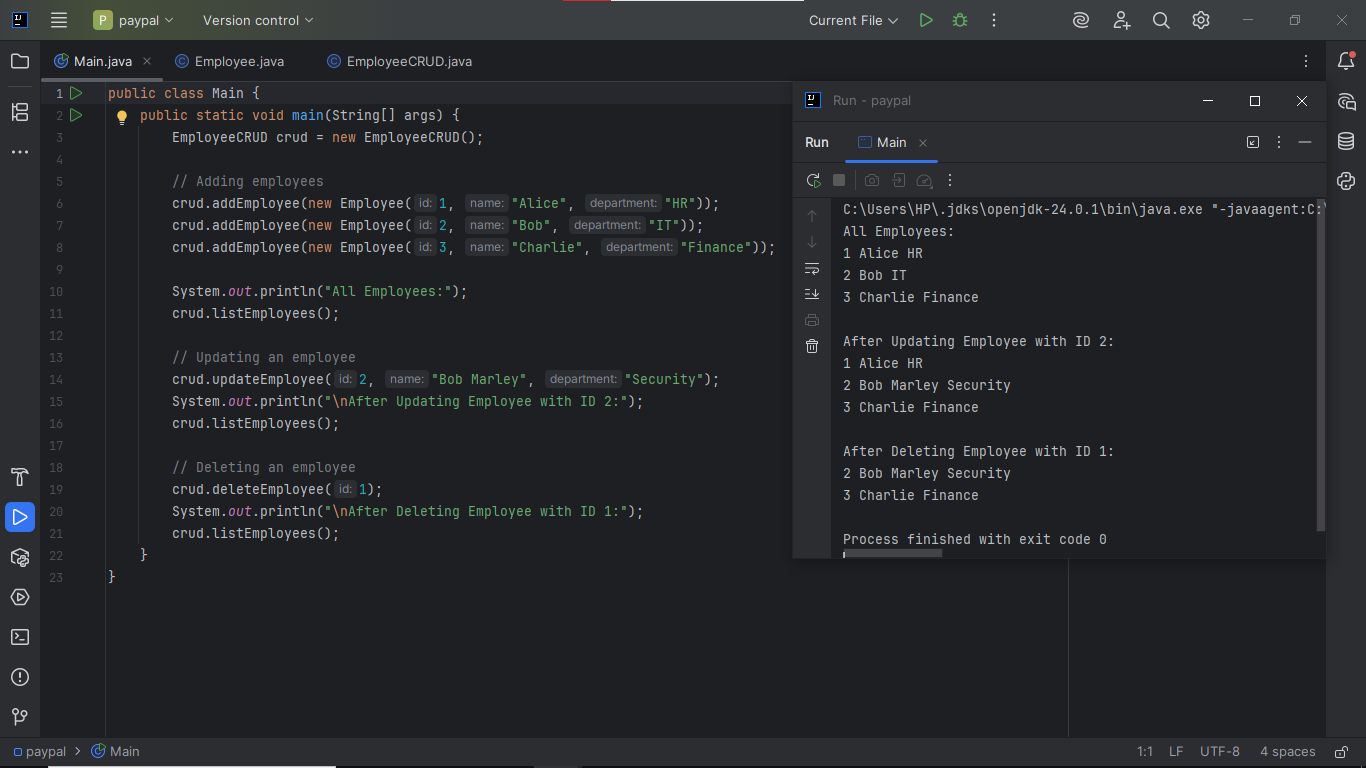
crud.deleteEmployee(1);

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

crud.listEmployees();

}

}

}

—----------------------------------------------------------------------------------------------------------------------------

### **7.** <https://github.com/Anuneet2003/rg-assignments/blob/feature-java/Week1/Core%20Java/Q7.java>

**EmployeeJDBC class:**

import java.sql.\*;

public class EmployeeJDBC {

private Connection connect() throws SQLException {

return DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "user", "password");

}

public void addEmployee(Employee emp) throws SQLException {

String sql = "INSERT INTO employees (id, name, department) VALUES (?, ?, ?)";

try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, emp.getId());

ps.setString(2, emp.getName());

ps.setString(3, emp.getDepartment());

ps.executeUpdate();

}

}

public void updateEmployee(Employee emp) throws SQLException {

String sql = "UPDATE employees SET name=?, department=? WHERE id=?";

try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setString(1, emp.getName());

ps.setString(2, emp.getDepartment());

ps.setInt(3, emp.getId());

ps.executeUpdate();

}

}

public void deleteEmployee(int id) throws SQLException {

String sql = "DELETE FROM employees WHERE id=?";

try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, id);

ps.executeUpdate();

}

}

public void listEmployees() throws SQLException {

String sql = "SELECT \* FROM employees";

try (Connection conn = connect(); Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

System.out.println(rs.getInt("id") + " " + rs.getString("name") + " " + rs.getString("department"));

}

}

}

}

### **Test Code: Main.java**

public class Main {

public static void main(String[] args) {

try {

EmployeeJDBC db = new EmployeeJDBC();

// Add employees

db.addEmployee(new Employee(1, "Alice", "HR"));

db.addEmployee(new Employee(2, "Bob", "IT"));

// List all employees

System.out.println("All Employees:");

db.listEmployees();

// Update an employee

db.updateEmployee(new Employee(2, "Bob Marley", "Security"));

System.out.println("\nAfter Update:");

db.listEmployees();

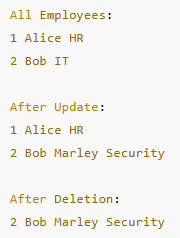
// Delete an employee

db.deleteEmployee(1);

System.out.println("\nAfter Deletion:");

db.listEmployees();

} catch (Exception e) {

e.printStackTrace();

}

}

}