**CORE JAVA ASSIGNMENT**

**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.

**Ans:** No, calculateTax() is not a pure function because it uses the instance variable rate which may change. A pure function should return the same output for the same inputs without depending on external or instance state.

**To make it pure:**

public class TaxUtil {

public double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

Now the method depends only on its parameters and has no side effects.

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)

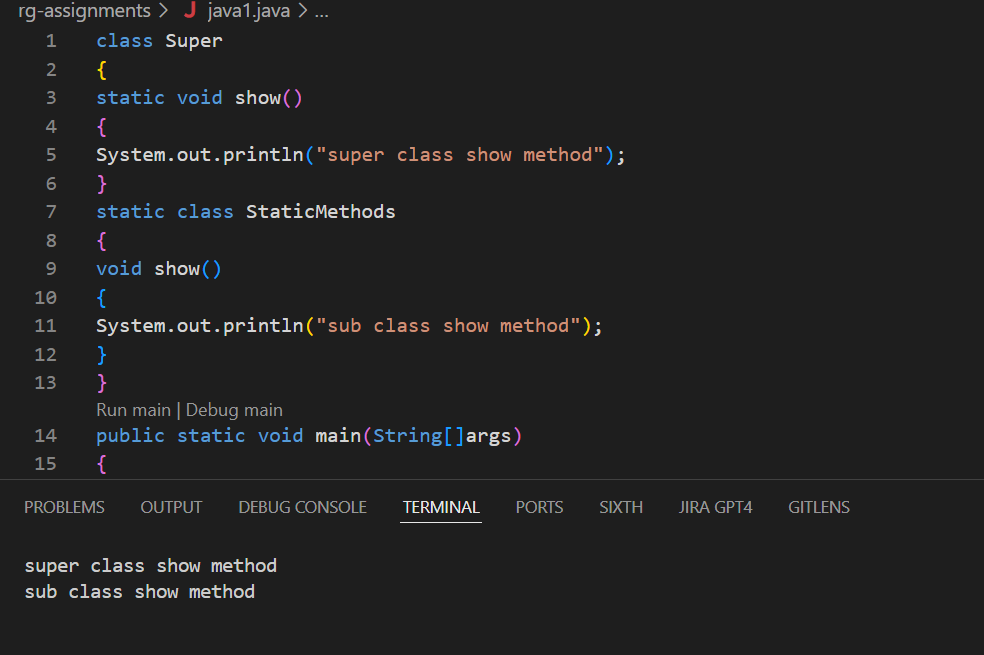
{

Super.show();

new Super.StaticMethods().show();

}

}



The first call invokes the outer static method, and the second call invokes the inner static class’s instance method. The program defines a static method in the outer class Super and a non-static method inside a static nested class StaticMethods. The outer class method is called directly using Super.show(), printing "super class show method". Then, an object of the nested class is created, and its show() method is called, printing "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();

}}

**Ans:** Java is case-sensitive, so Show() ≠ show().Then only the code will run

**OUTPUT:** display method

display method

10

10



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

**Ans:** The Singleton Design Pattern ensures that only one instance of a class is created and provides a global access point to that instance. It is commonly used for managing shared resources like database connections or configuration settings. In Java, this is done by making the constructor private, using a static variable to hold the instance, and providing a static method to return it.

class Singleton {

// Static variable to hold single instance

private static Singleton instance;

// Private constructor to prevent instantiation

private Singleton() {

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

// Public method to provide access to the instance

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton(); // lazy initialization }

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();

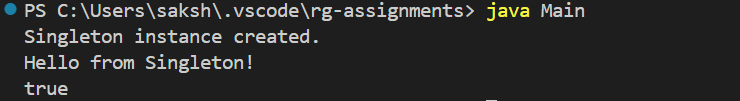
System.out.println(s1 == s2);

}}

**OUTPUT:** Singleton instance created.

Hello from Singleton!

True



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

**Ans:** Encapsulation is a key principle of Object-Oriented Programming (OOP) that hides the internal details of a class and only exposes what is necessary through public methods.

To ensure a class is encapsulated, we:

1. Declare all class variables as private.
2. Provide public getter and setter methods to access and update those variables.

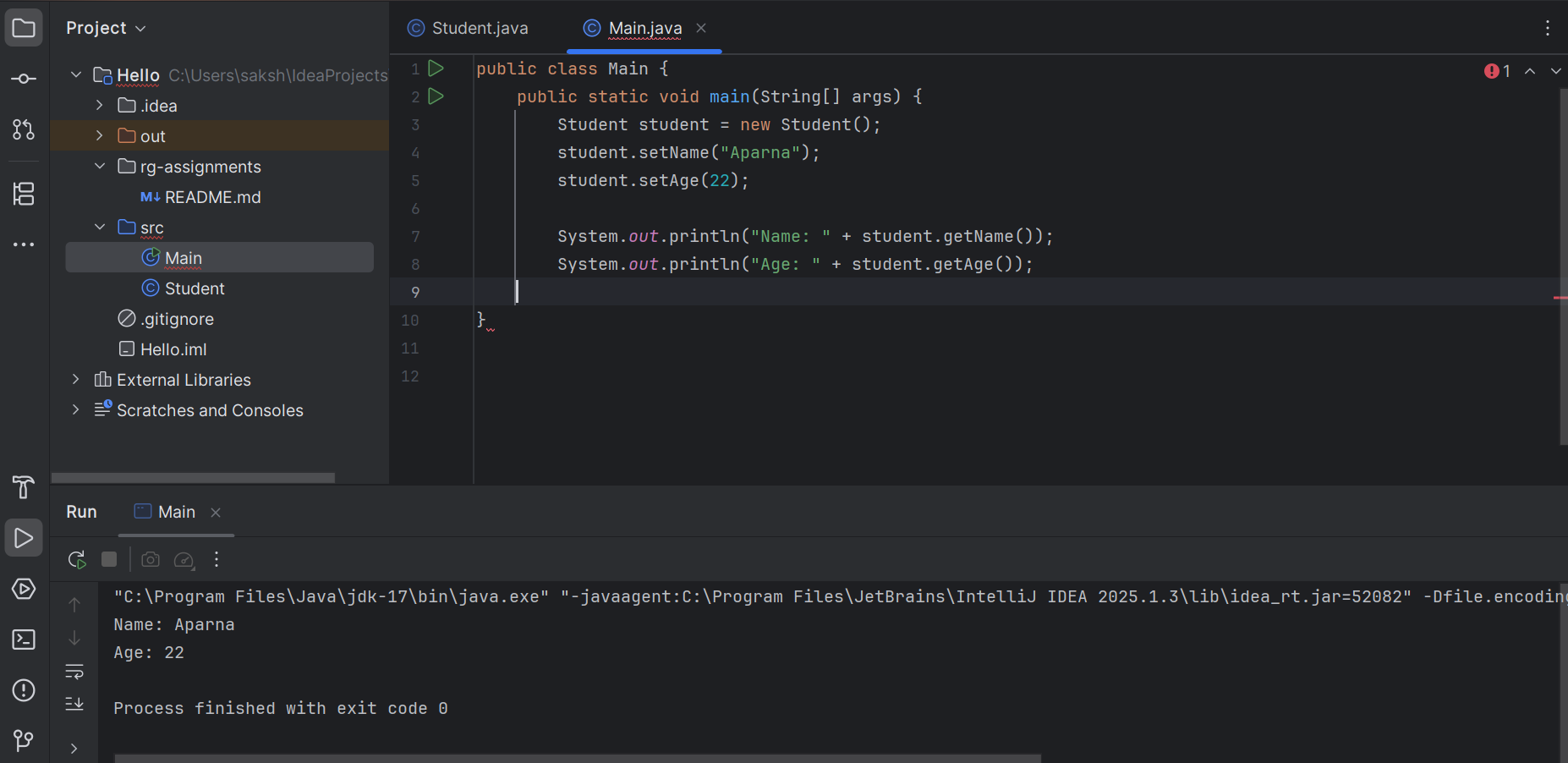
**Example:**

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;  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 Student student = new Student();  
 student.setName("Aparna");  
 student.setAge(22);  
 System.*out*.println("Name: " + student.getName());  
 System.*out*.println("Age: " + student.getAge());  
}

**Output:** Name: Aparna

Age: 21



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:**

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 and Setters

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;

}

@Override

public String toString() {

return id + " - " + name + " - " + department;

}}

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class EmployeeCRUD {

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

// Create

public void addEmployee(Employee employee) {

employeeList.add(employee);

System.out.println("Employee added: " + employee);

}

// Read

public void displayAllEmployees() {

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

for (Employee emp : employeeList) {

System.out.println(emp);

}}

// Update

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

for (Employee emp : employeeList) {

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

emp.setName(newName);

emp.setDepartment(newDepartment);

System.out.println("Employee updated: " + emp);

return;

} }

System.out.println("Employee with ID " + id + " not found."); }

// Delete

public void deleteEmployee(int id) {

Iterator<Employee> iterator = employeeList.iterator();

while (iterator.hasNext()) {

Employee emp = iterator.next();

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

iterator.remove();

System.out.println("Employee with ID " + id + " deleted.");

return;

} }

System.out.println("Employee with ID " + id + " not found.");}

public static void main(String[] args) {

EmployeeCRUD crud = new EmployeeCRUD();

// Create

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

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

// Read

crud.displayAllEmployees();

// Update

crud.updateEmployee(2, "Ravindra", "DevOps");

// Read again

crud.displayAllEmployees();

// Delete

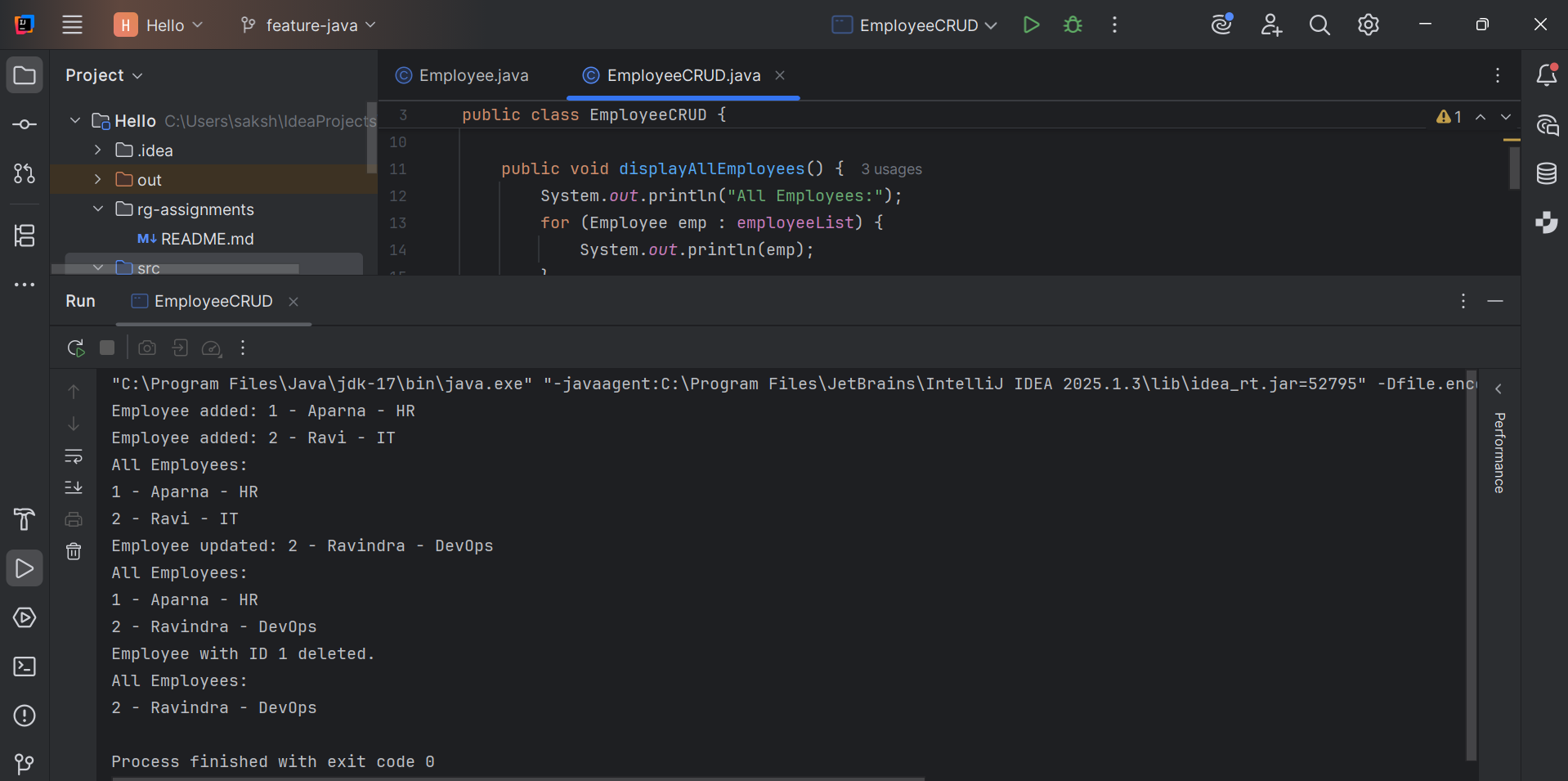
crud.deleteEmployee(1);

// Final Read

crud.displayAllEmployees();

}}

**Output:**



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;

}

Ans: **MySQL**

CREATE DATABASE test\_db;

USE test\_db;

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

department VARCHAR(50) NOT NULL

);

**Employee.java**

public 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; }

@Override

public String toString() {

return id + " - " + name + " - " + department;

}

}

**JDBCUtils.java**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class JDBCUtils {

private static final String URL = "jdbc:mysql://localhost:3306/test\_db?useSSL=false";

private static final String USER = "root";

private static final String PASSWORD = "root"; // Change to your MySQL password

public static Connection getConnection() throws SQLException {

return DriverManager.getConnection(URL, USER, PASSWORD);

}

public static void printSQLException(SQLException ex) {

for (Throwable e : ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

}

}

}

}

**EmployeeJDBC.java**

import java.sql.\*;

public class EmployeeJDBC {

public void insertEmployee(Employee emp) {

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

try (Connection con = JDBCUtils.getConnection();

PreparedStatement ps = con.prepareStatement(sql)) {

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

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

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

ps.executeUpdate();

System.out.println("Inserted: " + emp);

} catch (SQLException e) {

JDBCUtils.printSQLException(e);

}

}

public void selectAllEmployees() {

String sql = "SELECT \* FROM employees";

try (Connection con = JDBCUtils.getConnection();

PreparedStatement ps = con.prepareStatement(sql)) {

ResultSet rs = ps.executeQuery();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String department = rs.getString("department");

System.out.println(id + " - " + name + " - " + department);

}

} catch (SQLException e) {

JDBCUtils.printSQLException(e);

}

}

public void updateEmployee(Employee emp) {

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

try (Connection con = JDBCUtils.getConnection();

PreparedStatement ps = con.prepareStatement(sql)) {

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

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

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

ps.executeUpdate();

System.out.println("Updated: " + emp);

} catch (SQLException e) {

JDBCUtils.printSQLException(e);

}

}

public void deleteEmployee(int id) {

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

try (Connection con = JDBCUtils.getConnection();

PreparedStatement ps = con.prepareStatement(sql)) {

ps.setInt(1, id);

ps.executeUpdate();

System.out.println("Deleted employee with ID: " + id);

} catch (SQLException e) {

JDBCUtils.printSQLException(e);

}

}

}

**TestCRUD.java**

public class TestCRUD {

public static void main(String[] args) {

EmployeeJDBC crud = new EmployeeJDBC();

// INSERT

crud.insertEmployee(new Employee(1, "Aparna", "SWE"));

crud.insertEmployee(new Employee(2, "Swasti", "IT"));

crud.insertEmployee(new Employee(3, "Ravi", "HR"));

// READ

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

crud.selectAllEmployees();

// UPDATE

crud.updateEmployee(new Employee(1, "Aparna Singh", "DevOps"));

// READ

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

crud.selectAllEmployees();

// DELETE

crud.deleteEmployee(2); // Swasti

// READ

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

crud.selectAllEmployees();

}

}

**Output**

Inserted: 1 - Aparna - SWE

Inserted: 2 - Swasti - IT

Inserted: 3 - Ravi - HR

After Insertion:

1 - Aparna - SWE

2 - Swasti - IT

3 - Ravi - HR

Updated: 1 - Aparna Singh - DevOps

After Update:

1 - Aparna Singh - DevOps

2 - Swasti - IT

3 - Ravi - HR

Deleted employee with ID: 2

After Deletion:

1 - Aparna Singh - DevOps

3 - Ravi - HR

Process finished with exit code 0