**CORE JAVA ASSIGNMENT**



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.

Solution :

A **pure function** is a function where the output depends only on its input and does not modify any state or depend on external state.

* In this code, calculateTax() uses the instance variable rate, which could potentially be changed by other code, making the output dependent on the object's state, not just the method's input.
* Therefore, **calculateTax() is NOT a pure function**.

TO make it into pure function , Use a static final constant:

public class TaxUtil {

private static final double RATE = 0.15;

public double calculateTax(double amount) {

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();

}

}

Output :

super class show method

sub class show method

Explanation :

* Super.show(); calls the static method in Super.
* new Super.StaticMethods().show(); creates an instance of the static inner class and calls its 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();

}

}

Output :

display method

display method

10

10

Explanation :

* this.display(); and display(); both call the overridden method in ThisUse.
* this.num and num both refer to the subclass field, which is set to 10 by the constructor.

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

**Explanation:**

The **singleton pattern** ensures that only one instance of a class is created and provides a global point of access to it.

public class Singleton {

private static Singleton instance;

private Singleton() {

// private constructor

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

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

**Explanation:**

**Encapsulation** is the concept of wrapping data (fields) and code (methods) together as a single unit and restricting direct access to some of the object's components.

public class Person {

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;

}

}

6)

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

CODE :

Employee Class :

class Employee{

private int id;

private String name;

private String department;

}

class Employee {

private int id;

private String name;

private String department;

// Constructors, getters, setters

}

**EmployeeCRUD Class:**

import java.util.\*;

public class EmployeeCRUD {

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

// Create

public void addEmployee(Employee emp) {

employees.add(emp);

}

// Read

public Employee getEmployee(int id) {

for (Employee e : employees) {

if (e.getId() == id) return e;

}

return null;

}

// Update

public boolean updateEmployee(int id, String name, String dept) {

Employee e = getEmployee(id);

if (e != null) {

e.setName(name);

e.setDepartment(dept);

return true;

}

return false;

}

// Delete

public boolean deleteEmployee(int id) {

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

}

}

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;

}

CODE :

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 and setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getName() { return name; }

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

public String getDepartment() { return department; }

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

}

EMPLOYEEJDBC CLASS :

import java.sql.\*;

public class EmployeeJDBC {

private Connection conn;

// Constructor to initialize connection

public EmployeeJDBC(Connection conn) {

this.conn = conn;

}

// CREATE: Add a new employee

public void addEmployee(Employee emp) throws SQLException {

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

PreparedStatement stmt = conn.prepareStatement(sql);

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

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

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

stmt.executeUpdate();

stmt.close();

}

// READ: Get employee by ID

public Employee getEmployee(int id) throws SQLException {

String sql = "SELECT \* FROM Employee WHERE id = ?";

PreparedStatement stmt = conn.prepareStatement(sql);

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

Employee emp = null;

if (rs.next()) {

emp = new Employee(rs.getInt("id"), rs.getString("name"), rs.getString("department"));

}

rs.close();

stmt.close();

return emp;

}

// UPDATE: Update employee details

public void updateEmployee(Employee emp) throws SQLException {

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

PreparedStatement stmt = conn.prepareStatement(sql);

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

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

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

stmt.executeUpdate();

stmt.close();

}

// DELETE: Remove employee by ID

public void deleteEmployee(int id) throws SQLException {

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

PreparedStatement stmt = conn.prepareStatement(sql);

stmt.setInt(1, id);

stmt.executeUpdate();

stmt.close();

}

}