# PayPal Pre-Work Assignment

## Week 2- Core Java

---

## Question 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.

### Answer:

The above function is not a pure function. A pure function in java is a function whose output remains the same for the same input and is always deterministic.
Since the rate value can be changed by calling the object externally, the output will change depending on the rate value given to it. It violates the property of a pure function.
To make it pure, we can pass rate as a parameter as well to the calculateTax function.

## Question 2:

What will be the output for the 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();
}
}
```

## Answer:

The output of the above code will be:

super class show method
sub class show method

First, the super class's method show will be called. Then a new sub class object StaticMethods will be created from superclass and its show function will be called.

```
super class show method
sub class show method
```

# Question 3:

What will be the output for 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();
}
}
```

### Answer:

The output of the above code will be:

display method
display method
10
10

This is due to the reason that everytime the methods or attributes areca;;ed , the methods and attributes inside the sub class are called as the methods and attributes of super class are overridden in the subclass.

## Question 4:

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

### Answer:

The Singleton design pattern ensures that a class has only one instance and provides a global point of access to that instance. It is useful when you want to restrict instantiation of a class to a single object, which can be accessed globally throughout the application.

```java
// Singleton class
public class Singleton {

    // Static variable to hold the single instance of the class
    private static Singleton instance;  3 usages

    // Private constructor to prevent instantiation from outside
    private Singleton() {  1 usage
        // Initialization code, if any
    }

    // Static method to get the single instance of Singleton class
    public static Singleton getInstance() {  1 usage
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    // Example method of the singleton instance
    public void showMessage() {  1 usage
        System.out.println("Hello, I am a singleton instance!");
    }

    // Example main method to demonstrate usage
    public static void main(String[] args) {
        // Get the singleton instance
        Singleton singleton = Singleton.getInstance();
        // Use the singleton instance
        singleton.showMessage();
    }
}
```

In the above example, we can see that only a single instance is created for the class which is declared inside the class itself. Constructor is kept private so that it may not be used outside to create a new instance. Also, we have created a method, which will allow us to create an instance of a class if not there, or will return the single existing instance of the class.

## Question 5:

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

### Answer:

To ensure encapsulation in a class:

- **Private Fields:** Declare class fields as private to restrict direct access from outside the class.
- **Public Methods:** Provide public methods (getters and setters) to access and modify the private fields, enforcing controlled interaction with the class's internal state.

```java
public class Student {
    private String name;  3 usages
    private int age;  3 usages

    // Constructor
    public Student(String name, int age) {  1 usage
        this.name = name;
        this.age = age;
    }

    // Getter for name
    public String getName() {  2 usages
        return name;
    }

    // Setter for name
    public void setName(String name) {  1 usage
        this.name = name;
    }

    // Getter for age
    public int getAge() {  2 usages
        return age;
    }

```

```java
    // Setter for age
    public void setAge(int age) {  1 usage
        if (age > 0 && age < 120) {  // Example validation
            this.age = age;
        } else {
            System.out.println("Invalid age input");
        }
    }

    public static void main(String[] args) {
        // Create a Student object
        Student student = new Student( name: "Alice",  age: 20);

        // Access and modify fields using getters and setters
        System.out.println("Initial name: " + student.getName());
        student.setName("Bob");
        System.out.println("Updated name: " + student.getName());

        System.out.println("Initial age: " + student.getAge());
        student.setAge(21);
        System.out.println("Updated age: " + student.getAge());
    }
}
```

In the above example, we are using private attributes name and age, and using public getters and setters to access their value.

## Question 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;
}
```

## Answer:

Employee.java:

```java
public class Employee {  no usages
    private int id;  4 usages
    private String name;  4 usages
    private String department;  4 usages

    public Employee(int id, String name, String department){  no usages
        this.id = id;
        this.name = name;
        this.department = department;
    }

    public String getName() {  no usages
        return name;
    }

    public void setName(String name) {  no usages
        this.name = name;
    }

    public String getDepartment() {  no usages
        return department;
    }

    public void setDepartment(String department) {  no usages
        this.department = department;
    }

    public int getId() {  no usages
        return id;
    }

    public void setId(int id) {  no usages
        this.id = id;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", department='" + department + '\'' +
                '}';
    }
}
```

EmployeeCRUD.java:

```java
import java.util.ArrayList;
import java.util.List;

public class EmployeeCRUD {  2 usages
    public List<Employee> employees;  6 usages

    public EmployeeCRUD(){  1 usage
        this.employees = new ArrayList<>();
    }

    public void addEmployee(Employee employee){  3 usages
        this.employees.add(employee);
    }

    Rename usages
    public Employee getEmployeeById(int id){
        for(Employee employee: this.employees){
            if(employee.getId() == id){
                return employee;
            }
        }
        return null;
    }
```

```java
    public void updateEmployee(Employee updatedEmployee){  1 usage
        boolean found = false;
        for(Employee employee: this.employees){
            if (employee.getId() == updatedEmployee.getId()){
                employee.setName(updatedEmployee.getName());
                employee.setDepartment(updatedEmployee.getDepartment());
                found = true;
            }
        }
        if (!found) {
            this.addEmployee(updatedEmployee);
        }
    }

    public void deleteEmployee(int id){  1 usage
        this.employees.removeIf(employee -> employee.getId() == id);
    }
}
```

Main.java

```
class Main{
    public static void main(String[] args) {
        EmployeeCRUD crud = new EmployeeCRUD();

        // Create (Add Employee)
        crud.addEmployee(new Employee( id: 1,  name: "John Doe",  department: "IT"));
        crud.addEmployee(new Employee( id: 2,  name: "Jane Smith",  department: "HR"));

        // Read (Get Employee by ID)
        Employee employee = crud.getEmployeeById(1);
        System.out.println("Found Employee: " + employee);

        // Update (Update Employee)
        Employee updatedEmployee = new Employee( id: 1,  name: "John Doe Updated",  department: "IT");
        crud.updateEmployee(updatedEmployee);
        System.out.println("Updated Employee: " + crud.getEmployeeById(1));

        // Delete (Delete Employee)
        crud.deleteEmployee( id: 2);
        System.out.println("Remaining Employees after deletion:");
        for (Employee emp : crud.employees) {
            System.out.println(emp);
        }
    }
}
```

Output:

```
Found Employee: Employee{id=1, name='John Doe', department='IT'}
Updated Employee: Employee{id=1, name='John Doe Updated', department='IT'}
Remaining Employees after deletion:
Employee{id=1, name='John Doe Updated', department='IT'}

Process finished with exit code 0
```

# Question 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;
}
```

## Answer:

Employee.java:

```java
    public void setId(int id) {  no usages
        this.id = id;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", department='" + department + '\'' +
                '}';
    }
}

```

EmployeeJDBC.java

```java
package Question7;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeJDBC {  2 usages
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/mydatabase";  5 usages
    private static final String USERNAME = "root";  5 usages
    private static final String PASSWORD = "Shubham123#";  5 usages

    private static final String INSERT_SQL = "INSERT INTO employees (id, name, department) VALUES (?, ?, ?)";
    private static final String SELECT_BY_ID_SQL = "SELECT * FROM employees WHERE id = ?";  1 usage
    private static final String UPDATE_SQL = "UPDATE employees SET name = ?, department = ? WHERE id = ?";  1 usag
    private static final String DELETE_SQL = "DELETE FROM employees WHERE id = ?";  1 usage
    private static final String SELECT_ALL_SQL = "SELECT * FROM employees";  1 usage

    public void addEmployee(Employee employee) {  1 usage
        try (Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
             PreparedStatement preparedStatement = connection.prepareStatement(INSERT_SQL)) {
            preparedStatement.setInt( parameterIndex: 1, employee.getId());
            preparedStatement.setString( parameterIndex: 2, employee.getName());
            preparedStatement.setString( parameterIndex: 3, employee.getDepartment());
            preparedStatement.executeUpdate();
            System.out.println("Employee added successfully");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

```

```java
public Employee getEmployeeById(int id) {  1 usage
    Employee employee = null;
    try (Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement(SELECT_BY_ID_SQL)) {
        preparedStatement.setInt( parameterIndex: 1, id);
        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            String name = resultSet.getString( columnLabel: "name");
            String department = resultSet.getString( columnLabel: "department");
            employee = new Employee(id, name, department);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return employee;
}
```

```java
public void updateEmployee(Employee employee) {  1 usage
    try (Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement(UPDATE_SQL)) {
        preparedStatement.setString( parameterIndex: 1, employee.getName());
        preparedStatement.setString( parameterIndex: 2, employee.getDepartment());
        preparedStatement.setInt( parameterIndex: 3, employee.getId());
        int rowsUpdated = preparedStatement.executeUpdate();
        if (rowsUpdated > 0) {
            System.out.println("Employee updated successfully");
        } else {
            System.out.println("Employee not found");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```java
public void deleteEmployee(int id) {  1 usage
    try (Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement(DELETE_SQL)) {
        preparedStatement.setInt( parameterIndex: 1, id);
        int rowsDeleted = preparedStatement.executeUpdate();
        if (rowsDeleted > 0) {
            System.out.println("Employee deleted successfully");
        } else {
            System.out.println("Employee not found");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
80      public List<Employee> getAllEmployees() {  1 usage
81          List<Employee> employees = new ArrayList<>();
82          try (Connection connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
83               Statement statement = connection.createStatement();
84               ResultSet resultSet = statement.executeQuery(SELECT_ALL_SQL)) {
85              while (resultSet.next()) {
86                  int id = resultSet.getInt( columnLabel: "id");
87                  String name = resultSet.getString( columnLabel: "name");
88                  String department = resultSet.getString( columnLabel: "department");
89                  employees.add(new Employee(id, name, department));
90              }
91          } catch (SQLException e) {
92              e.printStackTrace();
93          }
94          return employees;
95      }
96  }
97
```

Main.java

```
1   package Question7;
2   import java.util.List;
3
4   public class Main {
5       public static void main(String[] args) {
6           EmployeeJDBC employeeJDBC = new EmployeeJDBC();
7
8           // Add Employee
9           Employee newEmployee = new Employee( id: 1,  name: "John Doe",  department: "IT");
10          employeeJDBC.addEmployee(newEmployee);
11          newEmployee = new Employee( id: 2,  name: "John Poe",  department: "HR");
12          employeeJDBC.addEmployee(newEmployee);
13
14
15          // Get Employee by ID
16          Employee retrievedEmployee = employeeJDBC.getEmployeeById(1);
17          System.out.println("Retrieved Employee: " + retrievedEmployee);
18
19          // Update Employee
20          retrievedEmployee.setName("John Doe Updated");
21          employeeJDBC.updateEmployee(retrievedEmployee);
22
23          // Delete Employee
24          employeeJDBC.deleteEmployee( id: 1);
25
```

```
25
26          // Get All Employees
27          List<Employee> allEmployees = employeeJDBC.getAllEmployees();
28          System.out.println("All Employees:");
29          for (Employee emp : allEmployees) {
30              System.out.println(emp);
31          }
32      }
33  }
34
```

Output:

```
C:\Users\Shubham\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=62489:C:\Program Files\JetBr
Employee added successfully
Employee added successfully
Retrieved Employee: Employee{id=1, name='John Doe', department='IT'}
Employee updated successfully
Employee deleted successfully
All Employees:
Employee{id=2, name='John Poe', department='HR'}

Process finished with exit code 0
```

Steps:

1) First downloaded, installed and configured MySQL.
2) Created Database and a table named employee.
3) Installed driver for mysql in the project.

Explanation:

In every function, first the connection to the database is tried, which is hosted on localhost. If found, then the placeholders for the SQL query for each are filled as per the required case. The database is accessed using root username and password. Then the operation is performed as per the required case, or error is printed whenever caught.