

## Week 1 | Assignment 2 | Core Java | By: Sejal Aggarwal

### Q1.

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.

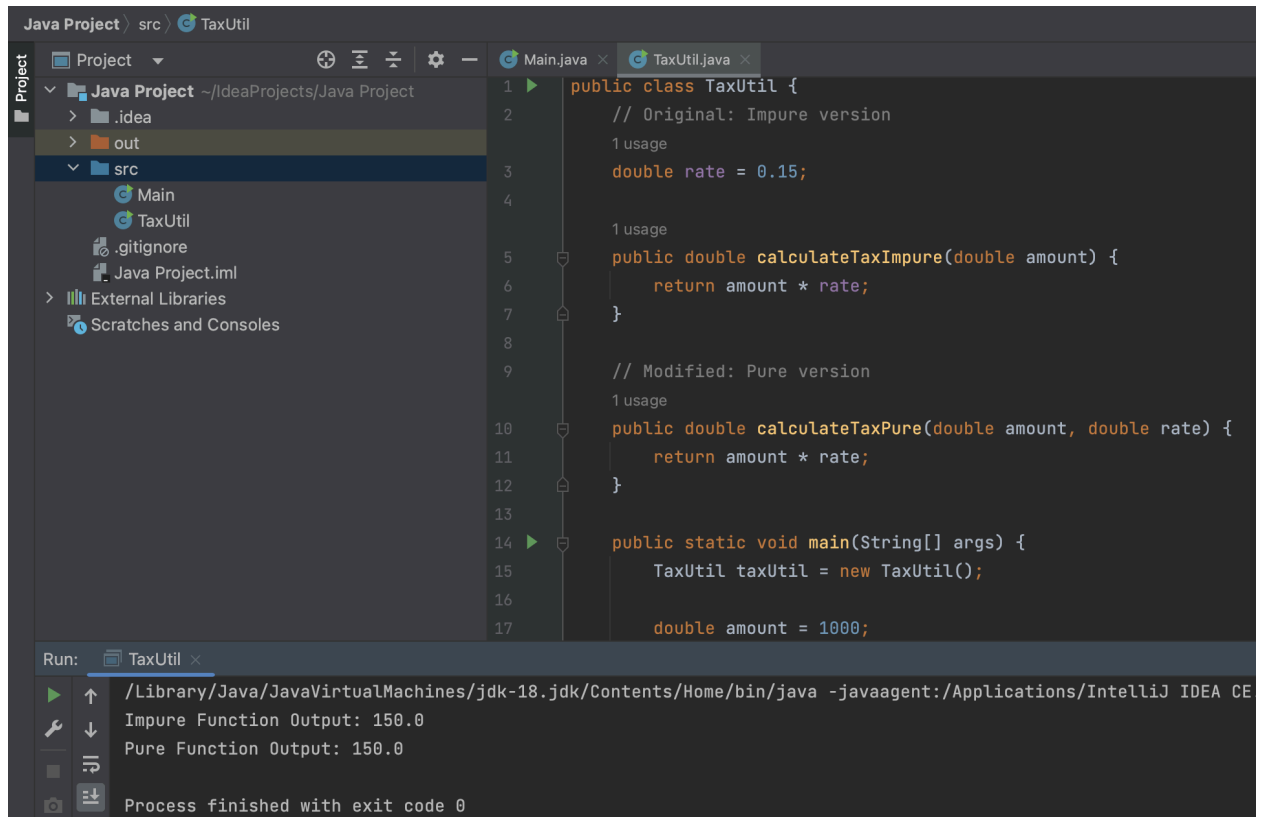
### Ans 1.

- No, method `calculateTax()` is not a pure function because it depends on the instance variable `rate`, which is external to the method and can change, breaking the pure function rules.
- Way to make the method `calculateTax()` pure is as follows:
  - Make `rate` a local variable or pass it as a parameter.

#### **Code - Modified: Pure version**

```
public class TaxUtil {  
    public double calculateTax(double amount, double rate) {  
        return amount * rate;  
    }  
}
```

O/p



## Q2.

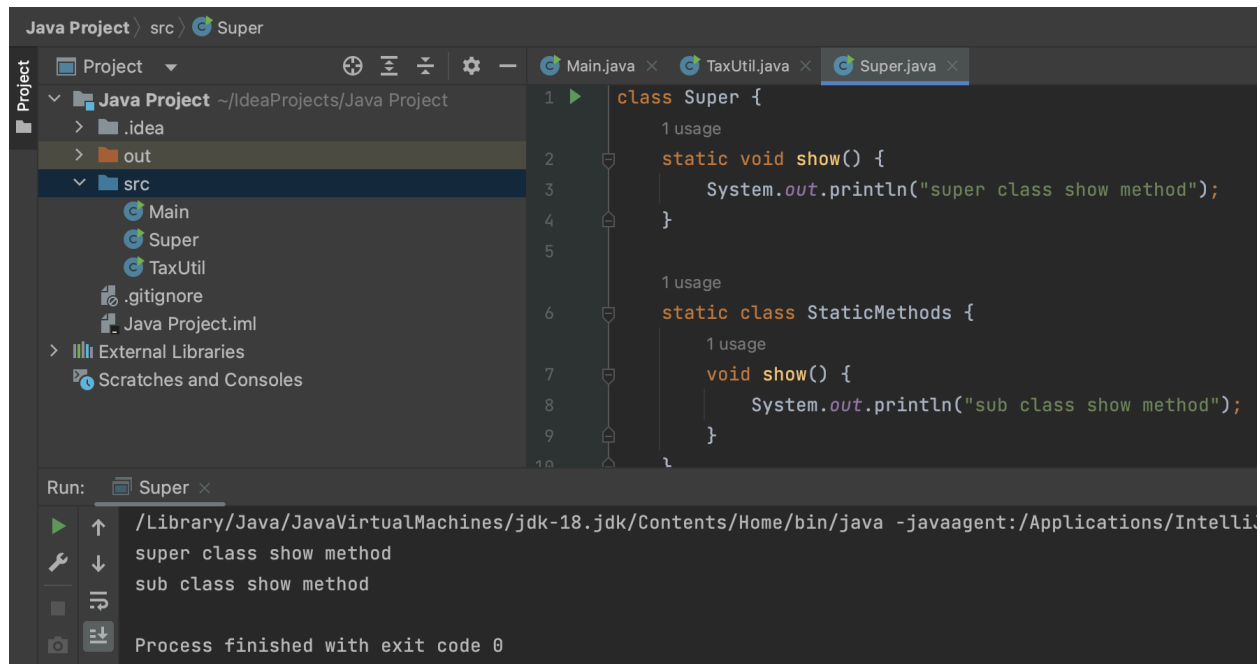
What will be the output for the following code?

```
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();
    }
}
```

```
}
```

**Ans 2.**

**O/p**



The screenshot shows an IDE with a project named 'Java Project'. The 'src' folder contains three files: 'Main', 'Super', and 'TaxUtil'. The 'Super.java' file is open, showing the following code:

```
1 class Super {  
2     1 usage  
3     static void show() {  
4         System.out.println("super class show method");  
5     }  
6  
7     1 usage  
8     static class StaticMethods {  
9         1 usage  
10        void show() {  
11            System.out.println("sub class show method");  
12        }  
13    }
```

The 'Run' tab shows the output of the program:

```
Run: Super ×  
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ  
super class show method  
sub class show method  
Process finished with exit code 0
```

**Q3.**

```
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 3.**

**O/p**

The screenshot shows an IDE with a project named 'Java Project'. The file explorer on the left shows the project structure, including a 'src' folder with files like 'Main', 'SingletonDesignPattern', 'Student', 'TaxUtil', and 'ThisUse.java'. The main editor displays the following Java code:

```

1  class Super {
2      int num = 20;
3
4      public void display() {
5          System.out.println("super class method");
6      }
7  }
9  public class ThisUse extends Super {
10     int num;
11

```

Below the code editor, the 'Run' tab is active, showing the execution of 'ThisUse'. The output is as follows:

```

/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/
display method
display method
20
20
Process finished with exit code 0

```

**Q4.**

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

**Ans 4.**

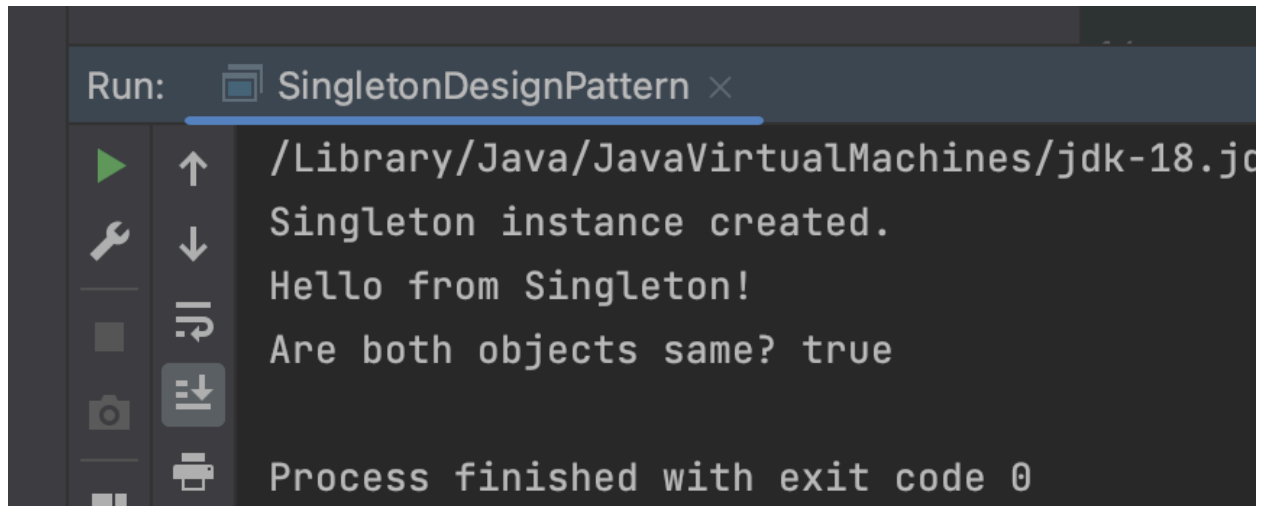
### Singleton Design Pattern

- The Singleton Design Pattern ensures that a class has only one instance and provides a global point of access to it.
- It is commonly used when exactly one object is needed to coordinate actions across a system.

### Coding Example

```
1  ▶ public class SingletonDesignPattern {
2      // Private static variable of the same class
3      3 usages
4      private static SingletonDesignPattern instance;
5      1 usage
6      private SingletonDesignPattern() {
7          System.out.println("Singleton instance created.");
8      }
9      // Public static method to provide access to the instance
10     2 usages
11     public static SingletonDesignPattern getInstance() {
12         if (instance == null) {
13             instance = new SingletonDesignPattern(); // Lazy initialization
14         }
15         return instance;
16     }
17
18     1 usage
19     public void showMessage() {
20         System.out.println("Hello from Singleton!");
21     }
22
23     ▶ public static void main(String[] args) {
24         // singleton instance
25         SingletonDesignPattern obj1 = SingletonDesignPattern.getInstance();
26         SingletonDesignPattern obj2 = SingletonDesignPattern.getInstance();
27
28         obj1.showMessage();
29
30         // Verifying both objects
31         System.out.println("Are both objects same? " + (obj1 == obj2));
32     }
33 }
```

O/p

A screenshot of a Java IDE's Run console. The title bar of the console window is labeled "Run: SingletonDesignPattern x". The console output shows the following text: 

```
/Library/Java/JavaVirtualMachines/jdk-18.jc  
Singleton instance created.  
Hello from Singleton!  
Are both objects same? true  
  
Process finished with exit code 0
```

 On the left side of the console, there is a vertical toolbar with icons for running, debugging, and other IDE functions.

**Q5.** How do we make sure a class is encapsulated? Explain with a coding example.

**Ans 5.**

### Encapsulation

- It is one of the fundamental principles of OOP (Object-Oriented Programming).
- It means hiding the internal details of an object and exposing only what's necessary using methods (getters/setters).
- It helps in data protection, control, and modularity.

### Steps to ensure a class is encapsulated?

- Make all data members private (access modifier).
- Provide public getter and setter methods to access/update private fields.
- Optionally, add validation in setters to control changes.

### Coding Example

```
//Encapsulation
public class Student {
    // Make fields private
    2 usages
    private String name;
    2 usages
    private int age;

    // Provide getters & setters
    1 usage
    public String getName() {
        return name;
    }
    1 usage
    public int getAge() { return age; }
    1 usage
    public void setName(String name) { this.name = name; }
    1 usage
    public void setAge(int age) {
        if (age > 0) { // validation
            this.age = age;
        } else {
            System.out.println("Invalid age!");
        }
    }
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Sejal");
        s.setAge(22);
        System.out.println("Name: " + s.getName());
        System.out.println("Age: " + s.getAge());
    }
}
```

O/p

```
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea
Name: Sejal
Age: 22

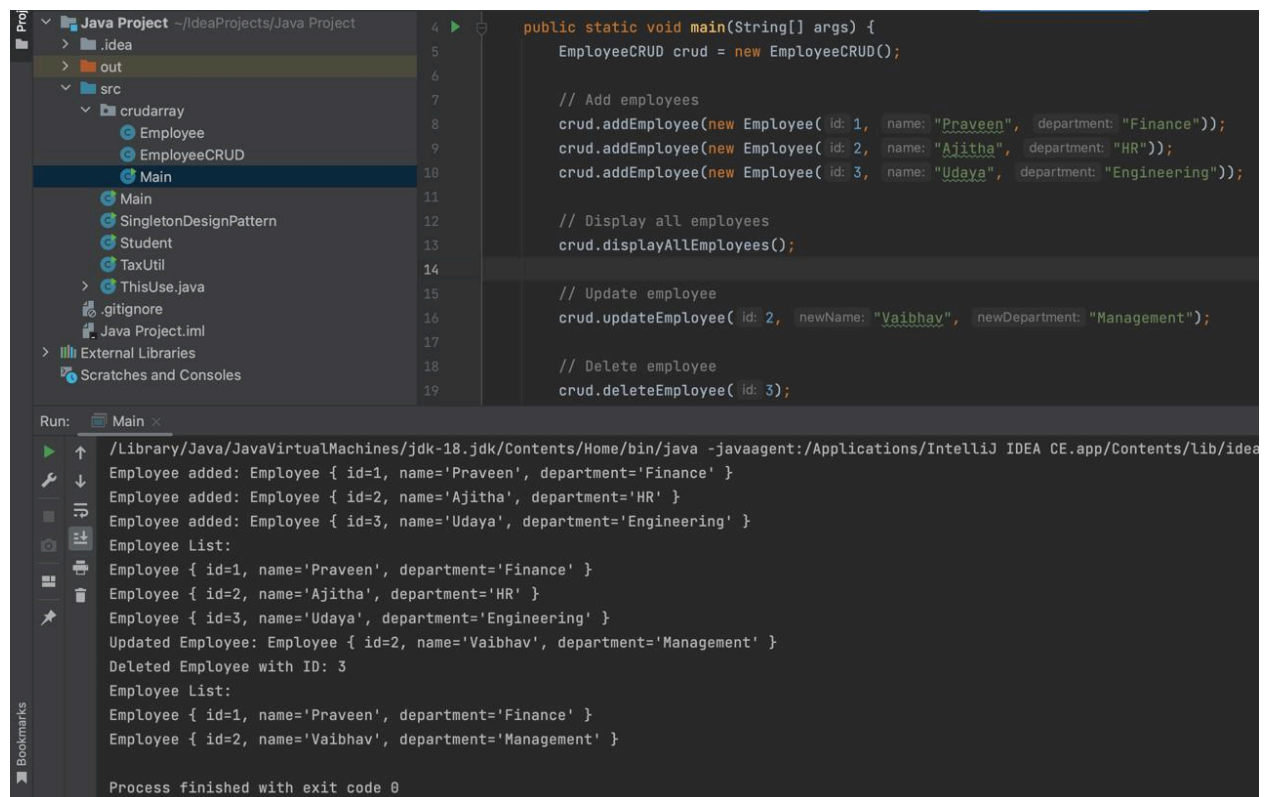
Process finished with exit code 0
```

Q6.

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;
}
```

Ans 6.



The screenshot displays the IntelliJ IDEA IDE with a Java project. The left sidebar shows the project structure with a package named 'crudarray' containing 'Employee', 'EmployeeCRUD', and 'Main' classes. The 'Main' class is selected, and its code is visible in the editor. The code defines a 'main' method that performs CRUD operations on an 'Employee' class. The output window at the bottom shows the execution results, including the addition of three employees, the display of the employee list, the update of an employee, and the deletion of an employee.

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

    // Add employees
    crud.addEmployee(new Employee( id: 1, name: "Praveen", department: "Finance"));
    crud.addEmployee(new Employee( id: 2, name: "Ajitha", department: "HR"));
    crud.addEmployee(new Employee( id: 3, name: "Udaya", department: "Engineering"));

    // Display all employees
    crud.displayAllEmployees();

    // Update employee
    crud.updateEmployee( id: 2, newName: "Vaibhav", newDepartment: "Management");

    // Delete employee
    crud.deleteEmployee( id: 3);
}
```

Run: Main x

```
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea
Employee added: Employee { id=1, name='Praveen', department='Finance' }
Employee added: Employee { id=2, name='Ajitha', department='HR' }
Employee added: Employee { id=3, name='Udaya', department='Engineering' }
Employee List:
Employee { id=1, name='Praveen', department='Finance' }
Employee { id=2, name='Ajitha', department='HR' }
Employee { id=3, name='Udaya', department='Engineering' }
Updated Employee: Employee { id=2, name='Vaibhav', department='Management' }
Deleted Employee with ID: 3
Employee List:
Employee { id=1, name='Praveen', department='Finance' }
Employee { id=2, name='Vaibhav', department='Management' }

Process finished with exit code 0
```



