

Week 1 | Assignment 2 | Core Java | By: Sanya Dureja

[Github Link](#)

Q1.

Given:

```
public class TaxUtil {  
    double rate = 0.15;  
  
    public double calculateTax(double amount) {  
        return amount * rate;  
    }  
}
```

- a) Would you consider the method calculateTax() a 'pure function'? Why or why not?
- b) If you claim the method is NOT a pure function, please suggest a way to make it pure.

Ans 1.

- a) 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.
- b) 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

The screenshot displays the IntelliJ IDEA IDE interface. On the left, the Project tool window shows the project structure: `Java Project` (root) contains `.idea`, `out`, and `src`. The `src` folder contains `Main` and `TaxUtil` files, along with `.gitignore` and `Java Project.iml`. The `External Libraries` and `Scratches and Consoles` sections are also visible.

The main editor window shows the `TaxUtil.java` file with the following code:

```
1 public class TaxUtil {
2     // Original: Impure version
3     1 usage
4     double rate = 0.15;
5
6     1 usage
7     public double calculateTaxImpure(double amount) {
8         return amount * rate;
9     }
10
11     // Modified: Pure version
12     1 usage
13     public double calculateTaxPure(double amount, double rate) {
14         return amount * rate;
15     }
16
17     public static void main(String[] args) {
18         TaxUtil taxUtil = new TaxUtil();
19
20         double amount = 1000;
```

Below the code editor, the Run tool window shows the execution of the `TaxUtil` class. The command used is `/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE`. The output shows the results of the `calculateTaxImpure` and `calculateTaxPure` methods, both returning 150.0.

```
Run: TaxUtil x
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE
Impure Function Output: 150.0
Pure Function Output: 150.0
Process finished with exit code 0
```

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 Java project. The code in `Super.java` is as follows:

```
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    }
14 }
```

The Run console shows the following output:

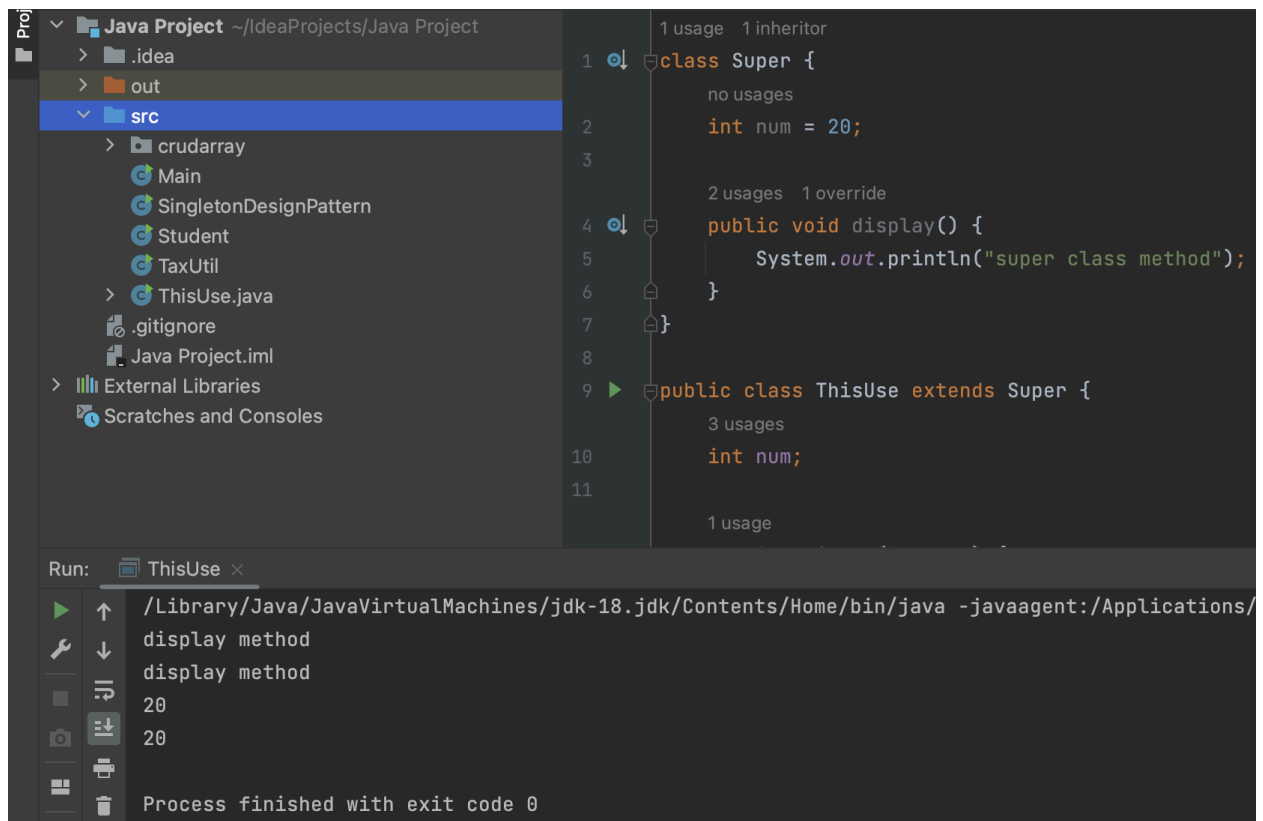
```
Run: Super x
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ
super class show method
sub class show method
Process finished with exit code 0
```

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 displays an IDE interface with a project named "Java Project" located at "~\IdeaProjects\Java Project". The project structure in the left sidebar includes a "src" folder containing files like "Main", "SingletonDesignPattern", "Student", "TaxUtil", and "ThisUse.java". The main editor window shows the code for "Super" and "ThisUse" classes. The "Super" class has a static variable "num" set to 20 and a "display" method that prints "super class method". The "ThisUse" class extends "Super" and has its own "num" variable. The "Run" tab at the bottom shows the execution of "ThisUse", with the output "display method" and the value "20" printed twice, indicating that the static variable was accessed and updated correctly.

```
1 class Super {  
2     int num = 20;  
3  
4     public void display() {  
5         System.out.println("super class method");  
6     }  
7 }  
8  
9 public class ThisUse extends Super {  
10     int num;  
11 }  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

Run: ThisUse x

/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.

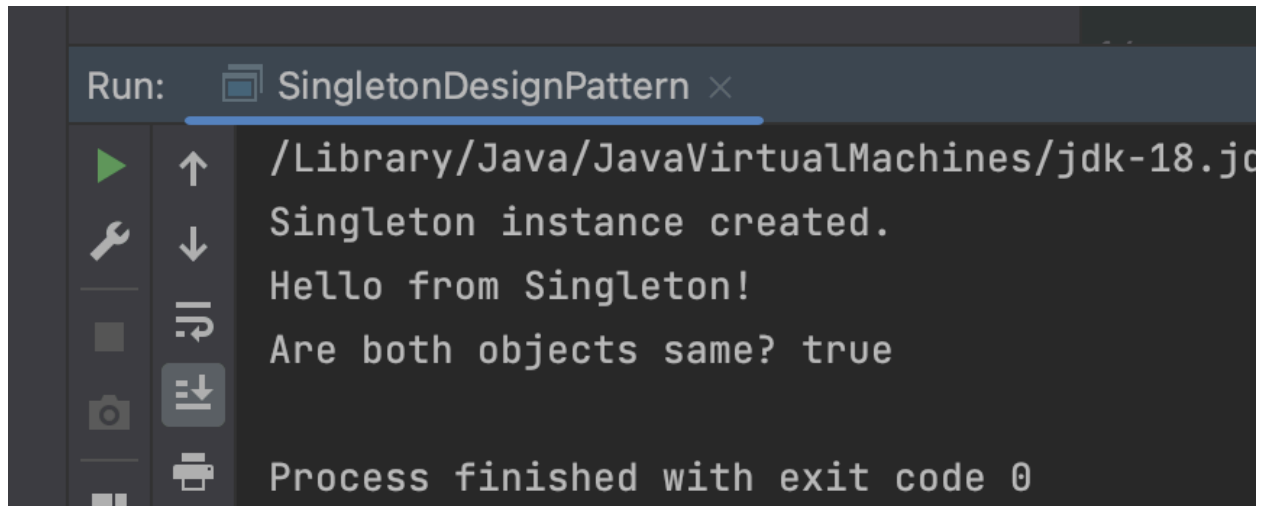
Singleton Design Pattern

- 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     1 usage
18     public void showMessage() {
19         System.out.println("Hello from Singleton!");
20     }
21     public static void main(String[] args) {
22         // singleton instance
23         SingletonDesignPattern obj1 = SingletonDesignPattern.getInstance();
24         SingletonDesignPattern obj2 = SingletonDesignPattern.getInstance();
25
26         obj1.showMessage();
27
28         // Verifying both objects
29         System.out.println("Are both objects same? " + (obj1 == obj2));
30     }
31 }
```

O/p



The screenshot shows a console window titled "Run: SingletonDesignPattern". The output text is as follows:

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

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.

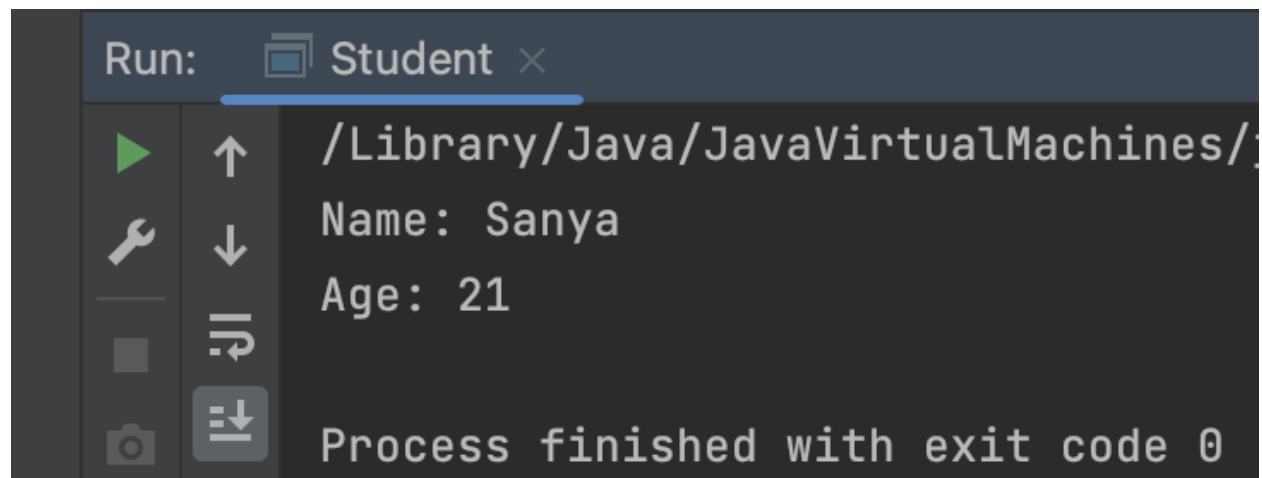
Coding Example

```
//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("Sanya");
        s.setAge(21);
        System.out.println("Name: " + s.getName());
        System.out.println("Age: " + s.getAge());
    }
}
```


O/p



The screenshot shows the 'Run' console of an IDE. The title bar reads 'Run: Student x'. The console output is as follows:

```
/Library/Java/JavaVirtualMachines/  
Name: Sanya  
Age: 21  
  
Process finished with exit code 0
```

On the left side of the console, there is a vertical toolbar with icons for running (green play button), debugging (wrench), and other actions. The 'Run' button (green play button) is currently active.

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.

O/p

The screenshot displays the IntelliJ IDEA IDE with a project named 'Java Project'. The project structure on the left includes a 'src' directory with subdirectories 'crudarray' and 'SingletonD'. The 'crudarray' directory contains files for 'Employee', 'EmployeeCRUD', and 'Main'. The 'Main.java' file is open in the editor, showing the following code:

```
package crudarray;

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

        // Add employees
        crud.addEmployee(new Employee( id: 1, name: "Sanya", department: "Finance"));
        crud.addEmployee(new Employee( id: 2, name: "Suhani", department: "HR"));
        crud.addEmployee(new Employee( id: 3, name: "Anu", department: "Engineering"));

        // Display all employees
        crud.displayAllEmployees();

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

The 'Run' tab at the bottom shows the execution output for the 'Main' class. The output is as follows:

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

Process finished with exit code 0
```