

## Week 1 | Assignment 2 | Core Java | Ankita Mohan

**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) The method calculateTax() is **not a pure function** because it uses the instance variable rate defined in the class. Since rate can be modified outside the method, the output of calculateTax() can vary even if the input amount remains the same. This breaks the rule of **determinism**, which is a key property of pure functions.
- b) To make it a **pure function**, we should remove the dependency on external state and pass rate as a parameter

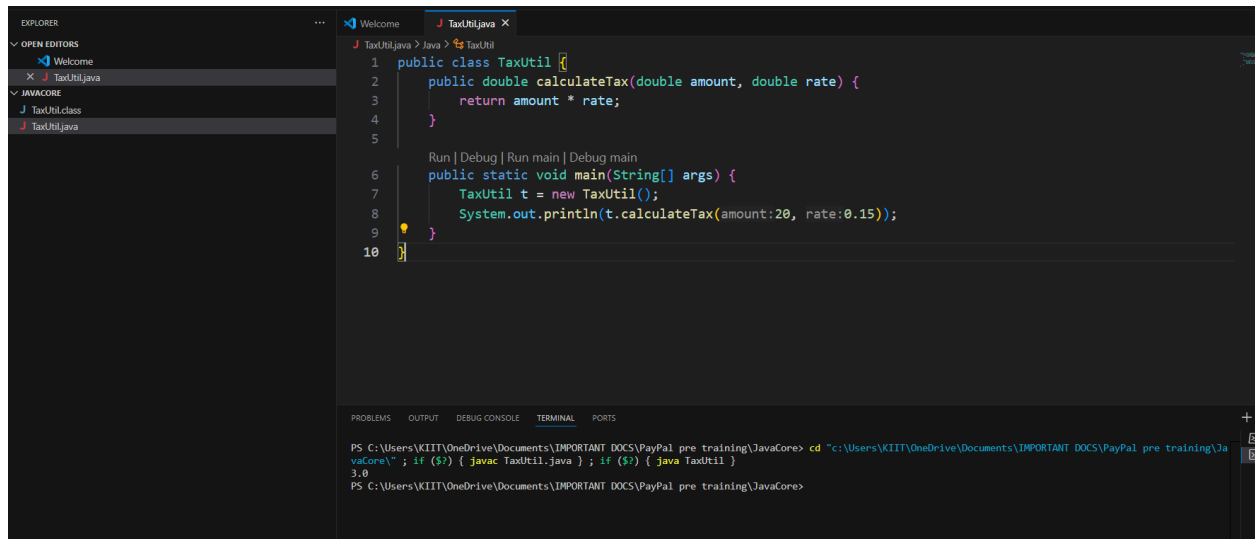
Way to make the method calculateTax() pure is as follows:

- 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**



```
1 public class TaxUtil {
2     public double calculateTax(double amount, double rate) {
3         return amount * rate;
4     }
5 }
6
7 public static void main(String[] args) {
8     TaxUtil t = new TaxUtil();
9     System.out.println(t.calculateTax(20, 0.15));
10 }
```

Run | Debug | Run main | Debug main

PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore> cd "C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore\" ; if (\$?) { javac TaxUtil.java } ; if (\$?) { java TaxUtil }
3.0
PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore>

**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.**

1. `Super.show();`

This calls the static method `show()` defined in the outer class `Super`.

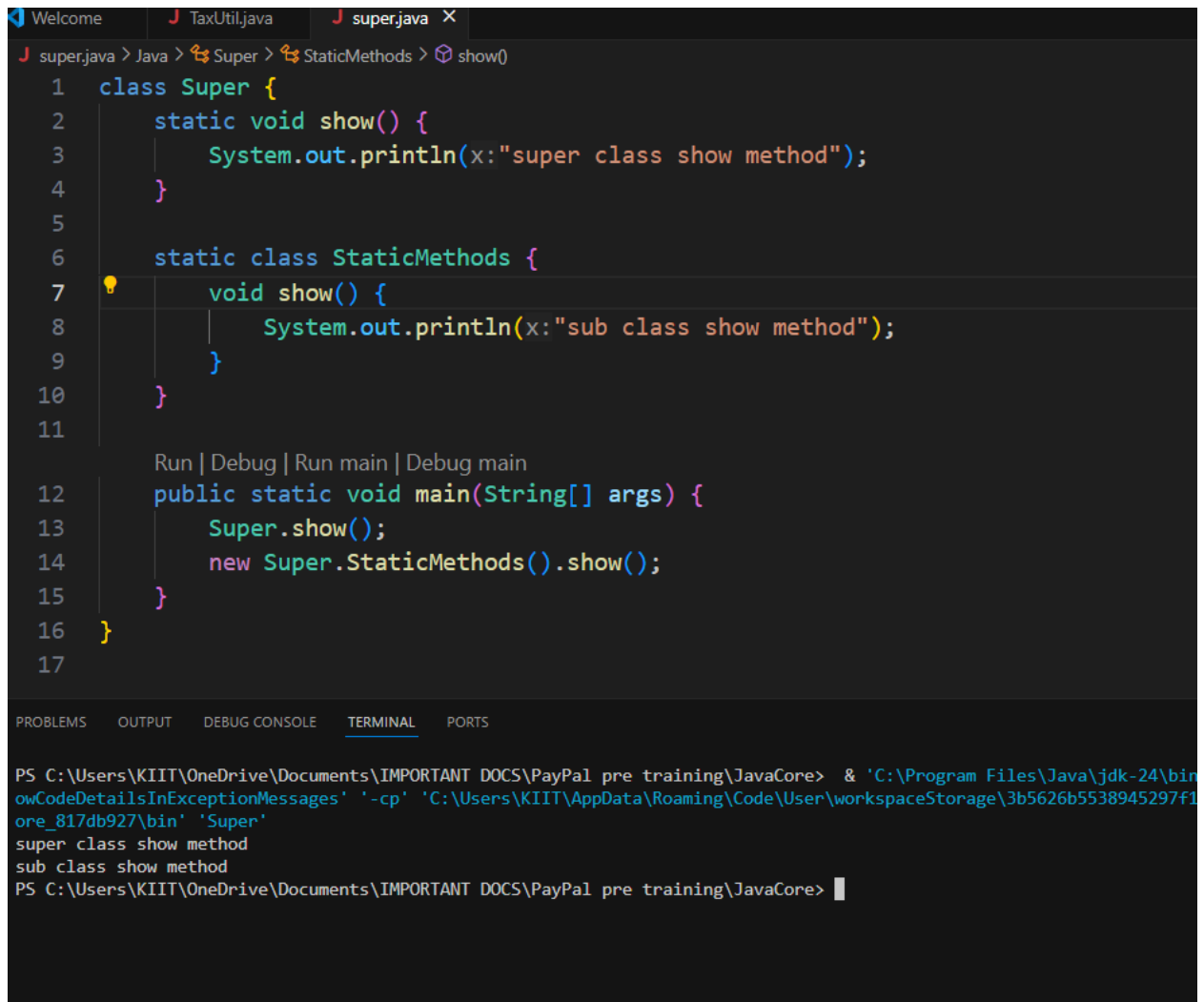
Output: super class show method

2. `new Super.StaticMethods().show();`

This creates an object of the static nested class `StaticMethods` and calls its `show()` method.

Output: sub class show method

O/p



The screenshot shows an IDE with a Java file named `super.java`. The code defines a class `Super` with a static method `show()` that prints "super class show method". It also has a static nested class `StaticMethods` with its own `show()` method that prints "sub class show method". The `main` method calls both `Super.show();` and `new Super.StaticMethods().show();`. The terminal output shows the execution of the program, which prints "super class show method" followed by "sub class show method".

```
1 class Super {
2     static void show() {
3         System.out.println(x:"super class show method");
4     }
5
6     static class StaticMethods {
7         void show() {
8             System.out.println(x:"sub class show method");
9         }
10    }
11
12    public static void main(String[] args) {
13        Super.show();
14        new Super.StaticMethods().show();
15    }
16 }
17
```

Run | Debug | Run main | Debug main

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore> & 'C:\Program Files\Java\jdk-24\bin\java.exe' -cp 'C:\Users\KIIT\AppData\Roaming\Code\User\workspaceStorage\3b5626b5538945297f1ore\_817db927\bin' 'Super'

super class show method

sub class show method

PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore> |

**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.**

ThisUse extends Super.

The child class overrides the display() method.

It also has its own num variable and constructor assigning this.num = 10

**O/p**

```
1 class Super {
2     int num = 20;
3
4     public void display() {
5         System.out.println(x:"super class method");
6     }
7 }
8
9 public class ThisUse extends Super {
10     int num;
11
12     public ThisUse(int num) {
13         this.num = num;
14     }
15
16     public void display() {
17         System.out.println(x:"display method");
18     }
19
20     public void Show() {
21         this.display(); // Calls overridden display() method
22     }
23 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore> cd "c:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore"
$?) { javac ThisUse.java } ; if ($?) { java ThisUse }
display method
display method
10
10
PS C:\Users\KIIT\OneDrive\Documents\IMPORTANT DOCS\PayPal pre training\JavaCore>
```

**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

```

Run: SingletonDesignPattern ×
  ▶ /Library/Java/JavaVirtualMachines/jdk-18.jd
  ⚙ 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

- Encapsulation means wrapping data (variables) and methods (functions) that operate on the data into a single unit — typically a class — and restricting direct access to some of the class's components.

### Steps to ensure a class is encapsulated?

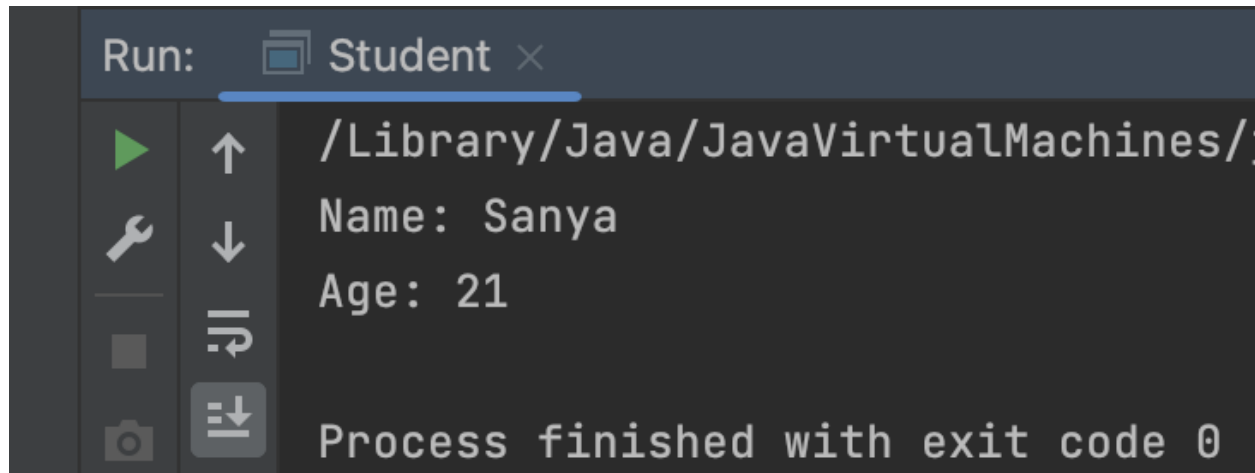
- Make all data members private (access modifier).
- Provide public getter and setter methods to access/update private fields.
- Optionally, add validation in setters to control changes.

### 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



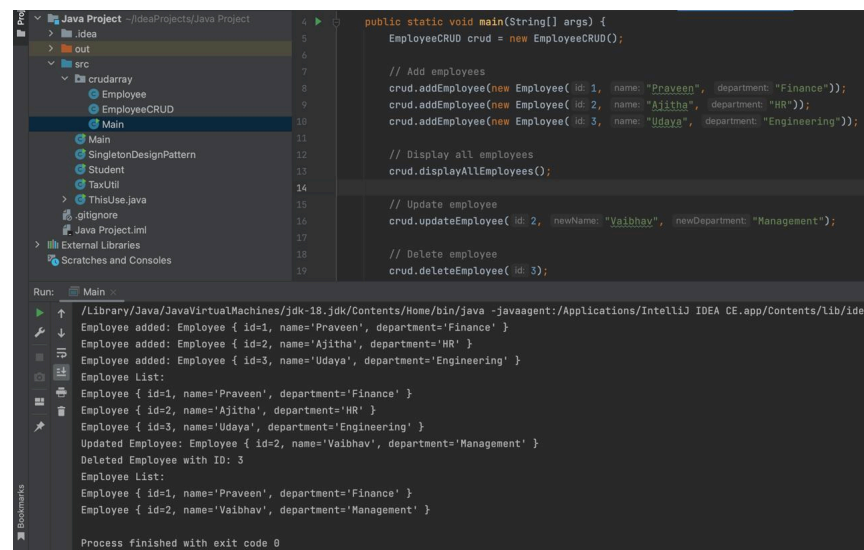
```
Run: Student ×  
/Library/Java/JavaVirtualMachines/  
Name: Sanya  
Age: 21  
Process finished with exit code 0
```

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.



```
public static void main(String[] args) {  
    EmployeeCRUD crud = new EmployeeCRUD();  
  
    // Add employees  
    crud.addEmployee(new Employee(1, "Praveen", "Finance"));  
    crud.addEmployee(new Employee(2, "Ajitha", "HR"));  
    crud.addEmployee(new Employee(3, "Udaya", "Engineering"));  
  
    // Display all employees  
    crud.displayAllEmployees();  
  
    // Update employee  
    crud.updateEmployee(2, "Vaibhav", "Management");  
  
    // Delete employee  
    crud.deleteEmployee(3);  
}
```

```
Run: Main ×  
/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
```