## 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;

}

}

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

**Ans 1.**

1. 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.
2. 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 {

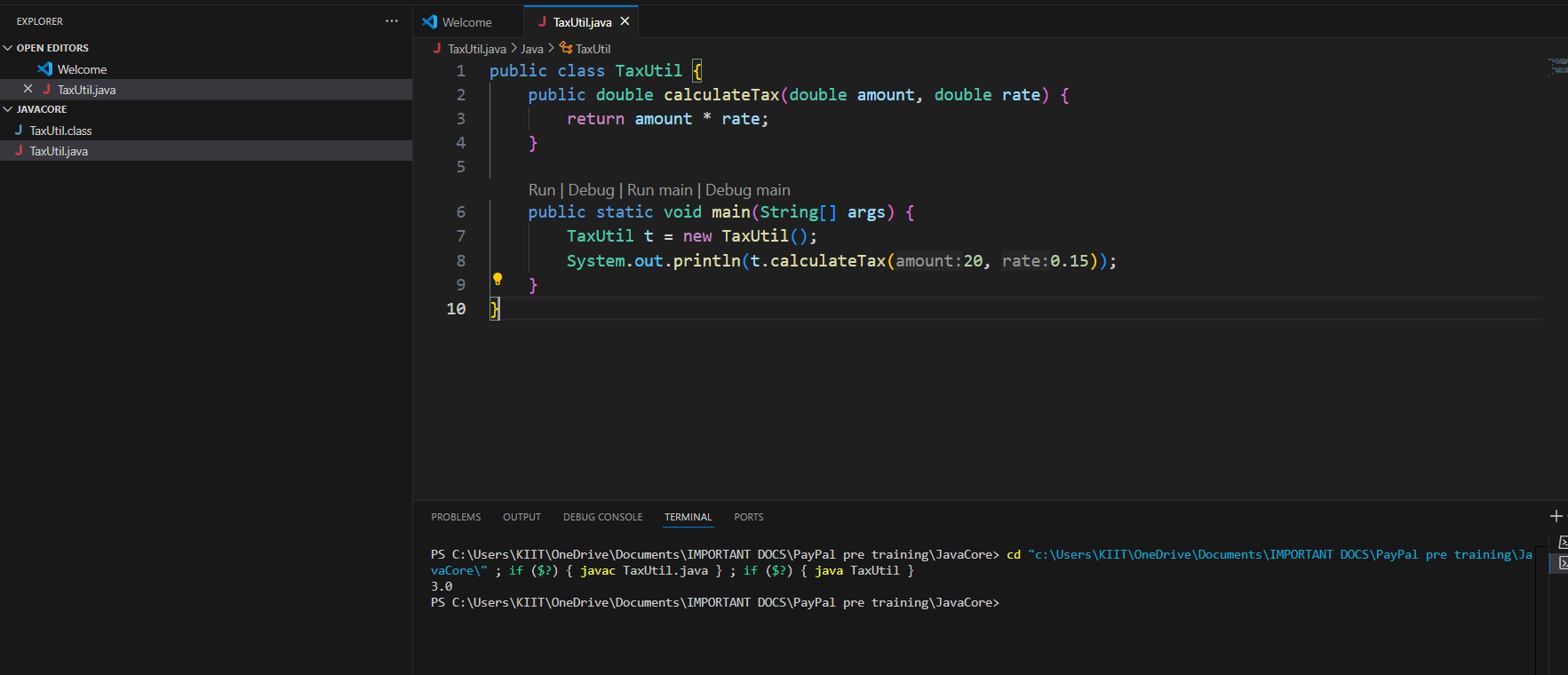
public double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

**O/p**

****

**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();

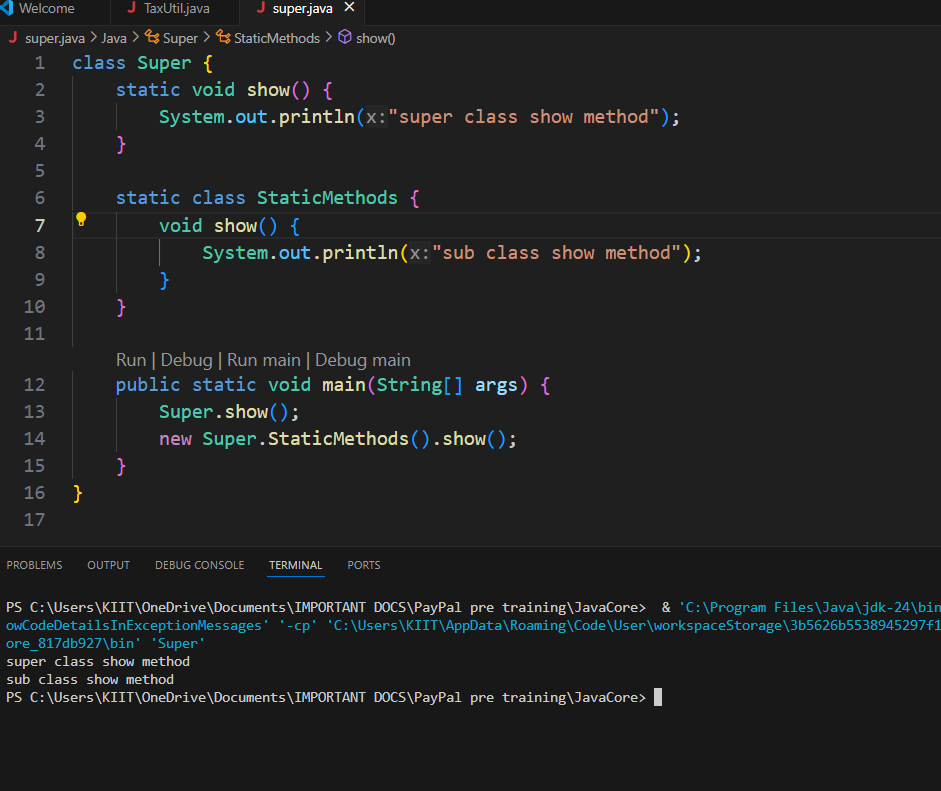
}

}

**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**

****

**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();

}

}

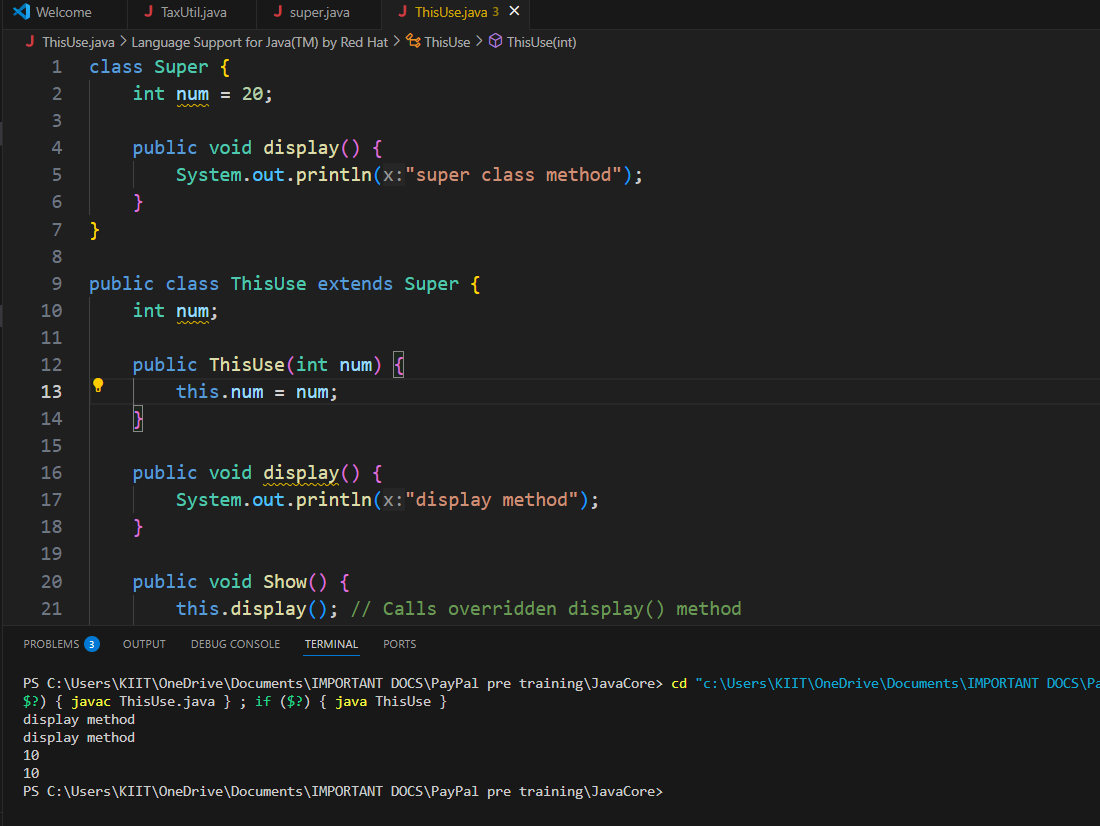
**Ans 3.**

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**

****

**Q4.**

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

**Ans 4.**

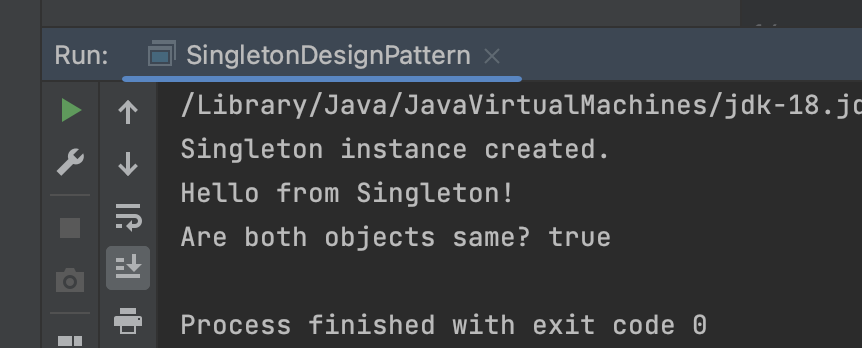
Singleton Design Pattern

* 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**



**O/p**

****

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

**Ans 5.**

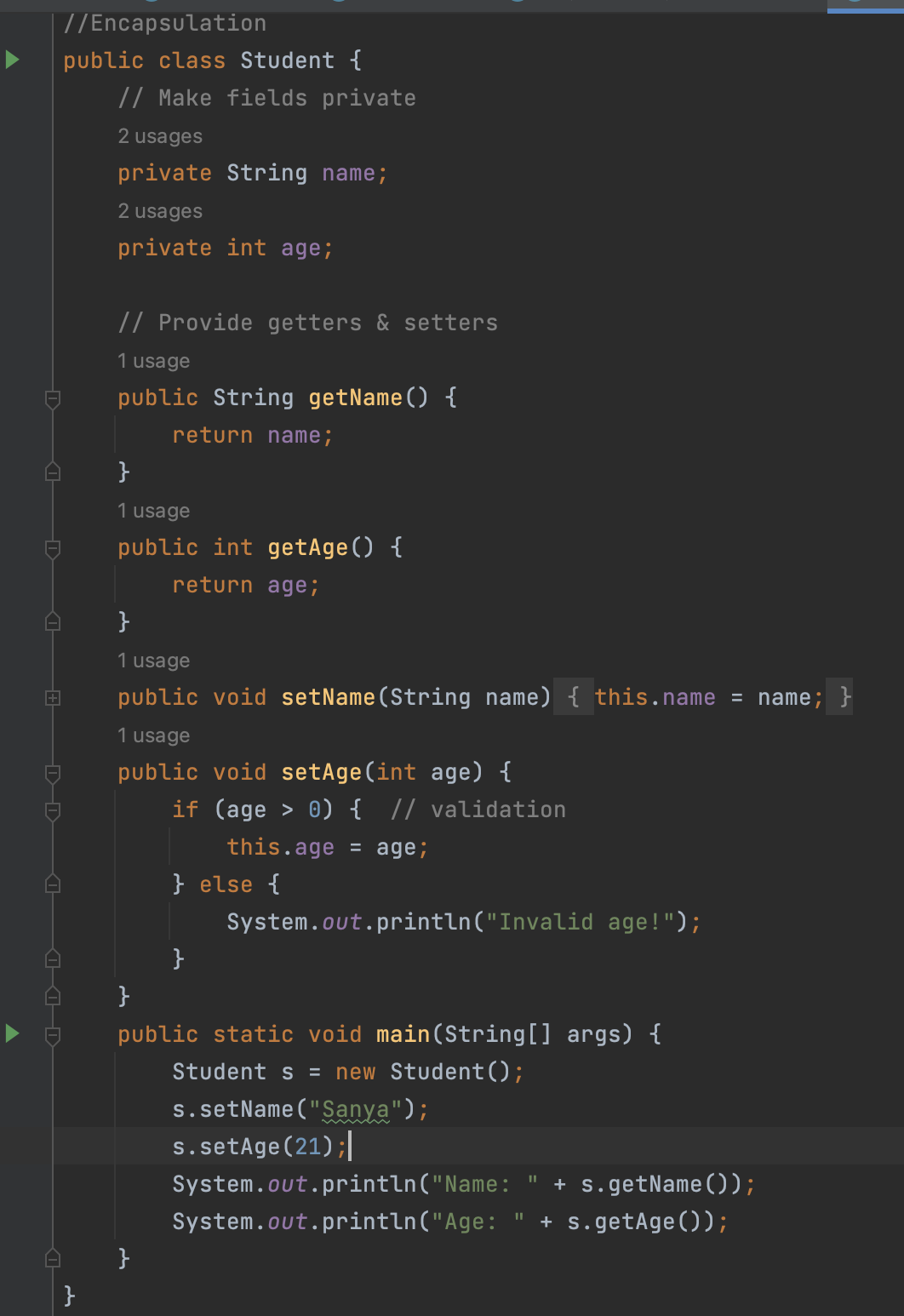
**Encapsulation**

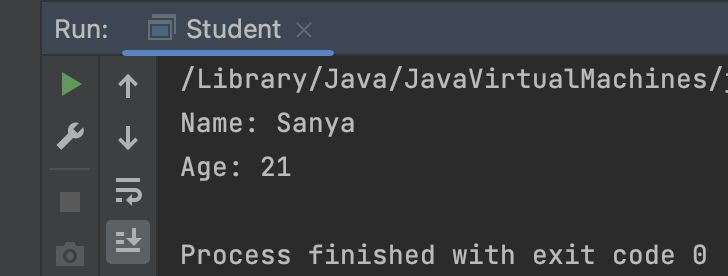
* Encapsulation means wrapping data (variables) and methods (functions) that operate on the data into a single unit — typically a class — and restricting direct access to some of the class's components.

**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**



**O/p**

**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.**

