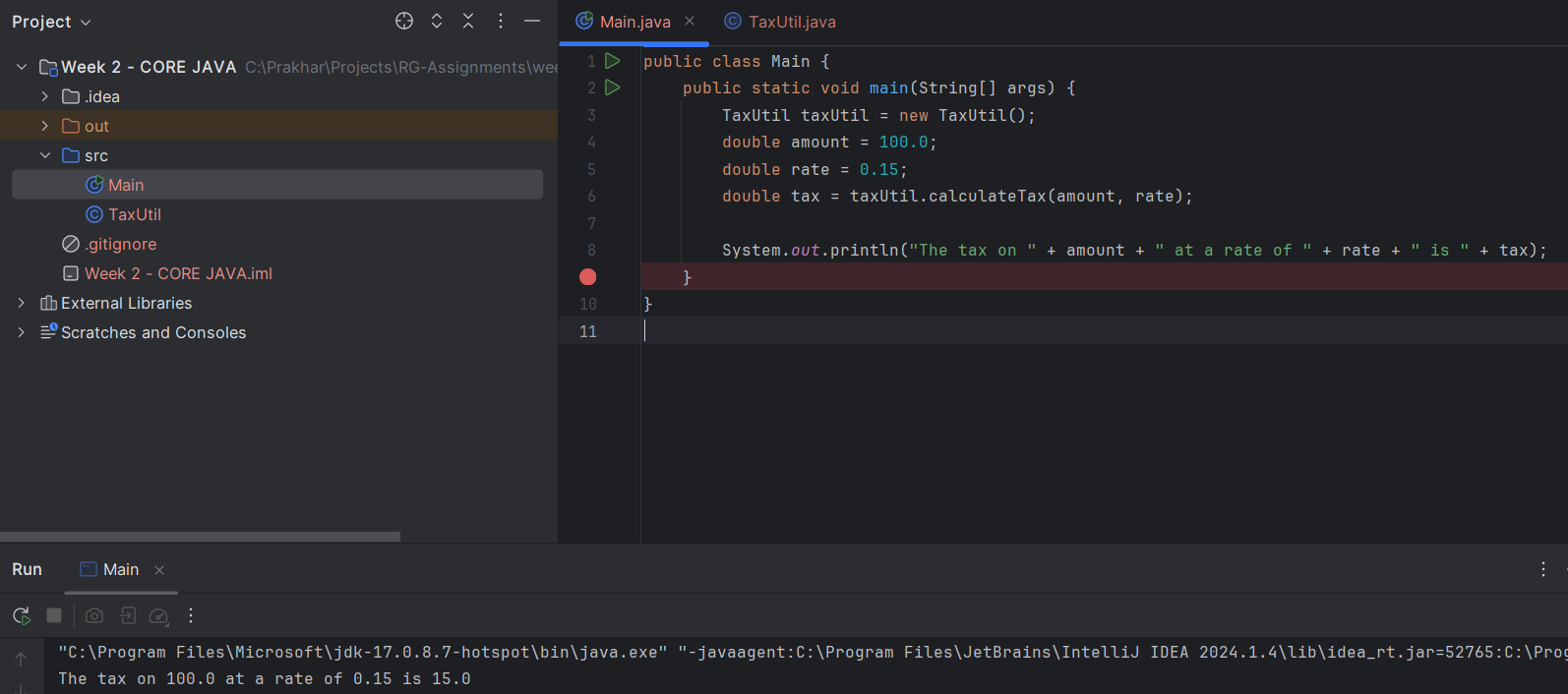
**Week 2 – Core Java**   
  
1. A pure function is a function that always produces the same output for the same input and has no side effects (it does not depend on or modify any state outside its scope).

In the given code, the calculateTax method is not a pure function. This is because it depends on the instance variable rate, which could potentially change. If rate is modified, the output of calculateTax will change for the same input, violating the principles of a pure function.

To make calculateTax a pure function, We should pass rate as a parameter to the method instead of using the instance variable.



2. What will be the output for 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.

**Explanation :**

The **Super.show()** call invokes the static method in the Super class, which prints "super class show method".

The new **Super.StaticMethods().show()** call first creates an instance of the nested static class StaticMethods and then invokes its show method, which prints "sub class show method".

**Output :**

super class show method

sub class show method

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();

}

}

Ans.   
**Explanation :**   
-> Super class has an instance variable num initialized to 20 and a method display that prints "super class method".

-> **ThisUse** class extends Super and has its own instance variable num. It has a constructor that initializes num and a method display that prints "display method". It also has a show method.

-> **ThisUse o = new ThisUse(10) :** creates an instance of ThisUse with num set to 10.

-> **o.show() :** calls the show method on this instance.

-> **this.display() :** calls the display method of the ThisUse class (since this refers to the current instance of ThisUse), so it prints "display method".

-> **display() :** without this also calls the display method of the ThisUse class, so it prints "display method" again.

-> **System.out.println(this.num) :** prints the value of this.num, which is 10.

-> **System.out.println(num) :** without this also prints the value of the instance variable num, which is 10.

**Output :**

display method

display method

10

10

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

Ans. The Singleton method or Singleton Design pattern is one of the simplest design patterns. It ensures a class only has one instance, and provides a global point of access to it.

It involves :

**Private Constructor**: To prevent instantiation from other classes.

**Static Instance:** A static variable that holds the single instance of the class.

**Static Method:** A static method to provide the global point of access to the instance.

Coding Example : simple Logger class that will ensure that all log messages are routed through the same instance, maintaining a consistent log.

public class Logger {

private static Logger instance;

private Logger() {

// private constructor

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("Log: " + message);

}

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("This is the first log message.");

logger2.log("This is the second log message.");

// Verify that logger1 and logger2 are the same instance

if (logger1 == logger2) {

System.out.println("Both logger1 and logger2 are the same instance.");

} else {

System.out.println("logger1 and logger2 are different instances.");

}

}

}

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

Ans. Encapsulation involves bundling the data (variables) and the methods that operate on the data into a single unit or class and restricting direct access to some of the object's components.

Two things to remember :

* Make all instance variables private.
* Provide public getter and setter methods

Example : simple example demonstrating encapsulation for a BankAccount class:

public class BankAccount {

// Private instance variables

private String accountNumber;

private double balance;

// Constructor to initialize the BankAccount object

public BankAccount(String accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

if (initialBalance >= 0) {

this.balance = initialBalance;

} else {

System.out.println("Initial balance must be positive.");

this.balance = 0;

}

}

// Public getter method for account number

public String getAccountNumber() {

return accountNumber;

}

// Public getter method for balance

public double getBalance() {

return balance;

}

// Public method to deposit money into the account

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: $" + amount);

} else {

System.out.println("Deposit amount must be positive.");

}

}

// Public method to withdraw money from the account

public void withdraw(double amount) {

if (amount > 0) {

if (amount <= balance) {

balance -= amount;

System.out.println("Withdrew: $" + amount);

} else {

System.out.println("Insufficient balance.");

}

} else {

System.out.println("Withdrawal amount must be positive.");

}

}

// Main method to demonstrate encapsulation

public static void main(String[] args) {

// Create a BankAccount object

BankAccount account = new BankAccount("123456789", 1000.0);

// Access and display the account number and balance

System.out.println("Account Number: " + account.getAccountNumber());

System.out.println("Initial Balance: $" + account.getBalance());

// Deposit money into the account

account.deposit(500.0);

System.out.println("Balance after deposit: $" + account.getBalance());

// Withdraw money from the account

account.withdraw(300.0);

System.out.println("Balance after withdrawal: $" + account.getBalance());

// Try to withdraw more money than the current balance

account.withdraw(1500.0);

// Try to deposit a negative amount

account.deposit(-100.0);

}

}

Program Output :   
Account Number: 123456789

Initial Balance: $1000.0

Deposited: $500.0

Balance after deposit: $1500.0

Withdrew: $300.0

Balance after withdrawal: $1200.0

Insufficient balance.

Deposit amount must be positive.

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;

}

Ans. MySql Command Line

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Code will be updated in the Java-feature branch