Task 1

What are streams?

It is like a source of data (like numbers or words). Instead of operating on elements of the list, we’re telling the stream to apply functions to itself— and give us output without changing the original pile.

Task 2

What do you understand by :

**Boilerplate code:** repetitive code ,

**Lack of parallelism** : can't easily run tasks at the same time , two tasks cannot run simultaneously. Example : For loop , each task will complete before starting another.

**Lack of Composition** : hard to mix and reuse small logical steps as code is longer, harder to read, and difficult to change or reuse.

Task3

List of Intermediate and terminal operations

**filter** Remove items based on a condition.  
 Example: .filter(n -> n % 2 == 0)

**map** Transform each element.  
 Example: .map(String::toUpperCase)

**flatMap** Replace each item with a stream and flatten.  
 Example: .flatMap(list -> list.stream())

**distinct()** Remove duplicate elements.

**sorted()** / **sorted** Sort elements naturally or with a comparator.

**peek** View elements as they pass through (mainly for debugging).

**limit(long maxSize)** Take the first *n* elements or skip the first *n*.

**Terminal Operations**

**forEach(Consumer<T>) / forEachOrdered** – perform an action for each element.

**collect(Collector)** – gather elements into a collection or structure.

**toArray() / toList() –** convert to array or list.

**reduce(...)** – combine elements into one result (e.g., sum).

**count()** – count elements.

**min() / max()** – get smallest or largest element via comparator.

**anyMatch(...), allMatch(...), noneMatch(...) –** check matching condition.

**findFirst() / findAny() – r**eturn an element if present

: Advantages of Streams:

1. Clean and Concise Code

2. Easier to Chain multiple Operations

Tak 4

package MultiThreading.Lambda;

@FunctionalInterface

interface MyInterface {

double getPiValue();

}

public class Task3 {

public static void main(String[] args) {

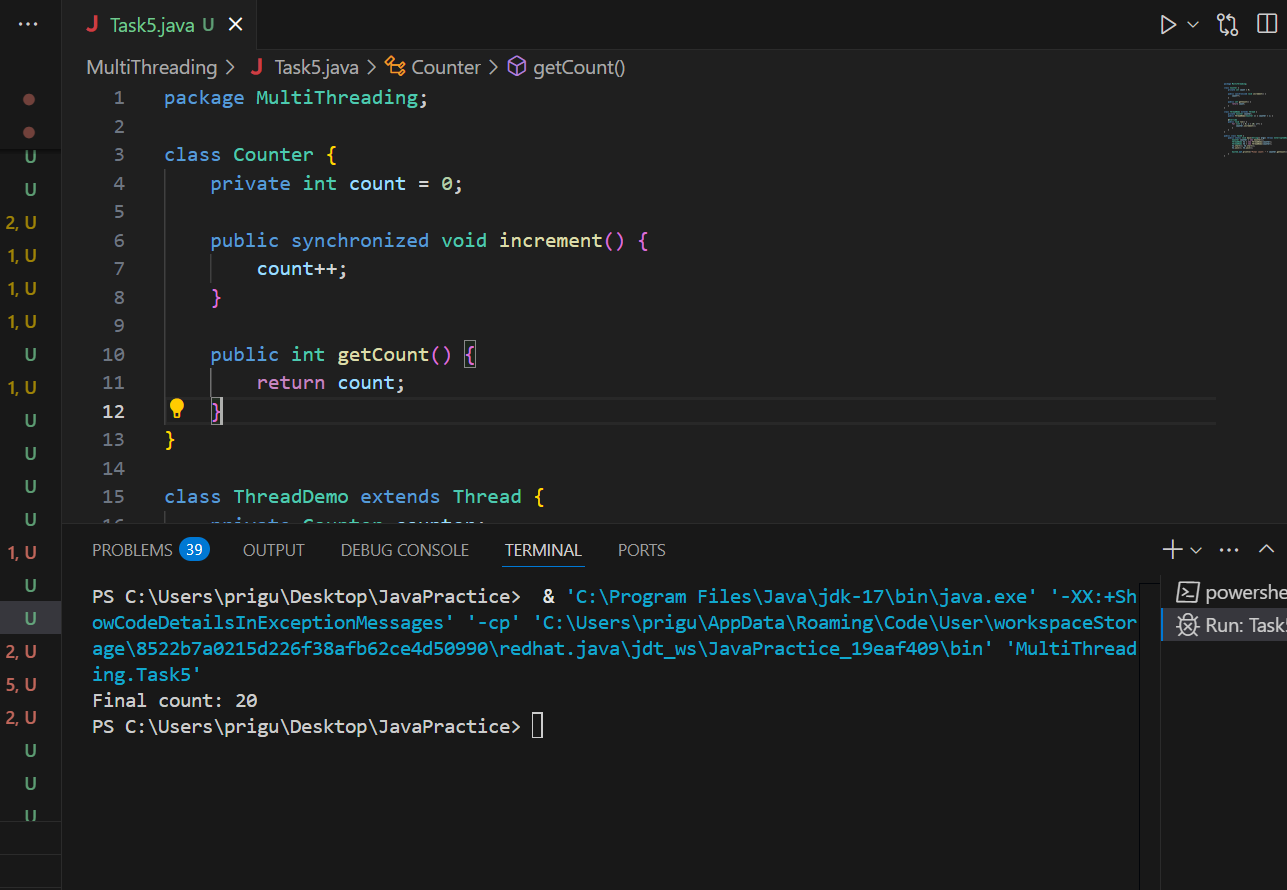
MyInterface ref;

ref = () -> 3.1415;

System.out.println("Value of Pi = " + ref.getPiValue());

}

}



Task5

package MultiThreading.Lambda;

public class Task5 {

@FunctionalInterface

interface MyInterface {

// abstract method

String reverse(String n);

}

public static void main( String[] args ) {

// declare a reference to MyInterface

// assign a lambda expression to the reference

MyInterface ref = (str) -> {

String result = "";

for (int i = str.length()-1; i >= 0 ; i--)

result += str.charAt(i);

return result;

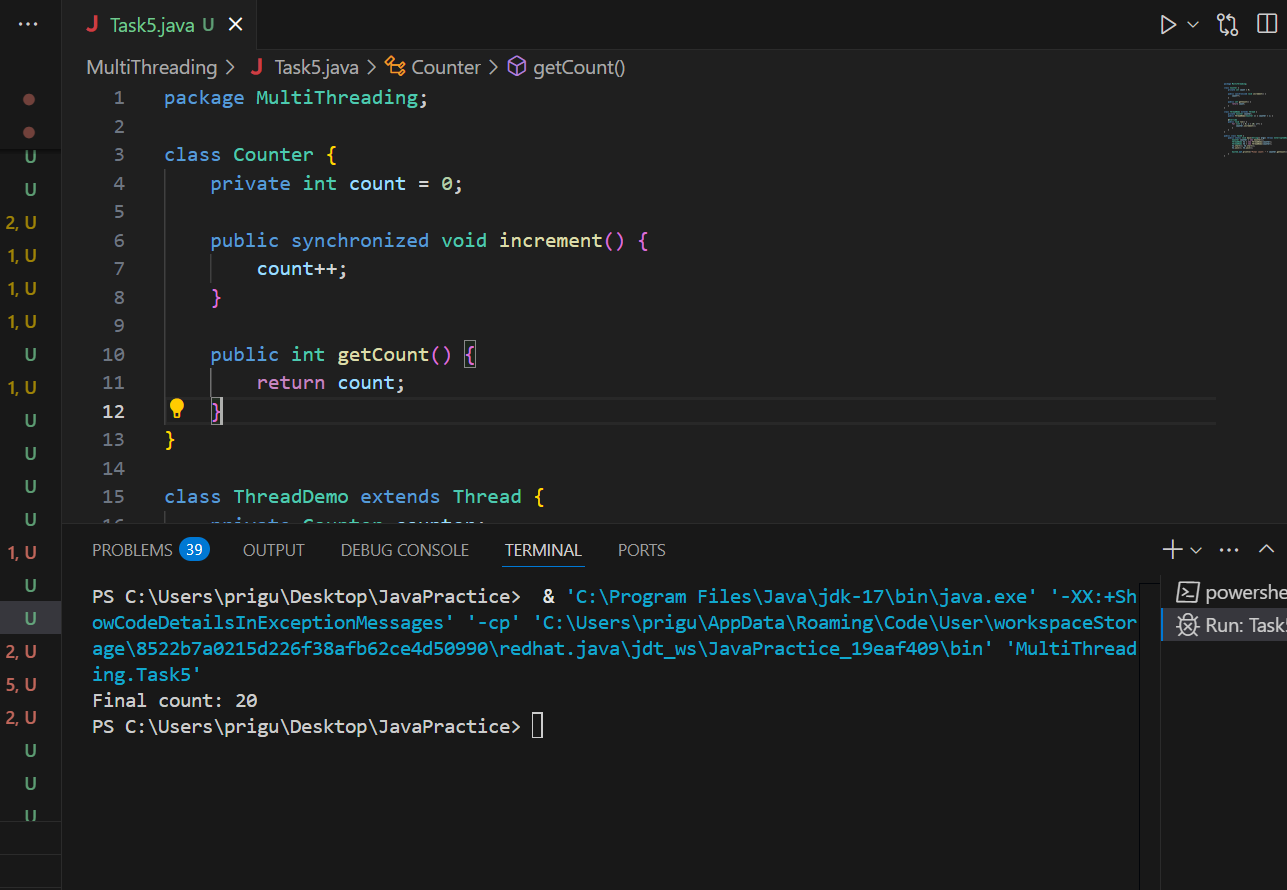
};

// call the method of the interface

System.out.println("Lambda reversed = " + ref.reverse("Lambda"));

}

}



Task 6

package MultiThreading.Lambda;

import java.util.ArrayList;

import java.util.List;

public class Task6 {

// create an object of list using ArrayList

static List<String> places = new ArrayList<>();

// preparing our data

public static List getPlaces(){

// add places and country to the list

places.add("Nepal, Kathmandu");

places.add("Nepal, Pokhara");

places.add("India, Delhi");

places.add("USA, New York");

places.add("Africa, Nigeria");

return places;

}

public static void main( String[] args ) {

List<String> myPlaces = getPlaces();

System.out.println("Places from Nepal:");

// Filter places from Nepal

myPlaces.stream()

.filter((p) -> p.startsWith("Nepal"))

.map((p) -> p.toUpperCase())

.sorted()

.forEach((p) -> System.out.println(p));

}

}



Task 7

package MultiThreading.Lambda;

// wap to create an arraylist and square them and give the output

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Task7 {

public static void main(String[] args) {

// Step 1: Create an ArrayList of numbers

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Step 2: Create another list to store squares

List<Integer> squareOfNums = numbers.stream()

.map(num -> num \* num)

.collect(Collectors.toList());

// Step 4: Print the squared numbers

System.out.println("Squared Numbers: " + squareOfNums);

}

}



Task 8

What do you understand by map()?

Map () takes each item from the list, do something to it, and return the new version and the original one remains unchanged.

it creates a **new stream** with transformed values.

Example : Squaring

Task 9

package MultiThreading.Lambda;

// wap to filter odd numbers

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Task9 {

public static void main(String[] args) {

// Step 1: Create an ArrayList of numbers

List<Integer> numbers = Arrays.asList(11, 22, 35, 45, 55);

//and now filter the odd elements

List<Integer> addNumbers = numbers.stream()

.filter(num -> num % 2 !=0)

.collect(Collectors.toList());

//// Step 4: Print the odd numbers

System.out.println(" Odd numbers are : "+ addNumbers);

}

}

Task 10 :

Filter() : Keep only certain elements

It Takes each element. then applies a condition (predicate) keeps only those that match the conditions and returns a new stream of fewer or equal elements.

Example : Odd numbers , Even Numbers

Task 11 🎉

package MultiThreading.Lambda;

//Wap to create an array list to remove duplicate values from the List.

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

public class Task11 {

public static void main(String[] args) {

// Step 1: Create an ArrayList of numbers

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(15);

numbers.add(15);

numbers.add(37);

numbers.add (37);

//and now filter the duplicate elements

List<Integer> RemovDups= numbers.stream()

.distinct()

.collect(Collectors.toList());

System.out.println(" Duplicate values removed : "+ RemovDups);

}

}

Task 12

What do you understand by distinct()?

The distinct() method is used to remove duplicate elements from a stream.

Task 13

package MultiThreading.Lambda;

//Task 13:

//Wap to create an arrayList of your friends using string and try to sort them and display

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Task13 {

public static void main(String[] args) {

// Step 1: Create an ArrayList of numbers

List<String> friends = Arrays.asList("Ananya", "Shanaya", "Manya", "Tanya", "Waanya");

List<String> SortedNames= friends.stream()

.sorted()

.collect(Collectors.toList());

System.out.println("Name of friends : "+ SortedNames);

}

}

Task 14 👍

package MultiThreading.Lambda;

import java.util.stream.Stream;

public class Tasks14 {

public static void main(String[] args) {

Stream<Integer> nums = Stream

.iterate(1, n -> n + 1) // Start from 1, keep adding 1

.limit(20); // Generate first 20 numbers

// Display only the first 10 numbers using forEach

nums

.limit(10) // Limit display to 10

.forEach(System.out::println); // Print each number

}

}

Task 15 👍

package MultiThreading.Lambda;

import java.util.stream.Stream;

public class Task15 {

public static void main(String[] args) {

Stream<Integer> nums = Stream

.iterate(1, n -> n+1)

.limit(20);

Stream<Integer> SkipNums = nums.skip(15);

SkipNums.forEach(System.out::println);

}

}

Task 16

Explain limit and skip methods..

Limit(n) : Returns a stream consisting of the first n elements of the stream. “ take the first n elements”

skip (n) : Returns the stream that skips the first n elements and processes the rest.

“ Ignore the first n elements.

Reduce()   
 used **to combine all elements of a stream into a single result (like sum, product, concatenation, etc.)** by repeatedly applying a combining function.

It is a terminal operation and is immutable

Task 17

Difference in mutable and immutable

### **Immutable = Cannot Change After Creation**

* Once created, the **object’s content cannot be modified**.
* Any change creates a **new object**.

### **Mutable = Can Be Changed**

* The object's content **can be changed without creating a new object**.

package MultiThreading.Lambda;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

public class Task17 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

Optional<Integer> sum = numbers.stream().reduce((x, y) -> x + y);

System.out.println("Sum of all elements: " + sum.orElse(0));

Optional<Integer> max = numbers.stream().reduce(Integer::max);

System.out.println("Maximum element: " + max.orElse(0));

List<String> strings = Arrays.asList("Hello", " ", "world", "!");

Optional<String> concatenatedString = strings.stream().reduce((x, y) -> x + y);

System.out.println("Concatenated string: " + concatenatedString.orElse(""));

}

}

Task 18

1. **Cleaner & Shorter Code**

**2. Efficient with Large Data (Lazy Evaluation)**

**3.Parallel Processing**

**4.Chaining Operations**

**5. Built-in Functional Methods**

Task 18

What are the debugging tools in Java.. list down a few..

IntelliJ IDEA Debugger

Eclipse Debugger

NetBeans Debugger

jdb (Java Debugger)

System.out.println()

Java Logger (java.util.logging)

Task 19:

Error Messages in Java

Compile time and run time

Compile time : grammatical mistakes … ;, {} , missing the code

Run time error or exceptions

Stack overflow error

Array index out of bounds

IO exception

Nulpointer exception

Task 20

Stack trace.. What will it do?

A stack trace is a report that shows the path your program took through the code before an error (exception) occurred.

It helps you debug by showing:

* 🔹 The error type
* 🔹 The exact line where the error happened
* 🔹 The method calls that led to it