Day 11 - 19th June 2025

Name: Aravind Kasanagottu

ID: MVSNARAV

Java 8 Features

👍

Streams

Lambda Exp

Functional Interfaces

Concrete methods….we can body of the methods

Default methods

Static methods

Collection API

Fork join methods

Method ref - class name :: method name

Comparable and comparators

Abstract and private methods

NIO

=====================

C++ – pure virtual functions

Void display() = 0;

=========================

Extends Class Name

Implement Interface Name

STreams —-- stream of data..

Task 1:

What are streams?

Streams in Java are used to read data from sources like files or write data to them. They help in handling input and output in a simple and continuous flow.

Task 2:

Write about :

Boilerplate code:

Boilerplate code is the standard, repetitive code that is required to set up a basic structure in programming, even if it doesn’t do much. In Java, things like the main method and class declarations are common boilerplate.

**Lack of Parallelism:**

It means a program or system does not perform multiple tasks simultaneously, which can lead to slower performance and inefficient use of resources.

**Lack of Composition**

It refers to the absence of combining smaller, reusable components to build more complex systems, making the code less modular and harder to maintain.

Task 3:

List of Intermediate and terminal operations

Intermediate Operations:

|  |  |
| --- | --- |
| filter() | Filters elements based on condition |
| map() | Transforms each element |
| flatMap() | Flattens nested structures |
| distinct() | Removes duplicates |
| sorted() | Sorts elements |
| limit(n) | Limits the number of elements |
| skip(n) | Skips the first n elements |
| peek() | Performs action without changing the stream (mainly for debugging) |
| **Terminal Operations:**   |  |  | | --- | --- | | forEach() | Performs action for each element | | collect() | Converts to list, set, map, etc. | | reduce() | Reduces elements to a single value | | count() | Counts number of elements | | min() / max() | Finds minimum or maximum | | anyMatch() | Returns true if any element matches | | allMatch() | Returns true if all elements match | | noneMatch() | Returns true if no elements match | | findFirst() | Returns first element (if present) | | findAny() | Returns any one element (useful in parallel streams) | |  |

========================================================================

Lambda Expressions

========================================================================

Task 4:

import java.lang.FunctionalInterface;

// this is functional interface

@FunctionalInterface

interface MyInterface{

    // abstract method

    double getPiValue();

}

public class Main {

    public static void main( String[] args ) {

    // declare a reference to MyInterface

    MyInterface ref;

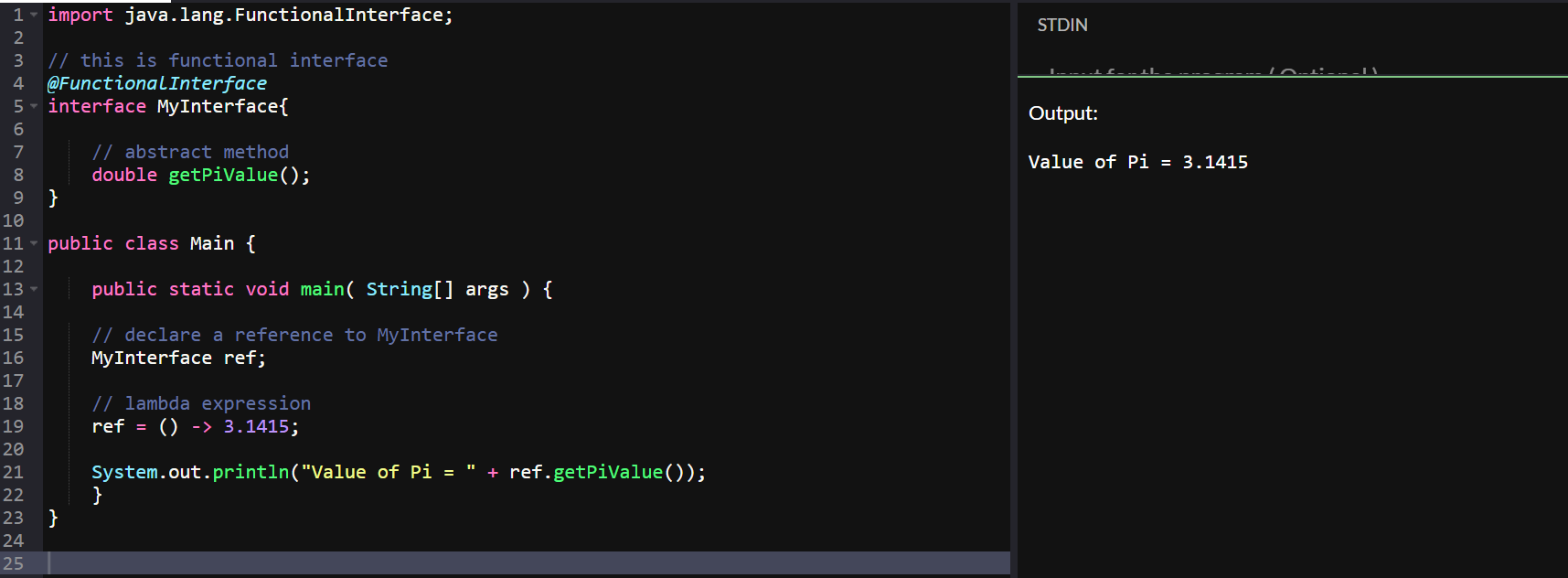
    // lambda expression

    ref = () -> 3.1415;

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

    }

}



Task 5:

@FunctionalInterface

interface MyInterface {

    // abstract method

    String reverse(String n);

}

public class Main {

    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;

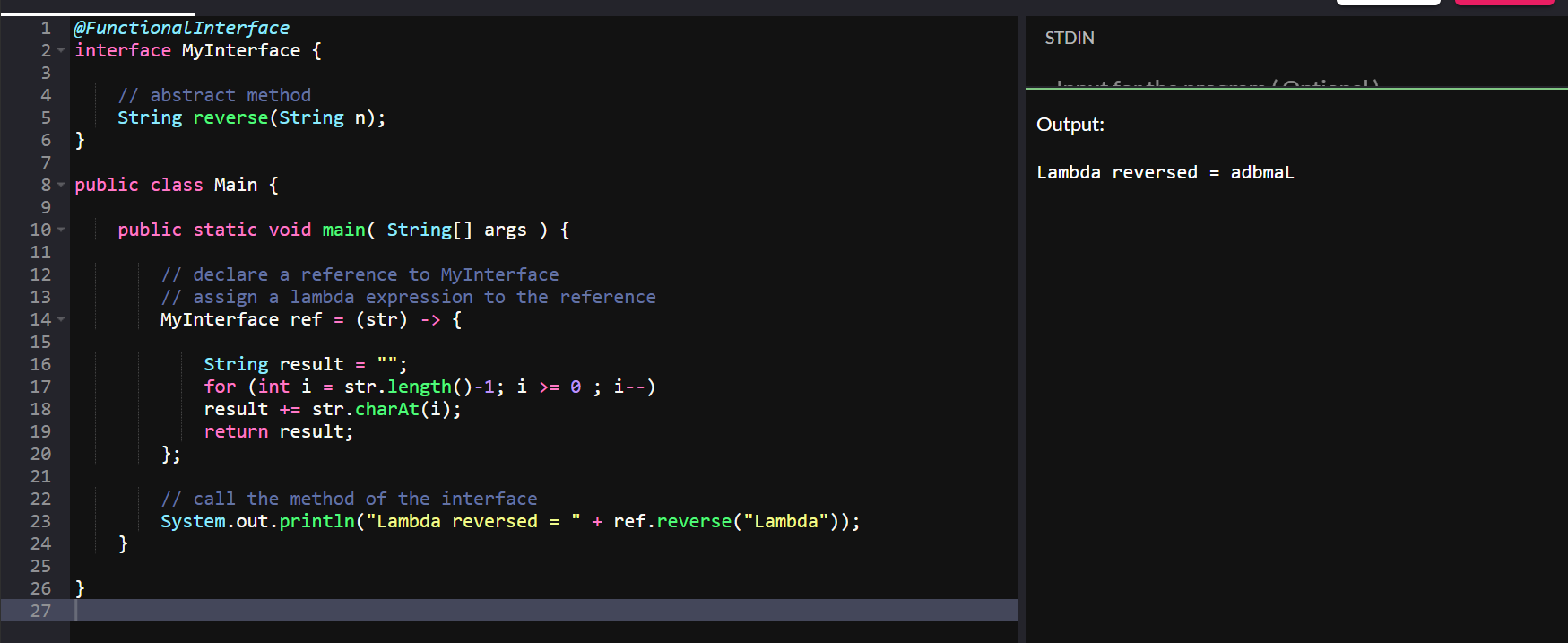
        };

        // call the method of the interface

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

    }

}



==================================

Task 6:

import java.util.ArrayList;

import java.util.List;

public class StreamMain {

    // 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"))

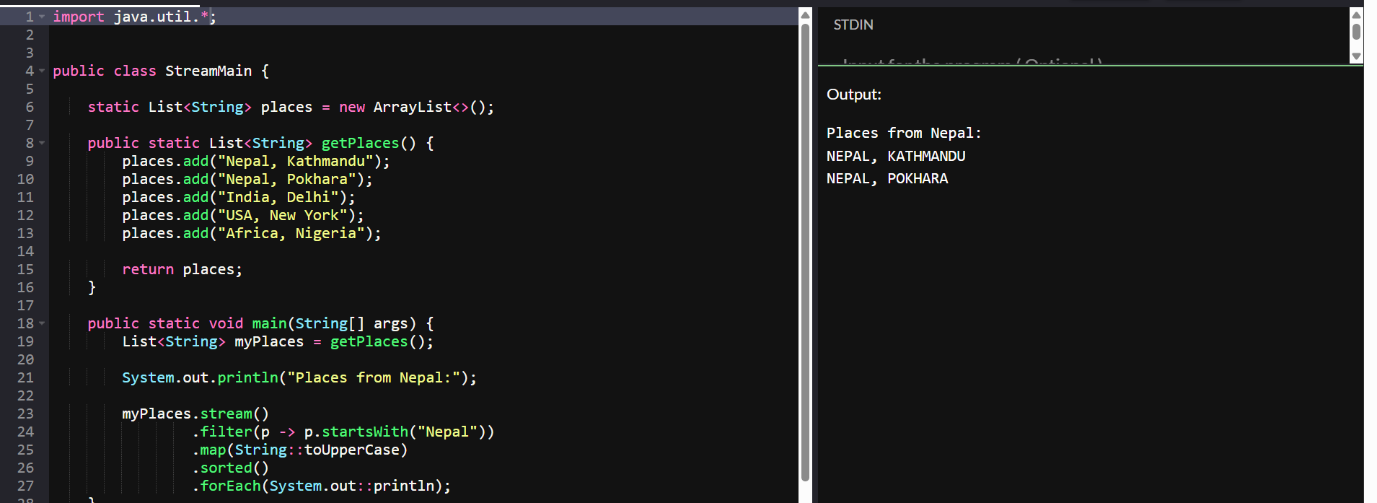
                .map((p) -> p.toUpperCase())

                .sorted()

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

    }

}



Task 7:

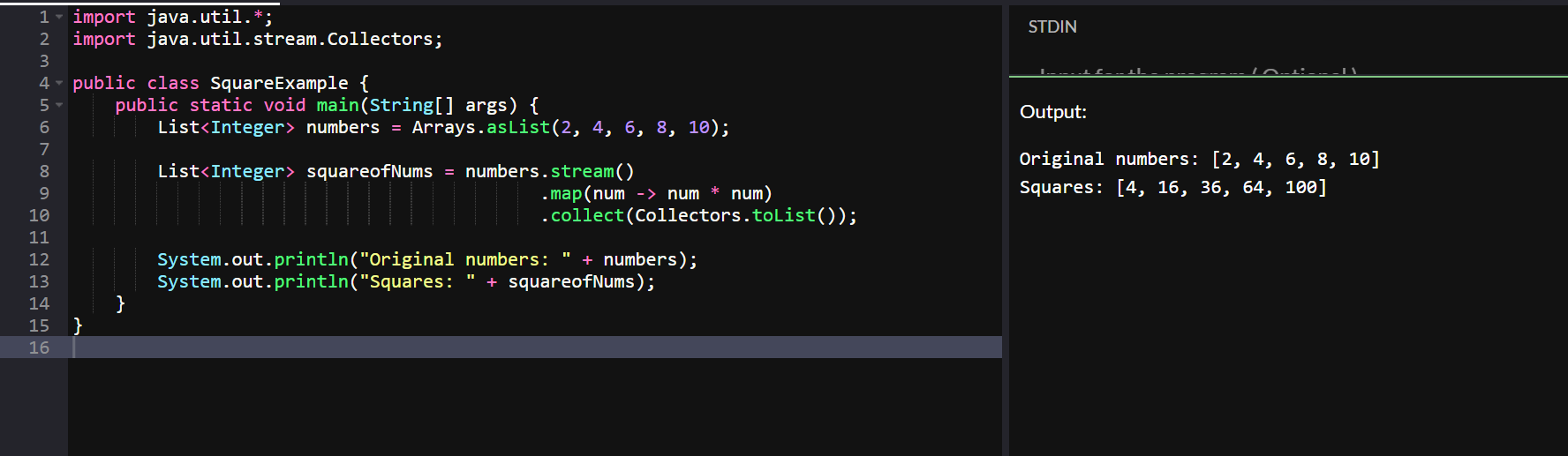
Write a code to create a array list to store 5 integers and display the square of each no..

Hint:

List<Integer> squareofNums = numbers.stream()

.map(num->num\*num)

.collect(Collectors.toList());



Task 8:

What do you understand by map()?

The map() function in Java is used to **transform each element** of a stream by applying a function to it and returning a new stream with the modified elements.

Task 9:

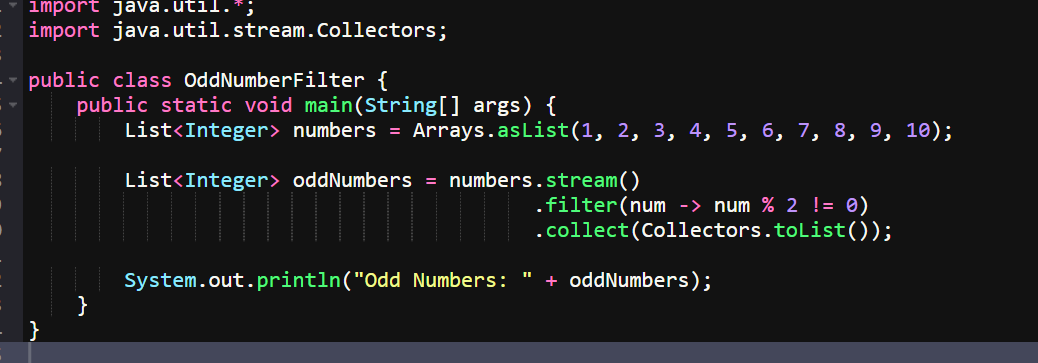
Write a code to create an array list and filter the values which are odd numbers and display them..

Hint:

List<Integer> addNumbers = numbers.stream()

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

.collect(Collectors.toList());



Task 10:

What do you understand by filter()?

The filter() method in Java is used to **select elements** from a stream that **match a given condition,** returning only those elements.

12.16 to 12.18

Task 11:

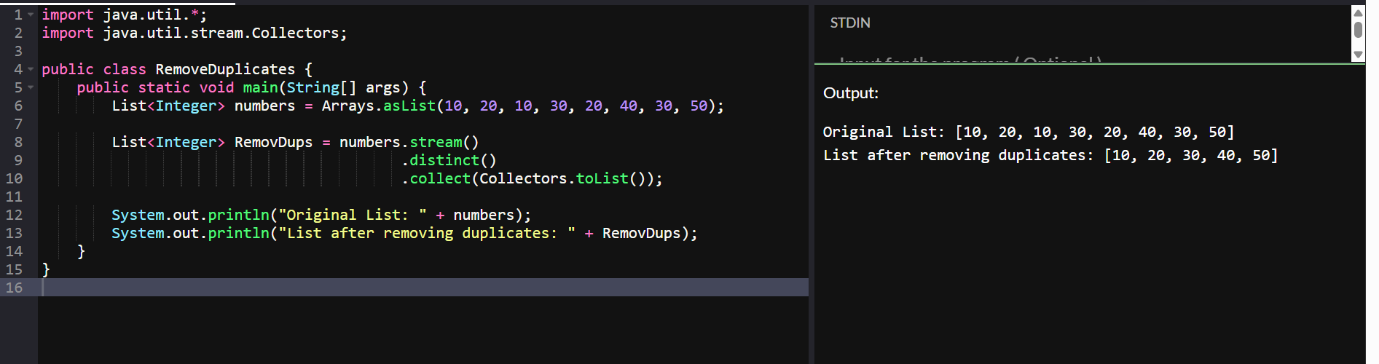
Wap to create an array list to remove duplicate values from the List.

Hint:

List<Integer> RemovDups= numbers.stream()

.distinct()

.collect(Collectors.toList());



Task12:

What do you understand by distinct()?

The distinct() method in Java is used to **remove duplicate elements** from a stream and return a stream with **only unique values.**

Sort

Limit

Skip

Terminal op

Foreach

Collect

redure

Task 13:

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

Hint:

List<String> SortedNames= Names.stream()

.sorted()

.collect(Collectors.toList());



Task 14:

Wap to run a loop / iterate()  and limit it to 20 values (1 to 2)

While displaying use for each to limit till 10 numbers.

Hint:

Stream<Integers> nums = Stream

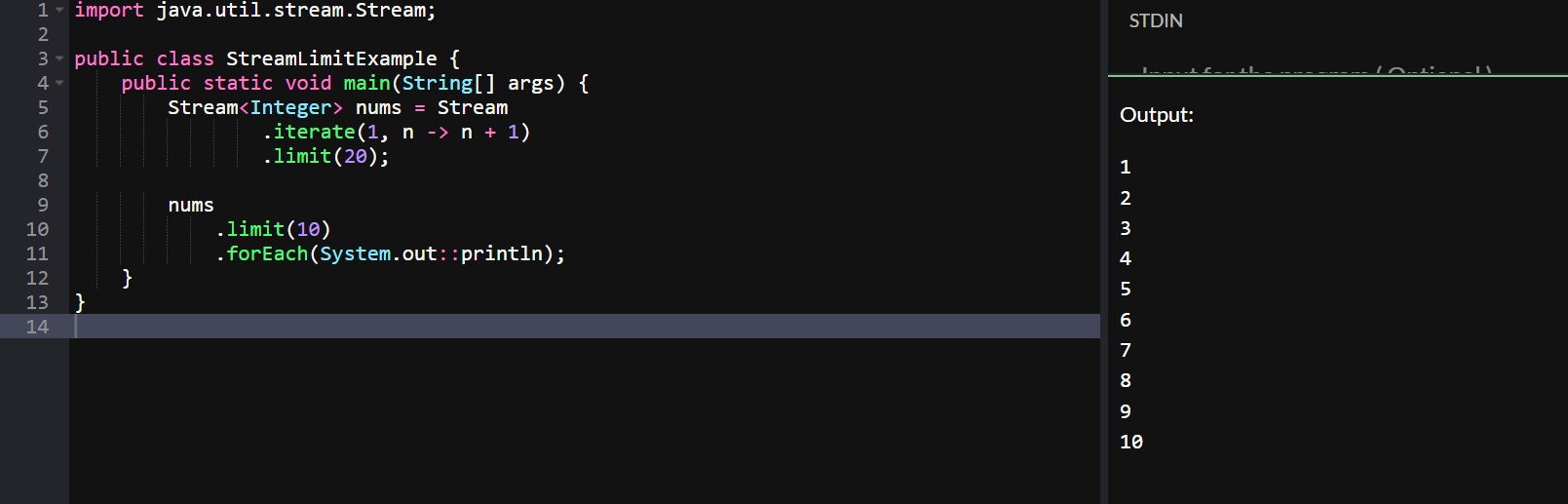
.iterate(1, n -> n+1)

.limit(20);

Nums

.limit(10)

.foreach(System.out::println);



Task 15: (similar to Task 14)

Wap to create an array List skip 15 numbers and print the output using foreach loop

HInt:

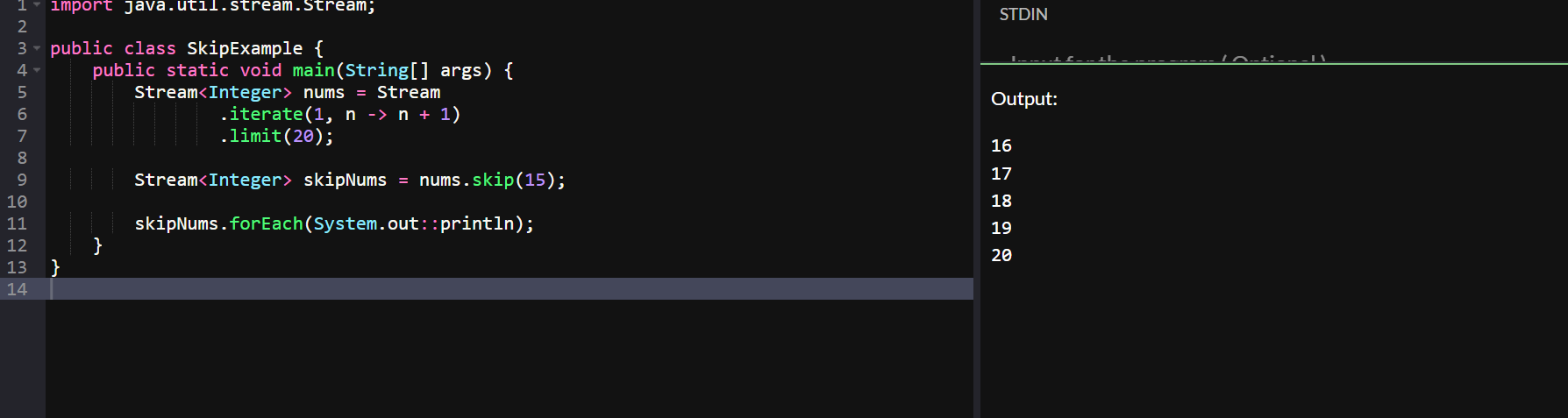
Stream<Integers> nums = Stream

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

.limit(20);

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

Nums.foreach(System.out::println);



Task 16:

Explain limit and skip methods..

The limit(n) method in Java Streams is used to restrict the number ofelements in a stream to the first n elements.

Example:

java

CopyEdit

Stream.of(1, 2, 3, 4, 5).limit(3)

// Output: 1, 2, 3

Task 17:

What is the difference between mutable and immutable?

 **Mutable** → Changeable

 **Immutable** → Unchangeable

Mutable ⇒ changeable

Int

Collect ()

Immutable ⇒ cannot be changed

Wrapper classes–  Integer, Long ,

reduce()

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

public class ReduceExample {

    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(""));

    }

}

When to use reduce and when to use collect..

Reduce will be used if you are expecting a single result from the stream (eg min, max , sum, product…)

Collect will be used if you are excepting a list of values… (list, set, map)

Task 18:

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

System.out.println() statements

Debugging with an IDE

Debugging with the command line

Debugging with logging frameworksUsing third-party tools

3 min  16.04 to 16.07

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?

Identify the error

Locate the code

Analyze the code

Solution also

=======================================================================

Task 19:

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

public class Example {

    private static final Logger logger = LogManager.getLogger(Example.class);

    public static void main(String[] args) {

        int x = 5;

        int y = 7;

        int sum = x + y;

        logger.debug("x = " + x);

        logger.debug("y = " + y);

        logger.debug("sum = " + sum);

    }

}

============================================================

Advantages of Streams:

========================================================================

**Info Box**

Data structures -  a few codes - for reference

<https://drive.google.com/drive/folders/1OXACrFwF5hQ2WVSHh2gP1EZDF_PcEy_U?usp=sharing>

========================================================================