**PROGRAM - 1**

import java.util.Scanner;

/\*\*

 \* The ArraysTest class demonstrates various sorting algorithms on arrays.

 \*/

public class ArraysTest {

    /\*\*

     \* The main method is the entry point of the program.

     \*

     \* @param *args* The command-line arguments.

     \*/

    public static *void* main(*String*[] *args*) {

*int*[] intArray = new *int*[5];

*Scanner* sc = new Scanner(System.in);

*String* a = sc.nextLine();

        System.out.println(a);

    }

    /\*\*

     \* The bubbleSortArrays method sorts an array using the bubble sort algorithm.

     \*/

    private static *void* bubbleSortArrays() {

*int*[] intArray = { 20, 14, 6, -11, 15, 58, 35, -22 };

        for (*int* i = 0; i < intArray.length - 1; i++) {

            for (*int* j = 0; j < intArray.length - 1; j++) {

                if (intArray[j] > intArray[j + 1]) {

*int* temp = intArray[j];

                    intArray[j] = intArray[j + 1];

                    intArray[j + 1] = temp;

                }

            }

        }

        printArray(intArray);

    }

    /\*\*

     \* The selectionSort method sorts an array using the selection sort algorithm.

     \*/

    private static *void* selectionSort() {

*int*[] arr = { 20, 14, 6, -11, 15, 58, 35, -22 };

        for (*int* i = 0; i < arr.length; i++) {

*int* min\_idx = i;

            for (*int* j = i + 1; j < arr.length; j++) {

                if (arr[j] < arr[min\_idx])

                    min\_idx = j;

            }

*int* temp = arr[min\_idx];

            arr[min\_idx] = arr[i];

            arr[i] = temp;

            printArray(arr);

            System.out.println("Each ==");

        }

        printArray(arr);

    }

    /\*\*

     \* The insertionSortArrays method sorts an array using the insertion sort algorithm.

     \*/

    private static *void* insertionSortArrays() {

*int*[] arr = { 20, 14, 6, -11, 15, 58, 35, -22 };

        for (*int* i = 1; i < arr.length; i++) {

*int* key = arr[i];

*int* j = i - 1;

            for (; j >= 0 && arr[j] > key; j--) {

                arr[j + 1] = arr[j];

                System.out.println("Each === ");

                printArray(arr);

            }

            System.out.println(" \n J value " + j);

            arr[j + 1] = key;

        }

        printArray(arr);

    }

    /\*\*

     \* The printArray method prints the elements of an array.

     \*

     \* @param *array* The array to be printed.

     \*/

    private static *void* printArray(*int*[] *array*) {

        for (*int* i = 0; i < array.length; i++)

            System.out.print(array[i] + " ");

    }

}

**PROGRAM - 2**

/\*\*

\* This class demonstrates how to move all the zeros in an array to the left side.

\*/

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class ArrayZerosToLeft {

public static void main(String[] args) {

// TODO Auto-generated method stub

Integer array[] = {1,1,0,1,0};

List<Integer> collect = Arrays.stream(array).sorted().collect(Collectors.toList());

System.out.println("Method One ====== "+collect);

int left = 0;

for(int i=0; i< array.length; i++)

{

if(array[i]== 0)

{

int temp = array[i];

array[i] = array[left];

array[left] = temp;

left++;

}

}

System.out.println("Method Two ====== "+Arrays.toString(array));

}

}

**PROGRAM - 3**

/\*\*

 \* This class represents a custom class that contains a main method.

 \* It demonstrates various operations using Java streams and functional programming concepts.

 \*/

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

import java.util.function.Predicate;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

public class CustomClass {

public static void main(String[] args) {

List<Course> courses = List.of(

new Course("Spring", "Framework", 98, 20000),

new Course("Spring Boot", "Framework", 95, 18000),

new Course("API", "Framework", 96, 20000),

new Course("Microservices", "Framework", 91, 25000),

new Course("FullStack", "FullStack", 96, 22000),

new Course("AWS", "Cloud", 99, 19000),

new Course("Azure", "Cloud", 97, 20000),

new Course("Docker", "Cloud", 91, 21000),

new Course("Kubernates", "Cloud", 92, 20000));

Predicate<Course> reviewScoreGretherThan95Predicate = course -> course.getReviewScore()>95;

Predicate<Course> reviewScoreGretherThan90Predicate = course -> course.getReviewScore()>90;

// System.out.println(courses.stream().allMatch(reviewScoreGretherThan95Predicate));

// System.out.println(courses.stream().noneMatch(reviewScoreGretherThan95Predicate));

// Comparator<Course> comparingByNoOfStudentsAndReviewScore = Comparator.comparing(Course :: getNoOfStudents).thenComparing(Course::getReviewScore).reversed();

// System.out.println(courses.stream().sorted(comparingByNoOfStudentsAndReviewScore).collect(Collectors.toList()));

// System.out.println(

// courses.stream().collect(Collectors.groupingBy(Course :: getCategory)));

//

// System.out.println(

// courses.stream().collect(Collectors.groupingBy(Course :: getCategory, Collectors.maxBy(Comparator.comparing(Course::getReviewScore)) )));

//

// System.out.println(

// courses.stream().collect(Collectors.groupingBy(Course :: getCategory, Collectors.mapping(Course::getName, Collectors.toList()) )));

int[] intArray = {15,13,14,17,9,10,5,7,2,18};

System.out.println(Arrays.stream(intArray).average());

System.out.println(IntStream.of(15,13,14,17,9,10,5,7,2,18).min());

System.out.println(IntStream.range(1, 10));

IntStream.rangeClosed(1, 10).forEach(System.out::print);

}

}

class Course {

private String name;

private String category;

private int reviewScore;

private int noOfStudents;

public Course(String name, String category, int reviewScore, int noOfStudents) {

super();

this.name = name;

this.category = category;

this.reviewScore = reviewScore;

this.noOfStudents = noOfStudents;

}

@Override

public String toString() {

return "Course [name=" + name + ", category=" + category + ", reviewScore=" + reviewScore + ", noOfStudents="

+ noOfStudents + "]";

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

public int getReviewScore() {

return reviewScore;

}

public void setReviewScore(int reviewScore) {

this.reviewScore = reviewScore;

}

public int getNoOfStudents() {

return noOfStudents;

}

public void setNoOfStudents(int noOfStudents) {

this.noOfStudents = noOfStudents;

}

}

**PROGRAM - 4**

/\*\*

 \* The CustomPrefquency class calculates the minimum capacity required to process a given list of integers.

 \* It finds all possible combinations of the integers in the list and calculates their sums.

 \* The minimum capacity is the largest sum that is less than or equal to the given process capacity.

 \*/

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Iterator;

import java.util.List;

import java.util.stream.Collectors;

public class CustomPrefquency {

public static void main(String[] args) {

List<Integer> inputList = Arrays.asList(2,-7,9,8);

// inputList.stream().collect(Collectors.toCollection(ArrayDeque :: new)).descendingIterator();

int processCapacity =11;

int minCapacity = minimumCapacity(inputList, processCapacity);

System.out.println(minCapacity);

}

private static int minimumCapacity(List<Integer> inputList, int processCapacity) {

int value =0;

List<List<Integer>> subSetList = findCombinationList(inputList);

System.out.println(subSetList);

System.out.println("After size == "+subSetList.size());

List<Integer> sumOfSubSet = new ArrayList<>();

for(List<Integer> list : subSetList)

{

int val = list.stream().reduce(0, Integer::sum);

if(val == processCapacity)

return val;

if(val <= processCapacity)

sumOfSubSet.add(val);

}

sumOfSubSet= sumOfSubSet.stream().distinct().sorted(Comparator.reverseOrder()).collect(Collectors.toList());

value = sumOfSubSet.get(0);

System.out.println(sumOfSubSet);

return value;

}

private static List<List<Integer>> findCombinationList(List<Integer> inputList) {

// TODO Auto-generated method stub

List<List<Integer>> result = new ArrayList<>();

findCombinationListHelper(inputList, 0, new ArrayList<>(), result);

// List<List<Integer>> result = findCombinationList2(inputList);

System.out.println("result = "+ result);

System.out.println("Size = "+ result.size());

System.out.println("Size = "+ result.stream().distinct().count());

return result;

}

private static void findCombinationListHelper(List<Integer> inputList, int startIndex, List<Integer> combinationList,

List<List<Integer>> result) {

System.out.println("findCombinationListHelper combinationList line 53 Before = "+ combinationList);

result.add(new ArrayList<>(combinationList));

System.out.println("findCombinationListHelper result line 53 After= "+ result);

for(int i = startIndex; i < inputList.size(); i++)

{

System.out.println("findCombinationListHelper inside loop startIndex = "+ startIndex);

combinationList.add(inputList.get(i));

System.out.println("findCombinationListHelper inside loop combinationList = "+ combinationList);

findCombinationListHelper(inputList, i+1, combinationList, result);

System.out.println("findCombinationListHelper inside loop before removing combinationList = "+ combinationList);

combinationList.remove(combinationList.size()-1);

System.out.println("findCombinationListHelper inside loop after removing combinationList = "+ combinationList);

}

}

public static List<List<Integer>> findCombinations(List<Integer> inputList) {

List<List<Integer>> result = new ArrayList<>();

int size = inputList.size();

for (int i = 0; i < size; i++) {

List<Integer> combination = new ArrayList<>();

for (int j = i; j < size; j++) {

combination.add(inputList.get(j));

result.add(new ArrayList<>(combination));

}

}

return result;

}

public static List<List<Integer>> findCombinationList2(List<Integer> inputList) {

List<List<Integer>> result = new ArrayList<>();

int size = inputList.size();

for (int i = 0; i < size; i++) {

List<Integer> combinationList = new ArrayList<>();

Iterator<Integer> iterator = inputList.listIterator(i);

while (iterator.hasNext()) {

combinationList.add(iterator.next());

result.add(new ArrayList<>(combinationList));

}

}

return result;

}

}

**PROGRAM - 5**

import java.util.Arrays;

import java.util.Map;

import java.util.stream.Collectors;

/\*\*

\* This class is used to find duplicate characters in a given string.

\*/

public class FindDuplicateChars {

public static void main(String[] args) {

String str = "The text need to be checked for duplication";

findDuplicateCharacter(str);

}

/\*\*

\* Finds duplicate characters in the given input string and prints them.

\*

\* @param input the string to be checked for duplicate characters

\*/

private static void findDuplicateCharacter(String input) {

Map<String, Long> collect = Arrays.stream(input.split("")).filter(i -> !i.isBlank())

.collect(Collectors.groupingBy(ch -> ch, Collectors.counting()))

.entrySet().stream().filter(i -> (i.getValue() > 1))

.collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));

System.out.println(collect);

}

}

**PROGRAM - 6**

import java.util.Arrays;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.stream.Collectors;

/\*\*

\* This class finds the first non-repeating element in a given string.

\*/

public class FindFirstNonRepeatElement {

public static void main(String[] args) {

String input = "iloveprogrammsng";

findFirstNonRepeatElement(input);

}

/\*\*

\* Finds the first non-repeating element in the given string.

\*

\* @param input the input string

\*/

private static void findFirstNonRepeatElement(String input) {

String string = Arrays.stream(input.split(""))

.collect(Collectors.groupingBy(ch -> ch, LinkedHashMap::new, Collectors.counting()))

.entrySet().stream()

.filter(i -> i.getValue() == 1)

.map(Map.Entry::getKey)

.findFirst()

.get();

System.out.println(string);

}

}

**PROGRAM - 7**

import java.util.Arrays;

import java.util.concurrent.CompletableFuture;

import java.util.concurrent.ExecutionException;

/\*\*

 \* This class finds the longest string from an array of strings and performs other operations.

 \*/

public class FindLongestString {

    public static *void* main(*String*[] *args*) throws *InterruptedException*, *ExecutionException* {

        // TODO Auto-generated method stub

*String* [] arrayInput = {"vivek"  , "kadiyan" , "ram" , "chaudhary vivek kadiyan"};

*String* string = Arrays.stream(arrayInput).reduce((word1, word2)*->* word1.length() > word2.length() ? word1: word2).get();

        System.out.println(string);

*int*[] intArray = {5,6,3,7,3,1,9,8,2,4};

*int* asInt = Arrays.stream(intArray).reduce((v1,v2)*->* v1 > v2 ? v1: v2).getAsInt();

        System.out.println(asInt);

*String* delimiter = "-";

*String* result = String.join(delimiter, "Edpresso", "is", "Good");

            System.out.println(result);

*CompletableFuture*<*String*> helloFuture = CompletableFuture.supplyAsync(() *->* "Hello");

*CompletableFuture*<*String*> greetingFuture = CompletableFuture.supplyAsync(() *->* "World");

*CompletableFuture*<*String*> combinedFuture = helloFuture.thenCombine(greetingFuture, (m1,m2) *->* m2 +" " +m1);

//          CompletableFuture<String> combinedFuture = helloFuture.thenCompose(null)

            System.out.println(combinedFuture.get());

    }

}

**PROGRAM - 8**

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.stream.Collectors;

/\*\*

 \* This class provides methods to find the Nth highest salary in a given map and list of students.

 \*/

public class FindNthSalaryInGivenMap {

    /\*\*

     \* The main method of the class.

     \*

     \* @param *args* The command line arguments.

     \*/

    public static *void* main(*String*[] *args*) {

        // TODO Auto-generated method stub

*Map*<*String*, *Integer*> map = new *HashMap*<>();

        map.put("vivek", 100);

        map.put("Rinku", 400);

        map.put("vishal", 600);

        map.put("shankar", 500);

        map.put("shyam", 500);

        map.put("tinku", 300);

        System.out.println(findNthSalary(map, 2));

*List*<*Student*> newList = new *ArrayList*<>();

        newList.add(new Student("vivek", 100));

        newList.add(new Student("Rinku", 400));

        newList.add(new Student("vishal", 600));

        newList.add(new Student("shankar", 500));

        newList.add(new Student("shyam", 500));

        newList.add(new Student("tinku", 300));

        System.out.println("Value from ListOfStudent Object " + findNthSalaryList(newList, 3));

    }

    /\*\*

     \* Finds the Nth highest salary in the given map.

     \*

     \* @param *map* The map containing the salaries.

     \* @param *i*   The value of N.

     \* @return The entry containing the Nth highest salary and the list of employees with that salary.

     \*/

    private static *Entry*<*Integer*, *List*<*String*>> findNthSalary(*Map*<*String*, *Integer*> *map*, *int* *i*) {

        // TODO Auto-generated method stub

*Entry*<*Integer*, *List*<*String*>> entry = map.entrySet().stream()

                .collect(Collectors.groupingBy(e *->* e.getValue(),

                        Collectors.mapping(Map.Entry::getKey, Collectors.toList())))

                .entrySet().stream().sorted(Collections.reverseOrder(Map.Entry.comparingByKey()))

                .collect(Collectors.toList()).get(i - 1);

        System.out.println(entry);

        return entry;

    }

    /\*\*

     \* Finds the Nth highest salary in the given list of students.

     \*

     \* @param *list* The list of students.

     \* @param *i*    The value of N.

     \* @return The entry containing the Nth highest salary and the list of students with that salary.

     \*/

    private static *Entry*<*Integer*, *List*<*String*>> findNthSalaryList(*List*<*Student*> *list*, *int* *i*) {

        // TODO Auto-generated method stub

*Entry*<*Integer*, *List*<*String*>> entry = list.stream()

                .collect(Collectors.groupingBy(e *->* e.getMark(),

                        Collectors.mapping(e *->* e.getName(), Collectors.toList())))

                .entrySet().stream().sorted(Collections.reverseOrder(Map.Entry.comparingByKey()))

                .collect(Collectors.toList()).get(i - 1);

        System.out.println(entry);

        return entry;

    }

}

class Student {

    public *String* name;

    public *int* mark;

    public Student() {

        // TODO Auto-generated constructor stub

    }

    public Student(*String* *name*, *int* *mark*) {

        this.name = name;

        this.mark = mark;

    }

    public *String* getName() {

        return name;

    }

    public *void* setName(*String* *name*) {

        this.name = name;

    }

    public *int* getMark() {

        return mark;

    }

    public *void* setMark(*int* *mark*) {

        this.mark = mark;

    }

}

**PROGRAM - 9**

import java.lang.reflect.Array;

import java.util.Arrays;

import java.util.stream.Collectors;

/\*\*

\* This class finds the continuous sequences of numbers in an array of strings.

\*/

public class FindTheContinuousSequencesOfNumbers {

public static void main(String[] args) {

// TODO Auto-generated method stub

String inputArr[] = {"1", "2","3","5","9", "a", "6", "7" ,"8" ,"4" ,"@", "-5", "-7" ,"-3", "-2" ,"-1"};

int array[] = Arrays.stream(inputArr).filter( i -> {

try

{

Integer.parseInt(i);

return true;

}

catch(Exception e)

{

return false;

}

}).mapToInt(i -> Integer.valueOf(i)).toArray();

System.out.println(Arrays.toString(array));

for(int i=0; i< array.length;i++)

{

StringBuffer result = new StringBuffer("");

for(int j=i+1; j<array.length; j++)

{

int next = array[i]+1;

if(next == array[j])

{

if(result.toString().isEmpty())

{

result.append(array[i]+","+array[j]);

}

else

result.append(","+ array[j]);

i=j;

}

else

break;

}

if(!result.toString().isEmpty())

{

System.out.println(result.toString());

}

}

}

}

**PROGRAM - 10**

import java.util.List;

/\*\*

 \* This class contains methods for printing numbers in different ways and calculating the sum of even numbers in a list.

 \*/

public class LearningMethods {

    /\*\*

     \* Prints all the numbers in the given list using a structured approach.

     \*

     \* @param *numbers* the list of numbers to be printed

     \*/

    public static *void* printAllNumberInListStructured(*List*<*Integer*> *numbers*) {

        for (*int* number : numbers) {

            System.out.println(number);

        }

    }

    /\*\*

     \* Prints all the numbers in the given list using a functional approach.

     \*

     \* @param *numbers* the list of numbers to be printed

     \*/

    public static *void* printAllNumberInListFunctional(*List*<*Integer*> *numbers*) {

        numbers.stream().forEach(LearningMethods::print);

    }

    /\*\*

     \* Helper method to print a single number.

     \*

     \* @param *number* the number to be printed

     \*/

    private static *void* print(*int* *number*) {

        System.out.println(number);

    }

    /\*\*

     \* Calculates the sum of even numbers in the given list.

     \*

     \* @param *numbers* the list of numbers

     \* @return the sum of even numbers

     \*/

    public static *int* printSumOfIntegerList(*List*<*Integer*> *numbers*) {

        return numbers.stream().filter(number *->* number % 2 == 0).reduce(0, ((a, b) *->* a + b));

    }

}

**PROGRAM - 11**

import java.util.ArrayList;

import java.util.Arrays;

/\*\*

\* The LeftRotationBlockSwapAlgorithm class demonstrates how to rotate an ArrayList to the left by a given number of positions using the Block Swap Algorithm.

\*/

public class LeftRotationBlockSwapAlgorithm {

public static void main(String[] args) {

ArrayList<Integer> arrList = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5,9,7,8,6));

int d = 4;

// Rotate ArrayList to the left by d positions using Block Swap Algorithm

reverse(arrList, 0, d - 1);

reverse(arrList, d, arrList.size() - 1);

reverse(arrList, 0, arrList.size() - 1);

System.out.println(arrList); // Output: [3, 4, 5, 1, 2]

}

// Reverse the elements of the ArrayList from start to end

private static void reverse(ArrayList<Integer> arrList, int start, int end) {

while (start < end) {

int temp = arrList.get(start);

arrList.set(start, arrList.get(end));

arrList.set(end, temp);

System.out.println("Inside loop ================ start = "+start +" end = "+end+"Output =" +arrList);

start++;

end--;

}

// System.out.println("================ " +arrList);

}

}

**PROGRAM - 12**

import java.util.List;

import java.util.Map;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.IntFunction;

import java.util.function.Predicate;

import java.util.function.UnaryOperator;

import java.util.stream.Collectors;

/\*\*

 \* This class represents the main class for learning Java functional programming.

 \* It contains various examples of using streams and functional interfaces in Java.

 \*/

public class MainClassOfLearning {

    public *String* value ="ClassVariable";

    public static *void* main(*String*[] *args*) {

*List*<*Integer*> numbers = List.of(12,9,13,4,6,2,4,12,15);

*List*<*String*> courses = List.of("Spring","Spring Boot","API","Microservices","FullStack","AWS","Azure","Docker","Kubernates");

*List*<*String*> courses2 = List.of("Spring","Spring Boot","API","Microservices","FullStack","AWS","Azure","Docker","Kubernates");

*String* course = "MyLearningCourseJavaFunctionalProgram";

//      LearningMethods.printAllNumberInListStructured(List.of(12,9,13,4,6,2,4,12,15));

        // Stream Method references

//      LearningMethods.printAllNumberInListFunctional(numbers);

        //Using Streams Reduce function

//      int value = LearningMethods.printSumOfIntegerList(numbers);

//      System.out.println(value);

        // Using Distinct, Comparator and sorted in Streams

//      numbers.stream().distinct().sorted(Comparator.comparing(i -> i%2==0)).forEach(System.out::println);

        // Streams using Strings

//      IntFunction<String> mapper = i -> String.valueOf((char)i);

//      course.chars().mapToObj(mapper).forEach(System.out::println);

//      Map<String, Long> collect = course.chars().mapToObj(mapper).collect(Collectors.groupingBy(Function.identity(),Collectors.counting()));

//      System.out.println(collect);

        // Stream Function

//      Function<Integer,String> functionMap = number -> number + " Added";

//      numbers.stream().map(functionMap).forEach(System.out::println);

        // Stream Predicate

//      Predicate<Integer> evenPredicate = number -> number %2 ==0;

//      numbers.stream().filter(evenPredicate).forEach(System.out::println);

        // Stream Consumer

        // BinaryOperator

//      BinaryOperator<Integer> binaryOperator = (x,y)-> x+y;

//      System.out.println(binaryOperator.apply(10, 30));

        //UnaryOperator

//      UnaryOperator<Integer> unaryOperator = x -> x\*3;

//      System.out.println(unaryOperator.apply(15));

//      String value = "MethodVariable";

//      this.value

//      System.out.println(value);

*MainClassOfLearning* me = new MainClassOfLearning();

        me.methodValue();

        courses.stream().flatMap(c1 *->* courses2.stream().map(course2 *->* List.of(c1, course2))).collect(Collectors.toList());

//      System.out.println(courses.stream()

//          .flatMap(c1 -> courses2.stream().map(course2 -> List.of(c1, course2)))

//          .filter(list -> flatMapPredicate(list)).collect(Collectors.toList()));

//

        System.out.println(courses.stream()

                .flatMap(c1 *->* courses2.stream().map(course2 *->* List.of(c1, course2)))

                .filter(list *->* flatMapPredicate(list)).collect(Collectors.toList()));

    }

    private static *boolean* flatMapPredicate(*List*<*String*> *list*) {

        return !list.get(0).equals(list.get(1))  && list.get(0).length() == list.get(1).length() ;

    }

    private *void* methodValue()

    {

*String* value = "MethodVariable";

        System.out.println(this.value);

    }

}

**PROGRAM - 13**

/\*\*

 \* The RecursionTest class demonstrates the concept of recursion by calculating the factorial of a number.

 \*/

public class RecursionTest {

    /\*\*

     \* The main method is the entry point of the program.

     \* It calls the fact method to calculate the factorial of different numbers and prints the results.

     \*

     \* @param *args* The command-line arguments.

     \*/

    public static *void* main(*String*[] *args*) {

        System.out.println("Factorial of 3 is " + fact(3));

        System.out.println("Factorial of 4 is " + fact(4));

        System.out.println("Factorial of 5 is " + fact(5));

    }

    /\*\*

     \* The fact method calculates the factorial of a given number using recursion.

     \*

     \* @param *n* The number for which factorial needs to be calculated.

     \* @return The factorial of the given number.

     \*/

    private static *int* fact(*int* *n*) {

*int* result;

        System.out.println("Inside fact function n = " + n);

        if (n == 1)

            return 1;

        result = fact(n - 1) \* n;

        System.out.println("Inside fact function result = " + result);

        return result;

    }

}

**PROGRAM - 14**

/\*\*

 \* The StaticTest class represents a test class with static methods and variables.

 \* It demonstrates the usage of static blocks and methods in Java.

 \*/

public class StaticTest {

    public static *String* value ="TestClass";

    public static *void* name() {

        System.out.println("Inside name " + value);

        value = "Name";

        System.out.println("Inside name "+ value);

        name2();

    }

    public static *void* name2() {

        System.out.println("Inside name 2 " + value);

        value = "Name 2 method";

        System.out.println("Inside name 2 "+ value);

    }

    static

    {

        System.out.println("Inside static block  "+ value);

        value = "static Block";

        name();

    }

    public static *void* main(*String*[] *args*) {

        System.out.println("Inside name main method "+ value);

    }

}

**PROGRAM - 15**

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.stream.Collectors;

/\*\*

 \* The TestDemo1 class is a demonstration class that performs various operations on a list of fruits.

 \* It calculates the frequency of each fruit, prints the frequency map, and performs additional operations on the fruits.

 \*/

public class TestDemo1 {

    public static *void* main(*String*[] *args*) {

        // {"grapes","lime","lemon","cherry","banana","apple","watermelon","apple","watermelon","lime",

        // "lemon","watermelon","apple","watermelon","lime","grapes","lime","lemon","cherry","banana","apple"}

*List*<*String*> listFruties = Arrays.asList("grapes", "lime", "lemon", "cherry", "banana", "apple", "watermelon",

                "apple", "watermelon", "lime", "lemon", "watermelon", "apple", "watermelon", "lime", "grapes", "lime",

                "lemon", "cherry", "banana", "apple");

*Map*<*String*, *Integer*> mapValues = new *HashMap*<>();

        for(*String* str : listFruties)

        {

            mapValues.put(str, mapValues.getOrDefault(str, 0)+1);

        }

//      System.out.println(mapValues);

        for(*Entry*<*String*, *Integer*> map : mapValues.entrySet())

        {

            System.out.println(map);

        }

*String* str = "watermelon";

*List*<*Character*> collect = str.chars().mapToObj(i *->* (*char*) i).collect(Collectors.toList());

        System.out.println(collect);

*Map*<*Character*, *Long*> collect2 = listFruties.stream().map(i *->* i.chars().mapToObj(a *->* (*char*) a).collect(Collectors.toList())).flatMap(i *->* i.stream()).collect(Collectors.groupingBy(e *->* e, Collectors.counting()));

                System.out.println(collect2);

//      Map<String, Long> collect = listFruties.stream().collect(Collectors.groupingBy(e -> e, Collectors.counting()));

//      System.out.println(collect);

    }

    public static *void* subsetListCreation(*List*<*Integer*> *inputList*)

    {

    }

    public static *void* findHelper(*List*<*Integer*> *inputList*, *int* *startIndex*, *List*<*Integer*> *compinationList*, *List*<*List*<*Integer*>> *resultList*)

    {

        resultList.add(compinationList);

        for (*int* i = startIndex; i < inputList.size(); i++) {

            compinationList.add(inputList.get(i));

            findHelper(inputList, i+1, compinationList, resultList);

            compinationList.remove(compinationList.size()-1);

        }

    }

}

**PROGRAM - 16**

**PROGRAM - 17**

**PROGRAM - 18**

**PROGRAM - 19**

**PROGRAM - 20**

**PROGRAM - 21**