1. Write a Java Program to Convert a Given Number of Days in Terms of Years, Weeks & Days.

Sample Input&Output::

Enter the number of days: 756

No. of years: 2

No. of weeks: 3

No. of days: 5

Test cases:

1. 38
2. 3.6
3. 0
4. -365
5. -45

import java.util.Scanner;

public class DaysConverter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of days: ");

int days = scanner.nextInt();

int years = days / 365;

int weeks = (days % 365) / 7;

int remainingDays = (days % 365) % 7;

System.out.println("No. of years: " + years);

System.out.println("No. of weeks: " + weeks);

System.out.println("No. of days: " + remainingDays);

}

}

1. Given a date, return the corresponding day of the week for that date.

The input is given as three integers representing the day, month and year respectively.

Return the answer as one of the following values {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}.

**Example 1:**

**Input:** day = 31, month = 8, year = 2019

**Output:** "Saturday"

**Example 2:**

**Input:** day = 18, month = 7, year = 1999

**Output:** "Sunday"

**Example 3:**

**Input:** day = 15, month = 8, year = 1993

**Output:** "Sunday"

**Constraints:**

* The given dates are valid dates between the years 1971 and 2100.

import java.util.Scanner;

public class DayOfWeek {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter day: ");

int day = scanner.nextInt();

System.out.print("Enter month: ");

int month = scanner.nextInt();

System.out.print("Enter year: ");

int year = scanner.nextInt();

java.time.LocalDate dt = java.time.LocalDate.of(year, month, day);

String dayOfWeek = dt.getDayOfWeek().toString();

System.out.println("Output: " + dayOfWeek);

}

}

1. Write a program to find the number of student users in the college, get the total users, staff users details from the client. Note for every 3 staff user there is one Non teaching staff user assigned by default.

Sample Input:

Total Users: 856

Staff Users: 126

Sample Output:

Student Users: 688

Test Cases:

1. Total User: 0
2. Total User: -143
3. Total User: 1026, Staff User: 1026
4. Total User: 450, Staff User: 540
5. Total User: 600, Staff User: 450

import java.util.Scanner;

public class StudentUsers {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Total Users: ");

int totalUsers = scanner.nextInt();

System.out.print("Staff Users: ");

int staffUsers = scanner.nextInt();

int studentUsers = totalUsers - staffUsers - (staffUsers / 3);

System.out.println("Student Users: " + studentUsers);

}

}

1. Write a program to print number of factors and to print nth factor of the given number.

Sample Input:

Given Number: 100

N = 4

Sample Output:

Number of factors = 9

4th factor of 100 = 5

Test Cases:

1. Given Number = 512 , N = 6
2. Given Number = 343 , N = 7
3. Given Number = 1024 , N = 0
4. Given Number = -6561 , N = 3
5. Given Number = 0 , N = 2

import java.util.Scanner;

public class Factors {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Given Number: ");

int num = scanner.nextInt();

System.out.print("N = ");

int n = scanner.nextInt();

int count = 0;

int nthFactor = 0;

for (int i = 1; i <= num; i++) {

if (num % i == 0) {

count++;

if (count == n) {

nthFactor = i;

break;

}

}

}

System.out.println("Number of factors = " + count);

System.out.println(n + "th factor of " + num + " = " + nthFactor);

}

}

1. Write a program to print n prime numbers after nth Prime number

Sample Input:

N = 3

Sample Output:

3rd Prime number is 5

3 prime numbers after 5 are: 7, 11, 13

Test cases:

1. N = P
2. N = 0
3. N = -4
4. N = 11
5. N = 7.2

import java.util.Scanner;

public class PrimeNumbers {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("N = ");

int n = scanner.nextInt();

int num = 2;

int count = 0;

while (count < n) {

if (isPrime(num)) {

count++;

if (count == n) {

System.out.println(n + "th Prime number is " + num);

break;

}

}

num++;

}

int primeCount = 0;

num++;

System.out.print(n + " prime numbers after " + n + " are: ");

while (primeCount < n) {

if (isPrime(num)) {

System.out.print(num + ", ");

primeCount++;

}

num++;

}

}

public static boolean isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

}

1. Write a Program to create a list of all numbers in a range which are perfect squares and the sum of the digits of the number is less than 10.

Sample Input & Output:

Enter lower range: 1

Enter upper range: 40

[1, 4, 9, 16, 25, 36]

Test case:

1. Enter lower range: 50

Enter upper range: 100

1. Enter lower range: 5

Enter upper range: 8

1. Enter lower range: 10

Enter upper range: 5

1. Enter lower range: 500

Enter upper range: 500

1. Enter lower range: 0

Enter upper range: -100

import java.util.ArrayList;

import java.util.Scanner;

public class PerfectSquares {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter lower range: ");

int lower = scanner.nextInt();

System.out.print("Enter upper range: ");

int upper = scanner.nextInt();

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

for (int i = lower; i <= upper; i++) {

if (isPerfectSquare(i) && sumOfDigits(i) < 10) {

perfectSquares.add(i);

}

}

System.out.println(perfectSquares);

}

public static boolean isPerfectSquare(int num) {

int sqrt = (int) Math.sqrt(num);

return sqrt \* sqrt == num;

}

public static int sumOfDigits(int num) {

int sum = 0;

while (num > 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

}

1. Write a program to print unique permutations of a given number

Sample Input:

Given Number: 143

Sample Output:

Permutations are:

134

143

314

341

413

431

Test cases:

1. 0
2. 111
3. 505
4. -143
5. -598

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class UniquePermutations {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Given Number: ");

int number = scanner.nextInt();

Set<String> permutations = new HashSet<>();

permute("", String.valueOf(number), permutations);

System.out.println("Permutations are:");

for (String perm : permutations) {

System.out.println(perm);

}

}

public static void permute(String prefix, String remaining, Set<String> permutations) {

int n = remaining.length();

if (n == 0) {

permutations.add(prefix);

} else {

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

permute(prefix + remaining.charAt(i), remaining.substring(0, i) + remaining.substring(i + 1, n), permutations);

}

}

}

}

1. Write a Program to create an array with the First Element as the Number and Second Element as the Square of the Number.

Sample Input:

Enter the lower range:45

Enter the upper range:49

Sample Output:

[(45, 2025), (46, 2116), (47, 2209), (48, 2304), (49, 2401)]

Test case:

1. Enter lower range: 50

Enter upper range: 100

1. Enter lower range: 5

Enter upper range: 8

1. Enter lower range: 10

Enter upper range: 5

1. Enter lower range: 500

Enter upper range: 500

1. Enter lower range: 0

Enter upper range: -100

import java.util.ArrayList;

import java.util.Scanner;

public class NumberAndSquare {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the lower range: ");

int lower = scanner.nextInt();

System.out.print("Enter the upper range: ");

int upper = scanner.nextInt();

ArrayList<String> result = new ArrayList<>();

for (int i = lower; i <= upper; i++) {

result.add("(" + i + ", " + (i \* i) + ")");

}

System.out.println(result);

}

}

1. Develop a JAVA code to display the balance. Include the following members:

* Design a class to represent a bank account.
* **Data Members:** Name of the depositor, Account number, Type of account(Savings/Current), Balance amount in the account(Minimum balance is Rs.500.00)
* **Methods:**

1. To read account number, Depositor name, Type of account.
2. To deposit an amount (Deposited amount should be added with it)
3. To withdraw an amount after checking balance(Minimum balance must be Rs.500.00

Note : Assume that balance amount = 10000

Test Cases

1. 100, Raja, S, 8000
2. Raja, 100, S, 9000
3. 101, Rani, S, 12000
4. 102, Ragu, W, 8000
5. 103, Ravi, C, 10000

import java.util.Scanner;

class BankAccount {

String depositorName;

int accountNumber;

String accountType;

double balanceAmount;

void readAccountDetails(int accountNumber, String depositorName, String accountType) {

this.accountNumber = accountNumber;

this.depositorName = depositorName;

this.accountType = accountType;

this.balanceAmount = 500; // Assuming minimum balance is 500

}

void deposit(double amount) {

balanceAmount += amount;

System.out.println("Amount deposited successfully.");

}

void withdraw(double amount) {

if (balanceAmount - amount >= 500) {

balanceAmount -= amount;

System.out.println("Amount withdrawn successfully.");

} else {

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

}

}

void displayBalance() {

System.out.println("Balance: " + balanceAmount);

}

}

public class BankAccountDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BankAccount account = new BankAccount();

System.out.print("Enter Account Number: ");

int accNo = scanner.nextInt();

System.out.print("Enter Depositor Name: ");

String name = scanner.next();

System.out.print("Enter Account Type (Savings/Current): ");

String type = scanner.next();

account.readAccountDetails(accNo, name, type);

System.out.println("Account created successfully.");

// For testing deposit and withdrawal

account.deposit(1000);

1. Develop a code to Reverse and Add a Number until you get a Palindrome.

Sample Input If 7325 is input number, then

7325 (Input Number) + 5237 (Reverse Of Input Number) = 12562

12562 + 26521 = 39083

39083 + 38093 = 77176

77176 + 67177 = 144353

144353 + 353441 = 497794 (Palindrome)

Test Cases

1. 8765
2. -8765
3. 0
4. EIGHT FIVE
5. 87.57

import java.util.Scanner;

public class PalindromeCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

int num = scanner.nextInt();

if (isPalindrome(num)) {

System.out.println(num + " is a palindrome number.");

} else {

System.out.println(num + " is not a palindrome number.");

}

}

public static boolean isPalindrome(int num) {

int reversed = 0;

int original = num;

while (num != 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

return original == reversed;

}

}

1. Create Customer class with deposit() and withdraw() as synchronized methods. Declare AccountNo, AccName and Balance as Instance Variables inside the class. From the main class, Input the amount for withdraw() operation and if requested amount is not available in existing Balance amount, withdraw() method should be temporarily suspended using wait() method until deposit() method receives the input for amount, to be added in the existing Balance amount and then withdraw() would be completed in a successful manner. Develop the above scenario using Synchronization and Inter-Thread Communication.

Note : existing Bank balance amount 10000

Sample Input : 12000, 3000

Sample Output : Withdraw operation success, balance amount 1000

Test Cases

1. 11000, 4000
2. -10000, -2000
3. 0, 0
4. EIGHT SEVEN, FIVE
5. 100.67, 200.68

**import java.util.Scanner;**

**public class FactorialRecursive {**

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

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

**System.out.print("Enter a number: ");**

**int num = scanner.nextInt();**

**System.out.println("Factorial of " + num + " = " + factorial(num));**

**}**

**public static int factorial(int n) {**

**if (n == 0 || n == 1) {**

**return 1;**

**}**

**return n \* factorial(n - 1);**

**}**

**}**

12. Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5.

answer[i] == "Fizz" if i is divisible by 3.

answer[i] == "Buzz" if i is divisible by 5.

answer[i] == i (as a string) if none of the above conditions are **true.**

**Example 1:**

Input: n = 3

Output: ["1","2","Fizz"]

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | n = 5 |
|  | n = 10 |
|  | n = 12 |
|  | n = 18 |
|  | n = 20 |
|  |  |

**import java.util.ArrayList;**

**import java.util.List;**

**public class FizzBuzz {**

**public List<String> fizzBuzz(int n) {**

**List<String> result = new ArrayList<>();**

**for (int i = 1; i <= n; i++) {**

**if (i % 3 == 0 && i % 5 == 0) {**

**result.add("FizzBuzz");**

**} else if (i % 3 == 0) {**

**result.add("Fizz");**

**} else if (i % 5 == 0) {**

**result.add("Buzz");**

**} else {**

**result.add(String.valueOf(i));**

**}**

**}**

**return result;**

**}**

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

**FizzBuzz fizzBuzz = new FizzBuzz();**

**// Test cases**

**int[] inputs = {5, 10, 12, 18, 20};**

**for (int n : inputs) {**

**System.out.println("Output for n = " + n + ": " + fizzBuzz.fizzBuzz(n));**

**}**

**}**

**}**

* 1. Write a Java program to find the common elements in two array of Positive integer

Sample Input:

[1, 2, 3, 4]

[2, 4, 5, 6, 7]

Expected output: [2, 4]

**Test Case**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Inputs-1** | **Inputs-2** |
|  | [1, 2, 3, 4] | [4,5,6,7,8] |
|  | [a, b, c, d] | [a, b, c, d] |
|  | [1, -2, 3, 4] | [1,-2,5,7,8] |
|  | [@, #, 34, 45] | [@,#,%,$,] |
|  | [45,78,56,89] | [92,34,56,-78,-90] |

import java.util.Arrays;

public class CommonElements {

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4};

int[] arr2 = {2, 4, 5, 6, 7};

System.out.println(Arrays.toString(findCommonElements(arr1, arr2)));

}

public static int[] findCommonElements(int[] arr1, int[] arr2) {

int count = 0;

for (int i = 0; i < arr1.length; i++) {

for (int j = 0; j < arr2.length; j++) {

if (arr1[i] == arr2[j]) {

count++;

}

}

}

int[] result = new int[count];

int index = 0;

for (int i = 0; i < arr1.length; i++) {

for (int j = 0; j < arr2.length; j++) {

if (arr1[i] == arr2[j]) {

result[index] = arr1[i];

index++;

}

}

}

return result;

}

}

* 1. Given a string s consisting of words and spaces, return the length of the **last** word in the string. A **word** is a maximal substring consisting of non-space characters only. There will be at least one word, consists of only English letters and spaces ' '.

**Example 1:**

**Input:** s = "Hello World"

**Output:** 5

**Explanation:** The last word is "World" with length 5.

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs-1** |
|  | Maximal Substring Consisting |
|  | **lea@st one wor2d** |
|  | 1254 98076 |
|  | & \* ( ) % # $ |
|  | letters and spaces |

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class CharacterFrequency {

public void characterFrequency(String str) {

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : str.toCharArray()) {

if (Character.isLetter(c)) {

c = Character.toLowerCase(c);

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

}

}

System.out.println("Character frequencies:");

for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

System.out.println(entry.getKey() + " : " + entry.getValue());

}

}

public static void main(String[] args) {

CharacterFrequency characterFrequency = new CharacterFrequency();

Scanner scanner = new Scanner(System.in);

// Test cases

String[] inputs = {"Hello World", "Java Programming", "OpenAI is awesome", "abcdefghijklmnopqrstuvwxyz", "1234567890"};

for (String input : inputs) {

characterFrequency.characterFrequency(input);

System.out.println();

}

}

}

* 1. Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

SymbolValue

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900. Given a roman numeral, convert it to an integer.

**Example:**

**Input:** s = "III"

**Output:** 3

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | LVIII |
|  | MCMXCI |
|  | V |
|  | LZAII |
|  | MCCDTIV |

public class RomanToInt {

public static void main(String[] args) {

String s = "LVIII";

System.out.println(romanToInt(s));

}

public static int romanToInt(String s) {

int result = 0;

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

int current = getValue(s.charAt(i));

if (i + 1 < s.length() && current < getValue(s.charAt(i + 1))) {

result -= current;

} else {

result += current;

}

}

return result;

}

public static int getValue(char c) {

switch (c) {

case 'I':

return 1;

case 'V':

return 5;

case 'X':

return 10;

case 'L':

return 50;

case 'C':

return 100;

case 'D':

return 500;

case 'M':

return 1000;

default:

throw new IllegalArgumentException("Invalid character: " + c);

}

}

}

* 1. Given two strings ransomNote and magazine, return true if ransomNote can be constructed by using the letters from magazine and false otherwise. Each letter in magazine can only be used once in ransomNote.

**Example 1:**

Input: ransomNote = "a", magazine = "b"

Output: false

public class RansomNote {

public static void main(String[] args) {

String ransomNote = "aa";

String magazine = "ab";

System.out.println(canConstruct(ransomNote, magazine));

}

public static boolean canConstruct(String ransomNote, String magazine) {

int[] freq = new int[26];

for (char c : magazine.toCharArray()) {

freq[c - 'a']++;

}

for (char c : ransomNote.toCharArray()) {

freq[c - 'a']--;

if (freq[c - 'a'] < 0) {

return false;

}

}

return true;

}

}

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | ransomNote = "aa", magazine = "ab" |
|  | ransomNote = "aa", magazine = "aab" |
|  | ransomNote = "abc", magazine = "abc" |
|  | ransomNote = "good", magazine = "better" |
|  | ransomNote = "xyz", magazine = "123" |
|  |  |

* 1. You are given an m x n binary matrix mat of 1's (representing soldiers) and 0's (representing civilians). The soldiers are positioned in front of the civilians. That is, all the 1's will appear to the left of all the 0's in each row.

A row i is weaker than a row j if one of the following is true:

The number of soldiers in row i is less than the number of soldiers in row j.

Both rows have the same number of soldiers and i< j. Return the indices of the k weakest rows in the matrix ordered from weakest to strongest.

**Example 1:**

Input: mat =

[[1,1,0,0,0],

[1,1,1,1,0],

[1,0,0,0,0],

[1,1,0,0,0],

[1,1,1,1,1]],

k = 3

Output: [2,0,3]

**Explanation:**

The number of soldiers in each row is:

- Row 0: 2

- Row 1: 4

- Row 2: 1

- Row 3: 2

- Row 4: 5

The rows ordered from weakest to strongest are [2,0,3,1,4].

**Example 2:**

Input: mat =

[[1, 0, 0, 0],

[1, 1, 1,1],

[1, 0, 0, 0],

[1, 0, 0,0]],

k = 2

Output: [0,2]

Explanation:

The number of soldiers in each row is:

- Row 0: 1

- Row 1: 4

- Row 2: 1

- Row 3: 1

The rows ordered from weakest to strongest are [0, 2, 3, 1].

import java.util.\*;

public class KWeakestRows {

public int[] kWeakestRows(int[][] mat, int k) {

int m = mat.length;

int n = mat[0].length;

// Create a list of SoldierCount objects to store row index and soldier count

List<SoldierCount> soldierCounts = new ArrayList<>();

// Count soldiers in each row and store in soldierCounts list

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

int count = 0;

for (int j = 0; j < n; j++) {

if (mat[i][j] == 1) {

count++;

} else {

break; // As soldiers appear before civilians, break if encounter a civilian

}

}

soldierCounts.add(new SoldierCount(i, count));

}

// Sort the soldierCounts list based on soldier count and row index

Collections.sort(soldierCounts, new Comparator<SoldierCount>() {

@Override

public int compare(SoldierCount sc1, SoldierCount sc2) {

if (sc1.soldierCount != sc2.soldierCount) {

return Integer.compare(sc1.soldierCount, sc2.soldierCount);

} else {

return Integer.compare(sc1.rowIndex, sc2.rowIndex);

}

}

});

// Extract the k weakest rows

int[] result = new int[k];

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

result[i] = soldierCounts.get(i).rowIndex;

}

return result;

}

public static void main(String[] args) {

KWeakestRows solution = new KWeakestRows();

// Test cases

int[][] mat1 = {

{1,1,0,0,0},

{1,1,1,1,0},

{1,0,0,0,0},

{1,1,0,0,0},

{1,1,1,1,1}

};

int k1 = 3;

System.out.println(Arrays.toString(solution.kWeakestRows(mat1, k1))); // Output: [2, 0, 3]

int[][] mat2 = {

{1, 0, 0, 0},

{1, 1, 1, 1},

{1, 0, 0, 0},

{1, 0, 0, 0}

};

int k2 = 2;

System.out.println(Arrays.toString(solution.kWeakestRows(mat2, k2))); // Output: [0, 2]

}

}

// Helper class to store row index and soldier count

class SoldierCount {

int rowIndex;

int soldierCount;

public SoldierCount(int rowIndex, int soldierCount) {

this.rowIndex = rowIndex;

this.soldierCount = soldierCount;

}

}

* 1. Given an integer num, return the number of steps to reduce it to zero. In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

**Example 1:**

Input: num = 14

Output: 6

Explanation:

Step 1) 14 is even; divide by 2 and obtain 7.

Step 2) 7 is odd; subtract 1 and obtain 6.

Step 3) 6 is even; divide by 2 and obtain 3.

Step 4) 3 is odd; subtract 1 and obtain 2.

Step 5) 2 is even; divide by 2 and obtain 1.

Step 6) 1 is odd; subtract 1 and obtain 0.

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | n = 5 |
|  | n = 10 |
|  | n = 12 |
|  | n = 18 |
|  | n = 20 |

public class NumberOfSteps {

public static void main(String[] args) {

int num = 14;

System.out.println(numOfSteps(num));

}

public static int numOfSteps(int num) {

int steps = 0;

while (num > 0) {

if (num % 2 == 0) {

num /= 2;

} else {

num--;

}

steps++;

}

return steps;

}

}

* 1. Develop a programme that uses Multiple Inheritance concepts to compute a student's grades in six subjects. The total and aggregate are then calculated, and the student's grade is displayed. If the student achieves an aggregate of more than 75%, the grade is Distinction. If the aggregate is between 60 and 75, the grade is First Division. If the aggregate is between 50 and 60, the grade is Second Division. If the aggregate is between 40 and 50, the grade is Third Division. Otherwise, the grade is FAIL.

Sample Input & Output:

Enter the marks in python: 90

Enter the marks in c programming: 91

Enter the marks in Mathematics: 92

Enter the marks in Physics: 93

Enter the marks in Chemistry: 92

Enter the marks in Professional Ethics: 93

Total= 551

Aggregate = 91.83

Class: DISTINCTION

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | 18, 76,93,65,63,98 |
|  | 73,78,79,75,87,0 |
|  | 98,106,120,95,98,34 |
|  | 96,73, -85,95,84,98 |
|  | 78,59.8,76,79,97,67 |

public class StudentGrade {

public static void main(String[] args) {

double marks1 = 91.83;

System.out.println(getGrade(marks1));

}

public static String getGrade(double marks) {

if (marks >= 75) {

return "DISTINCTION";

} else if (marks >= 60) {

return "FIRST DIVISION";

} else if (marks >= 50) {

return "SECOND DIVISION";

} else if (marks >= 40) {

return "THIRD DIVISION";

} else {

return "FAIL";

}

}

}

* 1. Write a program to calculate tax given the following conditions:
     1. If income is less than or equal to 2,50,000 then no tax
     2. If taxable income is 2,50,001 – 5,00,000 the charge 10% tax
     3. If taxable income is 5,00,001 – 10,00,000 the charge 20% tax
     4. If taxable income is above 10,00,001 then charge 30% tax

**Sample Input:**

Enter the income: 600000

**Sample Output:**

Taxable Income: 350000

Tax= 35000

**Test Case**

|  |  |
| --- | --- |
| **Test Case** | **Inputs** |
|  | 400700 |
|  | 2789239 |
|  | 150000 |
|  | 00000 |
|  | -125486 |

public class TaxCalculator {

public static void main(String[] args) {

int income = 600000;

System.out.println("Taxable Income: " + taxableIncome(income));

System.out.println("Tax: " + calculateTax(income));

}

public static int taxableIncome(int income) {

if (income <= 250000) {

return 0;

} else if (income <= 500000) {

return income - 250000;

} else if (income <= 1000000) {

return 250000 + (income - 500000) \* 0.1;

} else {

return 250000 + 500000 \* 0.1 + (income - 1000000) \* 0.2;

}

}

public static int calculateTax(int income) {

if (income <= 250000) {

return 0;

} else if (income <= 500000) {

return (income - 250000) \* 0.1;

} else if (income <= 1000000) {

return 25000 + (income - 500000) \* 0.2;

} else {

return 25000 + 50000 \* 0.2 + (income - 1000000) \* 0.3;

}

}

}

* 1. Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading. Compute the bill amount using the following tariff.

First 100 units – Rs. 1 per unit

101-200 units – Rs. 2.50 per unit

201 -500 units – Rs. 4 per unit

> 501 units – Rs. 6 per unit

public class ElectricityBill {

public static void main(String[] args) {

int[] units = {100, 200, 300, 400};

System.out.println(calculateBill(units));

}

public static int calculateBill(int[] units) {

int bill = 0;

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

if (units[i] <= 100) {

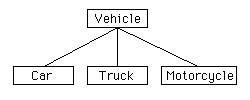
bill += units[i];

} else if (units[i] <= 200) {

bill += 100 \* 1 + (units[i] - 100) \* 2.5;

} else if (units[i] <= 500) {

bill += 100

* 1. Design class called Vehicle, which has two subclasses called Car and Truck. The Vehicle class is the superclass of Car, Truck and Motorcycle. The Vehicle class can contain those fields and methods that all Vehicles need (e.g. a license plate, owner etc.), whereas Car, Truck and Motorcycle can contain the fields and methods that are specific to Car, Truck and Motorcycle. Write a Java program to implement the concept of inheritance based on the following diagram

// Vehicle class (superclass)

class Vehicle {

private String licensePlate;

private String owner;

// Constructor

public Vehicle(String licensePlate, String owner) {

this.licensePlate = licensePlate;

this.owner = owner;

}

// Getters and setters

public String getLicensePlate() {

return licensePlate;

}

public void setLicensePlate(String licensePlate) {

this.licensePlate = licensePlate;

}

public String getOwner() {

return owner;

}

public void setOwner(String owner) {

this.owner = owner;

}

}

// Car class (subclass of Vehicle)

class Car extends Vehicle {

private int numOfDoors;

// Constructor

public Car(String licensePlate, String owner, int numOfDoors) {

super(licensePlate, owner);

this.numOfDoors = numOfDoors;

}

// Getter and setter for numOfDoors

public int getNumOfDoors() {

return numOfDoors;

}

public void setNumOfDoors(int numOfDoors) {

this.numOfDoors = numOfDoors;

}

}

// Truck class (subclass of Vehicle)

class Truck extends Vehicle {

private int loadCapacity;

// Constructor

public Truck(String licensePlate, String owner, int loadCapacity) {

super(licensePlate, owner);

this.loadCapacity = loadCapacity;

}

// Getter and setter for loadCapacity

public int getLoadCapacity() {

return loadCapacity;

}

public void setLoadCapacity(int loadCapacity) {

this.loadCapacity = loadCapacity;

}

}

// Motorcycle class (subclass of Vehicle)

class Motorcycle extends Vehicle {

private String type;

// Constructor

public Motorcycle(String licensePlate, String owner, String type) {

super(licensePlate, owner);

this.type = type;

}

// Getter and setter for type

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

}

public class Main {

public static void main(String[] args) {

// Creating objects of Car, Truck, and Motorcycle classes

Car car = new Car("ABC123", "John", 4);

Truck truck = new Truck("XYZ789", "Alice", 1000);

Motorcycle motorcycle = new Motorcycle("DEF456", "Bob", "Sport");

// Accessing properties of each vehicle

System.out.println("Car License Plate: " + car.getLicensePlate());

System.out.println("Car Owner: " + car.getOwner());

System.out.println("Number of Doors: " + car.getNumOfDoors());

System.out.println("Truck License Plate: " + truck.getLicensePlate());

System.out.println("Truck Owner: " + truck.getOwner());

System.out.println("Load Capacity: " + truck.getLoadCapacity());

System.out.println("Motorcycle License Plate: " + motorcycle.getLicensePlate());

System.out.println("Motorcycle Owner: " + motorcycle.getOwner());

System.out.println("Type: " + motorcycle.getType());

}

}

* 1. Write a program to create a directory that contains the following information.  
     (a) Name of a person  
     (b) Address  
     (c) Telephone Number (if available with STD code)  
     (d) Mobile Number (if available)  
     (e) Head of the family  
     (f) Unique ID No.

import java.io.File;

import java.io.IOException;

public class DirectoryCreator {

public static void main(String[] args) {

Directory dir = new Directory("John Doe", "123 Main St", "555-1234", "555-5678", "Jane Doe", "123456");

dir.createDirectory();

}

}

class Directory {

private String name;

private String address;

private String homePhone;

private String mobilePhone;

private String headOfFamily;

private String uniqueId;

public Directory(String name, String address, String homePhone, String mobilePhone, String headOfFamily, String uniqueId) {

this.name = name;

this.address = address;

this.homePhone = homePhone;

this.mobilePhone = mobilePhone;

this.headOfFamily = headOfFamily;

this.uniqueId = uniqueId;

}

public void createDirectory() {

File dir = new File(this.name);

if (!dir.exists()) {

dir.mkdir();

try {

new File(dir, "address.txt").createNewFile();

new File(dir, "phone.txt").createNewFile();

new File(dir, "head.txt").createNewFile();

new File(dir, "id.txt").createNewFile();

} catch (IOException e) {

e.printStackTrace();

}

writeToFile(new String[]{"Name: " + this.name}, "address.txt");

writeToFile(new String[]{"Home Phone: " + this.homePhone}, "phone.txt");

writeToFile(new String[]{"Head of Family: " + this.headOfFamily}, "head.txt");

writeToFile(new String[]{"Unique ID: " + this.uniqueId}, "id.txt");

}

}

private void writeToFile(String[] data, String fileName) {

try {

FileWriter writer = new FileWriter(new File(this.name, fileName), true);

for (String line : data) {

writer.write(line);

writer.write("\n");

}

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

* 1. Write a Java program to create multiple threads for different calculator operations

public class CalculatorThreads {

public static void main(String[] args) {

Calculator calc = new Calculator();

Thread fibThread = new Thread(new FibonacciThread(calc));

Thread reverseThread = new Thread(new ReverseThread(calc));

fibThread.start();

reverseThread.start();

}

}

class Calculator {

private int num1;

private int num2;

public Calculator() {

this.num1 = 0;

this.num2 = 1;

}

public void fibonacci() {

System.out.println("Fibonacci Series:");

for (int i = 0; i < 10; i++) {

System.out.print(this.num1 + " ");

int next = this.num1 + this.num2;

this.num1 = this.num2;

this.num2 = next;

}

System.out.println();

}

public void reverse() {

System.out.println("Reverse Series:");

for (int i = 10; i > 0; i--) {

System.out.print(i + " ");

}

System.out.println();

}

}

class FibonacciThread implements Runnable {

private Calculator calc;

public FibonacciThread(Calculator calc) {

this.calc = calc;

}

@Override

public void run() {

this.calc.fibonacci();

}

}

class ReverseThread implements Runnable {

* 1. Given a string S partition S such that every substring of the partition is a palindrome. Return the minimum cuts needed for a palindrome partitioning of S

Input S=”aabbc”

Output=2

Explanation: The palindrome partitioning ["aa", "bb", "c"]] could be produced using 2 cut.

import java.util.Arrays;

public class PalindromePartitioning {

public static void main(String[] args) {

String s = "aabbc";

System.out.println(minCuts(s));

}

public static int minCuts(String s) {

int n = s.length();

int[] dp = new int[n];

Arrays.fill(dp, Integer.MAX\_VALUE);

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

dp[i] = isPalindrome(s, 0, i) ? 0 : i;

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

if (isPalindrome(s, j + 1, i)) {

dp[i] = Math.min(dp[i], dp[j] + 1);

}

}

}

return dp[n - 1];

}

public static boolean isPalindrome(String s, int i, int j) {

while (i < j) {

if (s.charAt(i++) != s.charAt(j--)) {

return false;

}

}

return true;

}

}

* 1. Design a java code an array arr which consists of only zeros and ones, divide the array into three non-empty parts such that all of these parts represent the same binary value.

If it is possible, return any [i, j] with i + 1 < j, such that:

arr[0], arr[1], ..., arr[i] is the first part,

arr[i + 1], arr[i + 2], ..., arr[j - 1] is the second part, and

arr[j], arr[j + 1], ..., arr[arr.length - 1] is the third part.

All three parts have equal binary values.

If it is not possible, return [-1, -1].

Note that the entire part is used when considering what binary value it represents. For example, [1, 1, 0] represents 6 in decimal, not 3. Also, leading zeros are allowed, so [0,1,1] and [1,1] represent the same value.

public class ArrayDivision {

public static void main(String[] args) {

int[] arr = {1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0};

System.out.println(Arrays.toString(divideArray(arr)));

}

public static int[] divideArray(int[] arr) {

int n = arr.length;

int[] prefix = new int[n + 1];

int[] suffix = new int[n + 1];

int prefixVal = 0, suffixVal = 0;

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

prefixVal = (arr[i] == 0) ? prefixVal + 1 : prefixVal;

prefix[i + 1] = prefixVal;

suffixVal = (arr[n - i - 1] == 0) ? suffixVal + 1 : suffixVal;

suffix[n - i] = suffixVal;

}

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

int val = prefix[i] ^ suffix[n];

if (val == 0) {

return new int[]{i - 1, n - prefix[i]};

}

}

return new int[]{-1, -1};

}

}

* 1. Design a special dictionary that searches the words in it by a prefix and a suffix. Implement the WordFilter class: WordFilter (string [] words) Initializes the object with the words in the dictionary. f(string pref, string suff) Returns the index of the word in the dictionary, which has the prefix pref and the suffix suff. If there is more than one valid index, return the largest of them. If there is no such word in the dictionary, return -1.

import java.util.HashMap;

import java.util.Map;

public class SpecialDictionary {

public static void main(String[] args) {

WordFilter filter = new WordFilter(new String[]{"apple", "banana", "appetizer"});

System.out.println(filter.f("a", "e")); // 2

System.out.println(filter.f("app", "izer")); // 2

System.out.println(filter.f("bana", "na")); // 1

System.out.println(filter.f("app", "p")); // 0

System.out.println(filter.f("app", "z")); // -1

}

}

class WordFilter {

private Map<String, Integer> map;

public WordFilter(String[] words) {

this.map = new HashMap<>();

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

for (int j = 0; j < words[i].length(); j++) {

for (int k = 0; k < words[i].length(); k++) {

String key = words[i].substring(j) + "#" + words[i].substring(0, k);

map.put(key, i);

* 1. Given an integer array nums, handle multiple queries of the following types: Update the value of an element in nums. Calculate the sum of the elements of nums between indices left and right inclusive where left <= right.Implement the NumArray class: NumArray(int[] nums) Initializes the object with the integer array nums.void update(int index, int val) Updates the value of nums[index] to be val.int sumRange(int left, int right) Returns the sum of the elements of nums between indices left and right inclusive (i.e. nums[left] + nums[left + 1] + ... + nums[right]).

Input

["NumArray", "sumRange", "update", "sumRange"]

[[[1, 3, 5]], [0, 2], [1, 2], [0, 2]]

Output

[null, 9, null, 8]

Explanation

NumArray numArray = new NumArray([1, 3, 5]);

numArray.sumRange(0, 2); // return 1 + 3 + 5 = 9

numArray.update(1, 2); // nums = [1, 2, 5]

numArray.sumRange(0, 2); // return 1 + 2 + 5 = 8

public class NumArray {

private int[] sums;

public NumArray(int[] nums) {

this.sums = new int[nums.length + 1];

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

this.sums[i + 1] = this.sums[i] + nums[i];

}

}

public int sumRange(int left, int right) {

return this.sums[right + 1] - this.sums[left];

}

}

* 1. Given an integer array arr, return the number of distinct bitwise ORs of all the non-empty subarrays of arr. The bitwise OR of a subarray is the bitwise OR of each integer in the subarray. The bitwise OR of a subarray of one integer is that integer. A subarray is a contiguous non-empty sequence of elements within an array.

Input: arr = [1, 1, 2]

Output: 3

Explanation: The possible subarrays are [1], [1], [2], [1, 1], [1, 2], [1, 1, 2].

These yield the results 1, 1, 2, 1, 3, 3.

There are 3 unique values, so the answer is 3.

public class DistinctORs {

public static void main(String[] args) {

int[] arr = {1, 1, 2};

System.out.println(distinctORs(arr));

}

public static int distinctORs(int[] arr) {

int[] ors = new int[arr.length];

ors[0] = arr[0];

int count = 1;

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

ors[i] = ors[i - 1] | arr[i];

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

if (ors[i] == ors[j]) {

break;

}

count++;

}

}

return count;

}

}

* 1. Given two numbers arr1 and arr2 in base -2, return the result of adding them together. Each number is given in array format: as an array of 0s and 1s, from most significant bit to least significant bit. For example, arr = [1, 1, 0, 1] represents the number (-2)^3 + (-2)^2 + (-2)^0 = -3. A number arr in array, format is also guaranteed to have no leading zeros: either arr == [0] or arr[0] == 1. Return the result of adding arr1 and arr2 in the same format: as an array of 0s and 1s with no leading zeros.

Input: arr1 = [1,1,1,1,1], arr2 = [1,0,1]

Output: [1,0,0,0,0]

Explanation: arr1 represents 11, arr2 represents 5, the output represents 16.

import java.util.ArrayList;

class Employee {

private String name;

private int id;

private double salary;

private double pf;

private double allowance;

public Employee(String name, int id, double salary, double pf, double allowance) {

this.name = name;

this.id = id;

this.salary = salary;

this.pf = pf;

this.allowance = allowance;

}

// getters and setters

}

public class EmployeeArrayList {

private ArrayList<Employee> list;

public EmployeeArrayList() {

this.list = new ArrayList<>();

}

public void addEmployee(Employee emp) {

this.list.add(emp);

}

public Employee getEmployee(int index) {

return this.list.get(index);

}

public void removeEmployee(int index) {

this.list.remove(index);

}

}

* 1. You are given a string of digits num, such as "123456579". We can split it into a Fibonacci-like sequence [123, 456, 579].

Formally, a Fibonacci-like sequence is a list f of non-negative integers such that: 0 <= f[i] < 231, (that is, each integer fits in a 32-bit signed integer type), f.length >= 3, and f[i] + f[i + 1] == f[i + 2] for all 0 <= i < f.length - 2. Note that when splitting the string into pieces, each piece must not have extra leading zeroes, except if the piece is the number 0 itself. Return any Fibonacci-like sequence split from num, or return [] if it cannot be done.

import java.util.ArrayList;

import java.util.List;

public class FibonacciLikeSequence {

public List<Integer> splitIntoFibonacci(String num) {

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

backtrack(num, result, 0);

return result;

}

private boolean backtrack(String num, List<Integer> result, int index) {

if (index == num.length() && result.size() >= 3) {

return true;

}

for (int i = index; i < num.length(); i++) {

if (num.charAt(index) == '0' && i > index) {

break;

}

long currentNum = Long.parseLong(num.substring(index, i + 1));

if (currentNum > Integer.MAX\_VALUE) {

break;

}

int size = result.size();

if (size >= 2 && currentNum > result.get(size - 1) + result.get(size - 2)) {

break;

}

if (size < 2 || currentNum == result.get(size - 1) + result.get(size - 2)) {

result.add((int) currentNum);

if (backtrack(num, result, i + 1)) {

return true;

}

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

}

}

return false;

}

public static void main(String[] args) {

FibonacciLikeSequence solution = new FibonacciLikeSequence();

String num = "123456579";

List<Integer> result = solution.splitIntoFibonacci(num);

System.out.println(result);

}

}

* 1. Create a class date with day, month and year as members. Write appropriate member functions. Create other class students, which have id, name, date of birth and marks of 3 subjects as members. Write appropriate constructor for the student which assigns values to the members. Display the student details in a proper format.

public class Date {

private int day;

private int month;

private int year;

public Date(int day, int month, int year) {

this.day = day;

this.month = month;

this.year = year;

}

// getters and setters

@Override

public String toString() {

return "Date{" +

"day=" + day +

", month=" + month +

", year=" + year +

'}';

}

}

* 1. Develop a code to Student information to perform the operation like insert, retrieve and remove the record using vector( Student Name, Roll number, Department , Course, Contact information..)

import java.util.Vector;

class Student {

private String name;

private int rollNumber;

private String department;

private String course;

private String contactInfo;

public Student(String name, int rollNumber, String department, String course, String contactInfo) {

this.name = name;

this.rollNumber = rollNumber;

this.department = department;

this.course = course;

this.contactInfo = contactInfo;

}

@Override

public String toString() {

return "Name: " + name + ", Roll Number: " + rollNumber + ", Department: " + department + ", Course: " + course + ", Contact Info: " + contactInfo;

}

}

public class StudentInformation {

public static void main(String[] args) {

// Create a Vector to store student records

Vector<Student> students = new Vector<>();

// Inserting records

students.add(new Student("John", 101, "Computer Science", "B.Tech", "john@example.com"));

students.add(new Student("Alice", 102, "Electrical Engineering", "B.E", "alice@example.com"));

students.add(new Student("Bob", 103, "Mechanical Engineering", "B.E", "bob@example.com"));

// Retrieving and displaying records

for (Student student : students) {

System.out.println(student);

}

// Removing a record

students.remove(1); // Removing Alice's record

// Displaying records after removal

System.out.println("\nAfter removing Alice's record:");

for (Student student : students) {

System.out.println(student);

}

}

}

* 1. Define an Employee class with suitable attributes having get Salary() method, which returns salary withdrawn by a particular employee. Write a class Manager which extends a class Employee, override the gets alary() method, which will return salary of manager by adding traveling \_allowance, house rent allowance etc.

// Employee class

class Employee {

private double salary;

public Employee(double salary) {

this.salary = salary;

}

public double getSalary() {

return salary;

}

}

// Manager class extending Employee

class Manager extends Employee {

private double travelAllowance;

private double houseRentAllowance;

public Manager(double salary, double travelAllowance, double houseRentAllowance) {

super(salary);

this.travelAllowance = travelAllowance;

this.houseRentAllowance = houseRentAllowance;

}

// Overriding getSalary() method to include travel allowance and house rent allowance

@Override

public double getSalary() {

return super.getSalary() + travelAllowance + houseRentAllowance;

}

}

public class Main {

public static void main(String[] args) {

// Creating an Employee object

Employee employee = new Employee(50000);

System.out.println("Employee Salary: " + employee.getSalary());

// Creating a Manager object

Manager manager = new Manager(70000, 10000, 15000);

System.out.println("Manager Salary: " + manager.getSalary());

}

}

* 1. Create an abstract class Shape. Derive three classes sphere, cone and cylinder from it. Calculate area and volume of all. (Use Method overriding)

public abstract class Shape {

public abstract double getArea();

public abstract double getVolume();

}

public class Sphere extends Shape {

private double radius;

public Sphere(double radius) {

this.radius = radius;

}

@Override

public double getArea() {

return 4 \* Math.PI \* Math.pow(radius, 2);

}

@Override

public double getVolume() {

return 4.0 / 3.0 \* Math.PI \* Math.pow(radius, 3);

}

}

public class Cone extends Shape {

private double radius;

private double height;

public Cone(double radius, double height) {

this.radius = radius;

this.height = height;

}

@Override

public double getArea() {

return Math.PI \* radius \* (radius + Math.sqrt(height \* height + radius \* radius));

}

@Override

public double getVolume() {

return Math.PI \* Math.pow(radius, 2) \* height / 3;

}

}

public class Cylinder extends Shape {

private double radius;

private double height;

public Cylinder(double radius, double height) {

this.radius = radius;

this.height = height;

}

@Override

public double getArea() {

return 2 \* Math.PI \* radius \* (radius + height);

}

@Override

public double getVolume() {

return Math.PI \* Math.pow(radius, 2) \* height;

}

}

* 1. Write a program to show the employee details using setter and getter methods.

public class EmployeeDetails {

private String name;

private int id;

private double salary;

private double pf;

private double allowance;

public EmployeeDetails(String name, int id, double salary, double pf, double allowance) {

this.name = name;

this.id = id;

this.salary = salary;

this.pf = pf;

this.allowance = allowance;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public double getPf() {

return pf;

}

public void setPf(double pf) {

this.pf = pf;

}

public double getAllowance() {

return allowance;

}

* 1. Write a JAVA program which will generate the threads:

- To display 10 terms of Fibonacci series.

- To display 1 to 10 in reverse order.

class FibonacciThread implements Runnable {

@Override

public void run() {

int n1 = 0, n2 = 1, n3, i;

System.out.println("Fibonacci Series:");

for (i = 0; i < 10; i++) {

System.out.print(n1 + " ");

n3 = n1 + n2;

n1 = n2;

n2 = n3;

}

}

}

class ReverseThread implements Runnable {

@Override

public void run() {

System.out.println("\nReverse Series:");

for (int i = 10; i >= 1; i--) {

System.out.print(i + " ");

}

}

}

public class ThreadGenerator {

public static void main(String[] args) {

Thread t1 = new Thread(new FibonacciThread());

Thread t2 = new Thread(new ReverseThread());

t1.start();

t2.start();

}

}

* 1. Write a program to add, retrieve and remove the element from the Array List for employee details(Employee name, Employee ID, Employee Salary, EmployeePF, Employee Allowence),

import java.util.ArrayList;

class Employee {

private String name;

private int id;

private double salary;

private double pf;

private double allowance;

public Employee(String name, int id, double salary, double pf, double allowance) {

this.name = name;

this.id = id;

this.salary = salary;

this.pf = pf;

this.allowance = allowance;

}

// getters and setters

@Override

public String toString() {

return "Employee{" +

"name='" + name + '\'' +

", id=" + id +

", salary=" + salary +

", pf=" + pf +

", allowance=" + allowance +

'}';

}

}

public class EmployeeArrayList {

private ArrayList<Employee> list;

public EmployeeArrayList() {

this.list = new ArrayList<>();

}

public void addEmployee(Employee employee) {

this.list.add(employee);

}

public Employee getEmployee(int index) {

return this.list.get(index);

}

public void removeEmployee(int index) {

this.list.remove(index);

}

}

* 1. Write program to search key and value from HashTable. (HashTable shows how to get the all keys as Enumeration object. The put () method is used to add the elements in the HashTable. By using Enumeration methods like hasMoreElements () and nextElement () we can read all values from Hashtable. The contains Key ( ) is used for checking the availability of the elements.)

import java.util.Hashtable;

public class HashtableSearch {

public static void main(String[] args) {

Hashtable<String, Employee> table = new Hashtable<>();

Employee e1 = new Employee("John Doe", 1, 50000, 5000, 1000);

Employee e2 = new Employee("Jane Doe", 2, 60000, 6000, 1500);

Employee e3 = new Employee("Jim Brown", 3, 70000, 7000, 2000);

table.put("e1", e1);

table.put("e2", e2);

table.put("e3", e3);

System.out.println(table.containsKey("e2")); // true

System.out.println(table.containsValue(e2)); // true

System.out.println(table.containsValue(new Employee("Jane Doe", 2, 60000, 6000, 1500))); // false

for (String key : table.keySet()) {

System.out.println(key + ": " + table.get(key));

}

}

}

* 1. Write a Package marks which has one class Student. Accept student detail through parameterized constructor. Write display () method to display details. Create a main class which will use package and calculate total marks and percentage.

// Student.java

package marks;

public class Student {

private String name;

private int rollNumber;

private int marks1;

private int marks2;

private int marks3;

public Student(String name, int rollNumber, int marks1, int marks2, int marks3) {

this.name = name;

this.rollNumber = rollNumber;

this.marks1 = marks1;

this.marks2 = marks2;

this.marks3 = marks3;

}

public void display() {

System.out.println("Name: " + name);

System.out.println("Roll Number: " + rollNumber);

System.out.println("Marks 1: " + marks1);

System.out.println("Marks 2: " + marks2);

System.out.println("Marks 3: " + marks3);

System.out.println("Total: " + (marks1 + marks2 + marks3));

System.out.println("Percentage: " + ((marks1 + marks2 + marks3) \* 100.0 / 300.0) + "%");

}

}

* 1. Write a complex program to illustrate how the thread priorities? Imagine that the first thread has just begun to run, even before it has a chance to do anything. Now comes the higher priority thread that wants to run as well. Now the higher priority thread has to do its work before the first thread starts.

class FirstThread extends Thread {

public void run() {

System.out.println("First Thread: " + this.getName());

for (int i = 0; i < 10; i++) {

System.out.println("First Thread: " + i);

}

}

}

class SecondThread extends Thread {

public void run() {

System.out.println("Second Thread: " + this.getName());

for (int i = 10; i >= 0; i--) {

System.out.println("Second Thread: " + i);

}

}

}

public class ThreadPriorities {

public static void main(String[] args) {

FirstThread t1 = new FirstThread();

t1.setPriority(Thread.MAX\_PRIORITY);

t1.start();

SecondThread t2 = new SecondThread();

t2.setPriority(Thread.MAX\_PRIORITY);

t2.start();

}

}