21.Example for switch expression and multi label case statements:

public class SwitchExample {

public static void main(String[] args) {

int number=20;

switch(number){

case 10: System.out.println("10");

break;

case 20: System.out.println("20");

break;

case 30: System.out.println("30");

break;

default:System.out.println("Not in 10, 20 or 30");

}

}

}

Output: 20

How do you write an interface with default and static method:

Default Interface Methods in Action

To better understand the functionality of default interface methods, let's create a simple example.Suppose we have a naive Vehicle interface and just one implementation. There could be more, but let's keep it that simple:

public interface Vehicle {

String getBrand();

String speedUp();

String slowDown();

default String turnAlarmOn() {

return "Turning the vehicle alarm on.";

}

default String turnAlarmOff() {

return "Turning the vehicle alarm off.";

}

public class Car implements Vehicle {

private String brand{

public String getBrand() {

return brand;

}

public String speedUp() {

return "The car is speeding up.";

}

public String slowDown() {

return "The car is slowing down.";

}

}

public static void main(String[] args) {

Vehicle car = new Car("BMW");

System.out.println(car.getBrand());

System.out.println(car.speedUp());

System.out.println(car.slowDown());

System.out.println(car.turnAlarmOn());

System.out.println(car.turnAlarmOff());

}

23.String joiner in java8:

StringJoiner is used to construct a sequence of characters separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix.

Pgm to demonstrate usage of break statement:

class GFG {

public static void main(String args[])

{

int i = 2;

switch (i) {

case 0:

System.out.println("i is zero.");

break;

case 1:

System.out.println("i is one.");

break;

case 2:

System.out.println("i is two.");

break;

default:

System.out.println("Invalid number");

}

}

Output:

2

Pgm to demonstrate usage of continue statement:

class GFG {

public static void main(String args[]){

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

if (i == 2)

continue;

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

}

}

26.pgm to print the output:

import java.util.Scanner;

public class Series

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int s = 0, c;

for (c = 1; c <= n; c++) {

s = s \* 10 + c;

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

}

}

}

Output:

Enter the number of terms: 6

1

12

123

1234

12345

123456

27. pgm for reading data from a file using FileReader

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

class ReadFile

{

public static void main(String[] args) throws IOException

{

Int ch;

FileReader fr=null;

try

{

fr = new FileReader("text");

}

catch (FileNotFoundException fe) {

System.out.println("File not found"); }

while ((ch=fr.read())!=-1)

System.out.print((char)ch);

fr.close();

} }String str = "File Handling in Java using "+ " FileWriter and FileReade

FileWriter fw=new FileWriter("output.txt"):

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

fw.write(str.charAt(i));

System.out.println("Writing successful");

fw.close();}

}

28 implement a factorial using recursion;

class FactorialExample2{

static int factorial(int n){

if (n == 0)

return 1;

else

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

}

public static void main(String args[]){

int i,fact=1;

int number=4;//

fact = factorial(number);

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

}

}

Output:

Factorial of 4 is: 24

29.Implement multiple inheritances using an interface:

interface AnimalEat {

void eat();

}

interface AnimalTravel {

void travel();

}

class Animal implements AnimalEat, AnimalTravel {

public void eat() {

System.out.println("Animal is eating");

}

public void travel() {

System.out.println("Animal is travelling");

}

}

public class Demo {

public static void main(String args[]) {

Animal a = new Animal();

a.eat();

a.travel();

}

}

Output

Animal is eating

Animal is travelling

30.pgm for enumeration:

class EnumExample1{

public enum Season { WINTER, SPRING, SUMMER, FALL)

public static void main(String[] args) {

for (Season s : Season.values())

System.out.println(s);

}}

Output:

WINTER

SPRING

SUMMER

FALL

31.pgm to implement overloading

public class MyClass

{

public int multiply(int num1, num2)

{

return num1 \* num2;

}

public float multiply(int num1, num2)

{

return num1 \* num2;

}

}

31.implement method for overriding :

package com.techvidvan.methodoverriding;

class Parent

{

void view()

{

System.out.println("This is a parent class method");

}

}

class Child extends Parent

{

void view()

{

System.out.println("This is a child class method");

}

}

public class MethodOverriding

{

public static void main(String args[])

{

Parent obj = new Parent();

obj.view();

Parent obj1 = new Child();

obj1.view();

}

}

Output:

This is a parent class method

This is a child class method

32 pgm to find duplicate values in a list:

public class DuplicateElement {

public static void main(String[] args) {

int [] arr = new int [] {1, 2, 3, 4, 2, 7, 8, 8, 3};

System.out.println("Duplicate elements in given array: ");

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

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

if(arr[i] == arr[j])

System.out.println(arr[j]);

}

}

}

}

Output:

Duplicate element in given array :

2

3

8

33.prgm to merge a two array:

import java.util.Arrays;

public class MergeArrayExample1

{

public static void main(String[] args)

{

int[] firstArray = {23,45,12,78,4,90,1};

int[] secondArray = {77,11,45,88,32,56,3};

int fal = firstArray.length;

int sal = secondArray.length;

int[] result = new int[fal + sal];

System.arraycopy(firstArray, 0, result, 0, fal);

System.arraycopy(secondArray, 0, result, fal, sal);

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

}

}

Output;

{23,45,12,78,4,90,1,77,11,45,88,32,56,3}

34.pgm to reverse a string using stack:

import java.io.\*;

import java.util.\*;

class ReverseStringUsingStack {

public static String ReverseString(String s)

{

char[] rS = new char[s.length()];

Stack<Character> st = new Stack<Character>();

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

st.push(s.charAt(j));

}

int j = 0;

while (!st.isEmpty()) {

rS[j++] = st.pop();

}

return new String(rS);

}

public static void main(String args[])

{

String s1 = " JavaTpoint ";

System.out.println(s1 + " <- Reverse -> " + ReverseString(s1));

String s2 = " Welcome to JavaTpoint ";

System.out.println(s2 + " <- Reverse -> " + ReverseString(s2));

}

}

36.pgm for Fibonacci series:

class FibonacciExample1{

public static void main(String args[])

{

int n1=0,n2=1,n3,i,count=10;

System.out.print(n1+" "+n2);

for(i=2;i<count;++i0

{

n3=n1+n2;

System.out.print(" "+n3);

n1=n2;

n2=n3;

}

}}

Output:

0 1 1 2 3 5 8 13 21 34

String string buffer

String is immutable. It is mutable.

It is slow in terms of executing the concatenation task It is fast in terms of executing the concatenation task.

Here the length of the string class is static. Here the length can be modified whenever required, as it is dynamic in behaviour.

It is less efficient. It is more efficient in nature as compared to the string class.

String consumes more as compared to the stringbuffer. StringBuffer uses less memory as compared to the string.

38 constructor overloading

Constructor overloading in Java refers to the use of more than one constructor in an instance class. However, each overloaded constructor must have different signatures. For the compilation to be successful, each constructor must contain a different list of arguments

forEach statement can be used along with lambda expression that reduces the looping through a Map to a single statement and also iterates over the elements of a list. The forEach() method defined in an Iterable interface and accepts lambda expression as a parameter

39.Java Program to swap two numbers without using temporary variable

import java.io.\*;

class Geeks {

public static void main(String a[])

{

int x = 10;

int y = 5;

x = x + y;

y = x - y;

x = x - y;

System.out.println("After swapping: + " x = " + x + ", y = " + y); }

}

Output

After Swapping: x =5, y=10

Implement exception handling without catch block:

You can handle exceptions still without having catch blocks also, only thing you need to do is declare the throws clause in your method signature, so that the calling function would handle the exception. Before throwing exception, it executes the finally block.

43.iterate the map values using lambda expression:

import java.util.\*;

public class MapIterateLambdaTest {

public static void main(String[] args) {

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

ranks.put("India", 1);

ranks.put("Australia", 2);

ranks.put("England", 3);

ranks.put("Newzealand", 4);

ranks.put("South Africa", 5);

// Iterating through

forEach using Lambda Expression

ranks.forEach((k,v) -> System.out.println("Team : " + k + ", Rank : " + v));

}

}

Output

Team : Newzealand, Rank : 4

Team : England, Rank : 3

Team : South Africa, Rank : 5 ,Team : Australia, Rank : 2 ,Team : India, Rank : 1