Keywords:

1. import 11.try 17.strictfp 26.do 35.boolean
2. Class 12.catch 18.instance of 27.for 36.byte
3. This 13.finally 19.enum 28.while 37.short
4. Super 14.throw 20.interface 29.if 38.int
5. Extends 15.throws 21.assert 30.else 39.char
6. Package 16.final 22.native 31.switch 40.float
7. Return 23.volatite 32.default 41.double
8. Public 24.implements 33.break 42.long
9. Private 25.abstract 34.continue 43.void
10. protected

* Keywords are special words which are used to recognize structure of program.
* Each keyword will be having some specific task.

Primitive Data Type:

byte theByte=-128;

short theshort=32\_000;

int theInt=123\_456\_789;

long theLong=1234567L;

float pi=3.14F;

double doublePi=3.14;

boolean isAdult=true;

char nameInitial='A';

Non-Primitive Data Type:

1.Class

2.String

3.Array…

public class NonPrimitiveDataTypes {

public static void main(String[] args) {

// non primitive data types

String name=new String("kuralarasan");

System.out.println(name.toUpperCase());

LocalDate now=LocalDate.now();

System.out.println(now);

System.out.println(now.getMonth());

}

}

prefix and postfix form of increment & decrement operators:

public class IncrementAndDecrement {

public static void main(String[] args) {

// ++n(pre-increment),++n(post increment)

int n=5;

System.out.println(n++);//5

System.out.println(n);//6

System.out.println(++n);//7

System.out.println(--n);//6

System.out.println(n--);//6

System.out.println(n);

}

}

Control Statements / Structures in java:

A statement that causes a jump of control from one part of the program to another is called control Statement or control Structure.

Types of control statements:

1.conditional statements [if,if else,else if ladder, nested if, switch]

2.iterative / Looping Statements [for, while, do-while,Enhanced for loop]

3.unconditional jump statements [break, continue]

Unconditional Statements:

break,continue(keywords) - used in iterative statements (loops)

break - whenever control reaches break, it comes out from the loop it is also used with switch

continue - whenever control reaches continue , it comes out only from the current iteration

public class UncontionalStatement {

public static void main(String[] args) {

for(int i=1;i<=10;i++) { // i=i+1 or i++

if(i==5) {

// break;

continue;

}

System.out.println("hello "+i);

}

System.out.println("bye");

}

}

Class and Objects:

Class - blueprint

Object - physical entity that follows blueprint

Ex:

class student{

//1.Methods

//2.variables(data members)

String name;

int rank;

void display() {

System.***out***.println("name = "+name);

System.***out***.println("rank = "+rank);

}

}

public class ClassAndObject {

public static void main(String[] args) {

student s1 =new student();

System.***out***.println("user1");

s1.name="kural";

s1.rank=1;

s1.display();

student s2=new student();

System.***out***.println("user2");

s2.name="naveen";

s2.rank=2;

s2.display();

}

}

User Input using Scanner class:

import java.util.Scanner;

public class ScannerUserInput {

public static void main(String[] args) {

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

System.***out***.println("enter your name");

String name=sc.next();

System.***out***.println("enter your age");

int age=sc.nextInt();

System.***out***.println("enter your phone");

long phone=sc.nextLong();

System.***out***.println("enter your temperature");

double temp=sc.nextDouble();

System.***out***.println("enter your charater");

char c=sc.next().charAt(0);

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

System.***out***.println("age : "+age);

System.***out***.println("phone : "+phone);

System.***out***.println("temperature : "+temp);

System.***out***.println("character : "+c);

sc.close();

}

}

Arrays Introduction:

Ex:1

public class ArraysDemo {

public static void main(String[] args) {

int a[]=new int[5];

a[0]=10;

a[1]=20;

a[2]=30;

a[3]=40;

a[4]=50;

System.***out***.println(a[3]);

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

System.***out***.println(a[i]);

}

System.***out***.println("-------------------------");

int b[]= {10,20,30,40,50,60,70,80,90,100};

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

System.***out***.println(b[i]);

}

System.***out***.println("-----------------------------");

String s[]= {"kural","madhan","naveen","power"};

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

System.***out***.println(s[i]);

}

}

}

Ex:2, Largest element in an array

import java.util.Scanner;

public class ArrayScannerUser {

public static void main(String[] args) {

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

int a[] = new int[5];

System.***out***.println("enter array elements");

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

a[i] = sc.nextInt();

}

// System.out.println(a[1]);

int max = a[0];

int min = a[0];

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

if (a[i] > max) {

max = a[i];

}

if (a[i] < min) {

min = a[i];

}

}

System.***out***.println("biggest number = " + max);

System.***out***.println("smallest number = " + min);

}

}

Ex:3,Two Dimensional array

import java.util.Scanner;

public class TwoDimensionalArray {

public static void main(String[] args) {

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

//

//// System.out.println(a[1][2]);

// for(int i=0;i<a.length;i++) { //0<3 a[0][]

// for(int j=0;j<a[i].length;j++) { //0,1,2

// System.out.print(a[i][j]+" ");

// }

// System.out.println();

// }

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

System.***out***.println("enter no of rows");

int row = sc.nextInt();

System.***out***.println("enter no of column");

int col = sc.nextInt();

int a[][] = new int[row][col];

for (int i = 0; i < row; i++) { // or i<a.length

for (int j = 0; j < col; j++) {// or j<a.length

System.***out***.println("enter for " + i + "," + j);

a[i][j] = sc.nextInt();

}

}

System.***out***.println("resultant matrix");

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

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

System.***out***.print(a[i][j] + " ");

}

System.***out***.println();

}

}

}

Matrix addtion:

Condition:

Order of both matrices should be same

import java.util.Scanner;

public class ArrayMatricesAddtion {

public static void main(String[] args) {

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

System.***out***.println("enter no of rows of m1");

int row1 = sc.nextInt();

System.***out***.println("enter no of column of m1");

int col1 = sc.nextInt();

System.***out***.println("enter no of rows of m2");

int row2 = sc.nextInt();

System.***out***.println("enter no of rows of m2");

int col2 = sc.nextInt();

if ((row1 == row2) && (col1 == col2)) {

int a[][] = new int[row1][col1];

int b[][] = new int[row2][col2];

int c[][] = new int[row1][col1];

System.***out***.println("enter the matrix 1 elements");

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

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

System.***out***.println("enter for " + i + "," + j);

a[i][j] = sc.nextInt();

}

}

System.***out***.println("enter the matrix 2 elements");

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

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

System.***out***.println("enter for " + i + "," + j);

b[i][j] = sc.nextInt();

}

}

System.***out***.println("resultant matrix");

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

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

c[i][j] = a[i][j] + b[i][j];

System.***out***.print(c[i][j]+" ");

}

System.***out***.println();

}

// this is extra loop

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

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

// System.out.print(c[i][j]+" ");

// }

// System.out.println();

// }

}

else {

System.***out***.println("order of matrices should be same");

}

}

}

MATRIX Multiplication condition:

No of columns in matrix 1 and No of rows in matrix 2 should be same

Column1==row2

Ex: a=[1 2] b=[1 2 3]

[3 4] [4 3 6]

public class ArrayMatricesMultiplication {

public static void main(String[] args) {

int a[][] = { { 1, 2 }, { 3, 4 } };// 2\*2

int b[][] = { { 1, 2, 3 }, { 4, 5, 6 } };// 2\*3

// 2\*2 2\*3 when multiple matrix apply only 1st matrix col and 2nd matrix row will same

// column 1 == row 2

int c[][] = new int[a.length][b[0].length];

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

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

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

c[i][j] = c[i][j] + (a[i][k] \* b[k][j]);

}

}

}

System.***out***.println("resultant matrix");

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

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

System.***out***.print(c[i][j]+" ");

}

System.***out***.println();

}

}

}

Transpose of a matrix:

public class ArrayMatricesTranspose {

public static void main(String[] args) {

int a[][]= {{1,2,3},{4,5,6}};

int c[][]=new int[a[0].length][a.length];

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

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

c[i][j]=a[j][i];

System.***out***.print(c[i][j]+" ");

}

System.***out***.println();

}

}

}

Sum of diagonal elements of a matrix:

Condition:

Matrix should be a square matrix

(no of rows and no of columns should be same) 3x3

Ex: a=[1 2 3]

[4 5 6]

[7 8 9]

public class ArrayDiagonal {

public static void main(String[] args) {

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

// principal (main/leading) diagonal

// secondary diagonal

int pd = 0;

int sd = 0;

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

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

System.***out***.print(a[i][j] + " ");

if (i == j) {

pd = pd + a[i][j];

}

if (i + j == a.length - 1) {

sd = sd + a[i][j];

}

}

System.***out***.println();

}

System.***out***.println("primary diagonal");

System.***out***.println(pd);

System.***out***.println("secondary diagonal");

System.***out***.println(sd);

}

}

Enhanced for loop:

public class EnhancedForLoop {

public static void main(String[] args) {

// Enhanced For Loop / For-Each Loop

int a[] = { 1, 2, 3, 4, 5, 6 };

for (int value : a) {

System.***out***.print(value + " ");

}

System.***out***.println();

for (int value : a) {

System.***out***.println(value);

}

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

for (int temp[] : b) {

for (int value : temp) {

System.***out***.print(value + " ");

}

System.***out***.println();

}

}

}

Jagged Array:

Definition:

Jagged Array is two-dimensional array with varying column counts in each row. The size of each row is not fixed, allowing for a jagged structure.

[no of columns of each row will not be same]

Ex: a=[1 2]

[3 4 5]

import java.util.Scanner;

public class JaggedArray {

public static void main(String[] args) {

// int a[][]= {{1,2},{3,4,5}};

//

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

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

// System.out.print(a[i][j]+" ");

// }

// System.out.println();

// }

int a[][] = new int[2][];

a[0] = new int[2];

a[1] = new int[3];

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

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

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

System.***out***.println("enter for value" + i + "," + j);

a[i][j] = sc.nextInt();

}

}

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

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

System.***out***.print(a[i][j]+" ");

}

System.***out***.println();

}

}

}

Patterns in Java:

public class Patterns {

public static void main(String[] args) {

// 0123

// 0\*\*\*\*

// 1\*\*\*\*

// 2\*\*\*\*

// 3\*\*\*\*

// for(int i=0;i<4;i++) { // (rows)

// for(int j=0;j<4;j++) { // (columns)

// System.out.print("\*");

// }

// System.out.println();

// }

// 0123

// 0\*\*\*\*

// 1\* \*

// 2\* \*

// 3\*\*\*\*

// for(int i=0;i<4;i++) { // (rows)

// for(int j=0;j<4;j++) { // (columns)

// if(i==0 || i==3||j==0||j==3) {

// System.out.print("\*");

// }

// else {

// System.out.print(" ");

// }

// }

// System.out.println();

// }

//

// int row=4;

// int col=4;

// for(int i=0;i<row;i++) { // (rows)

// for(int j=0;j<col;j++) { // (columns)

// if (i == 0 || i == row - 1 || j == 0 || j == col - 1) {

// System.out.print("\*");

// }

// else {

// System.out.print(" ");

// }

// }

// System.out.println();

// }

// \*

// \*\*

// \*\*\*

// \*\*\*\*

int row=4;

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

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

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

// System.out.print(j+" ");

System.out.print("\*");

}

System.out.println();

}

}

}

Constructors and its types:

CONSTRUCTORS (method)

1.same name as class name

2.never return anything

3.used to allocate memory and initialize object

class abcd {

int num1, num2, result;

// default constructor

abcd() {

num1 = 10;

num2 = 20;

}

//Parameterized constructor

abcd(int a,int b){

num1=a;

num2=b;

}

void display() {

result = num1 + num2;

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

}

}

public class Constructor {

public static void main(String[] args) {

// CONSTRUCTORS (method)

//

// same name as class name

// never return anything

// used to allocate memory and initialize object

abcd obj = new abcd();

System.out.println("default constructor");

obj.display();

abcd obj1=new abcd(10,20);

System.out.println("parameterized constructor");

obj1.display();

}

}

Constructor overloading:

constructor overloading is same class name but different parameters

class abcd {

int num1, num2, result;

// default constructor

abcd() {

num1 = 10;

num2 = 20;

}

//Parameterized constructor

abcd(int a,int b){

num1=a;

num2=b;

}

// contructor overload

abcd(int a,int b,int c){

Num1=a;

Num2=b;

Num3=c;

}

}

Encapsulation:

getters and setters

->binding data members and methods into a single entity

class Demo{

// instance variables,methods

private int age=23;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

public class Encapsulation {

public static void main(String[] args) {

Demo obj=new Demo();

obj.setAge(50 );

System.out.println(obj.getAge());

}

}

Polymorphism:

--> compile time (method overloading)

--> run time (method overriding)

Method overloading:

class Overload {

int num1, num2, result;

void sum(int a, int b) {

num1 = a;

num2 = b;

result = num1 + num2;

System.out.println(result);

}

void sum(double a, double b) {

// num1=(int)a;

// num2=(int)b;

// result=num1+num2;

double num1 = a;

double num2 = b;

double result = num1 + num2;

System.out.println(result);

}

void sum(int a, double b) {

num1=a;

num2=(int)b;

result=num1+num2;

System.out.println(result);

}

}

public class MethodOverloading {

public static void main(String[] args) {

Overload obj = new Overload();

obj.sum(10, 20);

obj.sum(10.5, 10.572);

obj.sum(10, 10.5);

}

Method Overriding(Run time polymorphism):

Achieved by Inheritance concept.

Rules:

-> Both Super & sub class methods should be having same name

-> Static & final Methods cannot be Overridden

class parents{

void show() {

System.out.println("parent");

}

}

class childs extends parents{

void show() {

System.out.println("child");

}

}

public class Methodoverride {

public static void main(String[] args) {

childs c=new childs();

c.show();

}

}

Varargs:(variable length arugments)

class Addition {

public int add(int i, int j) {

return i + j;

}

public int add(int i, int j, int k) {

return i + j + k;

}

int sum = 0;

// variable length aruguments--> ...

public int add(int... i) {

// enchanced for loop

for (int val : i)

sum += val;

return sum;

}

// public int add(int a, int... i) {

// // enchanced for loop

// for (int val : i)

// sum += val;

// System.out.println(a);

// return sum;

// }

}

public class VariableAruguments {

public static void main(String[] args) {

Addition obj = new Addition();

int result = obj.add(1, 10);

System.out.println(result);

int result1 = obj.add(1, 10, 20);

System.out.println(result1);

int result2 = obj.add(1, 2, 3, 4, 5, 6);

System.out.println(result2);

}

}

Inheritance and its types:

1.Super(Base,parent)Class

2.sub(Derived,child)Class

Accessing variables and methods of super class by using sub class object

1.single ->parent to child

2.multilevel -> grandParent to parent to child

3.hierarchical-> parent to child1 and parent to child2

4.Multiple(not supported directly)

5.Hybrid(not support)

class GrandParent {

int a;

void display() {

System.out.println("Grandparent");

}

}

class parent extends GrandParent {

int b;

void show() {

System.out.println("parent");

}

}

class Child extends parent {

int c;

void print() {

System.out.println("child");

}

}

class child1 extends parent {

void show1() {

System.out.println("child1");

}

}

class child2 extends parent {

void show2() {

System.out.println("child2");

}

}

public class Inherit {

public static void main(String[] args) {

Child c = new Child();

c.a = 10;

c.b = 20;

c.c = 30;

System.out.println(c.a);

System.out.println(c.b);

System.out.println(c.c);

c.display();

c.show();

c.print();

System.out.println("hierarchical:");

child1 c1=new child1();

c1.show1();

c1.show();

System.out.println("--------");

child2 c2=new child2();

c2.show2();

c2.show();

}

}

This keyword:

class This1 {

// instance varible

int num1, num2;

void getData(int num1, int num2) {

// local variable

this.num1 = num1;

this.num2 = num2;

}

void getData1(int a, int b) {

// local variable

num1 = a;

num2 = b;

}

void display() {

System.out.println(num1);

System.out.println(num2);

}

}

public class ThisKeyword {

public static void main(String[] args) {

This1 a = new This1();

a.display();

a.getData(10, 20);

a.display();

a.getData1(20, 30);

a.display();

}

}

Final Keyword:

* Final (Variable , method , class)
* variable -> used to declare as constant
* method -> cannot be overridden
* class -> cannot be inherited

class a{

final void show(){

}

}

final class b extends a{

void show() {//error

}

}

class c extends b{//error

}

public class FinalKeyword {

public static void main(String[] args) {

final int a = 5;

// a=10; //error

System.out.println(a);

}

}

Super Keyword:

* Achieved by inheritance concept
* used to access immediate parent class variables,methods and constructors from child class

note: name of Base class and derived class variables and methods should be same

class parentfather {

// constructors

public parentfather() {

// super(); by default is present

System.out.println("in parent default");

}

public parentfather(int val) {

// super();

System.out.println("in parent parameterized " + val);

// super();//error

}

// methods

//variable

int a = 20;

public void display() {

System.out.println("in parent display");

}

}

class childson extends parentfather {

// constructors

public childson() {

// super(); by default is present

System.out.println("in child default");

}

public childson(int i) {

// super();// by default is present

super(10 + 2);

System.out.println("in child parameterized " + i);

}

// methods

// variable

int a = 10;

public void display() {

super.display();

System.out.println("in child display");

super.display();

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

System.out.println("a = " + super.a);

}

}

public class SuperKeyword {

public static void main(String[] args) {

childson obj = new childson();

childson obj1 = new childson(10);

obj.display();

}

}

Static Keyword:

class Students{

String name;

int age;

static String teacher;

void display(){

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

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

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

}

static {

Students.teacher="nanthani";

}

}

public class StaticKeyword {

public static void main(String[] args) {

Students s1=new Students();

Students s2=new Students();

s1.name="kural";

s1.age=23;

// s1.teacher="nanthani";//or

// Students.teacher="nanthani";

s2.name="kuralarasan";

s2.age=23;

// s2.teacher="karthi";

s1.display();

s2.display();

}

}

Methods:

Ex:1

class Welcome{

void greet(String name,int age) {

System.out.println("Welcome "+name+" age is "+age);

}

public int add(int a,int b) {

greet("kural",23);

return a+b;

}

}

public class Methods {

public static void main(String[] args) {

System.out.println("hi");

Welcome obj=new Welcome();

// obj.greet("kural", 23);

int op=obj.add(10, 20);

System.out.println(op);

}

}

Ex:2

//public class Methods1 {

// public static void displayFunction() {

// System.out.println("in display");

// }

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

// displayFunction();

// }

//} // or

public class Methods1 {

public void display() {

System.out.println("in display");

}

public static void main(String[] args) {

new Methods1().display();

}

}

Abstraction (abstract):

->Abstract method (method having only declaration)

->Definition will be written in Dervied class

->abstract class (contains atleast one abstract method)

->concreate class (class which doesn't contain any abstract method)

->we cannot create object for abstract class

abstract class Empolyee{

abstract void display();//declaration

void show() {

System.out.println("Abstract show");

}

}

class Servant extends Empolyee{ //derived class

void display() {

System.out.println("Abstract method in derived class");

}

}

public class Abstract {

public static void main(String[] args) {

Servant obj=new Servant();

obj.display();

obj.show();

}

}

Inner class: - a class inside a class.

There are four types of inner class

1. nested inner class
2. Method Local inner classes
3. Static nested classes
4. Anonymous inner classes

1.nested inner class:

class outer{

//variable

//methods

//constructors

//class

class inner{

int a=10;

public void show() {

System.out.println("in nested inner class show method");

}

}

}

public class NestedInnerClass {

public static void main(String[] args) {

outer out=new outer();

outer.inner obj=out.new inner();

obj.show();

}

}

2.Static nested inner classes:

class outers{

static class inner{

int a=10;

public void show() {

System.out.println("in static inner class show method");

}

}

static class inner1{

public void show() {

System.out.println("in static inner1 class show method");

}

}

}

public class StaticInnerClass {

public static void main(String[] args) {

outers.inner obj=new outers.inner();

obj.show();

outers.inner1 obj1=new outers.inner1();

obj1.show();

}

}

1. Method Local inner classes:

Note: the private inner class can only be accessed within the scope of the outer class.

Outside the outerclass, the private inner class is not accessible.

class Outers {

private class Inner{

public void show() {

System.out.println("in private inner show method ");

}

}

public void getAccess() {

class Inner {

public void show() {

System.out.println("in method local inner class ");

}

}

Inner in = new Inner();

in.show();

}

}

public class MethodLocalInnerClass{

public static void main(String[] args) {

Outers out = new Outers();

out.getAccess();

}

}

4.Anonymous inner classes:

class Test1 {

public void test() {

System.out.println("in test");

}

}

//class Test2 extends Test1{

// public void test() {

// System.out.println("in new test");

// }

//}

public class AnonymousInnerClass {

public static void main(String[] args) {

// Test2 t=new Test2();

// t.test();

// AnonymousInnerClass

Test1 t = new Test1() {

public void test() {

System.out.println("in new test");

}

};

t.test();

}

}

Abstract Anonymous inner class:

abstract class Abstracts {

public abstract void show();

}

//class Abstractext extends Abstracts {

//

// @Override

// public void show() {

// System.out.println("in show");

// }

//

//}

public class AbstractAnonymousInnerClass {

public static void main(String[] args) {

// Abstractext obj = new Abstractext();

// obj.show();

Abstracts obj=new Abstracts() {

@Override

public void show() {

System.out.println("in show Abstract Anonymous InnerClass");

}

};

obj.show();

}

}

Why multiple inheritance is not supported in java:to achieve (interfaces)

class a1{

public void display() {

System.out.println("in a1 display");

}

}

class a2{

public void display() {

System.out.println("in a2 display");

}

}

class a3 extends a1,a2{

}

//Mutiple Inheritance is not supported in java to achieve (interfaces)

public class MutipleInheritanceError {

public static void main(String[] args) {

a3 obj=new a3();

obj.display();

}

}

Interface:

// class extends class

// class implements interface

// interface extends interface

Interface Basics:

//abstract class b{

// abstract void display();

// void show() {

//

// }

//}

//difference between

interface Ab{

// variables - by default static and final

// methods - only abstract methods

// no contructors

int age=23;// static and final

void show();// by default - public abstract

void display();

}

class Aimp implements Ab{

@Override

public void show() {

// age=30; //error

System.out.println("in display");

}

@Override

public void display() {

System.out.println("in show");

}

}

public class Interface {

public static void main(String[] args) {

// Ab obj;

// obj=new Aimp();

//or

// Ab obj=new Aimp();

//or

Aimp obj=new Aimp();

obj.show();

obj.display();

System.out.println(Ab.age); // direct to call so it is static

}

}

Multiple inheritance with Interface:

interface parent1{

void display();

}

interface parent2{

void show();

}

class MyClass{

public void summa() {

System.out.println("in summa");

}

}

class childs1 extends MyClass implements parent1,parent2{

@Override

public void show() {

System.out.println("in show");

}

@Override

public void display() {

System.out.println("in display");

}

}

//We can achieve multiple inheritance with interface

public class MultipleInheritanceInterface {

public static void main(String[] args) {

childs1 obj=new childs1();

obj.display();

obj.show();

obj.summa();

}

}

Types of interface:

1.Normal Interface -> more than 1 method (>1)

2.Functional interface with lambda expression -> (1 method) - single abstract method (SAM)

3.Marker interface (serializable) (0 methods)-permission

// default and static method in interface

1. Normal interface:

interface MyInterface {

void display();

void show();

}

public class NormalInterface {

public static void main(String[] args) {

MyInterface obj = new MyInterface() {

@Override

public void display() {

System.out.println("in display anonymous interface");

}

// more then a method

public void show() {

System.out.println("in show");

}

};

obj.display();

obj.show();

}

}

1. Functional interface:(1 method) - single abstract method (SAM)

only Functional Interface we can write lambda expression

Ex:1

@FunctionalInterface

interface MyInterface {

void display();

// void show();// error because only 1 method

}

//class MyInterfaceImp implements MyInterface {

// @Override

// public void display() {

// System.out.println("in display");

//

// }

//}

public class FunctionalInterfaceDemo {

public static void main(String[] args) {

// reference //object

// MyInterface obj = new MyInterfaceImp();

// obj.display();

// MyInterface obj = new MyInterface() {

//

// @Override

// public void display() {

// System.out.println("in display anonymous interface");

// }

// };

// only FunctionalInterface we can write lambda expression

MyInterface obj = () -> {

System.out.println("in lambda expression");

};

obj.display();

}

}

Ex:2

@FunctionalInterface

interface MyInterface {

// void display(int i);

// void display(int i,int j);

int add(int i, int j);

}

public class FunctionalInterfaceDemo {

public static void main(String[] args) {

// (i)

// MyInterface obj = (int i) -> {

// System.out.println("in lambda expression " + i);

// };

// obj.display(10);

// MyInterface obj = (i,j) -> {

// System.out.println("in lambda expression " + i+" "+j);

// };

// obj.display(10,20);

// MyInterface obj = (i, j) -> {

// return i + j;

// }; //or

MyInterface obj = (i, j) -> i + j;

int result = obj.add(5, 10);

System.out.println(result);

}

}

1. Marker interface (serializable):(0 methods)-permission

interface permission {

}

class MyClass1 implements permission{

public void show() {

System.out.println("in show method");

}

}

public class MarkerInterface {

public static void main(String[] args) {

MyClass1 obj = new MyClass1();

if (obj instanceof permission) {

obj.show();

} else {

System.out.println("no access granted");

}

}

}

Default and static method in interface:

Abstract class - declare and define methods

Interface - only method declaration

interface wierd {

default void show() {

System.out.println("in show default method");

}

static void display() {

System.out.println("in static method");

}

}

class wierdImp implements wierd {

static void display() {

System.out.println("in static method 1");

}

}

public class DefaultAndStaticMethodInterface {

public static void main(String[] args) {

// default

wierdImp obj = new wierdImp();

obj.show();

// static

wierd.display();

wierdImp.display();

}

}

Enum:

1.Enum (Group of named constants) Basics

2.Comparing constants using if else and switch

3.Enum class

enum shape {

// 0 1 2

CIRCLE(0), TRIANGLE(3), SQUARE(4);

private int slides;

private shape() {

System.out.println("constructors called");

}

private shape(int i) {

slides = i;

}

public int getSlides() {

return slides;

}

}

public class Enum {

public static void main(String[] args) {

// 1.Enum (Group of named constants) Basics

// 2.Comparing constants using if else and switch

// 3.Enum class

shape obj = shape.CIRCLE;

System.out.println(obj);

shape obj1 = shape.SQUARE;

System.out.println(obj1);

System.out.println(obj1.ordinal());// index value

shape[] obj2 = shape.values();

for (shape val : obj2) {

System.out.println(val);

}

if (obj == shape.CIRCLE) {

System.out.println("no side");

} else if (obj == shape.SQUARE) {

System.out.println("four sides");

} else if (obj == shape.TRIANGLE) {

System.out.println("three sides");

} else {

System.out.println("wrong");

}

switch (obj1) {

case CIRCLE:

System.out.println("no side");

break;

case SQUARE:

System.out.println("four sides");

break;

case TRIANGLE:

System.out.println("three sides");

break;

default:

System.out.println("wrong");

}

System.out.println(obj.getClass());

System.out.println(obj.getClass().getSuperclass());

System.out.println(obj.getSlides());

System.out.println(obj1.getSlides());

}

}

Exceptional handling mechanism:

Ex:1

public class ExceptionHandling {

public static void main(String[] args) {

// try,catch,throw,throws,finally

try {

int a = 5 / 1;

System.out.println("no EXception "+a);

int b = 5 / 0;//this is Exception

System.out.println("Exception "+b);

} catch (Exception e) {

System.out.println("Exception " + e);

}

System.out.println("hi");

}

}

Ex:2

public class ExceptionHandling1 {

public static void main(String[] args) {

try {

int a = 5 / 1;

System.out.println("no EXception of a is " + a);

try {

int c[] = { 2 };

c[0] = 10;

System.out.println("no exception " + c[0]);

c[10] = 35;// exception

System.out.println("inner try");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("inner try exception " + e);

}finally {

System.out.println("inner finally bye");

}

} catch (Exception e) {

System.out.println("Exception " + e);

}

finally {

System.out.println("finally bye");

}

}

}

Ex:3 throw

public class ExceptionHandlingthrow {

public static void main(String[] args) {

try {

int a=5/0;

// throw is used to raised a exception if remove a also exception

throw new ArithmeticException();

}catch (Exception e) {

System.out.println("exception "+e);

}

}

}

Ex:4 throws

public class ExceptionHandlingthrows {

public void throwsDemo() throws NullPointerException {

String str=null;

System.out.println(str.length());

}

public void slave() {

try {

throwsDemo();

}catch (NullPointerException e) {

System.out.println("Exception "+e);

}

}

public static void main(String[] args) {

ExceptionHandlingthrows obj=new ExceptionHandlingthrows();

obj.slave();

}

}

User input using BufferedReader:

public class UserInputUsingBufferedReader {

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

// InputStreamReader isr = new InputStreamReader(System.in);

// BufferedReader br = new BufferedReader(isr);

// or

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

// System.out.println("Enter any message str");

// String str = br.readLine();

// System.out.println("your message " + str);

//

// System.out.println("enter your int num");

// int num=Integer.parseInt(br.readLine());

// System.out.println(num+2);

System.out.println("enter your double num");

double num1 = Double.parseDouble(br.readLine());

System.out.println(num1);

}

}

what is MultiThreading? - one process do and do another process

why we need threads? - subprocess - threads implementing Multithreading in java by

1. using Thread class

2. using Runnable Interface

Thread:

Individual unit of a process

1. using Thread class

Ex:1

class hi extends Thread {

public void show() {

for (int i = 0; i < 5; i++) {

System.out.println("hi");

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class hello extends Thread {

public void show() {

for (int i = 0; i < 5; i++) {

System.out.println("hello");

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class MultiThreadingUsingThreadClass {

public static void main(String[] args) {

hi obj = new hi();

hello obj1 = new hello();

obj.show();

obj1.show();

}

}

Ex:2

class hi extends Thread { // thread inside interface run() method

@Override

public void run() { // so we use run

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

System.out.println("hi");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class hello extends Thread {

@Override

public void run() {

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

System.out.println("hello");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class MultiThreadingUsingThreadClass {

public static void main(String[] args) {

// hi obj = new hi();

// hello obj1 = new hello();

// obj.start();

// obj1.start();

hi obj = new hi();

hello obj1 = new hello();

obj.start();

try {

Thread.sleep(500);

} catch (InterruptedException e) {

}

obj1.start();

}

}

1. using Runnable Interface (Functional interface) - Lambda expression

Ex:1

//class hi implements Runnable {

// @Override

// public void run() {

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

// System.out.println("hi");

// try {

// Thread.sleep(1000);

// } catch (InterruptedException e) {

// e.printStackTrace();

// }

// }

// }

//}

//

//class hello implements Runnable {

// @Override

// public void run() {

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

// System.out.println("hello");

// try {

// Thread.sleep(1000);

// } catch (InterruptedException e) {

// e.printStackTrace();

// }

// }

// }

//}

//

//public class MultiThreadingUsingRunnableInterface {

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

//// implementing Multithreading in java by

////// 1. using Thread class

////// 2. using Runnable Interface

//// (Functional interface) - Lambda expression

//

// Runnable obj = new hi();

// Runnable obj1 = new hello();

// Thread t1=new Thread(obj);

// t1.start();

// try {

// Thread.sleep(500);

// } catch (InterruptedException e) {

// }

// Thread t2=new Thread(obj1);

// t2.start();

//

// }

//}

//public class MultiThreadingUsingRunnableInterface {

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

//// Runnable obj = new Runnable() {

//// @Override

//// public void run() {

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

//// System.out.println("hi");

//// try {

//// Thread.sleep(1000);

//// } catch (InterruptedException e) {

//// }

//// }

//// }

//// };

//// Runnable obj1 = new Runnable() {

//// @Override

//// public void run() {

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

//// System.out.println("hello");

//// try {

//// Thread.sleep(1000);

//// } catch (InterruptedException e) {

//// }

//// }

//// }

//// };

// // or Lambda expression

// Runnable obj = () -> {

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

// System.out.println("hi");

// try {

// Thread.sleep(1000);

// } catch (InterruptedException e) {

// }

// }

// };

// Runnable obj1 = () -> {

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

// System.out.println("hello");

// try {

// Thread.sleep(1000);

// } catch (InterruptedException e) {

// }

// }

// };

//

// Thread t1 = new Thread(obj);

// t1.start();

// try {

// Thread.sleep(500);

// } catch (InterruptedException e) {

// }

// Thread t2 = new Thread(obj1);

// t2.start();

//

// }

//}

//or

public class MultiThreadingUsingRunnableInterface {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

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

System.out.println("hi");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

}

});

t1.start();

try {

Thread.sleep(500);

} catch (InterruptedException e) {

}

Thread t2 = new Thread(() -> {

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

System.out.println("hello");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

}

});

t2.start();

}

}

Thread class methods:

//1.join()->last sentence join 2.isAlive()->boolean

//3.getName() 4.setName()

//5.getPriority() 6.setPriority() 7.sleep()

public class ThreadClassMethods {

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

Thread t1 = new Thread(() -> {

for (int i = 0; i < 5; i++) {

System.out.println("hi");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

}

}, "First Thread");// set name we can declare here also

t1.start();

try {

Thread.sleep(500);

} catch (InterruptedException e) {

}

Thread t2 = new Thread(() -> {

for (int i = 0; i < 5; i++) {

System.out.println("hello");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

}

});

t2.start();

// System.out.println(t1.isAlive());//true

// t1.setName("thread 1");

// System.out.println(t1.getName());

// t1.setPriority(Thread.MAX\_PRIORITY);

// System.out.println(t1.getPriority()+" priority");

t1.join();

// System.out.println(t1.isAlive());//false

t2.join();

System.out.println("good evening");

}

}

Thread Synchronization:

class Counter {

private int count;

public synchronized void increment() {

count = count + 1;

}

public int getCount() {

return count;

}

}

public class ThreadSynchronization {

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

// (Thread safe (synchronized keyword))

Counter c = new Counter();

Thread t1 = new Thread(() -> {

for (int i = 0; i < 5000; i++) {

c.increment();

}

});

t1.start();

Thread t2 = new Thread(() -> {

for (int i = 0; i < 5000; i++) {

c.increment();

}

});

t2.start();

t1.join();

t2.join();

System.out.println(c.getCount());

}

}

Strings:

Ex:1

// stack and heap memory in jvm

public class Strings {

public static void main(String[] args) {

String str=new String("abcd");

System.out.println(str);

String s="hello";

System.out.println(s);

// string is object immutable

s="abcd";

System.out.println(s);

String s1="abcd";

System.out.println(s1);

}

}

Ex:2 String Class Methods:

public class StringClassMethods {

public static void main(String[] args) {

String s = "hello";

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

// indexof

System.out.println(s.indexOf('e'));

// charAt(index)

System.out.println(s.charAt(0));

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

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

System.out.println(s.concat("world"));

s = s + "world";

System.out.println(s);

String s1="hello";

String s2="hello";

System.out.println(s1.compareTo(s2));

// 0 -> String content are equal

// +ve

// -ve

String s3=" hello ";

s3=s3.trim();

System.out.println(s3.length());

String s4="hello world";

s4=s4.replace('e', 'o');

System.out.println(s4);

String s5="kuralarasan";

System.out.println(s5.substring(5));

System.out.println(s5.substring(0, 5));

System.out.println(s5.subSequence(0, 5));

System.out.println(s5.contains("kural"));

System.out.println(s5.lastIndexOf('a'));

String s6="abc";

String s7="ABC";

System.out.println(s6.equals(s7));

System.out.println(s6.equalsIgnoreCase(s7));

String s8="i,wanna,you";

String ss[]=s8.split(",");

for(String temp:ss) {

System.out.println(temp);

}

String sing="aeiou";

char ch[]=sing.toCharArray();

for(char c:ch) {

System.out.println(c);

}

String rev=String.valueOf(ch);

System.out.println(String.valueOf(ch));

}

}

Ex:3

public class Stringsdifference {

public static void main(String[] args) {

// "==" vs ".equals()" method

String s1="hello";

String s2="hello";

System.out.println(s1.equals(s2));

System.out.println(s1==s2);

String s3="hello";

String s4=new String("hello");

System.out.println(s3.equals(s4));

System.out.println(s3==s4);

}

}

StringBuffer and StringBuilder:

->Mutable strings

->both are same

->StringBuilder - non thread safe

->StringBuffer - thread safe

public class StringBufferDemo {

public static void main(String[] args) {

StringBuffer obj=new StringBuffer("hello");

StringBuilder obj1=new StringBuilder("world");

obj=obj.append(" world");

System.out.println(obj);

// System.out.println(obj.reverse());

System.out.println(obj.length());

System.out.println(obj.substring(2));

System.out.println(obj.substring(2,4));

obj.insert(0, "user ");

System.out.println(obj);

obj.delete(2, 4);

System.out.println(obj);

obj.replace(1,2,"y");

System.out.println(obj);

obj.setCharAt(0, 'j');

System.out.println(obj);

obj1.insert(0, "user ");

System.out.println(obj1);

}

}

Wrapper class:

boxing, auto-boxing, unboxing, auto-unboxing

public class WrapperClass {

public static void main(String[] args) {

// primitive data types int,double,char

// Integer,Double,Character

int i = 5;

System.out.println(i);

Integer obj = new Integer(5); // boxing

System.out.println("boxing "+obj);

Integer obj1 = 7; //auto-boxing

Integer obj2 = i;

System.out.println("auto-boxing "+obj1);

System.out.println("auto-boxing "+obj2);

int val=obj2.intValue();//unboxing

int val2=obj2;//auto-unboxing

System.out.println("unboxing "+val);

System.out.println("auto-unboxing "+val2);

System.out.println("-------------------------------");

double d=18.37;

Double dd=d;

System.out.println("auto boxing "+dd);

double dummy=dd.doubleValue();

double dummy1=dd;

System.out.println("unboxing "+dummy);

System.out.println("auto-unboxing"+dummy1);

}

}

Collection and generics Introduction:

Iterable this color-> interface

Collection

1. list 2.queue 3.set

->Array list ^ ->priority Queue ->hashset

->Linked list -------------|Deque sortedset

->Vector ->treeset

->stack this color -> class

Map

->hashmap

->hashTable

import java.util.ArrayList;

import java.util.Collection;

public class CollectionsFramework {

public static void main(String[] args) {

// collection and generics

// collection API / collection Framework

int a=5;

int arr[]= {99,29,48,48,92};

// Collection values=new Collection();//error because of interface

//generics

Collection<Integer> values=new ArrayList<Integer>();

values.add(10);

values.add(20);

values.add(30);

values.add(10);

// values.add("hello");// error because of Integer generics

System.out.println(values);

for(int val:values) {

System.out.println(val\*2);

}

System.out.println("-----------------");

System.out.println("list interface");

//remove method

while(values.contains(10)) {

values.remove(10);

}

System.out.println(values.contains(10));

System.out.println(values);

}

}

Ex:1

import java.util.ArrayList;

import java.util.Collection;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;

public class ListInterface {

// list-ArrayList and LinkedList

public static void main(String[] args) {

// List<Integer> values=new ArrayList<Integer>(); // or

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

values.add(10);

values.add(20);

values.add(30);

values.add(11);

values.add(9);

values.add(1, 8);

Collections.sort(values);

System.out.println("sort "+values);

System.out.println("size "+values.size());

values.set(0,11);

System.out.println("get "+values.get(0));

System.out.println("set "+values);

// Collection -> value

// ArrayList -> index

values.remove(0);

System.out.println("remove "+values);

System.out.println("value 20 index of "+values.indexOf(20));

values.remove(values.indexOf(20));

System.out.println("remove index of 20 "+values);

values.add(40);

Iterator it=values.iterator();

System.out.println("iterator "+it.next());

System.out.println("iterator "+it.next());

while(it.hasNext()) {

System.out.println(it.next());

}

for(int i:values) {

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

}

}

}

Ex:2

Set interface - unordered and unique elements ex: 10,20,10 output: 10,20

1.HashSet - unsorted order

2.TreeSet- to get sorted order

DrawBack: no index number

import java.util.HashSet;

import java.util.Set;

import java.util.TreeSet;

public class SetInterface {

public static void main(String[] args) {

Set<Integer> s=new HashSet<Integer>();

s.add(10);

s.add(20);

s.add(30);

s.add(34);

s.add(23);

s.add(10);

System.out.println("HashSet "+s);

Set<Integer> tree=new TreeSet<Integer>();

tree.add(10);

tree.add(20);

tree.add(30);

tree.add(34);

tree.add(23);

tree.add(10);

System.out.println("TreeSet "+tree);

}

}

Ex:3

Map Interface - <key,value> pair

1.HashMap - non synchronized

2.HashTable - synchronized

3.TreeMap

right side Genetics is not important

import java.util.HashMap;

import java.util.Hashtable;

import java.util.Map;

import java.util.TreeMap;

public class MapInterface {

public static void main(String[] args) {

// <key,value> rules follow <set,List>

// key is unordered

// value is ordered

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

hashmap.put("kural", 100);

hashmap.put("raghul", 80);

hashmap.put("don", 10);

hashmap.put("murgan", 100);

hashmap.put("kural", 97);

hashmap.put(null, null);

System.out.println("HashMap " + hashmap);

System.out.println(hashmap.get("kural"));

System.out.println("size "+hashmap.size());

System.out.println("key "+hashmap.keySet());

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

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

}

System.out.println(hashmap.values());

System.out.println(hashmap.containsKey("kural"));

System.out.println(hashmap.containsValue(100));

System.out.println("remove "+hashmap.remove("kural"));

System.out.println(hashmap);

System.out.println("---------------------------------------------");

Map<String, Integer> hashtable = new Hashtable<String, Integer>();

hashtable.put("kural", 100);

hashtable.put("raghul", 80);

hashtable.put("don", 10);

hashtable.put("murgan", 100);

hashtable.put("kural", 97);

// hashtable.put(null, null); // error

System.out.println("HashTable " + hashtable);

System.out.println(hashtable.get("kural"));

System.out.println("size "+hashtable.size());

System.out.println("key "+hashtable.keySet());

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

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

}

System.out.println(hashtable.values());

System.out.println(hashtable.containsKey("kural"));

System.out.println(hashtable.containsValue(100));

System.out.println("remove "+hashtable.remove("kural"));

System.out.println(hashtable);

System.out.println("-----------------------------------------");

// key is ordered

// value is unordered

Map<String, Integer> treemap = new TreeMap<String, Integer>();

treemap.put("kural", 100);

treemap.put("raghul", 80);

treemap.put("don", 10);

treemap.put("murgan", 100);

treemap.put("kural", 97);

treemap.put("abc", 97);

// treemap.put(null, null);//error

System.out.println("TreeMap " + treemap);

}

}

Access Modifiers:

1. Public - allow all class
2. Private - within a class
3. Default- within a package
4. Protected - non child class of outside package not allowed.

Date Time API:

public class DateTimeAPI {

public static void main(String[] args) {

LocalDate ld=LocalDate.now();//human

System.out.println(ld);

System.out.println(ld.getYear());

System.out.println(ld.getMonth());

System.out.println(ld.getDayOfMonth());

Instant i=Instant.now();//machine

System.out.println(i);

LocalTime lt=LocalTime.now();

System.out.println(lt);

LocalDateTime ldt=LocalDateTime.now();

System.out.println(ldt);

// for(String val:ZoneId.getAvailableZoneIds()) {

// System.out.println(val);

// }

}

}