**Java Class**

**Contents:**

**Basic Core Java:**

1. Basic knowledge of JAVA ------------------------------------------------------------------------------------------------------- 003

2. Features of Java ------------------------------------------------------------------------------------------------------------------ 003

3. Internal Details of Hello Java Program -------------------------------------------------------------------------------------- 003

4. Difference between JDK, JRE and JVM ------------------------------------------------------------------------------------ 003

5. Variable and Datatype in Java ------------------------------------------------------------------------------------------------ 005

6. Operators in java ----------------------------------------------------------------------------------------------------------------- 010

7. Control Statements -------------------------------------------------------------------------------------------------------------- 014

8. Read Input From User ---------------------------------------------------------------------------------------------------------- 020

9. Array --------------------------------------------------------------------------------------------------------------------------------- 021

10. String Function ----------------------------------------------------------------------------------------------------------------- 025

**OOPS Concepts:**

1. Basic knowledge of OOPS ------------------------------------------------------------------------------------------------ 028

i) Naming conventions

ii) Object and Class

1. METHOD and Method Overloading ------------------------------------------------------------------------------------ 029
2. MEMORY ALLOCATION ---------------------------------------------------------------------------------------------- 035
3. Blocks (static & non-static) ---------------------------------------------------------------------------------------------- 038
4. Constructor and Constructor Overloading ----------------------------------------------------------------------------- 041
5. this keyword & static keyword ------------------------------------------------------------------------------------------- 051
6. Call by Value and Call by Reference ------------------------------------------------------------------------------------ 053
7. Static and non-static reference variable --------------------------------------------------------------------------------- 058
8. Composition(Has a Relationship) --------------------------------------------------------------------------------------- 059
9. INHERITANCE(Is a Relationship) -------------------------------------------------------------------------------------- 060
10. Constructor chaining ------------------------------------------------------------------------------------------------------- 064
11. Method Overriding --------------------------------------------------------------------------------------------------------- 069
12. TYPE CASTING(Up, Down, instanceOf) ------------------------------------------------------------------------------ 071
13. Polymorphism -------------------------------------------------------------------------------------------------------------- 079
14. ABSTRACTION(Abstract class, Interface) ---------------------------------------------------------------------------- 083
15. Loose-Coupling ------------------------------------------------------------------------------------------------------------ 093
16. Packages -------------------------------------------------------------------------------------------------------------------- 095
17. Encapsulation -------------------------------------------------------------------------------------------------------------- 097
18. Library ---------------------------------------------------------------------------------------------------------------------- 100
19. String class --------------------------------------------------------------------------------------------------------------- 103
20. Array (class type) ------------------------------------------------------------------------------------------------------- 105

**Collection:**

1. Basic of Collection ----------------------------------------------------------------------------------------------------- 113
2. Type of Collection------------------------------------------------------------------------------------------------------ 114
3. Collection Interface
4. List interface -------------------------------------------------------------------------------------------------- 115
5. Set interface --------------------------------------------------------------------------------------------------- 119
6. Queue interface ----------------------------------------------------------------------------------------------- 131

**Exception:**

1. Basic of Exception ---------------------------------------------------------------------------------------------------- 137
2. Type of Exception ---------------------------------------------------------------------------------------------------- 137
3. Try Catch -------------------------------------------------------------------------------------------------------------- 138
4. Finally block ---------------------------------------------------------------------------------------------------------- 138
5. Throw throws -------------------------------------------------------------------------------------------------------- 139
6. Custom exception ---------------------------------------------------------------------------------------------------- 140

**Thread:**

1. Basic of Thread ------------------------------------------------------------------------------------------------------- 143
2. Life cycle of a Thread (Thread States) ----------------------------------------------------------------------------- 143
3. Thread Scheduler in Java -------------------------------------------------------------------------------------------- 144
4. How to create thread ------------------------------------------------------------------------------------------------- 144
5. Sleep() method of Thread class ------------------------------------------------------------------------------------- 145
6. Daemon Thread ------------------------------------------------------------------------------------------------------- 146
7. Java Thread Pool ------------------------------------------------------------------------------------------------------ 147
8. Synchronization ------------------------------------------------------------------------------------------------------- 148
9. Deadlock --------------------------------------------------------------------------------------------------------------- 148
10. Inter-thread communication in Java ( wait(), notify(), notifyAll() ) ------------------------------------------- 148

**Map:**

1. Hash Map --------------------------------------------------------------------------------------------------------------- 149

**Basic Knowledge of programming.**

**JVM** Execution If error it call Runtime error

Codding Source file (abc.java)

======

Compiling 1.Check Syntax 2.Rule check 3. Translate If error its call compile time error

Byte code class file (abc.class)

Jre

* A.java creating with any OS then compiles it and create byte code .class file and then copy it to any other device or OS like mac, Unix, solarize to run it but one thing remember that every os must have jre ….
* Java program should be return the syntax java statements and it should be save as “.java”.
* The “.java” file is known as source file.
* The java compiler is used to compile the source code to the byte code. Before compilation compiler check the syntax and the rules of java language if the source code is not following the syntax and rules then compiler those error which is known as compile time error (CTE).
* The compiler generate “.class” file which content the byte code format. “.class” file is executable file in java language
* Jvm is responsible for executing the “.class file”, it is an interpreter which introvert the byte codes and execute it to provide the result.
* Jre stands for java ran time environment provides an environment for jvm to execute the “.class file” .Jre is system library which interacts with os but executing the program.
* The “.class” file can be executed on any os provided in that os jre is installed. Hence java is known as platform independent.
* “.class” file is jre dependent but independent of os.
* Jre is develop separately for each os

You can write java source file with Text Editor-notepad, edit+, notepad ++. Or IDE (Integrated Development Environment) –eclipse, net beans.

**Java edition:**

1. Java SE- standard edition-(desktop edition)
2. Java EE-enterprise edition –(web app)
3. Java ME- Micro edition-(embedded application)

**Features of Java:**

* **Simple:** syntax is based on C++ (so easier for programmers to learn it after C++).  removed many confusing and/or rarely-used features e.g., explicit pointers
* **Object-Oriented:** Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
* **Platform independent:** Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).
* **Secured:** I) No explicit pointer

ii) Programs run inside virtual machine sandbox.

* **Robust:** Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling.
* **Architecture neutral:** There is no implementation dependent features e.g. size of primitive types is set.
* **Portable:** We may carry the java bytecode to any platform.
* **High Performance:** Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)
* **Multithreaded:** We can write Java programs that deal with many tasks at once by defining multiple threads. Threads are important for multi-media, Web applications etc.
* **Distributed:** We can create distributed applications in java. We may access files by calling the methods from any machine on the internet.
* Etc. ….

public class Program1 {----(class declaration)

public static void main(String[] args) {-------->(main method)

System.out.println("Hi Three tech");----print something.

}

}

**Compilation (command prompt)**

C:\Users\Programmer>D:

D:\>cd D:\JECM3\_1\BASICS\INTRODUCTION

D:\JECM3\_1\BASICS\INTRODUCTION>javac Program1.java

D:\JECM3\_1\BASICS\INTRODUCTION>java Program1

Ex,

class Program2 { public static void main(String[] args) { System.out.println("Jspider");//String System.out.println(12345);//intiger System.out.println(123.34);//floating point System.out.println(true);Boolean System.out.println('j');//character

} }

**Output is:**

Jspider 12345 123.34 true j

Ex,

+numaricaddition

+stringconcatination

class Program3 {

public static void main(String[] args) {

System.out.println(20+20);//addition

System.out.println("java"+"program");//string concatination

}

}

**Output:**

40

Javaprogram

*Ex, of operator association*

class Program4 {

public static void main(String[] args) {

System.out.println("Hello World!" +20);

System.out.println("Hello World!" +20+20);

System.out.println("Hello World!" +(20+20));//after string it will concat,it is based on the operator

System.out.println(20+"Hello World!");

System.out.println(20+20+"Hello World!");

}

}

**Output is:**

Hello World!20

Hello World!2020

Hello World!40

20Hello World!

40Hello World!

**Note:**

**Values:**

**1) Numeric:**

1. Integer
2. Float point

**2) Character:**  ‘ ‘

**3) Boolean:** true or false

**4) String:**  “ “

**Keywords:**

* Predefined words
* Has its own meaning
* Reserved words
* Lower case

E.g. public, void

**Identifier:**

* Name given by programmer
* Alpha numeric, should begin with alphabet
* Special underscore is allowed

**Variable name/ method name:**

* First later should be in lower case
* Subsequent words should begin with uppercase

Exam, empId; createSavingPoint

**Class name:**

* First later of each word should be in uppercase

Exam, AddEmploy, DeleteEmploy

**Variable declaration**

|  |  |
| --- | --- |
| **Data types** | **Variable Name** |
| Byte (8bit) |  |
| Int(4byte) | Integer number |
| Long(8byte) |  |
| Short(2byte) |  |
|  |  |
| Float (4byte) | Floating Number |
| Double(8byte) |  |
|  |  |
| Char(2byte) | Character |
|  |  |
| Boolean(1bit) | Boolean values |

Note: String is not a data type its class type object

**Variable initialization:**

Variable name= value;

Ex,

emId = 1856

empSalary = 12500.50

empGrade = ‘A’

Memory

|  |  |
| --- | --- |
| Location Id | Memory Location |
| emId: | [1856] |
| empSalary: | [12500.50] |
| empGrade: | [‘A’] |

**Declaration, Initialization:**

Ex,

class pro1 {

public static void main(String[] args) { //variable declaration int empId; double empSalary; char empGrade;

//variable initialization empId=12649; empSalary=1234.500; empGrade='A';

System.out.println("empId "+empId); System.out.println("empSalary "+empSalary); System.out.println("empGrade "+empGrade);

} }

Output is:

empId 12649 empSalary 1234.5 empGrade A

Ex,

class pro1 { public static void main(String[] args) {

//variable declaration

int empId;

double empSalary=1234.500;

char empGrade='A';

System.out.println("empId "+empId); //use the it but not initialize it so it gets error

System.out.println("empSalary "+empSalary);

System.out.println("empGrade "+empGrade);

}

}

Output:

Error

**Note,**

In java language all local variables should be initialized before using in any of the operation. Other ways compiler goes error.

**Ex, of re-assignment**

class pro2 { public static void main(String[] args) {

int n1=12; System.out.println("n1 " +n1);

n1=23;//re-assign System.out.println("n1 " +n1);

n1=45;//re-assign System.out.println("n1 " +n1); } }

Output:

n1 12 n1 23 n1 45

**Ex, Final value:**

class pro3{ public static void main(String[] args) {

final int n1=34;//value assigned will be final System.out.println("n1 value: " +n1);

n1=45;//error, cannot re-assign value to final variable System.out.println("n1 value: " +n1); } }

Note, in java language constant can be achieved declaring as final.

Whenever variable declaring as final we can initialized only once but we cannot re-initialized in the code again.

**Ex. of Normal value which we can reassign**

class pro4 { public static void main(String[] args) { int n1=25; int n2=45; int n3=0;

System.out.println("n1 value !"+n1); System.out.println("n1 value !"+n2); System.out.println("n1 value !"+n3);

n2=n1;//copy value of n1 to n2 n3=n2;//copy value of n2 to n3

System.out.println("n1 value !"+n1); System.out.println("n1 value !"+n2); } }

Output is:

n1 value !25 n1 value !45 n1 value !0

n1 value !25 n1 value !25 n1 value !25

**Ex, Use of final value or where we can use the final value:**

class pro6{ public static void main(String[] args) {

double r1=2.1; final double pi=3.14; double a1,c1;

a1=pi\*r1\*r1; c1=2\*pi\*r1;

System.out.println("area : "+a1); System.out.println("cercun : "+c1);

} }

Output:

area : 13.8474 cercun : 13.188

**Ex, long and float value**

class pro5 { public static void main(String[] args) {

long empPhNum=999999999999999l; float empsal=12334335.34f;

System.out.println("emp ph no "+empPhNum); System.out.println("emp sal no "+empsal);

} }

Note, if u not use “l” in the last in long value then it getting error because of default value is integer in java

And also float value “f” in default in java is double value

**Use of Unary Operator: ++ or –**

# Difference between post increment (n++) and pre increment(++n)

**Post Increment(n++)** **:** First execute the statement then increase the value by one.  
**Pre Increment (++n) :** First increase the value by one then execute the statement.

**Example:**

|  |  |
| --- | --- |
|  | class IncrementTest{      public static void main(String[] args){            System.out.println("\*\*\*Post increment test\*\*\*");          int n = 10;          System.out.println(n);      // output  10          System.out.println(n++);    // output  10          System.out.println(n);      // output  11            System.out.println("\*\*\*Pre increment test\*\*\*");          int m = 10;          System.out.println(m);      // output  10          System.out.println(++m);    // output  11          System.out.println(m);      // output  11      }  } |

Int K=0;

K++=post increment

++k=pre increment

Ex,

class Pro7 { public static void main(String[] args) {

int k=0; k++; System.out.println("after incr: "+k); k--; System.out.println("after dec: "+k);

} }

class Pro8 { public static void main(String[] args) {

int i=0; int j=0;

j=i++; System.out.println("i value "+i); System.out.println("j value "+j);

} }

Output is:

i value 1 j value 0

Ex, class Pro8 { public static void main(String[] args) {

int i=0; int j=0;

j=i + i++; //0 + 0 // use current value of i and than increment value of i by 1

System.out.println("i value "+i); System.out.println("j value "+j);

} }

Output is:

i value 1 j value 0

Ex, class Pro8 { public static void main(String[] args) {

int i=0; int j=0;

j=I + i++ + i; //0 + 0 +1 // use current value of i and then increment value of i by 1

System.out.println("i value "+i); System.out.println("j value "+j);

} }

Output is:

1 1

Ex,

class Pro9 { public static void main(String[] args) {

int i=0; int j=0;

j=++I; //1 // first increment value of i by 1 and use the increment value

System.out.println("i value "+i); System.out.println("j value "+j);

} }

Output is:

1 1

Ex,

class Pro9 {

public static void main(String[] args) {

int i=0; int j=0;

j=++i + ++i; //1+2 first increment value of i by 1 and use the increment value

System.out.println("i value "+i); System.out.println("j value "+j); } }

Output 2 3

Ex,

class pro9 {

public static void main(String[] args) {

int i=0; int j=0;

j=i + ++i + i++ + i; //0+ 1 + 1 + 2

System.out.println("i value "+i); System.out.println("j value "+j); } }

Output is: 2 4

Ex, class pro9 { public static void main(String[] args) {

int i=0; int j=0;

j=++i + ++i + i + i++ + ++i; // 1+ 2+ 2+ 2 + 4

System.out.println("i value "+i); System.out.println("j value "+j);

}

}

Output is:

4 11

**Methods:**

* To implement a task /operation
* Reusability--- write once /run many times

**Syntax** <Modifier> retuntype methodname (<argument>)

{ Function Return value; }

Discus more in OPPS concept continue ………….

**Control Statements:**

**Switch Case:**

**Ex,**

**class** Pro5 {

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

**char** grade='A';

**switch**(grade){

**case** 'A': System.*out*.println("first class with distinction");

**break**;

**case** 'B': System.*out*.println("first class");

**break**;

**case** 'C': System.*out*.println("Second class");

**break**;

**case** 'D': System.*out*.println("Just pass");

**break**;

**case** 'E': System.*out*.println("Get Lost");

**break**;

**default**: System.*out*.println("Invalid grade");

}

}

}

Out put:

first class with distiction

Note: switch case jump to the correct statement. But if is test one by one check condition statement.

**For Loop:**

**class** Pro6 {

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

**for** (**int** i=1; i<=5; i++){

System.*out*.println("I Love Programming");

}

}

}

o/p:

I Love Programming

I Love Programming

I Love Programming

I Love Programming

I Love Programming

*Ex, of Increment.*

**class** Pro6 {

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

**for** (**int** i=1; i<=5; i++){ System.*out*.println(i);

}

}

}

o/p:

1

2

3

4

5

*Ex, of decrement:*

**class** Pro6 {

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

**for** (**int** i=5; i>=1; i--){

System.*out*.println(i);

}

}

}

o/p:

5

4

3

2

1

*Ex, Use of print statements:*

**class** Pro6 {

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

**for** (**int** i=5; i>=1; i--){

System.*out*.print(i);

}

System.*out*.println();

}

}

o/p:

54321

Program ended

*Ex6, Use of Inner for loop:*

**class** Pro6 {

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

**for**(**int** k=1; k<=5; k++){ //rows

**for** (**int** i=5; i>=1; i--){ //column

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

}

System.*out*.println();

}

System.*out*.println("Program ended");

}

}

o/p:

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

Ex,

**class** Pro6 {

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

**for**(**int** k=1; k<=5; k++)//rows

{

**for** (**int** i=5; i>=1; i--)//column

{

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

}

System.*out*.println();

}

System.*out*.println("Program ended");

}

}

o/p: 5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

Program ended

Ex,

**public** **class** Pro6 {

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

**for**(**int** k=1; k<=5; k++)//rows

{

**for** (**int** i=5; i>=1; i++)//column

{

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

}

System.*out*.println();

}

System.*out*.println("Program ended");

}

}

o/p:

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

4 4 4 4 4

5 5 5 5 5

Ex,

public class Pro6 { public static void main(String[] args) { for(int k=1; k<=5; k++){ //rows for (int i=1; i<=k; i++){ //column System.out.print(i+" "); } System.out.println(); } System.out.println("Program ended"); } }

o/p: 1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Program ended

Ex,

public class Pro6 { public static void main(String[] args) {

for(int k=5; k>=1; k--)//rows { for (int i=1; i<=k; i++)//column { System.out.print(i+" "); } System.out.println(); } System.out.println("Program ended"); } }

o/p:

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Program ended

**While loop:**

**Ex,**

class Pro7 { public static void main(String[] args) { int i=1; while (i<=5) { System.out.print(i); i++;

} } }

o/p: 12345

**If Statements:**

Ex,

class Pro7 { public static void main(String[] args) { int i=5; if (i>7) //check the condition is true or not System.out.print("number above 7"); System.out.print("number below 7"); //if multi statements then u should use curly braces otherwise you will get both statements

}

}

class Pro7 { public static void main(String[] args) { int i=5; if (i<7) //check the condition is true or not { System.out.print("number above 7"); } System.out.print("number below 7"); //if multi statements then u should use curly braces otherwise you will get both statements

}

}

o/p: number below 7

* We can write for loop below

class Pro9 { public static void main(String[] args) {

int i=1; for (; i<=5; i++ ){ System.out.print(i); } System.out.println(); } }

Ex,

class Pro9 { public static void main(String[] args) { int i=1; for (; i<=5;){ System.out.print(i); i++; } System.out.println();

} }

class Pro10{ static void square(int n){ //int arg method System.out.println("Square of n "+n);

int res=n\*n; System.out.println("Square is "+res);

if(n<=9){ square(n+1);//recursive call, } }

public static void main(String[] args) { square(5); //passing int value to the method } }

o/p:

Square is 25 Square of n 6 Square is 36 Square of n 7 Square is 49 Square of n 8 Square is 64

Square of n 9

Square is 81

Square of n 10

Square is 100

* It runs perfectly but if the condition change like if(n>=9) and pass the value like square (10) then error is occurred. called stackoverflow .

**Steps to Read Input From User:**

Step1: import java.util.Scanner;

Step2: Scanner sc1=new Scanner (System.in);

System.in=represent standard input device

Step 3; use methods to read inputs

1. nextInt()used to read integer value from keyboard return type of this method is integer.
2. nextDouble()used to read double value from keyboard ,return of this method is double
3. next() used to read string value from keyboard ,return of this method is string .

Ex,

import java.util.Scanner;

class Pro11 { public static void main(String[] args) {

Scanner sc=new Scanner(System.in); String stName; int stAge;

System.out.println("Enter your Name: "); stName=sc.next();

System.out.println("Enter Your age: "); stAge=sc.nextInt();

if (stAge>18){ System.out.println("eligable for vote");

}else{ System.out.println("not eligable!"); } } }

o/p:

Enter your Name: Azam

Enter Your age: 22

eligable for vote

/

Ex,

import java.util.Scanner;

class Pro12 { static void square(int n){ System.out.println("Square of n "+n);

int res=n\*n; System.out.println("Square is "+res);

if(n>1){ square(n-1);//recursive call //when error is called stackoverflow . }

}

public static void main(String[] args) { Scanner sc=new Scanner(System.in); System.out.println("Enter any number"); int num=sc.nextInt(); square(num); } }

**Array:**

int x1=4, int x2=5,int x3=6,int x4=7,

int n=n; In above we declare variable and value like this , but if we need same data type value at that time we need create array .**Array is nothing but a collection of same Data type.**

*Array declaration:*

Datatype [] array name;

*Array initialization:*

Arrayname= new datatype[size];

*Declaration and initialization:*

Int[] arr1= new int[5];

Default value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |

Index 0 1 2 3 4

Ex,

class Pro1 { public static void main(String[] args) { System.out.println("Program started");

int[] arr1=new int[5];

//array of int type , we store 5 int values

// every array has a property name length, which contains size of the array

System.out.println("array size"+arr1.length);

//store values separately

arr1[0]=25;

arr1[1]=26;

arr1[2]=27;

arr1[3]=24;

arr1[4]=29;

System.out.println("4th element"+arr1[3]);

for(int i=0; i<arr1.length-1; i++){

System.out.println(arr1[i]);

}

System.out.println("Program ended");

}

}

o/p: Program Start 24

25 26 27 24 29 Program ended

Ex,

class Pro1 { public static void main(String[] args) { System.out.println("Program started");

int[] arr1=new int[5];

//array of int type , we store 5 int values

// every array has a property name length, which contains size of the array

System.out.println("array size"+arr1.length);

//store values

arr1[0]=25;

arr1[1]=26;

arr1[2]=27;

arr1[3]=24;

arr1[4]=29;

System.out.println("Array element in reverse order");

for(int i=arr1.length-1; i>=0; i--){

System.out.println(arr1[i]);

}

System.out.println("Program ended");

}

}

o/p:

reverse output is happen.

Ex,

import java.util.Scanner;

class Pro2{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in); System.out.print("Enter the array size: "); int size=sc.nextInt(); int [] arr=new int[size];//read values from keyboard System.out.print("Now Enter the array elements: ");

for (int i=0; i<arr.length ; i++ ){ System.out.println("<"+i+"> "); arr[i]=sc.nextInt(); } System.out.println("============================");

System.out.println("Arry elements"); for (int i=0; i<arr.length ; i++ ){ System.out.println(arr[i]); } } }

o/p:

Enter the array size: 5

Now Enter the array elements

<0>1

<1>2

<2>3

<3>4

<4>5

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

Arry elements are:

1

2

3

4

5

Ex,

import java.util.Scanner;

class Pro4{

//method arg is arry type // print the element of given arry

static void printArray(int[] arg){ System.out.println("Arry elements"); for (int i=0; i<arg.length ; i++ ){ System.out.print(arg[i]+" "); } }

/\* methods takes an argument of int array type. Fills the array with values and returns Filled array \*/

static int[] fillArray(int[] arg){

System.out.println("filling an Array ");

Scanner sc=new Scanner(System.in);

//read values from keyboard

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

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

arg[i]=sc.nextInt();//passing an arry

}

return arg;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the array size: ");

int size=sc.nextInt();

int [] arr=new int[size];

int [] arr2=fillArray(arr);//passing an empty array

printArray(arr2); } }

**String Function:**

String str; declaration.

Str=”jspiders”; initialization.

How to store

Str variables Names

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| J | S | P | I | d | e | R | s |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

System.out.println(str);jspiders

* **length()**returns total no of chars in a given string
* **charAt**returns char present at specified index
* **indexOf(char**)returns first occurrence position of given char
* **contains(charseq**)returns true if given sequence of char is present in the string
* **startsWith(charseq)--->**returns true if string begins with specified sequence of char
* **endsWith(charseq)**returns true string ends with specified sequence of char
* **substring (startindex**)returns string from the specified start index till end of the string
* **substring(startindex, endindex**)returns string between start index and end index of the string, end index position is excluded
* **equals(string**)returns true if the current string is equal to given string
* **toUpperCase(**)returns string which is converted to upper case
* **toCharArray()**converts the string to char array and returns the array. Mostly used
* **split(value**)split the given string based on specified value and return string array. Mostly used

Pr1:

class Pro1 { public static void main(String[] args) {

String str= "javadeveloper"; System.out.println("given string"+ str);

int tot=str.length(); System.out.println("total char"+tot);//o/p is: total char13

char c1=str.charAt(4); System.out.println("chaar @ 4 pos: "+c1);// o/p is: char @ 4 pos: d

int i1=str.indexOf('e'); System.out.println("first occ of e : "+i1);//o/p : first occ of e : 5

int i2=str.indexOf('e',6); System.out.println("2nd occ of e : "+i2);//o/p: 2nd occ of e : 7

int i3=str.lastIndexOf('e'); System.out.println("Last occ of e: "+i3);//o/p: last occ of e: 11

} }

Pro2:

class Pro2 { public static void main(String[] args) {

String str= "javadeveloper";

System.out.println("given string: "+ str);// o/p: javadeveloper

boolean b1=str.contains("dev"); System.out.println("contain 'dev' : "+ b1);//op: true

boolean b2=str.startsWith("java"); System.out.println("contain 'java' : "+ b2);//op: true

boolean b2=str.endsWith("per"); System.out.println("contain 'per' : "+ b2);//op: true

String s1=str.substring(8); System.out.println(s1);// op: loper

String s2=str.substring(4,11); System.out.println(s2);// op: develop

} }

class Pro3 { public static void main(String[] args) {

String str1= "javadeveloper"; String str2= "javadeveloper";

if (str1.equals(str2)){ System.out.println("String are same");

}else{

System.out.println("String are not same");

}

}

}

Pro4:

class Pro4 { public static void main(String[] args) { System.out.println("Program started");

String[] str ={"Ramesh","Samesh","Tamesh"}; System.out.println("array size: "+str.length);

for (int i=0; i<str.length;i++ ){ System.out.println(str[i].toUpperCase()); }

System.out.println("Program ended"); } }

o/p: Program started

array size: 3

RAMESH

SAMESH

TAMESH

Program ended

Pro5:

class Pro5{

public static void main(String[] args) { System.out.println("Program started");

String str ="Ramesh Samesh Tamesh"; System.out.println("given string: "+ str);

String[] strArr=str.split(" ");

for (int i=0; i<strArr.length;i++ ){ System.out.println(strArr[i].toUpperCase()); } System.out.println("Program ended"); } }

Note,

* ***Main method arguments should be string because is use***
* ***Whenever we execute program JVM creates an array of strings file the values of command line and passé to main function of the class which is be executed.***

Tomorrow pattern

Pattern is done by Teja Mem.

**Object Oriented Program** (OOPS): (whatever we see it’s an object)

Ex, 1:

class Demo1 {//functional class static void test() { System.out.println("Running user define method"); } }

class Demo2{//functional class static void disp() { System.out.println("running user define method"); } }

class Demo3 {//Main class public static void main(String[] args) { System.out.println("running demo3"); } }

Note ,

When it compile its compile three class but run only main method class.

In java src file any no of class body unique the function the compiler generates separate class file for each class body, we can or jvm can execute only those class file which as main method.

class class\_name-class declaration

{

**Member of class:**

1. Variabledata member
2. Methodsfunction member

**Member type:**

1. Static member/ class member
2. Non-static member/instance member.

Ex,

Clss demo1{

static int k=12;//static data member Int j=34; //data member

static void test(){ static function member Int i=0;local variable }

Void disp(){non static fun }

Class demo 2{

static int k=12; static void test(){ // static mem

}

**Access spacifier: (**restrict access to members)

1. Private
2. Package level
3. Protected
4. public

How to access static members of class?

Syntax,

classname.StaticMemberName;is called reference.

Ex,

class Demo1{ //functional class static int k=12; static void test() { System.out.println("Running user define method"); } }

class Demo2{ //functional class static void disp() { System.out.println("running user defuine method"); } }

class Demo3{ //Main class public static void main(String[] args) { System.out.println("k value: "+Demo1.k); Demo1.test(); } }

o/p: k value: 12 Running user define method

Ex,

class D2{

static int i=12; static int j=13; }

class MainClass2 { public static void main(String[] args) {

System.out.println("i value: "+D2.i); System.out.println("j value: "+D2.j); System.out.println("------------------------ ");

D2.i=67;//reassignment D2.j=23;//reassignment

System.out.println("i value: "+D2.i); System.out.println("j value: "+D2.j); } }

* Class is a definition block is used to define the members of class ,
* The member can be data member or function member
* The data members are used to store data required for the program when as the function members are used to operate on the data members
* The members are classified into tow type static and non-static,

1. The static members are declared using static keyword and is also known as class members
2. The non-static member declared without static keyword it’s also known as instance members of the class.

* Static data members of a class can be rename by any class
* The members of a class can be access from any class based on the access spicier
* The static members of a class are refer by using class name

Ex,

class D3{ static int x;//data member static void test(){ int i=56;//local variable System.out.println("i valu: "+i); System.out.println("x valu: "+x); //you can access global variable to anywhere e.g. “x” but local variable can’t use outside the block e.g. “i” } }

class MainClass3 { public static void main(String[] args) { System.out.println("x valu: "+D3.x); D3.test(); } }

o/p: x value 0

I value 56

X value 0

**How to access non-static member:**

class Demo1{ Int k=20;//non-static member Void test(){// non-static function } }

**Instance creation or object Creation:**

1. New Operator
2. Constructor

new Constructor();

constructorname==class name

A copy of all non-static members of the class are loaded in the memory.

Ex,

|  |  |  |
| --- | --- | --- |
| Demo1 obj1=new Demo1();  K[20]  Test(){  ==  ==  }  Memory location | Demo1 obj2=new demo1();  K[20]  Test(){  ==  ==  }  Memory location | Demo1 obj3=new demo1();  K[20]  Test(){  ==  ==  }  Memory location |

**Reference variable:**

* Used to refer instance of class
* Are declared using class name

Syntax className reference\_variable\_name;  
Demo1 obj1;

1. Obj1-null
2. Obj1=new Demo1();

Ex,

class Demo1{ int k=20; void test() { System.out.println("Runnig non-static method"); } }

class MC4 { public static void main(String[] args) {

Demo1 obj1=new Demo1(); System.out.println("k value is : "+ obj1.k); obj1.test(); System.out.println("obj1 location value is : "+ obj1); } }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Demo2{

Int x=20;

Int y=40

}

//instance

Demo2 obj1=new Demo2();

Demo2 obj2=new Demo2();

Syso(obj1.x)20;

Syso(obj1.y)40;

Obj1.x=50; //re assignment of obj1 object, it will not effect to any other object like obj2 even main class Demo1

Obj1.y=60;//re assignment

Sop(obj2.x)20

Sop(obj2.y)40

Sop(obj1.x)50

Sop(obj1.x)60

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex,

class Demo2 {

int x1=20; int x2=40; }

class MC5{ public static void main(String[] args) {

Demo2 obj1=new Demo2(); Demo2 obj2=new Demo2();

System.out.println("1st insatance x1 valu: "+obj1.x1);// 20 System.out.println("1st instance x2 valu: "+obj1.x2);// 40

System.out.println("2nd instance x1 valu: "+obj2.x1);//20 System.out.println("2nd insatnx1 valu: "+obj2.x2);//40 System.out.println("--------------------------- ");

obj1.x1=50; obj1.x2=60;

System.out.println("1st insatance x1 valu: "+obj1.x1);//50 System.out.println("1st instance x2 valu: "+obj1.x2);//50

System.out.println("2nd instance x1 valu: "+obj2.x1);//20 System.out.println("2nd insatnx1 valu: "+obj2.x2);//40 } }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex,

class Demo3{ double y1=56.23; boolean b1=true; }

class MC6 { public static void main(String[] args){ Demo3 obj1=new Demo3();

Demo3 obj2; obj2=obj1; //copy value obj1 to obj2

System.out.println("obj1 value: "+obj1); System.out.println("obj2 value: "+obj2);

obj1.y1=98.23; //now it also reassign to obj2 obj1.b1=false;

System.out.println("y1 value: "+obj2.y1); System.out.println("b1 value: "+obj2.b1);

}

}

o/p:

obj1 and obj2 address are same

y1 value: 98.23 , b1 value: false

**Class and Objects:**

Any entity digit with known class, state and behavior is known as object, the state represent the character of the object whereas the behavior represents functionality of the objects

* A class is design of objects which defines the properties of the objects
* A class is object which defines states and behavior of an object.
* States of the objects are defined by data members whereas the behaviors of the objects are define by non-static function member of the class.

**Reference variables:**

* The instance of the class can be created by using new operator and constructor of the class.
* The new operator is used to instance of class whereas the constructors are used to initialize the instance.
* Whenever we create instance of the class copy of non-static member are loaded in the memory, to refer each instance we should create reference variable or object references.
* Reference variables are special variable which is used to refer an instance of the classes.
* The reference variable should be declared by using class name.
* For a reference variable we can assign either null value or an instance of class.
* An instance can refer by any no of reference variables. In such case modifying the state of the instance from 1 reference variable will reflect in another reference variable also
* The instance created will always have non-static data member and non-static function member
* The modifying the states of one statements will not update the entire instance.

The variables declaration using data type are known as positive variable, is used to store primitive values.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex, of create objects:

class Circle{

double rad; final static double pi=3.14;

void area() { double a1=pi\*rad\*rad; System.out.println("area is: "+a1); }

void circum() { double c1=2\*pi\*pi; System.out.println("circum is: "+c1); } }

class MC7 { public static void main(String[] args) {

Circle cir1=new Circle(); cir1.rad=2.1; cir1.area(); cir1.circum();

Circle cir2=new Circle(); cir2.rad=3.1; cir2.area(); cir2.circum();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class NoteBook{

//data members int pages=50; double price=25.50;

//function member void open() { System.out.println("oppening"); }

void close() { System.out.println("closing"); }

void turnpages() { System.out.println("turn pages"); }

}

class MC9{ public static void main(String[] args) {

NoteBook b1=new NoteBook(); b1.open(); b1.close(); b1.turnpages(); } }

**MEMORY ALLOCATION:**

To execute any program the JVM makes use of following memory area:-

1. **Heap Area:**

The Heap Area is used to store the instance in the program. In the heap area the memory allocated is random.

The new operator loads the non-static member into heap area.

1. **Static Pool Area:**

This area is used for storing the static member of the class .The pool will be created for each class; the class loader program of JVM is responsible to load the static member of the class to the static position.

1. **Method Area:**

The method area is used to store the definition statements of methods.

1. **Stack Area:** The stack area is used for execution purpose and normal statements to which have to be executed in JVM should be come to stack area.

The local variable components always store in stack memory area.

**JVM**

**HEAP AREA**

**(**Non-Static**)**

Storage

**STACK AREA**

**(**Execution**)**

**METHOD AREA**

**(**Methods**)**

Storage

**STATIC POOL AREA**

**(**Static Member**)**

Storage

Garbage collector

**HEAP AREA (**non-static member**)**

Inst of class2-obj1 Inst of class3/2-obj2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | **Value** |  | **Identifier** | **Value** |
| Int i | 3 | int j | 5 |
| Disp(); |  |  |  |
|  |  |  |

Temporary value Temporary value

|  |
| --- |
| Disp(); |
| test(); |
| Main()  Obj1.i address |

**STACK AREA (**Execution**)**

e.g.

Called

Caller

**METHOD AREA/BODY(**static + non-static**)**

Main () st test () disp()

**STATIC POOL AREA (**only static member**)**

Main Class Class1

St int k=2;

St test();

Static main()

Method body

Method body

Method body

Ex,

class Demo1 { static int k=25; int j=5; }

class MainClass1 { public static void main(String[] args) {

System.out.println(" static k value: "+Demo1.k);//call static member from static pool area

Demo1 obj1=new Demo1();// obj1 object create to Heap Area and address the member of class Demo1

System.out.println(" non-st j value: "+obj1.j);//calling from heap area

}

}

Ex,

class Demo2{ static void test() { System.out.println("Running static test() method"); return; }

void disp(){ System.out.println("Running non-static disp() method"); } }

class MainClass2 { public static void main(String[] args) {

Demo2.test(); //calling static method from static pool area where store Demo2 class static member Demo2 obj1=new Demo2(); //obj1 object create to Heap Area and address the member of class Demo1 obj1.disp();//address disp method to heap area where created obj1 object. return; } }

Ex,

class Demo3{ static void test() { System.out.println("Running static test() method"); return; }

void disp(){ System.out.println("Running disp() method");

Demo4 obj2=new Demo4(); System.out.println("x value: "+obj2.x); } }

class Demo4{ int x=34; }

class MainClass3 { public static void main(String[] args) { Demo3.test(); Demo3 obj1=new Demo3(); obj1.disp(); return; } }

**Blocks (**static & non-static**):**

Ex, class MainClass1 { static{ System.out.println("Running 1st st block"); }

public static void main(String[] args) { System.out.println("Running main"); }

static{ System.out.println("Running 2nd st block"); } }

o/p:

Running 1st st block

Running 2nd st block

Running main

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ex, class D1{ int k=20; static{ System.out.println("Running D1 class st block"); } }

class MC2{ static{ System.out.println("Running main class st block"); }

public static void main(String[] args) {

System.out.println("Running main");

D1 obj=new D1();

System.out.println("k value "+obj.k);

System.out.println("End main");

}

}

o/p: Running main class st block

Running main

Running D1 class st block

k value 20

End main

Ex,

class D3{ static int k=12; static{ System.out.println("k value "+k);//12 k=34; }

static{ System.out.println("k value "+k);//34

Int k=45; System.out.println("k value "+k);//45 } }

class MC3{ public static void main(String[] args) { System.out.println("Running main");

System.out.println("k value "+D3.k);//34

System.out.println("End main"); } } o/p: Running main k value 12 k value 34 k value 45 k value 34 End main

Ex, class S1{ int j=34; { System.out.println("Running non st block"); } }

class MC4 { public static void main(String[] args) { System.out.println("p str");

S1 obj1=new S1(); System.out.println("j value "+obj1.j);

S1 obj2=new S1(); System.out.println("j value "+obj2.j);

System.out.println("p end");

}

}

o/p:

p str

Running non st block

j value 34

Running non st block

j value 34

p end

Java language provides initialize block to initialize the data member of the class

There are two types of blocks

1. **Static initialize block :**

* Static initialize block is used to initialize the static members of the class. The static block are executed at the time of class loading by class loader , we can define multiple static blocks ,JVM executed sequentially
* If main class run contains main method and static blocks then JVM first run static block and then main method

1. **Non-Static initialize:**

* The non-static initialize blocks is used to initialize the non-static data member of the class
* The non-static blocks is executed whenever instance class is to be created
* In a class we can define multiple non-st blocks , which will be executed sequentially

**Constructor:**

Object creation:

1. New operator
2. Constructor of class

New constructor ()://initialize non-static data member

1. Allocate memory
2. Load non-static member
3. Calls constructor of class

EVERY CLASS MUST HAVE CONSTRUCTOR

1. Compiler defined constructor(default constructor)
2. User defined constructor.

Syntax, Constructor\_name(arg)🡪constructor declaration

{

Write code to do initialize

}

1. Cons name must be same as class name
2. No return type.

Ex, class Sample1 { int k=34;

Sample1() { System.out.println("running user define constrructor"); }

void test() { System.out.println("running test method"); } }

class MainClass1 { public static void main(String[] args) {

Sample1 obj1=new Sample1(); Obj1.test(); System.out.println("k value: "+obj1.k);

}

}

o/p: running user define constructor running test method k value: 34

Ex, class Sample1 { int k; Sample1() //user define constructor{ System.out.println("running user define constrructor"); k=96;//initialize the k value in the constructor }

void test() { System.out.println("running test method"); } }

class MainClass2{ public static void main(String[] args) { Sample1 obj1=new Sample1(); obj1.test(); System.out.println("k value: "+obj1.k);

}

}

o/p:

running user define constrructor

running test method

k value: 96

Ex, class S2{ int k=21;

//user define arg constructor //parameterized constructor S2(int arg1){ System.out.println("Running user define parameterized constructor");

k=arg1;

}

}

class MC3 { public static void main(String[] args) {

S2 obj1=new S2(67); System.out.println("k value: "+obj1.k); System.out.println("-------------------------------------");

S2 obj2=new S2(37); System.out.println("k value: "+obj2.k);

}

}

o/p:

Running user define parameterized constructor

k value: 67

-------------------------------------

Running user define parameterized constructor

k value: 37

*Constructor is continue*………..

**Tejja mem Array String:**

**package** tejja;

**public** **class** Mango {

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

String str = "Mango";

**char**[] ch=str.toCharArray();

**for**(**int** i=ch.length-1;i>=0;i--){

System.*out*.print(ch[i]);

}

}

}

**package** tejja;

**public** **class** Sun {

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

String str ="Sun rises in the east";

String[] str1=str.split(" ");

//op sud be like "east rises in the Sun"

String temp;

temp=str1[0];

str1[0]=str1[str1.length-1];

str1[str1.length-1]=temp;

**for**(**int** i=0;i<=str1.length-1;i++){

System.*out*.print(str1[i]+" ");

}

}

}

Assignment from tejja mem

String= “Sun rises in the east”

o/p=Sun sesir in eht east;

**Constructor:**

Ex,

class S3{ int k; double d;

//user define argument constructor or parameterized constructor S3(int arg1,double arg2){ System.out.println("Running int, double constructor"); k=arg1; d=arg2; }

void test(){ System.out.println("k value: "+k); System.out.println("d value: "+d); } }

class MainClass4 { public static void main(String[] args) {

S3 obj1=new S3(25,7.5); S3 obj2=new S3(95,47.5); obj1.test(); obj2.test(); } }

o/p:

Running int, double constructor

Running int, double constructor

k value: 25

d value: 7.5

k value: 95

d value: 47.5

**Constructor Overloading:**

Ex,

class S4{ int k; double d;

//constructor overloading S4() { System.out.println("running no arg cons"); }

S4(int arg1) { System.out.println("running int arg cons"); k=arg1; }

S4(double arg1) { System.out.println("running double arg cons"); d=arg1; }

S4(int arg1,double arg2) { System.out.println("running int, double arg cons"); k=arg1; d=arg2; }

void test(){ System.out.println("k value: "+k); System.out.println("d value: "+d); } }

class MainClass5 {

public static void main(String[] args) {

S4 obj1=new S4(); obj1.test();

S4 obj2=new S4(23); obj2.test();

S4 obj3=new S4(7.5); obj3.test();

S4 obj4=new S4(45,95.4); obj4.test(); System.out.println("Program Ended"); } }

o/p:

running no arg cons

k value: 0

d value: 0.0

running int arg cons

k value: 23

d value: 0.0

running double arg cons

k value: 0

d value: 7.5

running int, double arg cons

k value: 45

d value: 95.4

Program Ended

Note,

* **Constructors are special member of the class which is used to initialize the data members of the class.**
* The constructors are executed whenever the instances of class are created.
* Every class must have constructor in order to create an instance of the class.
* The constructor can be created either by compiler or by user.
* The constructor define by compiler is known as default constructor
* Compiler defines a constructor if the class does not have any user define constructor. If in case the class is having user define constructor then the compiler not define default constructor.
* The constructor define by programmer is known as user define constructor .the user can define either 0 or with argument constructor.
* The constructor defines with arguments constructor are known as parameter constructor.
* Whenever as object created using parameter constructor we need to pass value to the constructor
* The constructor cannot be declare as static

**Constructor Overloading:**

In a class defining multiple constructor with deferent parameters is known as constructor overloading

* When defining overloading constructor the constructor should deferent in terms of parameter type or parameter length.
* Constructor overloading help us to create an instance of class to the different initialization.
* While defining a constructor the constructor name should be same as class name, and constructor should not have return type.

**Program define total object of a class:**

Ex,

class Car { static int count; Car(){ System.out.println("creating car instance"); count++; }

static void total(){ System.out.println("total car manufacture: "+count); } }

class MC6{ public static void main(String[] args) { for (int i=1;i<=15 ;i++ ){ new Car(); } Car.total(); } }

o/p:

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

total car manufacture: 15

Ex,

class Car { String brand; static int count; Car(){ System.out.println("creating car instance"); count++; }

Car(String arg1){ System.out.println("creating car instance of brand: "+arg1);

brand=arg1; count++; } static void total(){ System.out.println("total car manu: "+count); } }

class MC7{ public static void main(String[] args) { for (int i=1;i<=10 ;i++ ){ new Car(); } for (int i=1;i<=10 ;i++ ){ new Car("Audi"); } Car.total();

}

}

o/p:

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

total car manu: 25

Ex,

class S1 { int k=12; { System.out.println("runing non static block"); k=56; }

S1(){ System.out.println("runing S1 constructor"); k=34; } }

class MC8 { public static void main(String[] args) { S1 s1=new S1(); System.out.println("k value is: "+s1.k); } }

o/p

runing non static block

runing S1 constructor

k value is: 34

*Use non-static block only so that you can easily increment overloaded constructor:*

Ex,

class Car { String brand; static int count;

{ count++; }

Car(){ System.out.println("creating car instance"); }

Car(String arg1){ System.out.println("creating car instance of brand: "+arg1); brand=arg1; }

static void total(){ System.out.println("total car manu: "+count); } }

class MC7{ public static void main(String[] args) {

for (int i=1;i<=10 ;i++ ){ new Car(); } for (int i=1;i<=10 ;i++ ){ new Car("Audi"); } Car.total(); } }

o/p:

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

creating car instance of brand: Audi

total car manu: 20

Ex,

class Student{ final int id; String name; Student(int arg1,String arg2){ System.out.println("creating instance!");

id=arg1; name=arg2; } }

class MC9 { public static void main(String[] args) {

Student st1=new Student(21458,"Ramest"); Student st2=new Student(31458,"Suresh");

System.out.println("id of first student: "+st1.id); System.out.println("Name of first student: "+st1.name);

System.out.println("id of 2nd student: "+st2.id); System.out.println("name of 2nd student: "+st2.name); } }

o/p:

creating instance!

creating instance!

id of first student: 21458

Name of first student: Ramest

id of 2nd student: 31458

name of 2nd student: Suresh

In a class if a data member as declare as final then it must be initialize in any of the below braces

* At the time of declaration
* Using respective blocks
* Using constructor

For each instance if we need it very separate constant values then we should use constructor to initialize the final data members.

*Location identify using “this”:*

Ex,

class Student{ final int id; String name;

Student(int arg1,String arg2){ System.out.println("creating instance!"); id=arg1; name=arg2; }

void dispinfo(){

System.out.println("object: "+this); System.out.println("student id: "+this.id); System.out.println("student name: "+this.name); } }

class MC10{ public static void main(String[] args){

Student st1=new Student(21458,"Ramest"); Student st2=new Student(31458,"Suresh");

System.out.println("st1: "+st1); System.out.println("st2: "+st2);

st1.dispinfo(); st2.dispinfo(); } }

o/p:

creating instance!

creating instance!

st1: Student@659e0bfd

st2: Student@2a139a55

object: Student@659e0bfd

student id: 21458

student name: Ramest

object: Student@2a139a55

student id: 31458

student name: Suresh

*For the local and global variable declaration time used to “this” key word.*

Ex,

class Student{ final int id; //global variable String name; //global variable

Student(int arg1,String arg2){ System.out.println("creating instance!"); id=arg1; name=arg2; }

void dispinfo(){ int id=1234; //local variable String name="Ashok";//local variable

System.out.println("object: "+this); System.out.println("student id: "+this.id);//calling global varaible System.out.println("student name: "+this.name); } }

class MC9{ public static void main(String[] args) {

Student st1=new Student(21458,"Ramest"); Student st2=new Student(31458,"Suresh");

System.out.println("st1: "+st1); System.out.println("st2: "+st2);

st1.dispinfo(); st2.dispinfo(); } }

o/p:

creating instance!

creating instance!

st1:Student@659e0bfd

st2:Student@2a139a55

object:Student@659e0bfd

student id: 21458

student name: Ramest

object:Student@2a139a55

student id: 31458

student name: Suresh

*If the global and local variable data member same at that time u should use “this” keyword. Otherwise its id=id is declare as local variable.*

Ex,

class Student{ int id;//global variable Student(int id){ //constructor variable is local variable but same name as global variable this.id=id; //initialize the constructor local variable to the global variable using this keyword } }

class MC11 { public static void main(String[] args){ Student st1=new Student(1234);//pass the value to the constructor int id local variable System.out.println("student id: "+st1.id);//call to the global variable id

}

}

o/p:

student id: 1234

* The java language provides special keyword by name “this” which is used to refer current object member
* “this” key word always points to the current object that is it refers to the current object
* “this” key word should be used either in a non-static method content or in the constructor body
* “this” keyword can’t be used in the static methods
* Whenever the local variable name and the data member name are same, in such case the data members name can be differentiate local variable name by using “this” keyword.

The constructor are used to initialize the object data members, after initialize the data member the constructor returns the object of the constructor

The return type of the constructor will be the class type

1. Passing the object
2. Returning object

**METHOD**

* ***Pass the object address to reference variable IF Method Argument is Class Type:***

Ex, class Sample1{

int i=34; double j=4.5;

void disp() { System.out.println("Runnigg disp of S1 "); } }

class D1 { void test(Sample1 arg1) //method argument is class type { System.out.println("running test() of D1"); System.out.println("i value: "+arg1.i); System.out.println("j value: "+arg1.j); arg1.disp(); } }

class MC1 { public static void main(String[] args) {

D1 obj=new D1(); obj.test(new Sample1());//passing an instance of S1 Class } }

o/p:

running test() of D1

i value: 34

j value: 4.5

Runnigg disp of S1

program ended!

**Passing a ref1 value of S1 class:**

class MC1 {

public static void main(String[] args) {

D1 obj=new D1();

S1 ref1=new S1();

obj.test(ref1);

//value of ref1 is copied to argument

//passing a reference of S1 Class

System.out.println("program ended!");

}

}

* U can change data member to the another class and it will copied to that class which is references and u will get change value in the main class

Ex,

class Sample1{

int i=34; double j=4.5;

void disp() { System.out.println("Runnigg disp of Sample1 "); } }

class Demo1 {

void test(Sample1 obj1) //method argument is class type { System.out.println("running test() of Demo1 class"); System.out.println("i value: "+obj1.i); System.out.println("j value: "+obj1.j); obj1.disp();

obj1.i=87;//reassign and it will change the Sample1 class variable also obj1.j=7.6;//reassign } }

o/p:

running test() of Demo1 class

i value: 34

j value: 4.5

Runnigg disp of Sample1

i value: 87

j value: 7.6

program ended!

class MainClass1 { public static void main(String[] args) {

Demo1 d1=new Demo1(); Sample1 ref1=new Sample1(); d1.test(ref1); //value of ref1 is copied to obj1 //passing an reference of Sample1 Class

System.out.println("i value: "+ref1.i); System.out.println("j value: "+ref1.j); System.out.println("program ended!"); } }

* If the method argument is a class type then while invoking such method we have to pass the instance of the class mentioned in the method argument.
* The instance can be passed directly or we can pass the reference of instance also
* If we passing the ref of instance then any modification to that instance will reflect in the other references

Heap Area

test()

arg1[address]

main()

ref[address]

Copied

***\*\* Java does not support pass by ref since java language is not be using pointer concept ,however we can pass the object address to reference variable not true pointers.***

**If Method returns type is class type:**

Ex 1,

class Sample1{ int i=34; double j=4.5; void disp() { System.out.println("Runnigg disp() of Sample1 class "); } }

class Demo1 {

Sample1 test(){//method return type is class type System.out.println("running test() of Demo1 class");

retrun new Sample1();//return new Sample1()

//returning instance of Sample1 class

}

}

class MainClass2 {

public static void main(String[] args) {

Demo1 d1=new Demo1();

Sample1 ref1=d1.test();

System.out.println("i value: "+ref1.i);

System.out.println("j value: "+ref1.j);

ref1.disp();

System.out.println("Hello World!");

}

}

Ex 2,

class Sample1 {

int i=34; double j=4.5; void disp(){ System.out.println("Running disp() of Sample1 class"); } }

class Demo1 {

Sample1 test() {//method return type is class type System.out.println("Running test() of Demo1 class");

Sample1 obj1= new Sample1();

System.out.println("i value: "+obj1.i);//34 System.out.println("j value: "+obj1.j);//4.5

obj1.i=76;//reassign obj1.j=7.6;//reassign

return obj1; } }

class MainClass3{ public static void main(String[] args) {

Demo1 d1=new Demo1(); Sample1 ref1=d1.test();

System.out.println("i value: "+ref1.i);//76 System.out.println("j value: "+ref1.j);//7.6 ref1.disp(); } }

o/p:

Running test() of Demo1 class

i value: 34

j value: 4.5

i value: 76

j value: 7.6

Running disp() of Sample1 class

* If the method return type is class type then such method should return the instance.
* Whenever the call an method which has class type as a return type, we should define a reference variable to store the return object.

A java method can return only one value if we have to return multiple values of same type then go for arrays

If we have to return multiple values of diff type then go for objects

A ref variable declare in the method context is known as local ref variable. The scope of local ref variable is limited to only that method context it can’t be access from other method context until unless the method returns the reference.

void test(){

Int x=10;

Sample1 ref1=new sample1();//local object

}

**Static and non-static reference variable:**

Ex, class Sample1{ ==== }

class Demo1{

static Sample1 ref1=new sample1();//static ref variable Sample 2 ref2=new sample2();//non-static ref variable }

How to call in the Main class, Demo1.ref1.member// for static ref variable

Demo1 d1=new Demo1(); d1.ref2.member;// for non-static ref variable

* A ref variable can be declare as the data member of the class
* It can be static data member or non-static data member
* If a ref variable static then we can only one copy of the ref variable in memory, this can also be call as static object
* If the ref variable is non-static then for each instance variable a copy of ref variable will be create this can also be called as non-static object.
* Whenever a ref variable is data member of the class then the memory allocation access either in the heap or in the static pool

1. If a ref variable is declare as static and final then we get a single copy of ref variable which cannot be reassign
2. If a non-static ref variable is final we get multiple copy of ref variable for each instance which cannot be re assign

**Composition**

*Non-static reference variable or non-static object example:-*

class Demo1 { int k=12; void test() { System.out.println("running test()"); } } class Sample1 { boolean b=true; Demo1 obj1=new Demo1();//non-st ref variable of type Demo1

**Sample 1 HEAP AREA**

ref1 add **Demo1**

obj1 add

void disp() { System.out.println("running disp()"); } }

class MainClass1 { public static void main(String[] args) {

Sample1 ref1=new Sample1(); System.out.println("b value: "+ref1.b); ref1.disp(); System.out.println("k value: "+ref1.obj1.k); ref1.obj1.test(); } }

For output command is ” javac \*.java” then “java MainClass1”

o/p: b value: true running disp() k value: 12 running test()

*Static reference variable or static object:*

class Demo1 { int k=12; void test() { System.out.println("running test()"); } } class Sample1 { boolean b=true; static Demo1 obj1=new Demo1();//static ref variable of type Demo1 void disp() { System.out.println("running disp()"); } }

class MainClass1 { public static void main(String[] args) {

**Sample 1 HEAP AREA**

Ref1 **Demo1**

Obj1

Sample1 ref1=new Sample1(); System.out.println("b value: "+ref1.b); ref1.disp();

System.out.println("k value: "+Sample1.obj1.k); Sample1.obj1.test(); }

}

**Static Pool Area**

Sample1

Obj1

o/p:

b value: true running disp() k value: 12 running test()

**INHERITANCE**

* A class acquiring/gathering property of another class is called as inheritance.
* The classes from where the properties are inherited are known as base class or Super class.
* The class through which properties are inherited is known as derived class or Sub class.
* Always subclass inherits the non-static properties from the super class.
* The static properties will never be inherited to the sub class.
* Whenever we create the instance of sub class that instance will always have the non-static properties of sub class and its super class

**There are 4 types of inheritance:-**

1. Single Inheritance: In this type of inheritance a sub class inherits properties from only one super class.
2. Multi-level inheritance: In this case the sub class inherits the properties of super class which inherits the properties of another super class. We can define any level of inheritance
3. Multiple Inheritances: Sub class inheriting from more than one super class is known as multiple inheritances java does not support multiple inheritances **through classes**.
4. Hierarchical Inheritance: In this type of inheritance more than one subclass inherits the properties from one super class, In other words the sub classes having common super class. This type of inheritance is used to **achieve generalization.**

Note,

* If a class is declare as final, such classes can’t have subclasses in other words we can’t inherit the properties of final class. We can create an instance of final class and we can access it as non-static properties.
* A constructor of a super class will not be inherited to subclass, but constructor plays major rule in inheritance
* Whenever we create an object of sub class it will always have the non-static properties of any level super class
* Object is a super class for every java class
* Every java class should define a super class, if user is not define a super class then compiler define default super class by name “Object”
* Each and every class created in java language is always having properties of Object class.

***Example of Single Inheritance*** when we create an instance of sample1 class then instance should be work on D1 class .

class Demo1 {

int k=12; void test() { System.out.println("runing test() of Demo1 "); } }

class Sample1 extends Demo1{

double d=45.67; void disp() { System.out.println("running disp() of Sample1"); } }

class MainClass1 { public static void main(String[] args) {

Demo1{

===

}

Sample1 extends Demo1{

====

}

Sample1 ref1=new Sample1(); System.out.println("d value: "+ref1.d); ref1.disp();

System.out.println("K value: "+ref1.k); ref1.test(); } }

o/p:

d value: 45.67 running disp() of Sample1

K value: 12 runing test() of Demo1

***Example of multi-level Inheritance:***

class D1{ //super class

int k=12; void test() { System.out.println("runing test() of D1 "); } }

Demo1{

===

}

class S1 extends D1{ //sub class

double d=45.67; void disp() { System.out.println("running disp() of S1"); } }

Sample1 extends Demo1{

====

}

Run1 extends Sample1{

==

}

class R1 extends S1{

boolean b=true; void view() { System.out.println("running view() of R1"); } }

class MC2 { public static void main(String[] args) {

R1 ref1=new R1();

System.out.println("K value: "+ref1.k); ref1.test(); System.out.println("d value: "+ref1.d); ref1.disp(); System.out.println("b value: "+ref1.b); ref1.view();

} }

o/p:

K value: 12 runing test() of D1 d value: 45.67 running disp() of S1 b value: true running view() of R1

***Exam of multiple Inheritance:***

Sample1{

====

}

Demo1{

===

}

Class Run1 extends Demo1,Sample1{ //error

====

}

**Why java doesn’t support multiple Inheritances**

Run1{

==

}

Answer is below after super () statements

***Exam of Hierarchical Inheritance:***

class Demo1{ //super class

int k=12; void test() { System.out.println("runing test() of Demo1 "); } }

class Sample1 extends D1{ //sub class

double d=45.67; void disp() { System.out.println("running disp() of Sample1"); } }

class Run1 extends Demo1{

boolean b=true; void view() { System.out.println("running view() of Run1"); } }

Demo1{

===

}

Sample1{

====

}

Run1{

==

}

class MainClass4 { public static void main(String[] args) {

Run1 ref1=new Run1(); System.out.println("b value: "+ref1.b); ref1.view(); System.out.println("K value: "+ref1.k); ref1.test();

Sample1 ref2=new Sample1(); System.out.println("d value: "+ref2.d); ref2.disp(); System.out.println("K value: "+ref2.k); ref2.test(); } }

o/p:

b value: true running view() of R1 K value: 12 runing test() of D1

d value: 45.67 running disp() of S1 K value: 12 runing test() of D1

**Constructor:**

**Class Diagram:**

|  |  |
| --- | --- |
| Sample1 | Class name |
| Name : type | Data member |
| Name(); return type | Function member & constructor |

**Constructor Calling System in Inheritance:**

* In an inheritance program the sub class constructor should make a call to the constructor of superclass.
* The call can be made either implicitly or explicitly.
* Sub class can call super class in the using “super () “statements.
* If compiler makes a call to super class is called as implicitly, implicitly call zero argument constructor ,in other words Compiler can call only the 0 arguments of the super class constructor ,if super class constructor designing parameter then compiler not make a implicitly call , we should go for explicitly call.

**Use of “super ()” statement and Rules:**

* “super ()” statements used to make a call to the super class constructor from sub class constructor.
* “super ()” statements should be used only inside the constructor body.
* “super ()” statements should be the first statements of constructor body; it can’t be used anywhere else.
* Multiple super statements are not allowed in the constructor.

**Constructor Chaining:**

* Constructor chaining is a phenomenon where subclass constructor makes a call the super class constructor; the super class constructor makes a call to its super class constructor. Constructor chaining can be done either implicitly or explicitly.

**Q. Why java doesn’t support multiple Inheritances?**

* Sub class constructor can’t call more than one super class constructor because multiple “super ()” statements are not allowed.
* The multiple inheritances lead to the ambiguity of diamond problem because super class properties can’t be inherited to some sub class in two different classes.

**Object**

**Diamond**

super();

Class B

Class A

Class C

***Example of Implicitly Constructor Chaining:***

class D1 { D1() { // zero argument parameterize constructor System.out.println("runing const of D1 "); } } class S1 extends D1{ S1() { System.out.println("running const of S1"); } } class MC5 { public static void main(String[] args) { S1 ref1=new S1(); } }

o/p: here no need to call super class constructor its automatically call whenever sub class create instance or call runing const of D1 running const of S1

***Exam of Explicitly constructor chaining:***

class D1{ D1(int arg1) { System.out.println("runing const of D1 "); System.out.println("arg1 value: "+arg1); } }

class S1 extends D1{ S1() { super(25);//calling to the super class constructor System.out.println("running const of S1"); } }

class MC5 { public static void main(String[] args) {

S1 ref1=new S1(); } }

o/p:

runing const of D1

arg1 value: 25

running const of S1

**Use of “this ()” statement and Rules:** Constructor of a class can call another constructor of same class using “this()” statements

* “this()” statements can be used call constructor of the current class, it can’t call the constructor of super class
* “this()”statements can be used to call method 0 argument or parameterize const.
* “this()”statements should be used only In the constructor body and it must be the first statements of the constructor. Multiple this statements are not allowed
* Recursive constructor calls are not allowed.

Ex,

class S1 { S1() {//zero argument constructor this(10); System.out.println("running zero argument constructor "); }

S1(int arg) { this(2.5); System.out.println("running int argument constructor ");

}

S1(double arg) { System.out.println("running double argument constructor "); } }

class MC6 { public static void main(String[] args) { S1 obj=new S1(); System.out.println("Program ended"); } }

o/p:

running double argument constructor

running int argument constructor

running zero argument constructor

Program ended

***Example Use of “this” and “super” keyword both and their differences:***

class D1 { int k=34; }

class S1 extends D1{

int k=56; void disp(){ System.out.println("k value of s1: "+this.k);//k value is 56 System.out.println("k value of s1: "+super.k);//k value is 34 } }

class MC7 { public static void main(String[] args) {

S1 obj=new S1(); obj.disp();

System.out.println("Ended Program"); } }

o/p:

k value of s1: 56

k value of s1: 34

Ended Program

1. Java provides the special keywords by “super” which is used to refer super class properties in subclass calling or from subclass.

Java also provides the special keywords by “this” which is used to identify or refer to that current class global properties and differentiate to local variable as well.

1. The super keywords should be used either in non-static method context or constructor context.

“this” keyword should be used in the method context.

1. It can’t be used in static context.

But “this” keyword can be used in static context.

**Method Overloading:**

Defining multiple methods in a class with same name and different arguments is known as method overloading. This should work either in an argument type or a length of the argument

* In class we can overload both static method and non- static methods
* The overloaded methods are executed based on the method arguments
* Method overloading is used to achieved compile time polymorphism

When we go for Method Overloading?

While develop an application or function if the come cross an operation to be perform with different parameters then we go for method overloading.

Ex 1,

class D1 { void test() { System.out.println("running no arg test()"); }

void test(int arg) { System.out.println("running int arg test()"); System.out.println(" arg value: "+arg); }

void test(double arg) { System.out.println("running double arg test()"); System.out.println(" arg value: "+arg); } }

class MC1 { public static void main(String[] args) { D1 obj= new D1();

obj.test(); obj.test(25); obj.test(2.5);

System.out.println("End Program"); } }

o/p:

running no arg test()

running int arg test()

arg value: 25

running double arg test()

arg value: 2.5

End Program

Ex 2,

class S1 { void disp() { System.out.println("Runnig disp ()"); } }

class D1 extends S1{ void disp(int arg) { System.out.println("Runnig disp (int)"); } }

class MC2 { public static void main(String[] args) { D1 obj1=new D1();

obj1.disp(); obj1.disp(235); System.out.println("Hello World!"); } }

o/p:

Runnig disp ()

Runnig disp (int)

Hello World!

**Method Overriding:**

Inheriting in a method on super class changing in the implementation in the sub class is known as method over ridding

1. To override a method inheritance is must.
2. When sub class overrides the method of super class sub class should rewrite the same method signature of the super class, it should change the implementation.
3. The sub class can’t override bellow method of super class:-
4. Static method-because the static methods can’t be inheriting to subclass
5. Final non- static method: because the final key word doesn’t allow to change the method implementation.it can be inheriting to the sub class
6. Private non- static methods: because the private access is restricted only to the class where it is declare
7. Using method overriding we can achieved runtime polymorphism

When we go for Method Overriding?

When developing an apps if we come cross the situation where the functionality should be written with different implementation then we go for method overriding.

Ex 1,

class S1 { void disp() { System.out.println("Runnig disp ()"); } }

class D1 extends S1{ void disp() {//override method disp() of S1 System.out.println("Runnig disp() of D1"); } }

class MC2 { public static void main(String[] args) {

D1 obj1=new D1(); obj1.disp(); } }

o/p:

Runnig disp() of D1

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ex 2, Very Important for Interview**

class MN { void disp(int x,int y) { System.out.println("before swap"); System.out.println("x value: "+x); System.out.println("y value: "+y);

System.out.println("swapping using temp"); int temp=0; temp=x; x=y; y=temp;

System.out.println("x value: "+x); System.out.println("y value: "+y); } }

class NN extends MN{ void disp(int x,int y) {

System.out.println("before swap"); System.out.println("x value: "+x); System.out.println("y value: "+y);

System.out.println("swapping without using temp"); x=x+y;//30

y=x-y; x=x-y;

System.out.println("x value: "+x); System.out.println("y value: "+y); } }

class MC3 { public static void main(String[] args){

NN obj1=new NN(); obj1.disp();

int a=10; int b=20; obj1.disp(a,b);

System.out.println("End Programm"); } }

o/p:

before swap

x value: 10

y value: 20

swapping without using temp

x value: 20

y value: 10

End Programm

**Type casting:**

*Casting one type of function to another type is known as type casting.*

There are two type of casting:-

1. **Primitive Data type casting** (Simple): Datatype----casted--->datatype .
2. **Class type casting** (V.V.Imp): classtype--- casted -- >classtype.
3. **Primitive Data type casting:**

In the Primitive data type casting, a data type is casted to another data type. There are two type of Primitive data type casting:-

1. **Widening:** Casting lower data type to any of the higher data type is known as Widening. The widening can be done implicitly by compiler hence it is also known as implicit widening or auto-widening.
2. **Narrowing:** Casting higher data type to any of the lower data type is known as Narrowing. The Narrowing should be explicitly performed in the code. Compiler can’t perform narrowing because whenever narrowing is performed data loss happens.

Syntax, datatype variable= (data type) value;

int x= (int)10.2;🡪double is casted to int.//Narrowing, must explicit double y= (double)10;🡪int is casted to double. //Widening =10; //implicit

Prog1, \*\*\*\*\*\*\*\*\*\*\*\*

class MC1 { public static void main(String[] args) { int x1=(int)3.2; //double is casted to int.//Narrowing, must explicit double y1=(double)3; //int is casted to double. Widening System.out.println("x value: "+x1); System.out.println("y value: "+y1);

int a=23; double b=59.99;

int x2=(int)b; double y2=(double)a;

System.out.println("x2 value: "+x2); System.out.println("y2 value: "+y2); } }

o/p:

x value: 3 y value: 3.0 x2 value: 59 y2 value: 23.0

\*\*\*\*\*\*\*\*\*\*\*

int x1=(int)3.2;//explicit narrowing double y1=3;//implicit widding

int x2=(int)b; //explicit narrowing double y2=a; //implicit widding

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pro2,

class NM{ int sqr(int num) { System.out.println("cal sqr of : "+num); int res=num\*num; return res; } }

class MC2 { public static void main(String[] args) { double x1=6.34; NM no=new NM(); int x2=no.sqr((int)x1); //cast x1 value to int and then pass to method argument

System.out.println("Result is: "+x2); } }

o/p:

Result is: 36

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* *While passing value to the method argument or constructor argument we can perform type casting .when returning a value of a method type casting can be perform.*

class MC3 { public static void main(String[] args) { char x='A'; int i=x;//implicit

System.out.println("x value: "+x); System.out.println("i value: "+i);

int j=78; char y=(char)j;//explicit

System.out.println("j value: "+j); System.out.println("y value: "+y); } }

o/p:

x value: A

i value: 65

j value: 78

y value: N

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*

\***without using java inbuilt keyword , convert “jspider” to “JSPIDER”:(MOST OF INTERVIEW QUESTION)**

class MC4 { static String upperCase(String arg){ char[] arr=arg.toCharArray();//converting string to char array char[] temp=new char[arr.length];

for (int i=0;i<arr.length ;i++ ){ int x=arr[i];//getting asccii valur of char temp[i]=(char)(x-32);//convert to upper case } String uprStr=new String(temp);//converting char aray to string return uprStr; }

public static void main(String[] args) { String str="jspider"; System.out.println("Giving str: "+str); String s1=upperCase(str); System.out.println("Giving str in upper Case: "+s1);

}

}

o/p:

Giving str: jspider

Giving str in upper Case: JSPIDER

1. **Class type casting:** Casting one class type to another class type is known as Class type Casting.

Prog1, Class S1{ int k=23; char x='j'; void test() { System.out.println("running test() of S1"); } } class D1 extends S1{ boolean b=true; float f=21.34f; void disp() { System.out.println("running disp() of D1");

}

}

class MC5 { public static void main(String[] args){

S1 ref1; //ref variable of type of S1 ref1=(S1) new D1(); //up casting, cast D1 to S1

D1 ref2; //rerf variable of D1 type ref2=(D1) new S1(); //down casting, this line goes error like Exception in thread }

}

o/p Exception in thread -------- S1 can’t casted to D1…..

To perform to class type casting we have to fulfill provide rules:-

1. The classes should have is a relationship(inheritance)
2. The object should have the properties of the class to which we have to casted
3. Class type casting can be done in two ways
4. Up casting
5. Down casting
   * + - Casting subclass type to super class type is known as up casting. Up casting can be done either implicit or explicit. The implicit casting done by compiler
       - Casting super class type to sub class type is known as   
         down casting. The down casting should be done explicit in the code.
       - Down casting should be perform only on the object which is already up casting
6. During compilation the compiler check the class type casting statements, if classes having is a relationship then compiler compiles statements otherwise compiler goes an error in compatible type.

\*classcastEx is a type of RuntimeExp which through by jvm and runtime.

**Why this Exception is occurred?**

Whenever an object of a class type is casted to another class type which is not having the properties in the object then jvm throughs classcastExp

**Why compiler doesn’t detect at compile time?**

Because os the compiler only check the is a relationship not the properties

**How to avoid ClassCastExp?**

The ClassCastExp can be avoided by using “instanceof” operator.

\*We can access only the member of which class is casting another class.

Ex, of above class.

System.out.println("k value: "+ref1.k);

ref1.test();

* When an instance of a class is casted to another class then we can access only those properties the class type which is casted.
* If a ref variable type of super class, to that ref variable we can assign any of its sub class object because of implicit up casting. In other word, a super class ref variable can point any of the sub class objects

Simple111

Demo1

Run1

Sample1 ref1; ref1=new Sample1 (); ref1=new Demp1 (); //implicit ref1=new Run1 (); //implicit

Run1 r1; r1=new run1; r1= new (run1) D1 // not possible because D1 have not property of R1 r1= new (run1) S1 // not possible because S1 have not property of R1

Ex,

class Sample1 { int k=23; char x='j'; void test() { System.out.println("running test() of Sample1"); } }

class Demo1 extends Sample1{ boolean b=true; float f=21.34f; void disp() { System.out.println("running disp() of Demo1"); } }

class Run1 extends Demo1{ boolean c=true; double d=21.34f; void view() { System.out.println("running view() of Run1"); } }

class MC6 { public static void main(String[] args) {

Sample1 {

}

Run1 r1; //ref variable of type of Run1 r1=new Run1();//using r1 ref we can access run1,demo1,simple1 r1.test(); r1.disp(); r1.view();

Run1{

}

Demo1{

}

Demo1 d1; d1=r1;//using d1 ref we can access ,demo1,simple1 d1.test(); d1.disp();

Sample1 s1; s1=r1;////using s1 ref we can access simple1 s1.test() } }

* If a method argument is class type, then during method invocation we can pass any of the sub class instances, if you pass super class instance then we get ClassCastException.

Ex,

class Sample1 { int k=23; char x='j'; void test() { System.out.println("running test() of Sample1"); } }

class Demo1 extends Sample1{ boolean b=true; float f=21.34f; void disp() { System.out.println("running disp() of Demo1"); } }

class Run1 extends Demo1{ boolean c=true; double d=21.34f; void view() { System.out.println("running view() of Run1"); } }

class Start { void execut(Sample1 arg){ System.out.println("running execut() of Stratt"); System.out.println("k value: "+arg.k); System.out.println("k value: "+arg.x); arg.test();

} }

class MC7 { public static void main(String[] args){ Start srt=new Start(); srt.execut(new Sample1()); System.out.println("---------------------"); srt.execut(new Demo1()); System.out.println("---------------------"); srt.execut(new Run1()); } }

**Down Casting:**

Ex, class Sample1 { int k=23; char x='j'; void test() { System.out.println("running test() of Sample1"); } }

class Demo1 extends Sample1{ boolean b=true; float f=21.34f; void disp() { System.out.println("running disp() of Demo1"); } }

class Run1 extends Demo1{ boolean c=true; double d=21.34f; void view() { System.out.println("running view() of Run1"); } }

class Start { void execut(Sample1 arg) { System.out.println("running execut() of Start"); System.out.println("k value: "+arg.k); System.out.println("x value: "+arg.x); arg.test();

//If you want to use the Demo1 Properties then you should first up casting then go to down casting Demo1 ref1= (Demo1)arg;//explicit down casting System.out.println("b value: "+ref1.b); System.out.println("f value: "+ref1.f); ref1.disp(); } }

class MC8{ public static void main(String[] args) { Start srt=new Start(); srt.execut(new Demo1()); //up casting } }

o/p: running execut method k value: 23 x value: j running test() method

b value true f value 21.34 running disp() method

**instanceof operator Demo:**

Ex, class Sample1 { int k=23; char x='j'; void test () { System.out.println("running test() of Sample1"); } }

class Demo1 extends Sample1{ boolean b=true; float f=21.34f; void disp() { System.out.println("running disp() of Demo1"); } }

class Run1 extends Demo1{ boolean c=true; double d=21.34f; void view() { System.out.println("running view() of Run1"); } }

class Start { void execut(Sample1 arg) { System.out.println("running execut() of Stratt"); System.out.println("k value: "+arg.k); System.out.println("x value: "+arg.x); arg.test();

if(arg instanceof Demo1){ Demo1 ref1=(Demo1)arg;//explicit Down casting

System.out.println("running execut() of Stratt"); System.out.println("b value: "+ref1.b); System.out.println("k value: "+ref1.f); ref1.disp(); }else{ System.out.println("instance doesn't have properties of Demo1"); } } }

class MC8{ public static void main(String[] args) { Start srt=new Start(); srt.execut(new Sample1()); //not done up casting } }

o/p:

running execut() of Stratt

k value: 23

x value: j

running test() of Sample1

instance doesn't have properties of Demo1

**Polymorphism**

An object showing different behavior at different stage of a its life cycle is known as polymorphism,

There are two type of polymorphism:

1. **Compile time polymorphism**
2. **Runtime polymorphism**

**1. Compile Time Polymorphism:**

In compile time polymorphism the method declaration is binded to the method definition at compile time by compiler.

Since the binding is happening at operation time it is known as compile time binding or static binding or early binding, for overloaded methods ,static methods and final methods compile time binding happens. Hence this method is example for compile time polymorphism.

**2. Runtime Polymorphism:**

In run time polymorphism the method declaration is binded to the method definition by the JVM during execution. Since binding is happening at runtime it is known as runtime binding or dynamic binding or late binding. The method overriding is an example for runtime polymorphism since binding happens at execution time.to achieved runtime polymorphism we have to fulfill the following concept.

1. Inheritance
2. Method overriding
3. Up casting

Whenever we create an instance of a subclass which is having override methods, if we refer that instance either by subclass ref or by super class methods we get override implementation only.

Ex,

class Sample1 {

int k=23; char x='j'; void test() { System.out.println("running test() of Sample1"); } }

class Demo1 extends Sample1{

boolean b=true; float f=21.34f; void test(){ System.out.println("running test() of Demo1"); } }

class MC1 { public static void main(String[] args) {

Sample1 srt=new Demo1(); //implicit up casting srt.test } }

o/p:

running test() of Demo1

**Example of compile time Polymorphism:**

class A { void test ()//declaration object { === //method body }

**Heap Area**

void test(int arg)//overloaded method { === } }

Method

Method

test() **Method Area**  test(int arg)

We can call it like

A a1=new A(); a1.test(); a1.test(10);

*Methods are binding based on the argument.*

**Example of Run time Polymorphism:**

Class A{ Void test() { ==== } } Class B extends A{ Void test()//override methods { ==== } }

test () test()

class A class B

Method body

Method body

A instance B instance

Calling like A a1=new A (); B b1=new B ();

*Methods are binding based on the instance.*

class animal{

noise()

}

Ex, class Animal { void noise(){ System.out.println("all animal makes noise"); } } class Cat extends Animal { void noise(){//override noise() of Animal class System.out.println("mew mew"); } } class Dog extends Animal { void noise(){//override noise() of Animal class System.out.println("bow bow"); } } class Snake extends Animal { void noise(){//override noise() of Animal class System.out.println("Hiss Hiss"); } }

class tiger{

noise() //override

}

class snakel{

noise() //override

}

class dog{

noise() //override

}

class cat{

noise() //override

}

class Similar{ void makenoise(Animal arg){ arg.noise(); } }

class TestAnimal{ public static void main(String[] args){

Cat c1=new Cat(); Dog d1=new Dog(); Snake s1=new Snake(); Similar sim=new Similar(); sim.makenoise(s1);

}

}

o/p:

Hiss Hiss //sim.makenoise (s1); at this line whatever you pass like s1, d1, c1 the makenoise(Animal arg) call that object method. E.g. if you pass s1 then output will be “Hiss Hiss”, if you pass d1 then output will be “vow vow”.

**ABSTRACTION**

**Ex 1,**

abstract class Demo1{ int k=34; static int i=54; abstract void test(); static void disp(){ System.out.println("Running static disp() method!"); } } class MainClass1{ public static void main(String[] args){

//Demo1 obj1=new Demo1 //because you can’t create object of abstract class

//but you can call the static member of abstract class System.out.println("i value: "+Demo1.i); Demo1.test(); } }

o/p:

i value: 54

st method!

* Defining a method with declaration and implementation is known as concrete method. The concrete method are complete method which has both declaration and definition
* Defining a method with only declaration, without implementation is known as Abstract methods. The abstract method should be declare using abstract key word and it should be define in the abstract class only
* A class declare with abstract key word is known as abstract class
* An abstract class can have both abstract and concrete methods
* In abstract class we can define static member and non-static members
* We can’t create an instance of abstract class ,hence we can’t refer the non-static members of abstract class
* Abstract keyword can’t be compatible bellow keywords

1. Static
2. Final
3. Private

**Ex 2,**

abstract class D2{

int k=34; abstract void test(); void disp(){ System.out.println("disp method!"); } }

abstract class S2 extends D2 { //S1 should be declare as abstract, since it inherits an abstract method from D2 class }

class MC2 { public static void main(String[] args) {

}

}

o/p: it will compile perfectly .

**Ex 3,*****either implementation or call the class abstract to the subclass***

abstract class D2{

int k=34; abstract void test(); void disp(){ System.out.println("disp method of D2"); } }

class S2 extends D2{ //here not declare S2 as abstract because here methods are override and implementation void test(){ System.out.println("test method implementation"); } }

class MC2 { public static void main(String[] args) {

S2 obj1=new S2(); System.out.println("k value: " +obj1.k); obj1.test(); obj1.disp(); } }

o/p:

k value: 34

test method implimentation

disp method!

* If a class extends an abstract class then the class provide the implementation to all the abstract methods of the abstract class , otherwise the class should be declare as abstract.

**Ex 4,** ***for multi-level Inheritance***,

abstract class D3{

abstract void test(); abstract void disp(); }

abstract class S2 extends D3{ //S1 should be declare as abstract, since it inherits an abstract method from D2 class //and also S1 also inheriting to the subclass S3.

void test(){ System.out.println("test method implimentation"); } }

class S3 extends S2{ void disp(){ System.out.println("disp method implimentation"); } }

class MC3 { public static void main(String[] args){

S3 obj1=new S3(); obj1.test(); obj1.disp(); } }

o/p:

test method implimentation

disp method implimentation

**Ex 5,** ***hierarchical inheritance***

abstract class D4{

abstract void test(); abstract void disp(); } class S4 extends D4{

void test(){ System.out.println("test method implementation S4"); } void disp(){ System.out.println("disp method implementation S4"); } } class S5 extends D4{

void disp(){ System.out.println("test method implementation S5"); } void test(){ System.out.println("disp method implementation S5"); } }

class MC4 { public static void main(String[] args) {

S4 obj1=new S4(); obj1.test(); obj1.disp();

S5 obj2=new S5(); obj2.test(); obj2.disp(); } }

o/p:

test method implimentation S4

disp method implimentation S4

disp method implimentation S5

test method implimentation S5

Ex 6,

abstract class D4{

abstract void test(); abstract void disp(); }

class S4 extends D4{ void test(){ System.out.println("test method implimentation of s4"); }

void disp(){ System.out.println("disp method implimentation of s4"); } }

class S5 extends D4{

void disp(){ System.out.println("disp method implimentation of s5"); }

void test(){ System.out.println("test method implimentation of s5"); } }

class MC4 { public static void main(String[] args) {

D4 obj1; obj1=new S5();//methods are binded to s5 implemt @ runtime obj1.test(); obj1.disp(); } }

o/p:

test method implimentation of s5

disp method implimentation of s5

* If class declare as abstract it is not mandatory to declare as abstract method in the abstract class hence an abstract class not a pure abstract body. To get pure abstract body we should go for interfaces

Ex 7,

abstract class D5{ void test(){ System.out.println("Running test() of abstract class D5"); }

class S6 extends D5{

}

class MC5 { public static void main(String[] args) {

S6 obj1=new S6(); obj1.test(); } }

o/p:

Running test() of abstract class D5

**Interfaces:**

Interface interface\_name{//🡨declaration

abstract void test(); abstract void disp(); body

} class class\_name implements interface\_name{

Override abstract methods }

**Ex 1,** interface Pen { abstract void write(); }

class BallPen implements Pen{ //sub class or implementation class ======= ======= }

class Main{

public static void main(String[] args){ Pen ref1; BallPen p1=new BallPen(); ref1=new Pen();//can’t declare Pen’s reference p1.write();//you can declare

Or

ref1=new BallPen();//you can declare

ref1.write();

***Example 2 of Multiple Interfaces:***

|  |  |
| --- | --- |
| interface Pencil{  abstract void write();  } | interface eraser{  abstract void eraser();  } |

class NatrajPencil implements Pencil, Eraser{ public void write(){

=== } public void eraser() {

==== } }

* An interface is a java type which is use to declare only abstract methods.
* An interface method should be abstract in nature, and it should be public in access.
* Inside interface body we can’t create concrete methods.
* Interface variables should be static and final, we can’t define non-static variables.
* By default the interface variable are static final and public.
* By default interface methods are abstract and public.
* Inside interface we can’t develop constructer.
* We can’t create an instance of interface type, we can declare a ref variable of interface type.
* Class should provide the implementation to the interface methods by using “implements” keywords.
* A class can implement any no of interfaces.
* The class which provides an implementation to the interface methods is known as implementation class.
* Whenever a class implements an interface the class must provide implementation to all the abstract methods of interfaces otherwise class should be declare as abstract.
* An interface can inherit of another interface by using extends keywords , an interface can’t inherits from a class

**Ex 3, Single**

interface D1{

int k=32; void test(); } class S1 implements D1{ public void test(){ System.out.println("test implents!"); } } class MC1{ public static void main(String[] args) { implements-🡪

D1(interface)//super interface

System.out.println("k value: "+D1.k); S1 obj=new S1(); obj.test(); } }

S1//implementation class

**Ex 4, Multi-level**

D2(interface)

interface D2{ void test(); } interface D3 extends D2{ void disp(); } extends class S2 implements D3{ public void disp(){ System.out.println("disp implents! in S2"); }

D3(interface)

public void test(){ System.out.println("test implents! in S2"); } implements }

class MC2{ public static void main(String[] args) { S2 obj=new S2(); obj.test(); obj.disp(); } }

S2(class)

**Ex 5, Multiple**

D3(interface)

D2(interface)

interface D2{ void test(); } interface D3{ void disp(); } class S2 implements D3,D2{ //multiple interface public void disp(){ System.out.println("disp implents in S2"); }

S2(class)

public void test(){ System.out.println("test implents! in S2"); } }

class MC3{ public static void main(String[] args) {

S2 obj=new S2(); obj.test(); obj.disp(); } }

**Ex 6, up casting with create Reference variable of Interface type**

interface D2{ void test(); } interface D3{ void disp(); } class S2 implements D3,D2{ public void disp(){ System.out.println("disp implents! in S2"); } public void test(){ System.out.println("test implents! in S2"); } }

class MC3{ public static void main(String[] args) {

S2 obj=new S2();

D2 ref1;//reference variable of interface type ref1=obj;//upcasting, class is casted to interface type ref1.test();

D3 ref2; //reference variable of interface type ref2=obj; //upcasting, class is casted to interface type ref2.disp();

object

} }

**Ex 7, extends and implements**

D1(interface)

Test();

interface D1{ //interface type

int k=32; void test(); } class R1{ //class type void view(){ implements System.out.println("view() "); } extends }

R1(class)

View();

S1(imple…class)

class S1 extends R1 implements D1{ public void test(){ System.out.println("test implents!"); } }

class MC4{ public static void main(String[] args) {

System.out.println("k value: "+D1.k);

S1 obj=new S1(); obj.test(); obj.view(); } }

**Abstraction and Interfaces:**

***A reference variable can of tow type;***

**1) Class type:** If the reference variable is class type then we can assign either class type instance or sub class type instance.

2**) Interface type:** If the ref variable is interface type or abstract class then we can assign only the instance of implementation class.

**Abstraction:**

* Abstraction is a process of hiding the class implementation from its utilization. The abstraction specified that don’t show how object functionality is implemented but provide an interface to use the functionality.
* To achieved the abstraction we have to follow the below steps:

1. Generalize the behavior of the classes in an interface.
2. Provide implementation in the class according to the specification
3. Refer that implementation of the class by using interface reference variable
4. The abstraction can be achieved either by using abstract class or interfaces
5. If we need pure abstraction then we should use interfaces. If we need concrete an abstract method then we should go for abstract class

**Advantages of abstraction:**

* We can achieved generalization
* We can hide the implementation
* We can achieve loose coupling between the objects.
* The abstraction is used to developed API’s(Application Programming Interfaces)
* The other major advantage of abstraction is any changes in the implementation will have very less impact on the utilization of the class.

This is small project implementation of banking transaction of saving and loan account using abstraction principle

**interface** IAccount {

**void** deposit(**double** amt);

**void** checkBalance();

**void** withdrawal(**double** amt);

}

**class** SavingAccount **implements** IAccount{

**int** accNum;

**double** accBal;

SavingAccount(**int** accNum,**double** accBal){

System.*out*.println("create a saving account");

**this**.accNum=accNum;

**this**.accBal=accBal;

}

**public** **void** deposit(**double** amt){

System.*out*.println("deposit amount: "+amt);

accBal=accBal+amt;

}

**public** **void** withdrawal(**double** amt){

System.*out*.println("withdrwal amt: "+amt);

accBal=accBal-amt;

}

**public** **void** checkBalance(){

System.*out*.println("current bal: "+accBal);

}

}

**class** LoanAccount **implements** IAccount{

**int** accNum;

**double** accBal;

LoanAccount(**int** accNum, **double** accBal){

System.*out*.println("creating laon acc");

**this**.accNum=accNum;

**this**.accBal=accBal;

}

**public** **void** deposit(**double** amt){

System.*out*.println("deposit amt"+amt);

accBal=accBal-amt;

}

**public** **void** withdrawal(**double** amt){

System.*out*.println("withdrwal amt: "+amt);

accBal=accBal+amt;

}

**public** **void** checkBalance() {

System.*out*.println("outstanding Balance: "+accBal);

}

}

**class** AccManager {

IAccount openAcc(**char** type, **double** initAmt){

IAccount acc1=**null**;

**if**(type=='s'){

System.*out*.println("oppening saving account");

acc1=**new** SavingAccount(9876543, initAmt);

}**else** **if**(type=='l'){

System.*out*.println("oppening loan account");

acc1=**new** LoanAccount(9876543, initAmt);

}

**return** acc1;

}

}

**class** TestAccount {

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

AccManager am1=**new** AccManager();

IAccount myAcc=am1.openAcc('l',50000.00);

myAcc.checkBalance();

myAcc.deposit(5000.00);

myAcc.checkBalance();

myAcc.withdrawal(10000.00);

myAcc.checkBalance();

}

}

o/p:

oppening loan account

creating laon acc

outstanding Balance: 50000.0

deposit amt5000.0

outstanding Balance: 45000.0

withdrwal amt: 10000.0

outstanding Balance: 55000.0

**Loose-Coupling:**

IAccount

deposit(); withdrawal(); checkBalance();

TestAccount

AccManager

LoanAcc

SavingAcc

Object creation layer

Object utilization layer

Object Implementation layer

* **Loose-Coupling:** Loose-Coupling is a concept of developing programs in a different layer in such a way that object utilization is independent of its implementation. It is develop in three different layers.

1. ***Object implementation layer:*** In this layer the functionality of the objects are implanted in the class, the declaration of the function are done in the interfaces. Here generalization concept is used
2. ***Object creation layer:*** In this layer the object of the class is created based on the requirement or need. Here factory design pattern is used
3. ***Utilization layer:*** in this layer the functionality of the object is used by using interface reference variables

* **The advantage of Loose-Coupling:**

Any changes in the implements layer will not have impact on the utilization layer. Hence modification and enhancement will be easy.

class Person{

Vehicle(interface)

start();

move();

stop();

Vehicle v1;

void initVehicle(Vehicle v1){ this.v1=v1; }

void cummut(){

v1.start(); v1.move(); implements v1.stop(); } }

Bike

Car

Car m800=new Car();

Bike rf=-new Bike();

Person p1=new Person();

p1 initVehicle(rf);

p1.cummut();

* If an instance of a class depends on the implements of another class instance directly then it is known as tight coupling. Such type of programming will always consumes more times and effort maintain and updating the code. Any changes in the implementation of dependent object will impact on the other object. to overcome this we have to use loose coupling which is done by using interface reference variable

**Packages:**

**Java Package:**

* Collection of java programs developed of particular functionality this known as package
* The package can have some package
* Collection of java package is known java library
* Is package should be declare using package declaration statements

The syntaxes are:

package packageaName;

or

package pack1.pack2;// one is parent 2nd one is child package

* The package declaration should be always the first line of the source file. In a source file we can have only one package declaration statements

***Project Structure:***

Project Name

Store only “.java” files Store only “.class” files

src

bin

Pn

P2

P2

P1

Pn

P1

🡪packages

..… .…

A.java

A.class

… … …. …. …. ….

Ex,

Project Name: BankApp

package deposit;//this is like P1 package

class SavingAccount { public static void main(String[] args) {

System.out.println("running saving account"); } }

package deposit;

class CurrentAccount{ //same package but different source file public static void main(String[] args) {

System.out.println("running current account"); } }

package loan;//this is like P2 package

class HomeLoan{ public static void main(String[] args) {

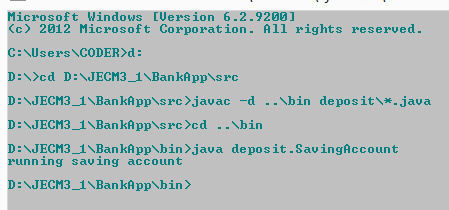
System.out.println("running HomeLoan class"); } }

package loan;

class AutoLoan {//this is also loan package but different source file public static void main(String[] args) {

System.out.println("running AutoLoan class"); } }

Output command and o/p is:



javac –d ..\bin deposit\\*.java

Comnd directory option location of the folder where .class files should store location of “.java” files

Compile all .java file in deposit folder

java packagename.classname🡪fully qualified class name

**Encapsulation:**

* Encapsulation is a process of binding the data members to the class body. Any data member used in the program should be declaring within the class body only. Encapsulation is used to protect the data member
* The member of the class can be restricted from other classes by using access specifier. Java provides four times of Access specifier:

1. **Private :**

A member having private access is restricted up to class level; it should be access only within the class where it is declared. The private member of the class can’t be access from outside the class body. The private is a very secure access.

1. **Package level:**

A member having package level access can be access from a class which belongs to same package. It can’t be access from outside the package.

1. **Protect :**

The protect members have a restriction up to class level, it can be access from the outside the package only by inheriting it.

The protect members are same as package level access but each members can be access outside the package.

1. **Public:**

The public members of a class have an access level beyond the package. It can be access from any packages. Public access has wider accessibility but less secure.

**Rule of encapsulation:**

1. Defining a class inside another class body is known as inner classes. For an inner classes we can define all the four access specifiers
2. The outer class should have either public access or package level access. We can’t declare outer class as private or protect
3. In a source file we can develop any no of java classes. If all the classes have a default access then we can give any name to the file. If any classes having public access then the file name should be the public class name. In a source file we can’t develop more than one public classes or interfaces.
4. In a class body the default access is package level. In an interface the default access is public
5. While overriding the methods the sub class has a permission to either return the same access level or to increase the access level but can’t reduce the access level

Public

Low accessibility but high security High accessibility but low security

Protect

Package

Private

1. The constructor define by compiler will always have same access as the class. For user define constructor we can provide any of the four access level.
2. If constructors are private then we can’t create an object of the class outside the class. Private constructors are used to provide restriction in the object creation.

Exm of private constructor.

**package** imptopics;

**public** **class** Calculator {

**private** Calculator(){

System.*out*.println("creating cal instance");

}

**void** divide(**int** n1, **int** n2){

System.*out*.println("dividing " +n1+"by "+n2);

**int** res=n1/n2;

System.*out*.println("result: "+res);

}

**public** **static** Calculator getInstance(){

**return** **new** Calculator();

}

}

**package** imptopics;

**public** **class** MainClass1 {

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

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

Calculator calci1=Calculator.*getInstance*();

calci1.divide(24, 6);

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

creating cal instance

dividing 24by 6

result: 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Singleton Class:**

**Ex,**

**package** imptopics;

**public** **class** Calculator {

**private** **static** **int** *count*;

**private** **static** Calculator *ref1*;

**private** Calculator(){

System.*out*.println("creating cal instance");

*count*++;

}

**void** divide(**int** n1, **int** n2){

System.*out*.println("dividing " +n1+"by "+n2);

**int** res=n1/n2;

System.*out*.println("result: "+res);

}

//return an instance of Calculator

**public** **static** Calculator getInstance(){

**if**(*count*==0){

*ref1*= **new** Calculator();

}

**return** *ref1*;

}

}

**package** imptopics;

**public** **class** MainClass1 {

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

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

Calculator calci1=Calculator.*getInstance*();

calci1.divide(24, 6);

Calculator calci2=Calculator.*getInstance*();

Calci2.divide(36, 4);

Calculator calci3=Calculator.*getInstance*();

Calci3.divide(91, 3);

System.*out*.println(calci1);

System.*out*.println(calci2);

System.*out*.println(calci3);

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

creating cal instance

dividing 24by 6

result: 4

dividing 36by 4

result: 9

dividing 91by 3

result: 30

imptopics.Calculator@19e0bfd

imptopics.Calculator@19e0bfd

[imptopics.Calculator@19e0bfd](mailto:imptopics.Calculator@19e0bfd)

* Singleton class is design pattern where it allows creating only one instance of the class. To design a singleton class we should define constructor as private so that the object creation is restricted
* The class must provide static methods which return the instance of the class. The method should have necessary logic where only one instance should be created or you can create a object with final keyword and return it;

**Library:** 🡪.jar (collection of classes)

java

lang

util

io

awt

Object

String

outputSt

Arraylis

Scanner

inputSt

All is stored in rt.jar file

…..

Whole thing in rt.jar file

**package** library1;

**import** java.lang.Object;//optional

/\*

\* Object class

\* ------------

\* root class in java class hierarchy

\*

\* 1) available in java.lang package

\* 2) contains no arguments constructor

\* 3) contains function members:

\* 1) toString()--> return type String

\* 2) hashCode()--> return type int value

\* 3) equals(Object)--> return type Boolean

\* 4) clone()--> return type object type

\* 5) finalize()-->return type void

\* 6) overloaded wait()--> return type void

\* 7) notify()--> return type void

\* 8) notifyAll()--> return type void

\*

\* e.g. Object o1;

\* o1=new Object();

\* String s1=o1.toString();

\* syso(s1)

\*/

**public** **class** Demo1 {

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

Object o1;

o1=**new** Object();

String s1=o1.toString();

//return string representation of the instance created

//fullyqualifiedclassname@address

//packagename.classname@address

System.*out*.println(s1);//java.lang.Object@addrss

}

}

o/p:

java.lang.Object@19e0bfd

*Example of toString method not overrided in a user define class*

ex,

**package** library1;

**public** **class** Sample1 {

}

**package** library1;

**public** **class** Demo2 {

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

// **TODO** Auto-generated method stub

Sample1 obj1=**new** Sample1();

String s1=obj1.toString();

// library1.Sample1@address

System.*out*.println(s1);

}

}

o/p:

library1.Sample1@19e0bfd

* Example of toString method overrided in user define class

**package** library1;

**public** **class** Sample1 {

**public** String toString(){

**return** "i overrided method";

}

}

**package** library1;

**public** **class** Demo2 {

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

// **TODO** Auto-generated method stub

Sample1 obj1=**new** Sample1();

String s1=obj1.toString();

System.*out*.println(s1);

}

}

o/p:

i overrided method

* A ref variable should either point to an object or null, if we print a ref variable which is pointing to an instance will implicit call toString method of the instance. If the toString method is overrided we should get overrided implementation otherwise we should get object class implementation
* If we print a ref variable pointing to null then output should be null.

**package** library1;

**public** **class** Sample1 {

**public** String toString(){

**return** "i overrided method again";

}

}

**package** library1;

**public** **class** Demo2 {

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

// **TODO** Auto-generated method stub

Sample1 obj1=**new** Sample1();

// i overrided method again

System.*out*.println(obj1);

}

}

o/p:

i overrided method again

**String class:**

* String is a class type define in the “java.lang” package
* String class is used to create string object which helps us to store string values
* In String class constructor are overloaded e.g.

1. 0 argument constructor
2. String type constructor
3. Char array type constructor

etc.

* String class is a final class hence it can’t have subclasses
* String class is immutable class, once we create string object we can’t change the state of the object
* String is a comparable type class, hence string object can be compare sorted
* String instance can be created in to ways

1. Without using new operator
2. With using new operator

* String Object created without new operator are storing in string constant area of heap which doesn’t allow duplicate string
* The string object which is created with new operator are stored in string non constant pool area of heap which allows duplicate strings
* In String class the following methods of Object classe is overrided

1. toString()
2. hashCode()
3. equals(Object)

**Example of Constant and Non-constant pool of String Class:**

**Ex 1,**

**package** library1;

**public** **class** Demo3 {

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

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

String s1="jspiders";//constant pool

String Pool

**Non-constant pool (new)**

“jspider”🡪s2

“jspider”🡪s4

**Constant pool**

“jspider”🡪s1

s3

String s2=**new** String("jspiders");//non CP

String s3="jspiders";

String s4=**new** String("jspiders");

//address of reference variable

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

System.*out*.println(s1==s3);

System.*out*.println(s2==s4);

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

//compare vlaue of object

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

System.*out*.println(s1.equals(s3));

System.*out*.println(s2.equals(s4));

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

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

false

true

false

false

true

true

true

true

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ex 2,**

**package** library1;

**public** **class** Demo4 {

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

String s1="java";

String s2="java";

System.*out*.println(s1);

System.*out*.println(s2);

s2="j2ee";

System.*out*.println(s1);

System.*out*.println(s2);

}

}

o/p:

java

java

java

j2ee

**StringBuilder and StringBuffer:**

\* These are mutable

\* These are final

\* Can’t compare, not a comparable type

\* StringBuffer methods are synchronize and thread safe

\* StringBuilder methods are not synchronize and not thread

**package** library1;

**public** **class** Demo6 {

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

// **TODO** Auto-generated method stub

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

StringBuffer sb1=**new** StringBuffer("developer");

System.*out*.println(sb1);

System.*out*.println("appeneing char");

sb1.append('s');

System.*out*.println("insert char");

sb1.insert(3,'w');

System.*out*.println(sb1);

System.*out*.println("replace char");

sb1.setCharAt(0,'D');

System.*out*.println(sb1);

System.*out*.println("reversing string");

sb1.reverse();

System.*out*.println(sb1);

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

developer

appeneing char

insert char

devwelopers

replace char

Devwelopers

reversing string

srepolewveD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Array (**class type**)**

1. **Data type array**

e.g. a) int[] arr= new int[5];

store 5 int value

b) char[] arr2= new char[5];

store 5 char value

1. **Class type array**

e.g. Demo1[] arr1=new Demo1[5];

//store 5 instance of Demo1 class

//storing instances

arr1[0]=new Demo1();

arr1[1]=new Demo1();

arr1[2]=new Demo1();

arr1[3]=new Demo1();

arr1[4]=new Demo1();

sop(arr1[0]);//call toString()

arr1[0].datamember//refer data member of 1st instance

arr1[3].datamember//refer data member of 4st instance

**Ex Array of class type where toString function is not overrided.**

**Ex,**

**package** lib2;

**public** **class** Demo1 {

**int** k=23;

}

**package** lib2;

**public** **class** Main {

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

Demo1[] arr1;//declaring array of class type

arr1=**new** Demo1[5];//initialize 5 memory location hold instances of Demo1 class

arr1[0]=**new** Demo1();

arr1[1]=**new** Demo1();

arr1[2]=**new** Demo1();

arr1[3]=**new** Demo1();

arr1[4]=**new** Demo1();

System.*out*.println("total elements: "+arr1.length);

System.*out*.println("Array elements: ");

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

System.*out*.println(arr1[i]);

}

}

}

o/p:

total elements: 5

Array elements:

lib2.Demo1@19e0bfd

lib2.Demo1@139a55

lib2.Demo1@1db9742

lib2.Demo1@106d69c

lib2.Demo1@52e922

**Ex Array of class type where toString method is overrided:**

**Ex,**

**package** lib2;

**public** **class** Demo1 {

**int** k=23;

**public** String toString(){

**return** "Demo1 [k="+k+"]";

}

}

**package** lib2;

**public** **class** Main {

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

Demo1[] arr1;//declareing arry of class type

arr1=**new** Demo1[5];//initialize 5 mem loc hold inst of demo1 class

arr1[0]=**new** Demo1();

arr1[1]=**new** Demo1();

arr1[2]=**new** Demo1();

arr1[3]=**new** Demo1();

arr1[4]=**new** Demo1();

System.*out*.println("total elements: "+arr1.length);

System.*out*.println("Array elements: ");

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

System.*out*.println(arr1[i]);

}

}

}

o/p:

total elements: 5

Array elements:

Demo1 [k=23]

Demo1 [k=23]

Demo1 [k=23]

Demo1 [k=23]

Demo1 [k=23]

**Class type array with inheritance:**

**Ex,**

**package** lib2;

**public** **class** Demo1 {

**int** k=23;

**public** String toString(){

**return** "Demo1 [k="+k+"]";

}

}

**class** Sample1 **extends** Demo1 {

**public** String toString(){

**return** "Sample1 [k="+k+"]";

}

}

**package** lib2;

**public** **class** Main {

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

Demo1[] arr1;//declareing arry of class type

arr1=**new** Demo1[5];//initialize 5 mem loc hold onst of demo1 class

arr1[0]=**new** Demo1();

arr1[1]=**new** Sample1();

arr1[2]=**new** Demo1();

arr1[3]=**new** Sample1();

arr1[4]=**new** Demo1();

System.*out*.println("total elements: "+arr1.length);

System.*out*.println("Array elements: ");

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

System.*out*.println(arr1[i]);

}

}

}

o/p:

total elements: 5

Array elements:

Demo1 [k=23]

Sample1 [k=23]

Demo1 [k=23]

Sample1 [k=23]

Demo1 [k=23]

**Ex,**

**package** lib2;

**public** **class** Student {

**int** id;

String name;

Student(**int** id,String name){

**this**.id=id;

**this**.name=name;

}

**public** String toString(){

**return** id+ "\t"+ name;

}

}

**package** lib2;

**public** **class** NoticeBoard {

**void** display(Student[] arg){

System.*out*.println("student details");

System.*out*.println("ID\tName");

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

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

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

}

}

}

**package** lib2;

**public** **class** MainClass2 {

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

// **TODO** Auto-generated method stub

Student[] stArr1;//declareing arry of class type

stArr1=**new** Student[5];//initialize 5 mem loc hold inst of demo1 class

stArr1[0]=**new** Student(123,"ram");

stArr1[1]=**new** Student(234,"sam");

stArr1[2]=**new** Student(345,"kam");

stArr1[3]=**new** Student(456,"lam");

stArr1[4]=**new** Student(567,"pam");

NoticeBoard obj1=**new** NoticeBoard();

obj1.display(stArr1);

}

}

student details

ID Name

--------------------

123 ram

234 sam

345 kam

456 lam

567 pam

**Ex,**

**package** lib2;

**public** **interface** Pen {

**void** write();

**void** refill();

}

**package** lib2;

**public** **class** BallPen **implements** Pen {

String inkcolor;

**double** price;

BallPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with BallPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "BallPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** MarkerPen **implements** Pen {

String inkcolor;

**double** price;

MarkerPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with MarkerPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "MarkerPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** SketchPen **implements** Pen{

String inkcolor;

**double** price;

SketchPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with SketchPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "SketchPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** MainClass4 {

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

BallPen[] arr1=**new** BallPen[5];

//array of ball pen store only ballpen instance

SketchPen[] arr2=**new** SketchPen[5];

//array of SketchPen store only SketchPen instance

Pen [] arr3;

arr3=**new** Pen[5];// initialize 5 memory location

arr3[0]=**new** BallPen("red",10.25);

arr3[1]=**new** MarkerPen("blue",20.25);

arr3[2]=**new** SketchPen("white",25.25);

arr3[3]=**new** BallPen("yellow",10.25);

arr3[4]=**new** MarkerPen("pink",10.25);

System.*out*.println("Array elements");

**for**(**int** i=0;i<arr3.length;i++){

System.*out*.println(arr3[i]);

}

}

}

o/p:

Array elements

BallPen[color: red, price: 10.25]

MarkerPen[color: blue, price: 20.25]

SketchPen[color: white, price: 25.25]

BallPen[color: yellow, price: 10.25]

MarkerPen[color: pink, price: 10.25]

Ex,

**package** lib2;

**public** **interface** Pen {

**void** write();

**void** refill();

}

**package** lib2;

**public** **class** BallPen **implements** Pen {

String inkcolor;

**double** price;

BallPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with BallPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "BallPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** MarkerPen **implements** Pen {

String inkcolor;

**double** price;

MarkerPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with MarkerPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "MarkerPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** SketchPen **implements** Pen{

String inkcolor;

**double** price;

SketchPen(String inkcolor,**double** price){

**this**.inkcolor=inkcolor;

**this**.price=price;

}

**public** **void** write(){

System.*out*.println("writting with SketchPen");

}

**public** **void** refill(){

System.*out*.println("rfliing the ink");

}

**public** String toString(){

**return** "SketchPen[color: "+inkcolor+", price: "+price+"]";

}

}

**package** lib2;

**public** **class** MainClass5 {

**static** **void** penStandDetails(Pen[] arg){

**int** count=0;

**int** bpCount=0;

**int** mpCount=0;

**int** spCount=0;

System.*out*.println("Given pen stand capacity: "+arg.length);

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

**if**(arg[i] !=**null**){

count++;

**if**(arg[i] **instanceof** BallPen){

bpCount++;

}

**else** **if**(arg[i] **instanceof** MarkerPen){

mpCount++;

}

**else** **if**(arg[i] **instanceof** SketchPen){

spCount++;

}

}

}

System.*out*.println("total: "+count);

System.*out*.println("bp: "+ bpCount);

System.*out*.println("mp: "+ mpCount);

System.*out*.println("sp: "+ spCount);

}

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

//BallPen[] arr1=new BallPen[5];

//array of ball pen store only ballpen instance

//SketchPen[] arr2=new SketchPen[5];

//array of SketchPen store only SketchPen instance

Pen [] arr3;

arr3=**new** Pen[10];// initialize 5 memory location

arr3[0]=**new** BallPen("red",10.25);

arr3[2]=**new** MarkerPen("blue",20.25);

arr3[4]=**new** SketchPen("white",25.25);

arr3[6]=**new** BallPen("yellow",10.25);

arr3[8]=**new** MarkerPen("pink",10.25);

*penStandDetails*(arr3);

}

}

**o/p:**

Given pen stand capacity: 10

total: 5

bp: 2

mp: 2

sp: 1

**Collection:**

|  |  |
| --- | --- |
| **Draw Back Arrays:**   * Size fixed * Homogenies type(same type) * Any DS implementation * No building algorithms * No built in forces to operate on elements | **Collection Frame Works or Library’s:**   * Resizable in nature * Heterogeneous type * Implements some DS * Built in algorithms  1. Short 2. Search  * Hues built in forces to operate on elements |

**Q) What is a Collection?**

**Ans.** Collection is an object which holds references of group of objects. The collection is also known as container which can hold any type of objects.

The collection can hold the object in three types:

1. **List**
2. **Set**
3. **Queue**

**Q) What is Collection Framework?**

**Ans.** Collection framework are Library of java classes and interfaces which is used to store, manage, manipulate , and remove group of objects, the library provides necessary function to do the operation on the objects store in the collection

**Fundamental of Collection:**

1. Collection is container which can holds any type of object and any number of objects.
2. It can resize on its own based on the requirements.
3. It is used to represent a group of objects as a single entity
4. Any instances added to the collection is casted to **Object class** type and stored inside the container
5. The object inside the container always shows **Object class** properties
6. Whenever an instance retrieve or remove from the container, the instance will show object class properties
7. If we have to get specific class properties then we should perform down casting explicitly.

elements

elements

**Add operation implicit Remove or retrieve downcast explicitly**

add C1 element to get specific properties

C2

add C2element

C1

…….

Cn

add Cn element

Container

**How works mail:**

**Login**

Mails are Stored

Code

Series

SQL array

request

Response

dispatch

**Web Page (**client**)**

**1. List Type Collection:**

* List is a type of collection where object are store linearly
* In list each elements are store with index
* List preserves the insertion order. Insertion null allow in the list.
* Duplicate elements can be insertion

**l1 (ref)**

l1.add(e1)

l1.add(e2)

l1.add(e3)

.

..

l1.add(en)

l1.size();

***To add an element to any collection:***

boolean add (object e);.

***To find total elements size in any collection:***

int size();

List collection class diagram:

Collection(i)

**Inherit**

List(i)

**Implements Implements Implements**

LinkedList(c)

Vector(c)

Array(c)

**Pro,**

**package** lib3;

**public** **class** Student {

**int** id;

String name;

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

**this**.id = id;

**this**.name = name;

}

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + "]";

}

}

**package** lib3;

**public** **class** Pen {

String color;

**double** price;

**public** Pen(String color, **double** price) {

**this**.color = color;

**this**.price = price;

}

**public** String toString() {

**return** "Pen [color=" + color + ", price=" + price + "]";

}

}

**package** lib3;

**public** **class** NoteBook {

**int** pages;

**double** price;

**public** NoteBook(**int** pages, **double** price) {

**this**.pages = pages;

**this**.price = price;

}

**public** String toString() {

**return** "NoteBook [pages=" + pages + ", price=" + price + "]";

}

}

**package** lib3;

**public** **class** Employee {

**int** id;

String name;

**public** Employee(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + "]";

}

}

**package** lib3;

**import** java.util.ArrayList;

/\*

\* ArrayList l1=new ArrayList();

\* Vector v1=new Vector();

\*

\* List ls1;

\*

\* ls1=new ArrayList();

\* Or

\* ls1=new Vector()

\*

\* ls1=new LinkedList();

\*/

**public** **class** ListDemo1 {

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

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

ArrayList l1=**new** ArrayList();//creating empty list type object

//add elements to list

l1.add(**new** Pen("black",10.25));

l1.add("jspider");

l1.add(**new** Student(123,"Hardik"));

l1.add(**null**);

l1.add(**new** NoteBook(25,24));

l1.add("jspider");

**int** tot\_ele=l1.size();

System.*out*.println("Total elements: "+tot\_ele);

//retrieve elements from list, Must be down casting

Student st1=(Student)l1.get(2);

System.*out*.println("Student id: "+st1.id);

tot\_ele=l1.size();

System.*out*.println("Total elements: "+tot\_ele);

//remove elements from list by remove, it is also done with down casting

Student st2=(Student)l1.remove(2);

tot\_ele=l1.size();

System.*out*.println("Total elements: "+tot\_ele);

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total elements: 6

Student id: 123

Total elements: 6

Total elements: 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## a) Implementation of ArrayList:

* ArrayList is a implementation class of List Interface
* ArrayList class implements three Marker interface (Empty interface)

1. Serializabel
2. Cloneable
3. RandomAccess

* ArrayList developed with resizable array data structure\*\*\*
* ArrayList object is created to the default capacity of ten
* The ArrayList grows with formula

new\_cap = (old\_cap\*3/2)+1

* ArrayList has three constructor

1. 0 argument constructor
2. Int argument constructor
3. Collection type argument constructor

Syntax

class ArrayList implements List, Serializable, Cloneable, RandomAccess{

}

ArrayList l1=new ArrayList(); l1(ref)--initially

l1.add(e1);

l1.add(e2);

l1.add(e3);

l1.add(e4);

.

. l1(ref)--finally

l1.add(e10);

l1.add(e11);

.

.

l1.add(en); new\_cap = (10\*3/2)+1 apply this formula ArrayList grow with size 16

**Advantage of ArrayList:**

* Accessing elements are easy
* Retrieval operation is faster
* Accessing any elements randomly takes same times

**Disadvantage of ArrayList:**

* Insertion and deletion operation will slow. Since elements should be shifted
* If ArrayList grow frequently then it consumes more memory and takes time to grow

# b) Implementation of Vector:

* The vector implements is exactly similar to ArrayList
* It is legacy class(old class) which is developed in JDK 1.0 version
* The vector methods are synchronized
* Vector class is a thread safe class
* Vector grows with a formula

new\_cap=old\_cap\*2

# c) Implementation of LinkedList:

* LinkedList is a implementation class of List interface
* LinkedList is also an implementation class of Queue. Hence LinkedList will have both List features and Queue features
* The LinkedList implements two Marker interfaces

1. Serializable
2. Cloneable

* LinkedList is developed with **doubly LinkList data structure**
* LinkedList grows with one elements at a time
* The LinkedList has 0 arguments constructor

**Advantage of LinkedList:**

* Insertion and deletion of elements will be faster
* It can be used to store as List and Queue also

**Disadvantage of LinkedList:**

* Retrieving of elements is slow

Note

1. Whenever we store an object which involves more retrieval option then we should use ArrayList
2. Whenever we want to store object where insertion as well as deletion are more then we should go for LinkedList

LinkedList l1=new LinkedList(); l1(ref)

l1.add(e1)

0 1 2 n-1

l1.add(e2)

|  |  |  |
| --- | --- | --- |
| PEA | e1 | NEA |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| PEA | en | NEA |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| PEA | e3 | NEA |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| PEA | e2 | NEA |
|  |  |  |

l1.add(e3)

: l1.add(en)

**2. Set Type of Collection:**

* Set is a type of collection which can store only unique elements
* Set collection doesn’t allow duplicate value
* Null insertion is allowed
* Set elements are store non linearly without index
* Set elements are retrieve randomly since it is not store with index

**Q) How set maintains uniqueness?**

**Answer.** The set maintains the uniqueness of the object based on the hash code of the object. Every object created in JVM will have a unique hash code number generated automatically based on the address of the object.

Whenever we have add an elements to the set, set compares the hash code number of the object to be added with the object already present in inside the set. If the hash codes are unique then it will inside the elements in to the set otherwise it will not insert

**Q) How to retrieve and remove an element to the set?**

**Answer.** Since set doesn’t store elements in any order and doesn’t have index, the retrieve and remove quotation should be done by using:

1. For-each loop
2. iterator

add e2 elements

en

e2

add e3 elements

e3

e1

**Set Container**

**Types of set:**

There are three types of set

1. **HashSet** -🡪store unique elements
2. **LinkedHashSet** -🡪store unique elements, but preserves insertion order
3. **TreeSet** --🡪stores unique elements but in stored order (ascending)

Collection (i)

Inherit

Set (i)

Implements Inherit

hashSet

StoredSet(i)

Inherit Inherit

NavigableSet(i)

LinkedHashSet

Implements

TreeSet

**Find hash code number:**

**package** library3;

/\*

\* hashCode() return hash code number of the object

\* return type int

\*/

**package** lib3;

**public** **class** Student {

**int** id;

String name;

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

**this**.id = id;

**this**.name = name;

}

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + "]";

}

}

**public** **class** HashCodeDemo {

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

Student st1=**new** Student(1123,"ramesh");

**int** hashValue1=st1.hashCode();

System.*out*.println("hashValue1: "+hashValue1);

Student st2=**new** Student(2234,"kamesh");

**int** hashValue2=st2.hashCode();

System.*out*.println("hashValue2: "+hashValue2);

}

}

**o/p:**

hashValue1: 27134973

hashValue2: 1284693

***hashCode method overrided in the student class and student id as hashCode number:***

**package** library3;

**public** **class** Student {

**int** id;

String name;

**int** age;

Student(**int** id,String name,**int** age)

{

**this**.id=id;

**this**.name=name;

**this**.age=age;

}

//Student id itself hash code number

**public** **int** hashCode(){

**return** id;

}

**public** String toString()

{

**return** "Student[id="+id+",name="+name+",age="+age+"]";

}

}

**public** **class** HashCodeDemo {

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

Student st1=**new** Student(1123,"ramesh");

**int** hashValue1=st1.hashCode();

System.*out*.println("hashValue1: "+hashValue1);

Student st2=**new** Student(2234,"kamesh");

**int** hashValue2=st2.hashCode();

System.*out*.println("hashValue2: "+hashValue2);

}

}

**o/p:**

hashValue1: 1123//student id value is hashcode number

hashValue2: 2234//student id value is hashcode number

**Note**

* In String class the hascode method is overrided to generate a hash code number based on the characters store in the string
* Equals method is also overrided in the string class to compare the string values based on hash code of the object

Ex,

**package** library3;

**public** **class** HashCodeDemo2 {

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

String str1="jspider";

**int** n1=str1.hashCode();

System.*out*.println(n1);

String str2=**new** String("jspider");

**int** n2=str2.hashCode();

System.*out*.println(n2);

System.*out*.println(str1.equals(str2));

}

}

o/p:

-1309843505

-1309843505

true

**1. HashSet:(** store unique elements**)**

Ex,

**package** library3;

**import** java.util.HashSet;

**public** **class** SetDemo1 {

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

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

HashSet set1=**new** HashSet();

set1.add("ramessh");

set1.add(**new** Student(324,"karim",25));

set1.add(**new** NoteBook(32,25.00));

set1.add(**null**);

**boolean** b1=set1.add("ramessh");

System.*out*.println(b1);

System.*out*.println("total elements: "+set1.size());

System.*out*.println("Set elements");

**for**(Object o1: set1){

System.*out*.println(o1);

}

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

false

total elements: 4

Set elements

null

NoteBook [page=32, price=25.0]

Student[id=324,name=karim,age=25]

ramessh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1.LinkedHashSet** -🡪store unique elements, but preserves insertion order

*In the above example if we create the set using LinkedHashSet it preserves the order of insertion:*

**package** library3;

**import** java.util.LinkedHashSet;

**public** **class** SetDemo1 {

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

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

LinkedHashSet set1=**new** LinkedHashSet();

set1.add("ramessh");

set1.add(**new** Student(324,"karim",25));

set1.add(**new** NoteBook(32,25.00));

set1.add(**null**);

**boolean** b1=set1.add("ramessh");

System.*out*.println(b1);

System.*out*.println("total elements: "+set1.size());

System.*out*.println("Set elements");

**for**(Object o1: set1){

System.*out*.println(o1);

}

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

false

total elements: 4

Set elements

ramessh

Student[id=324,name=karim,age=25]

NoteBook [page=32, price=25.0]

null

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2. TreeSet** --🡪stores unique elements but in stored order (ascending)**:**

**package** library3;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo {

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

TreeSet s1=**new** TreeSet();

s1.add("ramesh");

s1.add("kamesh");

s1.add("aamesh");

s1.add("lamesh");

s1.add("uamesh");

System.*out*.println("Total elements: "+s1.size());

System.*out*.println("Set elements");

**for**(Object o1: s1){

System.*out*.println(o1);

}

}

}

o/p:

Total elements: 5

Set elements

aamesh

kamesh

lamesh

ramesh

uamesh

**Note:** *if you add another object like pen object then it show exception;*

**package** library3;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo {

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

TreeSet s1=**new** TreeSet();

s1.add("ramesh");

s1.add("kamesh");

s1.add("aamesh");

s1.add("lamesh");

s1.add("uamesh");

s1.add(**new** Pen("blue",10));

System.*out*.println("Total elements: "+s1.size());

System.*out*.println("Set elements");

**for**(Object o1: s1){

System.*out*.println(o1);

}

}

}

o/p:

Exception in thread "main" java.lang.ClassCastException: library3.Pen cannot be cast to java.lang.Comparable

at java.util.TreeMap.put(Unknown Source)

**Creating a Table of Student class Properties With the help of HashSet:**

**package** library3;

**public** **class** Student {

**int** id;

String name;

**int** age;

Student(**int** id,String name,**int** age)

{

**this**.id=id;

**this**.name=name;

**this**.age=age;

}

//Student id itself hash code number

**public** **int** hashCode(){

**return** **this**.id;

}

**public** **boolean** equals(Object arg){

Student st1=(Student)arg;

**return** **this**.hashCode()==st1.hashCode();

}

**public** String toString()

{

**return** "Student[id="+id+",name="+name+",age="+age+"]";

}

}

**package** library3;

**import** java.util.HashSet;

**public** **class** SetDemo2 {

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

HashSet s1=**new** HashSet();

s1.add(**new** Student(123,"Ramesh", 21));

s1.add(**new** Student(234,"tamesh", 22));

s1.add(**new** Student(345,"yamesh", 23));

s1.add(**new** Student(567,"uamesh", 24));

s1.add(**new** Student(123,"Ramesh", 25));

System.*out*.println("Total Student: "+s1.size());

System.*out*.println("Student Details");

System.*out*.println("ID\tName\tAge");

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

**for**(Object o1: s1){

Student st1=(Student)o1;

System.*out*.println(st1.id+"\t"+st1.name+"\t"+st1.age);

}

}

}

o/p:

Total Student: 4

Student Details

ID Name Age

-----------------------------

567 uamesh 24

345 yamesh 23

234 tamesh 22

123 Ramesh 21

* **Sort in Object:**

Three condition for sorted,

* In java language if we have to sort a group of object then the class has must implements Comparable interface otherwise we can’t sort collection of object or array of object
* During implementation we have to decide on which state or properties of the object it should be sorted.

**Algorithm:**

1. arr[i]>arr[i++]🡪if true swap position

If false don’t swap

1. arr[i]-arr[i++]🡪if positive swap

If 0 or (-ve) then don’t swap

|  |
| --- |
| Comparable(I) |
| Method compareTo(): int |

public int CompareTo(Object arg)//method

Compare current object property with passed object property

1. If current object property is bigger than passed object property

***return +ve*;**

1. If current object property is exactly same as passed object property implements

***return 0*;**

1. If current object property is smaller than passed object property

|  |
| --- |
| Student(c) |
|  |

***return –ve;***

*Example of Sorting in ascending order:*

**package** lib3;

**public** **class** Student **implements** Comparable{

**int** id;

String name;

**int** age;

**public** Student(**int** id, String name, **int** age) {

**this**.id = id;

**this**.name = name;

**this**.age = age;

}

//Student id itself hashCode Number

**public** **int** hashCode(){

**return** id;

}

//Compare Student class object properties based on id;

**public** **int** compareTo(Object arg) {

Student st1=(Student)arg;

**return** (**this**.id-st1.id);

}

**public** **boolean** equals(Object arg){

Student st1=(Student)arg;

**return** **this**.hashCode()==st1.hashCode();

}

}

**package** lib3;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo2 {

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

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

TreeSet s1=**new** TreeSet();

s1.add(**new** Student(123,"Ramesh", 21));

s1.add(**new** Student(234,"Tamesh", 22));

s1.add(**new** Student(345,"Samesh", 23));

s1.add(**new** Student(567,"Umesh", 24));

s1.add(**new** Student(121,"Jagesh", 25));

System.*out*.println("Total Student: "+s1.size());

System.*out*.println("Student Details");

System.*out*.println("ID\tName\tAge");

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

**for**(Object o1: s1){

Student st1=(Student)o1;

System.*out*.println(st1.id+"\t"+st1.name+"\t"+st1.age);

}

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total Student: 5

Student Details

ID Name Age

-----------------------------

121 Jagesh 25

123 Ramesh 21

234 Tamesh 22

345 Samesh 23

567 Umesh 24

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Example of Sorting in descending order:*

**package** lib3;

**public** **class** Student **implements** Comparable{

**int** id;

String name;

**int** age;

**public** Student(**int** id, String name, **int** age) {

**this**.id = id;

**this**.name = name;

**this**.age = age;

}

//Student id itself hashCode Number

**public** **int** hashCode(){

**return** id;

}

//Compare Student class object properties based on id;

**public** **int** compareTo(Object arg) {

Student st1=(Student)arg;

**if**(**this**.id>st1.id){

**return** -1;

}**else** **if**(**this**.id<st1.id){

**return** 1;

}**else**{

**return** 0;

}

}

**public** **boolean** equals(Object arg){

Student st1=(Student)arg;

**return** **this**.hashCode()==st1.hashCode();

}

}

**package** lib3;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo2 {

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

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

TreeSet s1=**new** TreeSet();

s1.add(**new** Student(123,"Ramesh", 21));

s1.add(**new** Student(234,"Tamesh", 22));

s1.add(**new** Student(345,"Samesh", 23));

s1.add(**new** Student(567,"Umesh", 24));

s1.add(**new** Student(121,"Jagesh", 25));

System.*out*.println("Total Student: "+s1.size());

System.*out*.println("Student Details");

System.*out*.println("ID\tName\tAge");

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

**for**(Object o1: s1){

Student st1=(Student)o1;

System.*out*.println(st1.id+"\t"+st1.name+"\t"+st1.age);

}

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total Student: 5

Student Details

ID Name Age

-----------------------------

567 Umesh 24

345 Samesh 23

234 Tamesh 22

123 Ramesh 21

121 Jagesh 25

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Using Multiple Sorting Array:**

**package** lib3;

**import** java.util.Comparator;

**public** **class** SortByAge **implements** Comparator{

**public** **int** compare(Object arg1, Object arg2) {

Student st1=(Student)arg1;

Student st2=(Student)arg2;

**return** (st1.age-st2.age);

}

}

**package** lib3;

**import** java.util.Comparator;

**public** **class** SortByName **implements** Comparator{

**public** **int** compare(Object arg1, Object arg2) {

Student st1=(Student)arg1;

Student st2=(Student)arg2;

**return** (st1.name.compareTo(st2.name));

}

}

**package** lib3;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo2 {

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

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

SortByAge ageWise=**new** SortByAge();

SortByName nameWise=**new** SortByName();

TreeSet s1=**new** TreeSet(ageWise);//you can sort (nameWise) also. Empty () sort id ways

s1.add(**new** Student(123,"Ramesh", 21));

s1.add(**new** Student(234,"Tamesh", 22));

s1.add(**new** Student(345,"Samesh", 23));

s1.add(**new** Student(567,"Umesh", 24));

s1.add(**new** Student(121,"Jagesh", 25));

System.*out*.println("Total Student: "+s1.size());

System.*out*.println("Student Details");

System.*out*.println("ID\tName\tAge");

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

**for**(Object o1: s1){

Student st1=(Student)o1;

System.*out*.println(st1.id+"\t"+st1.name+"\t"+st1.age);

}

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total Student: 5

Student Details

ID Name Age

-----------------------------

123 Ramesh 21

234 Tamesh 22

345 Samesh 23

567 Umesh 24

121 Jagesh 25

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Note (Data structure of Set):

* TreeSet is implemted using hash table data structure
* linkedhashSet is build in high brid data structure it is developed more than one data structure

1. hashTabel 🡪uniqness
2. linklist🡪preserved order

* TreeSet is implemted with binary data structure

**3. Queue Type Collection:**

* Queue is a type of collection where elements are stored and processed in a FIFO order
* Queue doesn’t allow null value
* Duplicates are allowed
* Elements are stored non-linearly without index

There are two types of Queue

1. LinkedList –it implements both List and Queue
2. PriorityQueue –it implements only Queue interface where elements are always stored in priority order. The priority is decided based on sort in the elements.

Add any elements

But not null 1) retrieve🡪returning ref To head elements Of que

1. Add e1 elements 2) remove🡪remove ref to head el of
2. Add e2 elements que

e2

e1

1. Add e3 elements

…….

en

e3

n) Add en elements

***Example of LinkedList Queue*,**

**package** library3;

**import** java.util.LinkedList;

**public** **class** QueueDemo2 {

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

// **TODO** Auto-generated method stub

LinkedList l1=**new** LinkedList();

l1.add("jspider");

l1.add(**new** Student(123, "Ramesh", 21));

l1.add(**new** Student(234,"Suresh",22));

Object o1=l1.peek();//returns ref of head elemetns

System.*out*.println(o1);

System.*out*.println("Queue size: "+l1.size());

Object o2=l1.poll();//remove head element

System.*out*.println("Head element removed: "+o2);

System.*out*.println("Queue size: "+l1.size());

Object o3=l1.poll();//remove head element, returns null if queue is empty

System.*out*.println("Head element removed: "+o3);

System.*out*.println("Queue size: "+l1.size());

}

}

o/p:

jspider

Queue size: 3

Head element removed: jspider

Queue size: 2

Head element removed: Student [id=123, name=Ramesh, age=21]

Queue size: 1

***Example of PriorityQueue***,

**package** library3;

**import** java.util.PriorityQueue;

**public** **class** QueueDemo3 {

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

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

//SortByAge ageWise=new SortByAge();

//PriorityQueue s1=new PriorityQueue(ageWise);

PriorityQueue s1=**new** PriorityQueue();

s1.add(**new** Student(2486,"Azam",25));

s1.add(**new** Student(2468,"Azim",24));

s1.add(**new** Student(2846,"Amir",22));

s1.add(**new** Student(4286,"Aliv",23));

s1.add(**new** Student(4862,"Isaaq",25));

System.*out*.println("Queue size: "+s1.size());

System.*out*.println("Remove head elements: "+s1.poll());

System.*out*.println("Queue size: "+s1.size());

System.*out*.println("Remove head elements: "+s1.poll());

System.*out*.println("Queue size: "+s1.size());

System.*out*.println("Remove head elements: "+s1.poll());

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

}

}

o/p:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Queue size: 5

Remove head elements: Student[id=2468,name=Azim,age=24]

Queue size: 4

Remove head elements: Student[id=2486,name=Azam,age=25]

Queue size: 3

Remove head elements: Student[id=2846,name=Amir,age=22]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Iterator:**

* Iterator is used to iterate each element of any collection
* Iterator is an interface, which has three abstract method

1. next()🡪 returns next element from collection , if next element is not present , no such throws Exception
2. hasNext()🡪 returns true if next element exist, else returns false
3. remove ()🡪 removes the next element from collection.

In all collection classes, as helper method exists by iterator (), which returns instance of implementation class of iterator interface. Return type is iterator

e.g. HashSet s1=new HashSet();

Iterator itr1=s1.iterator();

itr1.next();

itr1.next();

itr1.next();

……

or

While(itr1.hashNext){

itr1.next();

}

**ListIterator:**

* ListIterator is used to iterator elements of only list type of collection
* ListIterator is an interface, which has 5 abstract methods

1. next()
2. hashNext()
3. remove()
4. Previouse()🡪 retruns previous elements of the of the list
5. hashPreviouse()🡪ret true if list has previous elemnts , else ret false
   * + - it be diredctional iterate

E.g. ArrayList l1=new ArrayList();

ListIterator itr1=l1.listIterator();

while(itr1.hashNext()){

itr1.next();

}

Or

while(itr1.hashPreviouse()){

itr1. Previouse ();

}

**package** library3;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.ListIterator;

**public** **class** ListDemo3 {

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

ArrayList l1=**new** ArrayList();

l1.add("jspider");

l1.add(**new** NoteBook(50,25.24));

l1.add(**null**);

l1.add(**new** Student(2314,"rames",21));

l1.add("jspider");

Iterator itr=l1.iterator();

**while**(itr.hasNext()){

Object o1=itr.next();

System.*out*.println(o1);

}

ListIterator itr2=l1.listIterator();

System.*out*.println("Next elements: "+itr2.next());

System.*out*.println("Next elements: "+itr2.next());

System.*out*.println("Next elements: "+itr2.next());

System.*out*.println("previous elements: "+itr2.previous());

System.*out*.println("previous elements: "+itr2.previous());

}

}

o/p:

jspider

NoteBook [page=50, price=25.24]

null

Student[id=2314,name=rames,age=21]

jspider

Next elements: jspider

Next elements: NoteBook [page=50, price=25.24]

Next elements: null

previous elements: null

previous elements: NoteBook [page=50, price=25.24]

**Wrapper Class:**

**package** lib3;

**public** **class** WrapperClassDemo1 {

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

**int** x1=34;

**double** y1=5.6;

//convert data type to Wrapper class type

//Boxing operation

Integer i1=**new** Integer(x1);

Double d1=**new** Double(y1);

System.*out*.println("i1 value: "+i1);

System.*out*.println("d1 value: "+d1);

//Wrapper class type to data type

//Un-boxing opration

**int** x2=i1.intValue();

**double** y2=d1.doubleValue();

System.*out*.println("x2 value: "+x2);

System.*out*.println("y2 value: "+y2);

}

}

o/p:

i1 value: 34

d1 value: 5.6

x2 value: 34

y2 value: 5.6

***Auto Boxing and Auto un-Boxing*:**

**package** lib3;

**public** **class** WrapperClassDemo1 {

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

**int** x1=34;

**double** y1=5.6;

//convert data type to Wrapper class type

//Auto Boxing operation

Integer i1=x1;

Double d1=y1;

System.*out*.println("i1 value: "+i1);

System.*out*.println("d1 value: "+d1);

//Wrapper class type to data type

//Auto Un-boxing opration

**int** x2=i1;

**double** y2=d1;

System.*out*.println("x2 value: "+x2);

System.*out*.println("y2 value: "+y2);

}

}

**o/p:**

i1 value: 34

d1 value: 5.6

x2 value: 34

y2 value: 5.6

**ex 2,**

**package** lib3;

**import** java.util.ArrayList;

**public** **class** WrapperClassDemo2 {

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

ArrayList l1=**new** ArrayList();

l1.add(2.5);//auto boxing and implicit up casting

l1.add(3.6);

Double d1=(Double)l1.get(0);//down casting

Double d2=(Double)l1.get(1);

**double** sum=d1+d2;//auto Un-boxing

System.*out*.println("Elements of Array List: "+d1 +", "+d2);

System.*out*.println("Sum is: "+sum);

}

}

**o/p:**

Elements of Array List: 2.5, 3.6

Sum is: 6.1

* Java library provides set of classes known as Wrapper classes which is used to convert primitive data types to an object type.
* All the wrapper classes are available “java.lang” package.
* All wrapper classes are final class, hence it can’t have subclass.
* All wrapper classes implements comparable interface, hence we can sort them.
* While storing a primitive type in the collection, the data type should be converted to object type using wrapper class
* Converting data type to wrapper class type is known as boxing operation. Boxing operation can be done either implicitly or explicitly. the implicit boxing is also known as auto boxing
* Converting a wrapper class type back to data type is known as un-box. From jdk 1.5 onwards can be done either implicitly or explicitly. The implicit un-boxing is also known as auto Un-boxing.

**Generic:** (add only that class properties)

**package** lib3;

**import** java.util.ArrayList;

**public** **class** GenericDemo1 {

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

//Un-safety

ArrayList<Employee> l1=**new** ArrayList<Employee>();

//Generic

l1.add(**new** Employee(123, "Ram"));

l1.add(**new** Employee(234, "Sam"));

l1.add(**new** Employee(345, "Ham"));

System.*out*.println("Employee List");

System.*out*.println("ID\tName");

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

**for**(Employee e1: l1){

System.*out*.println(e1.id+"\t"+e1.name);

}

}

}

**o/p:**

Employee List

ID Name

------------

123 Ram

234 Sam

345 Ham

**Exception:**

E.g. classCastException, NoSuch Exception

**package** lib3;

**public** **class** Demo1 {

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

**int** i=10;

**int** j=0;

**try**{

j=i/0;

}

**catch**(ArithmeticException exp){

System.*out*.println("Unable to divided by 0");

}

System.*out*.println("i value: "+i);

System.*out*.println("j value: "+j);

}

}

o/p:

Unable to divide by 0

i value: 10

j value: 0

**Single try and multiple catch block:**

**package** lib3;

**public** **class** Demo1 {

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

**int** i=10;

**int** j=0;

**int**[] arr1=**new** **int**[5];//index range 0 to 4

**try**{

j=i/5;

arr1[5]=j;//store j value in index 5

System.*out*.println("exit try block");

}

**catch**(ArithmeticException exp){

System.*out*.println("Unable to divided by 0");

}

**catch**(ArrayIndexOutOfBoundsException exp){

System.*out*.println("array is out of index");

}

System.*out*.println("i value: "+i);

System.*out*.println("j value: "+j);

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

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

System.*out*.println(arr1[k]);

}

}

}

**o/p:**

array is out of index

i value: 10

j value: 2

array elements are

0

0

0

0

0

**Nested try catch block:**

**package** lib3;

**public** **class** Demo3 {

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

**int** i=10;

**int** j=0;

**int**[] arr1=**new** **int**[5];//index range 0 to 4

**try**{

System.*out*.println("running outer try block");

**try**{

System.*out*.println("running inner try block");

j=i/0;

System.*out*.println("exiting inner try block");

}

**catch**(ArithmeticException exp){

System.*out*.println("Unable to divided by 0");

}

arr1[5]=j;//store j value in index 4

System.*out*.println("exiting outer try block");

}

**catch**(ArrayIndexOutOfBoundsException exp){

System.*out*.println("index out of range");

}

System.*out*.println("i value: "+i);

System.*out*.println("j value: "+j);

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

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

System.*out*.println(arr1[k]);

}

}

}

**o/p:**

running outer try block

running inner try block

Unable to divided by 0

index out of range

i value: 10

j value: 0

array elements are

0

0

0

0

0

* An exception is occurred in inner most and can propagate to the outer catch block, but vise verse is not valid

**Finally block:**

**package** lib3;

**public** **class** Demo4 {

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

**int** i=10;

**int** j=0;

**try**{

System.*out*.println("running inner try block");

j=i/0;

System.*out*.println("exiting inner try block");

}

**catch**(ArithmeticException exp){

System.*out*.println("Unable to divided by 0");

//System.exit(0); //force termination

}

**finally**{

System.*out*.println("I run always...");

}

System.*out*.println("i value: "+i);

System.*out*.println("j value: "+j);

}

}

**o/p:**

running inner try block

Unable to divided by 0

I run always...

i value: 10

j value: 0

**Throwable:(**define properties**) / hierarchy of Exception**

Keyword “through” 🡪throwable instance;

1. **Error**
2. StackOverFlowError
3. NoSuchMethodDefFoundError
4. ….
5. **Exception**

*Unchecked Exception or Un Caught Exception*

1. RuntimeException
2. ArthmeticException
3. classCastException
4. NumberFormateException
5. ………

*Checked Exception or Cought Exception*

1. IOException
2. SQLException
3. ClassNotFoundException
4. …….

**Unchecked Exception auto propagate** **checked Exception explicit propagate**

|  |
| --- |
| M2()  Checked excep  no handler |
| M1() |
| Main() |

|  |
| --- |
| M2()  Un Checked excep  No handaler |
| M1() |
| Main() |

If no handler

Then use keyword “throws”

For explicit propagate

***Ex, of unchecked Exception:***

**package** lib3;

**public** **class** Calculator {

**void** divide(**int** n1,**int** n2){

System.*out*.println("deviding n1 by n2");

**int** res=n1/n2;

System.*out*.println("result is: "+res);

}

}

**package** lib3;

**public** **class** Demo5 {

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

Calculator c1=**new** Calculator();

**try**{

c1.divide(26,0);

}

**catch**(ArithmeticException exp){

System.*out*.println("unable to devided by 0");

}

}

}

o/p:

deviding n1 by n2

unable to devided by 0

**Generic Exception handler:**(it catches whatever Exception occurred)

**try**{

j=i/0;

arr1[5]=j;//store j value in index 4

System.*out*.println("exit try block");

}

**catch**(ArithmeticException exp){

System.*out*.println("Unable to divided by 0");

}

**catch**(ArrayIndexOutOfBoundsException exp){

System.*out*.println("array is out of index");

}

**catch**(Exception exp){

System.*out*.println("Generic Exception");

**printStackTrace():***its print what the Exception occured*

**package** lib3;

**public** **class** Demo1 {

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

**int** i=10;

**int** j=0;

**int**[] arr1=**new** **int**[5];//index range 0 to 4

**try**{

j=i/0;

arr1[5]=j;//store j value in index 5

System.*out*.println("exit try block");

}

**catch**(Exception exp){

System.*out*.println("Generic Exception");

exp.printStackTrace();//details of exception , where when and reason

//stack information

}

System.*out*.println("i value: "+i);

System.*out*.println("j value: "+j);

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

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

System.*out*.println(arr1[k]);

}

}

}

o/p:

Generic Exception

java.lang.ArithmeticException: / by zero

at lib3.Demo1.main(Demo1.java:10)

i value: 10

j value: 0

array elements are

0

0

0

0

0

* We can define our won class as a type of exception by extending either exception class or runtime exception class. If users define class is extending directly exception class then it’s come under checked exception category, if it is extending from runtime exception class then it’s come under unchecked exception category.

**package** lib3;

**public** **class** Sequence1 {

**void** printNumbers() **throws** InterruptedException{

**for**(**int** i=1; i<=5; i++){

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

Thread.*sleep*(1000);

}

}

}

**package** lib3;

**public** **class** Demo7 {

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

Sequence1 seq1=**new** Sequence1();

**try** {

seq1.printNumbers();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

o/p:

i=1

i=2

i=3

i=4

i=5

**Thread :( Multi-task)**

Stack

thread

**Heap**

Stack

thread

Static pool

Stack

thread

Method

object

***Multiple task thread performing same class:***

**package** threads;

**public** **class** Sequence1 {

**void** printNumbers() **throws** InterruptedException{

Thread curTh=Thread.*currentThread*();

String thName=curTh.getName();

**for**(**int** i=1; i<=5; i++){

System.*out*.println(thName+": i="+i);

Thread.*sleep*(1000);

}

}

}

**package** threads;

**public** **class** NumberThread1 **extends** Thread{

**public** **void** run(){

//define the task

Sequence1 seq1=**new** Sequence1();

**try**{

seq1.printNumbers();

}**catch**(InterruptedException e){

e.printStackTrace();

}

}

}

**package** threads;

**public** **class** Demo1 {

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

NumberThread1 nt1=**new** NumberThread1();

NumberThread1 nt2=**new** NumberThread1();

//set name to thread object

nt1.setName("NT1-Thread");

nt2.setName("NT2-Thread");

nt1.start();

nt2.start();

}

}

o/p:

NT2-Thread: i=1

NT1-Thread: i=1

NT1-Thread: i=2

NT2-Thread: i=2

NT1-Thread: i=3

NT2-Thread: i=3

NT1-Thread: i=4

NT2-Thread: i=4

NT1-Thread: i=5

NT2-Thread: i=5

***Multiple threads performing different class:***

**package** threads;

**public** **class** Sequence2 {

**void** printNumbers() **throws** InterruptedException{

Thread curTh=Thread.*currentThread*();

String thName=curTh.getName();

**for**(**int** k=10; k<=15; k++){

System.*out*.println(thName+": k="+k);

Thread.*sleep*(1000);

}

}

}

**package** threads;

**public** **class** NumberThread2 **extends** Thread{

**public** **void** run(){

//define the task

Sequence2 seq1=**new** Sequence2();

**try**{

seq1.printNumbers();

}**catch**(InterruptedException e){

e.printStackTrace();

}

}

}

**package** threads;

**public** **class** Demo1 {

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

NumberThread1 nt1=**new** NumberThread1();

NumberThread2 nt2=**new** NumberThread2();

//set name to thread object

nt1.setName("NT1-Thread");

nt2.setName("NT2-Thread");

nt1.start();

nt2.start();

}

}

o/p:

NT2-Thread: k=10

NT1-Thread: i=1

NT1-Thread: i=2

NT2-Thread: k=11

NT2-Thread: k=12

NT1-Thread: i=3

NT2-Thread: k=13

NT1-Thread: i=4

NT1-Thread: i=5

NT2-Thread: k=14

NT2-Thread: k=15

1. ***Thread scheduler:***

* To allocate resource to user defined thread
* Based on thread priority

Priority 1 to 10 set priority (8)

Default is 5 priorities

1. ***Garbage collector thread:***

To clean memory

Low priority thread

1. ***Thread main:***

* To begin execution by calling main method of class(static statements)
* Passing command line argument as String Array

**package** threads;

**public** **class** NumberThread3 **implements** Runnable{

**public** **void** run(){

**for**(**int** i=1; i<=5; i++){

System.*out*.println(": i="+i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**package** threads;

**public** **class** Demo2 {

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

// **TODO** Auto-generated method stub

NumberThread3 nt1=**new** NumberThread3();

Thread th1=**new** Thread(nt1);

//creating thread object which has Runnable type object

th1.start();//begin thread execution , call run() of Runnable type

}

}

**Thread Safe:**

Online Banking

Account

Th1

5000

Mobile Banking

Th2

5000

ATM 5000

Th3

Achieving thread safety by

1. **Immutable**
2. **Thread Synchronization**

**Ex1,**

**package** threads;

**public** **class** CommonResource {

**synchronized** **void** printNumber1(){

Thread curTh=Thread.*currentThread*();

//returns reference to current running Thread

String thName=curTh.getName();

System.*out*.println("Print Number 1 to 25 ");

**for** (**int** i=1; i<=25; i++){

System.*out*.println(thName+": i="+i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** threads;

**public** **class** NumberThread1 **extends** Thread{

CommonResource cr;

**public** NumberThread1(CommonResource cr){

**this**.cr=cr;

}

**public** **void** run(){

cr.printNumber1();

}

}

**package** threads;

**public** **class** StartThreads {

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

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

CommonResource cr1=**new** CommonResource();//sharing cr1 to 3 thread

NumberThread1 th1=**new** NumberThread1(cr1);

NumberThread1 th2=**new** NumberThread1(cr1);

NumberThread1 th3=**new** NumberThread1(cr1);

th1.setName("Th1-Thread");

th2.setName("Th2-Thread");

th3.setName("Th3-Thread");

th1.start();

th2.start();

th3.start();

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

}

}

**o/p:** output will be first Th3 running and print 1 to 25 with 1 second delay then second Th2 running and print 1 to 25 with 1 second delay then Th1 running and print 1 to 25 with 1 second delay.

It’s mean CommonResource cr1 object sharing to 3 threads.

**Ex2,**

**package** threads;

**public** **class** CommonResource {

**synchronized** **public** **void** printNumber1() **throws** InterruptedException{

Thread curTh=Thread.*currentThread*();

//returns reference to current running Thread

String thName=curTh.getName();

System.*out*.println(thName+" entering to wait state");

wait();//leaves the lock temporarily, enter to wait state

System.*out*.println(thName+" Resuming back to work..... ");

}

**synchronized** **public** **void** printNumber2() **throws** InterruptedException{

Thread curTh=Thread.*currentThread*();

//returns reference to current running Thread

String thName=curTh.getName();

System.*out*.println(thName+" Notifying other thread s");

notifyAll();//Notify the thread which is in wait state to resuming back

System.*out*.println(thName+" Resuming back to work..... ");

}

}

**package** threads;

**public** **class** NumberThread1 **extends** Thread{

CommonResource cr;

**public** NumberThread1(CommonResource cr){

**this**.cr=cr;

}

**public** **void** run(){

**try** {

cr.printNumber1();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**package** threads;

**public** **class** NumberThread2 **extends** Thread{

CommonResource cr;

**public** NumberThread2(CommonResource cr){

**this**.cr=cr;

}

**public** **void** run(){

**try** {

cr.printNumber2();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**package** threads;

**public** **class** StartThreads {

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

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

CommonResource cr1=**new** CommonResource();//sharing cr1 to 3 thread

NumberThread1 th1=**new** NumberThread1(cr1);

NumberThread1 th2=**new** NumberThread1(cr1);

NumberThread1 th3=**new** NumberThread1(cr1);

NumberThread2 th4=**new** NumberThread2(cr1);

th1.setName("Th1-Thread");

th2.setName("Th2-Thread");

th3.setName("Th3-Thread");

th4.setName("Th4-Thread");

th1.start();

th2.start();

th3.start();

**try** {

Thread.*sleep*(5000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

th4.start();

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

}

}

**o/p:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Th1-Thread entering to wait state

Th3-Thread entering to wait state

Th2-Thread entering to wait state

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Th4-Thread Notifying other thread s

Th4-Thread Resuming back to work.....

Th2-Thread Resuming back to work.....

Th3-Thread Resuming back to work.....

Th1-Thread Resuming back to work.....

**MAP :( PART of collection)**

* Pair of object
* Key-value pair of object
* Key can be any object, should be unique
* Value can be any type of object
* Value is mapped by key
* Value is retrieved based on its key

Based on key

key🡪value pair Kn key points to Vn value

Cn Kn🡪Vn

C1 elements

C2 elements

C2 K2🡪V2

Cn elements

C1 K1🡪V1

Run time example = search engineering

**Map (I)**

**Sorted Map**

**Hash Table**

**Hash Map**

**Navigable Map**

**Tree Map**

**package** lib3;

**import** java.util.HashMap;

**public** **class** MapDemo1 {

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

HashMap m1=**new** HashMap();

m1.put(25, **new** Student(12345,"Ashok",21));

m1.put(**null**, **true**);

System.*out*.println("map size: "+m1.size());

Student s1=(Student)m1.get(25);

System.*out*.println("s id: "+s1.id);

System.*out*.println("s name: "+s1.name);

System.*out*.println("s age: "+s1.age);

m1.remove(25);

System.*out*.println("map size: "+m1.size());

}

}

o/p:

map size: 2

s id: 12345

s name: Ashok

s age: 21

map size: 1

**Difference between three Maps:**

|  |  |  |
| --- | --- | --- |
| Hash Map | Hash Table | Tree Map |
| 1. Null can be key 2. Method are not synchronized 3. Not a thread safe 4. Unsorted map 5. Performance high | 1. Null can’t be key 2. Method are synchronized 3. Thread safe class 4. Unsorted map 5. Performance low | 1. Null can’t be key 2. Method are not synchronized 3. Not a thread safe 4. Sorted map 5. Performance high |

**Thread and its Life Cycle:**

After lock is released wait for lock to be released

start() resource completion

allocated of task stop()

time elapsed sleep(t)

time elapsed wait()

on receiving notification

**Difference between sleep and wait methods:**

|  |  |
| --- | --- |
| Sleep () | Wait () |
| 1. Sleep is static method of thread class 2. Whenever a thread execute a sleep function it carries the object lock and sleeps for specify time 3. Sleep can’t be overcome thread dead lock 4. Sleep can be used any context | 1. Wait is a non-static method of object class 2. Whenever a thread execute a thread it doesn’t carry an object lock, its leaves the lock and goes to the wait statements 3. Wait can be used thread dead lock 4. Wait should be used in synchronized method context |

**Garbage collection process:**

Garbage collector:

* To collect un-used object from heap
* Low priority thread
* Finalize()

1. Garbage collection is process of removing an object from the heap , it is built on several algorithms
2. Garbage collector is a thread which is responsible to collect un-used object from the heap
3. Whenever g collector is running is destroys the un-used object by calling finalize() method
4. The object lying on the heap without any reference variable is known as de-reference object, this objects are eligible for garbage collection. A program can call garbage collector by using a statements “System.gc”
5. It is good practice for a developer to assign a null value to the reference variables after using the object,, so that the object becomes eligible for garbage collection

**Java Bean class:**

* Develop of a java class with private data members, public constructor ,complete getters and setters methods known as java Bean Class,

By getter method is used to provide to re access

Where setter method is used to write access to the private data members

* The java bean class is used in developing DAO(Data Access Object) layer

DTO (Data Transaction Object)

* It is very good example of encapsulation

Ex,

**package** lib3;

**public** **class** Employee1 {

**private** **int** id;

**private** String name;

**private** **double** salary;

**public** Employee1(**int** id, String name, **double** salary) {

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

}

**package** lib3;

**public** **class** MainClass1 {

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

Employee1 e1=**new** Employee1(1234, "Asok", 25000.00);

System.*out*.println("employee salary: "+e1.getSalary());

e1.setSalary(50000.00);

System.*out*.println("employee salary: "+e1.getSalary());

}

}

o/p:

employee salary: 25000.0

employee salary: 50000.0

**File Handling:**

1. Delete
2. Move
3. Copy
4. Write
5. Read

**File**

Program

* Files arranged in an operating system are called as **File System**.
* The programs returns to handle File System are called as **File Handling**.
* The action performs on a file are

1. Create
2. Delete
3. Write
4. Read
5. Copy

Etc….

To do these actions we make use of IO package

IO package contents the class which confined:

1. File
2. File write
3. File read
4. Output stream
5. Input stream
6. **File:**

File class is use for performing the creation and deletion action. In contains argument constructor the type of arguments is String type

File f=new File (“D:\\Azam\\jecm3-1\\file”);

makedir();🡪 it is used for creation of directory , the return type of this method is Boolean. The directory is successfully created its returns true. It returns false in two conditions

1. The file is not successfully created its returns false 2. With path specified is wrong its returns false

exists();🡪 which is used in checking the files if present or not . Returns type is Boolean.

delete();🡪 it is used in deleting the files. Returns type is Boolean.

CreateNewFile();🡪it is used for creation of new file. The return type of this function is Boolean. In throws and exception known as IO exception

File f=new File (“D:\\Azam\\jecm3-1\\file\\myfile.text”);

length();🡪it is used a finding the length of the file. Returns type of this method is long.

***Example 1 of creating a file:***

**package** fileHandling;

**import** java.io.File;

**public** **class** FileImplementation {

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

File f1=**new** File("D:\\Java workspace\\Pattern\\file");

**if**(f1.exists()){

System.*out*.println("alrady exixsts");

}**else**{

f1.mkdir();

}

}

}

o/p: when you run that program it will create a “file” folder in that directory where you mentioned.

***Examples 2 under the file create new Myfile:***

**package** fileHandling;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** FileImplementation {

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

File f1=**new** File("D:\\Java workspace\\Pattern\\file\\Myfile");

**if**(f1.exists()){

System.*out*.println("alrady exixsts");

}**else**{

**try** {

f1.createNewFile();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

**O/p:** when run the program it will create “Myfile.text” file in the “file” directory, which size is 0 kb. And it’s an empty text file.

***Example 3 where your file exists:***

**package** fileHandling;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** FileImplementation {

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

File f1=**new** File("D:\\Java workspace\\Pattern\\file\\Myfile");

**if**(f1.exists()){

System.*out*.println("alrady exixsts");

System.*out*.println(f1.exists());

System.*out*.println(f1.getAbsolutePath());

}**else**{

**try** {

f1.createNewFile();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

o/p:

alrady exixsts

true

D:\Java workspace\Pattern\file\Myfile

1. **File writer:**

File writer class is use to write only characters to the file. It consists of overloaded constructor

write(String);🡪this methods write input to the output stream. To move to the next line into use “\r\n”;

* write() methods writes String continuously
* write() method can write only one character at a time to the output stream

flash();🡪 flash is a contains from output stream to the file

close();🡪 is stream .it flashes the data first in the output stream and then closes the stream.

***Example 4 writing some text in the Myfile:***

**package** fileHandling;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** FileWriterImple {

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

FileWriter fr=**null**;

**try**{

fr=**new** FileWriter("D:\\Java workspace\\Pattern\\file\\Myfile");

fr.write("hello");

fr.write(" last class of java");

}**catch**(IOException e){

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

}

**finally**{

**try**{

fr.flush();

fr.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

**o/p:** showing null but you can see on your Myfile.text

hello last class of java

1. **File reader:**

* file reader reads characters from given file
* it has overloaded constructor
* opens input stream

read(char[]);🡪reads all the character in sequence which fits the specified array size

Drawback is you can read only one char at once

close();🡪closes the data from input in stream and closes the stream.

***Example 5 of reading the file:***

**package** fileHandling;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** FileWriterImple {

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

File f1=**new** File("D:\\Java workspace\\Pattern\\file\\Myfile");

FileReader fr=**null**;

**try**{

fr=**new** FileReader("D:\\Java workspace\\Pattern\\file\\Myfile");

**long** l1=f1.length();

**char**[] ch=**new** **char**[(**int**)l1];

fr.read(ch);

System.*out*.println(ch);

}**catch**(IOException e){

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

}

**finally**{

**try**{

fr.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

o/p:

hello last class of java

**BufferedReader:**

***Example 6 of reading the files with BufferedReader. Whatever you write in the Myfile.text, the output will show you everything.***

**package** fileHandling;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** BufferedReaderDemo {

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

FileReader fr=**null**;

BufferedReader bf=**null**;

**try**{

fr=**new** FileReader("D:\\Java workspace\\Pattern\\file\\Myfile");

bf=**new** BufferedReader(fr);

String s1=bf.readLine();

**while**(s1!=**null**){

System.*out*.println(s1);

s1=bf.readLine();

}

}**catch**(IOException e){

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

}

**finally**{

**try**{

fr.close();

}**catch**(IOException e){

e.printStackTrace();

}

}

}

}

**o/p:** after edit Myfile.text with some text, the program will return all the text like

hello last class of java

so be carefull

**Serializable**

**Inner class**

**Runnable**

Runnable(i)

run();void

**implements**

Sample1

**sample1** is runnable type only run not create new stack

Thread

sample1 s1=new sample1();

Thread th1=new Thread(s1);

th1.start();🡪allocate new stack

🡪call run() method of s1

extends

Thread 🡪runnabel

Sequence

🡪create new stack

Sequence seq=new Sequence ();

Seq.start();🡪calling of seq object

**package** misc;

**public** **class** Sequence1 **implements** Runnable {

**public** **void** run(){

Thread curTh=Thread.*currentThread*();

//returns ref of current thread

String thName=curTh.getName();

**for**(**int** i=1; i<=5; i++){

System.*out*.println(thName+": i="+i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**package** misc;

**public** **class** Demo4 {

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

Sequence1 seq1=**new** Sequence1();

//runnable type

Thread th1=**new** Thread(seq1);

th1.setName("my thread");

th1.start();

}

}

o/p:

my thread: i=1

my thread: i=2

my thread: i=3

my thread: i=4

my thread: i=5

**package** misc;

**import** java.io.Serializable;

**public** **class** Employee **implements** Serializable {

**int** id;

String name;

**transient** **double** salary;

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

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId(){

**return** id;

}

**public** **void** setId(**int** id){

**this**.id=id;

}

**public** String geName(){

**return** name;

}

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

**this**.name=name;

}

**public** **double** getSalary(){

**return** salary;

}

**public** **void** setSalary(**double** salary){

**this**.salary=salary;

}

}

**package** misc;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.ObjectOutputStream;

//serialization code

**public** **class** Demo5 {

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

FileOutputStream fout=**new** FileOutputStream("D:\\store\\emp.ser");

ObjectOutputStream objOut=**new** ObjectOutputStream(fout);

Employee e1= **new** Employee(1234, "Ram", 25000.00);

objOut.writeObject(e1);

objOut.close();

fout.close();

}

}

**package** misc;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FilterInputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

//deserializable code

**public** **class** Demo6 {

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

// **TODO** Auto-generated method stub

FileInputStream fIn=**new** FileInputStream("D:\\store\\emp.ser");

ObjectInputStream objIn=**new** ObjectInputStream(fIn);

Employee e1=(Employee)objIn.readObject();

System.*out*.println("id: "+ e1.getId());

System.*out*.println("name: "+ e1.geName());

System.*out*.println("salary: "+ e1.getSalary());

fIn.close();

objIn.close();

}

}

o/p:

id: 1234

name: Ram

salary: 0.0

* writing a java object in a file is known as sereilization
* the serialization can be done by using FileIputStream and ObjectInputStream
* The jvm can do serialization only those object which is of type serializable, and class must implements serialize marker interface
* The object serialization which to achieved persistence of java object
* Reading java object from the serialize point is known as de-serialization, the de-serialization can be done by using FileInputStream and ObjectInputStream

**Inner class:**

**package** misc;

**public** **class** Demo1 {

**static** **class** Sample1{

**static** **int** *var1*=34;

**int** var2=45;

**void** test() {

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

}

**static** **void** disp(){

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

}

}

}

**package** misc;

**public** **class** MainClass1 {

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

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

Demo1.Sample1 ref1=**new** Demo1.Sample1();

System.*out*.println("var1 value: "+Demo1.Sample1.*var1*);

ref1.test();

System.*out*.println("var2 value: "+ref1.var2);

Demo1.Sample1.*disp*();

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var1 value: 34

non-static

var2 value: 45

static

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Non-static inner class:

**package** misc;

**public** **class** Demo2 {

**class** Sample1{

**int** var2=45;

**void** test() {

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

}

}

}

**package** misc;

**public** **class** MainClass2 {

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

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

Demo2 obj=**new** Demo2();

Demo2.Sample1 ref1=obj.**new** Sample1();

ref1.test();

System.*out*.println("var2 value: "+ref1.var2);

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

non-static

var2 value: 45

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In the method:

**package** misc;

**public** **class** Demo3 {

**void** test(){

**class** Sample2{//local inner class

**int** k=12;

**void** disp(){

System.*out*.println("running disp()");

}

}

Sample2 s2=**new** Sample2();

System.*out*.println("k value: "+s2.k);

s2.disp();

}

}

**package** misc;

**public** **class** MainClass3 {

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

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

Demo3 d3=**new** Demo3();

d3.test();

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

k value: 12

running disp()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Inner annymos class:

**package** misc;

**public** **interface** Demo7 {

**void** test();

}

**package** misc;

**public** **class** MainClass7 {

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

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

Demo7 obj1=**new** Demo7(){

**public** **void** test(){

System.*out*.println(" implents of test() with annonymouse class");

}

};

obj1.test();

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

implents of test() with annonymouse class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*