**Note: -**Inside the Array Reference variable we can pass two types of elements

**1]Address of the Array Object** [Object Creation is Mandatory].

**2]Null Keyword**

**Null Keyword: -**

-Null is a Keyword in java.

-Null Means Nothing/Empty.

-The Default value of null is Null itself.

-If we pass Null into an Array References variable it does not allow to programmer to point to any of the Array Object.

**NullPointerException: -**

-If we pass Null into an Array Reference Variable and If we try to Define the Length of the Array Object since there is no Array Object created, we get a Runtime Exception called as **NullPointerException**.

Ex: -

Class P2

{

Public static void main(String args[])

{

Int[] arr;

arr=null;

System.out.println(arr);**//null**

}

}

Ex: -

Class P3

{

Public static void main(String args[])

{

int[] a;

a=new int[2];

System.out.println(a);

System.out.println(a.length);

int[] b;

b=null;

System.out.println(b);

System.out.println(b.length)//**Exception**

}

}

QNo1.Write a program to count the number of Even Numbers present inside the Array. int[] n={2,5,10,6,7,12,8,9}

**Ans: -**

class CountEven

{

public static void main(String[] args)

{

int[] arr={2,5,10,6,7,12,8,9};

System.out.println("Address"+arr);

int count=0;

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

{

if (arr[i]%2==0)

{

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

count++;

}

}

System.out.println("The Number of Even Number In Array:-"+count);

}

}

QNo2.Write a program to count the number of Positive numbers inside the array. int[] n={2,-5,10,-6,7,-12,8,-9}

**Ans: -**

class CountPos

{

public static void main(String[] args)

{

int[] arr={2,-5,10,-6,7,-12,8,-9};

int count=0;

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

{

if (arr[i]>0)

{

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

count++;

}

}

System.out.println("The Number of Positive Number is:-"+" "+count);

}

}

QNo3.Write a program to find he summation of Even numbers present inside the array.

**Ans: -**

class SummationEven

{

public static void main(String[] args)

{

int[] arr={2,23,34,141,34,54,234};

int sum=0;

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

{

if (arr[i]%2==0)

{

sum+=arr[i];

}

}

System.out.println(sum);

}

}

QNo4.Write a program to count the number of Uppercase and Lowercase Characters present inside the array.Int[] n={‘A’,’a’,’B’,’b’,’C’,’c’}

**Ans: -**

class CountLetter

{

public static void main(String[] args)

{

char[] arr={'A','a','B','b','C','c','f'};

int Ucount=0;

int Lcount=0;

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

{

if (arr[i]>='A' && arr[i]<='Z')

{

Ucount++;

}

}

System.out.println("The Uppercase Letter Is:-"+Ucount);

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

{

if (arr[i]>='a' && arr[i]<='z')

{

Lcount++;

}

}

System.out.println("The Lowercase Letter Is:-"+Lcount);

}

}

QNo5.Write a program to count the number of Vowels present inside the array

**Ans: -** class CountVowels

{

public static void main(String[] args)

{

char[] ch={'A','a','f','e','h','i'};

int Count=0;

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

{

if (ch[i]=='a'||ch[i]=='e'||ch[i]=='i'||ch[i]=='o'||ch[i]=='u')

{

Count++;

}

}

System.out.println("The Number of Vowels Is:-"+Count);

}

}

QNo6.Write a program to count the number of Uppercase Vowels present inside the array.

**Ans: -**

class CountUppercaseVowels

{

public static void main(String[] args)

{

char[] ch={'A','B','c','E','e','I','O','U'};

int Count=0;

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

{

if (ch[i]=='A'||ch[i]=='E'||ch[i]=='I'||ch[i]=='O'||ch[i]=='U')

{

Count++;

}

}

System.out.println("The Uppercase Vowels Are:-"+Count);

}

}

**Passing An Array To Method**

-We need to Create the Array Object Reference Variable.

-We Need to call the Method and Pass Address of the Array Object.

#Program

Class P1

{

Static public int demo(int n[])

{

int sum=0;

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

{

Sum=sum+n[a];

}

return sum;

}

Public static void main(String args[])

{

int[] n={1,21,3,5,6};

int sum=demo(n);

System.out.println(sum);

int[] m={19,18,20,21,22};

System.out.println(demo(m));

}

}

**Global Variable: -**

-Global variable are declared inside the Class Block.

-Global variable have default value.

-Global variable are classified into two types: -

**1]Static Global Variable**

**2]Non-Static Global Variable**

**1]Static Global Variable**

-A variable which declared inside the class block and Prefixed with the **Static** Keyword is called Static Variable.

-We can declare more than one Static variable inside the Class Block but variable name has to be different.

**Syntax: -**

**Static Datatype Variable Name=Value;**

**Ex**: -

Static int n=10;

-Static variable consist of **Default** **Value**.

We can Access the Static variable in two ways: -

**1]Directly By [Its Variable Name].**

**2]Class Name as Reference**

#class p1

{

Static int n=10;

Public static void main (String[] args)

{

System.out.println(n);

System.out.println(P1.n);

}

}

**Note: -**We can Access a static variable in two ways for printing the value

Present inside the static variable for Initializing the static variable.

#class p2

{

Static int n;

Public static void main (String[] args)

{

System.out.println(n);

System.out.println(P1.n);

N=10;

System.out.println(n);

P1.n=20

System.out.println(n);

}

}

**Note: -**If a class consist of both Static variable and Local Variable with Same Variable name after declaration of Local variable if we try to access any variable Directly by its Variable name the compiler always gives Priority for Local Variable first.

-On such case to access a static variable we need to take the help of **Class Name as Reference.**

#class P3

{

Static int n=10;

Public static void main(String[] args)

{

System.out.println(n); //Directly by Its Name

n=20; //Directly by Its Name

System.out.println(n); //Directly by Its Name

int n=30;

System.out.println(n); //Directly by Its Name

System.out.println(n); //Directly by Its Name

n=60;

System.out.println(n); //Directly by Its Name

System.out.println(P3.n); //Class Name as Reference

P3.n=100; //Class Name As Reference

System.out.println(P3.n); //Class Name as Reference

}

}

**O/P: -**

**🡺10**

**🡺20**

**🡺30**

**🡺30**

**🡺60**

**🡺20**

**🡺100**

**Note: -** We can access the Static variable from one class to another class with help of **Class Name As Reference**.

#class P4

{

Static int n=10;

}

#class P5

{

Static int n=20;

Public static void main(String args[])

{

System.out.println(n); //Directly by Its Name

System.out.println(P5.n); //Class Name as Reference

System.out.println(P4.n) ;//Class Name as Reference

}

}

**Note: -**After declaration of static variable when we declare the local variable inside the main method and if we try to access directly then the compiler searches for the local variable because the local variable has the first priority.

-The compiler searches the local variable Before that Statement.

Class P1

{

Static int n=10;

}

Class p2

{

Static int n=20;

Public static void main(String args[])

{

System.out.println(n);

System.out.println(P2.n);

System.out.println(P1.n);

P1.n=50;

System.out.println(P1.n);

}

}

**O/p:-**

**🡺20**

**🡺20**

**🡺10**

**🡺50**

Class P3

{

Static int a=10;

Static int b=20;

Public static void main(String args[])

{

System.out.println(a);

System.out.println(b);

}

}

**Address Of Next**

**Variable**

**a**

**~~0~~ 10**

**LINK**

**Main**

**1.Directly Name 🡺Local Var🡺Present🡺Access**

**2.Not Present🡺Static Var**

**{**

**Sop(a);**

**Sop(b);**

**}**

**~~0~~ 20**

**b**

gg

**1.Directly Name 🡺Local Var🡺Present🡺Access**

**2.Not Present🡺Static Var**

**Static Area**

**Stack Area**

**Loading Process of Static Variable: -**

**1]Searching**

**2]Loading**

**3]Allocates with Default Value**

**4]Passing Actual Value of Variable**

**Steps: -**

1.First the main method comes for execution

2.Before execution of main method we need to keep Loads the static variable for the execution

3.Before execution of main method compiler **Searches** for the Static variable inside the class Block. **[Searching Is Happens]**

4.If its present then it gets loaded **one by one** inside the **Static Area**. **[Loading Is Happens]**. Again, compiler searches for another static variable whether it is present it gets loaded or Not Present.

5.Then compile allocates/assigns the default value to the static variable. **[Allocation Default value Is Happens]**

6.After that Compiler Assigns Actual value to the Static variable if it actual value is not, they’re then nothing worry about it because the default value is Present. **[Passing Actual Value of Variable]**

7.Here our static variable is gets ready for execution. Then the main method is coming for the execution of static variable.

**Static Block**

-A block which is prefixed with **static** keyword and declared inside the class block is called as Static Block

**Syntax: -**

**Static**

**{**

**Statements;**

**}**

-Static block always gets executed before the execution of Main Method Begins.

-A programmer cannot call a static block because static block does not consist

of **Any Name.**

-Static block does not return any value back to programmer because static block does not consist any **ReturnType**

-We can declare more than one static block inside the class block.

-All the static block gets executed in **Top to Bottom** order.

**Ex: -**

Class p4

{

Static

{

System.out.println(“Static Block-1”);

}

Public static void main(String args[])

{

System.out.println(“Main Method”);

}

Static

{

System.out.println(“Static Block-2”);

}

}

**Static Initializer Block [SIB]: -**

-With the help of Static Block we can initialize a Static variable before the Execution of Main method Begins.

-Hence Static Block it is also called as Static Initializer Block [SIB].

**Ex: -**

Class P5

{

Static int a;

Static

{

a=200;

System.out.println(a);

}

Public static void main (String args [])

{

System.out.println(a);

}

}

**{**

**a=200;**

**Sop(a);**

**}**

**static**

**1.Directly Name 🡺Local Var🡺Present🡺Access**

**2.Not Present🡺Static Var**

**a**

**~~0~~ 200**

**Main**

**{**

**Sop(a);**

**}**

**1.Directly Name 🡺Local Var🡺Present🡺Access**

**2.Not Present🡺Static Var**

gg

**Static Area**

**Stack Area**

-We can access the static variable from static block in two ways:-

**1]Directly By variable Name**

**2]Class Name As Reference.**

**Ex:-**

Class P6

{

Static int a;

Static int b;

Static

{

System.out.println(a);

A=10;

Int a=20;

System.out.println(a);

System.out.println(P6.a)

}

Static

{

B=50;

Int b=40;

System.out.println(P6.b);

System.out.println(b);

}

Public static void main(String args[])

{

System.out.println(a);

System.out.println(b);

}

}

**Static -1**

**{**

**Sop(a);**

**A=10;**

**Int a=30;**

**Sop(a)**

**Sop(P6.a);**

**}**

**a**

**~~0~~ 30**

**b**

**~~0~~ 50**

**Static -2**

**{**

**B=50**

**Int b=40;**

**Sop(P6.b);**

**Sop(b);**

**}**

**Main**

**{**

**Sop(a);**

**Sop(b);**

**}**

**Static Method: -**

-A method which is declared inside the class Block and Prefixed with Static keyword is called as Static Method.

-We can declare more than one Static method inside the class Block.

-We can Access the Static Method in Two ways: -

**1]Directly by Method Name**

**2]Class Name as Reference.**

Class P7

{

Static public void test()

{

System.out.println(“test()”);

}

Public static void main(String[] args)

{

test();

P7.test();

}

}

**O/P: -**

**🡺test()**

**🡺test()**

-We can Access the static method from one class to Another class with the help

Of **Class Name as Reference.**

Ex: -

Class P8

{

Static public void test()

{

System.out.println(“test()”);

}

}

#Class P9

{

Public static void main(String args[])

{ //Access the Previous Class Method

System.out.println(P8.test());//Class Name As Reference

}

}

-**Main Method** gets loaded before the execution of **Static Block**.

-Inside the class block we can declare the Three types of Static Members

**1]Static Variable**

**2]Static Method**

**3]Static Block**

QNo1.Can we access a static method from a Static Block? [In how many Ways]

**Ans: -**

#Class P1

{

Static public void demo()

{

System.out.println(“Demo()”)

}

Static

{

Demo();

P1.demo()

}

}

QN02.Can we access the static variable from a Programmer Created Static Method? [In how Many Ways]

Ans: -

QNo3.Can we Initialize a static variable from a Programmer created Parameterized Static Method.

QNo4.Can we access the Static variable from a static block [In how many ways]

QNo5.can, we access a static variable and a static method from One Class to another class with the help Static block?

**Ex: -**

Class P9

{

Static int a=10;

Static int b=20;

Public static void test()

{

System.out.println(a);

System.out.println(P9.a);

}

Static

{

P9.test();

}

Public static void main(String [] args)

{

Test();

System.out.println(P9.a);

}

**O/P: -**

**🡺10**

**🡺10**

**🡺10**

**🡺10**

**🡺10**

**static**

**Frame Created For Execution**

a

{

P9.test()//mcs

}

~~0~~ 10

**Frame Deleted After Execution**

b

**Test()**

~~0~~ 20

{

Sop(a);

Sop(P9.a);

}

**Test()**

**Frame Created For Execution**

**main**

00

**test()**

{

Sop(a);

Sop(P9.a);

}

**Frame Deleted After Execution**

**Static Area**

**main**

{

test()//mcs

Sop(P9.a);

}

**Steps: -**

**Stack Area**

**1.Static Variable**

**2.Static Method**

**3.Static Block**

**4.Execution Main Method Begin**

**Que.Loading Process of The Class [Class Loading Process]**

**Que.Loading Process of Static Members**

🡺During Loading Process of Static members **Static Variable** gets loaded first.

🡺Once after Loading process of static variable is completed **Static Method** gets loaded Next.

🡺Once after the loading process of **Static Variable** and **Static Method** is completed then all the **Static Blocks** of the Class get Executed in Top-to-Bottom Order.

🡺Once after the execution of all The Static Blocks Is completed then Execution of **Main Method** will Begins.

**Note: -**

🡺Programmer does not create Memory Allocation for Static Members.

🡺Static Members gets Memory Allocated Automatically during the Compile Time.

**Loading**  **Execution**

Static Variable

**Static Block**

{

Directly

Class Name Reference

}

**Static Method**

**Main**

{

Directly

Class Name Reference

}

{

Directly

Class Name Reference

}

**Non-Static Members: -**

Non-Static Members are classified into three types: -

**1]Non-Static Variable**

**2]Non-Static Method**

**3]Non-Static Block**

-All the Non-Static Members are declared inside the class block and Not Prefixed with Static Keyword.

**Ex:-**

Class

{

**//N0n Static Variable**

Datatype VariableName;

Modifier ReturnType MethodName (formal Arguments)

{

**//Non-Static Method**

Statements;

}

{

**//Non-Static Block**

Statements;

}

}

-For all the Non-static members compiler does not allocates /creates memory

Automatically.

-For all the Non static members an object for the Entire Class is created.

-Before creating an object for the entire class, we need to create Object Reference Variable for the class First such that we can store the Address of the Object which is created for the Entire Class.

-In order to create an object for the class we need to take the help **Constructor.**

**Constructor**

**-**Constructor it is used to create an object for the class with help of New Keyword.

**Syntax: -**

**ClassName objectReferenceVariable=new Keyword ClassName ();**

**Eg: -**

**P1 var=new P1()**

-We can create more than one object for the class.

**Ex: -**

Class P1

{

Public static void main(String args[])

{

//Object Creation for Class P1

P1 var=new P1();

P1 var1=new P1();

System.out.println(var);

System.out.println(var1);

}

}

**Note: -** We can create an Object for One class from Another Class.

#class P1

{

}

#class P2

{

Public static void amin(String args[])

{

//for another class Object Creation

P1 var =new P1();

System.out.println(var);

//for its own class

P2 var1=new P2();

System.out.println(var1);

}

}

O/P:-

🡺P1@123652

🡺P2@367392

**Non-Static Variable**

-The variable which is declared inside the class block and Not Prefixed by Static Keyword is called as Non-Static Variable.

-In order to load the Non-Static variable, we need to create Object for the Class.

-We can access and Initialize a Non-Static variable with the help of object Reference Variable.

-We can declare more than one Non-Static variable inside the class block but different variable names.

-Non-Static variable consist of Default Values.

Ex:-

Class P3

{

Int a;

Int b=20;

Public static void main(String args[])

{

P3 var=new P3();

System.out.println(var);

System.out.println(var.a);

var.a=10;

System.out.println(var.a);

System.out.println(var.b);

}

}

🡺P3@1735675

🡺0

🡺10

🡺20

#class P4

{

Int a;

Public static void main(String args[])

{

P4 var=new P4();

System.out.println(var);

System.out.println(var.a);

var.a=10;

System.out.println(var.a);

P4 obj=new P4();

System.out.println(obj);

System.out.println(obj.a);

obj.a=20;

System.out.println(obj.a);

System.out.println(var.a);

}

}

P4 var

P4@100

P4@100

P4 var=new P4();

Sop(var);

Sop(var.a);

var.a=10;

Sop(var.a);

a

Instance of class P4

~~0~~ 10

a

~~0~~ 20

P4@200

P4 obj

P4@200

P4 obj=new P4();

Sop(obj);

Sop(obj.a);

obj.a=20;

sop(obj.a);

sop(var.a);

Instance of class P4

**Heap Area**

**Constructors: -**

-Constructor has a Same Name as Class Name.

-Constructor does not return any value back to programmer because it does not have any ReturnType.

-For Execution of Constructor Calling is Mandatory.

Constructors are classified into two types: -

**1]No Argument Constructor**

**2]Parameterized Constructor**

**1]No Argument Constructor**

-A constructor declare inside the class block and does not any arguments such constructor is called as No Argument Constructor.

Ex:-

Class P3

{

P3()

{

System.out.println(“P3()”);

}

Public static void main(String args[])

{

P3 var=new P3();

P3 var1=new P3();

}

}

**2]Parameterized Constructor**

-A constructor which consist of Arguments and declared inside the class block

Is called as Parameterized constructor.

Ex:-

#class P3

{

P3(int a, int b)

{

System.out.println(a);

System.out.println(b);

}

Public static void main(String args[])

{

P3 var=new P3(20,50);

P3 var1=new P3(50,100);

}

}

**Constructor Overloading: -**

-A class consist of more than one Constructor with Same class Name and different Arguments is called as Constructor Overloading.

Ex:-

Class P4

{

P4(int a,int b)

{

System.out.println(a+b);

}

P4(char ch)

{

System.out.println(ch);

}

Public static void main(String args[])

{

P4 var=new P4(20,50);

P4 var1=new P4(‘a’);

}

}

**Default Constructor: -**

-If a programmer fails to declare at list one constructor inside the class block

The compiler Automatically adds the constructor of No Argument Type such constructor are called as Default Constructor.

**Note: -**There is no concept of default Parameterized Constructor in Java

**Reason: -**Compiler cannot define a variable name.

Ex:-

Class P4

{

Public static void main(String args[])

{

P4 obj=new P4();

}

}

To

Class P4

{

//Default Constructor

P4()

{

}

Public static void main(String args[])

{

P4 obj=new P4();

}

}

**Note: -**inside the class block if a programmer declares at list one constructor

The compiler does not add Default Constructor on such case we get compile Time Error.

Ex:-

#Class P5

{

P4(int a)

{

}

Public static void main(String args[])

{

P5 obj=newP5()**//CTE**

}

}

#Class P6

{

P6()

{

System.out.println(“P6()”);

}

Public static void main(String args[])

{

P6 obj=new P6();

System.out.println(obj);

}

}

P6 obj

P6@100

P6()

()

P6@100

{

P6 obj=new P6();

Sop(obj);

}

main

{

Sop(“P6”);

}

**Heap Area**

#Class P7

{

Int a=10;

P7()

{

Sop(a);//local,static,Non-Static

A=20;

Sop(a);

}

Public static void main(String args[])

{

P7 obj =new P7();

Sop(P7.a);//local,Static,CTE

}

}

#Class P8

{

Int a;//20//100

P8()

{

System.out.println(a);//0

A=20;

System.out.println(a);//20

Int a=25;//50

System.out.println(a);//25

A=50;

System.out.println(a);//50

System.out.println(this.a);//20

this.a=100;

System.out.println(this.a);//100

}

Public static void main(String args[])

{

P8 obj=new P8();

System.out.println(obj.a);//100

}

}

**Class P8**

Int a;

P8@100

**O/P:-**

🡺0

🡺20

🡺25

🡺50

🡺20

🡺100

🡺100

P8()

a

~~0~~ 20

{

System.out.println(a);

A=20;

System.out.println(a);

Int a=25;

System.out.println(a);

A=50;

System.out.println(a);

System.out.println(this.a);

this.a=100;

System.out.println(this.a);

}

obj

P8@100

HEAP AREA

main

P8()

Loading Of Non-Static Constructor

1.Non-Static Variable

2.Non-Static Method

3.Non-Static Block

{

P8 obj=new P8();

Sop(obj.a)

}

O/P:-🡺25 🡺25🡺50🡺25🡺25🡺25🡺50🡺25🡺25🡺100

P8@100

Int a;//25

obj

a

~~0 25~~ 100

P8()

Obj1

P8@100

P8@200

main

{

a=25;

System.out.println(a);//25

int A=50;

System.out.println(this.a);//25

System.out.println(a);//50

}

P8@200

~~0 25~~ 150

a

P8()

{

P8 obj=new P8();

Sop(obj.a);//25

P8 obj1=new P8();

Sop(obj1.a);//25

Obj.a=100;

Sop(obj1.a);//100

Obj1.a=150;

Sop(obj.a);//150

}

}

Loading Of Non-Static Constructor

1.Non-Static Variable

2.Non-Static Method

3.Non-Static Block

P8()

Loading Of Non-Static Constructor

1.Non-Static Variable

2.Non-Static Method

3.Non-Static Block

**This Keyword: -**

-This keyword it is used to point the current object based on the object creation.

Ex:-

Class P1

{

P1()

{

System.out.println(this);

}

Public static void main(String args[])

{

P1 ob=new P1();

System.out.println(ob);

}

}

-This keyword it is also point to the address of the class.

-Whenever the name of the instance and local variable both are the same then our runtime environment JVM gets confused that which one is local variable & which one is the instance variable to avoid this problem we use the this keyword.

-It is also used to call the No Argument Constructor of the current class.

-It is also used to the call Parameterized constructor of the current class.

**Initializing a Non-Static Variable with the Help of Constructor**

#Program

Class Student

{

String name;

Int rollno;

String Mnumber;

Student(String n, int r,String m)

{

name=n;

this.rollno=r;

this.Mnumber=m;

}

Public static void main(String args[])

{

Student s1=new s1(“Sagar”,05,”3269832896”);

System.out.println(s1.Name);

System.out.println(s1.rollno);

System.out.println(s1.Mnumber);

}

}

String name;

Int rollno;

String Mnumber;

Student

S1@100

S1

S1@100

Name

null Sagar

rollno

S1

0 123

**Student(String n,int r,String m)**

{

Name=n;

This.rollno=n;

This.Mnumber=m;

}

Mnumber

null 938926257

Main()

**Student s1=new Student(“sagar”,123,”9838926257”);**

**Sop(s1.name);**

**Sop(s1.rollno);**

**Sop(s1.Mnumber);**

**HeapArea**

Student(String n,int r,String m)

Loading Of Non-Static Members

1.Non-Static Variable

2.Non-Static Method

3.Non-Static Block

**Non-Static Block: -**

-A block which is inside the class block and Not Prefixed with Static keyword is called as Non-Static Block.

**Syntax: -**

{

Statements;

}

-For the execution of the Non-Static Blocks object creation is mandatory for the class.

-A programmer cannot call the Non-Static block because it does not consist any Name.

-A Non-Static block does not return any value back to programmer because it does not consist of any ReturnType.

-Non-Static block gets executed only once for each object creation.

-We can declare the more than one Non-Static block inside the class block and all the Non-Static blocks gets executed in Top-to-Bottom order.

#Program

Class P7

{

{

System.out.println(“NS-BLOCK-1”);

}

{

System.out.println(“NS-BLOCK-2”);

}

Public static void main(String args[])

{

P7 obj=new P7();

P7 obj1=new P7();

}

}

#Program

Class P2

{

Static

{

//Executes Before the execution of the Main Method Begins

System.out.println(“Static Block”);

}

{

//Executes during the object creation

System.out.println(“Non-Static Block”);

}

P2()

{

//Executes after the Object creation

System.out.println(“P2()”);

}

Public static void main(String args[])

{

P7 obj=new P7();

}

}

**Non-Static Method**

-A method which is declared inside the class block and not prefixed by Static keyword is called as Non-Static Method.

-We can declare more than Non-Static method inside the class block.

-In order to load Non-Static method Object creation for the Entire is Mandatory.

-We can access the Non-Static method with the help of Object Reference variable.

#Program

Class P1

{

Public void test()

{

System.out.println(“test()”);

}

Public static void main(String args[])

{

P1 obj=new P1();

obj.test();

P1 obj1=new P1();

obj1.test();

}

}

Assignment Question

**Non-Static Block**

QNo1.Can we access the variable from a non-static block.[In two Ways](Directly🡺this keyword)

QNo2.Can we Initialize a Non-Static variable from a Non-Static Block

[In two Ways](Directly🡺this keyword).

QNo3.Can we call a No Argument Non-Static method from the Non-Static Method

QNo.4.Can we call Parameterized Non-Static Method from the Non-Static Block.

QNo5.Can we call a No Argument and a Parameterized method of Non-Static from No Argument and Parameterized Constructor.[In How Many Ways]

**Non-Static Method**

QNo1.Can we Access the and Initialize a Non-Static variable from a No Argument Non-Static Method[in How Many ways]

QNo2.Can we Access and Initialize a Non-Static variable from a Parameterized Non-Static Method.

QN03.Write A Program to find the summation of First 10 Number and Return the result using Non-Static Method.

QNo4.Can we perform method overloading with the help of Non-Static Method.

QNo5.Can we perform the recursive method with the help of Non-Static method

QN06.Can we call One Non-Static Method from Another Non-Static Method

QN07.Can we Access the Static variable from main method.[In how Ways].

Que.Why Non Static block required Object creation.

Que.String[] instead …

Que.what is use of creating object inside the class block.

Que.Can we call main method inside main method

Main(new String[0])

🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺

This Keyword:

This keyword can be used only inside the non static members (non static block ,no argument constructor,parameterized constructor and non static method)

This keyword it is used to point current object.

We cannot use this keyword inside the static members (static block, static method)

Important:

We can access a non static variable a non static method from static block and static method with help of object reference variable.

In order to access object reference variable object creation for class is mandatory.

**Instance initializer block:**

We can initialize a non static variable from non static block during object creation in two ways

1.directly

+2.using this keyword

Hence non static block is also called as instance initializer block(IIB)

Program:

Class P5{

Public static void main(String[] args){

Int n=10;

{

}

}

**Loading Process of non static method:**

To non static members of the class object creation for the class is mandatory.

During the object creation non static variables are always gets loaded first.

Once after all the non static variable gets loaded completely then non static method gets loaded next.

Once after all the non static variable and non static method gets loaded completely then non static block gets executed in top to bottom order.

Once after all the non static block gets executed in top to bottom In order then execution of constructor begins.

What is the use of Constructor:

1.With help of constructor we can load non static members of the class inside the object.

2.with help of constructor we can initialize the non static variable after object creation.

3.with help of constructor we can access both non static and static members.

**Object Oriented Programming:**

Object Oriented programming it is concept of (structure)where we take all the real time examples and convert into a programming language.

Object:

Object it is an real time example object consist of 2 element:

1.properties

2.behaviours

Properties are converted into a variable.

Behaviors are converted into a methods.

Variable and method are the property of the class.

Hence object (real time example) is converted into a class.

**Class:**

Class it is called as blue print of the object.

Object Oriented Programming consist of following Pilers

1.Encapsulation

2.Inheritence

3.Polymorphism

4.abstraction

1.encapsulation:

It is process of binding between properties and behaviours of an object.

Advantage:

Data hiding

2.Inheritence

It is process of acquiring the properties and behaviours of parent class inside the child class is called inheritance.

Advantage:

Reusability

3. Polymorphism

It is process of an object which exhibit more than one form is called as polymorphism.

4.Abstractions

It is process of hiding the implementation is called as abstraction.

**Encapsulation:**

It is process of binding between properties and behaviours of an object.

Encapsulation is process of the achieving the data hiding is called as encapsulation.

**Data Hiding**:

Hiding the data (Value) from the outside world is called data hiding.

We can hide the data with help of private keyword.

**Private Keyword:**

Private keyword it is an access modifier. When we prefix any variable with private keyword we can not access the private variable from another class.

(for printing and modifying the value)

In order to access private variable we need to create two methods

1.getter method

2.setter method

**1.Getter method:**

Getter method it is used to access private variable (printing ,reading method header of getter method)

Access modifier return type getMethodName(){

Return private variable;

}

Return type of getter method depends on the variable which is private.

**2.Setter method:**

Setter method It is used to modify the private variable.

Method header of setter method

Access modifier void setMethodName(datatype Variablename){

This.private\_variable=new variable\_name;

}

Program:

Class Student

{

//n-s variable

String name;

int rollno;

Private String Mnumber;

Student(String n,int r,String m)

{

This.name=n;

This.rollno=r;

This.Mnumer=m;

}

//getter method

Public String getNumber()

{

Return Mnumber;

}

//setter method

Public void setNumber(String number)

{

This.Mnumber=number;

}

//non static method

Public void details()

{

System.out.println(this.name);

System.out.println(this.rollno);

System.out.println(this.Mnumber);

}

}

Class School

{

Public static void main(String[] args)

{

Student s1=new Student(“Sager”,123, “9876543211”);

System.out.println(s1.Mnumber);//CTE (variable is private)

//calling getter method

System.out.println(s1.getNumber());

//s1.Mnumber=”9874543611” //CTE (variable is private)

//calling setter method

S1.setNumber(“9876543211”);

System.out.println(s1.getNumber());

}

Null sagar

Name

Rollno

Mnumber

Getter and setters

Details(){

}

**Private**

Student(String n,int r,String m){

Name=n;

Rollno=r;

Mnumber=m

}

0 125

Null 9876

Main(){

Students1=newStudent(“Sager”,123, “9876543211”);

System.out.println(s1.Mnumber);//CTE (variable is private)

//calling getter method

System.out.println(s1.getNumber()); //s1.Mnumber=”9874543611” //CTE (variable is private)

//calling setter method

S1.setNumber(“9876543211”);

System.out.println(s1.getNumber());

}

Student@1

1.non static variables

2.non static method

3.non static block

4.constructor

**Create one program using setter and getter method and private keyword.**

**Program:**

Class Gmail

{

Private String gamil;

Private String password;

Gmail(String gamil,String password)

{

This.gamil=gmail;

This.password=password;

}

Public void setGmail(String gamil)

{

This.gamil=gamil;

}

Public void setPassword(String Password)

{

This.password=password;

}

Public String getGamil()

{

Return gmail;

}

Public String getPassword()

{

Return password;

}

}

Class GmailDriver

{

Public static void main(String[] args)

{

Gmail g1=new Gmail(“rushimisal5@gmail.com”, “Rushikesh@9096”);

// System.out.println(g1.gamil); //CTE

System.out.println(g1.getGamil() );

g1.setPassword(“124”);

System.out.println(g1.getPassword());

}

}

**Diagram:**

Null rushim

gamil

password

Getter and setters methods

**Private**

Gmail(String gamil,String password){

This.gamil=gmail;

This.password=password;

}

Null 125

Gamil@1

Main(){

Gamil g1=new Gmail(“rushimisal5@gmail.com”, “Rushikesh@9096”);

// System.out.println(g1.gamil); //CTE

System.out.println(g1.getGamil() );

g1.setPassword(“124”);

System.out.println(g1.getPassword());

}

}

1.non static variables

2.non static method

3.non static block

4.constructor

**Relationship:**

Relationship It is classified into two types

1.has-a-releationship

2.is-a-relationship

**1.has-a- relationship:**

Has-a-relationship it behaves like a complete whole object and multiple parts of it

Examples:

1.company has a employee

2.bike has a engine

3.youtube has a subscribers

4.grosary store has a customers

5.Aroplane has a passengers

**2.is-a-relationship:**

Is- a-relationship it behaves like a parent and child.

Examples:

1.flult is a musical instrument

2.facebook is a social media application

3.laptop is a gadget

4.camera is a device

5.bike is a vehicle

**Inheritance:**

Definition:

It is process of acquiring the properties and behaviors from parent class to child class is called as Inheritance.

We can perform inheritance with help of extends keyword.

Extends keyword represent is-a relationship.

Parent class it is also called as super class.

Child class it is also called as sub class.

It is process of acquiring the properties and behaviors of super class to

Sub class is called as Inheritance.

**Program:**

Class A{

Static int a=100;

Public static void tets(){

System.out.println(“test()”)

}

Public static void main(String[] args){

//members of its own class

System.out.println(a);

Test();

}

}

Class B extends A{

Static int b=10;

Public static void demo(){

System.out.println(“demo()”);

}

Public static void main(String[] args){

//members of its own class

System.out.println(b);

Demo();

//members of its parent class

System.out.println(a);

Test();

}

Variable -a

Method-tets()

Main(){

Tets()--

}

parent class

or

Incorpo

rated

members

extends

Variable -b

Method-demo()



Main(){

B

Demo()

A

Tets()

}

Child clas

Or Incorpo

rated

members

Sub class

Inherited

Members

Properties are called as variable (static variable,non static variable ,local variable)

Behaviors are called as method(static and non static)

In the above program class A it is behaving like super class or parent class of class of B

Here class B it is behaving like child of parent class A

Child class B can access the members of its own class such members are called as incorporated members.

Child class can also access the members of its parent class such members for child class are called as inherited members.

**Loading process of static members using inheritance:**

**Program:**

Class A{

Static int a=100;

Static{

System.out.println(“SIB-A”);

}

Public static void test(){

System.out.println(“test()-A”);

}

}

Class B extends A{

Static int b=20;

Static {

System.out.println(“SIB-B”);

}

Public static void demo(){

System.out.pritnln(“Demo()”);

}

Public static void main(String[] args){

//calling the members directly by its name

System.out.println(b);

Demo();

System.out.println(a);

Test();

}

}

**Note-1: calling the members of its own class and its parent class with help of class name as reference**

Public static void main(String[] args){

//calling the members Class name as reference

System.out.println(B.b);

B.Demo();

System.out.println(A.a);

A.Test();

}

**Note-2:**

**We can access static members with help of object reference variable only when there is no non static members with the particular name.**

Public static void main(String[] args){

B obj=new B();

System.out.println(obj.b);

Obj.demo();

//members of the parent class

System.out.println(obj.a);

Obj.test();

}

**Note-3**

**We can access static members in four ways**

**1.direclty by its name(variable,method).**

**2.with help of class name as reference.**

**3.with help of object reference variable**

**4.with the help of this keyword.(From Non-Static Context)**

**5.with the help of super keyword.(From Non-Static Context)**

Super most class

Object

* 1. static variable
  2. static method
  3. static block

constructor invocation A constructor execution

B

Main(){

Sop(b);

Demo();

Sop(a);

Test();

}

Directly --->local --🡪static variable own class-🡪static variable parent clas Directly🡪static method of own class

Directly-🡪local-🡪static variable own class -🡪static variable par Directly🡪own class static method -🡪parent class static method

stack



Object class static members

A

0 100

A

Test()

B

0 20

B

Demo()

B

Main()

Static area

**Loading process of non static members using inheritance:**

**In order to load non static members we need to crate an object for child class. Such that we can load the members of own class and its parent class**

**Program:**

Class A{

Int a=100;

Public void test(){

System.out.println(“test()”);

}

{

System.out.println(“IIB-A”);

}

}

Class B extends A{

Int b=20;

Public void demo(){

System.out.println(“demo() “);

}

{

System.out.println(“IIB -B”);

}

Public static void main(String[] args){

//object creation for child class

B obj=new B();

System.out.println(obj):

System.out.println(obj.a);

System.out.println(obj.b);

Obj.demo();

Obj.test();

}

}

Object(){

}

B@100 address

Object class non static members

Instance of object class

B@100

B Obj=new B();

Sop(obj.a);

Obj.demo();

Obj.test();

A(){

Super();

}

B(){

Super();

}

A

0 100

Instance of class A

Tets(){}

B

0 20

Instance of class B

demo(){}

1.non static variable 1.non static variable

2.non static method heap memory

3.non static block T

4.constuctor b

**Constructor Chaining:**

**It is process of one constructor calling another constructor is called constructor Chaining.**

**Constructor chaining is classified into two types:**

**1.Super call statement:**

**Super call statement is used to call the constructor of its parent class.**

**Super call statement should be always declared as first statement inside the constructor.**

**Example:**

**A () {**

**Super ();**

**}**

**We can not declare super call statement in between the constructor we get compile time error.**

**Example:**

**A(){**

**System.out.println(“A()”); ------>CTE**

**Super ();**

**}**

**super call statement of no-argument type gets added automatically as the first statement inside the constructor.**

**Example:**

**A(){**

**System.out.println(“A()”);**

**}**

**To**

**A(){**

**Super (); ---->gets added automatically**

**System.out.println(“A()”);**

**}**

**Program:**

Class A{

Int a=100;

Public void test(){

System.out.println(“test()--A”);

}

{

System.out.println(“IIB--A”);

}

A(){

System.out.println(“A()”);

}

}

Class B extends A{

Int b=20;

Public void demo(){

System.out.println(“Demo()---B”);

}

{

System.out.println(“IIB--B”);

}

B(){

System.out.println(“B()”);

}

}

Class C extends B{

Int c=20;

Public void key(){

System.out.println(“key()--B”);

}

{

System.out.println(“IIB--C”);

}

C(){

System.out.println(“c()”);

}

Public static void main(String[] args){

C c=new C();

}}

C@100 address

Object class non static members

Instance of object class

Object(){

}

Instance of class A

A

0 100

Tets(){}

B(){

Super();

}

B

0 20

Instance of class B

demo(){}

B(){

Super();

}

B

0 20

Instance of class C

Key(){}

C(){

Super();

}

1.non static variable 1.non static variable

2.non static method heap memory

3.non static block T

4.constuctor b

C Obj=new C();

C @100

**Super Most Class**

Object

1. static variable
2. static method
3. static block

constructor invocation A constructor execution

B

C

**Program:**

Class A{

Int a=100;

Public void test(){

System.out.println(“test()”);

}

{

System.out.println(“IIB-A”);

}

}

Class B extends A{

Int b=20;

Public void demo(){

System.out.println(“demo() “);

}

{

System.out.println(“IIB -B”);

}

Public static void main(String[] args){

//object creation for child class

B obj=new B();

System.out.println(obj):

System.out.println(obj.a);

System.out.println(obj.b);

Obj.demo();

Obj.test();

}

}

🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺🡺

**Note 1: -**

-If a parent class consist of parameterized constructor and to call this constructor from child class writing the super() call statement and passing value is Mandatory inside the child class constructor.

Ex:-

#program

**OBJECT CLASS**

Class A

{

Super();

A(int a)

{

System.out.println(“A(int)”);

}

**Constructor Invocation**

}

Class B extends A

**Constructor Execution**

{

B()

{

Super(12);**//Mandatory**

System.out.println(“B()”);

}

Public static void main(String args[])

{

B obj=new B();

}

}

Constructor Invocation[Calling]:-

B()🡺B(12)🡺A()🡺Object Class()

Constructor Execution:-

Object class()🡺A()🡺B()

**OBJECT CLASS**

#Program2

Class A

{

A(int a)

{

Super();

System.out.println(“A(int)”);

}

A(double a)

{

**Constructor Invocation**

Super();

**Constructor Execution**

System.out.println(“A(double)”);

}

}

Class B extends A

{

B()

{

Super(‘a’);

System.out.println(“B()”);

}

Public static void main(String args[])

{

B obj=new B();

}

}

**This Call Statement[this()]: -**

-This call statement it is used to call the constructor of its own class.

-This call statement does not gets added automatically.

-Writing this call statement as the first statement inside the Constructor Block is Mandatory for the Programmer.

-With the help of this call statement we can call No Argument Constructor &

Parameterized Constructor of its Own Class.

-We cannot define this call statement in Between the Constructors we get Compile Time Error.

Ex: -

#Program

Class A

{

A(int a)

{

This();

System.out.println(“A(int)”);

}

A(double a)

{

System.out.println(“A(double)”);

}

A()

{

This(123.123);

System.out.println(“A()”);

}

Public static void main(String args[])

{

A obj=new A(12);

}

}

Constructor Invocation: -

A(12)🡺A(int a)🡺A()🡺A(double a)

Constructor Execution:-

A(double a)🡺A()🡺A(int a)🡺A(12)

#Program 2

Class A

{

A(int a)

{

This();

System.out.println(“A(int)”);

}

A(double a)

{

This(123);

System.out.println(“A(double)”);

}

A()

{

System.out.println(“A()”);

}

}

Class B extends A

{

B()

{

Super(123.123);

System.out.println(“B()’);

}

B(char ch)

{

This(123);

System.out.println(“B(char)”);

}

B(int a)

{

Super(150);

System.out.println(“B(int)”);

}

Public static void main(String args[])

{

B obj=new B();

B obj1=new B(123);

B obj2=new B(‘a’);

B obj3=new A(‘a’);

}

}

**B obj=new B();**

Constructor Invocation: -

B()🡺B()🡺A(double a)🡺A(int a)🡺A()🡺Object()

Constructor Execution: -

Object()🡺A()🡺A(int a)🡺A(double a)🡺B()🡺B();

**B obj1=new B(123);**

Constructor Invocation: -

B(123)🡺B(int a)🡺A(int a)🡺A()🡺Object()

Constructor Execution: -

Object()🡺A()🡺A(int a)🡺B(int a)🡺B(123);

**B obj2=new B(‘a’);**

Constructor Invocation: -

B(‘a’)🡺B(char ch)🡺A(int a)🡺A()🡺Object()

Constructor Execution: -

Object()🡺A()🡺A(int a)🡺B(char ch)🡺B(‘a’);

**A obj3=new A(‘a’);//Widening**

Constructor Invocation: -

A(‘a’)🡺A(int a)🡺A()🡺Object()

Constructor Execution: -

Object()🡺A()🡺A(int a)🡺A(‘a’);

**QNo1.Can we write this call statement and super call statement together inside a single constructor.**

**Ans: -** No we cannot write the Super() statement and this() statement

Together neither we get compile Time Error.

**#Program**

class P2

{

P2()

{

this();

super();

}

public static void main(String[] args)

{

P2 obj=new P2();

}

}

P2.java:6: error: call to super must be first statement in constructor

super();

^

P2.java:5: error: recursive constructor invocation

this();

QN02.Can we write this call statement & super call statement together inside a single program.

**Ans: -**

**#Program**

class P3

{

P3()

{

this(123);

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

}

P3(int a)

{

super();

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

}

public static void main(String[] args)

{

P3 obj=new P3();

}

}

super()

this()

Press any key to continue . . .

QN03.Write a difference between this keyword and this call statement.

Ans: -

|  |  |  |
| --- | --- | --- |
| SR.NO | This Keyword | This() Statement |
| 1. | This keyword Points to object of the current class. | this() statement used to call current class constructor |
| 2. | Inside this keyword the address of the current object is Present. | Using this() statement we can call both No Argument & Parameterized Constructor of the current class |
| 3. | Using this keyword we access the Non-static & Static Members | Using this() statement we can Pass the Value to the Parameterized Constructor |
| 4. | Inside the class if we declare local variable and Non-Static Variable to access the Non-Static variable we need to help of this keyword. | This call statement is not added automatically. |
| 5. | This is keyword in Java. | This() statement Should be first  Statement inside the constructor. |
| 6. | This keyword should be used in only Non-Static members. [Non-StaticMethod,Non-Static Block,Constructor] | This() statement we can only use inside constructor. |
| 7. | Using this keyword we can call Non-Static methods inside Non-Static Members | To call Parameterized Constructor of own class Passing a value inside this() statement is Mandatory. |

QNo.4Write a Difference between This() & super() statement.

QN05.When Super() statement is Mandatory.

**Types of Inheritance: -**

Inheritance is classified into 5 types

1]Single-Level Inheritance

2]Multilevel Inheritance

3]Multiple Inheritance

4]Hierarchical Inheritance

5]Hybrid Inheritance

**1]Single-Level Inheritance**

-Its process of acquiring the Properties or Behaviour from Parent Class to its child class is called as Single-Level Inheritance.

-Single level Inheritance it is also called as One-Level Inheritance.

Class A

Variable

Methods

Access Members:-

Own

Parent

Parent Class/

Super Class

**Extends One Level**

Class B

Variable

Methods

Access Members:-

Own

Parent

Child Class

/Base Class

**2]Multilevel Inheritance**

Acquiring the Properties and behaviour from more than one Level is called as Multilevel Inheritance.

Class A

Variable

Methods

Access Members:-

Own

Parent

Parent

Parent Class/

Super Class

One Level

**Extends**

Class B

Variable

Methods

Access Members:-

Own

Parent

Child

Child Class

/Base Class/

Sub Class Parent

Class C

**Extends**

Variable

Methods

Access Members:-

Own

Parent

One Level

Child Class

/Base Class/

Sub Class

Child

**3]Hierarchical Inheritance**

-A parent class consist of more than one child class is called as Hierarchical Inheritance.

**Diagram 1:-**

Parent Class

**Class A**

Variable

Methods

Access Members:-

Own

Parent

**Extends** **Extends**

Child Class Child Class

**Class C**

**Class B**

Variable

Methods

Access Members:-

Own

Parent

Variable

Methods

Access Members:-

Own

Parent

**Diagram 2:-**

Parent class

**Class A**

Variable

Methods

Access Members:-

Own

Parent

**Extends** **Extends** **Extends**

Child Class Child Class

**Class D**

**Class C**

**Class B**

Variable

Methods

Access Members:-

Own

Parent

Variable

Methods

Access Members:-

Own

Parent

Variable

Methods

Access Members:-

Own

Parent

**4.Multiple Inheritance: -**

-A subclass consist of more than one Super class Is called as Multiple Inheritance.

-We cannot perform multiple inheritance in java with the help of classes.

Class B

Class A

Variable

Methods

Access Members:-

Own

Parent

Variable

Methods

Access Members:-

Own

Parent

**Extends Extends**

Class C

Variable

Methods

Access Members:-

Own

Parent

**Diamond Problem: -**

Parent Parent

Class B

Class A

Class Class

**{**

**B()**

**{**

**}**

**}**

**{**

**A()**

**{**

**}**

**}**

**Diamond Problem**

Child Class

**Class C**

**{**

**C()**

**{**

**Super()**

**}**

**}**

**C obj=new C();**

In the above structure the sub class C consist more than one Parent Class [Class A, Class B]

-When we create an object for sub class c the Constructor of Class C will be called for execution first.

-The First statement inside the constructor of Class C super() statement.

-The super() statement it is used to call the constructor of its Parent class.

-Since class c consist of two Super Classes the super() statement will get Confused which Parent class Constructor to be called for Execution.[Class A Class B].

-Because of this confusion which occurs during constructor Chaining we cannot perform Multiple inheritance in java.

-This Confusion Is also called Diamond Problem[Ambiguity Problem].

**5.Hybrid Inheritance**

-Its combination of Multiple & Hierarchical Inheritance in java.

Class A

**{**

**A()**

**{**

**}**

**}**

**{**

**D()**

**{**

**}**

**}**

**{**

**C()**

**{**

**}**

**}**

**{**

**B()**

**{**

**}**

**}**

**Extends Extends**

Class C

Class B

**Diamond**

**Problem**

**Extends Extends**

Class D

QNo1.create two Realtime example for Single level Multilevel & hierarchical Inheritance.Each class should consist of three Properties and two Behaviour.

**POLYMORPHISM: -**

-An object exhibiting more than one form is called as Polymorphism.

**POLY 🡺Many**

**MORPHISM 🡺 Different Forms.**

-Polymorphism is classified into two types

**1]CompileTime Polymorphism**

**2]RunTime Polymorphism**

**1]CompileTime Polymorphism**

-It is process of binding between Method Call Statement and Method Implementation during the CompileTime Is called as CompileTime Polymorphism.

-CompileTime Polymorphism it is also called as Early Binding.

-**BINDING🡺 is the connection 🡺EarlyBinding**

-CompileTime Polymorphism is Classified into Two Types: -

**1]Method Overloading**

-A class consist of More than one Method with Same Name and Different Arguments is called is **Method** **Overloading.**

-Inside the class block we cannot declare more than one Method with Same Name & Same Arguments.

**Ex: -**

Class P2

{

public static void test ()

{

System.out.println(“test ()”);

}

public static void test (int a)

{

System.out.println(“test(int)”);

}

public static void main(String args[])

{

test (10);

}

}

Class P3

{

public static void test (double a)

{

System.out.println(“test (double)”);

}

public static void test (int a)

{

System.out.println(“test(int)”);

}

public static void main(String args[])

{

test (‘a’);

}

}

**test (double b)**

**{**

**test (double a)**

**}**

**{**

**test(int a)**

**}**

**test(int a)**

**main()**

**{**

**Test(‘a’);**

**}**

**2]Constructor Overloading**

-A class consist of more than one Constructor with Same class Name and different Arguments is called as Constructor Overloading.

Ex:-

Class P4

{

P4(int a,int b)

{

System.out.println(a+b);

}

P4(char ch)

{

System.out.println(ch);

}

Public static void main(String args[])

{

P4 var=new P4(20,50);

P4 var1=new P4(‘a’);

}

}

**2]RunTime Polymorphism**

-It is a Process of Binding between Method Call Statement and Method Implementation during RunTime Is called as RunTime Polymorphism.

-RunTime Polymorphism it is also called as Late Binding.

-Runtime Polymorphism it is classified into two types: -

**1]Derived Typecasting**

**2]Method Overriding**

**1]Derived Typecasting / Non-Primitive Typecasting**

-It is process of converting address [Reference] from One class to Another Class Is Called as Non-Primitive Typecasting or Derived Typecasting.

-It is also called As Non-Primitive Typecasting[It is Process of converting the One Non-Primitive datatype into Another Non-Primitive Datatype].

-Derived Typecasting is classified into two types: -

**1]UpCasting**

**2]DownCasting**

**1]UpCasting**

-It is the process of converting the Address [Reference] from Child Class to Parent class is called as UpCasting.

-In order to perform UpCasting we need to create an object of an Child Class [Such that we can load the Members Of its Own class & Parent Classes].

-We need to create an object reference variable for Parent class and Pass the Address from Child Class.

**#Program**

Class Employee

{

Public void work()

{

System.out.println(“Work is Mandatory For An Employee”);

}

}

Class SoftwareEngineer

{

Public void app()

{

System.out.println(“Work Regarding Software Applications”);

}

}

Class SoftwareDev

{

Public void coding()

{

System.out.println(“Dev the Software by Creating Code”);

}

}

Class Company

{

Public static void main(String args[])

{

//Object creation for Child Class

SoftwareDev sd=new new SoftwareDev();

sd.coding();//Members of its own Class

sd.App();//Members of its Parent class

sd.work();//Members of its Parent class

System.out.println();

//UpCasting from Child To Parent Class

//SoftwareDev to SoftwareEngineer

SoftwareEngineer se=sd;

se.App();//Members of its Parent class

se.work();//Members of its Parent class

se.coding();//Members of its Child Class(CTE)

System.out.println();

//UpCasting from Child To Parent Class

//SoftwareEngineer to Employee

Employee e=se;

e.work();//Members of its Parent class

e.App();//Members of its Child class(CTE)

e.coding();//Members of its Child Class(CTE)

}

}

**sd@123**

Instance Of Object Class Is created

**Object Class Non-Static Members**

Instance Of Employee Class Is created

**WORK**

Employee e=se;

Instance Of SoftwareEngineer Class Is created

**APP**

SoftwareEngineer=se;

Instance Of SoftwareDev Class Is created

**CODING**

SoftwareDev=se;

Constructor Invocation

SoftDev()🡺SoftEngg🡺Emp()🡺Object Class

Constructor Execution

Object Class🡺 Emp()🡺SoftEngg🡺 SoftDev()

QNo1.Create A Realtime example for UpCasting using following Inheriatnce

1]SingleLevel

2]MultiLevel

3]Hierarchical

And Explain program by Tracing.

**2]DownCasting**

-It is the process converting address [Reference] from Parent Class to Child Class is called as **DownCasting**.

-We cannot perform the DownCasting Automatically.

-In order to perform DownCasting we need to take help of Cast Operator.

-In order to perform DownCasting we need to create an Object for Child Class.

Such that we can load the Members of its Own Class and Parent Classes.

-In order to perform the DownCasting first we have to perform the Upcasting then you can perform the DownCasting.

**#Program**

Class Employee

{

Public void work()

{

System.out.println(“Work Is Mandatory for an Employee”);

}

}

Class SoftwareEngineer extends Employee

{

Public void app()

{

System.out.println(“Work Regarding Software Applications”);

}

}

Class SoftwareTest extends SoftwareEngineer

{

Public void testing()

{

System.out.println(“Finding the Bug in the Software App”);

}

}

Class Company

{

Public static void main(String args[])

{

//UpCasting

Employee e=new SoftwareTest();

System.out.println(e);

e.work();//Members of its Own Class

e.app();//Members of its Child Class(CTE)

e.testing();//Members of its Child Class(CTE)

System.out.println();

//DownCasting from Super to Sub Class

//Employee to SoftwareEngineer

SoftwareEngineer se=(SoftwareEngineer)e;

se.app();//Members of its Own Class

se.work();//Members of its Parent Class

se.testing();//Members of its Child Class(CTE)

System.out.println();

//DownCasting from Super to Sub Class

//SoftwareEngineer to SoftwareTest

SoftwareTest st=(SoftwareEngineer)se;

st.testing();//Members of its Own Class

st.app();//Members of its Parent Class

st.work();//Members of its Parent Class

}

}

**st@123**

**Object Class Non-Static Members**

Instance Of Object Class Is created

**WORK( )**

Instance Of Employee Class Is created

Employee e

**APP( )**

Instance Of SoftwareEngineer Class Is created

SoftwareEngineer se=( SoftwareEngineer)e;

**TESTING( )**

SoftwareTest st=(SoftwareTest)se;

Instance Of SoftwareTest Class Is created

**ClassCastException**

-If we create an object for Parent class and if we try to convert the reference from Parent to Child class[DownCasting].since there is no Instance of Child Class Present inside the object we get an RunTime Exception called **ClassCastException**.

**#Program**

Class Employee

{

Public void work()

{

System.out.println(“Work Is Mandatory for an Employee”);

}

}

Class SoftwareEngineer extends Employee

{

Public void app()

{

System.out.println(“Work Regarding Software Applications”);

}

}

Class SoftwareTest extends SoftwareTest

{

Public void testing()

{

System.out.println(“Finding the Bug in the Software App”);

}

}

Class Company

{

Public static void main(String args[])

{

Employee e=new Employee();

e.work();

System.out.println()

//DownCasting from Parent to Child

//Employee to SoftwareEngineer

SoftwareEngineer se=(SoftwareEngineer)e;//Exception

se.work();

Se.app();

}

}

**st@123**

**Object Class Non-Static Members**

Instance Of Object Class Is created

**WORK( )**

Instance Of Employee Class Is created

Employee e

Instance Of SoftwareEngineer Class Is Not Created

SoftwareEngineer se=( SoftwareEngineer)e;

SoftwareTest st=(SoftwareTest)se;

Instance Of SoftwareTest Class Is Not created

-Here the SoftwareEngineer and SoftwareTest Class Instances is Not created

So we cannot perform the DownCasting without UpCasting.

-So we have to first perform Upcasting then we can do DownCasting.

-In order to perform Upcasting and DownCasting That we need to create the first Object for Child Class.

**2]Method Overriding**

-It is process of Overriding Parent class Method Implementation by Child Class Method Implementation is called as method Overriding.

**Rules To Perform Method Overriding**

1]There should be Is-a Relationship[extends]

2]Method header both Parent Class and Child Class Has to be Same.[AccessModifier,ReturtType,MethodName,FormalArguments]

-We need to create and object for Child Class.

#Program

Class Bank

{

Public void Roi()

{

Return 0;

}

}

Class Sbi extends Bank

{

Public void Roi()

{

Return 12;

}

}

Class City

{

Public void main(String args[])

{

Sbi s=new Sbi();

System.out.println(s);

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

//Upcasting From Sbi to Bank

Bank oB=s;

oB.Roi();//Method Overriding

System.out.println(oB.Roi());

}

}

**Instance of Sbi class is created**

**Instance of Bank class is created**

**Instance of Object class is created**

**Roi()**

**{**

**Return 0;**

**}**

**Roi()**

**{**

**Return 0;**

**}**

**Object class Ns Members gets Loaded**

Bank b b.Roi()

sbi@123

**UpCasting**

Sbi s

sbi@123

-In order to override the Method, we should fallow to **Is-a** Relationship [**extends**].

-In both class we have declare the same method **[Same Modifier, Same ReturnType, Same MethodName, Same FormalArguments]**.

-Then we have to perform upcasting from Child Class to Parent Class

-After Upcasting using Parent Reference variable we trying to access the Method of own class with Same Name.

- Then that time the Child Class Method is gets executed, Child Class method is overridden in Parent class.

🡺Because It will search Child Class Method First if it is find then It will Executed

Assignment Questions

QNo1.Can we Override Static Method Verifying and explain with a Program?

Ans:-No cannot override the static method

class P1

{

static public void display()

{

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

}

}

class P2 extends P1

{

static public void display()

{

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

}

}

class Driver

{

public static void main(String[] args)

{

B oB=new B();

A oA=oB;

oA.display();

}

}

QNo2.Can we Override Non-Static Variable

Ans:-No we cannot override Non-Static variable.

class A

{

int a=10;

static public void display()

{

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

}

}

class B extends A

{

int a=20;

static public void display()

{

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

}

}

class Demo

{

public static void main(String[] args)

{

B oB=new B();

A oA=oB;

System.out.println(oA.a);

}

}

QNo3.Can we Verify Static Variable

Ans:-No we cannot override Static variable.

class One

{

static int a=10;

static public void display()

{

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

}

}

class Two extends One

{

static int a=20;

static public void display()

{

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

}

}

class DriverDemo

{

public static void main(String[] args)

{

B oB=new B();

A oA=oB;

System.out.println(oA.a);

}

}

QNo4.Create Two Realtime Example for Method Overriding using

1]Single-Level Inheritance

2]Multi-Level Inheritance

3]Hierarchical Inheritance

**METHOD SHADOWING**

-Its process of Hiding Child Class Method Implementation by Parent class Method Implementation is called as Method Shadowing.

-We can perform Method Shadowing with help of Static Methods.

-We cannot perform Method shadowing with Non-Static Methods.

**#Program**

Class A

{

Static public void test()

{

System.out.println(“A”);

}

}

Class B extends A

{

Static public void test()

{

System.out.println(“B”);

}

Public static void main(String args[])

{

B obj=new B();

Obj.test();

A obj1=new A();

Obj1.test();

}

}

**VARAIBLE SHADOWING**

-It is process of hiding Child Class Property(Variable) From Parent class Property is called as Variable Shadowing.

-We can perform the Variable Shadowing with the help of Both Static & Non-Static Variable.

**ABSTRACTIONS**

-It is a Process of Hiding the implementation.

-We can Perform Abstraction with the help of Abstract Keyword.

-To Hide the Implementation, we need to take the help of methods.

-Methods has two Elements: -

**1]Method Header**

**2]Method Body [Implementation]**

**ABSTRACT METHOD**

-A Non-Static method declared inside the Class Block terminated by Semicolon and Prefixed by Abstract Keyword Is called As Abstract Method.

-We can declare More than One Abstract Method Inside the Class Block.

-We cannot Declared More than One Abstract Method with Same Name and Same Arguments.

**METHOD HEADER OF ABSTRACT METHOD**

**Syntax: -**

**abstract** **access\_modifier** **return\_type** **method\_name(args);**

Ex:-

abstract public void demo();

abstract public void demo(int a);

abstract public void demo(int a,int b);

abstract public void String Key(Staring var);

**ABSTRACT CLASS**

-A class Prefixed by Abstract Keyword is called as Abstract Class.

-If a class Consist of at list One Abstract Method then it is Mandatory to make the class as Abstract.

-

**#PROGRAM**

**abstract** class P1 **//abstract Class**

{

**abstract** public void demo();**//abstract method**

**abstract** public void demo(int a);**//abstract method**

}

**NOTE:-**

-We Cannot Prefixed the static Method Header With the abstract Keyword[We get Compile Time Error].

-**Reason: -We cannot perform method Overridding with the help of static Methods.**

-We can use abstract Keyword only for Non-Static Methods because we can perform Method Overridding Only with the help Of Non-Static Methods.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IMOPRATANT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**==============================================**

There two Data Hiding Concept in Java: -

1]Encapsulation**//For the Properties//Variables**

2]Abstraction**//For the Behaviour//Methods**

**CONCRETE METHODS**

-The Method which consist of Method Implementation[Method Body] Is called as Concreate method

-Concrete Methods is classified into two types:-

**1]Non-Static Concreate Method**

**2]Static Concreate Method**

QN01.Can we declared both abstract method and Concreate Method inside the abstract class.

Ans: -Yes, We can declare both Abstract Method and Concrete Method In Abstract Class

**Ex: -**

**#PROGRAM**

abstract class P1

{

abstract public void demo();**//abstract Method**

**//Concrete Method**

Public void test()**//NS Method**

{**//Method Implementation**

System.out.println(“NS-METHOD-TEST()”);

}

Public static void key()**//Static Method**

{**//Method Implementation**

System.out.println(“Static -Method-Key()”);

}

}

QNo2.Can we declare Only Concrete Method inside Class.

Ans: - Yes, we declare.

====================IMPORTANT RULES=====================

1]If class Consist Concrete Method, then it does not require to declare Class as an Abstract.

2]If Class Consist One or More than One Abstract Method Then it is Mandatory to declare the class as Abstract.

**Ex: -**

**#PROGRAM**

abstract class P1

{

**//Concrete Method**

Public void test()**//NS Method**

{**//Method Implementation**

System.out.println(“NS-METHOD-TEST()”);

}

Public static void key()**//Static Method**

{**//Method Implementation**

System.out.println(“Static -Method-Key()”);

}

}

QNo1.Can we create object for abstract class

Ans:- No we cannot create the object for the Abstract Class.

We get Compile Time error.

abstract class Abstract //abstract class

{

abstract public void demo();//abstract Method

//Concrete Method

public void test()//NS Method

{//Method Implementation

System.out.println("NS-METHOD-TEST()");

}

public static void key()//Static Method

{//Method Implementation

System.out.println("Static -Method-Key()");

}

public static void main(String args[])

{

Abstract oA=new Abstract();//We cannot create the Object for the

Abstract Class

System.out.println(oA);

}

}

====================IMPORTANT RULES=====================

🡺Abstract.java:19: error: Abstract is abstract; cannot be instantiated

Abstract oA=new Abstract ();//We cannot create the Object for the Abstract Class

^

1 error

====================IMPORTANT RULES=====================

🡺We cannot create the object for the Abstract class

🡺But we can create Object Reference Variable For Abstract Class

QNo2.Can we create Object Reference Variable for the Abstract Class.

-For Abstract class we cannot create object but we can create Object Reference Variable.

EX:-

Abstract class P1

{

Abstract public void test();

}

Class Demo

{

Public static void main(String args[])

{

P1 obj =new P1();**//Not Possible**

P1 obj1;**//Possible**

}

}

**CONCREATE CLASS**

-Class consist of Only Concrete Methods and if Object creation is Possible such class are called as **Concrete Class**.

#Program

abstract class P1 //Abstract Class

{ //abstract method

Abstract public void test();

}

Class P2 extends P1

{

//Concrete Method

Public void test()

{

System.out.println(“Test()-Class-P2”);

}

}

Class Demo

{

Public static void main(String ags[])

{

**//Object Creation for Child Class P2**

P2 oP2=new P2();

**//Method Overriding**

oP2.test();

**//Upcasting From P2 Class to P1 class**

P1 oP1=oP2;

oP1.test();

}

}

Class P1

Class P1

abstract test();

P1 oP1=oP2;

oP1.test();

**Abstract Class**

**UPCASTING**

**Extends**

Test

{

Sop(“Test()-Class-P2”);

}

**Concrete Class**

P2 oP2=new oP2();

oP2.test()

**===================IMPORTANT NOTE 2=================**

If a Parent class consist of **Abstract Method** and if we are **Not Overriding** the Abstract Method from its Child Class then it is **Mandatory** to Make **Child Class** as an **Abstract Class.**

Then the Child Class it will start behaving like **Abstract Class.**

**#PROGRAM**

Abstract class P1

{

//Abstract Method

abstract public void test();

}

Class P2 extends P1

{

//Concrete Method

Public void key()

{

System.out.println(“Key()-Class-P2”);

}

}

Class Demo

{

Public static void main(String args[])

{

P2 0P2=new P2();

}

}

Class P1

Class P2

abstract test();

P1 oP1=oP2;

oP1.test();

**Abstract Class**

**UPCASTING**

**Extends**

Key()

{

Sop(“Key()-Class-P2”);

}

**Abstract Class**

P2 oP2=new oP2();

oP2.test();

**===================IMPORTANT NOTE 2=================**

If a Parent class consist of **Abstract Method** and if we are **Overriding the abstract Method** from Its Child Class then it is **Not Mandatory** to make Child Class as **Abstract Class.**

**#Program**

abstract class P1 //Abstract Class

{ //abstract method

Abstract public void test();

}

Class P2 extends P1

{

//Concrete Method

Public void test()

{

System.out.println(“Test()-Class-P2”);

}

}

Class Demo

{

Public static void main(String ags[])

{

**//Object Creation for Child Class P2**

P2 oP2=new P2();

**//Method Overriding**

oP2.test();

**//Upcasting From P2 Class to P1 class**

P1 oP1=oP2;

oP1.test();

}

}

Class P1

Class P2

abstract test();

P1 oP1=oP2;

oP1.test();

**Abstract Class**

**UPCASTING**

**Extends**

Test

{

Sop(“Test()-Class-P2”);

}

**Concrete Class**

P2 oP2=new oP2();

oP2.test();

QNo1.Write Difference between Abstract Class and concrete Class

QNo2.Write a Difference between Abstract Method and Concrete Method

**=========== IMPORTANT ABOUT ABSTARCT ==========**

1]We can write the Static Members [Static Variable, Static Method, Static Block ]inside the Abstract Class.

2]We can write the Non- Static Members [Non -Static Variable, Non -Static Method, Non -Static Block ]inside the Abstract Class.

3]We can use the Non-Static block for execution for that you have create the Object For the Child Class and Child Class Should be Concrete Class.

4]We can also write constructor both 1]No-Argument Constructor 2]Parameterized Constructor in the Abstract Class but we cannot create the object for the Abstract class for execution that constructor we have to create the child class constructor (if programmer fails to declare the No-Argument Constructor) then compiler automatically adds the default constructor and inside the default constructor the first statement (By Default) is super()🡺it calls the parent class constructor so the parent class No-argument constructor get executed.

**INTERFACE**

-Interface it is a blue print of the class

-With the help of interface, we can perform **100% Abstraction.**

-In Interface if we declare a Non-Static Method the Compiler Automatically Converts the Non-Static Method into **abstract** **Public** **Non-Static** **Method**.

-In interface Abstract keyword and Public Modifier it is Not Mandatory because The Internal Nature Of Interface Is Abstraction Process.

-Interface is inbuild in Abstract.

-So we does not required to make the interface as an Abstract.

**STRUCTURE OF INTERFACE: -**

**interface** **Program\_Name**(**interface\_name**)

**{**

**Members**;

**}**

**Ex: -**

**Interface Program1**

**{**

**}**

**Members Of Interface**

**1]Abstract Non-Static Method**

**2]Static Concrete Method [Static Method]**

**3]Static Final Variable.**

**1]Abstract Non-Static Method**

**Ex: -**

**interface**

**{**

**//abstract method (Ns-Static)**

**Abstract public void demo();**

**}**

**#Program**

**Interface P1**

**{**

**//Ns -Method-Header Without Abstract And Access Modifier**

**void demo();**

**int demo(int a);**

**}**

**to**

**interface P1**

**{**

**//Compiler Adds abstract and Access Modifier for NS-Method**

**abstract public void demo();**

**abstract public void demo(int a);**

**}**

**2]Static Concrete Method [Static Method]**

-If we declare static method inside interface without any Modifier the Compiler Automatically Converts the Static Method Into Public Static Concrete Method.

-Here the public modifier is added Automictically.

**#Program**

interface P1

{

//static Method without Access Modifier

Static void demo()

{

System.out.println(“Static Method Demo()”);

}

}

to

interface P1

{

//Compiler Adds Modifier Automatically Static Method(Concrete)

Public static void demo()

{

System.out.println(“Static Method Demo”);

}

}

**3]Static Final Variable.**

-Inside interface if we declare a static variable the compiler Automictically Converts the static variable into Final Static Variable.

-Here Final Keyword is added Automatically

**Final**

-final Keyword it is a Modifier

-If we Prefix any Variable with the final Keyword the value present inside the variable becomes Constant and we Cannot Modify the value Present Inside the final Variable.

**#PROGRAM**

Interface P1

{

//Static Variable (Global)

Static int n;

}

To

Interface P1

{

//Static Variable (Global)

final static int n=10;

}

Final static int a=10;

**a**

P1

**~~0~~ 10**

final

(Constant)

-In interface the compiler does make the default value is final hence it is Mandatory for programmer to Pass a Value.

**=================IMPORTANT NOTE ON INTERFACE==================**

-In interface if we declare a **Non-Static Variable** the **Compiler** **Automatically** Converts the **Non-Static Variable** Into **final Static Variable.**

**#Program**

Interface P1

{

//NS-Variable

Int n=100;

}

to

Interface P1

{

//final Static-Variable

Final static int n=100;

}

QNo1.Can we Create Object For interface

Ans: - No we cannot create Object for Interface but we create Object Reference Variable.

#Program

Interface P1

{

Public static void main(String args[])

{

P1 obj=new P1()//CTE

P1 obj1;//Possible;

}

}

QNo.2 Can we Access the Static method in the Interface & How?

-Yes we Access the static method in the interface By using the **Interface Name (Same As the Class Name As the reference).**

**=================== NOTE ===================**

-We can also Access the Interface Static Variables using the Interface Same as the class Name

-We can also access the Non-Static Variable because every variable in the interface is **static final** in Nature so the if you are declare the non static variable in the interface the compiler automatically makes it static final in Nature.

-So we can Access the Non-Static variable using Interface

**#Program**

interface A

{

static int a=20;

static void demo()

{

System.out.println("This Is Static Method");

}

}

interface B

{

abstract void display();

}

class MyClass implements A,B

{

public static void main(String[] args)

{

MyClass oM=new MyClass();

A.demo();//Using Interface Name you can Access the Static Method

oM.display();

System.out.println(A.a);

}

public void display()

{

System.out.println("This is override Method");

}

}

**=================== NOTE ===================**

-After JDK 1.7 We can write the method implementation in Interface.

-Before JDK 1.7 we cannot write the method implementation in Interface.

-Since Java8 we can have Concrete Methods as well inside the Interface.

-For That we need to use the **default** Keyword.

**default keyword**

-default is the keyword in java.

-Interface contains the method behaviour public static abstract.

-For breaking that behaviour the default keyword is used.

-default breaks behaviour in interface methods it makes the behaviour as the by default.

-Once you prefixed default keyword with method its behave like normal keyword.

**default Methods**

-The methods come with body(with implementation) is called as default method.

-The default methods always prefix with **default** keyword.

-So we does not need to the override the method in the child class.

**#Program**

interface Interface

{

int a=10;// You Cannot Change the Value In the Interface.

abstract void demo();

//default method

**default** void display()//Default Is keyword In Java

{

System.out.println("This is Display Method");

}

}

class InterfaceMain implements Interface

{

public void demo()

{

System.out.println("This is Demo Method");

}

public static void main(String[] args)

{

InterfaceMain oI=new InterfaceMain();

oI.demo();

oI.display();

//For Child Class Object Reference variable you can Access The Interfaces Implemented Method

//We cannot modify interface variable value

// System.out.println(oI.a=20); //CTE

}

}

**=================== NOTE ===================**

-If both interface contains the same method header and we call using the child class

-Object Reference then the compiler confuse which method to call so it will throw an Compile time error called

(MyClass.java:17: error: class MyClass inherits unrelated defaults for display() from types A and B

//class MyClass implements A,B

//^

**#Program**

interface A

{

default void display()

{

System.out.println("A Says Hello");

}

}

interface B

{

default void display()

{

System.out.println("B Says Hello");

//System.out.println(a);

}

}

class MyClass implements A,B

{

public static void main(String[] args)

{

MyClass oM=new MyClass();

oM.display();

}

}

**To Avoid:**

**-To avoid the that error there is following way**

**1]1st way to change the Method Header[Method Name] in one Interface so compiler didn’t confuse to call method. It is familiar to differ the method name.**

-In the first way we change the method name so compiler didn’t confuse which method to call.

**2]2nd way change the Method Signature[Formal Arguments].**

-In second way we need change the method signature means pass the different types of arguments so the both interface contains different method signature.

**3]3rd way is Override the Method In Child Class.**

-In third way we override this method In the child class so the compiler executed the child class method implementation.

**=========================== 1st Way ==========================**

**#Program**

interface P1

{

default void display()

{

System.out.println("P1 Says Hello");

}

}

interface P2

{

default void demo()

{

System.out.println("P2 Says Hello");

//System.out.println(a);

}

}

class Main implements P1,P2

{

public static void main(String[] args)

{

Main oM=new Main();

oM.display();

}

}

**========================== 2nd Way ==========================**

**#Program**

interface One

{

default void display()

{

System.out.println("One Says Hello");

}

}

interface Two

{

default void display(int a)

{

System.out.println("Two Says Hello");

System.out.println(a);

}

}

class Demo implements One,Two

{

public static void main(String[] args)

{

Demo oD=new Demo();

oD.display();

oD.display(15);

}

}

**======================= OUTPUT =====================**

One Says Hello

Two Says Hello

15

Press any key to continue . . .

**========================== 3rd Way ==========================**

**#Program**

interface Amol

{

default void display()

{

System.out.println("One Says Hello");

}

}

interface Ganesh

{

default void display(int a)

{

System.out.println("Two Says Hello");

System.out.println(a);

}

}

class Driver implements Amol,Ganesh

{

public static void main(String[] args)

{

Driver oD=new Driver();

oD.display();

}

public void display()

{

System.out.println("This Is Overide Method");

}

}

**======================= OUTPUT =====================**

This Is Overide Method

Press any key to continue . . .

**INHERITANCE USING INTERFACE**

-With the help of extends Keywords we can perform Inheritance using interface

#Program

interface P1

{

}

interface P2 extends P1

**1]Single-Level Inheritance**

**Interface P1**

**EXTENDS**

**Interface P2**

**2]Multi-Level Inheritance**

**Interface P1**

Interface P3 extends P2

**EXTENDS**

**Interface P2**

Interface P3 extends P2

**EXTENDS**

**Interface P3**

Interface P3 extends P2

**3]Hierarchical Inheritance**

**Interface P1**

**EXTENDS**

**Interface P2**

**Interface P3**

Interface P2 extends P1 Interface P3 extends P1

**4]Multiple Inheritance**

**Interface P2**

**Interface P1**

**EXTENDS**

**Interface P1**

-With the help of Interface we can perform **Multiple Inheritance** and Solved **Diamond Problem** because Interface does not Support **Constructors**.

-There Is no Concept of Constructor in Interface So we cannot perform Constructor Chaining Hence with the help of interface we perform Multiple Inheritance In java.

#Program

Interface P1

{

}

Interface P2

{

}

Interface P3 extends P1,P2

{

}

**5]Hybrid Inheritance**

Interface P3

Interface P1

Interface P2

**EXTENDS**

**EXTENDS**

Interface P3

**INHERITANCE USING CLASS AND INTERFACE**

-With the help Extends Keyword and Implements Keyword we can perform Inheritance between Class and Interface.

#Program

Interface P1

{

}

Class P2 implements P1

{

}

**EXAMPLE NO: - 1**

**#Program 1**

Interface P1

**implements**

Class P2

**EXAMPLE NO:- 2**

**#Program 2**

Interface P1

{

}

Interface P2

{

}

Class P3 implements P1,P2

{

}

Interface P2

Interface P1

**Implements**

Class P3

**EXAMPLE NO: - 3**

**#Program 3**

Class P1

{

}

Interface P2

{

}

Class P3 extends P1 implements P2

{

}

Interface P2

Class P1

**Extends** **Implements**

Class P3

interface P1

{

void test ();

}

interface P2

{

void demo ();

}

Class P3 implements P1,P2

{

Public void test()

{

System.out.println(“Test()-P3”);

}

Public void demo()

{

System.out.println(“Demo()-P3);

}

Public static void main(String[] args)

{

P3 obj =new P3();

obj.test();

obj.demo();

System.out.println();

System.out.println(“Upcasting From P3 To P1”);

//Upcasting From P3 to P1

P1 obj1=obj;

obj1.test();//Method Overriding

System.out.println();

System.out.println(“Upcasting From P3 to P2”);

P2 obj2=obj;

Obj2.demo();//method overriding

}

}

Interface P2

interface P1

{

Void demo();

}

{

Void test();

}

**Implements** **Implements**

Class P3

{

P3 obj =new P3();

//Upcasting From P3 to P1

P1 obj1=obj;

obj1.test();//Method Overriding

System.out.println(“Upcasting From P3 to P2”);

P2 obj2=obj;

obj2.demo();//method overriding

}

Interface does not inherit object class

**OBJECT CLASS**

Interface P2

Class P1

P1

{

Super();

}

**NO CONSTRUCTOR**

**Extends** **Implements**

Class P3

P3

{

Super();

}

Implements Parent I I I

Extends Child I CC AC

I CC AC AC CC AC

AC CC CC AC I I

CC

QNo.1 Design a Realtime example for all the types of inheritance using interface, abstract Class, Concrete Class

Ans:-

**1]Single-Level Inheritance**

interface Student

{

void data();

}

class Amol implements Student

{

String name="Amol Pawar";

int rollno=24;

public void data()

{

System.out.println(name);

System.out.println(rollno);

}

}

class Program1

{

public static void main(String[] args)

{

Amol oA=new Amol();

oA.data();

}

}

====================================OUTPUT====================================

Amol Pawar

24

Press any key to continue . . .

**2]Multi-Level Inheritance**

interface Bank

{

void loan(double loan,float time);

void account();

void deposite(double d);

void withdraw(double w);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

============================= SBI BANK ================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

abstract class Sbi implements Bank

{

String name;

int age;

String Add;

String phoneNo;

String AccType;

double loAmm;

double Ammount;

double deposite;

double withdraw;

static int rate=7;

static

{

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

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WELCOME TO SBI BANK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

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

}

public void details(String strName,int age,String PNo,String Add,String acctype)

{

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

System.out.println(" Please Fill Details ");

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

System.out.println("Enter Your Name ----");

System.out.println("Enter Your Age ----");

System.out.println("Enter Your Phone Number ----");

System.out.println("Enter Your Address ----");

System.out.println("Enter Account Type ----");

this.name=strName;

this.age=age;

this.phoneNo=PNo;

this.Add=Add;

this.AccType=acctype;

}

public void account()

{

this.details("Amol",24,"9373428581","Pandharpur","Saving Account");

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

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\* Account Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*");

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

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

System.out.println("Account Holder Age :"+age);

System.out.println("Account Holder Phone Number :"+phoneNo);

System.out.println("Account Holder Address :"+Add);

System.out.println("Account Type :"+AccType);

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

System.out.println(" Your ==>==> "+AccType+" ==>==> Is Created ");

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

System.out.println();

}

public void deposite(double d)

{

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

this.deposite=d;

System.out.println("Your Deposite Ammount Is: "+deposite);

Ammount+=d;

System.out.println("Your Account Balance Is: "+Ammount);

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

System.out.println();

}

double Si;

public void loan(double PA,float time)

{

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

System.out.println("Your Loan Ammount Is :"+PA);

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

Si=(PA\*rate\*time)/100;

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

System.out.println("Your Interest Ammount Is :"+Si);

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

}

}

class SbiAtm extends Sbi

{

public void withdraw(double withdraw)

{

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

Ammount=Ammount-withdraw;

System.out.println("Your withdraw Ammount Is: "+withdraw);

System.out.println("Your Account Balance Is: "+Ammount);

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

System.out.println(" ==> ==> Thanks For Using SBI BANK SERVICE ==> ==> ");

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

}

}

class Program2

{

public static void main(String[] args)

{

//SbiAtm oSA=new SbiAtm();

//Upcating From Dev3 To Interface Bank

//Bank oB=oSA;

AxisAtm oAA=new AxisAtm();

/\*oAA.account();

oAA.deposite(10000);

oAA.withdraw(500);

oAA.loan(100000,30);\*/

Bank oB=oAA;

oB.account();

oB.deposite(1000);

oB.withdraw(500);

}

}

**3]Hierarchical Inheritance**

//Hierarchical Inheritance

interface Vehicles

{

//abstract methods

void changeGear(int a);

void speedUp(int a);

void applyBreakes(int a);

}

class Bicycle implements Vehicles

{

int speed;

int gear;

public void changeGear(int newGear)

{

gear=newGear;

}

public void speedUp(int increment)

{

speed=speed+increment;

}

public void applyBreakes(int decrement)

{

speed=speed-decrement;

}

public void printStates()

{

System.out.println("Speed : "+speed+ "gear : "+gear);

}

}

class Bike implements Vehicles

{

int speed;

int gear;

public void changeGear(int newGear)

{

gear=newGear;

}

public void speedUp(int increment)

{

speed=speed+increment;

}

public void applyBreakes(int decrement)

{

speed=speed-decrement;

}

public void printStates()

{

System.out.println("Speed : "+speed+ "gear : "+gear);

}

}

class Program3

{

public static void main(String[] args)

{

//Creating Instance For Class Bicycle

Bicycle oBic=new Bicycle();

oBic.changeGear(2);

oBic.speedUp(3);

oBic.applyBreakes(1);

System.out.println("Bicycle Present State :");

oBic.printStates();

//Creating Instance For Class Bike

Bike oBik=new Bike();

oBik.changeGear(2);

oBik.speedUp(4);

oBik.applyBreakes(3);

System.out.println("Bike Present State :");

oBic.printStates();

}

}

**4]Multiple Inheritance**

interface QSpiders

{

void testing();

}

interface JSpiders

{

void developing();

}

class TestYantra

{

void working()

{

System.out.println("Employee Is Working");

}

}

class Employee extends TestYantra implements QSpiders,JSpiders

{

public void testing()

{

System.out.println("Employee Is Testing The Software");

}

public void developing()

{

System.out.println("Employee Is Developing the Software");

}

}

class Program4

{

public static void main(String args[])

{

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

Employee oE=new Employee();

oE.working();

Employee Dev=new Employee();

Dev.developing();

Employee Test=new Employee();

Test.testing();

}

}

**5]Hybrid Inheritance**

class Student

{

String name="Amol";

int rollno=24;

}

interface Project

{

int pmark=200;

}

abstract class Exam extends Student

{

int iSub1=86,iSub2=87,iSub=85,iSub3=75,iSub4=97;

int theory=iSub1+iSub2+iSub3+iSub4;

}

class Result extends Exam implements Project

{

int Total=theory+pmark;

double Per=((Total\*100)/600);

public void display()

{

System.out.println(Per);

}

}

class Program5

{

public static void main(String[] args)

{

Result oR=new Result();

oR.display();

}

}

QNo2.Design two Realtime example for hybrid and multiple Inheritance

**Ans : -**

**1]Multiple Inheritance**

class Frontend

{

public void responsive(String str1,String str2)

{

System.out.println(str1+" & "+str2+" Is used to Backend Language");

}

}

interface Backend

{

//abstarct class

public void connServer();

}

class Language extends Frontend implements Backend

{

String strLang;

public void print(String l)

{

this.strLang=l;

}

public void connServer()

{

this.print("Php");

System.out.println(strLang+" Is used to Backend Language");

}

}

class MultipleInheri

{

public static void main(String[] args)

{

Language oL=new Language();

oL.connServer();

oL.responsive("HTML","CSS");

}

}

**2]Hybrid Inheritance**

**Ans:-**

QNo3.Write a difference between Concrete class Abstract Class and Interface

|  |  |  |  |
| --- | --- | --- | --- |
| SrNo | Concrete  Class | Abstract  Class | Interface |
| 1 | A concrete class is declared using class. | An abstract class is declared using abstract modifier. | An interface class is declared using interface keyword. |
| 2. | A concrete class can be directly instantiated using the new keyword. | An abstract class cannot be directly instantiated using the new keyword. | An Interface class cannot be directly instantiated using the new keyword. |
| 3. | A Concrete class cannot contain an abstract method. | A Abstract class can contain an abstract method. | A Interface class can contain an abstract method. |
| 4. | A concrete class can be declared as final. | An abstract class cannot be declared as final. | An Interface class cannot be declared as final. |
| 5. | Concrete class can **have only Concrete** methods. | Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. |
| 6. | Concrete class can have **final, non-final, static and non-static variables**. | Abstract class can have **final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 7 |  | A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 8. | An **Concrete class** can be extended using keyword "extends". | An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 9. | An **Concrete class** can extend another Java class and implement multiple Java interfaces. | An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 10. | Static & Nonstatic members are allowed. | Static members are allowed. | Variables are not allowed in interface. Hence any data declaration is ‘ public static final ’; |

QNo4, Design a Realtime example for UpCasting, DownCasting and Method Overriding Using Abstraction and interface

Ans:-

interface Mobile

{

void call();

void sms();

void clickPhoto();

void PlayGames();

void watchMovies();

}

abstract class Xiomi implements Mobile

{

public void call()

{

System.out.println("You can do Call");

}

public void sms()

{

System.out.println("You can do SMS");

}

public void clickPhoto()

{

System.out.println("You can Click Better Photos");

}

public void watchMovies()

{

System.out.println("You can Watch Movies");

}

}

class Poco extends Xiomi

{

public void PlayGames()

{

System.out.println("It Is a Gaming Phone");

}

}

class Program6

{

public static void main(String[] args)

{

Poco oP=new Poco();

Mobile oM=oP;

oP.call();

oP.PlayGames();

}

}

QNo5.Design a Realtime Example for Encapsulation using Inheritance and Polymorphism.

Ans:-

interface Vehicle

{

void run();

}

class Bike implements Vehicle

{

private int cc;

private String brand;

int speed;

Bike(int cc,String brand,int s){

this.cc=cc;

this.brand=brand;

this.speed=s;

}

public int getCc(){

return this.cc;//---directly---static cc----non static

}

public void setCc(int cc){

this.cc=cc;

}

public void run()

{

System.out.println("Bike Is Running on Speed "+speed+"Km");

}

}

class Encapsulation

{

public static void main(String[] args){

Bike b1=new Bike(150,"Hero",100);

//System.out.println(b1.cc);

;;;;;;;;;;;;

b1.setCc(230);

System.out.println(b1.getCc());

//230

Vehicle oV=b1;

oV.run();

}

}

* **OBJECT CLASS**

-Object Class It is a **Super Most** Class in Java.

-Object Class It is an **Inbuilt Class**.

-Every Class Automatically extends [Inherits] **Object Class** in java.

-Object Class It is An **Concrete Class** In Java.

-We Can create an **Object** for Object Class

-Object class it is Present inside **Java.lang** Package[Folder].

-The Fully Qualified Name of Object Class is **Java.lang.Object[Address Of the Class]**

#Program

Class P1

{

Public static void main(String args[])

{

Object obj=new Object();

System.out.println(obj);

}

}

**Package Name**

**JAVA**

**JAVA**

**lang**

**io**

**lang**

**Class Object**

**{**

**Members;**

**}**

**sql**

**util**

**===============OBJECT CLASS METHODS===============**

1.public string toString()

2.public int hashcode()

3.public boolean equals(Object obj)

4.protected void finalize()

5.protected object clone()

6.public final void wait(long miliSeconds)

7.public final void wait(long int)

8.public final void notifyAll()

9.public final void notify()

10.public final class getClass()

11.public final void wait(long milisecond,int time)

**1.public String toString ();**

-The toString() method it used to convert the address of the object into string format.

-It is a Non-Static Method which is Present inside the object class.

-toString() method it is implicitly(Automatically)called by the Compiler.

-The ReturnType of the toString method is String.

-After Overriding the toString() method it will return our own address in String format.

**Method Header of toString Method.**

**Public String toString();**

#Program

Class P1

{

Public static void main(String args[])

{

P1 obj=new P1();

System.out.println(obj); toString() called by implicitly

System.out.println(obj.toString());toString() called by explicitly

}

}

**2.public int hashCode();**

-hashCode() Method it is used to convert the address into the integer format.

-It is Non-Static Method which is Present inside the Object Class

-The Return Type Of hashCode() Method is int.

-The hashcode method is not called by compiler implicitly(Automatically).

-After Overriding the hashCode() method it will return the properties in Integer format.

**The Method Header Of HashCode Method is:-**

**public int hashCode();**

#Program

Class P1

{

Public static void main(String args[])

{

P1 obj=new P1();

System.out.println(obj);

System.out.println(obj.hashCode());toString() called by explicitly

}

}

QNo.1Overriding toString Method of Object class.

Ans:-

-By overriding toString() method of object class we can create our own Address.

#Program

Class Emp

{

//Ns-Variable

Int Eid;

Emp(int n)

{

this.Eid=n;

}

Public String toString()

{

Return “TCS”+Eid;

}

}

Class Company

{

Public static void main(String args[])

{

Emp obj=new Emp(100);

System.out.println(obj);

Emp obj=new Emp(101);

System.out.println(obj);

}

}

**OBJECT**

INSTANCE OF OBJECT CLASS IS CREATED

**EMP**

****

INSTANCE OF EMP CLASS IS CREATED

~~0~~ 100

Emp obj=new Emp (100);

Sop(obj);

QNo1.Can we override hashcode of object class

Ans:-Yes we can override the hashcode() method of Object class such that we can create our own Address in integer format.

QNo1.Design two Realtime example to override the hashcode and toString method of object class.

**3.public boolean equals();**

-equals() method it is used to compare the address or the Reference of two different object.

-It is a Non-Static method present inside the Object class.

-The ReturnType of the equals method is a boolean condition[true/false]

-The equals(Object o) method it is an Parameterized Method which accept the Address or References of the classes.

-So passing the Address or Reference from child class to Parent Class**[UPCASTING]** of the class is Mandatory.[Automatically Happens the Upcasting]

-In order to pass address of the child class to Parent class Calling of the equals(Object o) method and Passing Value is Mandatory.

-After Overriding the equals() method the equals method will compare the properties of two different object.

**Method Header of Equals Method**

**public boolean equals(Object o);**

**Class Name Variable Name**

**-equals(object var)**

**We can store the address of**

**Object reference Variable a Class (Object)**

**Of the Object class**

**#Program**

Class P1

{

Public static void main(String args[])

{

P1 obj=new P1();

P1 obj1=P1();

System.out.println(obj);

System.out.println(obj1);

System.out.println(obj.equals(obj1));//false

}

}

**#Program**

Class P2

{

Public static void main(String args[])

{

P2 obj=new P2();

P2 obj1=obj;

System.out.println(obj);

System.out.println(obj1);

System.out.println(obj.equals(obj1));

}

}

**O/P**

**🡺P2@123**

**🡺P2@123**

**🡺true**

**QNo1:Can we override equals() method of the object class**

**Ans:-**Yes we can override equals() method of Object class such that we can compare the members of Two Different object.

**#Program**

Class Emp

{

Int Eid;

Emp(int n)

{

This.Eid=n;

}

Public boolean equals(Object var)

{

boolean n=false;

if(this.Eid==((Emp)var).Eid)

{

N=true;

}

Return n;

}

}

Class Company

{

Public static void main(String args[])

{

Emp e1=new Emp();

Emp e2=new Emp();

System.out.println(e1);

System.out.println(e2);

System.out.println(e1.equals(e2));

}

}

e2@123

e1@123

100

Int Eid

Public boolean equals(Object var)

{

boolean n=false;

if(this.Eid==((Emp)var).Eid)

{

N=true;

}

Return n;

}

EMP

Int Eid

Public boolean equals(Object var)

{

boolean n=false;

if(this.Eid==((Emp)var).Eid)

{

N=true;

}

Return n;

}

100

EMP

OBJECT CLASS

//NS-METHODS LOADED

OBJECT CLASS

//NS-METHODS LOADED

QNo1.Create Two real Time Example to Override The Equals Method of Object class by tracing with Explanation

QNo2.Create Realtime Example to Override equals Method Of Object class by comparing two Variable.

**WRAPPER CLASSES: -**

-With the help of wrapper classes, we can convert Primitive into Non-Primitive Datatype.

|  |  |  |  |
| --- | --- | --- | --- |
| SRNO | Primitive  Values | Primitive Datatype | Size |
|  |  | byte | 1 Byte |
| 1 | Integer | short | 2Byte |
|  |  | int | 4 Byte |
|  |  | long | 8 Byte |
|  |  |  |  |
| 2 | Float | float | 4 Byte |
|  |  | double | 8 Byte |
|  |  |  |  |
| 3 | Character | char | 2Byte |
|  |  |  |  |
| 4 | Boolean | boolean | 1 Bit |

|  |  |  |
| --- | --- | --- |
| SrNo | Non-Primitive  Datatypes | Size |
| 1 | String |  |
| 2 | Array |  |
| 3 | Class |  |

|  |  |  |
| --- | --- | --- |
| SRNO | Primitive Datatypes | In-Built Classes |
| 1 | byte | Byte |
| 2 | short | Short |
| 3 | int | Integer |
| 4 | long | Long |
| 5 | float | Float |
| 6 | double | Double |
| 7 | char | Character |
| 8 | boolean | Boolean |

-All the Wrapper classes are inbuild classes in java

-All the Wrapper classes are Present inside the **java.lang** Package.

-The Fully Qualified Name of Wrapper Class is **java.lang.ClassName**

-All the Wrapper classes are Final in Java.

#Program

final class Byte

{

//members of the class(final)

}

-All the wrapper classes Extends Object Class and implements two Interface.

1]Comparable Interface

2]Serializable Interface

**SERIALIZABLE**

**INTERFACE**

**COMPARABLE**

**INTERFACE**

**OBJECT CLASS**

**Extends**

**implements implements**

Wrapper Classes

final Byte/

final Short/

final Double/

final Long/

final Float/

final Double/

final Character/

final Boolean

-For all the Wrapper classes we can Create The Object in two Ways:-

1]Using Assignment Operator (=)

2]Using New Keyword

#Program

class P1

{

public static void main(String[] args)

{

Integer obj=100;

System.out.println(obj);//100

Integer obj1=new Integer(100);//It is Not Possible After JDK 8

System.out.println(obj1);//100

}

}

**BOXING**

-Boxing it is process of converting the **Primitive Datatype** into a **Non-Primitive Datatype.**

**Primitive Conversion Non-Primitive**

**Datatype Datatypes**

**byte**

**Byte**

**Short**

**short**

**Integer**

**int**

**Long**

**long**

**Float**

**float**

**Double**

**double**

**Boolean**

**boolean**

**Character**

**char**

#Program

Class P1

{

Public static void main(String args[])

{

int n=100;

Integer obj=Integer.valueOf(n);

System.out.println(n);

System.out.println(obj);

boolean b=true;

Boolean obj1=Boolean.valueOf(b);

System.out.println(n);

System.out.println(obj1);

}

}

-We can perform Boxing Process with the help of valueOf() method which is present inside all the wrapper classes.

-It Is an Static Method which is present inside in All the Wrapper Classes.

-We can Access the valueOf() method with the help of Class Name As Reference.

QNo1.Converting All the Primitive Datatype Into String[Non-Primitive]

Ans: -We can convert All the Primitive Datatype into String with the help of

**valueOf()** method.

-Value Of Method it is an **Static Method** which is Present inside String Class.

-We can access the valueOf() method with the help of Class Name As Reference.

-We cannot perform Auto Boxing Process for Converting to Primitive Datatype into String

**String**

**byte**

**short**

**int**

valueOf ()

**long**

**float**

**double**

**boolean**

**char**

#Program

Class P4

{

Public static void main(String args[])

{

int n=100;

String obj=String.valueOf(n);//Method Overloading for String

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

System.out.println(obj);

boolean b=true;

String obj1=String.valueOf(b); //Method Overloading for String

System.out.println(b); // Class

System.out.println(obj1);

double d=11.35;

String obj2=String.valueOf(d); //Method Overloading for String

System.out.println(d); // Class

System.out.println(obj2);

}

}

**AUTO-BOXING**

-From **JDK 1.5** version onwards we perform Boxing process Automatically.

#Program

Class P1

{

Public static void main(String args[])

{

int n=100;

Integer obj=n;//AutoBoxing

System.out.println(n);

System.out.println(obj);

boolean b=true;

Boolean obj1=b;//AutoBoxing

System.out.println(n);

System.out.println(obj1);

}

}

**UNBOXING**

-Unboxing it is the process of converting the **Non-Primitive Datatype** into **Primitive Datatype**.

**Non-Primitive Conversion Primitive**

**Datatype Datatypes**

**byte**

**Byte**

**short**

**Short**

**int**

**Integer**

**long**

**Long**

**float**

**Float**

**double**

**Double**

**boolean**

**Boolean**

**char**

**Character**

-We can perform the Unboxing Process with the help of Non-Static Method which is Present inside all the Wrapper Classes.

#Program

Class P5

{

Public static void main(String args[]))

{

Integer obj=123;

int n=obj.intValue();

System.out.println(obj);

System.out.println(n);

Boolean obj1=false;

boolean n1=obj1.booleanValue();

System.out.println(obj1)

System.out.println(n1);

}

}

**AUTO-UNBOXING**

-From **JDK 1.5** Version Onwards we can perform Unboxing Process Automatically.

#Program

Class P6

{

Public static void main(String args[]))

{

Integer obj=123;

int n=obj;

System.out.println(obj);

System.out.println(n);

Boolean obj1=false;

boolean n1=obj1;

System.out.println(obj1)

System.out.println(n1);

}

}

**CONVERTING STRING TO PRIMITIVE DATAYPE[UNBOXING]**

**String**

**ByteparseByte()**

**byte**

**ShortparseShort()**

**short**

**IntegerparseInt()**

**int**

**LongparseLong()**

**long**

**FloatparseFloat**

**float**

**DoubleparseDouble()**

**double**

**BooleanparseBoolean()**

**boolean**

**CharparseChar()**

**char**

-We can convert String into all the Primitive datatype with the help of Static Method which is present inside the All the wrapper classes.[parseDataype**()**]

-We cannot perform Auto UnBoxing Process for Converting String into Primitive Datatypes.

Class P7

{

Public static void main(String args[])

{

String obj=”100”;

int n=Integer.parseInt(obj);//Depends on Datatype

System.out.println(obj);

System.out.println(n);

String obj1=”true”;

Boolean n1=Boolean.parseBoolean(obj1);

System.out.println(obj1);

System.out.println(n1);

}

}

**NUMBER FROMAT EXCEPTION**

-When we try to convert String into Primitive Datatype the value which is present inside String object if it is Not Similar to its Respective Primitive Datatype we get an Runtime Exception called as Number Format Exception.

Not Same(Exception)

String obj= ;

n=Integer.parseIn(obj);

#Program

Class P8

{

Public static void main(String args[])

{

String obj=”A”;

int n=Integer.parseInt(obj);

System.out.println(obj);

System.out.println(n);

}

}

**charAt()**

#Program

Class P1

{

Public static void main(String args[])

{

String obj=”ABCD”;

Char ch=obj.charAt(2);

System.out.println(obj);

System.out.println(ch);

}

}

===================== **IMPORATANT** ===================

-The toString() method ,the hashCode() method and equals() method of object class it is overridden from all the wrapper classes.Such that it won’t return the Address instead of it will return the Value present inside the object.

-Here Method overriding process it is already done internally

Class P2

{

Public static void main(String args[])

{

Integer obj=new Integer(100);

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

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

Integer obj1=new Integer(100);

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

System.out.println(obj.equals(obj1));

}

}

Integer@100

toString();

hashCode();

equals();

Instance of

Object Class.

toString();

{

}

hashCode();

{

}

equals();

{

}

Instance Of the

Integer Class.

Inbuild Class

Present inside the Java.lang Package

Integer obj=

l.l.I@100

address

Java.lang.Integer@100

100

Object block

SOP(obj.toString()); of Memory

[Instance of the

toString()

{

}

toString()

{

}

Value Integer Class]

100

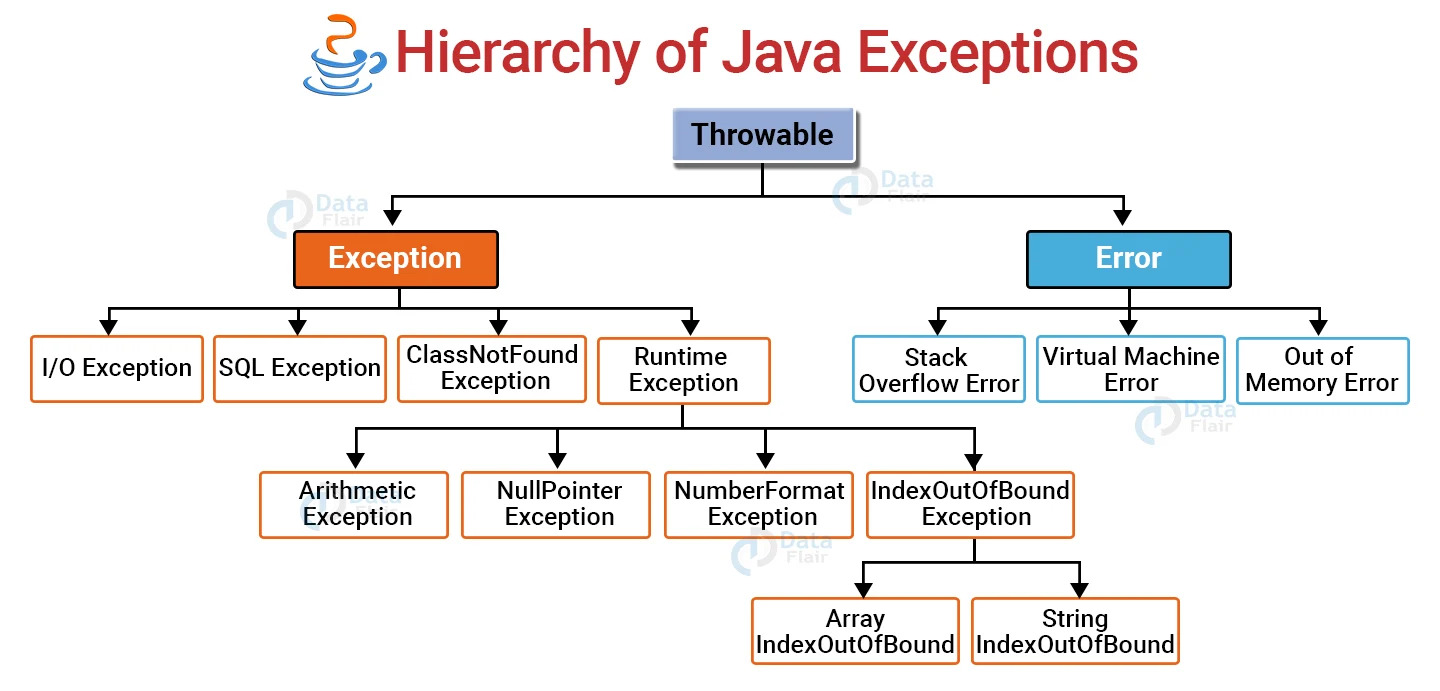
O/P:-

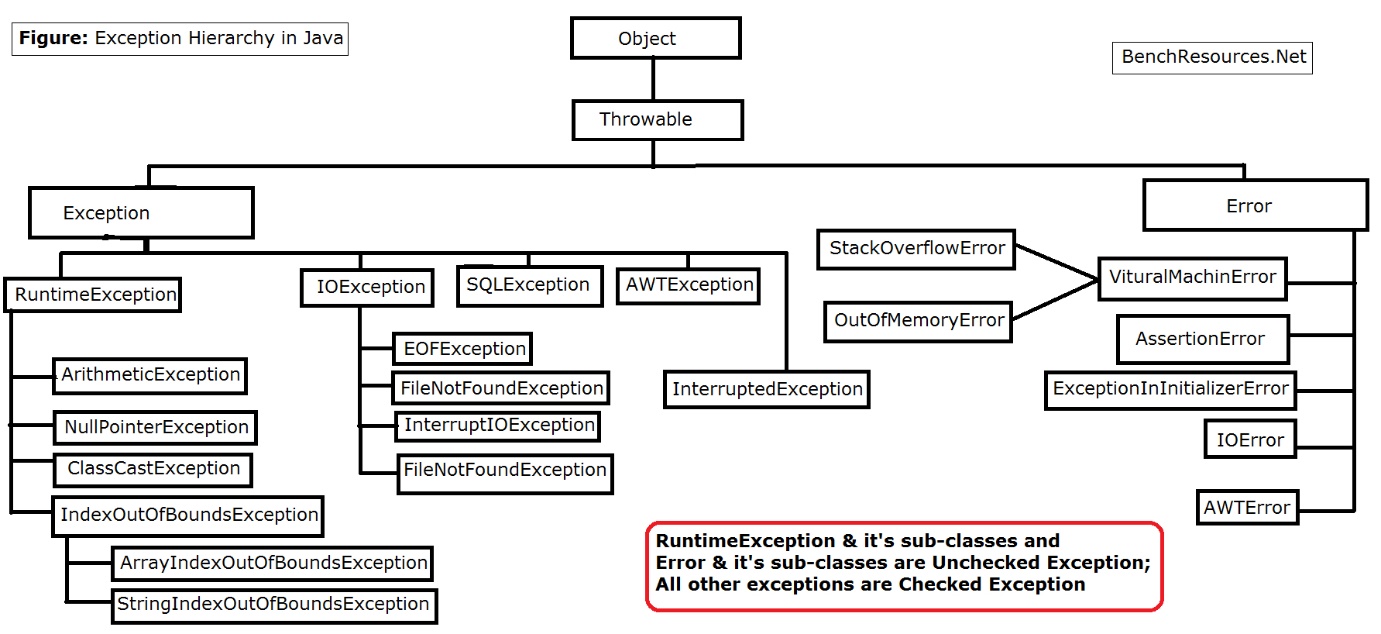
Method Overriding

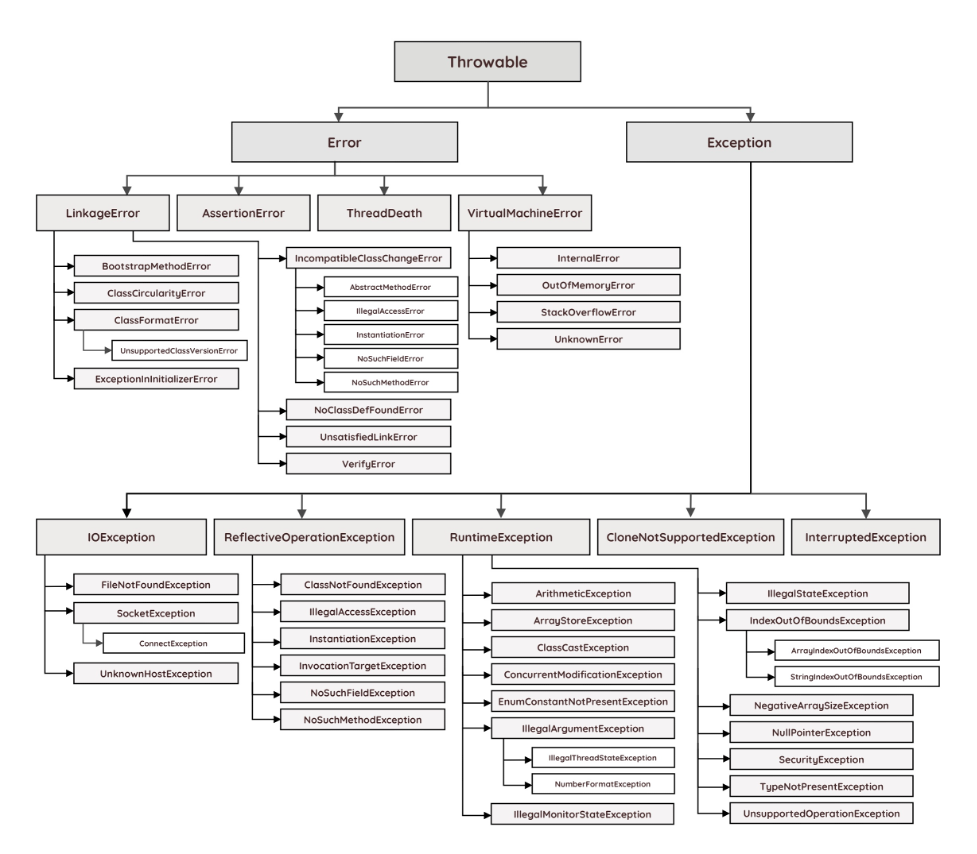
**EXCEPTION**

-Exception it is an Abnormal situation which occurs during the execution the program.

-Whenever an exception occurs the Execution of the program Forcefully Stopped.







**Exception**

Exception it is an abnormal situation which occurs during execution of the program.

Whenever an exception occurs the execution of the program is forcefully stopped.

Whenever an exception occurs two things will take place:

1.it will throw a throwable type of object

Throwable type of object means address of the particular exception class is created.

2.It will check for exception handling mechanism.(It will check the address reference variable to catch the Exception.)

Exception it is classified into two types

1.checked exception.

2.unchecked exception.

**1.Checked Exception:**

Checked exceptions is **compiler aware** exception.

For checked exception declaration and handling the exception is mandatory.

**2.Unchecked Exception:**

unchecked exception is **compiler unaware** exception.

For unchecked exception declaration and handling the exception is not mandatory.

The runtime exception and its subclasses and error its subclasses are unchecked exception.

Object class

Throwable

IndexOutOfBoundsException

StringIndexOutOfBoundsException

ArrayIndexOutOfBoundsExcpetion

NullPointerException

ArithmeticException

SQL Exception

Error

FileNotFoundException

StackOverFlowError

RuntimeException

**getMessage()**

-It is a Non-Static method of Throwable class.

-The ReturnType of this is String value.

-This methos is used to return the reason of that Exception occurred in String format.

**printStackTrace()**

-It is Non-Static method of Throwable class

-It used to print the Back trace of Exception which is occurred.

**Exception Class**

-It is Super class of Exception subclasses.

-These classes partially checked and partially unchecked nature.

-Because of RunTime Exception and its subclasses Exception Class is called unchecked is Nature.

**Runtime Exception:**

It is Subclass of Exception class.

The exception which is occurred during the RunTime such exception is called as RunTime Exception.

All the subclasses of the runtime exception class are unchecked exception.

For all the unchecked exception declaration and handling exception is not mandatory (Only for compile time).

**StringIndexOutOfBoundsException:**

It is an unchecked exception when we try to extract a character from given String based on its index value if the index value is greater than the or equals length of given string, we get unchecked exception called StringIndexOutBoundsException.

**Program:**

Class P1

{

Public static void main(String[] args)

{

String s1=”ABCD”;

System.out.println(s1.charAt(0));

System.out.println(s1.charAt(2));

System.out.println(s1.charAt(5)); //-🡪RuntimeException

}

}

**ArithmeticException:**

It is unchecked exception. When we try to divide any thing by zero we get runtime exception called arithmetic exception.

**Program:**

Class P2{

Public static void main(String[] args){

System.out.println(“Main Start”);

Int a=1;

Int b=0;

Int result=a/b; //--Exception

System.out.println(“Main Ends”);

}

}

**Exception Handling Mechanism:**

The mechanism which is used to handle the exception is called exception handling mechanism.

-We can use the Exception Handling Mechanism for both Checked and unchecked Exception.

Exception Handing mechanism consist of two blocks

**1.try block**

**2.catch block**

**Syntax:**

**Try**

**{**

//The Statement which is responsible to create the Exception.

**}**

**Catch(** ClassName Object\_Reference\_Variable**)**

**{**

//Catch block is responsible to handle the Exception.

**}**

**Try block:**

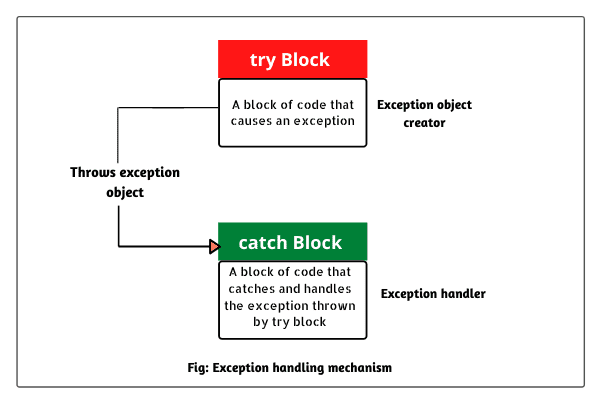
Any statement which is responsible for an **exception** those statement has to be written inside try block.

It will throw a throwable type of object to catch block. (Address of particular exception class)

**Catch block:**

It is used to catch the throwable type of object given by the try block.

If the exception is caught handled the execution of program will continue**.**

****

**Note:**

Catch block is responsible to handle the exception.

Class P1{

Public static void main(String[] args){

System.out.println(“Main start”);

Int n=1;

Int m=0;

Try{

Int rea=n/m;

}

Catch(ArithemeticException e){

System.out.println(e);

}

System.out.println(“Main Ends”);

}

}

n=1

ArithemeticException

Trigger Exception type of class

m=0;

try{

int res=n/m;

} java.lang.ArithematicException

Object

Throwable type of

object

Catch(ArithemeticException e){

sout(e);

Start

}

Execution of program is stopped

Exception Present

No Yes

Checks

Exception Handling Mechanism Present ?

Yes

No

Execution of program is stopped

Execution of program is continued

END

**Try With multiple Catch block:**

A try block consist of more than one catch block is called as try with multiple catch block.

**Syntax:**

try{

Statements;

}

catch(class\_name variable\_name){

Statement;

}

catch(class\_name variable\_name){

Statements;

}

catch(class\_name variable\_name){

Statements;

}

**Program:**

Class P2

{

Public static void main(String[] args)

{

System.out.println(“Main Start”);

Int n=1;

Int m=0;

Try{

Int res=n/m;

}

catch(ClassCastException e){

System.out.println(“1st catch block”);

System.out.println(e);

}

Catch(Exception e){

System.out.println(“2nd catch block”);

System.out.println(e);

}

System.out.println(“Main ends”);

}

}

Try{

Int res=10/1;

} class\_name

It will check for

1.same\_class\_name

2.parent\_class\_name

Catch(ClassCastException e){

}

Catch(Exception e){

Sout(e);

}

Class P1{

Public static void main(String[] args){

System.out.println(“Main Start”);

Int n=1;

In m=0;

Try{

Int res=n/m;

}

Catch(Exception e){

System.out.println(“1st catch block”);

System.out.println(e);

}

Catch(ArithmeticException e ){

System.out.println(“2ed catch block”);

System.out.println(e);

}

System.out.println(“Main ends”);

}

}

Try{

}

Catch(ClassCastException e)

Child class\_name

To

Upcasting

Parent class\_name upcasting

Parent

}

Catch(Exception e){

}

Try{

}

Parent class\_name

To

downcasting

child class\_name upcasting

Parent

Catch(Throwable obj){

}

Catch(Exception obj){

}

**Note:**

The class Name of multiple catch block it should be in child class to parent class order.

If the class name is written in terms of parent class to child we get compile time error.

IndexOutOfBoundsExcpetion

RunTime Exception

Child To Parent Order (Upcasting)

Exception

Throwable

**Nested try catch block:**

**A try catch block is inside another try catch block is called Nested try catch block.**

Try{

Try{

Statements;

}

Catch(class\_name variable\_name){

}

Statements:

}

Catch(class\_name variable\_name){

Statements;

}

**Finally Block:**

Any task which is mandatory for execution even if exception in the program is handled or not handled such task are written inside finally block.

We can use finally block with help of try and catch block.

To use finally block at least try block is mandatory.

**Syntax 1:**

try{

Statements;

}

catch(class\_name variable\_name){

Statements;

}

finally{

Statements;

}

**Syntax 2:**

Try{

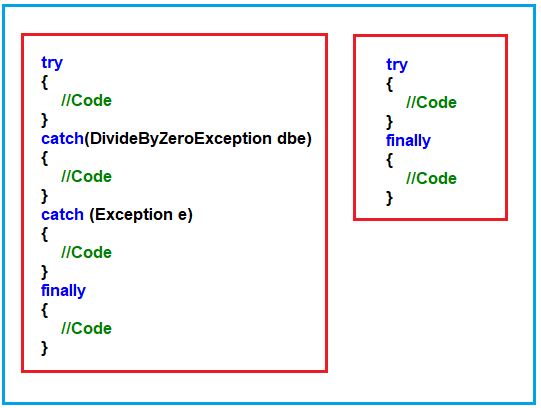
Statement;

}

Finally{

Statements;

}



**Program:**

Class P2

{

Public static void main(String[] args)

{

System.out.println(“main start”);

Int n=1;

Int m=0;

Try

{

Int res=n/m;

}

Catch(Exception obj)

{

System.out.println(obj);

}

Finally

{

System.out.println(“Mandatory task”);

}

System.out.println(“main ends”);

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case | Exception present? | Exception handled  (Catch block) | Execution of finally block ? | Execution of the program  Is completed or continued |
| Case 1 | Present | Handled | Completed | Continued |
| Case 2 | Present | Not handled  (catch block present) | Completed | Not Continued |
| Case 3 | Present | Not Handled  (catch block not present) | Completed | Not  Continued |

**Error:**

Error it is an inbuilt class in java.

Error it is subclass of Throwable class.

Error it is unchecked exception in java.

For error and its subclasses deceleration and handling the exception is not mandatory.

**StackOverFlowError:**

It is a subclass of error class.

It is unchecked exception.

For stack over flow error declaration and handling the exception is not mandatory (For compile time only).

**Program:**

Class P3{

Public static void main(String[] args){

System.out.println(“main method”);

main(null);

}

}

**Question:**

1.Can we handle Stack Overflow Error with help of exception handling

Mechanism.

**CHECKED EXCEPTION:-**

-All the subclasses of Exception class except Runtime Exception are **Checked Exception.**

-For all the Checked Exception Declaration and Handling the Exception is Mandatory.

**File Not Found Exception**

-FileNotFoundException it is an subclass of Exception class.

-It is an checked Exception.

-Declaration and Handling the Exception for FileNotFoundException is Mandatory.

#Program

Import java.io.\*;

Class P4

{

Public static void main(String args[]) throws FileNotFoundException

{

FileOutputStream obj=new FileOutputStream(“D:\\TextFile.txt”);

}

}

**Throws Keyword**

-throws it is an Keyword in Java.

-throws keyword it is used to declare the Exception.

-We can use throws keyword with the help of Method Header..

-With the Help of throws keyword we can declare More than One Exception.

**METHOD HEADER**

**Syntax:-**

AccessModifier ReturnType MethodName(Arguments) throws Exception1,Exception2……ExceptionN

**Example:-**

Public static void main(String args[]) throws FileNotFoundException

**SQL EXCEPTION**

QNo1.Can we Handle the FileNotFoundException explain with the program?

#Program

Import java.io.\*;

Class P4

{

Public static void main(String args[]) throws FileNotFoundException

{

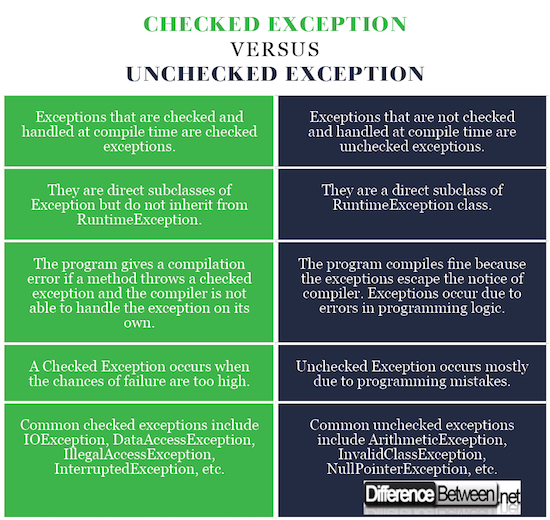
FileOutputStream obj=new FileOutputStream(“D:\\TextFile.txt”);

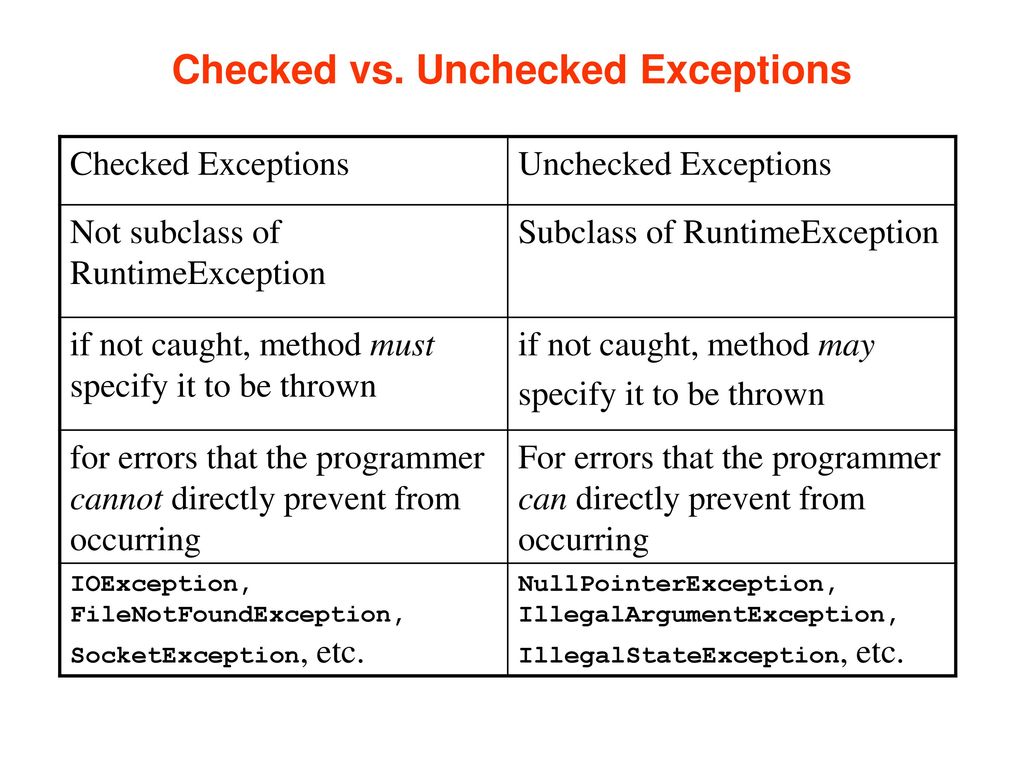
}

}

QNo2.What is the difference between Checked and Unchecked Exception?

|  |  |
| --- | --- |
| Checked Exception | Unchecked Exception |
| Checked exceptions occur at compile time. | Unchecked exceptions occur at runtime. |
| The compiler checks a checked exception. | The compiler does not check these types of exceptions. |
| These types of exceptions can be handled at the time of compilation. | These types of exceptions cannot be a catch or handle at the time of compilation, because they get generated by the mistakes in the program. |
| They are the sub-class of the exception class. | They are runtime exceptions and hence are not a part of the Exception class. |
| Here, the JVM needs the exception to catch and handle. | Here, the JVM does not require the exception to catch and handle. |
| Examples of Checked exceptions:   * File Not Found Exception * No Such Field Exception * Interrupted Exception * No Such Method Exception * Class Not Found Exception | Examples of Unchecked Exceptions:   * No Such Element Exception * Undeclared Throwable Exception * Empty Stack Exception * Arithmetic Exception * Null Pointer Exception * Array Index Out of Bounds Exception * Security Exception |





QNo3.What’s the difference between try,catch,finally block?

|  |  |  |  |
| --- | --- | --- | --- |
| SrNo | Try  Block | Catch  Block | Finally  Block |
| 1 | try is a Keyword  in java | Catch is a keyword in Java | Finally it also keyword  In a java |
| 2 | Try is block  In java | Catch it is an Block in java | Finally it is an Block in java |
| 3 | The Statement which is responsible for Exception that has to be written | The Statement which are executing after exception handling that has to be written inside the catch block | The statement which are mandatory for Execution even if Exception Handled or Not Handled such Statement has to be written inside the finally Block |
| 4 | Try Block used for the Throw and Throwable type of Object | Catch Block is used to Catch and throwable type of object which is thrown by try block | Finally block is used to Execute Mandatory Statement. |
| 5 | Without catch Block or finally Block we cannot use try Block. | Without try block we cannot use catch block. | Without try block we cannot use finally block |
| 6 | We can write only one try block with one catch Block. | We can write More than one catch Block with Single try Block. | We can write only one Finally block with only one try Block. |
| 7 | Try Block Throw  Throwable type of  object | Catch Block catch the Reference of the throwable type of Object. | Finally it is Mandatory. |

**Throw Keyword**

-throw it is an Keyword in Java.

-throw keyword it is declared inside the Method Block.

-With the help of throw keyword we can stop the Execution of the Program.

-With the help of throw keyword we can throw the only one Exception at a time.

**#Program**

Class P5

{

Public static void vote(int age)

{

If(age>=18)

{

System.out.println(“Eligible For Vote”);

}

Else

{

throw ArithemeticException(“Not Eligible For Vote”);

//Creating the Object by Programmer

//and throw to the Compiler

}

}

Public static void main(String args[])

{

Vote(9);

}

}

-The Programmer creates the Exception and throws to the Compiler but the compiler cannot create the Object Reference Variable compiler does not store the address so the Execution of the Program Stopped Forcefully.

-The Programmer create the object and throws to the compiler using throw Keyword.

-Here Programmer Performs Following Steps By its Own self: -

1]Programmer wites the Code

2]Creates the throwable type Object

3]That throwable type of object threw by the Programmer to the Compiler.

4]Programmer Stopped the Execution.

**USER DEFINED EXCEPTION / CUSTOM EXCEPTION**

- A programmer Created Exception is called as Custom Exception.

-For Custom Exception You have to create your own Concreate Class.

-Creates the throwable type Object

-That throwable type of object threw by the Programmer to the Compiler.

-For That user created Concreate Class that is extends by Object Class.

That class won’t behave like Exception so we have make it as to Behave like Exception.

-For that Concreate Class extends to Exception class.

**#Program**

Class AgeNotValid extends Exception

{

AgeNotValid(String var)

{

Super(var);

}

}

Class P5

{

Public static void vote(int age) throws AgeNotValid

{

if(age>=18)

{

System.out.println(“Eligible For Vote”);

}

else

{

throw new AgeNotValid(“Not Eligible To Vote”);

}

}

Public static void main(String[] args)

{

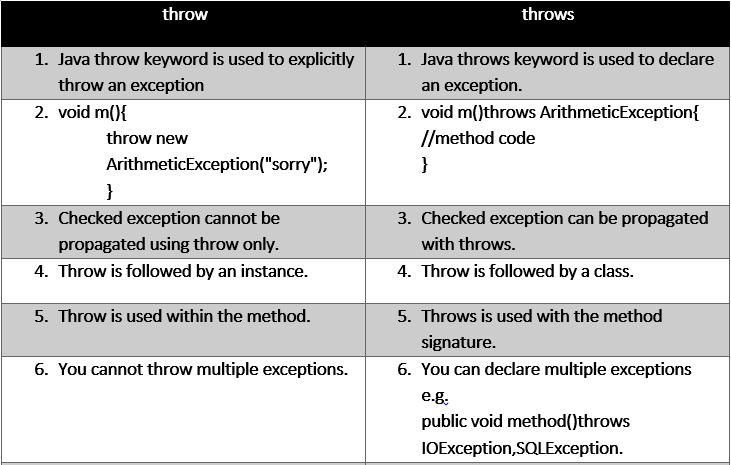
vote(9);

}

}

QNo1.Write difference between throw and throws keyword.

Ans:-



### Throw vs Throws in Java

|  |  |
| --- | --- |
| THROW | THROWS |
| A throw is used to throw an exception explicitly | A throws to declare one or more exceptions, separated by commas. |
| Can throw a single exception using throw | Multiple can be thrown using Throws |
| This keyword is used in the method | Signature method is used with keyword throws |
| Only unchecked exceptions propagated using throw keyword. | To raise an exception throws keyword followed by the class name and checked exception can be propagated. |
| Throw keyword is followed by the instance variable | Throws keyword is followed by the exception class |

QNo2.Design three Real time Example for Custom Exception and Explain?

**#Program 1**

class EmployeeException extends Exception

{

public EmployeeException(String s)

{

super(s);

}

}

class Employee

{

void empIDCheck(int EmpID) throws EmployeeException

{

if(EmpID<=0 || EmpID>999)

{

throw new EmployeeException("Invalid Employee ID");

}

}

public static void main(String[] args)

{

Employee emp = new Employee();

try

{

emp.empIDCheck(0);

}

catch (EmployeeException e)

{

System.out.println("Exception caught");

System.out.println(e.getMessage());

}

}

}

**#Program 2**

class UserException extends Exception

{

int num1;

UserException(int num2)

{

num1=num2;

}

public String toString()

{

return ("Status code = "+num1) ;

}

}

class SampleException

{

public static void main(String args[])

{

try

{

throw new UserException(400);

}

catch(UserException e)

{

System.out.println(e) ;

}

}

}

**#Program 3**

// A Class that represents use-defined exception

class MyException extends Exception {

public MyException(String s)

{

// Call constructor of parent Exception

super(s);

}

}

// A Class that uses above MyException

public class CustomException {

// Driver Program

public static void main(String args[])

{

try {

// Throw an object of user defined exception

throw new MyException("Custom Exception");

}

catch (MyException ex) {

System.out.println("Caught");

// Print the message from MyException object

System.out.println(ex.getMessage());

}

}

}

**1]LIST**

-List it is an Interface in java.

-List it is present inside the java.util package.

-The Fully Qualified Name of List is **java.util.list.**

-List interface it is a child of **Iterator Interface.**

-We cannot create the object for the List Interface[Because it does not support Constructor] but we can create the Object for its Child Classes[**ArrayList,Vector**].

-With the help of List we can Add or Remove and Use and Object[Read] from Anywhere but we cannot Modify the Object.

-List is classified Into Two Types

**1]Linked List**

1]Single Linked List

2]Double Linked List

1]Single Linked List

Address of Own Class

Value

Address of the Next Object

**OBJECT**

**EXAMPLE:-**

Java.lang.String

“ LIST”

“java.lang.Boolean”

Instance of

String Class

Java.lang.Boolean

true

“java.lang.Double”

Instance of Boolean

Class

Java.lang.Double

10.11

NULL

Instance of Double

Class

**2]Double Linked List**

Java.lang.String

Address Of Previous Object

“ LIST”

“java.lang.Boolean”

**EXAMPLE:-**

Java.lang.String

NULL

“ LIST”

“java.lang.Boolean”

Instance Of

String Class

Java.lang.Boolean

Java.lang.String

true

“java.lang.Double”

Instance of

Boolean Class

Java.lang.Double

Java.lang.Boolean

10.11

NULL

Instance of

Double Class

**2]ArrayList**

-ArrayList it is an Concreate Class In Java.

ArrayList it is Inbuild Class which is Present inside **java.util** Package.

-The Fully Qualified name of ArrayList is **java.util.Arraylist.**

-With the Help of ArrayList we can Add ,Remove, Read an Object.

-With the help of ArrayList we can Store **Heterogenous** type of objects.

-We can create an Object for ArrayList because it is an Concreate Class.

-We can create More than one Objects for the ArrayList but we need to import ArrayList Class from util package first.

#Program

Import java.util.ArrayList;

Class P1

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

ArrayList obj1=new ArrayList();

}

}

-In ArrayList we can Store Duplicate Values.

-ArrayList Maintains Insertion Order[FIFO](First Input First Output).

-ArrayList Follows the FIFO order

#Program

Class P2

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

System.out.println(obj);

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

Obj.add(“ArrayList”);

obj.add(true);

obj.add(true);

obj.add(10.11);

System.out.println(obj);

System.out.println(isEmpty());

}

}

**isEmpty()**

-isEmpty() it is an Non-Static Method.

-isEmpty() it is an boolean Method.

-isEmpty() method it is used to checks whether it is Empty or Not.

-We can call isEmpty() method with the help of Object Reference.

-isEmpty() Present inside the ArrayList class.

QNo2.What is Vector?

QNo3.Write a Difference between Vector and ArrayList

#Program 1

import java.util.ArrayList();

Class P1

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(3);//java.lang.Integer

obj.add(“Apple”);//java.lang.String

obj.add(“Ice-cream”);

obj.add(“Mango”);

System.out.println(obj);

System.out.println(obj.contains(“Kiwi”));

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

System.out.println(obj.get(2));//get(index\_Value)

}

}

**O/p:-**

🡺[3,Apple,Ice-Cream,Mango]

🡺false

🡺Ice-cream

**contains()**

-It is used to check whether the object is Present or Not

-The ReturnType is Boolean Condition.

**size()**

-It is used to count the Number Of Object

-The ReturnType is in integer format.

**get()**

-It is used to Access the object based on it’s Index Value.

**Syntax:-**

get(index\_value);

**#Program 2**

import java.util.ArrayList();

Class P2

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(“Amar”);//java.lang.String

obj.add(“Akbar”);

obj.add(“Anthony”);

for(int iv=0;iv<obj.size();iv++)

{

System.out.println(obj.get(iv));

}

}

}

**O/p:-**

**🡺Amar**

**🡺Akbar**

**🡺Anthony**

**IndexOutOfBoundsException**

-When we try to Access an object which is Present inside the ArrayList based on its index value

-If the Index value is Greater or Equals to the Size of ArrayList we get an Unchecked Exception called as IndexOutBoundsException.

**#Program3**

Import java.util.ArrayList;

Class P3

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

Obj.add(“Amar”);

Obj.add(Akbar”);

System.out.println(obj.get(0));

System.out.println(obj.get(1));

System.out.println(obj.get(2))//Exception

}

}

**#Program 4**

import java.util.ArrayList();

Class P4

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();//1st

obj.add(“Tiger”);//java.lang.String

obj.add(“MarieGold”);

System.out.println(obj);

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

ArrayList obj1=new ArrayList();//2nd

obj.add(“ParleG”);//java.lang.String

obj.add(“GoodDay”);

System.out.println(obj1);

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

Obj1.add(obj);//Passing The Entire ArrayList To the 2nd ArrayList Object

System.out.println(obj1);

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

}

}

**O/p:-**

**🡺[tiger,MarieGold]**

**🡺2**

**🡺[ParleG,GoodDay]**

**🡺2**

**🡺🡺[tiger,MarieGold[ParleG,GoodDay]]**

**#Program 5**

import java.util.ArrayList();

Class P5

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();//1st

obj.add(“Tiger”);//java.lang.String

obj.add(“MarieGold”);

System.out.println(obj);

ArrayList obj1=new ArrayList();//2nd

obj.add(“ParleG”);//java.lang.String

obj.add(“GoodDay”);

System.out.println(obj1);

Obj1.addAll(obj);//Passing All the Elements

System.out.println(obj1);

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

}

}

**O/p:-**

**🡺[tiger,MarieGold]**

**🡺[ParleG,GoodDay]**

**🡺[ ParleG,GoodDay,tiger,MarieGold]**

QNo1.Can we Handle IndexOutOfBounds Exception explain with Program

Ans:-

**#Program6**

Import java.util.ArrayList;

Class P3

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

Obj.add(“Amar”);

Obj.add(Akbar”);

System.out.println(obj.get(0));

System.out.println(obj.get(1));

System.out.println(obj.get(2))//Exception

}

}

**add()**

-With the help of add() we add objects to ArrayList

-We can Add Objects in Three Ways:-

1]add(value)

2]add(index\_value, value)

3]add(object\_reference\_variable) : Adding the Entire Object of One ArrayList to Another

**addAll()**

-addAll(object\_reference\_varaible) :Adding all the Objects of One ArrayList to Another

**#Program 7**

import java.util.ArrayList();

Class P7

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();//1st

obj.add(“jawa”);//add(Value)

obj.add(“hero”);

obj.add(“rx100”);

System.out.println(obj);

Obj.add(1,”tvs”);//add(index\_value,value)

System.out.println(obj);

}

}

**O/p:-**

🡺[**jawa,hero,rx100]**

**🡺[jawa,tvs,hero,rx100]**

**MAKING ARRAYLIST AS HOMOGENEOUS**

-For making ArrayList to Homogeneous Then we have take help of

**<>** Operator Name:-

1]Angular Bracket

2]Diamond Bracket

3]Generics

-For Making the ArrayList Homogenous we Have write the Same Type of data

Inside the <> operator

**#Program 8**

import java.util.ArrayList();

Class P8

{

Public static void main(String args[])

{

ArrayList<String> obj=new ArrayList<String>();//1st

obj.add(“jawa”);//add(Value)

obj.add(“hero”);

obj.add(“rx100”);

System.out.println(obj);

}

}

**O/p:-**

🡺[**jawa,hero,rx100]**

Public static void main(String args[])

{

ArrayList obj=new ArrayList();//1st

obj.add(“jawa”);//add(Value)

obj.add(“hero”);

obj.add(true)//CTE

obj.add(“rx100”);

System.out.println(obj);

Obj.add(1,”tvs”);//add(index\_value,value)

System.out.println(obj);

}

**O/p:-**

🡺CTE

**3]VECTOR**

-Vector it is an Concreate Class In Java.

- Vector it is Inbuild Class which is Present inside **java.util** Package.

-The Fully Qualified name of ArrayList is **java.util.Vector.**

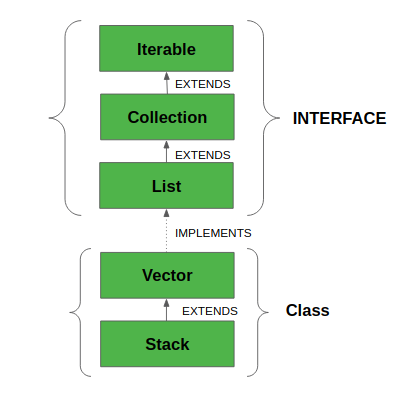
-With the Help of Vector we can Add ,Remove, Read an Object.

-With the help of Vector we can Store **Heterogenous** type of objects.

-We can create an Object for Vector because it is an Concreate Class.

-We can create More than one Objects for the ArrayList but we need to import Vector Class from util package first.

- It extends [AbstractList](https://www.geeksforgeeks.org/abstractlist-in-java-with-examples/) and implements [List](https://www.geeksforgeeks.org/list-interface-java-examples/) interfaces.



### ArrayList Vs vector

| **S. No.** | **ArrayList** | **Vector** |
| --- | --- | --- |
| 1. | ArrayList is not synchronized. | Vector is synchronized. |
| 2. | ArrayList increments 50% of the current array size if the number of elements exceeds ts capacity. | Vector increments 100% means doubles the array size if the total number of elements exceeds its capacity. |
| 3. | ArrayList is not a legacy class. It is introduced in JDK 1.2. | Vector is a legacy class. |
| 4. | ArrayList is fast because it is non-synchronized. | Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in a runnable or non-runnable state until the current thread releases the lock of the object. |
| 5. | ArrayList uses the Iterator interface to traverse the elements. | A Vector can use the Iterator interface or Enumeration interface to traverse the elements. |
| 6 | ArrayList performance is high | Vector performance is low |
| 7 | Multiple threads is allowed | only one threads are allowed |

### Methods in Vector Class

| **METHOD** | **DESCRIPTION** |
| --- | --- |
| [add(E e)](https://www.geeksforgeeks.org/vector-add-method-in-java/) | Appends the specified element to the end of this Vector. |
| [add(int index, E element)](https://www.geeksforgeeks.org/vector-add-method-in-java/) | Inserts the specified element at the specified position in this Vector. |
| [addAll(Collection<?](https://www.geeksforgeeks.org/vector-addall-method-in-java/)  [extends E> c)](https://www.geeksforgeeks.org/vector-addall-method-in-java/) | Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection’s Iterator. |
| [addAll(int index,](https://www.geeksforgeeks.org/vector-addall-method-in-java/)  [Collection<? extends E> c)](https://www.geeksforgeeks.org/vector-addall-method-in-java/) | Insert all of the elements in the specified Collection into this Vector at the specified position. |
| [addElement(E obj)](https://www.geeksforgeeks.org/vector-addelement-method-in-java/) | Adds the specified component to the end of this vector, increasing its size by one. |
| [capacity()](https://www.geeksforgeeks.org/vector-capacity-method-in-java/) | Returns the current capacity of this vector. |
| [clear()](https://www.geeksforgeeks.org/vector-clear-method-in-java/) | Removes all of the elements from this Vector. |
| [clone()](https://www.geeksforgeeks.org/vector-clone-method-in-java-with-examples/) | Returns a clone of this vector. |
| [contains(Object o)](https://www.geeksforgeeks.org/vector-contains-method-in-java/) | Returns true if this vector contains the specified element. |
| [containsAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-containsall-method-in-java/) | Returns true if this Vector contains all of the elements in the specified Collection. |
| [copyInto(Object[] anArray)](https://www.geeksforgeeks.org/vector-copyinto-method-in-java/) | Copies the components of this vector into the specified array. |
| [elementAt(int index)](https://www.geeksforgeeks.org/vector-elementat-method-in-java/) | Returns the component at the specified index. |
| [elements()](https://www.geeksforgeeks.org/vector-elements-method-in-java/) | Returns an enumeration of the components of this vector. |
| [ensureCapacity(int minCapacity)](https://www.tutorialspoint.com/java/util/vector_ensurecapacity.htm) | Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument. |
| [equals(Object o)](https://www.geeksforgeeks.org/vector-equals-method-in-java/) | Compares the specified Object with this Vector for equality. |
| [firstElement()](https://www.geeksforgeeks.org/vector-firstelement-method-in-java/) | Returns the first component (the item at index 0) of this vector. |
| [forEach(Consumer<?](https://www.geeksforgeeks.org/vector-foreach-method-in-java/)  [super E> action)](https://www.geeksforgeeks.org/vector-foreach-method-in-java/) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |
| [get(int index)](https://www.geeksforgeeks.org/vector-get-method-in-java/) | Returns the element at the specified position in this Vector. |
| [hashCode()](https://www.geeksforgeeks.org/vector-hashcode-method-in-java/) | Returns the hash code value for this Vector. |
| [indexOf(Object o)](https://www.geeksforgeeks.org/vector-indexof-method-in-java/#:~:text=indexOf(Object%20element)%20method%20is,does%20not%20contain%20the%20element.) | Returns the index of the first occurrence of the specified element in this vector,  or -1 if this vector does not contain the element. |
| [indexOf(Object o, int index)](https://www.geeksforgeeks.org/vector-indexof-method-in-java/#:~:text=indexOf(Object%20element)%20method%20is,does%20not%20contain%20the%20element.) | Returns the index of the first occurrence of the specified element in this vector, searching forwards from the index, or returns -1 if the element is not found. |
| [insertElementAt(E obj, int index)](https://www.geeksforgeeks.org/vector-insertelementat-method-in-java/) | Inserts the specified object as a component in this vector at the specified index. |
| [isEmpty()](https://www.geeksforgeeks.org/vector-isempty-method-in-java/) | Tests if this vector has no components. |
| [iterator()](https://www.geeksforgeeks.org/vector-iterator-method-in-java-with-examples/) | Returns an iterator over the elements in this list in a proper sequence. |
| [lastElement()](https://www.geeksforgeeks.org/vector-lastelement-method-in-java/) | Returns the last component of the vector. |
| [lastIndexOf(Object o)](https://www.geeksforgeeks.org/vector-lastindexof-method-in-java/) | Returns the index of the last occurrence of the specified element in this vector,  or -1 if this vector does not contain the element. |
| [lastIndexOf(Object o, int index)](https://www.geeksforgeeks.org/vector-lastindexof-method-in-java/) | Returns the index of the last occurrence of the specified element in this vector, searching backward from the index, or returns -1 if the element is not found. |
| [listIterator()](https://www.geeksforgeeks.org/vector-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence). |
| [listIterator(int index)](https://www.geeksforgeeks.org/vector-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence),  starting at the specified position in the list. |
| [remove(int index)](https://www.geeksforgeeks.org/vector-remove-method-in-java/) | Removes the element at the specified position in this Vector. |
| [remove(Object o)](https://www.geeksforgeeks.org/vector-remove-method-in-java/) | Removes the first occurrence of the specified element in this Vector. If the Vector does not contain the element, it is unchanged. |
| [removeAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-removeall-method-in-java/) | Removes from this Vector all of its elements contained in the specified Collection. |
| [removeAllElements()](https://www.geeksforgeeks.org/vector-removeallelements-method-in-java-with-example/) | Removes all components from this vector and sets its size to zero. |
| [removeElement(Object obj)](https://www.geeksforgeeks.org/vector-removeelement-method-in-java-with-example/) | Removes the first (lowest-indexed) occurrence of the argument from this vector. |
| [removeElementAt(int index)](https://www.geeksforgeeks.org/vector-removeelementat-method-in-java/) | Deletes the component at the specified index. |
| [removeIf(Predicate<? super E> filter)](https://www.geeksforgeeks.org/vector-removeif-method-in-java/) | Removes all of the elements of this collection that satisfy the given predicate. |
| [removeRange(int fromIndex,](https://www.geeksforgeeks.org/vector-removerange-method-in-java-with-example/)  [int toIndex)](https://www.geeksforgeeks.org/vector-removerange-method-in-java-with-example/) | Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| [replaceAll(UnaryOperator<E> operator)](https://www.javatpoint.com/java-vector-replaceall-method) | Replaces each element of this list with the result of applying the operator to that element. |
| [retainAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-retainall-method-in-java-with-examples/) | Retains only the elements in this Vector contained in the specified Collection. |
| [set(int index, E element)](https://www.geeksforgeeks.org/vector-set-method-in-java/#:~:text=set()%20method%20is%20used,Vector%20class%2C%20with%20another%20element.&text=Parameters%3A%20This%20function%20accepts%20two,above%20syntax%20and%20described%20below.) | Replaces the element at the specified position in this Vector with the specified element. |
| [setElementAt(E obj, int index)](https://www.geeksforgeeks.org/vector-setelementat-method-in-java-with-example/) | Sets the component at the specified index of this vector to be the specified object. |
| [setSize(int newSize)](https://www.geeksforgeeks.org/vector-setsize-method-in-java-with-example/) | Sets the size of this vector. |
| [size()](https://www.geeksforgeeks.org/vector-size-method-in-java/) | Returns the number of components in this vector. |
| [sort(Comparator<? super E> c)](https://www.geeksforgeeks.org/collections-sort-java-examples/) | Sorts this list according to the order induced by the specified Comparator. |
| [spliterator()](https://www.geeksforgeeks.org/arraylist-spliterator-method-in-java/) | Creates a late-binding and fail-fast Spliterator over the elements in this list. |
| [subList(int fromIndex, int toIndex)](https://www.geeksforgeeks.org/vector-sublist-method-in-java/) | Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive. |
| [toArray()](https://www.geeksforgeeks.org/vector-toarray-method-in-java-with-examples/) | Returns an array containing all of the elements in this Vector in the correct order. |
| [toArray(T[] a)](https://www.geeksforgeeks.org/vector-toarray-method-in-java-with-examples/) | Returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array. |
| [toString()](https://www.geeksforgeeks.org/vector-tostring-method-in-java-with-example/) | Returns a string representation of this Vector, containing the String representation of each element. |
| [trimToSize()](https://www.geeksforgeeks.org/vector-trimtosize-method-in-java-with-example/) | Trims the capacity of this vector to be the vector’s current size. |

**SETS**

-Set it is an Interface In java

-It is a Sub Interface of Iterator Interface.

-It is Present inside the **java.util** package.

-The Fully Qualified Name **java.util.Set.**

-We cannot create object for Set Interface but we can create Object for its child Classes[HashSet,TreeSet]

**1]HashSet**

-HashSet is a Concreate Class in Set.

-HashSet Present inside the **java.util** Package.

-The Fully Qualified Name oh HashSet is **java.util.HashSet**

-We can store the Heterogenous Type objects inside the HashSet.

-HashSet Does Not maintain insertion order.

-HashSet it will remove **Duplicate Objects** Automatically.

-We can create an object for HashSet but we need to import HashSet from **java.util** package.

**#Program**

Class P1

{

Public static void main(String args[])

{

HashSet obj=new HashSet();

obj.add(123);

obj.add(true);

obj.add(“HashSet”);

obj.add(true);

obj.add(‘a’);

obj.add(123);

obj.add(10.123);

System.out.println(obj);

}

}

**O/P**

🡺[a,hashSet,10.123,123,true]

**2]TreeSet**

-TreeSet is an Concreate class.

-TreeSet it is Present inside **java.util** Package

-We can Store Homogeneous Type of objects inside TreeSet.

-TreeSet does not Allow Duplicate Object

-TreeSet it will Automatically Sort the object in Ascending Order.

-We can create an Object for TreeSet but we need to import TreeSet from

Util package first.

**#Program**

**Import java.util.TreeSet;**

Class P2

{

Public static void main(String args[]);

{

TreeSet obj=new TreeSet();

obj.add(3);

obj.add(3);

obj.add(2);

obj.add(1);

obj.add(2);

obj.add(4);

}

}

**O/P**

**🡺**[1,2,3,4]

**=================== IMPORTANT ===================**

-If we try to pass different type object inside TreeSet which is Not Comparable Type we get an Unchecked Exception called as ClassCastException

**import java.util.TreeSet;**

Class P3

{

Public static void main(String args[]);

{

TreeSet obj=new TreeSet();

obj.add(3);

obj.add(3);

obj.add(true)//ClassCastException

obj.add(4);

}

}

Que.Can we Handle ClassCastException of TreeSet Explain with the Program

Ans:- Yes we can handle the ClassCastException Of TreeSet

**remove(value) MEHTOD**

**#Program**

Class P4

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(123);

obj.add(true);

obj.add(‘a’);

System.out.println(obj);

obj.remove(true);

System.out.println(obj);

}

}

**O/P**

**🡺[123,true,a]**

**🡺[123,a]**

**remove(index\_value) MEHTOD**

**import java.util.ArrayList;**

**#Program**

Class P4

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(123);

obj.add(true);

obj.add(true);

obj.add(‘a’);

System.out.println(obj);

obj.remove(2);

System.out.println(obj);

}

}

**O/P**

**🡺[123,true,true,a]**

**🡺[123,a]**

-To remove All the Duplicate Object of ArrayList we need to convert ArrayList to HashSet.

**#Program**

**import java.util.ArrayList;**

**import java.util.HashSet;**

Class P5

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(123);

obj.add(true);

obj.add(‘a’);

obj.add(true);

obj.add(true);

obj.add(‘a’);

obj.add(true);

obj.add(true);

System.out.println(obj);

HashSet obj1=new HashSet(obj);

System.out.println(obj1);

}

}

O/P

🡺[123,true,a,true,true,a,true,true,]

🡺[123,true,a]

Que1.How To remove An integer Object from ArrayList.

Ans:-

**#Program**

**Import java.util.ArrayList;**

Class P6

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

obj.add(123);

obj.add(true);

obj.add(true);

obj.add(‘a’);

System.out.println(obj);

obj.remove(0);//IndexValue

System.out.println(obj);

}

}

**O/P**

**🡺[123,true,true,a]**

**🡺[123,a]**

============================== **IMPORATANT** ==============================

-When we try remove to an Integer Object from ArrayList by The Type Of Object by Passing The Integer Value.

-The Value it is Automatically Converted into Index Value.

**#Program**

import java.util.ArrayList;

class P7

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

Obj.add(1);

Obj.(true);

Obj.add(‘a’);

System.out.println(obj);

System.out.println(1);//Considered As a Index Value

System.out.println(obj);

}

}

**#Program**

**import java.util.ArrayList;**

class P8

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

Obj.add(1);

Obj.(true);

Obj.add(‘a’);

Obj.add(false);

System.out.println(obj);

System.out.println(Integer.ValueOf(1));

System.out.println(Boolean.valueOf(false));

System.out.println(obj);

}

}

========================== **IMPORTANT** ==========================

-To remove the Integer Object by The Type Of Object we need to take the help

valueOf() Method which is Static Method.

**removeAll()**

-It is an Non-Static Method where it is used to compare the One ArrayList

With Another ArrayList and Remove the Same type Of Object

**#Program**

**Import java.util.ArrayList;**

Class P9

{

Public static void main(String args[])

{

ArrayList obj =new ArrayList();//1st

0bj.add(1);

obj.(true);

System.out.println(obj);

ArrayList obj1=new ArrayList();//2nd

obj.add(‘a’);

obj.add(true);

System.out.println(obj1);

obj1.removeAll(obj);

System.out.println(obj1);

}

}

**O/p**

**🡺[1,true]**

**🡺[a,true]**

**🡺[a]**

**retainAll()**

-It is used to compare One ArrayList with Another ArrayList and remove Different Type Of Objects and Retain the Same Type.

**#Program**

**Import java.util.ArrayList;**

Class P10

{

Public static void main(String args[])

{

ArrayList obj =new ArrayList();//1st

0bj.add(1);

obj.(true);

System.out.println(obj);

ArrayList obj1=new ArrayList();//2nd

Obj1.add(‘a’);

Obj1.add(true);

System.out.println(obj1);

obj1.retainAll(obj);

System.out.println(obj1);

}

}

**O/p**

**🡺[1,true]**

**🡺[a,true]**

**🡺[true]**

**Que.Write A Program to Sort Reverse order the Object Present inside the ArrayList**

-We can Sort the Reverse the Object Present inside the ArrayList with the help of Two inbuild Methods which is Present inside Collections Class.

**1]sort()**

**-It is Static Method**

**-Present inside the Collections Class**

**-It will sort the Same Type of Element.**

2**]reverse()**

**-It is Static Method**

**-Present inside the Collections Class**

**-It will reverse the list.**

-sort() and reverse() are Static Method Which is Present inside the Collections Class we can Access the two Methods with the help of Class Name As a Reference.

**#Program**

**Import java.util.ArrayList;**

**Import java.util.Collections;**

Class 11

{

Public static void main(String args[])

{

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

obj.add(1);

obj.add(2);

obj.add(3);

obj.add(4);

System.out.println(obj);

Collections.sort(obj);

System.out.println(obj);

Collections.reverse();

System.out.println();

}

}

QNo1.What is sort() and reverse()?

Ans:

**1]sort()**

**-It is Static Method**

**-Present inside the Collections Class**

**-It will sort the Same Type of Element.**

2**]reverse()**

**-It is Static Method**

**-Present inside the Collections Class**

**-It will reverse the list.**

QNo2.To sort an ArrayList Making ArrayList is Homogenous Is Mandatory or Not Mandatory?

Que.Can we read the Object of ArrayList with the help of SuperMost Class.[Object Class]

Ans:-

**#Program**

Import java.util.ArrayList;

Class P12

{

Public static void main(String args[])

{

ArrayList obj =new ArrayList();

Obj.add(100);

Obj.add(‘a’);

Obj.add(true);

//for Each Loop or Advanced for loop

for(Object var : obj)

{

System.out.println(var);//or var.toString()

}

}

}

**FOR EACH LOOP / ADVANCED FOR LOOP**

**Syntax**

**for(ClassName var\_Name : var\_Name)**

**{**

**Statements;**

**}**

**Example:-**

**for(Object var : obj)**

**{**

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

**}**

Java.lang.Integer

Java.lang. Integer

Java.lang.Boolean

true

NULL

Java.lang.Boolean

Java.lang.Character

‘a’

Java.lang.Character

100

Java.lang. Character Java.lang. Boolean

Overriding

**UPCASTING**

Object var

OBJECT

toString()

{

}

Java.lang.Boolean

for(Object var : obj )

Integer

Character

Boolean

Java.lang.Boolean

Java.lang.Boolean

{

Integer

toString()

{

}

Sop(var.toString());

}

**#Program**

Class P13

{

Public static void main(String args[])

{

ArrayList obj=new ArrayList();

Obj.add(100);

Obj.add(‘a’);

Obj.add(true);

Iterator var=obj.iterator();//NS-Method

While(var.hasNext())

{

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

}

}

Java.lang.Boolean

Java.lang.Character

Java.lang.Integer

**true**

Java.lang.Character

Java.lang.Integer

**‘a’**

**Null**

Java.lang.Boolean

**Null**

**100**

Java.lang.Character

O/P

**true**

**100**

**‘a’**

**While(var.hasNext())**

Java.lang.Boolean

**{**

**Sop(var.next());**

**100**

**}**

**hasNext()**

It is used to check verify whether the current Object Consist the Address of Next Object or Not.

**next()**

-It is used to Access the value which is Present in current Object.

QNo1.Verify All the Program of ArrayList using Vector,HashSet,TreeSet?

QNo2.Sort And reverse the order for Hash and TreeSet

QNo3.Write the difference between HashSet and TreeSet