Command line Arguments

We can send the input at the runtime in the command line.

We can send the data while printing java filename.

Eg: java filename 1 2 3 4

↓ ↓

starts command Arguments

Eg: java filename 1 2 3

↓ ↓

command Arguments

We can pass the input values in this way to the program by using the command

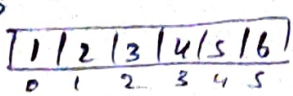
These values are stored in (String arg[ ]) [ ] an array

public static void main(String args[])

always it should be of String type. we cant modify it and we should always keep [ ] because it stores the data values in it.

Variable name can be as per user choice Eg: args, s, etc

We can give many as value as we need.



s[1]=2.

s[6]= → Array Index Out of Bounds Exception

The argument which are send at the command line is called Command line Arguments. The argument values are store in the main method. array.

Initially it takes the arguments as string type.

```
class CommandLineDemo
{
public static void main(String args[])
{
System.out.println(args[0]+"  "+args[1]);
float c= Integer.parseInt(args[0])+ Float.parseFloat(args[1]);
System.out.println(c);
}
}
```

Output:

```
javac CommandlineDemo.java
java CommandlineDemo 16 2.3
16 2.3
18.3
```

To convert string type variable into int type values by using predefined class called Wrapped classes.

Integer	Long
Float	Character
Double	Boolean
Short	Byte.

Integer.parseInt() is used to convert it into a integer value. It is predefined method. It is a static type.

→ Write a Java program to read two characters from command line and check whether those two are same or not.

```
class CommandlineDemo
{
public static void main(String args[])
{
char c1=Character.parseChar(args[0]);
char c2= character.parseChar(args[1]);
```

```
if(c1==c2)
{
System.out.println("Two characters are same");
}
else
{
System.out.println("Two characters are not same");
}
}
```

Output:-

```
javac CommandLineDemo a A
Two character are not same
```

→ We can take the many values as we required, there is no limit for giving the values. It depends on memory space.

Arrays:-

Array is a collection of homogeneous data items which are stored in continuous memory locations.

In Java, Arrays are dynamically allocated

To declare Array ⟶ datatype arrayname[];

or

datatype[] arrayname; Ex: int a[10];
int[] a;

Difference between declare & define

While defining the array the memory is allocated.

While declaring it acts just as reference.

It says that it will allocated some memory

Arrays should be allocated in memory continuously and it should contain similar elements in it.

For defining

Syntax:

arrayname = new datatype[size];

declaration of array => int a[];

defining of array => a = new int[10];

int a = new int[10];

While declaration, we may or may not specify the size

System.out.println(a)

[I@15db34]

↓ Integer ↓ some address

Normally when we declare the reference variable for class and print that reference variable. then we get [classname@1234] ↓ some address

Initializing the values → array literal (constant)

1) int a[] = new int[]{1,2,3,4,5,10};

When we declare the value into the array, then we should not mention the size.

2) int a[] = {1,2,4,5,10};

Example of defining & Initializing an array

- int b[] = new int[5];
- float f[] = new float[]{1.6, 3.4, 3.6, 6.8};
- char c = new char[10];
- double[] d = {3.6, 4.2, 3.194, 60.34};

→ When we define the array, by defaultly it value is '0'

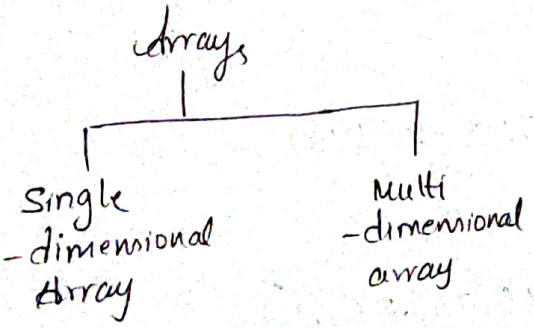
→ ~~We should~~ the size of the array should always be of int type or but not short or long.

```
class ArrayDemo
{
 public static void main(String s[])
  {
   int a[]=new int[5];
   System.out.println(a[0]+" "+a[1]+" "+a[2]);
  }
}
```

Output

0 0 0

```
class ArrayDemo
  {
  public static void main(String s[])
    {
short int b=10;
   int a[]=new int[b];
   System.out.println(a[0]+" "+a[1]+" "+a[2])
  }
}
```



Eg:

```
class ArrayDemo
{
 public static void main(String s[])
  {
   int a[]=new int[10];
   Scanner sc=new Scanner(System.in);
```

```
for(int i=0; i<10; i++)
                  → a.length
 {
  a[i]=sc.nextInt();
 }
for(int i=0; i<10; i++)
 {
System.out.println(a[i]);
 }
}
}
```

Instead of '10' we can declare it as a.length

a.length is a member of an array class.
It's not a method.
So we can directly use it

→ Write a Java program to read a single dimensional array and find some of the all the elements in an array

```
class Array
{
public static void main(String args[])
{   float sum=0;
 float a[]=new float[10];
 Scanner sc=new Scanner(System.in);
for(int i=0; i<10; i++)
 {
  a[i] = sc.nextFloat();
 }
for(int i=0; i<10; i++)
                  ↓
                a.length
 {
 sum=sum+i;
 }
System.out.println("Sum is "+sum);
}
}
```

Output

```
6.0
5.0
4.0
3.0
2.0
1.0
2.0
3.0
4.0
5.0
Sum is 35.0
```

Here for specifying the result within the particular decimal values. we can use

```
System.out.printf("%.3.2f", sum);
System.out.format("%.0.2f", sum);
```

// check it.

* Write a java program to read a character array and print its Ascii values.

```
import java.util.Scanner;
class Charact
{
public static void main(String args[])
 {
char a[]=new char[10];
Scanner s1=new Scanner(System.in);
 for(int i; i<10; i++)
  {
  a[i]= s1.next.charAt(0);
   }

 for(int i=0; i<10; i++)
  {
 System.out.printf("%d", a[i]); //This is not supported in java present version.
  } s.o.printf("(int)a[i]);
 }
}
```

Output

a 2 2 : 9 0 b c d }

↓

a-z :

A-Z :

2-dimensional Array

It is an Array of Array in Java.

Syntax

```
datatype a[][] = new datatype [rowsize][columnsize];
```

Eg:

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

a[0][0] = 1
a[0][1] = 2
a[1][0] = 3
a[1][1] = 4
```

Here for initializing two ways are possible, one as above declared as above and by using Scanner

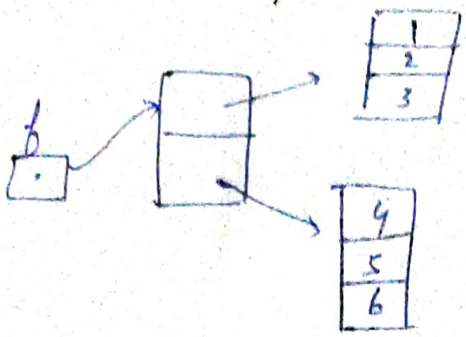
W.A.P to read & write a 2 dimensional array

```
class TwoDimensionalArrayDemo
{
public static void main(String s[])
{ Scanner sc=new Scanner(System.in);
  float f[][]=new float[2][3];
  System.out.println("enter array elements");
  for(int i=0;i<2;i++)
  {
    for(int j=0;j<3;j++)
    {
      f[i][j]=sc.nextFloat();
    }
  }
  System.out.println("the array elements are");
  for(int i=0;i<2;i++)
  {
    for(int j=0;j<3;j++)
    {
      System.out.println(f[i][j]+" ");
    }
  }
}
```

Output

```
Enter array elements
1 5 9
7 8 2
the array elements are
1 5 9
7 8 2
```

Internal 2d representation in Java



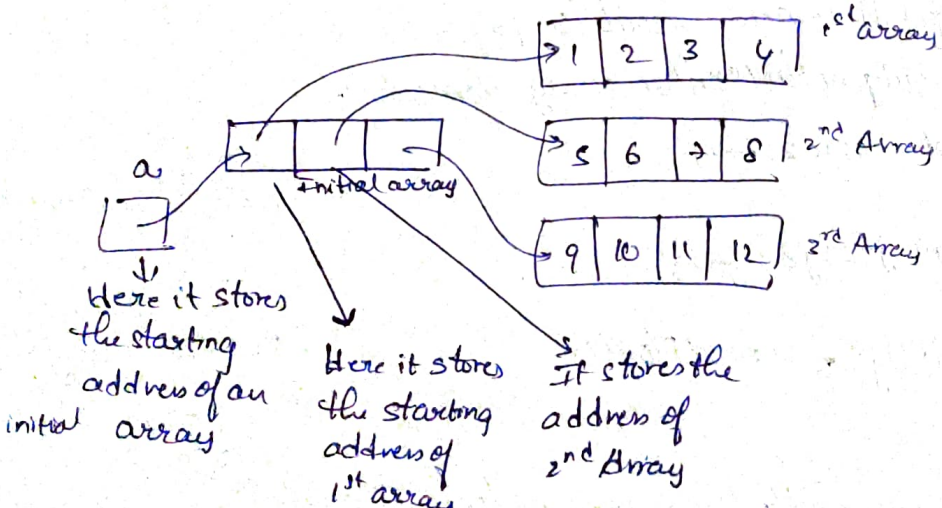
Jagged Arrays

In jagged Arrays the column size is not fixed, we can change the column size as per user choice, but the row size should be fixed.

Eg: a[4][ ]

1 2 3 4 5
6 →
8 9 10
11 12 13 14

int a[][]=new int[3][4];



Representation of 2d in Arrays.

int a[][]=new int[3][ ];

The column size is not fixed and takes the size of variable length, is called jagged array.
it is not fixed of column size.

In single dimensial array we mention size as int and it stores the values of all the types

where as in two dimensional Array we can stores the values of different types like char, float, int, etc... Here in rows can be change of any data type, but the columns in a row should be of same type

Ex: [3][ ]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ a & b & & \\ 4.5 & 6.0 & 1.8 & \end{bmatrix}$$

This is possible in jagged arrays.

Program

```
class TwoDimensionalArray ArraysDemo
{
public static void main(String S[])
{
  Scanner s1=new Scanner(System.in);
  int a[][]=new int[3][];
   a[0]=new int[4];
   a[1]=new int[3];
   a[2]=new int[5];
  System.out.println("enter first row data");
  for(int i=0;i<a[0].length;i++)
     a[0][i]=s1.nextInt();
  System.out.println("enter 2nd row data");
  for(int i=0;i<a[1].length;i++)
     a[1][i]=s1.nextInt();
  System.out.println("Enter 3rd row data");
  for(int i=0;i<a[2].length;i++)
     a[2][i]=s1.nextInt();
}
```

```
for(int i=0;i<a[0].length;i++)
  System.out.println(a[0][i]);
  }
}
```

By using an Object class, we can use the object type arrays. An Object class is the super class of all the classes.

```
class TwoDimensionalArrayDemo
{
public static void(String s[])
{
  Scanner s1=new Scanner(System.in);
  Object a[][]=new object[3][];
     a[0]=new Integer[4];
     a[1]=new Float[3];
     a[2]=new character[3]
System.out.println("enter first row data");
   for(int i=0;i<a[0].length;i++)
     a[0][i]=s1.nextInt();
  System.out.println("enter 2nd row data");
    for(int i=0;i<a[1].length;i++)
      a[1][i]=s1.nextFloat();
  System.out.println("enter the 3rd row data");
    for(int i=0;i<a[2].length;i++)
      a[2][i]=s1.next().charAt(0);
  }
}
```

a is of the object class, then we can't pass the primitive values. so we have to pass only object type values
Only for primitive datatypes we can create wrapper class.
String is not a primitive datatype. So we can't create the wrapper class.

1-D Array

```
int a[]=new int[3]; ✓
int[] a=new int[3], ✓
int a[] ={1,2,3,4}; ✓
int a[]= new int[]; ✗
```

2-D Array

```
int a[][]=new int[2][3]; ✓
int[][] a=new int[2][3]; ✓
int a[][]={1,2,3,4}; ✗
int a[][] =new int[2][]; ✓
int a[][] = new int[][]{{1,2,3,4},{1,2,3,4}}; ✓
```

In 2D row size should mention:

```
int[] a[]=new int[2][]; ✓
```

Jagged Array is an Array, where in each row has variable length of columns are data, is called Jagged Arrays

Array & function

Passing a single dimensional array to a function and also returning an array from a function.

```
class Array
{
// static int sum=0;
static int function(int b[]);
{
sum=0;
for(int i=0;i<b.length;i++)
```

```
        sum=sum+i;
        return sum;
    }
    public static void main(String args[])
    {
      int a[]=new int[10]
    Scanner s=new Scanner(System.in);
for(i=0;i<10;i++)
a[i]=s.nextInt();
int sum=function(a);
    } System.out.println("the sum is"+sum);
    }
}
```

Output:

9
8
7
6
5
4
3
2
1
10

8 4 20 3

Program:

```
class UpdateArray
{
public static void main(String args[])
{
int a[]=new int[10];
Scanner s=new Scanner(System.in);
for(int i=0;i<a.length;i++)
a[i]=s.nextInt();
sum_of_array_elements(a);
System.out.println(a[2]+" "+a[4]+" "+a[9]+" "+a[6]);
}
static int Sum_of_array(int b[])
{
b[2]++;
b[4]--;
b[9]=b[9]+10;
}
}
```

Every method data is stored in stack memory. the objects are stored in heap memory.

Stack is used to execute function recursively

→ When we passes the array to function it does not allocate the Memory and it is copy reference variable.

→ the reference are stored in stack.

→ Heap is stored only objects it does not store local variables.

→ The every method should be stored in the stack whatever languages.

→ the stack will be deleted when the execution is completed.

→ To execute the function we create concept of stack- method variables are stored in stack. until this method is executed

Program

```
Class ArrayDemo
{
public static void main(String s[])
{
int a=10, b=20, c;
 c=a+b;
System.out.println(c);
Subtract(a,b);
Mul(a,2);
}
static void subtract(int c, int d)
{
S.o.println(c-d);
}

static void mul(int e, final int f)
 {
s.o.println(e*f);
}}
```

when the main execute the stack will open.

Output

30
-10
20

23/12/19

```
class Demo
{
int a=20;
float b=30.63;
void display()
{
int c=30;
System.out.println(c);
}
public static void main(String s[])
{
Demo d1=new Demo();
System.out.println(d1.a+" "+d1.b);
d1.display();
d1.func();
}
void func()
{
char c='$';
System.out.println(c);
}
}
```

o/p :-

20 30.63

30

$

In this program the stack can be open. when the method is created. If the object is present it will be the stored in the heap memory.

Copying an array into another array

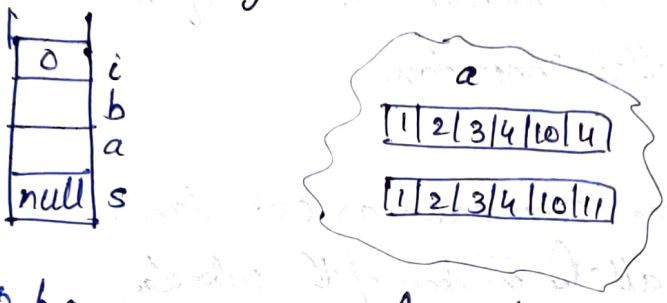
void arraycopy(srcarray, srcindex, desarray, desindex, pos)

Program

```
class Demo
{
public static void main(String s[])
```

```
{
int a[]={1,2,3,4,10,11};
 int b[a.length];
System.arraycopy(a,0,b,0,a.length);
 for(int i=0;i<b.length;i++)
 System.out.printf("%d ",b[i]);
 }
}
```

a[] is dynamically created it is stored in the Heap



a.0, b.0 means it copy the a elements 0 index to b elements.

Output

1 2 3 4 10 11

Nested classes

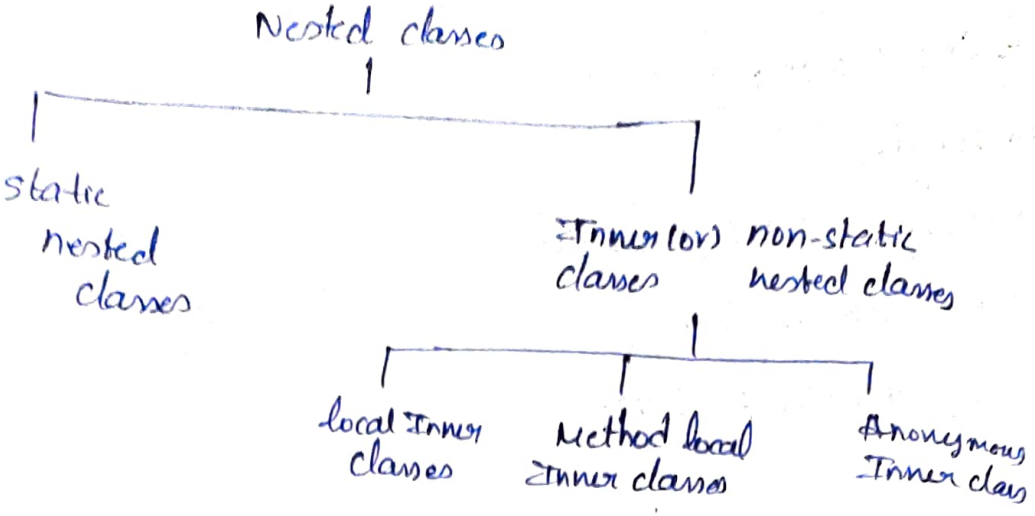
A class which is enclosed in another class is known as nested class.

syntax

```
class Outerclass
{
≡≡
class Innerclass
 {
 ≡≡
 }
≡≡
}
```

nested class.

We have the many nested classes based on the behaviour of the classes. they are



Inner classes

A class which is declare without static modified inside another class is called Inner class.

Properties:

(i) The inner class have same name as outer class.

(ii) The inner can access all the members of the outer class including private data.

(iii) The outer class can't access inner class member.

(iv) .class files generated both inner class as well as outer class.

Ex.
```
class A
  {
   ≡   ⟶ A.class
  class B
    {
     ≡    A$B.class
    }
  }
```

(v) The inner class data can be access only through the outer class object.

(vi) The inner class can have the following members in it

1. Instance variables
2. Instance Method
3. Constructors

Mi) The inner class should not contain any static members.

1. How to create an object for Inner class?

Ans Syntax:

Outerclass.Innerclass objectname = outerclass object.new Innerclass();

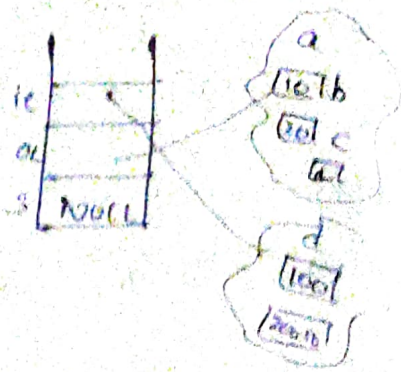
Ex: class A
{
≡
class B
{
≡
}
}

A a = new A(); → outerclass
A.B b = a.new B(); Inner class

Ex:

```
class Outerclass
{
int a=10;
float b=20.30;
private char c='z';
void display()
{
System.out.println("inside display");
}
class Innerclass
{
int d;
float e;   } Instance variables
Innerclass(int a, float b)
{
this.d=a;
this.e=b;
}
void fun() → Instance Methods
{
System.out.println("a+" "+b");
display();
```

```
    }
}
public static void main(String s[])
{
Outerclass oc=new Outerclass();
Outerclass.Innerclass ic=oc.new Innerclass(100,200.36);
ic.display();
System.out.println(ic.d+" "+ic.e);
ic.func();
}
}
```



→ Finalize is a Method automatically called the garbage.
→ garbage collector can be run JVM.
→ garbage collector is a program that can be run on JVM the JVM will keep the time to destroy the object.
→ Finalize Method is a object class.
→ they are 7 important methods in object class. In that "Finalize" is one of them.

Program

```
class Outerclass
  {
  int a=10;
  float b=20.36;
private char c='z';
void display
```

Output:

inside display
100 200.6
10 20.36
inside display.

→ even for the constructor the static memory will be created.

→ If we write a static method (or) members in the outerclass they can be accessed by inner class.

static nested classes

A class which is declared with static modified inside another class is called static nested class

Syntax of static nested class

```
class Outerclass
  {
  ==
static class Nestedclass
  {
  ==
  }
==
}
```

Syntax to create an object for static nested class

```
Outerclass name.Innerclassname
    objectname = new Outerclassname.
                 Innerclassname();
```

Program

```
class outer
 {
  int a=10;
 static int b=20;
 Void display()
  {
  System.out.println(a);
  }
 static void fun1()
  {
  System.out.println(b);
  }
 class Inner
  {
  int c=30;
 void fun2()
  {
 System.out.println(a+" "+b);
  display(); //cannot be accessed.
  fun1();
  }
}
public static void main(String s[])
 {
  Outer o=new outer();
 Outer.Inner i=o.new Inner();
     i.fun2();
  System.out.println(i.c);
  }
 }
```

original output

10 20

20

30

→ A static nested class can contain both instances data member and static data member.

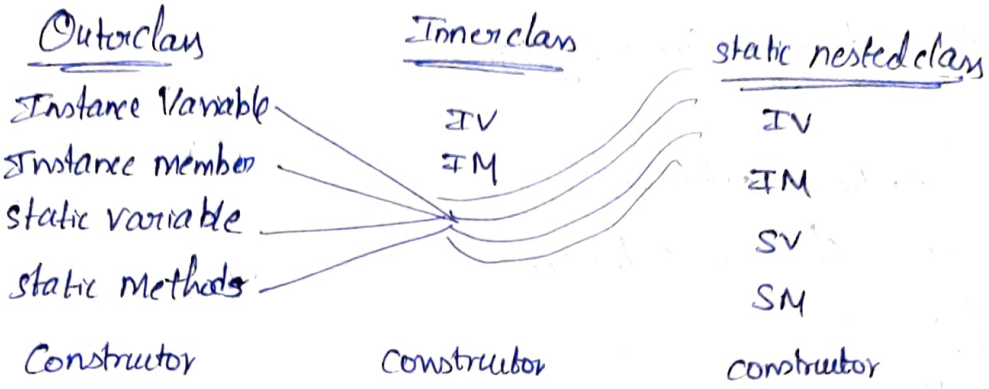
Program

```
class outer
 {
  int a=10;
 static int b=20;
 void display()
  {
 System.out.println(a);
 }
 Static void fun1()
  {
 System.out.println(b);
  }
 static class Inner(or) Nested
    {
     int c=30;
    void fun2()
    {
 S.o.pln(a+" "+b);
 display();
 fun1();
  }
 Static int d=40;
 Static void fun3()
  {
  S.o.pln(b);
   fun1();
  }
}
public static void main(String s[])
  {
 Outer.Inner i=new Outer.Inner();
    i.fun2();
 Outer.Inner.fun3();
  Sopln(Inner.d);
  }
}
```

→ We can write the Main in the innerclass because in the static nested class the static can be access.

→ It can create a class as its own.

→ we don't call fun3 we should write before it belongs to which class.



→ static member can't directly access a non-static member. If we want to access we want to be create an object.

Program.

```
class outer
 {
 int a=10;
 static int b=20;
 void display()
 {
s.o.pln(a);
 }
static void fun1()
 {
s.o.pln(b);
 }
 static nested class
 {
  int b=100;
 static char c='C';
 void fun1()
 {
s.o.pln(a+" "+b);
disp1();
disp2();
}
```

```
outer(int a)
{
System.out.println(a);
}
```

```
static void fun2()
{
S.o.println(b);
disp2();
}
}
public static void main(String args[])
{
  Outer ob=new outer(30);
  Outer.Inner obj=new Outer.Inner();
   obj.fun1();
   obj.disp();
  Inner.fun2();          (or)
  S.o.pln(B.C);
    A.disp2();
  }
}
class outerclass
  {
  int a;
static int b=60;
void display()
{
S.o.pln("a="+a+"b="+b);
}
static void fun1()
{
S.o.pln("This is function1");
S.o.pln("b="+b);
}
Outer class int c)
{
 a=c;
}
static class Nestedclass
{
```

```
int d=23;
static float f=36.24;
void fun2()
 {
So.pln(a+" "+b);
display();
fun1();
 }
static void fun3()
 {
So.pln(b+" "+f);
   fun1();
 }
}
Public static void main(String s[])
 {
Outerclass oc=new Outerclass(80);
 oc.display();
Outerclass.fun1();
Outerclass.Nestedclass nc=new outerclass.Nestedclass();
 nc.fun2();
nc.fun3();
 }
}
```

static nested class

It is the latest version.

→ In static nested class member can access only the static members of its outer class.

→ They can't access the instance members of outerclass

Outer

IM	IV
SM	IM ✓
IV	SV
SV	SM ✓

Static Nested Classes

→ static nested class members can't access the instance member of outer class.

1) How to create an object for static nested classes?

Syntax:

```
Outerclass.staticnestedclass objname = new Outerclass.staticnestedclass();
```

eg:

```
class outer
 {
 int a;
 static float b=2.0;
  Outer(int a)
 {
 this.a=a;
  }
 void display()
 {
 S.o.pln(a);
 S.o.pln(b);
 }
 static void fun1()
  {
 S.o.pln(b);
 }
}
→ static nested class
  {
   int e;
 static int f=20;
  nested class(int e)
  {
  this.e=e;
 So.pln("1- parameterized constructor");
  }
 void fun2()
  {
```

```
S.o.println(b);
S.o.println(fun1);
}
static void fun3()
{
S.o.println(fun1);
S.o.println(b);
}
```

→ The nested class can access only static members but not the instance variables.

```
public static void main(String ST[])
{
Outer O=new Outer(10);
Outer.staticnestedclass ob=new outerclass.staticnestedclass();
ob.fun2();
ob.fun3();
ob.display();
outer.fun1();
}
}
```

1) What is the differences b/w non-static and static nested class?

A) 1) Nested static class doesn't need reference of outer class but non static nested class or inner class requires outer class reference you cannot create instance of inner class. This is by for most important thing to consider while making a nested class static or non-static.

2) Static class is actually static member of class and can be used in static context

Eg: static method or static block of outer class.

3) Another diff b/w is that you cannot access non-static members.

Eg: Method and field into nested static class directly.
If you do you will get error like "non-member cannot be used in static context". While inner class can access both static and non-static member of outer class.

Method local Inner class

We write the method in inner class is called as method local Inner class.

Ex:
```
class outer
{
==
void m()
{
 class inner
   {
   ==
   }
}
```

In class we can write all Instance, non Instance means static. In the class also we can write any thing

→ In the one function definition in another function definition we can't write. But we have the one concept.

Ex:
```
void fun1()
{
void fun2()
 {
 }
}
```
× It is not possible but we have the one concept. Method local Inner class.

By using these concept we can write the method definition inside the method.

Program
```
class outer
{
 int a=10;
void m1()
{
```

```
class Inner
 {
void sum(int x,int y)
  {
System.out.println(x+y);
 }
}
Inner i=new Inner();
  i.sum(10,20);
  i.sum(280,600);
}
public static void main(String s[])
 {
 Outer O=new Outer();
 O.m1();
  }
}
```

Output:
30
880

→ If write the method in inside class and the same inner should be in same Method.

→ In same class we can write the method.

→ In method inside we create the inner class and the object should create in same method.

Rules are in Non-static

* Writing an inner class inside and outer class method (both static (or) non-static) is called Method local inner class.
* An object for innerclass should be create inside that Method only.
* The inner class members can be access within that method only.
* the inner class contain only Instance Members but not static members.

* the method local inner class access all the members of its outer class.

* We should not be put static in before of the class.

Ex:

```
static class Inner
 {
 void sum()
 {
 ==
 }
}
```

X

We can be write static method before

Ex: static void m1()
```
{
class Inner
{
Void sum()
{
==
}
}
```

Static Rules

1) If a inner class is declared inside static method of outer class then the inner class can access only the static members of outer class.

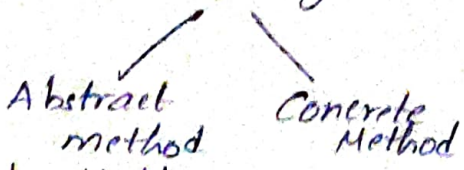
30/12/19 Unit-3

* Abstract classes
* Interfaces
* Inheritance
* final
* super
* packages
* Exceptional Handling

Abstract classes

A class which contains atleast one abstract method is known as abstract class.

Methods are two types.



Concrete Method:

A method which has an implementation (or) body is defined is called as concrete method.

Abstract method:

A method which does not have defination is called as abstract method.

Ex: for Concrete Method

```
Void display()
{

}
```

Ex: for Abstract method

```
abstract Void display();
```

abstract is a keyword in JAVA.

For class also we should preceed with abstract keyword. Atleast one abstract method in its abstract class and it may or may not contain concrete method., that can be any number of Abstract class in Abstract method

We should not create object instances for abstract class. Normally we create object for the classes, it is used to access the data members and data methods. Where as in abstract classes, we contain abstract method which do not contain any implementation detail so, we cannot access it through instance.

We should not create constructor for the Abstract class because constructor are directly called while creating an object.

Ex:

```
abstract class AbstractDemo
 {
  int a=10;
  Void display()
  {
  System.out.println(a);
  }
  abstract void fun1();
 }
```

concrete method ← the void display() block

abstract method ← abstract void fun1();

[The private]

The private members are accessible only upto that class, cannot access it in subclass.

Syntax

private datatype varname;

To inherit the parent class properties to the child class, we have to define the abstract methods present in the parent class in child class. orelse it becomes an abstract class.

Extends is keyword is used to inherit the parent class to its child class.

```
Class Mainclass extends AbstractDemo
 {
 Void fun1()
  {
  System.out.println("inside fun1");
  }
```

```
public static void main(String s[])
{
Mainclass mc=new Mainclass();
    mc.fun1();
    mc.display();
    System.out.println(mc.a);
    }
}
```

Output
inside fun1
10
10

We can access the parent class properties by creating the object in child class.

We can't create abstract data member in java. We can create only abstract methods & abstract classes.

```
Abstract class AbstractDemo
{
  int a=10;
  void display()
  {
    System.out.println(a);
  }
 abstract void fun1();
 abstract void fun2();
}
class Mainclass extends AbstractDemo
{
 void fun1()
 {
 System.out.println("inside fun1");
 }
}
class A extends Mainclass
{
 void fun2()
   {
 S.o.p(a);
```

```
}
public static void main(String s[]);
{
A a1=new A();
a1.display();
a1.fun1();
a1.fun2();
}
}
```

Output:

```
10
inside fun1
10
```

→ Although it is a super class, we cannot access the private data members and member methods, that are present in that class.

We can write static members inside an abstract class.

In concrete methods, we contain common features in it. whereas in abstract methods we write the uncommon features in it.

```
abstract class Domestic_Animal
{
Void no_of_legs()
{
S.o.p("Domestic Animal has 4 legs");
}
abstract void sound();
abstract void bark();
abstract void size_of_animal();
}
```

Interfaces:-

A class which contain all uncommon features in it.

Interface is a class which contains all abstract methods and do not contain any concrete methods

Syntax:

```
interface interfacename
{
// abstract methods
// constant data members
}
```

We can keep data members which contains a constant value in it.

We can provide the default methods and static methods in interface. after JAVA 8, before this version they do not have the facility to use this upto JAVA 7. they can use only abstract methods & constant data members in it.

Ex:

```
interface Sample
{
 abstract void display();
 abstrait void fun1();
 final int b=10;
}
```

A class will implement an interface.

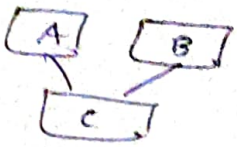
A class which implement the interface should define all the abstract methods of the interface.

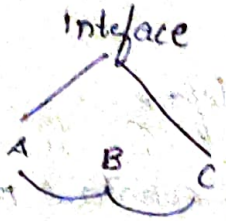
```
class InterfaceDemo implement Sample
 {
```

```
void display()
{
System.out.println("I am used to display");
}
Void fun()
{
System.out.println("This is functions");
}
}
```

In JAVA multiple inheritance is not possible. it throws an error. Through interfaces we can allow multiple inheritance

Multiple Inheritance





class B extends A,C ✗
class B implements A,C ✓

In JAVA after extends it shouldnt contain more than one class. A single can implement n' no.of classes

Ex: A B C D E
↓ ↓ ↓ ↓ ↓
I I I C C

I = interface
C = Normal class

class D implements A,B,C extends E ✓

It is possible because after extends we contain only one class.

An abstract class can implements the another class.

Program.

```
interface I
{
abstract void display();
abstract void fun1();
}
abstract class Abc implements I
{
void display()
{
System.out.println("inside display");
}
void fun1()
{
System.out.println("inside fun1");
}
abstract void f2();
}
```

We cannot create an object for the inteface.

In interface it should contain only constant data members.

In abstract class it can contain instance data member and constant data members.

Inheritance.

Aquiring the properties from Base class to derived class.

(or)

Inherit the properties.

Inheritance is OOPs feature where in one class [derived class] aquires all the properties of another class [Base class].

The class which inherit the properties from the another class is called Derived class/sub class/ child class.

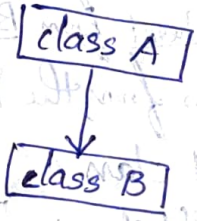
The class which provide the properties to the another class is called as Base class/ Parent class/ Super class..

Types of Inheritance

There are 5 types of inheritance those are

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance.
4. Heirarchial Inheritance.
5. Hybrid Inheritance.

Single Inheritance



class B is inheriting the Properties from class A

The derived class aquires the properties from the single Base class is called Single Inheritance.

Ex: Program:-

```
class Baseclass
{
int a=10;
float b=25.6;
void display()
{
System.out.println("the add is "+(a+b));
}
void func()
{
System.out.println("this is Base class");
}}
```

```
class Childclass extends Baseclass
{
void fun2()
{
System.out.println("this is derived class");
}
public static void main(String s[])
{
Childclass c=new Childclass();
c.fun1();
c.display();
c.fun2();
}
}
```

Output:-

This is base class
the add is 35
the is derived class

Properties that child class cannot inherit from Base class

- The constructor cannot inherit to child class from the Parent class
- Static blocks also cannot inherit to child class.
- Private member also cannot inherit to child class.

Ex. Program

```
class A
{
private int a=10;
A()
{
System.out.println("It is default constructor");
}
static {
System.out.println("It is a Base class");
}
```

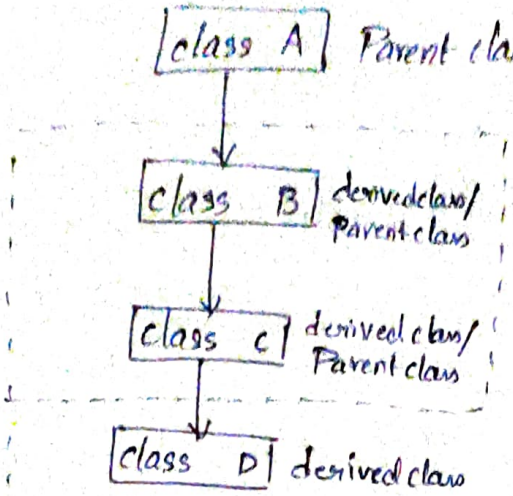
```
int b=30;
Void display()
{
int a=15;
int b=30;
System.out.println("the addition is"+(a+b));
}
}
class B extends A
{
int s=10;
Void fun1()
{
System.out.println(s);
}
public static void main(String args[])
{
B b1=new B();
Sop(b1.a()); // This is a private member
b1.display()
b1.fun1();
b1.l();
b1.s();
}
}
```

Output:

```
'a' cannot be accessed.
It is a Base class
It is default constructor
the addition is: 45
10
10
```

When we create an object for the child class then the Base class memory is allocated and it calls the no-argument constructor, if it is present or else it calls the default constructor.

Multilevel Inheritance



If a class in inheriting the another class which is a parent class for one more class then it is called Multilevel inheritance.

In Multilevel inheritance one class can acts as both parent and derived class.

In the above figure class B and class C are can be derived class as well as parent class.

Eg Program

```
class Baseclass
{
private int a=10;
void display();
{
System.out.println("this is a Super class");
}

Baseclass();
{
System.out.println("this is no-argument constructor");
}
```

```
static void fun1()
  {
  System.out.println("this is function 1");
  }
}
class Derived_Base_class extends Baseclass
 {
 static void fun2()
  {
  int l=10, b=20;
  System.out.println("the subtraction is "+(b-a));
  }
 Void display2();
  {
 System.out.println("this is a base class & derived class");
  }
 Derived_Base_class()
 {
 int m=15;
 System.out.println(m);
 }
 int n=30;
}
class DerivedClass extends Derived_Base_class
{
static void display3()
  {
  System.out.println("this is a sub-class");
  }
 public static void main(String s[])
 {
 Derivedclass d=new Derivedclass();
```

```
d.m1();
d.display2();
d.display();
d.fun1();
d.fun2();
d.display3();
}
}
```

Output

```
15
this is no argument constructor
30
this is baseclass & derived class
this is a super class
this is a function1
the subtraction is 10
this is a sub-class
```

Static keyword can be kept in sub-classes but should not precede with Outer-class.

Ex:

```
class Baseclass
{
 int a=10;
 void display()
 {
 System.out.println("inside display");
 }
}
class Derivedclass extends Baseclass
 {
 void display()
 {
System.out.println(" inside derived class display");
 }
 Public static void main(String args[])
 {
Derived class d=new Derivedclass();
Baseclass b=new Baseclass();
b.display();
d.display();
}
}
```

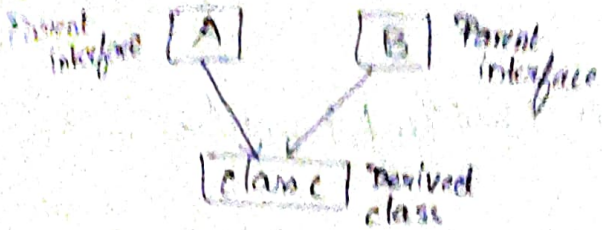
Output

```
inside display
inside derivedclass disp
```

the method can be overridden between the two classes the method which contains in Derived class can be overridden in sub class this is called Overridden methods.

If we put final keyword before any method it cannot be inherited to sub class.

Multiple Inheritance:



If the class inherit the properties from more than one parent interfaces then it is called Multiple inheritance

Note:

In java Multiple Inheritance through classes is not possible

Ex:

class c extends A,B ✗ Not possible

So, Multiple Inheritance in java can be achieved through Interfaces.

Ex: class c implements A,B ✓ possible.

Program

```
interface Parentclass1
{
default void displayes
{
System.out.println("inside display");
}
abstract void funtes
final int a=10;
```

```
}
interface Parentclass2
{
static void fun2()
{
System.out.println("Inside fun2");
}
abstract void fun3();
}
class Derivedclass implements Parentclass1, Parentclass2
{
Derivedclass d;
void fun1()
{
System.out.println("this is function 1");
}
void fun3()
{
System.out.println("this is function 3");
}
Public static void main(String args[])
{
Derivedclass d=new Derivedclass();
d.a();
d.display(); x parent1.display();
d.fun1();
d.fun2(); x parent2.fun2();
d.fun3();
}
}
```

Output

```
10
inside display
this is function1
Inside fun2
this is function 2
```

The concrete methods in interfaces cannot be accessed; they are accessed only by the interfaces name. by objects.

Only default and static methods are accessed by interface name. Where as data members can be accessed by interface name as well as object.

Program:

```
class parent1
{
 final int a=10;
   void fun1()
  {
 System.out.println("inside display");
  }
}
class parent2
{
 void fun1()
  {
 System.out.println("inside fun2");
 }
}
class Derivedclass extends parent1,parent2
 {
 void fun2()
  {
 System.out.println("inside fun2");
 }
 void fun3()
 {
 System.out.println("inside fun3");
 }
public static void main(String args[])
 {
Derivedclass d=new Derivedclass();
 d.fun1();
d.fun2();
 }
}
```

X This is not possible in Java because the parent classes are having the same method signature (fun1) and if the derivedclass object calls that method. the compiler will get confused. To which method a call should be. This is the reason JAVA does not support multiple Inheritance.

Heirarchial Inheritance:-

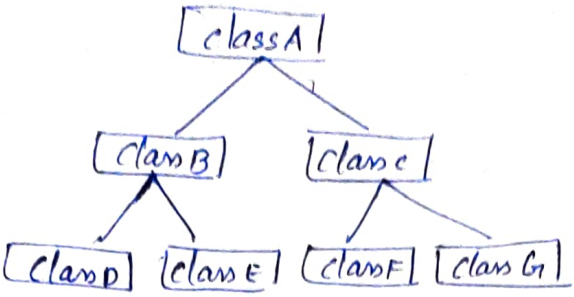


Fig:- Heirarchial Inheritance

If More than one class is derived from a single base class then it is called as Heirarchial Inheritance.

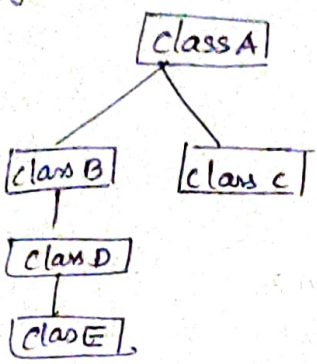
Example

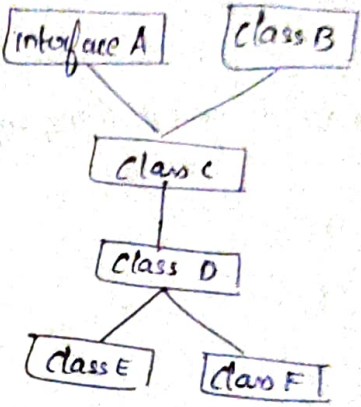
```
class A
 {
 Void fun1()
  {
 System.out.println("Inside fun1");
  }
 final int d=10;
  }
class B extends A
 {
 Void fun2();
  {
 System.out.println("Inside fun2");
 }
 }
class C extends A
 {
 Void fun3();
  {
 System.out.println("Inside fun3");
  }
 }
```

```
class Main
{
public static void main(String args[]);
{
B b = new B();
C c = new C();
b.fun2();
c.fun1();
b.d();
a.fun3();
}
}
```

Output:
Inside fun2
Inside fun1
10
Inside fun3

Hybrid Inheritance:-





Combining ~~more than~~ one or more models of inheritance to form a new type of inheritance is known as Hybrid inheritance.

Ex

```
class A
{
void fun1()
{
System.out.println("Inside fun1");
}
}
class B extends A
{
void fun2()
{
System.out.println("Inside fun2");
}}
```

```
class C extends A
 {
 void fun3()
  {
 System.out.println("Inside fun3");
 }
 }
class D extends B
  {
 void funD()
  {
 System.out.println("Inside D function");
 }
 }
class test
 {
 public static void main(String args[])
  {
 D d=new D();
 C c=new C();
 c.fun2();
 c.fun1();
 d.funD();
 d.fun3();
 d.fun1();
 }
}
```

Output

```
Inside fun2
Inside fun1
Inside D function
Inside fun3
Inside fun1
```

Super

Super is a reference variable. It contain immediate Parent class ~~super~~ object. ~~address~~

Uses of Super Keyword

1. There are 3 uses of super in Java.
2. ~~To access the data members of the~~ ✗
3. Super ~~used~~ to can be used to refer to the immediate parent class data members.

super.parentclass datamembers;

② super can be used to refer to the immediate parent class instance methods.

Syntax super.parentclass instance methods;

③ super can be used, to refer to the immediate parent class constructor [both no arg constructor & parameterized constructor]

Syntax: super();
(or)
super(parameters);

Ex:-

```
class A
{
 int a=10;
 char c='s';
void funA()
{
System.out.println(a+" "+c);
}
}
class B extends A
{
 int a=20;
 float b=10.36;
 char c='z';
 void funB()
 {
 System.out.println("Inside function B");
 }
public static void main(String args[])
{
 B b1=new B();
System.out.println(super.a+" "+super.c);
System.out.println(b1.a);
```