Object oriented programming - fundamentals.

Java: => compiler & interpreter based lang

* It is a simple & Prgful OOPL & it is III to C++.
* Conceived by James Gosling. birth => Sun Microsoft s/m.
* It is a platform independent Pgmmg lang => compile once & run anywhere
*

When Java is compiled, it is not compiled into platform specific m/c.

for Eg: * If you have written some code in Java & you have compiled that, you can execute the same code on any platform.

* The only thing you need is to install the corresponding JVM => Environment in which Java Pgms execute.

* So the .class files can be copied to any m/c running a JVM & they will work.

Javac => Compiler that converts source code to byte code.
JVM => interpreter " " byte code to m/c lang code.

Java src code is converted into byte code (.class file) after compilation the (.class file) is interpreted & converted to the m/c langu code by using JVMs.

Comment line:

```
class Welcome => class is a keyword to def
{
Public Static Void main (String args[]).
{
System.out.println ("Welcome to Java forograms").
}
}
```

OOPS UNIT 4

Class Add
{
int Num1, Num2, Sum;
Void Init_num (int N1, int N2)
{

Num1 = 10
Num2 = 20
Sum = 30

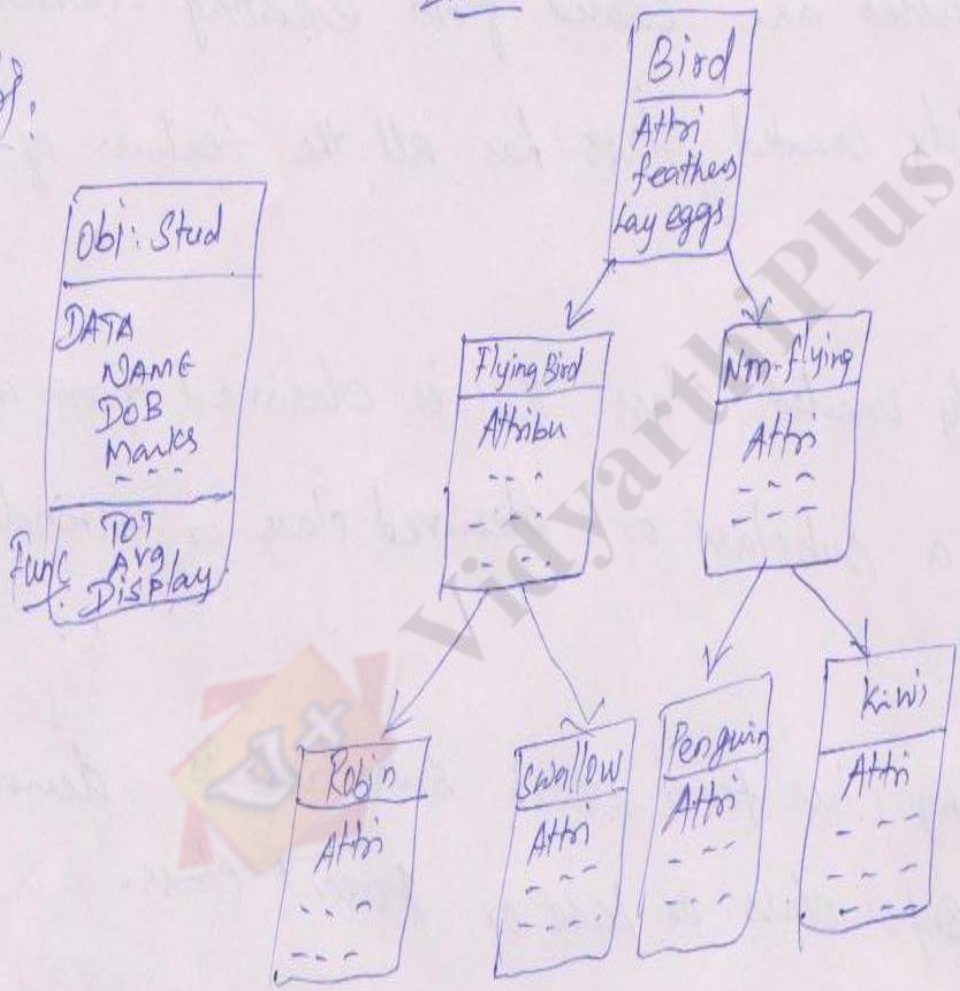
Num1 = 1
Num2 = 2

Num1 = 10
2 = 20
Object Number = 1 Obj Num = 2.

Obj.

Obj: Stud
DATA
NAME
DOB
Marks
Func
TOT
Avg
Display

Inhen

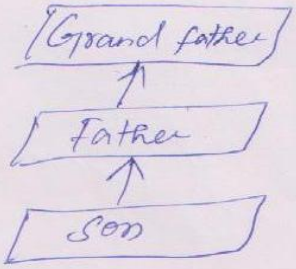


Inheritance:

* process of creating new objects by acquiring the pptys of the existing objects w/o affecting it.
* new classes are created from existing classes.
* The newly created class has all the features of the existing class
* The newly created class that is derived from another class is called a subclass or derived class or extended or child class.
* The existing class from which subclass is derived is called a super class or base or parent class.

General form:

```
[access specifier] class derived_class_name extend Super class
{
  // implementing code
};
```



Classes & objects:

* classes are templates to create objects.
* They define the structure of the obj.
* class is a collection of fields (data) & methods (function) that operate on that data.

General form:

```
[access_specifier] class class_Name[extends Super_class]
{
  field declaration
  Method  "
}
```

Packages:

↳ collection of related classes.

* Normally classes and interfaces are inherited within a class.
* This induces the reusability within the programs.
* But to create a class in more than one ~~package~~ program that should be declared in package.
* Two types of package:
  - → User defined
  - → predefined (Built in Package)

Pre defined package:

* It contains more that 60 Java * packages. Eg: java.lang, java.util, java.io, java.awt, java.net, java.applet & javax.swing are most commonly used packages.
* These packages are used to the internal representation of our pgms & data.

User defined package:

```
Package package name;
```

Packages also provide access protection. These packages are also act as a Containers of classes.

Advantages of using package:

* classes can be easily reused.
* Two classes in diff. package can have same name.
* Packages provide way to hide classes.

Packaging also help us to avoid class name collision when we use the same class name as that of other.

Step 1:

```
class c1
{
Public void m1()
{
S.O.P("Method m1 of class c1");
}
PSVM ( )
{
c1 obj = new c1();
obj.m1();
}
}.
```

```
Package Mypackage;
class Hello world
{
PSVM ( )
{
S.O.P("Hello world");
}
}
```

1. Create a My package directory in bin directory.
2. Compile above Src file as Hello world.Java.
   ie c: javac Helloworld.Java.
3. Put the class file Hello world class in "Mypackage" directory.
4. Run package => java Mypackage.Hello world.

OOPS UNIT 4

```
Public class ComplexNumber
{
int real;
int image;
Public ComplexNumber() { } //default constru
Public ComplexNumber(int r, int i) //Param Cons.
{
real = r;
image = i;
}
Public void ComplexAdd(ComplexNumber c1, ComplexNumber c2)
{
real = c1.real + c2.real;
image = c1.image + c2.image;
}
Public void view()
{
S.O.p(real + "+i" + image);
}}
```

```
import Complex.ComplexNumber;
class Comp
{
psvm( )
{
ComplexNumber a = new ComplexNumber(3,2);
"  b = new "  (1,3);
a.view();
b.view();
ComplexNumber c = new ComplexNumber();
c.view();
}
}.
```

java Comp.

o/p
3+i2
1+i3
4+i5

* Create a sub directory in the name of the Pkg. Nd complex.
* Enter into the created directory by using the following command

cd Complex.

You will get the prompt like the following

C:\Java JDK1.6.0\bin\Complex> edit complexNumber.java

java Pkgname.clsname

javac complexNum.java

Suppose we have a src file called ... to put this file in a package "mypackage" then the following code.

```
Pakge mypackage;
class Helloworld {
Psvm ( ) {
S.o.p ("HW");
} }
```

Create a directory "mypackage"
Save the src file as Helloworld.java in the created directory
Set the class path as set CLASSPATH=
Goto the "mypkg" directory & compile the pgm as

C:\mypakage > javac Helloworld.java

Run the program. Error will occur. ← java Helloworld

This is bcoz the class Helloworld belongs to the package mypackage.mypackage.Helloworld.

Field Comments:

This adds description to the public fields, generally that means static constants.

Eg

```
/**
 * max_count for objects
 */
Public static final int max_count = 10;
```

Package & Overview Comments:

* class, method & Variable comments are placed directly into Java src files. delimited by /** .... */.

* But to generate packet comments, a separate file is to be added in each package directory.

Method 1:

An HTML file named package.html is created. All the text between the tags <body> --- </body> are extracted.

Method-2:

* A java file named package-info.java is created & supplied.

* The file must contain an initial javadoc comment with /** &*/ followed by a package stmt.

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Changecase:

```
class changecase
{psvm ( )
{String s = "Test of string";
String sl = s.toLowerCase ();
String su = s.touppercase ();
S.O.P ("Actual String = "+s);
S.O.P ("UpperCase = "+su);
   " ("Lower case = "+sl);
}}
```

Substring:

1) Substring (int n) => Substring from the nth character.
2) Substring (int m, int n) => it gives the mth character to the nth character Not included the nth character - The total num of character is n-m.

Eg:

```
Class Subtest.
{
 PSVm( )
{
String test = "abcdefgh";
S.O.P ("Given String is" + test);
S.O.P ("Substring from 3 to end" + test.Substring (3));
"("Substring from 0-3" + test.Substring (0,3));
"("  "  "  2-4" + test.Substring (2,4));
("  "  "  3,3 }}
```

O/P
Given stg: abcdefgh
3 to end: defgh
0-3 : abc
2-4 : cd

Array:

Collection of similar data types that share a common name. All the variables of the array can have the same name. Each elt in an array can be accessed by using the index number or sub script. In Java array is an

1) declaration
2) Allocate m/r (Instantiation).

Declaration:

* An array can be declared by specifying the data type followed by [] with array name. Array declaration only declares the name of an array.
* No m/r space is alloted. So we cannot use it before instantiation.
* Size should not be given to an array at the time of declaration.

General form:

```
data type arrayname [];
int mark [];
```

Instantiation:

* process of giving initial values to the array elts.
* It can be done in 2 ways.

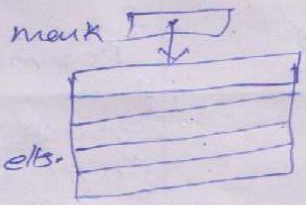
General Form:

After declaration or at the time of declaration.
Instantiation is nothing but allocating space to an array.
This is done by using new operator.
once an array is instantiated. Size cannot be changed.

General form:

```
array_name = new data_type [size];
```

Eg: `mark = new int [5];`



Initializing an arrays:

* process of giving initial values to the array elts.
* 2 ways.

Method 1: Initial values are given to the elts at the time of declaration. In this method no need to use new operator to allocate space to the array.

```
data type arrayname [] = {initial values};
int mark[] = {1, 2, 3, 4, 5};
```

ADDITIONAL ANSWER BOOK

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Method 2:

* Initial values are given to the elts by accessing each elt using the index number.
* Here maximum size of array must be stated in advance.

Syntax: array name [index] = initial value;

mark [3] = 4;

Types of array:

* Single
* Multi

Single dimensional Array:

```
class array
{
PSVM ( )
{
int regno [ ] = new int [5];
S.O.P("Default value");
for(int i=0; i< regno.length ; i++)
S.O.P (regno[i]);
//Store the value
for (int j=0; j < regno.length ;j++)
regno[j] = j+100;
for(int k=0; k<5; k++)
S.O.P ("k = "+k+"."+regno[k]); }}
```

O/P.

Default value.

0
0
0
0
0

k=0 100
k=1 101
k=2 102
k=3 103
k=4 104

OOPS UNIT 4

```
{
PSVM( )
{int marks[] = new int [5];
S.O.P ("Elts are " + mark.length);
S.O.P("Default values" + marks[0]);
marks[0] = 50;
marks[1] = 60
       2 = 70
       3 = 80
       4 = 90
S.O.P("value of 1st elt: " + marks[0]);
" ("\n print the values in a single row");
for (int i=0; i< marks.length ; i++)
 {
  S.O.P(marks[i]+ "\t");
 }
}
}.
```

o/p.

Elets are: 5

Value of 1st elt: 50

print the value in a single row

50 60 70 80 90.

2D array.

```
Class TwoDarray {
PSVM( ) {
int twoD[][] = new int [4][5];
int i, j, k=0;
for(i=0; i<4; i++)
for (j=0 ; j<5; j++) {
twoD[i][j] = k;
k++;
}
for(i=0; i<4; i++) {
for (j=0; j<5; j++)
S.O.p(twoD[i][j] + " ");
" ();
}}}.
```

	1	2	3	4
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

```
class day
{PSVM( )
{
String days[] = {"Sun", "Mon", };
for(int i=0; i<7; i++)
{
S.O.P((i+1) + " " + days[i] + "DAY");
}
}
```

One dimensional Array:

```
class day
{
 Psvm ( )
 {
String days [] ={"SUN", "MON", "TUES", " ", ...};
for (int i=0; i<7; i++)
{
 S.O.P(("i+1)+" "+days[i]+"DAY");
}}
}
```

i=0 0<7 i=1.

O/P
1 SUNDAY
2 MONDAY

Two Dimensional array:

```
class unit
{
 Psvm ( )
{
int matrix [][] = {{1,0,0},{0,1,0},{0,0,1}};
S.O.P ("\t 3x3 Unit matrix");
for (int i=0; i< matrix.length; i++)
{
for (int j=0; j< matrix.length; j++)
{S.O.P ("\t"+matrix[i][j]);
}S.O.Pln ( );
}
}.
```

3x3 unit matrix.

1 → a[0][0] a[0][1] a[0][2]

2 → a[1][0] a[1][1] a[1][2]

a[2][0] a[2][1] a[2][2]

Constructor:

→ Special type of method to initialize an object.

→ When an object is instantiated, the method is invoked automatically.

→ Constructor do not have any return type.

→ A constructor may be defined with i/p or o/p arguments.

→ A constructor is normally invoked by using new operator.

→ It has the same name as the class name.

Syntax:

```
[access specifier] class_name ([args])
{ stmts;
}
```

Eg:

```
Class sample
{ int Num;
  Sample()
  {
   Num = 0;
   S.O.P("Default Constructor");
  }
Sample (int N)
{ Num = N
  System.out.println("Parameterized Constructor");
}
 Void Print()
{
S.O.P("Number is" + Num);
}
```

```
" S2 = new Sample (10);
S1. print ();
S2. print ();
}}
```

Types of Constructor:

* Default Constructor.
* Parameterized "

Packages:

* Packages are containess for classes that group the classes based on their functionality.

*Importing Classes:

Method 1: java.util.Date birthday = new java.util.Date();

Method 2: import package1[.package2]... [.class_name];

Static imports:

If import static java.lang.Math.* is included, the following statement is enough to compute $\sqrt{x^2+y^2}$.

Sqrt (Pow(x, 2) + Pow (y, 2))

no need to use like the following.

Math. Sqrt (Math. Pow (x, 2) + Math. Pow (y, 2))

Eg:

```
import mypackage.*;
class Myprog
{
PSVM ()
{
Myclass Myobj = new Myclass ();
My obj. print (10);
```

Accessing fields:

* Fields of a class can only be accessed by normal methods of its own class.
* The methods of other classes & static methods need obj to access the member data.
* Dot (.) operator is used b/w obj name & field name.

Syntax:

```
Obj_name . field_name;
```

Eg: A.num1 = 10;
A.Num2 = 20;

Accessing methods:

Syntax:

```
obj_name . method_name (actual_arguments);
```

```
A. Init_num (10, 20);
A. Add_num ();
```

Implicit/Explicit arguments:

* Methods of a class receive two types of argument while calling objects.
* Obj_name in a method call is the implicit arg.
* Argument passed b/w the parenthesis are explicit argument.

```
A. Init_num (10, 20);
```

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

OBJECTS:

* It is a block of m/r that contains space to store all the instance variables.
* once a class has been defined, any num. of obj can be created in that class.
* It contains 3 key characteristics:
  - → Identity - Student (Object)
  - → Behaviour = Good, poor, medium. Method.
  - → state - Field function: Total, Avg, Grade.

Eg Add A.
obj variable.

Creating Objects:

Obj are created using new operator..

Gener form for obj declaration,

Declaration : class name obj name;

Instantiation : obj name = new class name (actual arg list)

Add A; //declares the obj

A = new Add (); // instatiates the obj

First instruction declares the obj.
Second " allocates m/r & returns reference to the obj A.

Add A = new Add ();

Add B = A // both pts to

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Access Specifier:

* Used to enable or restrict access to the member of a class from outside.
* Also known as Visibility modifier.
* 3 types.
  - → Private
  - → protected
  - → Public.

Private:

* Fields & methods that are to be highly safe are declared with private specifier.
* only object of the same class can access a Private Variable or method.
* If you can declare in Constructor, it is accessible only from its own class.
* This is the highest degree of protection.
* They cannot be inherited by subclass.

```
Eg: Private int Sense;
    Private void notaccess() { }.
```

Public:

* It is accessible from all other

```
Eg:
Public class myclass // class with public
{ Public Boolean flag;
  Public void test public ()// method.
  { }
```

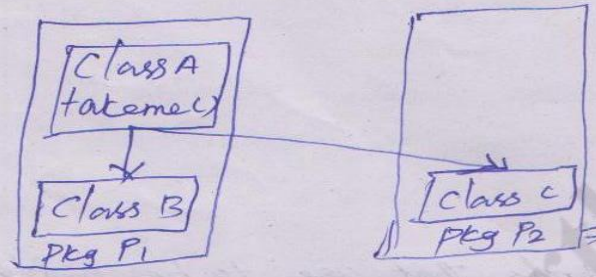
* the public is an access specifier. So the main m as unprotected & therefore making it accessible to all cl class.

* only one public class can be defined in each source file.

protected:

Variables, methods & inner class are declared protected it is accessible only from within its own class & its subclass.

Eg: protected int protect_var;
protected void finalize () throws Throwable { }



class A ⇒ superclass.
class B & C are derived from class A.

⇒ collection of related classes.

Access specifier in class A	class B	class C.
protected takeme()	Accessible in sub class B	Accessible in C [even it is in other pkg P2].
takeme() [w/o an AS]	Accessible it in the same pkg.	Not accessible in diff pkg.

private ⇒ It is accessible only from within own class.
protected ⇒ It is " from within its class & its sub class.
Public ⇒ " " " from all classes.

The other such modifiers are used to determine how the data members & methods are used in other class & objects.

Static means there will be only one instance of that object.
If an obj is not declared as static a new copy of that object is made for every instance of that class that

ADDITIONAL ANSWER BOOK

(3)

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Static Members:

* Members in a class can be declared with static keyword as prefix.
* Static member can be accessed directly using the class name.
* ~~An object is~~ needed to access a static member data.
  * static field
  * static method.

→ only one instance is created & that is shared by all the objects.

Syntax:

[access specifier] static data type field name;

```
class mainclass
{
public int Num1, Num2, sum;
static int Count = 0; //static declaration.
Void Init_num (int N1, int N2)
{ Num1 = N1;
  Num2 = N2;
}
Void Add_num ()
{ sum = Num1 + Num2;
}
Void point_num ()
{
```

```
mainclass A = new mainclass();
mainclass.Count++;
A.Init_num(10,20);
A.Add_num();
A.Print_num();
mainclass B = new
mainclass.Count++;
B.      (3,8);
{
}.
```

Num1 = 10
2 = 20
Sum = 30
Count = 1
Num1 = 3
2 = 8
Sum = 1
Count = 2

Static Method:

[Access specifier] static return type method name (formal argu..)

```
{stmt;
 return value
}.
```

Eg:

```
class Add
{
static int Num1, Num2, Sum;
static void Init_num(int N1, int N2)
{
  Num1 = N1;
  Num2 = N2;
  }
Static Void Add_num()
  {Sum = Num1 + Num2;
}
```

```
class mainclass
{
PSVM( )
{
Add.Init_num(10,20);
Add.Add_num();
Add.print_num();
{{.
```

Documentation Comments:

A Javadoc comment precedes any class, interface, Method or field declaration and is similar to a multi-line comment except that it starts with a forward slash followed by asterisks (/**).

Types of Comments:

Class Comments:

```
Eg: /**
    * This class is for calculator.
    */
```

Method comments:

```
/**
* This method return the square of number.
*/.
Public double square (double number)
{return num * num;
}.
```

Arrays:

* Array is a Collection of similar data type values that share a Common name.

* Each element in an array can be accessed by using the index number or Subscript.

Eg: mark [2]

* Instantiation.

Array Declaration:

method 1: data_type array_name [];
method 2: data_type [] array_name;

Eg:
int mark [];

Array instantiation:

array_name = new datatype [size];

Eg: mark = new int [5];

Array length:

array_name.length.
size = mark.length;

Anonymous Arrays:

array_name = new datatype [] {values_list};

Mark = new int [] {10, 20, 30, 40, 50};

Array copying:

destination_array = source_array;

Copy of () method:

array_name = Arrays.copyof (existing_array, array_size);

Num = Arrays.copyOf (num, 10);

Enhanced for() loop.

for (Type Variable: Identifier)
{ stmts;
}.

// replace all occurances of x

in a string replaced by using the method replace().

Removing spaces in the edges:

Spaces in the begin & endings can be removed by using the meth trim()

newstr = oldstr.trim();

Comparing Strings:

Strs are compared by using 4 methods. equals(), equalsIgno CompareTo() & CompareToIgnorecase(). case(),

Eg:

Str1.equals(Str2); if str1 & 2 are = return true else false.

" .equalsIgnorecase(Str2); Same as above but ignores the case.

" .compareTo(Str2); //Str1 = Str2 = 0, +ve, -neg

" Ignorecase();

Finding string length

To get the length of a string length() method is used.

Eg:

int len = Str.length();

Multi Dimensional Array:

```
int [][]a;
  (or)
int a[][];
int a[][]= new int[2][2];
```

Ragged Arrays:

```
int a[][] = new int[2][];
a[0]: new int[5];
a[1] : new int[10];
```

Strings:

Eg: 1) String s= new String ("Sunil Kumar");

2) String s;
   s= new String (" ");

3) String s= "Sunil" + "kumar";

Operations on String:

String Concatenation:

Eg:
```
String s= "Sunil" + "kumar";
```

Substring Extraction:

```
String s = new String ("sunilkumar");
String n1 = s.Substring(0,5); //not including 5th character.
"  n2 = s. "  (5); "Sunil" will be stored in n1. "kumar" in n2.
```

Converting to lowercase:

```
Newstr = Oldstr.toLowerCase();
```

Converting to uppercase:

```
Newstr = Oldstr.toUpperCase();
```

Stg char can be converted into lowercase letters by using tolowercase() method.

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Strings:

* In java strings are the collection of unicode characters.
* Strings are handled as objects. These objects are created using string class which is defined in the java std lib.
* Java string is not Null terminated.

Eg:

Operations on String:

String Concatenation:

+ Sign can be used to concatenate 2 strgs. During

Eg:

Substring Extraction:

Substring from a large string can be extracted using Substring() method.

Eg:

Register No. : _____ Page No. : _____

Signature of the Hall Invigilator (with date)

Finalize Method:

* It is similar to destructor in c++.
* But manual m/r reclamation is not needed bcoz Java does automatic garbage collection.
* In some situations, objects utilize a resource other than m/r. Such as a file or objects that uses S/m resources.
* Finalize method must be defined in these situations to free the resources when it is no longer needed.
* Finalize method will be called before the garbage collector removes the object.
* only one finalize can be defined to a class.

Eg:

```
protected void finalize() throws Throwable
{ try
  { close();
  }
  finally
  { super.finalize();
  }
}.
```

Static:

Purpose of using Constructor, destructor, Garbage collection

* Static method use no instance variables of any object of the class they are defined in
* When we use static we don't need to create any instance of that class.
* Static as a global function: You define something static when you don't need to create an instance of that object.
* static keywood refers to the m/r allocation in Java. It means if a variable declared as static it only have one m/r space to store data.

Diff b/w Static & Dynamic

* When a method (or variable) is declared as static it belongs to the whole class.
* When it is not declared as static it belongs to each obj of the class.

Garbage Collection:

* When we run program, it uses m/r by storing variables, fns or obj. After using these, they remain in the m/r unused.
* So a subroutine runs & clears unused m/r from these objects. Now this m/r is available for other pgms.
* This subroutine is called gc.
* G/c is the process of automatically detecting m/r that is no longer in use, & allowing it to be used again.

Constructor:

It is used to do something automatically when you create a new obj of a class. for Eg: if you want that every obj of a class have its initial value 0 you can do this using cons. method. Or if you want whenever you make a new obj a msg is given that " " you can do this using constructor.

Destructor:

* The purpose of a destr is usually to clear off unused Variable & clean up the m/r.
* Java has built in m/r handling mechanism that clear off unused [illegible] called G/c

GENERAL FORM OF CLASS DEFINITION ①

```
[access specifier] class class_name [extends Super_class]
{
   [Fields declaration]
   [Methods definition]
}
```

Eg:

```
import java.io.*;
class Add
{ int Num1, Num2, Sum;
  Void Init_num (int N1, int N2) // Method definition.
{ Num1 = N1;
  Num2 = N2;
}
  Void Add_num ()
  { Sum = Num1 + Num2;
}
  Void Print_num ()
{
  System.out.println ("Num1 = " + Num1);
  System.out.println ("Num2 = " + Num2);
  System.out.println ("Sum = " + Sum);
}}class mainclass
{
Public static void main (String args[])
{
  Add A = new Add ();
   A.Init_num (10, 20)
   A.Add_num ();
   A.Print_num ();
  }
}
```

O/P:

Two steps
* Field declaration
* Method definition

General Form of Field declaration:

[access specifier] data_type field_name;

Eg:
```
Class Add
{ int Num1, Num2, sum; //Fields
  ....
}
```

Types of Field initialization
* Explicit initialization method
* Initializing with Constructors method
* Initializing block method.

Explicit Field initialization:

Eg:
```
import java.io.*;
Class Add
{ int Num1=0, Num2=0, Obj_num=Assign_id();
  Static int count;
  {
    Num1 = N1;
    Num2 = N2;
  }
  int Assign_id()
  { return ++count;
  }
```

```
{
  System.out.println("Num 1 = " + Num1);
  System.out.println("Num 2 = " + Num2);
System.out.println("Object number = " + Obj_num);
}}
class mainclass
{
  Public Static Void main(String args[])
  {Add A = new Add(1,2), B = new Add(10,20);
  System.out.println("object A");
  A.print_num();
  System.out.println("\n Object B");
  B.Print_num();
}}
```

Constructor Field initialization

Eg:

```
import java.io.*;
class Add
{int Num1, Num2, Obj_num;
Static int count;
Add(int N1, int N2) // Constructor
{Num1 = N1;
 Num2 = N2;
 Obj_num = ++count;
}
Void Print_num()
{System.out.println("Num 1 = " + Num1);
                  ("Num 2 = " + Num2);
                  ("Object number = " + Obj_num);

}}
```

```
Class MainClass
{
  Public static Void main (String args[])
  {
   Add A = new Add(1,2), B = new Add(10,20);
   System.out.println("Object A");
   A.Print_num();
   System.out.println("\n Object B");
   B.Print_num();
  }}
```

Initialization blocks

```
Class Add
{
 int Num1, Num2, Obj_num;
 Static int Count;
 //initialization block
 {
   Num1 = 0;
   Num2 = 0;
   Obj_num = ++ Count;
 }
 Add (int N1, int N2)
 {
   Num1 = N1;
   Num2 = N2;
 }
 Void Print_num()
 {System.out.println("Num1 = "+ Num1);
                ("Num2 = "+ Num2);
                ("Object number = "+ Obj_num);

 }}
```

this Keyword

Eg:

```
Void change_val(int Num1, int Num2)
{
  this.Num1 = Num1;
  this.Num2 = Num2;
}
```

Methods:

General Form

```
[access specifier] return_type method_name(formal_arg_list)
{ Statements;
  [return value]
}
```

Refer Eg: page no: (1)

Eg:

```
Class Add
{ int Num1, Num2, Sum;
  Void Init_num(int N1, int N2)
{
```

——— X ———

Types of Method:

* Mutators
* Accessors

Mutators:

Eg:

```
Public void change(int n1, int n2)
{ Num1 = n1;
  Num2 = n2; //Num1 & Num2 are the fields.
}
```

Accessors:

```
Public int Get_Num1()
{ return Num1;
}
```