Basic Concepts

JDK Installation

Loops.

- for loop
- while loop
- do while loop
- enhanced for loop

if else.

Switch Case

Variables

methods.

Oops:→

→ Method overloading

→ Constructor.

→ Constructor overloading.

→ Inheritance.

→ Method overriding.

→ Abstract Class.

→ Interface.

→ Type Casting.

Polymorphism:→

Abstraction

Encapsulation

Packages.

imports.

Design patterns.

Object class

String class.

Arrays.

Boxing & Unboxing via wrapper classes.

Collections API

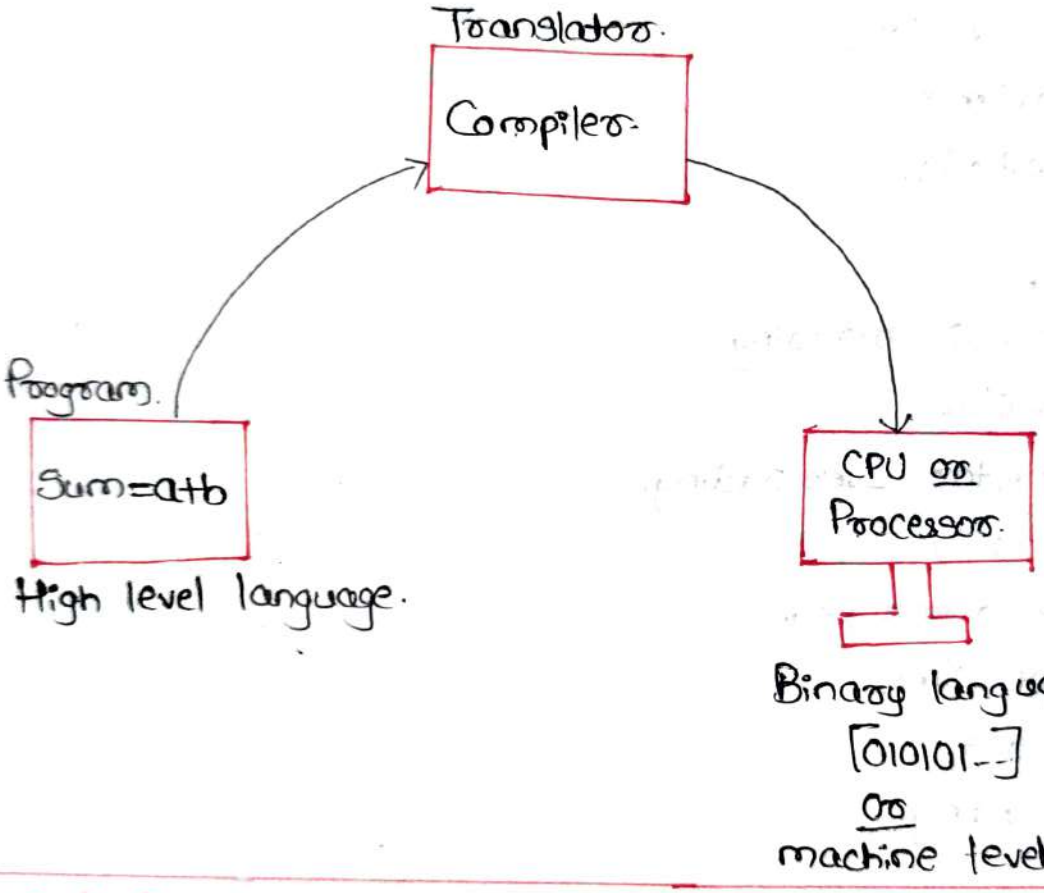
Generics

Exception handling.

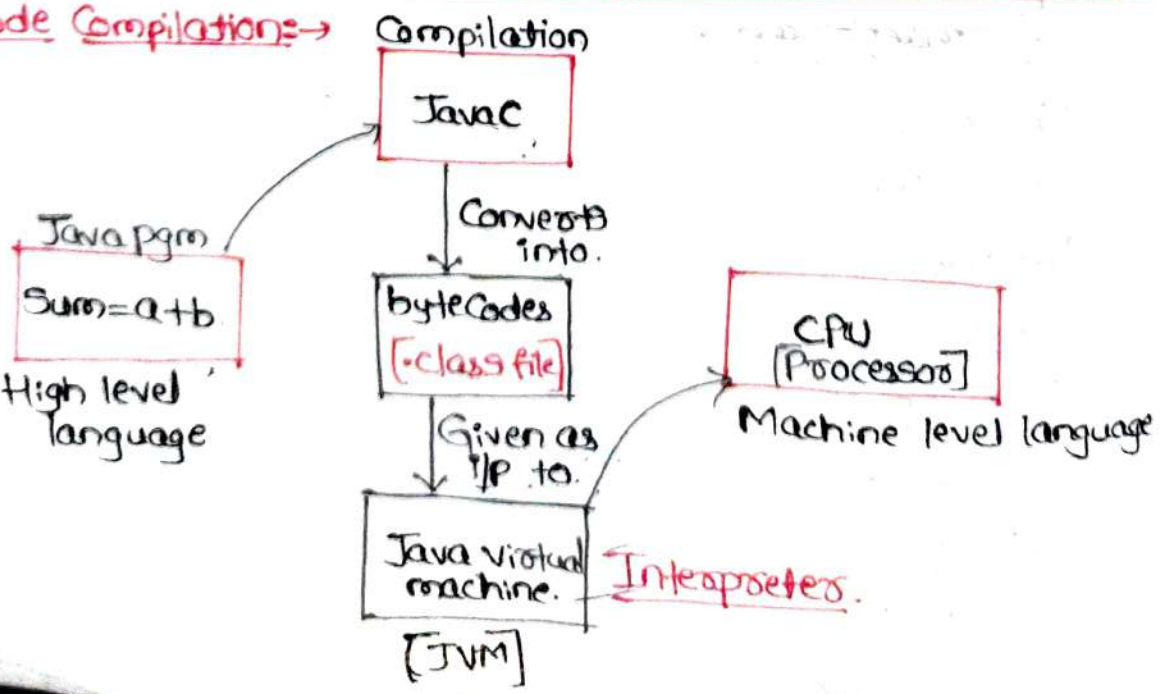
File handling.

Multithreading.

Compilation.



Java Code Compilation:→



Program:→

It is the set of instructions.

Java pgm:→

Set of instructions written according to java syntax. [Rules]

There are 3-stages of java pgm development:→

1) Coding or Composition.
2) Compilation.
3) Execution.

1) Coding or Composition:→

Developing a pgm according to java syntax. is called as Coding.

NOTE:→

All the java pgms should be saved with filename.java.

2) Compilation:→

Converting the java pgm into byteCodes. using javaC Compiler is called as Compilation.

NOTE:→

All the bytecodes will be stored in .class file.

3) Execution:→

Converting the bytecodes of .class file into machine level language. or binaries. using JVM [Java Virtual machine]

NOTE:→

JVM is interpreter.

Java

→ Developed by James Gosling.

→ Ist Sun

But now Oracle

> Java is open source & Platform independent.

Java 8 is a recent version.

Types of java :→

JSE [Java Standard Edition]

JEE [Java Enterprise edition]

JME [Java micro edition]

JDK

[Java Development Kit]

↓

Javac

libraries.

JRE

[Java Runtime Environment]

↓

JVM

Platform

libraries

→ Steps to set the java path in slm environment variable "path".

Go to C-drive.
↓
Program files.
↓
Java folder.
↓
Jdk1 folder.
↓
bin.
↓
Copy the path [Click on address bar & Copy the path.]
↓
Close.

→ How to reach slm environment variable "path".

Click on start
↓
right click on my Computer.
↓
Click on advance slm settings.
↓
Click on environment variable button.
↓
In slm variables.
↓
locate the variable path in slm variables section & click on edit button.
↓
** Go to the end of the line [variable value field]
↓
Paste the Copied path.
↓
Click on ok.
↓
ok
↓
ok.

Command prompt :→

Click on run or type as cmd

```
javac -version↵
javac 1.7.0_21

java -version↵
java version "1.7.0_21".
```

Editors

1) Notepad.
2) Notepad++
3) Editplus.

IDE [Integrated Development Environment]

1) Netbeans
2) Eclipse.

Pgm-1:→

```
Class Sample
{
Public Static void main(String args[])
{
System.out.println("Hello world");
}
}
```

Execution

General Syntax for Compilation.

```
javac filename.java↵
```

Execution Syntax.

```
java classfilename.↵
```

Tokens

1) Keywords
2) Identifiers
3) Literals.

Keywords.

Predefined words which have some Predefined meaning.

Ex:→
Class, public, Static, void, main, true, false, int.

Rule:→
1) Should be in lowercase.

Identifiers

Programmer defined words.

Ex:→
Pgm name, function name, class name, variable name.

Rules:→
1) Alphanumeric.
2) Can't start with the nos.
3) No special characters except '$', '_'.
4) Keywords can't be identifier.

Literals.

Values used.

4-types of literals.

1) Numeric literals.
Ex:→ 10
10.5.
2) String literals.
Ex:→ "Hello".
3) Character literals.
Ex:→ 'a',
'A',
'0'
'1'
'90' x
4) Boolean literals
Ex:→ true, false.

1) Java is case sensitive pgming language.
2) There are 3-imp tokens.
1) Keywords
2) Identifiers
3) Literals.

1) Keywords:→

Keywords are the predefined words in the Pgming language which will have some predefined meaning

Ex:→ class, public, static, void, int, float, double.

Rule:→

1) All keywords should be written in lowercase.

2) Identifiers:→

Identifiers are programmer defined words or The words for which programmer will give some kind of meaning. are called as identifiers

Ex:→

Class name, variable name, function name, interface name, package name, annotation name

Rules:→

1) These can be alphanumeric.
2) No special characters allowed apart from $ & _.
3) Though it is alphanumeric it can't start with no. it either start with alphabet, _, $.
4) Keywords can't be identifiers.

3) Literals:→

Literals are values used in the pgm.

There are 4-types of literals.

1) Numeric literals.
2) Character literals.
3) String literals.
4) Boolean literals.

1) Numeric literals:→

Any numeric values. Considered as numeric literals

Ex:→ 10, 10.5 etc.

2) Character literals:→

Any thing which is enclosed within single quotes Called as character literals.

& We should enclose only one character within the single quotes.

Ex:→ 'A', 'g', '7', ' ', '90'

↓

Invalid character literal

3) String literals:→

Anything which is enclosed within double quotes is considered as string literals.

Ex:→

"Hello", "1000", "Hello1000", "+-*", "g", " ",

"",

↓

Null string.

4) Boolean literals:→

There are only 2-boolean literals.

1) true
2) false.

System.out.println:→

Using System.out.println we can print the literals directly.

```
Class Sample1
{
  public static void main(String[] args)
  {
    System.out.println("welcome");
    System.out.println(100);
    System.out.println(100.5);
    System.out.println('A');
    System.out.println(true);
    System.out.println(false);
  }
}
```

O/P:→

welcome
100
100.5
A
true
false

③

```
Class Sample3
{
  public static void main(String[] args)
  {
    System.out.println("I LOVE INDIA"+"
                        AND ALSO MANDYA");
  }
}
```

O/P:→

I LOVE INDIAAND ALSO MANDYA

(or)

```
Class Sample3
{
  public static void main(String[] args)
  {
    System.out.println("I LOVE INDIA"+" "+"
                        AND ALSO MANDYA");
  }
}
```

O/P:→

I LOVE INDIA AND ALSO MANDYA

④

```
Class Sample4
{
  public static void main(String[] args)
  {
    System.out.println(100+10);
    System.out.println(100-10);
    System.out.println(100*10);
    System.out.println(100/10);
  }
}
```

O/P:→

110
90
1000
10.

⑤

```
Class Sample5
{
   Public Static void main(String[]args)
  {
    System.out.println("Hi"+10);
  }
}
```

O/P:→

Hi10

"Hi" + 10
↓
"Hi" "10"
↓
"Hi10".

Imp

⑥

```
Class Sample6
{
 public static void main (String[] args)
 {
  System.out.println ("Hi"+10+10);
  }
}
```

O/P:→

Hi1010.

"Hi"+10+10.
↓
"Hi10"+10.
↓
"Hi1010".

(7) Class Sample7

```
{
  public static void main(String[] args)
  {
    System.out.println("Hi:"+(10+10));
  }
}
```

O/P:→

Hi:20.

(8) Class Sample8

```
{
  public static void main(String[] args);
  {
    System.out.println("Hi"+10");
    System.out.println("Hi:"+10.5);
    System.out.println("Hi:"+true);
    System.out.println("Hi:"+false);
    System.out.println("Hi:"+'A');
    System.out.println("Hi:"+null);
  }
}
```

O/P:→

Hi10
Hi:10.5
Hi:true
Hi:false
Hi:A
Hi:null.

NOTE:→

→Using System.out.println we can print the result of a valid expression.

→

```
class Sample9
{
  public static void main(String[] args)
  {
    System.out.println("Hello world");
  }
}
```

O/P:→

```
javac Sample9.java↵
java Sample9↵
Hello world.
```

[Without any new line only Content is displayed].

NOTE:→

System.out.print()

1) System.out.print() will print the given Content & after printing the Content no new line will be given.

2) We Can't Use System.out.print without giving any Content. [It will result in Compile time error.]

System.out.println()

1) Using System.out.println will print the given Content & after printing the Content a new line will be given.

2) We can use System.out.println without giving any Content it will give a new line.

System.out.println() ✓

System.out.print() ✗ Compile time error.

[Variables] Data	[Functions or methods] Operations
Addition of 2 nos. } 2-data.	[Add] 1 operation
Multiple of 3 nos } 3-data.	[mul] 1-operation
SI = $\frac{P t r}{100}$ } 3-data.	2-operation
Login → USN, Pwd } 2-data.	1) validation. 2) Display the new page.

There are 3-stages involved in variable usage.

1) Declaration.
2) Initialization.
3) Utilization.

1) Declaration :→

It means creation of variables.

General Syntax :→

type variablename;

Here type specifies the data type of the variable & the amount of data that can be stored in the variable & the size of the data that can be stored in the variable.

Type can be of 2-types. :→

1) Primitive data type.
2) Java type or derived type.

1) Primitive data type :→

Those data types which are predefined in the language is called as primitive datatype.

There are only 8- primitive data types in java :-

1) byte
2) Short
3) int
4) long.
5) float
6) double.
7) char.
8) boolean.

Non-primitive
String

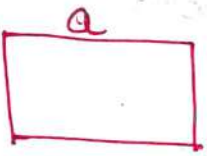
NOTE :->

→ String is not a primitive data type in java. its a builtin class.

→ Any variable which is created or declare by using any 1 of these above 8- primitive data types. is called as primitive variables.

→ Variable name is an identifier. So, for variable name we can give any valid identifier.

Ex :-> int a;



2) Initialization :->

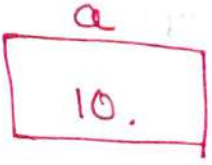
Storing the value for the created variable is called as initialization.

General Syntax

Variable name : value ;

Ex :->

a = 10;



3) Utilization / Usage :→

Using the value stored in the variable for some purpose is called as utilization.

Ex:→

1) Printing the value of variables on the screen.
2) Using the value of a variable / some arithmetical operation etc.

```
System.out.println(a);
```

NOTE:→

→ Using System.out.println() we can print the value stored in the variable.

Ex:→

```
System.out.println(a);
```

↓ We can print value of a.

Ex:→

```
class Simple1
{
public static void main(String[] args)
{
//declaration
    int a;
    double d;
    char ch;
//Initialization
    a=10;
    d=10.5;
    ch='A';
// Usage
System.out.println(a);
System.out.println(d);
System.out.println(ch);
}
}
```

O/P:→

10

10.5

A

or

C:\RCME2>cd variables

C:\RCME2\variables>javac Simple1.java↵

C:\RCME2\variables>java Simple1↵

10

10.5

A

(2)

```
Class Simple2
{
 public static void main(String[] args)
 {
    // declaration
     int empId;
     double empsal;
     char empgrade;

     // Initialization
      empId = 100;
      empsal = 9000.5;
      empgrade = 'A';

      // Utilization
     System.out.println("Your employee Id is" + empId);
     System.out.println("Your ~~emp~~Salary is" + empsal);
     System.out.println("Your grade is" + empgrade);
 }
}
```

O/P:→

Your employee Id is 100.
Your salary is 9000.5
Your grade is A.

③

```
Class Simple3
{
  public static void main (String[] args)
  {
    int a; // declaration.
    a=10; // Initialization.
    System.out.println(a); // Usage.
    a=20; // Re-initialization
    System.out.println(a); // Usage
    a= 10+20; // Re-initialization.
    System.out.println(a); // Usage
  }
}
```

O/P:→

10
20
30.

④ Creation + Initialization.

```
Class Simple4
{
  public static void main(String[] args)
  {
    int stdId=10; // declaration and initialization
    System.out.println("The student Id is" +stdId);
    double percentage = 90.5;
    System.out.println("The percentage is" +percentage);
    boolean result = true;
  } System.out.println("The value of result is" +result);
}
```

O/P:->

The student id is 10.

The percentage is 90.5

The value of result is true.

⑤

```
Class Simple5
{
  public static void main(String[] args)
  {
    int a=10;
    int a=90; // Error.
    System.out.println(a);
  }
}
```

O/P:->

Compile time error because we can't Create 2-variable with the same name inside a same method or block.

But;

```
Class Simple5
{
  Public static void main (String[] args)
  {
    int a=10;
      a=90;
    System.out.println(a);
  }
}
```

O/P:->

90.

⑥

```
Class Simple6
{
  public static void main(String [] args)
  {
    int a=10;
    int A=90;
    System.out.println(a);
  }
}
```

O/P:->

10.

⑦

```
Class Simple7
{
  public static void main(String[] args)
  {
    int a=10;
    System.out.println(a=100);
  }
}
```

Here we can perform operation but can't Create new variable.

O/P:->

100.

⑧

```
Class Simple8
{
  public static void main(String[] args)
  {
    int a=10;
    double a=10.5;
    System.out.println(a);
  }
}
```

O/P:-> Error.

⑨ Class Simple9

```
{
  public static void main(String[] args)
  {
    int a, b=20, c;
    a=10;
    c=30;
   System.out.println(a);
   System.out.println(b);
   System.out.println(c);
  }
}.
```

O/P:→

10
20
30.

⑩ Decision making Statements.

① if else.

```
Class Simple10
{
 public static void main(String[] args)
 {
   int age=2;
    if(age >=13)
    {
     System.out.println("Account Created
                          successfully");
    }
    else
    {
     System.out.println("Sorry, Kid");
    }
 }
}
```

O/P:→

Sorry, Kid.

② Multiple conditions. [Control flow statement]

→If else is the one of the control flow statement.
→Here we can regulate the flow of statements which will be executed.
→Where ever we need to take decision we go for if else.

⑪

```
Class Simple11
{
 public static void main(String[] args)
 {
   int percentage=85;
    if(Percentage >=70)
    {
     System.out.println("Your result is FCD");
    }
   else if(Percentage>=60 && Percentage<70)
    {
     System.out.println("Your result is FC");
    }
   else if(Percentage >=50 && percentage<60)
    {
     System.out.println("Your result is SC");
    }
   else
    {
     System.out.println("Failed, Attend again!!");
    }
 }
}.
```

O/P:→ Your result is FCD.

IQ :=>

```
Class Simple12
{
public static void main(String[] args)
 {
   int a=10;
   int b=10;
    if(a=b) // --> Error !!
   {
     System.out.println("Equal");
   }
   else
   {
     System.out.println("UnEqual");
   }
 }
}.
```

O/P :=>

Compile time Error.

We should always use Comparision Operator '==' Otherwise it will result into Compiletime error.

(13).

```
Class Simple13.
{
  public static void main(String[] args)
  {
    int a=10;
    int b=10;
      if(a==b)
      {
       System.out.println("Equal");
      }
  }}.
```

O/P :=> No o/p.

(14) When single statement then using braces is not manditory.

```
Class Simple14
{
  public static void main(String[] args)
  {
    int a=10;
    int b=10;
     if(a=b)
      System.out.println("Equal");
     else
      System.out.println("Unequal");
  }
}.
```

O/P :=> Equal.

(15) For loop.

for(; ;)

Initialization ↓ Condition ↓ Increament/Decreament.

```
Class Simple15.
{
 public static void main(String[] args)
  {
    for(int i=1; i<=5 ; i++)
    {
     System.out.println("Hi");
    }
  }}.
```

O/P :=> Hi
Hi
Hi
Hi Hi

→ If you want to execute certain statements repeatedly we can go for "for loop".

→ for loop is one of the control flow statement.

⑯ Write a java pgm to print natural nos from 1-30

```
Class Simple16
{
  public static void main(String[] args)
  {
    for (int i=1; i<=30; i++)
    {
      System.out.println(i);
    }
  }
}
```

O/P:→ 1 ... 30.

⑰ Write a java pgm to print even nos. b/w 1-30.

```
Class Simple17
{
  public static void main(String[] args)
  {
    for (int i=1; i<=30; i++)
    {
      if (i%2==0)
      {
        System.out.println(i);
      }
    }
  }
}
```

O/P:→

2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30.

Assignment:→

Write a java pgm to print odd nos from 1-30.

```
Class assign1
{
  public static void main(String[] args)
  {
    for (int i=1; i<=30; i++)
    {
      if(i%2!=0)
      {
        System.out.println(i);
      }
    }
  }
}
```

Write a java pgm to print all the nos which are divisible by 3 b/w 3-30.

```
Class assign2
{
  public static void main(String[] args)
  {
    for (int i=3; i<=30; i++)
    {
      if (i%3==0)
      {
        System.out.println(i);
      }
    }
  }
}
```

Write a java pgm to print fizz if the no. is divisible by 3. and it should print buzz if the no. is divisible by 5 & it should print fizzbuzz if the no. is divisible by both 3&5. for the range of nos 3-30.

```
Class Simple19
{
Public static void main(String args)
{
 for (int i=3 ; i<=30;i++)
 {
  if (int i%3==0 && i%5==0)
  {
  System.out.println ("Fizz Buzz="+i);
  }
  else if (i%3==0)
  {
  System.out.println ("Fizz="+i);
  }
  else if (i%5==0)
  {
  System.out.println("Buzz="+i);
  }
 }
}
}
```

O/P:→

```
Fizz =3
Buzz =5
fizz =6
fizz =9.
Buzz =10.
fizz =12
fizz buzz =15.
fizz =18.
Buzz =20
fizz =21
fizz =24
Buzz =25
fizz =27
fizz buzz = 30
```

To print odd nos blw 1-30.

```
Class Simple20
{
 public static void main(String[] args);
 {
  for (int i=1; i<=30; i=i+2)
  {
   System.out.println(i);
  }
 }
}
```

O/P:→

1, 3, 5, 7 --- , 29.

To print even nos blw 1-30.

```
Class Simple20
{
public static void main(String[] args)
{
 for (int i=2; i<=30; i=i+2)
 {
  System.out.println (i);
 }
}
}
```

O/P:→ 2, 4, 6, 8 --- 30.

(21) To display

```
*
* *
* * *
* * * *
* * * * *
```

```
Class Simple21.
{
 public static void main(String[] args)
 {
    for(int i=1; i<=5; i++)//outer for loop
    {
      for(int j=1; j<=i; j++)//Inner
                             for loop.
      {
       System.out.print("*");
      }
      System.out.println();
    }
 }
}
```

O/P:->

```
*
* *
* * *
* * * *
* * * * *
```

Working

i	j	i<=5	j<=i
1	1	1<=5	1<=1
			2<=1 ✗
2	1,2	2<=5	1<=2
			2<=2
			3<=2 ✗
3	1,2,3	3<=5	1<=3
			2<=3
			3<=3
			4<=3 ✗
4	1,2,3,4	4<=5	1<=4
			2<=4
			3<=4
			4<=4
			5<=4 ✗
5	1,2,3,4,5	5<=5	1<=5
			2<=5
			3<=5
			4<=5
			5<=5
			6<=5 ✗
6		6<=5 ✗	

(22) To display

```
* * * * *
* * * *
* * *
* *
*
```

```
Class Simple22
{
 public static void main(String[] args)
 {
   for(int i=5; i>=1; i--)//outer for loop
   {
    for(int j=1; j<=i;j++) //Inner for loop
    {
     System.out.print("*");
    }
    System.out.println();
   }
 }
}
```

O/P:->

```
* * * * *
* * * *
* * *
* *
*
```

(23) To display

```
*
* *
* * *
* * * *
* * * * *
* * * * *
* * * *
* * *
* *
*
```

```
Class Simple23.
{
public static void main(String[] args)
{
 for(int i=1; i<=5; i++)
  {
   for(int j=1; j<=i; j++)
   {
    System.out.println("*");
   }
   System.out.println();
  }
  for(int i=5; i>=1; i--)
  {
   for(int j=1; j<=i; j++)
    {
     System.out.print("*");
    }
    System.out.println();
  }
}}.
```

O/P:=>

```
*
**
***
****
*****
****
***
**
*
```

(24) Display

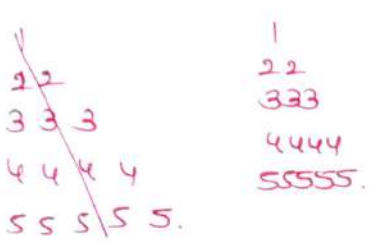
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5.
```

```
Class Simple24
{
 public static void main(String[] args)
 {
  for(i=1; i<=5; i++)
  {
   for(j=1; j<=i; j++)
   {
    System.out.print(j); //use j.
   }
   System.out.println();
  }
 }.
}.
```

O/P:=>

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

(25) Display



```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5.
```

```
1
22
333
4444
55555.
```

```
Class Simple25.
{
 public static void main(String[] args)
 {
  for(i=1;i<=5;i++)
  {
   for(j=1;j<=i;j++)
   {
    System.out.print(i); // Use i.
   }
    System.out.println();
  }
 }
}.
```

O/P:→

```
1
22
333
4444
55555.
```

To print

```
0
1 0
0 1 0
1 0 1 0
0 1 0 1 0.
```

```
Class Sample
{
 psvm(String[] args)
 {
  for(int i=1;i<=5;i++)
  {
   for(int j=1;j<=i;j++)
   {
    S.O.P((i+j)%2);
   }
   SOPln();
  }
 }
}.
```

(1)

(2+1)
3%1

26/10/15.

Methods or Functions.

Methods are nothing but block of statements which will be executed whenever it is invoked / called.

→ Methods are developed for reuseability.

Syntax:→

Method declaration or Method Signature.

accesslevel Modifier returntype Methodname (Arguments)

Method definition or body

{

}

① accesslevel.

1) Public
2) Private
3) Protected
4) default.

② Modifier.

1) static
2) Non-static.

③ Returntype.

int
char
double
:
:
void.

④ Method name

identifier.

⑤ Arguments.

i/Ps.

Return type & return value should match.

① Access level

Any method should have access levels & it can have any of the 4-access levels.

They are:→

1) Public
2) Private
3) Protected.
4) Default.

② Modifiers

These are can be either static or non-static.

③ Return type:→

→ Return type of method specifies what type of o/p that method returns.

→ Return type of method can be any of the primitive data type / java type.

→ Return type can be void.

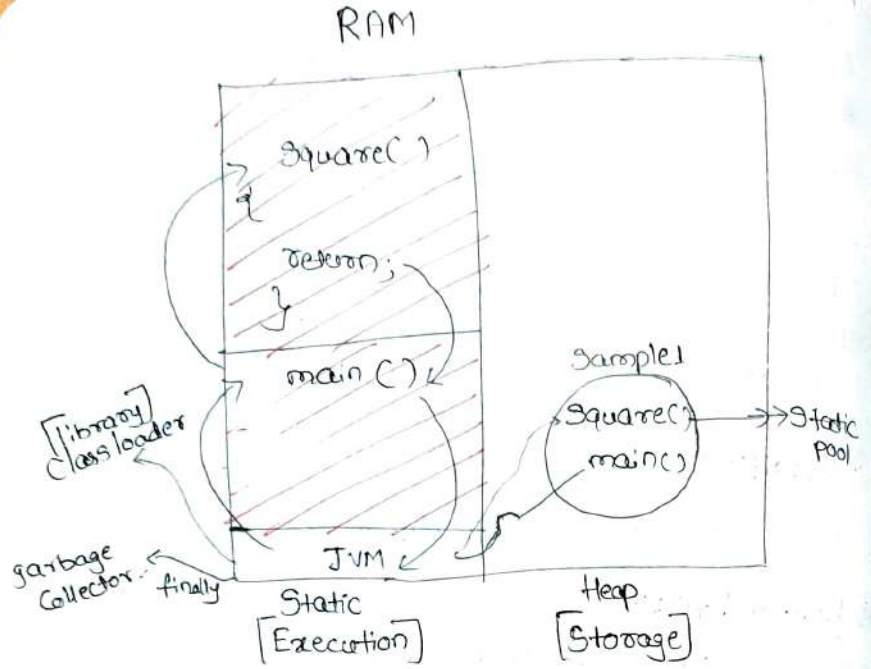
* → Return type and return value should always match.

④ Method name

It is an identifier & it can have any valid identifier.

⑤ Arguments.

Arguments are nothing but i/p for a method.



1) The entire pgm execution happens in RAM memory.

→ RAM memory will be divided into 2-memories.

1) Stack.
2) Heap.

1) Stack is for execution purpose.
2) Heap is for storage purpose.

→ JVM will be loaded on to the Stack.

→ JVM will call a built-in library "Classloader".

→ Classloader will load all the static members of the class into static pool in the heap memory

NOTE:→

Anything which is declared with the keyword static are static members.

→ JVM will check the availability of main method in the static pool.

→ If the main method is available in static pool then it will fetch it & load it on to the stack for execution

→ Control will be given to main method. so that, Main method can execute its statements.

→ From main method we have to call square method

→ As square method is in the static pool we should use static pool name [class name] to call it

Ex :→ sample1.square(3)

→ When we call square method as shown above, Square method will be loaded on to the stack & Control will be given to square method so that it can execute its statements.

→ Once square method execute its statements Completly then square method will return the Control back to main method.

→ The square method will be erased from the stack.

→ Now main method resume its [continue its] execution & finally it will return Control back to JVM.

→ The main method will be erased from the stack.

→ JVM will call garbage Collector which is a daemon thread. [lowest priority / background thread] which will clear Content from heap memory.

Finally JVM will exit from the stack.

```
Class Sample1
{
  Static void Square(int num)
   {
     int Sq = num*num;
    System.out.println("The result is"+Sq);
     return;
   }

   public static void main(String[] args)
   {
    System.out.println("The main starts...");
    Sample1.square(2);
    System.out.println("The main ends...");
   }
}
```

O/P:→

C:\RCME2\methods : javac Sample1.java

java Sample1

The main starts

The result is 4

The main ends.

```
Class Sample2
{
   Static void Cube(int num)
   {
     int C = num*num*num;
    System.out.println("The result is"+C);
     return;
   }

   public static void main(String[] args)
  {
    System.out.println("The main starts...");
   //Using main me the Cube findn
   Sample2.Cube(2)
   // Using main the Cube method to find Cube of 2
   Sample2.Cube(2); // Method invocation
   System.out.println("----------");
   Sample2.Cube(3);
  System.out.println("The main ends...");
  }
}
```

O/P:→

The main starts...

The result is 8

The result is 27

The main ends...

```
class Sample3
{
 static void add(int num1, int num2)
 {
   int sum=num1+num2;
  System.out.println("The result is"+sum);
  return;
 }

 public static void main(String[] args)
 {
 System.out.println("The main starts...");
 Sample3.add(2,3);
 System.out.println("=========");
 Sample3.add(10,20);
 System.out.println("The main ends...");
 }
}
```

O/P:→

```
The main starts....
The result is 5.
==========
The result is 30.
The main ends...
```

```
class Sample4
{
 static void cube(int num)
 {
   int c=num*num*num;
   System.out.println("The result is "+c);
 }

 public static void main(String[] args)
{
System.out.println("The main starts...");
 Sample4.cube(3);
System.out.println("==========");
 Sample4.cube(2);
System.out.println("The main ends...");
 }
}
```

O/P:→

```
The main starts.
The result is 9.
==========
The result is 4
The main ends.
```

→ If the methods return type is void then developing return statement is not mandatory.

→ If the programmer has not developed return statement or void methods then Compiler will adding automatically at Compile time.

→ Void means return nothing. So, if a methods return type is void we can't return any value.

27/01/15.

int as return type.

```
Class samples.
{
  static int square(int num)
   {
     int sq=num*num;
     return sq;
   }
  public static void main(String[] args)
  {
    System.out.println("The main starts...");
     int res = samples.square(9);
    System.out.println("The result is"+res);
    System.out.println("The main ends ...");
   }
}
```

O/P:→

```
The main starts..
The result is 81.
The main ends.
```

```
Class samples
{
 static int Cube(int num)
 {
   int C=num*num*num;
   return C;
 }
 public static void main(String[] args)
 {
   System.out.println("The main starts...");
   int res= ~~Cube~~ samples.Cube(9);
   System.out.println("The result is"+res);
   System.out.println("The main ends...");
  }
}
```

O/P:→

```
The main starts...
The result is 6859.
The main ends....
```

```
class Sample7
{
  static int test()
  {
    return 10;
  }
  public static void main(String[] args)
  {
    System.out.println("The main starts...");
    int res = Sample7.test()
    System.out.println("The result is"+res);
    System.out.println("The main ends");
  }
}
```

O/P:→

```
The main starts.
The result is 10.
The main ends.
```

→Using System.out.println we can print the return value of the method directly.

```
class Sample8
{
  static double test()
  {
    return 10.5;
  }
  public static void main(String[] args)
  {
    System.out.println("The main starts...");
    System.out.println(Sample8.test());
    System.out.println("The main ends...");
  }
}
```

O/P:→

```
The main starts.
10.5
The main ends.
```

```
class Sample9
{
  static boolean test()
  {
    return true;
  }
  public static void main(String[] args)
  {
    System.out.println("The main starts");
    System.out.println("The result is"+Sample9.
                                    test());
    System.out.println("The main ends");
  }
}
```

O/P:→

```
The main starts
The result is true
The main ends.
```

Void :→

If you not expecting any value from method after processing then those methods are void methods.

Non-void :→

If you are expecting some value from a method & based on that value if you want to do further processing then we should go for non-void methods

```
Class Sample10.
{
 Static int test()
 {
  return ; // int should return value.
 }

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

O/P :→

Error.

```
Class Sample10
{
 Static int test()
 {
        //error because it should
          have return statement.
 }
}
```

Void method we Can't Call in S.O.P.

```
Class Sample11
{
 Static void test()
 {
  return;
 }

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

 Sample11.test();

 System.out.println("The main ends");
}
}.
```

O/P :→

The main starts
The main ends.

Given two int values return true if sum of them is 30 or one of them is 30.

```
Class Sample12
{
Static boolean test (int a, int b)
 {
   if (a==30 || b==30 || a+b== 30)
   {
     return true;
   }
   else
   {
     return false;
   }
 }

public static void main (String args)
 {
  System.out.println("The main starts...");
  System.out.println("The result is" + Sample12.test(
  System.out.println("The result is" + Sample12.test(
  System.out.println("The result is" + Sample 12.test
                                        (80,8
  System.out.println("The main ends...");
 }
}
```

O/P:→

```
The main starts.
The result is true.
The result is true
The result is false.
The main ends.
```

TestCase

test (10,20) = true.
test (30,50) = true.
test (80,80) = false.

② There are 2-monkeys A & B. If both the monkies are Smiling then we are in trouble. and if both the monkies are not Smiling then also we are in trouble return true if we are in trouble.

TestCase.

test (true, true) = true.
test (false, false) = true.
test (true, false) = false.
test (false, true) = false.

```
Class Sample13
{ Static
  Static boolean test (boolean a, boolean b)
 {
   if (a==true && b==true)
   {
     return true;
   }
  else if (a==false && b==false)
   {
     return true;
   }
```

By doing optimization
// return (a==b);

```
        else
        {
            return false;
        }
    }

public static void main (String[] args)
{
System.out.println ("The main starts...");
System.out.println ("The result ::" + Sample13.test
                                        (true,true));
System.out.println ("The result ::" + Sample13.test
                                        (false,false));
System.out.println ("The result ::" + Sample13.test
                                        (true,false));
System.out.println ("The result::" + Sample13.test
                                        (false,true));
System.out.println ("The main ends");
}
}
```

O/P :→

```
The main starts...
The result is true.
The result is true.
The result is false.
The result is false.
The main ends.
```

3) Given 2 ints return twice their sum. If both are same. otherwise just return their sum.

Test Case.

test (10,10) = 40.

test (10,20) = 30.

```
Class Sample14
{
Static int test (int a, int b)
{
  if if (a==b)
  {
    return 2*(a+b);
  }
  else
  {
    return a+b;
  }
}

Public Static void main (String[] args)
{
System.out.println ("The main starts...");
System.out.println ("The result ::" + Sample14.
                                        test(10,10));
System.out.println ("The result ::" + Sample14.
                                        test(10,20));
System.out.println ("The main ends...");
}
}
```

O/P:->

Optimization of Sample 14 by using ternary operator:->

```
Condition ? exp1 : exp2

Class Sample14
{
Static int test (int a, int b)
{
return (a==b) ? 2*(a+b) : (a+b);
}.
```

Assignment:->

There is a loud talking parrot. If the parrot is talking b/w 20hrs (8PM) to 8hrs (8AM).. then we are in trouble.

Test Case

test (true, 21) = true.
test (false, 21) = false
test (true, 10) = false.

It should accept 1-24 only

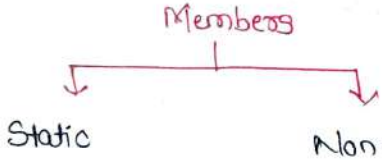
```
Class Sample14
{
Static boolean test (boolean a, int b)
{
if (a==true) && (b<=8 || b>=20))
{
return true;
}
else if (a==false &&
{
return false;
}
}

public Static void main(String args[])
{
System.out.println("The main starts..");
System.out.println("The result is"+ Sample14.
                                test(true,21));
System.out.println("The result is" + Sample14.
                                test(false,21));
System.out.println("The result is" + Sample14.
                                test(true,10));
System.out.println("The main ends");
}
}.
```

return a==true && (b<=8 || b>=20));

O/P:-> The main starts.
The result is true
The result is false.
The result is false.
The main ends

28/10/15.

```
Class A
{
    fields ⇒ Attributes or variables or data members.
      or
    behaviors ⇒ methods
                  or
                functions.
                  or
                member functions.
}
```



Class:→

It is the Collection of fields & behaviors.

fields:→

It means variables.

Behaviors:→

It means methods.

Variables are also called as data members or attributes.

Methods are also called as functions or member functions.

→ Whatever the class Comprises or Consists of are Considered as members of a class.
or
Whatever is ~~seters~~ written inside the class defination block are Considered as members of a class.

There are 2-Categories of members.

1) Static members.
2) Non-static members.

1) Static members:→

→ Any member which is declared by using the keyword static are called as static members.

→ Static members includes static variables, static methods & static blocks. or SIB [static initialization block].

→ Static members can be accessed by using ~~classname~~ Classname.member name. [because static members will be loaded on to the static pool. & static pool name will be Classname].

Non-Static

Any member which is not declared with the keyword static are called as non-static members

→ Non-static members includes non-static variables, non-static methods, Constructors & non-static blocks or IIB [Instance Initialization Block]

→ To access non-static members of the class we should Create the object of a class. Or we should create the instance of a class. Or we should instantiate the class.

Non-Static pgms.

```
Class sample1
{
  void test1()
  {
    System.out.println("Running test method");
  }

  Public static void main(String[] args)
  {
    System.out.println("The main starts");
    //Sample1.test1();

    Sample1 sv=new Sample1();
      sv.test1();
    System.out.println("The main ends");
  }
}
```

O/P:→ The main starts
Running test method
The main ends.

If you want to access non-static members of the class we should use the following syntax.

General Syntax

Classname referance variable = new classname();

→ new is an operator which will Create a new address space randomly in the heap memory.

→ classname() :→ It will load all the ~~static~~ non-static members of that class into the address space.

→ referance variable :→ It is an identifier which is used to hold the address of the object.

→ class name :→ It indicates the referance variables type.

```
Class demo1
{
  Void test1()
  {
    System.out.println("Running test method");
  }

  public static void main(String[] args)
  {
    System.out.println("The main starts");
    ~~Syste~~ demo1 sv=new demo1(); // Creating the object of demo1 class.
    sv.test1();
    System.out.println("----------");
    sv.test1();
  }
}
System.out.println("The main ends");
```

O/P:→

The main starts.

Running test1 method.

Running test1 method.

The main ends.

2-non-static members.:→

```
Class sample2
{
 Void test1()
 {
  System.out.println("running test1 method");
 }
 Void test2()
 {
  System.out.println("running test2 method");
 }
 public static void main(String[] args)
{
System.out.println("The main starts");
 Sample2.ref = new Sample2();
  ref.test1();
 System.out.println("-------");
  ref.test2();
 System.out.println("The main ends");
}
}
```

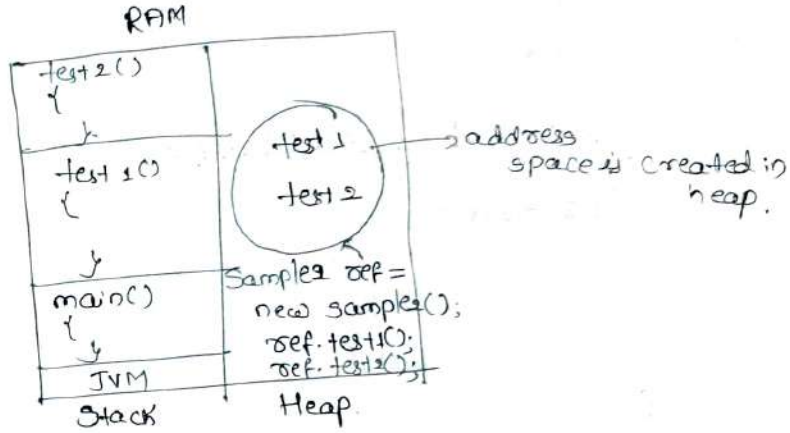
O/P:→

The main starts.

running test1 method

running test2 method.

The main ends.



1- non static member, 1- non static variable.

```
Class sample3
{
 int a=90; // non-static variable.
 void test1() // non static method.
 {
  System.out.println("Running test1 method");
 }
 public static void main(String[] args)
 {
  System.out.println("The main starts");
  Sample3 s1 = new sample3();
  System.out.println("The value of a is"+s1.a);
  s1.test1();
```

```
        System.out.println("The main ends");
    }
}
```

O/P:→

The main starts
The value of a is 90.
Running test1 method.
The main ends

Static method & variables:→

```
Class Sample4
{
  Static int a=10;
  Static void test1()
  {
   System.out.println("Running test1 method");
  }

  public static void main(String[] args)
  {
   System.out.println("The main starts");
   System.out.println("The value of a is" +
                      Sample4.a);
     Sample4.test1();
   System.out.println("The main ends");
  }
}
```

O/P:→

The main starts.
The value of a is 10
Running test1 method.
The main ends.

Assignment:

```
Class Sample5
{
  Static int a=10;
   int b=90;
  Static void test1();
   {
    System.out.println("Running test1 method");
   }
    void test2()
    {
     System.out.println("Running test2 method");
    }
   public static void main(String[] args)
  {
   System.out.println("The main starts");
   System.out.println("The value of a is"+Sample5.a);
     Sample5 ref=new Sample5();
   System.out.println("The value of b is"+ref.b);
       ref.test1(); Sample5.test1(); // Static test1
   System.out.println("
                      ref.test2(); // non static test2
  } System.out.println("The main ends");
```

O/P:→

The main starts.
The value of a is 10.
The value of b is 90.
Running test1 method.
Running test2 method.
The main ends.

29/04/15

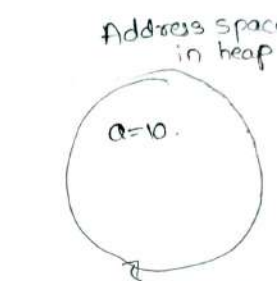
Advantages of Non-Static members:→

1) We can create multiple objects for a class.
2) As we can create multiple objects for a class, we can have multiple copies of non-static members

```
Class Sample6.
{
  int a=10;
  public static void main(String args[])
  {
    System.out.println("The main starts...");
    Sample6 ov1 = new Sample6(); // Creating an object
    Sample6 ov2 = new Sample6(); // Creating another object
    System.out.println("The value of ov1.a=" + ov1.a);
    System.out.println("The value of ov2.a=" + ov2.a);
    System.out.println("The main ends");
  }
}
```

O/P:→

The main starts
The value of ov1.a = 10;
The value of ov2.a = 10
The main ends.



Sample6 ov1 = new Sample6()

Sample6 ov2 = new Sample6

all objects are independent.

```
Class Sample7
{
  int a=10;
  public static void main(String[] args)
  {
    System.out.println("The main starts");
    Sample7 ov1 = new Sample7();
    Sample7 ov2 = new Sample7();
    ov1.a=20;
    System.out.println("The value of ov1.a=" + ov1.a);
    System.out.println("The value of ov2.a=" + ov2.a);
    System.out.println("The main ends");
  }
}
```

O/P:→

The main starts

The value of $ov_1.a = 20$

The value of $ov_2.a = 10$.

The main ends.

NOTE:→

We can have multiple copies of non-static variables or non-static methods & values of each copy might differ.

```
Class Sample8.
{
  Static int b=10;
  Public static void main(String args[])
  {
    System.out.println("The main starts..");
    System.out.println(" The value of b is"+
                                  Sample8.b);
      Sample8.b=20;
    System.out.println("The value of b is"+
                                  Sample8.b);
    System.out.println("The main ends...");
  }
}
```

The main starts.
The value of b is 10
The value of b is 20.
The main ends.

NOTE:→

→We can have only single copy of static members

→In one execution cycle only one static pool will be created for a class by class loader so, we can have only one copy of static members.

Static

1) When the requirements are fixed.
2) When we want single copy of the data.

Ex:→ Company name for employees of same organization.

Non-Static.

1) When the req. are not fixed
2) When we wants multiple copies of data.

Ex:→ Student id, employee id.

```
Class employee
{
  int empid=10;
  public static void main(String[] args)
  {
    System.out.println(" The main starts");
    Syst.
    employee ov1 = new employee();
    employee ov2 = new employee();
     ov1.empid=10;
     ov2.empid=20;
    System.out.println("The 1st employee id::"+
                                  ov1.empid);
```

```
System.out.println("The 2nd employee id::"+ mv2.empid);
System.out.println("The main ends..");
    }
}
```

O/P:->

The main starts.
The 1st employee id::10
The 2nd employee id::20.
The main ends.

Variables

Any variables [Primitive, referance] can be Categorized into local & global variables

Local variables

1) These variables which are declared inside a method or a block.
2) Local variables cann't be accessed outside the method or block, in which it is declared or created. [i.e. it will have method Scope]
3) These variables cann't be categorized into static or non-static.

Global variables

1) These variables which are declared outside any method but with the class defination block
2) These variables can be accessed from all parts of the pgm.
3) These variables can be categorized into Static & non-Static variables.
4) Static global variables are also called as "class variables".
5) Global non-static variables are also called as "instance variables".

Local variables :->

```
Class Simple1
{
  static void test1()
  {
   int a=10; // local variable "a" of test1()
  System.out.println("The value of a is"+a);
  }
  public static void main(String[] args)
  {
  System.out.println("The main starts...");
   Simple1.test1();
  // System.out.println("The value of a is"+a);
                                    //error.
   System.out.println("The main ends");
  }
}
```

O/P:->

The main starts.
The value of a is 10.
The main ends.

Global variables:→

Class

We can have local variables of same name in multiple methods. :→

```
Class Simple2
{
  Static void test1()
  {
    int a=10; // local variable 'a' for test1 method
    System.out.println("The value of a is"+a);
  }
  public static void main(String[] args)
  {
    System.out.println("The main starts..");
    Simple2.test1();
      int a=20; // local variable "a" for main method
    System.out.println("The value of a is"+a);
    System.out.println("The main ends..");
  }
}.
```

O/P:→

```
The main starts.
The value of a is 10.
The value of a is 20
The main ends.
```

NOTE:→

local variables will be Created when the method enters the stack for execution. & it will be destroyed when the method leaves the Class after execution.

→ local variables are stack members.

→ We can have local variables with the same name across different methods/blocks.

The arguments of method are local.

Imp

```
Class Simple3
{
  static void test1(int a) // Method arguments are local variables.
  {
    System.out.println("The value of a is"+a);
  }
  public static void main(String[] args)
  {
    System.out.println("The main starts");
    Simple3.test1(10);
    System.out.println("The value of a is"+a); // Error
    System.out.println("The main ends");
  }
}.
```

O/P:→

```
The main starts.
The value of a is 10.
The main ends.
```

Supply value through variable not directly

```
Class Simply4
{
 static void test1(int a)
 {
  S.O.P("The value of a is"+a);
 }
 public static void main(String[] args)
 {
 S.O.P("The main starts...");
 int a=10; // Variable.
 Simply4.test1(a);
} } S.O.P("The main ends");
```

O/P:→

```
The main starts.
The value of a is 10.
The main ends.
```

NOTE:→

We can supply value to a method through Variable.

I:Q

***

Find sum of the digits.

```
Class Simple5.
{
 static void SumofDigits(int num)
 {
   int res=0;
  while (num>0)
  {
    int temp=num%10;
   res = res+temp;
   num= num/10;
  }
 System.out.println("The SumofDigits is "+res);
 }
 public static void main(String[] args)
{
System.out.println("The main starts");
    int ~~Sum~~ num=123;
 Simple5.SumofDigits(num);
  num=546;
  Simple5.SumofDigits(num);
 System.out.println("The main ends");
}
```

O/P:→

```
The main starts.
The sum of digits is 6
The " " " 15
The main ends
```

NOTE:→

→ If you are finding modulus of any no. with 10. then we will get the last digit of that no.

Ex:→

123 % 10 = will result in 3.

```
10)123(12
   10
   ---
   23
   20
   ---
    3
```

→ If you are dividing any no. by 10. then we will get the resultant no. excluding last digit of that no.

Ex:→

$\frac{123}{10} = 12$.

$\frac{110}{10} = 11$

→ While loop

If you wants to execute certain statements repeatedly then we can go for while loop.

General syntax:→

```
while (Condition)
{
    Statements...
}
```

→ Condition should be a boolean expression

→ If any expressions is result in either true/false is called as boolean expression.

Assignment:→

Write a java pgm to check whether a given no is armstrong or not?

30/10/15 armstrong no:→

1) It is 3-digit no.
2) Sum of Cube of each no. of digit is no. itself.

Ex:→ 371.

$3^3 + 7^3 + 1^3$

$27 + 343 + 1 = 371$.

```
Class simples
{
    Static void armstrong(int num)
    {
        int res=0;
        while
```

```
Class SimpleG
{
Static int SumOfDigits(int num)
 {
   int res=0;
   while(num>0)
   {
     int temp=num%10;
     res=res+(temp*temp*temp);
     num=num/10;
   }
   return res;
 }

Static void CheckArmstrong(int num, int res)
 {
   if(num==res)
   {
   System.out.println("The number "+num+
         " is Armstrong");
   else
   {
   System.out.println("The number"+num+
         " is not armstrong");
   }
 }

public static void main(String[] args)
 {
   System.out.println("The main starts...");
    int num=467;
    int res=SimpleG.SumOfDigits(num);
    SimpleG.CheckArmstrong(num,res);
   System.out.println("The main ends");
  }
}
```

O/P:→

① The main starts.
The number 467 is not armstrong.
The main ends.

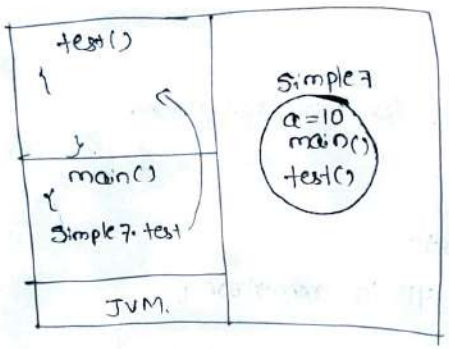
② The main starts.
The number 371 is armstrong.
The main ends.

Global Variable :→

```
Class Simple7
{
Static int a=10;
Static void test1()
{
 System.out.println("from test1:"+Simple7.a);
}
public static void main(String[] args)
{
 S.O.P("The main starts...");
 S.O.P("from main:"+Simple7.a);
   Simple7.test1();
 S.O.P("The main ends");
}
}
```

o/p:→

The main starts.
from main :: 10.
from test :: 10.
The main ends.



If programmer not initialized the [Global variables] variables value Then it will return defaut values.

```
Class Simple8
{
    Static int a;
    Static double d;
    Static char ch;
    Static String s;
    Static boolean b;
public static void main(String[] args)
{
    S.O.P ("The main starts");
    S.O.P ("The value of int var is" + Simple8.a);
    S.O.P ("The value of double var is" + Simple8.d);
    S.O.P ("The value of char var is" + Simple8.ch);
    S.O.P ("The value of String var is" + Simple8.s);
    S.O.P ("The value of boolean var is" + Simple8.b);
    S.O.P ("The main ends");
}
}.
```

o/p:→

The main starts.
The value of int var is 0
" " " double " " 0.0
" " " char " " — space.
" " " String " " null.
" " " boolean " " false.

For local variables no default values.

```
Class Simple9
{
    public static void main(String[] args)
    {
        int a;
        double b;
        char ch;
        boolean b;
        String s;
        System.out.println("The value of a is" +a);
        S.O.P ("The value of b is" +b);
        S.O.P ("The value of ch is" +ch);
```

```
        S.O.P(" The value of b is" +b);
        S.O.P(" The value of s is" +s);
    }
}
```

O/P:→ Compile time Error.

NOTE:→

→ For Global variables default values will be assigned based on the data-type of the variables.

→ For local variables default values will not be assigned.

NOTE:→

→ Before using any variable for some purpose that variable should have some value.

→ As global variables will have default values we can use it straight away without explicitly initializing it.

→ Local variables should be initialized explicitly before usage.

Ex:→

```
Class Simple10.
{
    public static void main(String[] args)
    {
        int a;
        S.O.P(a);
    }
}
```

O/P:→ Error.

1.9

①
```
Class Simple11
{
    public static void main(String[] args)
    {
        int a;
        a++;
        S.O.P(a);
    }
}.
```

O/P:→ Error.

②
```
Class Simple12
{
    Public static void main(String[] args)
    {
        int a;
        S.O.P("Done");
    }
}
```

O/P:→ Done.

③
```
Class Simple13
{
    Public static void main(String[] args)
    {
        int a;
        a++; // Error.
        S.O.P("Done!!");
    }
```

O/P:→ Error.

④ Class Simple14

```
{
  public static void main(String[] args)
  {
    int a;
    S.O.P("a++");
  }          ↳ String.
}
```

O/P:→ a++.

*. Final Keyword.

```
Class Simple14
{
  final static double PI=3.14;
  public static void main(String[] args)
  {
    S.O.P("The value of PI is" +
                        Simple14.PI);

    S.O.P("The value of PI is" +
                        Simple14.PI);
    // Simple14.PI=9.0 // Error.
  }
```

O/P:→

The value of PI is 3.14.
" " " " 3.14.

→ For final global variable we should initialized at the time of declaration only.

→ final local variable we can initialize after declaration also.

```
Class Simple15
{
  final static int a=10; // Global variable
  public static void main(String[] args)
  {
    S.O.P("The value of a is" + Simple15.a);
    final int b;
      b=20; // local variable.
    S.O.P("The value of b is" +b);
  }
}
```

O/P:→

The value of a is 10.
" " " b " 20.

Final :→

→ Final is a keyword.

→ If you want to declare constant variables then we should declare that variable with the keyword final.

Ex:→

```
final static int a=10;
```

→ Final variable's value can't be changed or modified or overridden.

→ Both local & Global variables can be final.

Rule:→

→ Global final variables should be initialized at the time of declaration itself. [other wise CTE.]

→ local final variables ~~should~~ can be declared once & initialize later [But only once].

I·Q

Final is a.

(1) identifier (2) Builtin Class.

(3) literal (4) Keyword.

Industry Conventions.

→ Industry Conventions are nothing but best practices followed in industry while developing java programs.

→ If we are following proper industry convention then, our programs will be readable & easily understandable by another developer.

NOTE:→

→ If we are not following industry convention then Code will not result in CTE.

(1) We should follow proper indentation while writing the code.

(2) Industry Convention to develop Classname.

→ All classnames should be developed in Capitalized format.

Ex:→ (1) Sample.

(2) SampleProgram.

(3) SampleProgramAgain

(3) Industry Conventions to develop variables & methods.

→ If a variable is having 1 word then it should be developed Completly in lowercase.

Ex:→ int num.

→ If a variable is having multiple words then 1st one should be completly written in lowercase. & 2nd word onwards 1st letter should be in uppercase.

Ex:→ (1) int employeeId.

(2) sumOfDigits()

(4) Industry Conventions to develop Constant variables.

→ Constant variables should be Completly written in uppercase.

Ex:→ (1) PI

(2) PI_VALUE. [2-words]

I:Q :→

Write a java pgm to reverse the given no.

```
Class Simple16
{
  Static void reverseOfNum (int num
  public static void main (String[] args)
{
   int res=0;
      while (num>0)
       {
        int temp=num%10;
        res =temp+ (res*10);
        num = num/10;
       }
      S.O.P ("The reverse is"+res);
}

public static void main (String[] args)
 {
    S.O.P ("The main starts");
    Simple16. reverseOfNum (123);
    Simple16. reverseOfNum (576);
    SO.P ("The main ends");
 }
}
```

O/P :→

```
The main starts
The reverse is 321
The reverse is 675
The main ends.
```

Write a java pgm to ~~rev~~ palindrome

```
Class Simple17
{
 Static ~~void~~ int reverseOfNum (int num)
 {
   int res=0;
   while (num>0)
   {
    int temp=num%10;
    res = ~~res+ (temp~~ temp + (res*10);
    num= num/10;
   }
   return res;
}

Static void palindrome (int num, int res)
{
   if(num==res)
  {
   S.O.P (" The number" + num + " is palidrome")
   }
   else
  {
   S.O.P (" The number " + num + " is not palind-
                                  -rome");
  }
}
```

```
public static void main(String[] args)
{
  S.O.P ("The main starts");
    int num=121;
  S.O.P ("The
    int res = Simple17.reverseOfNum (num);
    Simple17.palindrome(num,res);
  S.O.P ("The main ends");
  }
}.
```

O/P:→

The main starts.
The number 121 is polindrome.
The main ends.

1) What is java?
2) What is JDK, JRE?
3) What are keyword, Identifier, literals?
4) What are the diff blw System.out.print & " " " (n)?
5) What is the variable? & explain different types of variables?
6) What is primitive data type & mention the primitive data types of java.
7) What is method? & why method
*8) Can we develop a method without return type?
Return type is mandatory. It should have return type

*9) What are the differances blw static & non-static members?
10) What are the differances blw local & global variables?
**11) Can we Compile a java pgm without main method? Yes. but only Compilation but no execution.

Ex:→

```
Class simple18
  {
    int a=10;
  }.
```

Yes, We can Compile java pgm without main method. but to run that pgm main method is manditory.

2/11/15.

Static members:→

→ Static members of the class can be accessed directly.

```
Class A
{
  Static void test()
  {
    S.O.P("Running test() method");
  }

  Public Static void main(String[] args)
  {
    S.O.P ("The main starts...");
    test(); // Directly.
    S.O.P("The main ends...");
  }
}.
```

O/P:→

The main starts.
Running test() method
The main ends.

② Class B
{
Static int a=10;
Public static void main (String args)
{
S.O.P("The main starts");
S.O.P ("The value of a is "+a); //directly
S.O.P ("The main ends...");
}
}.

O/P:→

The main starts.
The value of a is 10.
The main ends.

③ Class C
{
Static int a=10;
Static void test()
{
S.O.P ("from test::"+a);
}
Public static void main (String[] args)
{
S.O.P ("The main starts...");
test();
S.O.P ("from main::"+a);
S.O.P ("The main ends...");
}
}.

O/P:→

The main starts.
from test::10.
from main::10.
The main ends.

Non static members:→

Class D
{
Static int a=10;
void test1()
{
S.O.P ("from test::" +a);
}
Public static void main (String[] args)
{
S.O.P ("The main starts");
D rv=new D();
rv.test1();
S.O.P ("from main::"+a);
S.O.P ("The main ends");
}
}.

O/P:→

The main starts.
from test1::10.
from main::10.
The main ends.

```
Class E
{
  int empId=10;
   void test()
{
S.O.P("Employee ID is" +empId);
}
Public static void main (String[] args)
{
  S.O.P("The main starts");
  E rv=new E();
     rv.test();
  S.O.P("The main ends");
}
}
```

O/P:→

The main starts.
Employee ID is 10.
The main ends.

NOTE:→

From one non-static method or Context we can access other non-static members directly.

```
Class F
{
 int empId=10;
 public static void main(String[] args)
 {
  S.O.P ("The value of empId is" +empId);
 }
}
```

→ Error.

O/P:→ CTE.

{ Non static variable empId Can't be referenced
Static Context directly.

NOTE:→

1) Static members can be accessed directly from Static Context.
2) Static members can be accessed directly from non-static context also.
3) Non-Static members can be accessed directly form another non-static Context.
4) Non-Static members Can't be accessed directly from static Context, object should be Created.

Bank

```
Class Bank
{
 int balance;
 Void balance( )
 {
   S.O.P(" Your Current balance is "+balance
 }
 Void withdraw(int wamount)
 {
   if(wamount<=balance)
   {
     balance = balance - wamount;
     S.O.P(" You have withdrawn "+wamount
                                  " Rs");
     balance();
   }
   else
   {
     S.O.P(" Insufficient funds");
   }
 }
 Void deposit(int damount)
 {
   balance = balance + damount;
   S.O.P("You have deposited "+damount+"Rs
   balance();
 }
Public static void main(String[] args)
{
 S.O.P(" welcome to Swiss Bank ");
 Bank bv1 = new Bank();
   bv1.balance();
 S.O.P(" Transaction details");
   bv1.deposit(1000);
 S.O.P("Transaction details");
   bv1.withdraw(400);
 S.O.P("Transaction details");
   bv1.withdraw(40000);
 S.O.P("** Thank You, Take Care **");
```

O/P:→

Welcome to Swiss bank.
Your current balance is 0.
Transaction details.
You have deposited 1000 Rs.
Transaction details.
You have withdrawn 400 Rs.
Transaction details.

Thank You, Take Care.

3/11/15.

Method Overloading.

```
Class Sample1
{
 Static void test().
  {
    S.O.P("Running test()");
  }

 Static void test(int a)
 {
   S.O.P ("Running test(int a)");
  }

 Static void test(double d)
 {
   S.O.P ("Running test(double d)");
  }

 Static void test(int a, double d)
 {
  S.OP ("Running test(int a, double d)");
 }

Static void test(double d, int a)
{
 S.O.P ("Running test(double d, int a)");
}

Public Static void main(String[] args)
{
   S.O.P("The main starts");
    test();
    test(90);
    test(90.5);
    test(90, 90.5);
    test(90.5, 90);
    S.O.P ("The main ends");
  }
```

O/P:→

```
The main starts
Running test()
Running test(int a)
Running test (double d)
Running test (int a, double d)
Running test (double d, int a)
The main ends.
```

```
Class Sample2
{
Static void add (int a, int b)
{
  int sum=a+b;
 S.O.P ("The addition result is" +sum);
}

Static void add (int a, int b, int c)
 {
    int sum=a+b+c;
 S.O.P ("The addition result is"+sum);
}

Public Static void main (String[] args)
{
```

```
        S.O.P("The main starts");
        add(10,20);
        add(30,40,50);
        S.O.P("The main ends");
    }
}
```

O/P:→

```
The main starts.
The addition result is 30
The addition result is 150
The main ends.
```

* Method overloading on non-static methods

```
Class Sample3
{
    void test1(int a)
    {
        S.O.P("from test1(int a)");
    }
    void test1(double d)
    {
        S.O.P("from test1(double d)");
    }
    public static void main(String[] args)
    {
        S.O.P("The main starts...");
        Sample3 sv = new Sample3();
            sv.test1(90);
            sv.test1(90.5);
        S.O.P("The main ends");
    }
}
```

O/P:→

```
The main starts.
from test1(int a)
from test1(double d)
The main ends.
```

To find area of square & rectangle.

```
Class Sample4
{
    void area(int side)
    {
        S.O.P("The area of Square is"+(side*side));
    }
    int area(int length, int breadth)
    {
        return (length*breadth);
    }
    public static void main(String[] args)
    {
        S.O.P("The main starts");
        Sample4 sv = new Sample4();
            sv.area(5);
        S.O.P("The area of rectangle is"+sv.area(2,3));
        S.O.P("The main ends");
    }
}
```

O/P:→

```
The main starts.
The area of square is 25.
The area of rectangle is 6.
The main ends.
```

One is static, another is non-static :→

```
Class Sample6.
{
    Static void test()
    {
      S.O.P("-from test()");
    }
    void test(int a)
    {
      S.O.P("-from test(int a)");
    }
    Public Static void main(String[] args)
    {
      S.O.P("The main starts");
      test();
      Sample6 sv = new Sample6();
      sv.test(90);
      S.O.P("The main ends");
    }
}
```

O/P:→

```
The main starts.
from test()
from test(int a)
The main ends.
```

I.Q :→ Ambiguity.

```
Class Sample7
{
  Static void test(int a)
  {
    S.O.P("-from test(int a)");
  }
  Static void test(int a)
  {
    S.O.P("from test(int a)");
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    test(90);
    S.O.P("The main ends");
  }
}
```

O/P:→ CTE.

If only variable names changes & ~~return type method~~ name remains same then CTE.

```
Class Sample8
{
  Static void test(int x, int y)
  {
    S.O.P("from test(int x, int y)");
  }
```

```
    static void test(int y, int x)
    {
     S.O.P ("from test(int y, int x)");
    }
    public static void main (String[] args)
    {
    S.O.P (" The main starts");
     test (10, 10);
    S.O.P("The main ends");
    }
}.
```

O/P:→ CTE

I.Q :→

We can overload main method also ?? :→

```
Class Sample8.
{
  public static void main(int a)
  {
   S.O.P (" main (int a)");
  }
  public static void main(double d)
  {
   S.O.P ("main (double d)");
  }
public static void main(String[] args)
{
   S.O.P (" The original main starts");
   main(10);
   main (10.5);
   S. O. P (" The original main ends");
  }
}
```

O/P:→

```
The original main starts.
main(int a)
main(double d)
The original main ends.
```

What is method overloading?

→ Developing multiple methods with the same name by variations in the arguments list. is called as method overloading.

→ Variation in the argument lists means
(i) variation in the no. of arguments.
(ii) Variation in the data type of the arguments.
(iii) Variation in the position of the arguments.

→ If there is a common task (operation) to be performed with the variations in the i/p. Then we will choose method overloading.

Ex:→

If we want to develop the pgm for addition of 2-nos & 3-nos then we should develop 2-methods which should receive 2-i/ps & 3-i/ps. instead of developing these 2-methods with different names best approach is to develop with the same name "add".

Through this method overloading
i) We can achieve consistency in the method names

which are developed for common purpose.

(ii) We can remember the method names easily.

(iii) ~~Pgm~~ There will be efficiency in pgm readability.

(iv) We can achieve compile time polimorphism.

(v) Note

→ We can overload both static & non-static methods.

→ While overloading if method names should be same & if method names are same there should be some kind of variation with the argument list. Other parts of the method can be anything.

→ **Can we overload main method?**

Yes, we can overload main method. & pgm execution starts from the main method with "String[] args" argument & other versions of the main method should be invoked by the pgmmer.

↳ Ans if only interviewer asks.

Constructor

→ It is the special method which is used to Construct or Create or initialize the object.

→ Whenever we create an object for a Class Constructor will be executed.

→ Constructor name will be similar to classname.

→ Constructors are like methods because - Constructor will never have a return type & Constructor Can't return any value.

→ Ex:→

```
① class ClassName
  {
    ClassName()
    {
    }
  }

② A av=new A();
```

→ Constructors are also called as non-static initializers which is used to initialize an object & Can't be static.

→ Constructor Can have all the access levels.

→ Why Constructor?

To create an object.

```
Class A
{
   A()
   {
     S.O.P("Running Constructor");
   }

   Public static void main(String[] args)
   {
    S.O.P("The main starts");
    A ov1 = new A();
    S.O.P("------");
    A ov2 = new A();
   S.O.P("The main ends");
   }
}
```

O/P:→

```
The main starts.
Running Constructor.
------
Running Constructor.
The main ends.
```

② Ans

Every java class should have a constructor If the pgmmer doesn't develop a constructor for a class then java gives a default Constructor for that class.

Default Constructor.

The Constructor developed by the compiler at Compile time automatically is called as default Constructor.

Ex:→

①
```
ClassName()
{
}
```

②
```
A()
{
}
```

Differances b/w methods & Constructor.

Method.	Constructor.
1) Method can have any name, it can also have class name.	1) Constructor name should be similar to class name.
2) Methods should have return type atleast void.	2) It should never have return type, not even void.
3) Methods can be static or non-static.	3) Constructors should be always non-static.
4) Methods may or may not return a value. [void & non-void methods]	4) It never return a value.
5) Java will not provide default methods. Java ~~provides default Constructor~~.	5) Java provides default Constructor.

I·O Ex:→

```
Class C
{
  Void C()// method with class name.
  {
    S.O.P ("Constructor Ah?"); // No.
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    C cv=new C(); //Call to default Constructor
    S.O.P ("The main ends");
  }
}
```

O/P:→

The main starts.
The main ends.

Imp 4-Steps in Object Creation:→

Ex

A av = new A();

1) New operator will create a new address space in the heap memory.
2) All the non-static members of the class will be loaded onto the address space with default values.

**

3) Constructor body will be executed on the stack like a method.

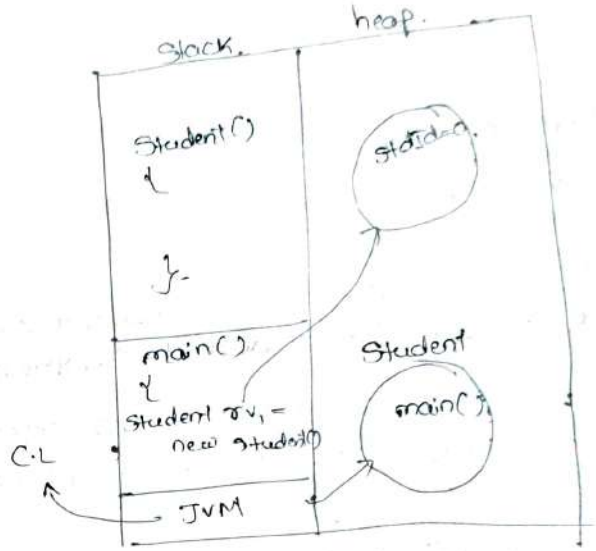
4) Address of the object will be assigned to reference variable.

Ex:→

```
Class Student
{
  int StdId;

  Student()
  {
    S.O.P ("Running Constructor");
  }

  Public static void main(String[] args)
  {
    S.O.P ("The main starts");
    Student rv1 = new Student();
    S.O.P ("The Student Id is" + rv1.StdId);
    S.O.P(" The main ends ");
  }
}
```

O/P:→

```
The main starts.
Running Constructor.
The Student Id is 0.
The main ends.
```




```
Class Employee
{
  int empId;
  Employee()
  {
    empId = 10;
  }

  Public static void main(String[] args)
  {
    S.O.P("The main starts...");
    Employee rv1 = new Employee();
    S.O.P ("The employee ID is" + rv1.empId);
    S.O.P ("-------");
    Employee rv2 = new Employee();
    S.O.P ("The employee ID is" + rv2.empId);
    S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
The employee Id is 10.
= = = = = = = = =
The employee Id is 10.
The main ends.
```

* IQ Why pgmmer should develop Constructor if java already gives a default Constructor?

→ If pgmmer wants to Construct the objects in his own way then pgmmer should develop Constructor.

→ Pgmmer Can utilize Constructor body for initialize the instant variables at the time of object Creation.

→ If pgmmer wants to execute some startup Code whenever object is Created for a Class. then pgmmer Can utilize Constructor body.

→ There are 2-types of Constructor.

(1) Parameterized Constructor
(2) Constructor without Arguments or No Args Constructor.

NOTE:→

Every Default Constructor will fall under "no args" Category.

Ex:→

```
A()
{
}
```

No Args

(2) A(int a)
{
}

Parameterized Constructor.

5/1/15

```
Class Demo1
{
  int empId;
  Demo1(int id)
  {
    empId = id;
  }
  public static void main(String [] args)
  {
    S.O.P("The main starts...");
    Demo1 ov1 = new Demo1(54652);
    S.O.P("The 1st employee ID::" + ov1.empId);
    S.O.P("= = = = = = = = =");
    Demo1 ov2 = new Demo1(7865);
    S.O.P("The 2nd employee ID::" + ov2.empId);
    S.O.P("The main ends...");
  }
}
```

O/P:→

```
The main starts.
empId=54652
The 1st empID :: 54652
= = = = = = = = =
The 2nd empID :: 7865.
The main ends.
```

Y:9

```
Class Demo2
{
 int empId
 Demo2(int id)
 {
  empId=id;
 }
public static void main(String[] args)
{
 S.O.P("The main starts...");
 Demo2 ov1=new Demo2(); // Error.
 S.O.P("The employee Id is"+ov1.empId);
 S.O.P("The main ends");
 }
}.
```

O/P:→ CTE.

1) If you wants to create an object for a class then you should utilize the available Constructor of that class.

2) If pgmmer develops a constructor for a class then java will not give default Constructor for that class.

So, the above pgm is resulting in error.

```
Class Demo3
{
 int StdId;
 Demo3(int id)
 {
  StdId=id;
 }
 Demo3()
 {
  S.O.P("Running Demo3()");
 }
 Public static void main(String[] args)
{
 S.O.P("The main starts");
 Demo3 ov1=new Demo3(10);
 S.O.P("The Student Id is"+ov1.StdId);
 S.O.P("=-=-=-=");
 Demo3 ov2=new Demo3();
 S.O.P("The Student Id is"+ov2.StdId);
 S.O.P("The main ends..");
 }
}
```

O/P:→

```
The main starts.
The Student Id is 10
Running Demo3()
The Student Id is 0.
The main ends.
```

Constructor overloading

Developing multiple Constructors for a class with the variation in no, Data type & position of the arguments.

Advantage:→

1) Through Constructor overloading we will get lots of options to create an object for a class.

Ex:→

```
Class Sample1
{
  Sample1()
  {
    S.O.P ("Running Sample1()");
  }

  Sample1 (int a)
  {
    S.O.P("Running Sample1 (int a)");
  }

  Sample1(double d)
  {
    S.O.P ("Running Sample1(double d)");
  }

  Sample1 (double d, int a)
  {
    S.O.P (" Running Sample1 (double d, int a)");
  }

  Sample1 (int a, double d)
  {
  S.O.P (" Running Sample1 (int a, double d)");
```

```
  public Static void main (String[] args)
  {
    Sample1 sv1 = new Sample1();
    Sample1 sv2 = new Sample1(10);
    Sample1 sv3 = new Sample1(10.5);
    Sample1 sv4 = new Sample1(10.5,10);
    Sample1 sv5 = new Sample1(10, 10.5);
    S.O.P (" Done!!");
  }
}
```

O/P:→

```
Running Sample1()
Running Sample1(int a)
Running Sample1 (double d)
Running Sample1 (double d, int a)
Running Sample1 (int a, double d).
Done!!
```

Volume of Box & Cubic

```
Class Box
{
  int length;
  int breadth;
  int height;

  Box(int l, int b, int h)
  {
    length = l;
    breadth = b;
  } height = h;

  Box (int side)
  {
   length = side;
   breadth = side;
   height = side;
  }

  Void volume()
  {
   int v = length * breadth * height;
   S.O.P("The volume of box is" +v);
  }

Public static void main(String[] args)
  {
    Box bv1 = new Box(2, 3, 4);
      bv1.volume();
    Box bv2 = new Box(5, 6, 7);
      bv2.volume();
    Box bv3 = new Box(8);
      bv3.volume();
  }
}.
```

O/P:→

```
The volume of Box is 24
The volume of Box is 210.
The volume of Box is 512.
```

* this() Calling Statement

→ this() Calling Statement is used to call one constructor from another Constructor of the same Class in case of Constructor overloading.

→ Generally for one object Creation only one Constructor will be executed. but if you want to execute multiple Constructors for one object Creation then we can make use of this() Calling Statement.

Ex:→

```
Class A
{
  A()
  {
   S.O.P(" from A()");
  }

  A(int a)
  {this(); // this() Calling Stmt.
   S.O.P("-from A(int a)");
  }
```

```
public static void main(String[] args)
{
  S.O.P("The main starts");
  A av=new A(90);
  S.O.P("The main ends");
}
}.
```

O/P:→

```
The main starts.
from A()
from A(int a)
The main ends.
```

(2)

```
Class B
{
  B()
  {
    S.O.P("from B()");
  }
  B(int a)
  {
    this();
    S.O.P("from B(int a)");
  }
  B(int a, double d)
  {
    this(90);
    S.O.P("from B(int a, double d)");
  }
```

```
public static void main(String[] args)
{
  S.O.P("The main starts");
  B bv=new B(10, 10.5);
  S.O.P("The main ends");
}
}.
```

O/P:→

```
The main starts.
from B()
from B(int a)
from B(int a, double d).
The main ends.
```

Recursive Constructors:→

```
Class C
{
  C()
  {
    this(90); //Error.
    S.O.P("from C()");
  }
  C(int a)
  {
    this();
    S.O.P("from C(int a)");
  }
```

```
C(int a, double d)
{
  this(90);
  S.O.P("from C(int a, double d)");
}

Public static void main(String[] args)
{
  S.O.P("The main starts");
  C r=new C(90, 90.5);
  S.O.P("The main ends");
}
}.
```

O/P:→ CTE.

Recursive Constructor invocation

Rules of this()

→ this Calling

```
Class F
{
  F()
  {
   S.O.P("from F()");
  }

  F(int a)
  {
   S.O.P("from F(int a)");
   this(); //Error.
  }

  public static void main(String[] args)
  {
   F r=new F(90);
   S.O.P("Done!!");
  }
}
```

O/P:→ CTE.

Rules to use this().

1) this() Calling statement should be first Statement inside Constructor.
2) We can't write multiple this() Calling Statements inside a single Constructor.
3) this() Calling statement can be developed only inside Constructor.

NOTE:→

Constructor can be invoked by its name only at the time of object Creation with new() operator.

Ex :→

```
new A();
```

16/1/15:

Inheritance. [is a relationship]

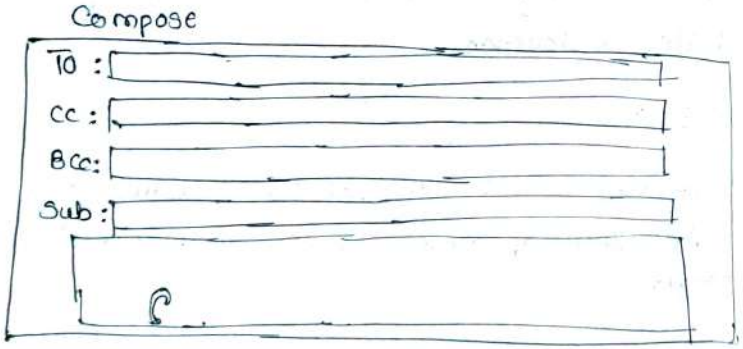
Inheritance is one of the principles of objec Oriented pgming language.

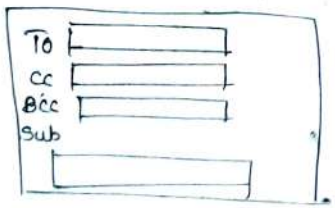
→ Inheritance is nothing but aquiring features or properties of one class into another class [With respect to classes]

→ Through inheritance we can achieve extensibi Code reusability, Code optimization, maintainabil

→ To achieve inheritance we should use extn Keyword.

** → Inheritance is only for non-static member






```
Class BasicCalci
{
    addition()
    Subtraction()
    multiplication()
    division()
}
```

Super or Base or parent.

```
Class ScientificCalci extends
{
    sinθ
    cosθ
    tanθ
    cotθ
}
```

Sub or Child or derived.

→ The class from which ~~they~~ we will aquire the features is called as SuperClass.

→ The class which will aquire the features is called as Subclass.

```
Class A
{
    Void test1()
    {
      S.O.P("Running test1 method");
    }
}

Class B extends A
{
    Void test2()
    {
      S.O.P("Running test2 method");
    }
}

Class Test1
{
  public static Void main(String[] args)
  {
    S.O.P("The main starts");
       B bv=new B();
          bv.test1();
          bv.test2();
    S.O.P("The main ends");
  }
}
```

O/P:->

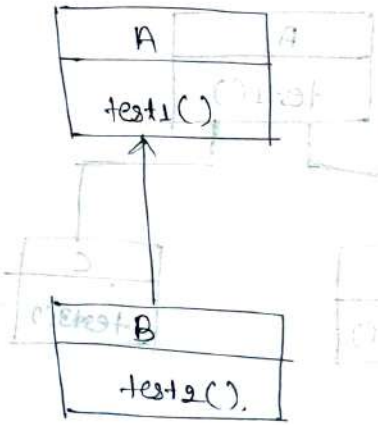
```
The main starts.
Running test1 method.
Running test2 method.
The main ends.
```

These are 4-types of inheritance.

1) Single level.
2) Multi "
3) Hybrid or hierarchical
4) Multiple.

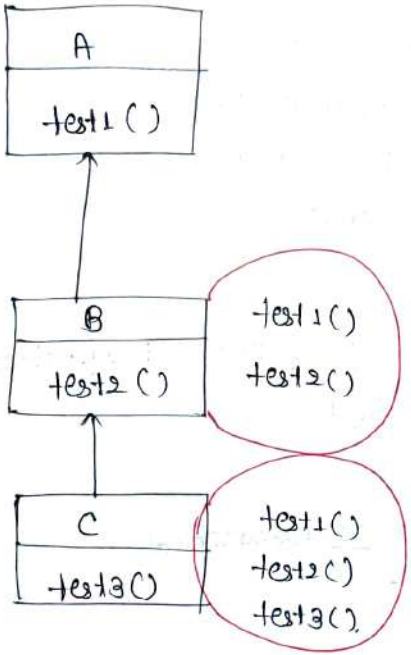
1) Single level inheritance.

One class inheriting from only one super class is called as single level inheritance.



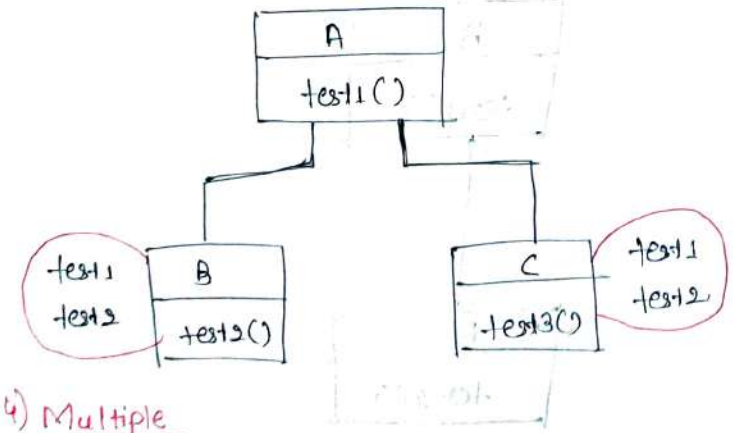
2) Multilevel:

One class inheriting from another subclass is called as multilevel inheritance.



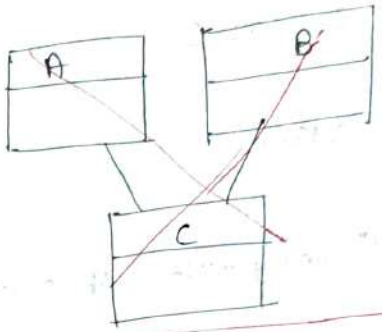
3) Hybrid / Hierarchical.

Lots of classes inheriting from Common Super-class is called as hybrid inheritance.



4) Multiple

One class inheriting from multiple immidiate Superclasses. Called as multiple inheritance.



Multiple inheritance is not possible in java pgming language through classes.

Multilevel inheritance

Ex :->

```
Class C
{
  void test1()
  {
    S.O.P ("Running test1() of C class");
  }
}

Class D extends C
{
  void test2()
  {
    S.O.P ("Running test2() of D class");
  }
}
```

```
Class E extends D
{
  Void test13()
  {
   S.o.p("Running test13() of E Class");
  }
}

Class Testeg2
{
 public static void main(String[] args)
{
 S.o.P("The main starts..");
  E e1 = new E();
    e1.test11();
    e1.test12();
    e1.test13();
  S.o.P("=======");
   D d1 = new D();
    d1.test11();
    d1.test12();
    S.o.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
The main ends
Running test11() of C class.
Running test12() of D Class.
Running test13() of E class.
```

```
Running test11() of C Class.
Running test12() of D Class
The main ends..
```

Hybrid

Ex:→

```
Class F
{
   void test11()
   {
    S.o.P("Running test11() of F Class");
   }
}

Class G extends F
{
  Void test12()
  {
   S.o.P("Running test12() of G class");
  }
}

Class H extends F
{
  Void test13()
  {
   S.o.P("Running test13() of H class");
  }
}
```

```
Class Tester3
{
  public static void main (String[] args)
  {
    S.O.P ("The main starts");
    G g1 = new G();
      g1.test1();
      g1.test2();
    S.O.P(" ==========");
    H h1 = new H();
      h1.test1();
      h1.test3();
    S.O.P ("The main ends");
  }
}.
```

O/P :→

```
The main starts....
Running test1() of F Class.
Running test2() of G Class.
----------
Running test1() of F Class
Running test3() of H Class.
The main ends....
```

Ques

```
Class Box
{
  int length;
  int breadth;
  int height;
  Box(int l, int b, int h)
  {
    length = l;
    breadth = b;
    height = h;
  }
  Box(int side)
  {
    this(side, side, side);
  }
  void volume()
  {
    int v = length * breadth * height;
    S.O.P("The volume of Box is "+v);
  }
  public static void main (String[] args)
  {
    Box b1 = new Box(8);
      b1.volume();
  }
}
```

Using this() Calling Statement we can utilize the initialization code written in other Constructors of the same class.

O/P :→ The volume of Box is 512.

this Keyword.

'this' Keyword is a default reference variable in java.

→ It is used to access current instance member this will be holding the current object address

```
Class A
{
  int empId;
  A(int empId)
  {
    this.empId = empId;
  }
  public static void main(String[] args)
  {
    S.O.P ("The main starts");
    A rv1 = new A(10);
    S.O.P ("The employee ID is" + rv1.empId);
    S.O.P ("The main ends");
  }
}
```

O/P :→

```
The main starts
The employee ID is 10
The main ends.
```

```
Class B
{
  int i;
  int j;
  B(int i, int j)
  {
    this.i = i;
    this.j = j;
  }
  public static void main(String[] args)
  {
    B rv = new B(10,20);
    S.O.P ("The value of i is" + rv.i);
    S.O.P ("The value of j is" + rv.j);
    S.O.P ("Done!!");
  }
}
```

O/P:→

```
The value of i is 10.
The value of j is 20.
Done!!
```

```
Class C
{
String name;
Void test (String name)
 {
  this
  test. name = name;
 }
public static void main (String[] args)
 {
  S.O.P (" The main starts");
   C rv1 = new C();
    rv1.test ("Ramesh");
   S.O.P (" The name is" +rv1.name);
    C rv2 = new C();
     rv2.test ("suresh");
    S.O.P (" The name is" +rv2.name);
    S.O.P (" The main ends");
 }
}
```

O/P:→

The main starts.
The name is Ramesh.
The name is suresh.
The main ends.

```
Class D
{
 int id;
 Void display (int id)
 {
  S.O.P (this.id);
  S.O.P (id);
 }
public static void main (String[] args)
{
 S.O.P (" The main starts");
 D rv1 = new D();
  rv1. display (90);
 S.O.P ("-----");
 D rv2 = new D();
  rv2. display (190);
 S.O.P (" The main ends");
 }
}
```

If local & Global variable names are same then to access global variable we can use this keyword

O/P:→

The main starts.
0
90

0
190.
The main ends.

Business Class & main Class.

→ In one java file we can write multiple classes.

→ The class in which we will develop business logics is called as business class.

→ The class in which we will develop main method is called as main class.

→ If a java file is having multiple classes then that java file name should be main class name.

→ Whenever we compile any java program which is associated with multiple classes then we will get multiple .class files based on class name.

```
Class Student // Business Class.
{
  Static int StdId = 90;
  Static void percentage()
  {
    S.O.P ("Running percentage method");
  }
}

Class A // main class.
{
  public static void main (String[] args)
  {
    S.O.P ("The main starts");
    Student.percentage();
    S.O.P ("The StudentID is" + Student.StdId);
    S.O.P ("The main ends");
  }
}
```

O/P:→ JavaC javaA.java

```
The main starts
Running percentage method.
The studentId is 90.
The main ends.
```

```
Class Employee.
{
  int empId = 10;
  Void Salary()
  {
    S.O.P ("Running Salary method");
  }
}

Class mainClass.
{
  public static void main (String[] args)
  {
    S.O.P ("The main starts");
    Employee ov = new Employee();
    ov.Salary();
    S.O.P ("The employee Id is" + ov.empId);
    S.O.P ("The main ends");
  }
}
```

O/P:→

```
The main starts.
Running Salary method.
The employee Id is 10.
The main ends.
```

```
Class C
{
   Public static void main (String[] args)
   {
     S.O.P("Hey world");
   }
}

Class D
{
 Public static void main (String[] args)
   {
     S.O.P("Hello world");
   }
}
```

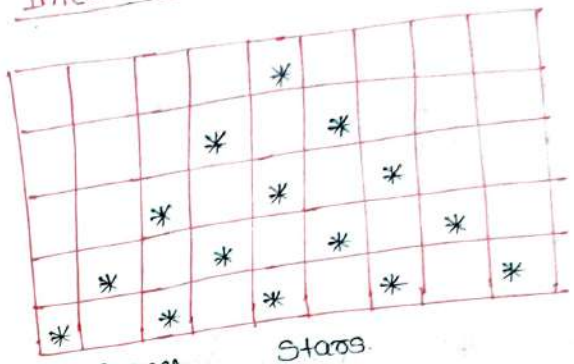
O/P :→

Java C D.java

java C
Hey world.

java D
Helloworld.

Interview.



Spaces	Stars
4	1
3	2
2	3
1	4
0	5.

```
Class A
{
 Public static void main (String[] args)
   {
    for (int i=1; i<=5 ; i++)
     {
      for (int S=5; S>5; S--)
     {
        S.O.P(" ");
     }
     for (int j=1; j<=i; j++)
     {
      S.O.P("*_ "); or S.O.P("*");
     }
      S.O.Pln( );
} } }
```

O/P:→

```
- - - - *
- - - * *
- - * * *
- * * * *
* * * * *
```

Spc	Str
4	1
3	2
2	3
1	4
0	5

Class A

```
{
public static void main (String[] args)
 {
  for (int i=1; i<=5; i++)
   {
    for (int s=5; s>5; s--)
    {
      S.o.P (" ");
    }
    for (int j=1; j<=i; j++)
    {
      S.o.P (j+" ");
    }
      S.o.Pln ();
   }
 }
}.
```

O/P:→

```
        1
      1   2
    1   2   3
  1   2   3   4
1   2   3   4   5.
```

Class B

```
{
 public static void main (String[] args)
 {
  for (int i=1; i<=5; i++)
   {
    char ch='A';
    for (int s=1; s<5; s--)
    {
      S.o.P (" ");
    }
    for (int j=1; j<=i; j++)
    {
      S.o.P (ch+" ");
      ch++;
    }
    S.o.Pln();
   }
 }
}.
```

O/P:→

```
        A
      A   B
    A   B   C
  A   B   C   D
A   B   C   D   E.
```

```
Class C
{
Public static void main (String[] args)
{
  Char ch = 'A';
  for (int i=1; i<=5; i++)
  {
    for (int s=1; s<5; s--)
    {
      S.O.P(" ");
    }
    for (int j=1; j<=i; j++)
    {
      S.O.P (ch+" ");
      ch++;
    }
    S.O.Pln ();
  }
}
}
```

O/P:→

A

B C

D E F

G H I J

K L M N O.

NOTE:→

```
Class D
{
Public static void main (String[] args)
{
  int a=1;
  for (int i=1; i<=5; i++)
  {
    for (int s=1; s<5; s--)
    {
      S.O.P(" ");
    }
    for (int j=1; j<=i; j++)
    {
      S.O.P (a+" ");
      a++;
    }
    S.O.Pln ();
  }
}
}
```

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

```
Class A
{
 Public Static void main(String[] args)
  {
   for(int i=1; i<=5; i++)
    {
      for(int j=1; j<=i; j++)
      {
       S.O.P((i+j)%2);
      }
      S.O.Pln();
    }
 }
}
```

O/P:→

```
0
1 0
0 1 0
1 0 1 0
```

```
Class C
{
public static void main(String[] args)
 {
  for(int i=5; i<=5; i++)
  {
  for(int s=5; s>i; s--)
        {
        S.O.P(" ");
        }
  for(int j=1; j<=i; j++)
  {
    S.O.P("*");
  }
  S.O.Pln();
 }
}
```

O/P:→

```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
Class D
{
Public Static void main(String[] args)
{
  for(int i=1; i<=5; i++)
  {
   for(int s=5; s>i; s--)
    {
     S.O.P(" ");
    }
  for(int j=1; j<=i; j++)
  {
    S.O.P("*");
  }
   S.O.Pln();
 }
```

```
for(int i=4; i>=1; i--)
 {
  for(int s=5; s>i; s--)
   {
    S.o.p(" * ");
   }
   S.o.pln();
  }
 }
}
```

O/P:→

* Write a java program to swap 2-nos with out using 3rd variable.

```
Class E
{
 Static void swap(int a, int b)
 {
  S.o.p("Before swapping");
  S.o.p("a="+a);
  S.o.p("b="+b);

  a=a+b;        a=2+3=5
  b=a-b;        b=5-3=2
  a=a-b;        a=5-2=3

  Or
    a=(b-a)+(b-a);
  S.o.p("After swapping");
  S.o.p("a="+a);
  S.o.p("b="+b);
 }
 public static void main(String[] args)
 {
  S.o.p("The main starts");
  swap(10,20);
  S.o.p("The main ends");
 }
}
```

Write a java program to generate 1st 10 fibonacci nos.

```
Class F
{
 Static void fibonacci(int num)
 {
  S.o.p("The fibonacci series is");
  int a=0;
  int b=1;
  S.o.p(a);
  S.o.p(b);
  for(int i=3; i<=num; i++)
  {
   c=a+b;
   S.o.p(c);
   a=b;
   b=c;
  }
  Public Static void main(String[] args)
 {
  S.o.p("The main starts");
  fibonacci(10);
 } S.o.p("The main ends");
```

Write a java program to check whether given no. is prime or not.

```
Class G
{
public static void main(String[] args)
{
  int num=17;
  boolean b=true;
  for(int i=2; i<=num-1; i++)
  {
    if(num%i==0)
    {
      b=false;
      break;
    }
  }
  if(b==true)
  {
   S.O.P("Prime number"+num);
  }
  else
  {
   S.O.P("Not a prime no"+num);
  }
}
}
```

```
Class G
{
static void prime(int num)
{
  boolean b=true;
  for(int i=2; i<=num-1; i++)
  {
    if(num%i==0)
    {
      boolean = false;
        break;
    }
  }
  if(b==true)
  {
   S.O.P("prime numbers"+num);
  }
  else
  {
   S.O.P("Not a prime number"+num);
  }
}
Public static void main(String[] args)
{
  prime(17);
}
}
```

To print 1st - 20 prime nos:→

```
Class G
{
Static void prime(int num)
{
S.O.P("The first 20 prime nos. are");
   int num=2;
   int Count=1;
   while (Count<=20)
  {
   boolean b=true;
   for (int i=2; i<=num-1; i++)
    {
      if (num%i==0)
      {
        boolean=false;
          break;
      }
    }
   if (b==true)
   {
    S.O.P("Prime number"+num);
      Count++;
    }
   num=num++;
}
Public static vàd main(String[] args)
   {
      prime();
} }
```

To generate +ve prime nos. within 20.

```
Class E
{
public static void main(String[] args)
 {
  int num=2;
  boolean b=true;
  for (i=2; i<=20; i++)
   {
    if (num%i==0)
    {
     boolean b=false;
       break;
    }
  }
  if (b=true)
  {
   S.O.P("Prime number"+num);
  }
  num++;
 }
}
```

13/11/2015

Static blocks or S.I [Static Initialization block.]

```
Class A
{
 Static
 {
  S.O.P ("from 1st SIB");
 }
public static void main(String[] args)
{
 S.O.P ("Done!!");
}
Static
{
 S.O.P("from 2nd SIB");
}
}.
```

O/P:->

from 1st SIB.
from 2nd SIB
Done!!

```
Class B
{
Static int a;
Static
{
 S.O.P ("from SIB"+a);
 a=10;
}
public static void main(String[] args)
{
 S.O.P("from main="+a);
}
}.
```

O/P:->

from SIB = 0
from main=10.

```
Class C
{
Static
 {
  S.OP ("from SIB");
 }
}.
```

O/P:->

Runtime Error.

"To run any Core java pgm main method is mandatory".

Non-static blocks:→

```
Class A
{
{ S.O.P("-from 1st IIB");
}
public static void main(String[] args)
{
 S.O.P("The main starts");
 A rv=new A();
 S.O.P("=======");
 A rv2= new A();
 S.O.P("The main ends");
}
{
 S.O.P("-from 2nd IIB");
}
}.
```

O|P :→

```
The main starts.
from 1st IIB.
-from 2nd IIB
=========
from 1st IIB.
-from 2nd IIB-
The main ends
```

We can use IIB's for Creating instance variables It is similar to Constructor.

```
Class B
{
 int empId;
{
 S.O.P("-from IIB="+empId);
 empId=90;
}
public static void main(String[] args)
{
 S.O.P("The main starts");
  B rv=new B();
 S.O.P("from main=" + rv.empId);
 S.O.P("The main ends");
}
}.
```

O|P:→

```
The main starts.
from IIB 0.
-from main 90.
The main ends
```

We Cann't write IIB Calling Statements:→

```
Class C
{
 C()
 {
  //--> default Calling Statement.
  S.O.P("-from Constructor");
 }
```

```
{
  S.O.P("from IIB");
}.

Public static void main(String[] args)
{
  S.O.P("The main Starts");
  C rv = new C();
  S.O.P("The main ends");
}
}.
```

O/P:→

```
The main starts.
-from IIB.
from Constructor.
The main ends.
```

```
Class D
{
 {
  S.O.P("-from 1st IIB");
 }
 Static
 {
  S.O.P("-from 2nd SIB");
 }
  D()
 {
  S.O.P("Running Constructor");
 }
```

```
public static void main(String[] args)
{
  S.O.P("The main starts");
   D rv = new D();
  S.O.P("The main ends");
}
Static
{
 S.O.P("-from 2nd SIB");
}
{
 S.O.P("-from 3rd IIB");
}
}.
```

O/P:→

```
-from 1st SIB
-from 2nd SIB.
The main starts.
-from 1st IIB
-from 2nd IIB.
Running Constructor.
The main ends.
```

Static blocks are also called as static initialization Block. Which will be executed before main method Static blocks can be used to initialize static. variable & execute some startup code before main method execution.

General Syntax.

```
Static
{
 }
```

→ Static blocks Cann't be Called explicitly.

→ We can develop multiple static blocks for a Class.

→ All the static blocks of the class will be executed in which they are developed.

→ Static blocks will be executed only once in one execution cycle.

→ We Cann't invoke static blocks explicitly it will be invoked automatically after static the static pool of the class is created [Before main]

→ To execute the static block main method is required (from JDK 1.7).

Non-Static blocks.

→ These are also called as instance initialization blocks which will be executed at the time of object creation.

→ Non-static blocks can be used to initialize instance variables.

→ Non-static blocks will not have any names

General Syntax :→

```
{
---
---
}.
```

→ Non static block will be executed whenever objects is created & it will be executed before Construction.

→ We can develop multiple non-static blocks for a class

→ Whenever we Create an object for a class all the non-static block of that class will be executed in the order in which they are developed.

difference b/w static & non-static blocks.

Static	Non-Static.
1) Executes before main method.	1) Executed at the time of object Creation.
2) Static blocks will be executed only once in one execution cycle.	2) These are executed whenever object is Created.

Post Increment:

```
Class A
{
public static void main(String[] args)
 {
   int i=0;
   S.O.P(i++);
   S.O.P(i);
 }
}.
```

O/P:→

0
1.

```
Class B
{
  Public static void main(String[] args)
  {
    int i=0;
    int j=i++;
    S.O.P("i="+i);
    S.O.P("j="+j);
  }
}
```

O/P:→

i=1
j=0.

```
Class C
{
  Public static void main(String[] args)
  {
    int i=1;
    int j=i++ + i;
    S.O.P("i="+i);
    S.O.P("j="+j);
  }
}
```

O/P:→

i=2
j=3

```
i++ + i
1   + 2
=3.
```

```
{
  int i=1
  int j= i++ + i + i++ + i;
}
```

O/P:→

i=3
j=8.

```
j= i++ + i + i++ + i
 = 1   + 2 + 2 + 3
 = 8.
```

```
{
  int i=1;
  int j= i+ i++ + i+i+ i++.
}
```

O/P:→

i=3
j=8.

```
= i+ i++ +i +i+i++
= 1 + 1 + 2 + 2 + 2
=8
```

Pre Increment.

```
Class A
{
  Public static void main(String[] args)
  {
    int i=1;
    int j=++i;
    S.O.P("i="+j);
    S.O.P("j="+j);
  }
}
```

O/P:→ j=2
i=2
j=2.

{

int i=1;

int j= ++i+i++ +++i;

}.

O/P:→

i=3

j=7

j= ++i+i++ +i

= 2+2+3.

= 7.

{

int i=1;

int j= ++i+ i++ +i+i++ + ++i.

}.

O/P:→

i=5

j=15.

⇒ ++i +i++ +i + i++ + ++i

⇒ 2+2+3+3+5

= 15

{

int i=1;

int j= i++ + i-- + ++i + --i + i++ + i + i++.

}

O/P:→

i=3

j=11 ,

i++ + i-- + ++i + --i + i++ + i + i++

⇒ 1+2+2+1+2+2+2

{

int i=1

int j= i++ + i-- + ++i + --i + i++ + i + i++;

}

O/P:→

i=3

j=12.

i++ + i-- + ++i + --i + ++i + i + i++

⇒ 1+2+2+1+2+2+2

= 12

17/11/15

Inheritance Continued Part.

Super Calling Statement:→

1. Super Calling stmt is used to call Super class Constructor in case of inheritance b/w Classes.
2. Super Calling stmt is used to call immidiate Super Class Constructor.
3. Through Super Calling stmt we can achieve Constructor chaining
4. Constructor chaining means subclass Constructor Calling its immidiate Super class Constructor.

NOTE:→
Constructor chaining can be achieved through Super Calling stmt only.

If there is inheritance b/w classes then Constructor chaining is manditory. Rule.

Ex:→

```
Class A
{
  A()
  {
    S.O.P("Running Constructor of Class A");
  }
}
Class B extends A.
{
  B()
  {
    Super();
    S.O.P("Running Constructor of Class B");
  }
}
Class C extends B
{
  C();
  {
    Super();
    S.O.P("Running Constructor of class C");
  }
}
Class Tester1
{
  Public static void main (String[] args)
  {
    S.O.P("The main starts..");
    C cv=new C();
    S.O.P("The main ends");
  }
}.
```

O/P:→

```
The main starts.
Running Constructor of Class A
  "          "       "    "  B
  "          "       "    "  C
The main ends.
```

* If Pgmmer not written super Calling stmt the Compiler itself Call a default super Calling stmt.

```
Class D
{
  D()
  {
   S.O.P("from Constructor of D Class");
  }
}

Class E extends D
{
  E()
  {
  S.O.P("from Constructor of E Class");
  }
}

Class Tester2
{
 public static void main(String[] args)
  {
   S.O.P("The main starts");
   E rv=new E();
   S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
from Constructor of D Class
  "      "    "  E  "
The main ends.
```

Default Constructor & Default Super

```
Class F
{
  F()
  {
   S.O.P("from F Class Constructor");
  }
}

Class G extends F
{
   /* --> Default Constructor
      G()
      {
      super(); --> Default super() Calling Statement
      }*/
}

Class Tester3.
{
 public static void main(String[] args)
  {
   S.O.P("The main starts");
   G rv=new G();
   S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
from F Class Constructor.
The main ends.
```

```
Class H
{
  H(int a)
  {
    S.O.P("from H(int a)");
  }
}

Class I extends H.
{
  I()
  {
    Super(10); // If it is not written then error.
    S.O.P("from I()");
  }
}

Class Testeay
{
  Public static void main(String[] args)
  {
    S.O.P("The main starts");
    I rv=new I();
    S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
from H(int a)
from I()
The main ends
```

NOTE

① If their is inheritance b/w classes & if the pgmmer has not developed super Calling Stmt then there will be a default super Calling stmt in the subclass Constructor.

② Super (parameter) will Call the immidiate Super Class Constructor with the matching parameter.

Ex:→ Super(10) will Call the immidiate super class Constructor with the integer argument

Rules of Super Calling Stmt :→

1) Super Calling Stmt should be always developed as the first stmt inside the Constructor
2) We Can't develop multiple super Calling Stmt inside single Constructor
3) We Can develop the super Calling stmt only inside a Constructor.

Similarities b/w this() Calling Stmt and Super() Calling Stmt.

1) Both Can be developed only inside Constructor
2) Both are used to Call Constructor only.
3) Both should be 1st stmt inside the Constructor.
4) we Can't develop multiple this Calling Stmts or multiple super Calling Stmt inside single Constructor.

differances b/w this & super stmts

this()	super()
1) It is used to call the current class constructor	1) It is used to call super class constructor.
2) It can be used in case of constructor overloading.	2) It can be used in case of inheritance b/w classes.
3) There will be no default this calling stmts.	3) There will be default super calling stmt.
4) By using this() calling stmt we can utilize the initialization code developed in the other constructor of the current class.	4) Through super() calling stmt we can achieve constructor chaining. [If there is constructor chaining then only super class non-static will be inherited to subclass objects.]

IQ :->

```
Class J
{
    J()
    {
     S.O.P("from J()");
    }
}

class K extends J
{
  K(int a)
  {
    S.O.P("from K(int a)");
  }

  K()
  {
    this(10);
    S.O.P("from K()");
  }
}

Class L extends K
{
    L()
    {
     S.O.P("from L()");
    }
}
```

```
Class Tstest5
{
public static void main(String[] args)
{
 S.O.P("The main starts");
  L rv=new L();
 S.O.P("The main ends");
 }
}
```

O/P:→

The main starts.
-from JC()
from K(int a)
from K()
from LC()
The main ends.

The default Class "Object."

```
Class A // extends Object.
{
  A()
  {
   // super(); /* it is by defaultly called*/
   S.O.P("from Constructor");
  }
 public static void main(String[] args)
 {
  S.O.P("The main starts");
   A rv=new A();
  S.O.P("The main ends");
 }
}
```

O/P:→

The main starts
from Constructor
The main ends.

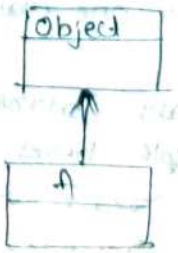
→ Object class is the superm most class for any class in java

or

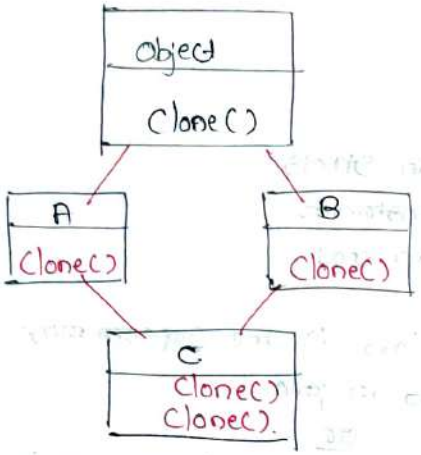
→ Any class is a subclass to object class. auto-matically either directly or indirectly.

[Through multilevels]

→ Object class is a builtin class in java
→ There are lots of non-static methods in object class which will be inherited to a every class of java.
→ All the non-static methods of object class will be available in every object of java
→ Some of the non-static methods are :→
  clone()
  finalize()
  hashCode() etc.



Why multiple inheritance is not possible in java through classes.?



1) Multiple inheritance is not possible in java through classes because it will create a-ambiguity.
2) In case of multiple inheritance one class will be inheriting from multiple immidiate Super classes.
3) In case of inheritance classes constructor chaining is manditory & Constructor chaining can be achieved through super calling statements.
4) In this scenario we may have to develop multiple super calling statement which is not possible because we can't develop multiple super calling statements in a single constructor.
5) Diamond pbm or path issue:→

→ In case of multiple inheritance there will be multiple paths established blw one class to object class.

→ In this scenario object class features will be inherited to one class multiple times through multiple paths which will create ambiguity of method usage this is called as diamond pbm or path issue.

Method overriding.

Changing the implementations of super class methods into the subclass according to the needs of the subclass called as method overriding.

Or

Redefining the superclass methods into subclass according to the needs of the subclass. is called as method overriding.

→ To achieve method overriding 3-things are manditory :→

1. Inheritance.
2. Non-static methods.
3. Signature or Complete method declaration should be same.

→ Whenever we perform method overriding superclass implementation of that method will be lost in the subclass.
i.e. we will get the latest implementation of the subclass.

Ex:->

```
Class A
{
  void test1()
  {
    S.O.P("test1() of Super Class");
  }
}

Class B extends Class A
{
  Void test1()
  {
    S.O.P("test1() of Sub Class");
  }
}

Class Test01
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    B bv = new B();
    bv.test1();
    S.O.P("The main ends");
  }
}
```

O/P:->

```
The main starts.
test1() of Sub Class
The main ends.
```

Advantages of method overriding-

1) Through method overriding we can achieve run time polymorphism.
2) We can achieve standardization
3) We can achieve abstraction.
4) Performance will be improved.

NOTE:->

1) In the ~~above~~ below pgm smoking is the method of super class getting overriden in the subclass with the new implementation. in subclass.
2) At the time of object creation itself method overriding will happen.

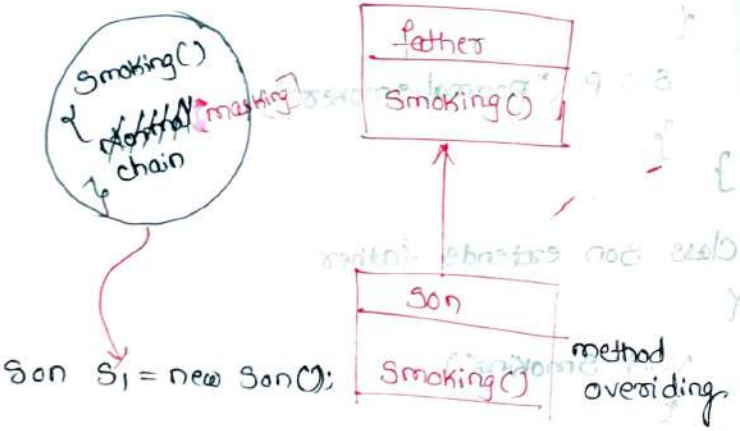
Ex:->

```
Class Father
{
  void Smoking()
  {
    S.O.P("normal smoker");
  }
}

Class Son extends father
{
  void Smoking()
  {
    S.O.P("Chain Smoker");
  }
}
```

```
Class Tester2
{
Public Static void main(String[] args)
{
S.o.p("The main starts");
Son S1 = new Son();
S1.smoking();
S.o.p("The main ends");
}
}
```

O/P:→

The main starts.
Chain Smoker
The main ends.



Super Keyword.

1) Super keyword is used to access super class non-static members in case of inheritance

2) Effective usage of super keyword is in case of method overriding.
i.e. Due to method overriding super class implementation of that method will be masked in the subclass.
So, if you wants to get old super class implementation then we should use super keyword

Ex:→

```
Class C
{
 void test1()
 {
  S.o.p("Super class implementation");
 }
}
Class D extends C
{
 void test1()
 {
  super.test1();
  S.o.p("Sub class implementation");
 }
}

Class Tester3
{
 Public static void main(String[] args)
 {
```

```
        S.o.P ("The main starts");
          D dv = new D();
           dv.test1();
          S.o.P("The main ends");
      }
  }.
```

O/P:→

The main starts.
Super class implementation.
Sub class "
The main ends.

→ Through method overriding we can improve the performance of an application.

→ Inheritance means "is-a relationship".

→ "has-a" relationship means what features an object is having like attributes or variables & functions.

Ex:→

```
Class person
{
  String fname="vijay";
  String lname="Malya";
   int age=30;
   void display()
   {
    S.o.P(" first name is "+fname);
    S.o.P(" last name is"+lname);
    S.o.P(" age is" ,age);
   }
}

Class Trainer extends Person
{
  double Salary = 90005;
    int yoe = 100;
  String subject=" JAVA";
    void display()
  {
    super.display();
    S.o.P(" Salary is "+Salary);
    S.o.P("Experience is "+yoe);
    S.o.P ("subject is" +subject);
  }
}

Class Testery
{
  public static void main (String[] args)
  {
   S.o.P (" The main starts");
  Trainer t1 = new Trainer();
    t1.display();
   S.o.P("The main ends");
  }
}.
```

method overriding

O/P:→

```
The main starts.
The first name is vijay
last name is malya
age is 90.
Salary is 9000.5
Experience is 100
Subject is JAVA
The main ends.
```

Assignment:→

Enhance the above pgm by using Constructor Concept.

19/11/15 — Final Keyword

```
Class E
{
  final void test1()
  {
    S.O.P("from test1() of Super class");
  }
}

Class F extends Class E
{
  /* void test1()
  {
    S.O.P("from test1() of Sub class");
  }*/
}

Class Tester5
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    F ov=new F();
    ov.test1();
    S.O.P("The main ends");
  }
}
```

O/P:→

The main starts.
from test1() of Super class.
The main ends

What is final?

final is a keyword which is used to avoid overriding.

Explaination:→

→ If a variable declared as final its value can't be changed.

→ If a method is declared as final we can't override that method.

→ If a class is declared as final we can't inherit that class, [So, we can't override any of the final class methods]

```
final Class A
{
  A()
  {
    S.O.P("from class A");
  }
}

Class B extends A   // Error
{
  *
}

Class Tester4
{
  PSVM(String[] args)
  {
  }
}
```

O/P:→ Error

Difference b/w method overloading & overriding

Overloading	Overriding
1) To perform method overloading inheritance is not manditory. i.e. we can overload with & without inheritance.	1) To perform method overriding inheritance is manditory.
2) for method overloading just method names should be same with the variation in type, no & position of the arguments.	2) for overriding Complete declaration of the method should be same including arguments.
3) We can overload both static & non-static methods.	3) We can override only non-static methods.
4) We can overload main method.	4) We can't override main method because it is static.
5) We can overload Constructors.	5) We can't override Constructors because Constructors can't be inherited.
6) We can achieve Compile-time polymorphism	6) We can achieve run time polymorphism
7) While overloading of methods return type can be different.	7) While overriding of methods return type should be same. or there is restriction on return type

NOTE :->

-> Can we override static methods?

No, We can't override static methods because static methods will not be inherited to the subclass. & to achieve method overriding inheritance is manditory.

Can we override main method?

No, because main is a static method & static methods will not be inherited to the subclass & to achieve method overriding inheritance is manditory.

Can we override Constructors?

No because Constructors will not be inherited to subclass & to achieve method overriding inheritance is manditory.

Ex:->

```
Class I
{
    void test1()
    {
        S.O.P("from test1() - Super");
    }
}

Class J extends I
{
    void test1(int a)
    {
```

```
    S.O.P("from test1 (int a) - Sub");
  }
}

Class Tester7
{
  Public static void main (String[] args)
  {
    S.O.P(" The main starts ");
      J ov = new J();
        ov.test1();
        ov.test1(90);
    S.O.P(" The main ends");
  }
}.
```

o/p:→

The main starts.
from test1 (int) - super
from test1 (int a) - Sub
The main ends.

Abstract class.

Concrete

```
Class A
{
  void test1() declaration.
  {
defination
  }
}
```

Abstract

```
abstract Class A
{
  abstract void test1();
}
```

Concrete methods.

The methods which will have declaration with defination are called as Concrete methods.

Ex:→

```
void test1()
{

}.
```

Concrete Class.

The class in which we can develop only Concrete methods. is called as Concrete class.

Ex:→

```
Class A
{

  void test1()
  {
  }
}
```

Abstract method.

The method developed with just declaration without definition is called as abstract method

Ex:→

```
abstract void test1();
```

Abstract Class.

The class in which we can develop abstract methods is called as abstract class.

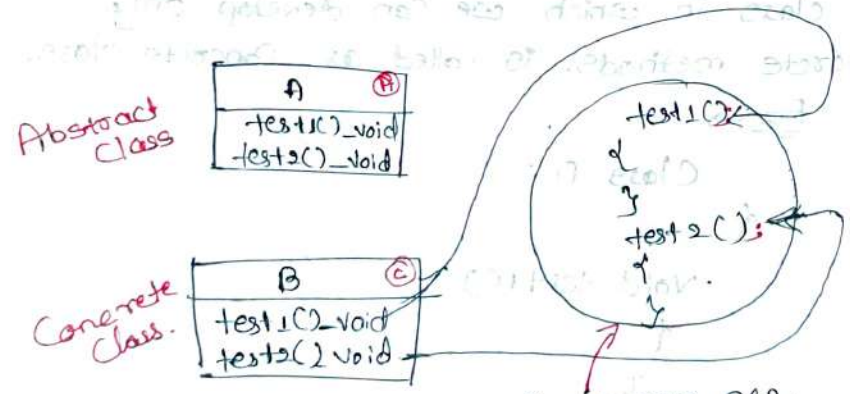
Ex:→

```
abstract Class A
{
  abstract void test1();
}
```

NOTE:→

→ abstract class & abstract methods should be declared with the keyword abstract.

* We can't create an object for abstract class.



→ We can't instantiate an abstract class. it should always have a sub class & the subclass should override all the inherited abstract methods.

→ NOTE

Giving a new implementation for the inherited abstract methods is also called as method overriding.

Advantage of abstract classes.

1) Standardization.
2) Abstraction.

```
abstract Class A
{
  abstract void test1();
  abstract void test2();
}

Class B extends A
{
  void test1()
  {
    S.O.P("Test1() overridden in class B");
  }
  void test2()
  {
    S.O.P("Test2() overridden in class B");
  }
}

Class Tester1
{
  public static void main(String[] args)
  {
```

```
S.O.P ("The main starts");
// A av = new A();   not possible to create abstract class objects
   B av = new B();
     av.test1();
     av.test2();
   S.O.P("The main ends");
  }
}.
```

O/P:→

The main starts.
Test1() overridden in class B

" " " "

The main ends

(2)

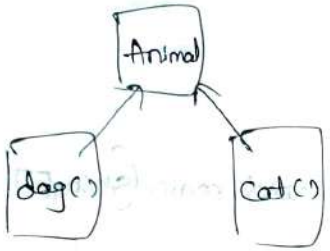
```
abstract class Animal
{
  abstract void move();
  abstract void sound();
}
Class Dog extends Animal
{
   void move()
    {
     S.O.P ("Dog walks");
    }
    void sound()
    {
     S.O.P(" Bow Bow");
    }
}
```

```
Class Cat extends animal
{
  void move()
  {
   S.O.P ("Cat walks");
  }
  void sound()
  {
   S.O.P ("Meow, Meow");
  }
}
Class Tester
{
 Public Static void main (String[] args)
 {
  S.O.P ("The main starts");
   dog av = new dog();
     av.move();
     av.sound();
   S.O.P (" ==========");
   Cat av1 = new Cat();
     av1.move();
     av1.sound();
   S.O.P("The main ends");
  }
}
```

The main starts.
dog walks
Bow Bow

Cat walks
Meow Meow
The main ends.

Inheritance diagram



Hybrid or hierarchical inheritance

```
abstract class College
{
  abstract void goidview();
}
Class student extends College
{
  int StdId;
  String name;
  double percentage;
  Student (int stdId, String name,
                    double percent)
  {
    this.StdId = stdId;
    this.name = name;
    this.percent = percent;
  }
  Void goidview()
  {
    S.O.P ("Student Id =" +StdId);
    S.O.P ("Student name =" +name);
    S.O.P ("Student percentage =" +percent);
  }
}
Class Faculty extends College
{
  int yoe;
  String name;
  this.salary = salary;
}
```

```
void
faculty(int yoe, string name, double salary)
{
  this.yoe = yoe;
  this.name=name;
  this.salary=salary;
}
Void gridview()
{
  S.O.P("faculty experience=" +yoe);
  S.O.P("Name="+name);
  S.O.P("Salary="+salary);
}
}

Class Testers
{
public static void main(String[] args)
{
  S.O.P("The main starts");
  // College cv1=new College();
  Student cv1= new Student(90, "Rama", 99.99);
    cv.gridview();
  S.O.P("****");
  faculty cv2=new faculty (100, "Girish",
                           9999.9);
    cv2.gridview();
} S.O.P("The main ends");
```

```
The main starts
Student Id = 90
Student name = Rama.
Student percent = 99.99.
****
faculty yoe = 100
faculty name = Girish
faculty sal = 9999.9
The main ends
```

```
abstract class DC
{
  abstract void test1();
    void test2()
    {
      S.O.P("test2() method");
    }
}

Class D extends C
{
    Void test1()
    {
      S.O.P("test1() overridden in class D");
    }
}

Class Tester4
{
  Public static void main(String[] args)
  {
    S.O.P("The main starts...");
Error // C r = new C();
      D r = new DC();
        r.test1();
        r.test2();
      S.O.P("The main ends");
  }
}
```

o/p :→

```
The main starts.
test2() method
test1() overridden in class D
The main ends.
```

20/1/15

Standardization.

```
abstract class Benz
{
  abstract void engine();
  abstract void speed();
    void logo()
    {
      S.O.P("Benz logo");
    }
}

Class AClass extends Benz
 {
    void engine()
    {
      S.O.P("AClass Benz engine");
    }

    void speed()
  {
    S.O.P("AClass Benz Can speed upto 200Km/hr");
  }
}

Class BClass extends Benz
{
  void engine()
  {
    S.O.P("BClass Benz engine");
  }

    Void speed()
  {
   S.O.P("BClass Benz Can speed upto 400Km/hr");
}}
```

```
Class Testers.
{
 public static void main (String[] args)
 {
  S.O.P ("The main starts");
  AClass av = new AClass();
        av. engine();
        av. speed();
        av. logo();
  S.O.P ("==========");
  BClass bv = new BClass();
        bv. engine();
        bv. speed();
        bv. logo();
  S.O.P ("The main ends");
 }
}
```

O/P:→

```
The main starts.
AClass Benz engine.
AClass Benz can speed upto 200Km/hr.
Benz logo.
==========
BClass Benz engine.
BClass Benz can speed upto 400Km/hr.
Benz logo.
The main ends
```

In Abstract class we can develop any kind of methods but object can't be created & it should have a subclass which must override only inherited abstract methods. i.e. The subclass need not override Concrete methods (for above pgm).

In Concrete class we can't have abstract methods.

```
abstract Class E
 {
  abstract void test1();
  abstract void test2();
 }

Class F extends E
{
   void test1()
   {
    S.O.P("test1() overridden in class F");
   }

/* void test2()
  {
   S.O.P("test1() overridden in class F");
  }
} */

Class Testers
{
 public static void main (String[] args)
 {
  S.O.P("The main starts");
   F fv = new F();
    fv. test1();
    fv. test2();
```

```
        S.O.P (" The main ends");
    }
}
```

O/P:→

CFE.

Error: [F is not abstract & does not override abstract method of test2() of abstract class E.]

NOTE:→

F is not abstract class & it is not giving inheritance to the test2 method of abstract class

---

One abstract class can inherited from another abstract class but it should have sub class.

```
abstract class G
{
  abstract void test1();
  abstract void test2();
}

abstract class H extends G
{
  void test1()
    {
      S.O.P("test1() overridden in class H");
    }
}

class I extends H
{
  void test2()
  {
  S.O.P(" test2() overridden in class I");
  }
```

```
Class Tester7
{
  public static void main(String[] args)
  {
    S.O.P(" The main starts");
    I rv = new I();
    rv.test1();
    rv.test2();
    S.O.P(" The main ends");
  }
}
```

O/P:→

```
The main starts.
test1() overridden in class H.
test2()   "       "   "   I
The main ends.
```

Q What is abstract class?

1) The class in which we can develop abstract method is called as abstract class.
2) Through abstract class we can achieve standa-rdization & abstraction.
3) In abstract class we can develop any kind of methods. i.e.
   a) We can develop only Concrete methods.
   b) " " " " abstract methods.
   c) Both abstract & Concrete methods
4) Abstract Class can't be instantiated it should always have a subclass & the subclass should follow 2-rules or Contracts.

Rule-1 :→ The subclass should override all the inherited abstract methods.

Rule-2 :→ If the subclass doesn't override all the inherited abstract methods then that subclass should be declared as abstract.

5) Abstract class is not 100% abstract - bcz we can develop any kind of methods in abstract class.

6) Through abstract class we can't achieve 100% abstraction.

Or

Through abstract class we can achieve upto 100% abstraction.

I.Q :→

1) In abstract class can we develop an abstract method as static? No,

```
abstract static void test1()
```

Static methods we can't override but abstract methods should be override.

Or

⇒ No, bcz static methods can't be overridden & abstract methods should be overridden.

2) Can we develop an abstract method as final?

No, bcz final methods can't be overridden and abstract methods should be overridden.

```
abstract final void test1() //Error.
```

3) Can we declare an abstract class ~~class~~ as final?

No, bcz final class can't be inherited & abstract class should be inherited.

4) Will abstract class have constructors?

Yes, To achieve constructor chaining abstract class will have constructor.

NOTE:→

① Every class is a subclass to object class so, even abstract class will be a subclass to object class. So, to achieve constructor chaining of object class there will be a abstract class

② Abstract class should always have subclass, so there should be constructor chaining b/w abstract class & its subclass. So, in abstract class constructor should be available.

Rule-1 :→ The subclass should override all the inherited abstract methods.

Rule-2 :→ If the subclass doesn't override all the inherited abstract methods then that subclass should be declared as abstract.

5) Abstract class is not 100% abstract. bcz we can develop any kind of methods in abstract class.

6) Through abstract class we can't achieve 100% abstraction.

or

Through abstract class we can achieve upto 100% abstraction.

I.Q :→

1) In abstract class can we develop an abstract method as static? No,

```
abstract static void test1()
```

Static methods we can't override but abstract methods should be override.

or

⇒ No, bcz static methods can't be overridden & abstract methods should be overridden.

2) Can we develop an abstract method as final?

No, bcz final methods can't be overridden and abstract methods should be overridden.

```
abstract final void test1() //Error.
```

3) Can we declare an abstract class class as final?

No, bcz final class can't be inherited & abstract class should be inherited.

4) Will abstract class have constructors?

Yes, To achieve constructor chaining abstract class will have constructor.

NOTE:→

① Every class is a subclass to object class so, even abstract class will be a subclass to object class. So, to achieve constructor chaining of object class there will be a abstract class

② Abstract class should always have subclass, so there should be constructor chaining b/w abstract class & its subclass. So, in abstract class constructor should be available.

23/10/15

Enumeration:→

If you want to group the fixed no. of constants then we will go for enumeration.

→ Through enumeration we can achieve uniformity while developing the application.

Ex:→

1) Days in a week
2) Months in a Year.
3) Keys of std i/p device or keyboard.

General Syntax to develop enum.

```
enum enumName
{

}
```

Ex:→

①
```
enum Months
{

}
```

②
```
enum Days
{

}.
```

Annotations.

→ If you want to give some information to the developer, compiler & run-time environment then they will use annotations.

→ Ex:→

Through one built-in annotation suppress warnings we can give information to the java Compiler to suppress the warnings. that is not to display the warnings.

→ Through override annotation we can give information to the developer about proper method overriding.

General Syntax:→

```
@interface name
{

}.
```

These @interface is annotation.

Ex:→

```
@interface Override
{

}
```

NOTE:→

→ Always annotations can be used on the top of variables, methods & classes.

→ Annotation should be starts with @ symbol.

Ex :->

1) @Suppressed("unused");
```
Class sample
{

}
```

2)
```
Class sample
{
@Suppressed("Unused");
  void engine()
  {
  }
}
```

```
class A
{
}

enum B
{
}

@interface C
{
}

interface D
{
}
```

Interfaces

1) Interface is another java type which is 100% abstract.
2) Through interface we can achieve multiple inheritance in java.
3) Through interface we can achieve Standardization.
4) Through interface we can achieve 100% abstraction.
5) We can't create object for interface.
6) Interface should always have a subclass & subclass should follow 2 rules or Contract.

1) Subclass should override all the inherited abstract methods.

2) If the subclass doesn't override all the abstract methods then the subclass should be declared as abstract.

3) If a class is inheriting from an interface then we should use the key word implements

→ In interface we can develop only abstract methods.

→ In interface all the methods will be public & abstract automatically.

→ Interface will not inherit from object class

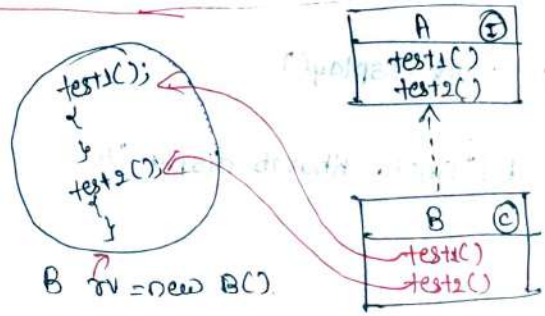
→ Interface will not have Constructors.

Ex:→

```
(1)  interface A
     {
        void test1();
        void test2();
     }

     class B extends implements A
     {
        public void test1()
        {
        S.O.P("test1() overridden in class B");
        }

        public void test2()
        {
        S.O.P("test2() overridden in class B");
        }
     }

     class Tester1
     {
        public static void main(String[] args)
        {
        S.O.P("The main starts");
        // A rv = new A();
        B rv = new B();
        rv.test1();
        rv.test2();
        S.O.P("The main ends");
        }
     }
```

O/P:→

```
The main starts
test1() overridden in class B
test2()   "       "     "
The main ends
```



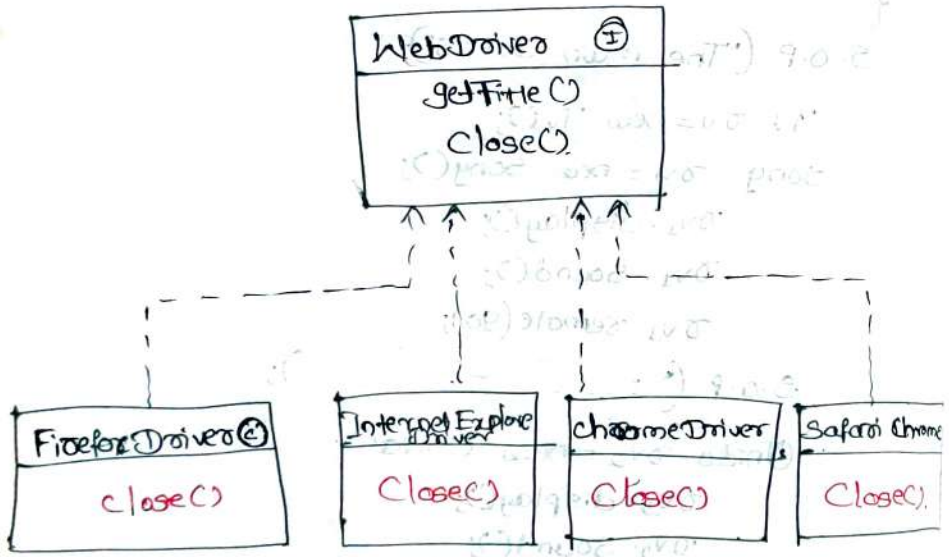
Standardization through inheritance.

```
interface Tv
{
    void display();
    void disound();
    void remote(int channel);
}

Class Sony implements Tv
{
  public void display()
  {
    S.O.P ("Sony LED display");
  }

  public void sound()
  {
    S.O.P("Sony DTS Sound s/m");
  }

  public void remote(int channel)
  {
    S.O.P ("Sony is in channel no."+channel)
  }
}

Class onida implements Tv
{
  public void display()
  {
    S.O.P ("Onida Kharab display");
  }

  public void sound()
  {
    S.O.P("Onida Kharab Sound s/m");
  }

  public void remote(int channel)
  {
    S.O.P("Onida is in channel no"+channel);
  }
}

Class Tester2
{
  Public Static void main (String[] args)
  {
    S.O.P ("The main starts");
    // Tv tv = new Tv();
    Sony tv1 = new Sony();
        tv1.display();
        tv1.sound();
        tv1.remote(90);
    S.O.P ("-----------");
    Onida tv2 = new Onida();
        tv2.display();
        tv2.sound();
        tv2.remote(100);
    S.O.P ("The main ends");
  }
}
```

O/P:→

The main starts.
Sony LED display
Sony DTS sound s/m.
Sony is in channel no 90.
Onida Khatab display.
Onida " Sound s/m.
Onida is in channel no 100.
The main ends.

In Selenium.




```
O/P:
Interface WebDriver()
{
    void getTitle();
    void close();
}
Class firefoxDriver implements WebDriver
{
  public void getTitle()
  {
   S.O.P("gets the title from firefox");
  }
  Public void close()
  {
  S.O.P("closes the firefox browser");
  }
}
Class chromedriver implements WebDriver
{
  public void getTitle()
  {
   S.O.P("gets the title from chrome");
  }
  public void close()
  {
   S.O.P("closes the chrome browser");
  }
}
```

```
Class Tester3
{
  public static void main(String[] args)
  {
    S.O.P ("The main starts");

    //WebDriver dv = new WebDriver();
    FirefoxDriver driver1 = new FirefoxDriver
      driver1.getTitle();
      driver1.Close();
      S.O.P (" ========== ");
      ChromeDriver driver2 = new ChromeDriv
                                    ();
        driver2.getTitle();
        driver2.Close();
      S.O.P("The main ends");
    }
  }
```

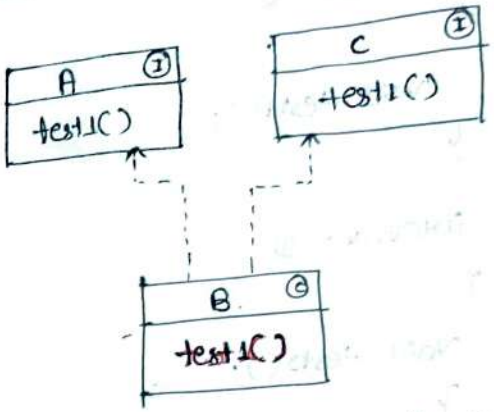
O/P:→

The main starts.
gets the title from firefox
Closes the firefox browser.

---

gets the title from chrome.
Closes the chrome browser.
The main ends.

Multiple inheritance using interface



1) Here Constructor chaining is not required to inherit blw the class and interfaces
2) Here no diamond path pbm bcz. interfaces doesn't inherited from Object class.
3) Interface only have abstract methods.

#

1) Interface will not have any Constructor.
2) If a class is inheriting from interface then no need of Constructor chaining.
3) Interface will not inherit from object class
4) In interface we can't have coding or the method does not have implementations or we can develop only abstract methods.

Ex:→

```
interface C
{
  void test1();
}

interface D
{
  void test2();
}

class E implements C,D
{
  public void test1()
  {
    S.O.P("test1() method overridden in
                          class E");
  }
  Public void test2()
  {
    S.O.P("test2() method overridden in
                          class E");
  }
}

class Testex4
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    E rv=new E();
    rv.test1();
    rv.test2();
    S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
test1() method overridden in classE
test2()   "        "      "
The main ends.
```

The class can inherit from another class & implements from multiple interfaces.

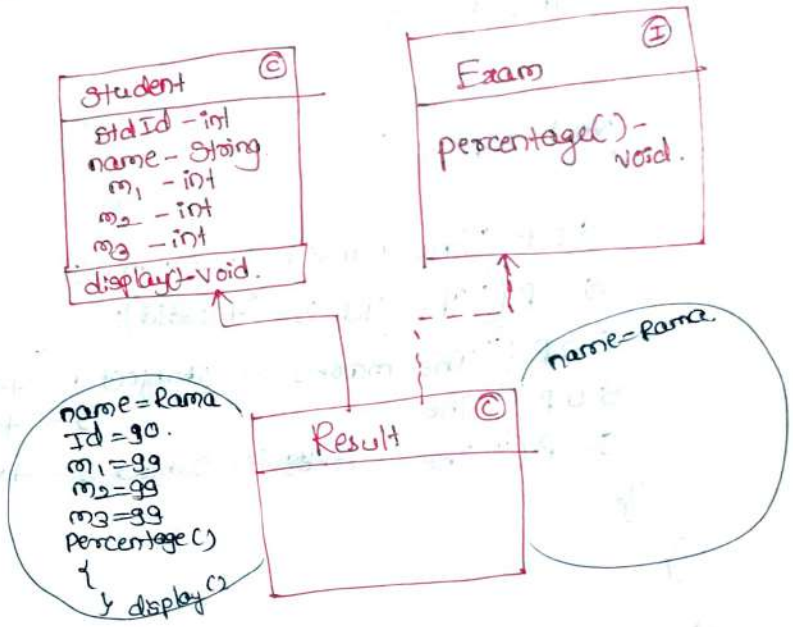
```
interface H
{
  void test2();
}

Class F
{
  void test1()
  {
    S.O.P("Running test of F Class");
  }
}

Class G extends F implements H
{
  public void test2()
  {
    S.O.P("test2() overridden in class G");
  }
}
```

```
class Testers
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    G rv=new G();
      rv.test1();
      rv.test2();
    S.O.P("The main ends");
  }
}
```

O/P:->

The main starts.
Running test1 of F class.
Test2() overridden in Class G.
The main ends.

I.Q




```
interface Exam
{
  void percentage();
}

Class Student
{
  int stdId;
  String stdname;
  int m1;
  int m2;
  int m3;
  Student(String stdname, int stdId, int m1,
                                  int m2, int m3)
  {
    this.stname = stdname;
    this.stdId = stdId;
```

```
        this.m1 = m1;
        this.m2 = m2;
        this.m3 = m3;
    }

    void display()
    {
     S.O.P("The name is="+stdname);
     S.O.P("The Id is= "+stdId);
     S.O.P("The marks in subject 1 ="+m1);
     S.O.P("The    "    "    "  2="+m2);
     S.O.P("The marks in subject 3 ="+m3);
    }
}

Class Result extends student implements Exam
{
  Result(String stdname, int stdId, int m1,
                    int m2, int m3)
  {
    super(stdId, stdname, stdId, m1, m2, m3);
  }

  public void percentage()
  {
     int total = m1+m2+m3;
     int percentage = (total*100)/300;
     S.O.P("Your percentage is"+
                          percentage);
  }
}
```

```
Class Testeg
{
 S.O.P(
 public static void main(String[] args)s
  {
   S.O.P("The main starts");
   Result ov1 = new Result("Rama", 90, 99, 99, 99);
       ov1.display();
       ov1.percentage();
   S.O.P("-------------");
   Result ov2 = new Result("Sita", 1, 7, 7, 7);
       ov2.display();
       ov2.percentage();
   S.O.P("The main ends");
  }
}
```

O/P:→

```
The main starts.
The name is Rama.
The Id is 90.
The marks in subject 1 = 99
"    "    "    "    2 = 99
"    "    "    "    3 = 99.
The Your percentage is 99.
---------------------------
The name is Sita.
The Id is 1.
The marks in subject 1 = 7
"    "    "    "    2 = 7
"    "    "    "    3 = 7
The main ends.
```

I.Q:- Can a interface inherit from another interface?

Yes, one interface can inherit from another interface. & to achieve this inheritance we should use extends keyword.

Ex:→ interface E extends F.

② Can we interface inherit from class?

No, we cannot interface inherit from class. because interface is 100% abstract & if it is inheriting from class then it might inherit Concrete methods.

① In interface we can develop variables & all variables will be public, static & final.

② Variables are cannot be abstract.

Ex:→
```
interface K
{
    int a=10; // public, static, final.
}

Class Tester8
{
  public static void main(String[] args)
  {
    S.o.P("The value of a is"+K.a);
    S.o.P("Done!!");
  }
}
```
O/P:→
The value of a is 10.
Done!!

I.Q:→

```
interface L
{
  int a;
}

Class Tester9
{
  public static void main(String[] args)
  {
    S.o.P("The value of a is"+L.a);
  }
}
```

O/P:→
C.T.E:

I.Q:→ What is marker interface?

→ The interface in which there will be no methods called as marker interface.

→ There are some builtin marker interfaces.

1) Cloneable
2) Serializable

Purpose:→

1) If you want to give some special information to the run time environment then we should use marker interfaces.

2) If you want to take some special permission from run time environment then we should use marker interface.

Ex:→

If a class is a subclass to Cloneable interface then only we can achieve cloning through Clone method. Otherwise cloning is not possible

```
① Class A implements Cloneable
  {
    Public void static void main (String[] args)
    {
      A ov1 = new A();
        try
        {
          ov1.close();
          S.o.p("Cloning Achieved");
        }
```

```
Catch (CloneNotSupportedException e)
    {
      S.o.p ("Cloning not possible");
    }
  }
}
```

O/P:→

Cloning Achieved.

25/11/15

What are the differances b/w abstract class & interface.

Abstract class	interface.
1) Abstract class is not 100% Abstract	1) 100% Abstract.
2) Here to develop an abstract method we should use a keyword abstract.	2) Here no need to declare the method with the keyword abstract.
3) Here methods will have programmer defined access levels.	3) Here methods or will be automatically public.
4) If a class is inheriting from abstract class then we should use extends keyword.	4) If a class is inheriting from an interface then we should use implements keyword.
5) Abstract class will have Constructors.	5) Interface can not have Constructors.

6) Abstract class will inherit from object class.

7) If a class is inheriting from an abstract class then Constructor chaining is required.

8) Through abstract class we can't achieve multiple inheritance.

9) Variables will not be automatically final in abstract class.

10) Here we can just declare a variable and initialize later.

11) Here we can develop static concrete methods.

12) Through abstract class we can achieve upto 100% abstraction.

6) Interface class will not inherit from object class.

7) If a class is inheriting from an interface then Constructor chaining is not required.

8) Through interface we can achieve multiple inheritance

9) Interface all the variables will be final, static and public.

10) In interface the variables should be initialized at the time of declaration itself.

11) Here we can never develop static concrete methods.

12) Through interface we can achieve 100% abstraction.

* Can we develop static methods in an interface?

No, bcz in interface we can develop only abstract methods & abstract methods can't be static.

Call by value

① test(int a)
{
}

test(90.5); ×
test("Hey"); ×
test(true); ×
test(90); ✓

int a = 90;
test(a); ✓

② test(double d)
{
}

~~double~~
test(true); ×
test("Hey"); ×
test(90); ×
test(90.5); ✓

double dd = 90.5;
test(dd); ✓

③ test(boolean b)
{
}

test(true); ✓

boolean bb = true; ✓
test(bb);

Call by reference.

① test(A a1)
{
}

② test(By b1)
{
}

A rv = new A();
test(rv);

By BV = new By();
test(rv);

There are 2-types of method invocations

① Call by value.
② Call by reference.

Call by reference :→

→ A method can receive an object as input. for such types of method we should supply object reference.

Ex:→

```
test(A a1)
{
}
```

The above method is receiving A class object as an i/p. So, to call that method we should supply the objects address of that A class. as follows.

A rv = new A();

NOTE:→

Call by value:→ Calling the method by suppling primitive data is called as call by value. type of method invocation.

① Ex:→ Refe Call by reference.

```
Class A
{
  int i;
  Static void test(A a1)
  {
    S.O.P("The value of i is" +a1.i);
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    A rv = new A();
    test(rv);
    S.O.P("The main ends");
  }
}
```

O/P:→

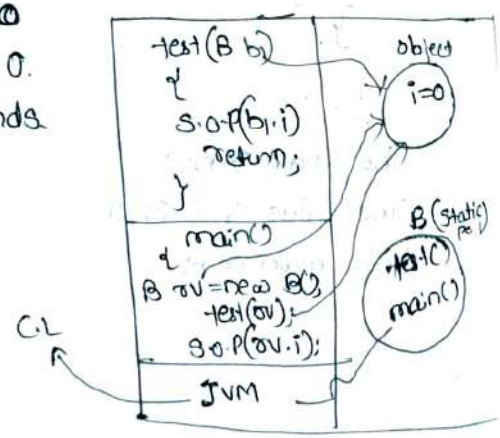
The main starts.
The value of i is 0.
The main ends.

② Class B

```
{
  int i;
  Static void test(B b1)
  {
    S.O.P("from test:: "+b1.i);
  }// return;

  Public Static void main(String[] args)
  {
    S.O.P("The main starts");
    B rv = new B();
      test(rv);
    S.O.P("from main:: "+rv.i);
    S.O.P("The main ends");
  }
}.
```

O/P:→

The main starts.
from test 10
from main 0.
The main ends



③ class C

```
{
  int i;
  Static void test(C c1)
  {
    S.O.P("from test:: "+c1.i);
    c1.i=90;
  }
  Public Static void main(String[] args)
  {
    S.O.P("The main starts");
    C rv=new C();
      test(rv);
    S.O.P("from main :: "+rv.i);
    S.O.P("The main ends.. ");
  }
}.
```

O/P:→

The main starts.
from test 0.
from main 90.
The main ends.

Swapping pgm. :→

```
Class D
{
  int i;
  int j;
  Static void test(D d1)
  {
     int temp = d1.i;
     d1.i = d1.j;
     d1.j = temp;
  }

  public static void main(String[] args)
  {
     S.O.P("The main starts");
     D rv = new D();
     rv.i = 10;
     rv.j = 20;
     S.O.P("======Before=======");
     S.O.P("rv.i =" + rv.i);
     S.O.P("rv.j =" + rv.j);
     test(rv);
     S.O.P("======After=======");
     S.O.P("rv.i= " + rv.i);
     S.O.P("rv.j = " + rv.j);
     S.O.P("The main ends");
  }
}
```

o/p:→

```
The main starts.
======Before=======
rv.i = 10
rv.j = 20.
======After=======
rv.i = 20
rv.j = 10.
The main ends.
```

26/11/15

2nd type of Call by reference.

```
Class E
{
 int i;
 Static void test(E e1)
 {
  S.O.P("The value of i is" +e1.i);
  //return;
 }
 public static void main(String[] args)
 {
  S.O.P("The main starts");
  test(new E());
  S.O.P("The main ends");
 }
}
```

E a1 = new E();
test(a1);

O/P:→

The main starts.
The value of i is 0.
The main ends.

```
test (A a1)
{
}
```

① A rv = new A();
test(rv);
xo1

② test(new A()).
xo1.

Returning objeClass object address.

Call by value.

```
int test()
{
 return 90.5;X
 return true;X
 return 90;✓
 int a=90;✓
 return a;
}
int res = test();✓
```

```
double test()
{
 return true;
 return 90.5;
 double dd=90.5;
 return dd;
}
double res=test();
```

```
boolean test()
{
 return true;
 boolean bb=false;
 return bb;
}
```

return address.

```
Static A test()
{
 A a1 = new A();
 return a1;
}
```

or

```
return new A();
```

```
A rv=test();
S.O.P("The value of i is" +rv.i);
```

① Class F

```
{
  int i;
  Static F test()
  {
    F f1 = new F();
    return f1;
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    F rv = test();
    S.O.P("The value of i is" + rv.i);
    S.O.P("The main ends");
  }
}.
```

O/P:→

The main starts.
The value of i is 0.
The main ends.

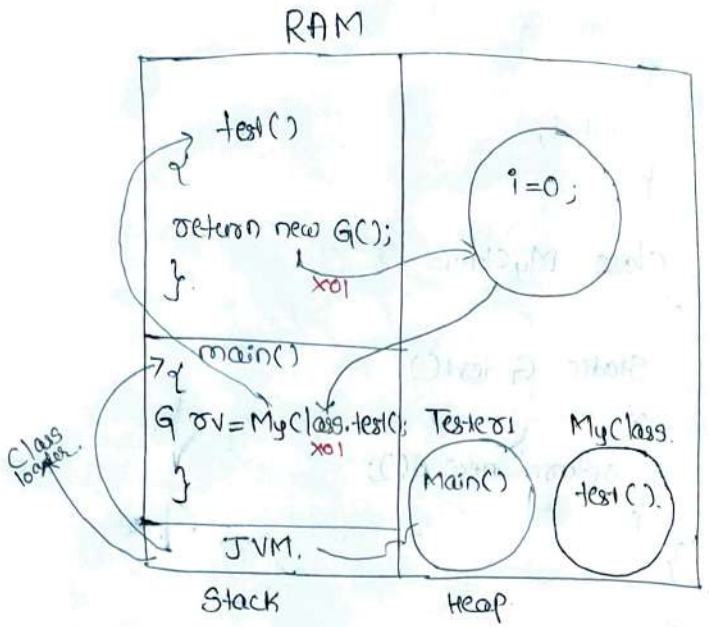
Static Method.

② Class G

```
{
  int i;
}

Class MyClass
{
  Static G test()
  {
    return new G();
  }
}

Class Tester1
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    G rv = MyClass.test();
    S.O.P("The value of i is" + rv.i);
    S.O.P("The main ends");
  }
}.
```

O/P:→

The main starts.
The value of i is 0
The main ends.



Non-Static method.

```
Class G
{
  int i;
}

Class MyClass
{
  G test()
   {
    return new G();
   }
}

class Test1
{
  public static void main (String[] args)
  {
    S.O.P ("The main starts");
    MyClass ref = new MyClass();
    G rv = ref.test();
    S.O.P ("The value of i is "+rv.i);
    S.O.P ("The main ends");
  }
}
```

O/P:→

The main starts.
The value of i is 0.
The main ends.

Can a method return an object?

Yes, a method can return an object. but the return type should be some Class name.

Type Casting :→

↓

Conversion.

~~2 types of~~

① Casting means Conversion.

② Type Casting means Converting data or object from one type into another type. is called as type Casting.

③ 2-types of type Casting.

① primitive Casting.

② Object "

① primitive Casting.

Casting a data from one primitive type to another primitive type is called as primitive Casting.

Homogeneous Stmts.

Those Statements in which there will be no type mismatch are called as homogeneous stmts

Ex:→

```
int a = 10;
```

Here we are storing integer data 10 inside an integer type variable a. So, there is no mismatch in the data type.

Heterogeneous stmts.

Those stmts in which there will be type mismatch are called as heterogeneous stmts

Ex:→

```
int a = 10.5;
```

Here we are trying to store double data 10.5 into integer variable a.

Here there is a mismatch in the data(10.5) & datatype (int a) so called as heterogeneous.

① `int a = (int) 10.5;` → This is primitive Casting.

② `double d = (double) 10;`

Ex:→

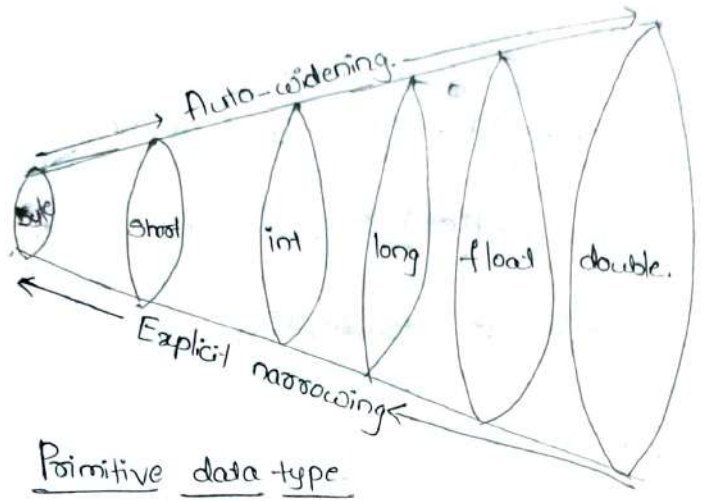
```
Class A
{
public static void main(String[] args)
{
  int a = (int) 10.9;
  S.o.p("The value of a is" +a);
  double d = (double) 10;
  S.o.p("The value of d is" +d);
}
}.
```

O/P :→

The value of a is 10.

The value of d is 10.0

3/2/15



Primitive data type

1-byte	byte	numbers
2- "	short	
4- "	int	
8- "	long	
4- "	float	
8- "	double.	

char
boolean.

1) Auto-widening

Ex:→ double d = 90;

2) Explicit narrowing

Ex:→ int a = (int)10.5;

There are 2 types of primitive Casting

1) widening
2) narrowing.

Widening→

Converting data from lower primitive type to higher primitive type is called as widening.

Auto-widening:→

Compiler performing the widening operation automatically at Compile time is called as auto-widening or implicit widening.

Explicit-widening:→

programmer performing the widening operation by developing the pgm is called as explicit widening.

Narrowing:→

Converting the data from higher primitive type to lower primitive type. is called as Narrowing.

→ There is no Concept of auto-narrowing. Pgmmer should perform Narrowing operation explicitly while developing the code. so, it is also called as explicit narrowing.

→ NOTE:→

Out of 8-primitive data type only numbers related data types will involved in widening

and narrowing.

Ex:→

```
Class B
{
public static void main(String[] args)
{
    S.O.P("The main starts");
    double d = 90; // Auto-widening.
    S.O.P("The value of d is"+d);
    int a=(int) 10.5; // Explicit-narrowing.
    S.O.P("The value of a is" +a);
    S.O.P("The main ends");
}
}.
```

O/P:→

```
The main starts.
The value of d is 90.0
"    "    "  a is 10.
The main ends.
```

Auto-widening.

```
Class C
{
  static void test(double d)
  {
    S.O.P("from test(double d) "+d);
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    test(90); // Auto widening.
    S.O.P("The main ends");
  }
}.
```

O/P:→

```
The main starts.
from test (double d) is 90.0
The main ends.
```

To Calculate Simple interest:→

```
Class D
{
  static void Si(double p, double t, double r)
  {
    double Si = (p*t*r)/100;
    S.O.P("The Simple interest is" + Si);
  }
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    Si(2.1, 2.2, 2.3);
    B.O.P("----------");
    Si(2, 2, 2.3);
  }
}, S.O.P("The main ends");
```

O/P:→

The main starts.

The Simple interest is 0.120.75

$ = = = = = = =

The Simple interest is

The main ends.

Method overloading.

```
Class E
{
  Static void test(int a)
  {
    S.O.P("from test (int a)");
  }

  Static void test(double d)
  {
    S.O.P ("from test (double d)");
  }

  public static void main(String[] args)
  {
    test(10);
    test ((double)20);
    S.O.P (" Done!!");
  }
}.
```

O/P:→

from test(int a)

from test (double d)

Done!!

```
int * int = int
int + int = int
int - int = int
int / int = int
```

```
double * int
  + double  = double.
  - ;
  / ;
```

```
Class F
{
  public static void main(String[] args)
  {
    double res=5/2;
    S.O.P("The result is "+res);
    int a=5;
    int b=2;
    double d=(double) a/b;
    S.O.P("The result is "+d);
  }
}.
```

O/P:→

The result is 2.0

The result is 2.5

```
public void percentage()
{
  int total=m1+m2+m3;
  double percent = ((double) total * 100)/300;
  S.O.P("The percentage is" +percent);
}
```

Imp

* Why primitive Casting is required?

(1) balance can be 500, 500.6
so,

```
double balance = 500.6
```

or

```
double balance = (double) 500;
```

↓ automated widening.

(2) In shopping malls,

bill is 500.6 → double
− 0.6
500 → int

so,

```
int bill = (int) 500.6; → Narrowing.
int bill = 500.
```

```
Class G
{
public static void main(String[] args)
 {
    S.O.P(Integer.MAX_VALUE);
    S.O.P(10);
    S.O.P(9880);
    S.O.P(9765467543L);
 }
}
```

O/P:→

```
2147483647
10
9880
9765467543
```

Here 2147483647 < 9765467543 so needs to use L.

Any integer no. is automatically int

" floating " " " double

```
Class I
{
public static void main(String[] args)
 {
    float f=10.2F;
    S.O.P("The value is "+f);
 }
}
```

O/P:→ The value is 10.2

→ To avoid too much of precision then use float

```
class J
{
public static void main(String[] args)
{
double res = (double)10/3;
S.O.P(res);
float f = (float)res;
S.O.P(f);
}
}
```

O/P:→

3.3333333333335

3.333333

65 90
A-Z

97 - 123
a-z

```
class K
{
public static void main(String[] args)
{
char ch1 = '9';
S.O.P(ch1);
// char ch2 = '90';
// S.O.P(ch2);
char ch3 = 90;
S.O.P(ch3);
int a = 'z';
S.O.P(a);
int b = '#';
S.O.P(b);
int c = 'a';
S.O.P(c);
S.O.P('A' + 'B');
}
}
```

O/P:→

9
Z
90
35
97
131.

T.C :→

Print a^2+b^2

```
class L
{
  Public static void main(String[] args)
    {
      char ch=253;
      S.O.P("a"+ch+"+b"+ch);
    }
}
```

O/P:→

a^2+b^2.

a^2 → 253 is the ascii value.

4/2/15

*** Object Casting

→ Casting the object from one class to another class type or interface type is called as object Casting.

→ To perform object Casting inheritance is mandatory

→ Homogeneous Struts.

Ex:→

① A a_1 = new A();

② B b_1 = new B();

Here we are storing A class object into A type reference variable. So, it is homogeneous Strut.

Heterogeneous Strut

① A a_1 = new B();

Here, we are storing B class object into A type reference variable which is a mismatch in types. So, it is called as heterogeneous Struts.

→ If you still wants to store B-class object into A type reference than we should perform Casting as shown below:

A a_1 = (A) new B();

→ Object Casting is also called as derived Casting or Class Casting.

There are 2-types of Object Casting.

① Up Casting.

② Down Casting.

Up Casting:→

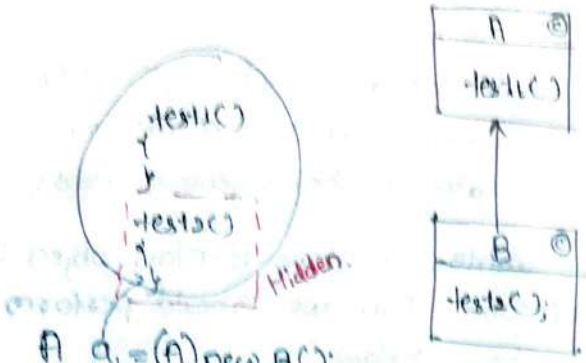
→ Casting or Converting the Subclass object into Super class type. Called as upCasting. Or

Storing the Subclass object into Superclass type reference. Called as upcasting operation.

*** Whenever we perform upCasting subclass features of that object will be hidden.

ie we can access only Super class features.

→ To achieve upcasting inheritance is required.



A a_1 = (A) new B();

a_1.test1();

a_1.test2(); ✗

Ex:→ Upcasting

```
Class A
{
  void test1()
  {
    S.O.P("Running test1() method");
  }
}

Class B extends A
{
  void test2()
  {
    S.O.P("Running test2() method");
  }
}

Class Tester1
{
  public static void main(String[] args)
  {
    S.O.P("The main starts");
    A a1 = (A) new B();
    a1.test1(); Upcasting
    // a1.test2(); Subclass features will be hidden.
    S.O.P("The main ends");
  }
}
```

o/p:→

```
The main starts
Running test1() method
The main ends.
```

Upcasting Automatic.

```
Class C
{
  void test1()
  {
    S.O.P("Running test1 method");
  }
}
Class D extends C
{
  void test2()
  {
    S.O.P("Running test2 method");
  }
}
Class Tester2
{
  public static void main(String[] args)
  {
```

```
        S.O.P("Main starts");
        C c1=new D(); → Upcasting is automatic
        c1.test1();
    // c1.test2();
        S.O.P("The main ends");
    }
}.
```

O/P:→

The main starts.
Running test1 method.
The main ends.

Auto Upcasting:→

Compiler performing the upcasting operation automatically at Compile time. this is called as auto upcasting or implicit upcasting.

Ex:→

```
Class Trainer
{
    void training()
    {
        S.O.P("Java Training");
    }
}

Class Hemanth extends Trainer
{
    void drunkard()
    {
        S.O.P("Drinking");
    }
}

Class Tester3
{
public static void main(String[] args)
    {
        S.O.P("The main starts");
        Trainer t1= new Hemanth();
            t1.trainer();
        // t1.drunkard();
        S.O.P("The main ends");
    }
}.
```

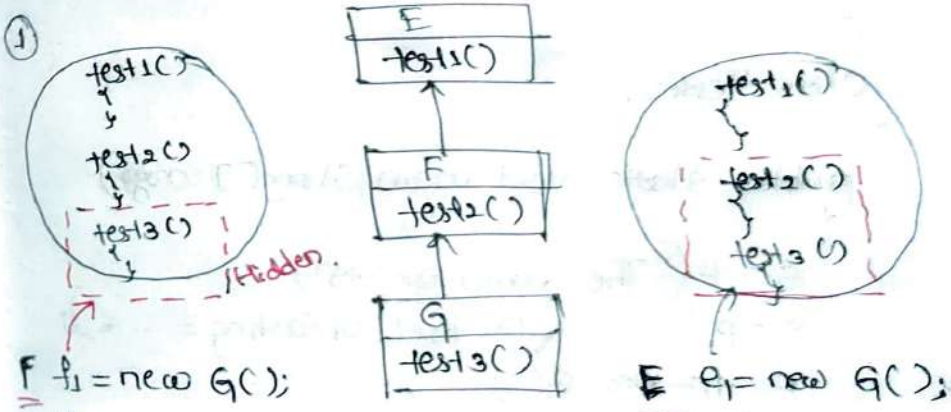
O/P:→

The main starts
Java training.
The main ends.

Upcasting in Case of multilevel inheritance.

In Case of multilevel inheritance whenever we perform upcasting we can access Superclass features and its Superclass features.

(1)



F f1 = new G();
f1.test1();
f1.test2();

E e1= new G();
e1.test1();

Ex:→

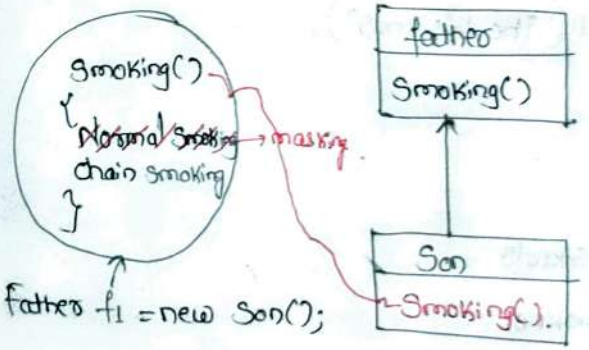
```
Class E
{
  Void test1()
  {
    S.o.p("Running test1 of E Class");
  }
}

Class F extends E
{
  Void test2()
  {
    S.o.p("Running test2 of F Class");
  }
}

Class G extends F
{
  Void test3()
  {
    S.o.p("Running test3 of G Class");
  }
}

Class Testerdy
{
  public static void main(String[] args)
  {
    S.o.p("The main starts");
    S.o.p("=== 1st level upcasting ===");
    F f1 = new G();
    f1.test1();
    f1.test2();
    // f1.test3();
    S.o.p("=== 2nd level upcasting ===");
    E e1 = new G();
    e1.test1();
    // e1.test2();
    // e1.test3();
    S.o.p("Done!!");
  }
}
```

O/P:→

```
=== 1st level upcasting ===
Running test1 of E Class.
"   test2 " F
=== 2nd level upcasting ===
Running test1 of E Class
Done!!
```

Imp Upcasting incase of method Overriding

Incase of upcasting we can access any overridden methods but we will get the latest subclass implementation.



```
class father
{
  void Smoking()
  {
    S.O.P("Normal Smokes");
  }
}

class Testers Son extends Father.
{
  void Smoking()
  {
    S.O.P("Chain Smokes");
  }
}

Class Testers
{
  public static void main(String[] args)
  {
    S.O.P("The life starts");
    Father f1 = new Son();
    f1.Smoking();
    S.O.P("The life ends");
  }
}
```

O/P:→

```
The life starts
Chain Smokers
The life ends.
```

1) Gov. approved identity proof.
2) Mail Copy
3) Spiders identity proof
4) Rough sheets.
5) Pen/Pencil/Erasers.

Shortlisted students needs to carry

1) CV
2) photo
3) Spiders identity proof.

- facebook
- Blogger.
- ISTQB.

UpCasting

1) How java is Platform independent?

2) Explain methods & its usage?

3) Can we develop a method without return type?
   → No.

4) Can we run pgm without main()?
   → No

5) Can we Compile pgm without main()?
   → yes

JM
* 6) What is Constructor?

* 7) What is " " & its types with pgm.

* 8) Why multiple inheritance is not possible through Classes?

9) What is method overloading & method overriding?