A free booklet
To all Java
Students

R.Nageswara Rao.
Composed and Edited By Renu kumar.

# Complete Notes On Core Java

First Edition

iNetSolv Solutions
Ph: 040-2373 0482, 2373 0483
Swathi Manor, Ameer Pet
Hyderabad-5000 016.

# Index - i

## (Core Java JSE)

# Index - ii

# Java Introduction

Sun Micro Systems incorporation's has divided java into 3 parts.

i. Java SE ( **Standard Edition** )
ii. Java EE ( **Enterprise Edition** )
iii. Java ME ( **Micro Edition or Mobile Edition** )

## Java SE (Standard Edition):

JSE deals with developing standard applications for a network and also exploring java Library.

In personal system users develops a program and compile then run it, this is standalone. This system will not get connected to any network (individual).

Application for networks means writing program for network may be LAN, WAN (System in Network)

## Java Library:

This is an in-built feature; means need not to re-develop for present program. It consists different new features, interfaces, classes etc.

$$\begin{tikzcd}
 & \text{Core Java} \\
\text{Java SE} \arrow[ur, dash] \arrow[dr, dash] & \\
 & \text{Advanced Java}
\end{tikzcd}$$

Java SE is First part of Java.

What is Core Java?

**Reply:** See Above (Java SE).

## Java EE (Enterprise Edition):

JEE deals with developing business solutions for internet and any network. To do Business on internet through a website we need JAVA EE.

## Java ME (Micro Edition or Mobile Edition):

JME deals with developing Embedded System and Wireless Applications.

$$\text{System} \longrightarrow \text{Hardware} + \text{Software}$$

Embedded means inserting (installing) a developed Software into a Hardware.

Big IC ->made-up of semiconductors material, ex: silicon chips or Germanium.

- The program written by the programmers will be stored in those chips. This type will be done by using **Java ME.**

- Wireless Markup language (WML) is used in wireless programming.
- Now a day Software Using Wireless Devices is developed by **Java ME.**
- Electronic device works with electricity and use electronic circuits components.

$$\left.\begin{array}{ll}\textbf{Voltage} & \textbf{- 1}\\ \textbf{Non Voltage} & \textbf{- 0}\end{array}\right\}\text{Bits Digits}$$

- Computer is an Electronic Device; it can understand only two things. The Presence of voltage is represented by 1.obsence of Voltage is represented by 0 these 0 and 1 are called Bits (Binary Digits).

- Microprocessor will Understand Binary Digits

## Machine Code:

Representing instruction and data using binary digit is called Machine Language or Machine Code. Computers can understand only machine codes.

## Translator:

Translator is a Program that converts a computer program into machine code.
There are three types of Translators.

i. **Interpreters**
ii. **Compiler**
iii. **Assembler**

## Interpreters:

- It converts line by line and only one line at a time.
- This type of translator is very slow when converting the code.

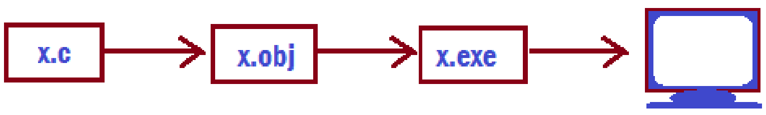
## Compiler:

- It converts all lines into machine code and gives to microprocessor.
- This type of translator is fast when process the code.

## Assemblers:

- It converts assembly language program into machine code.

**Note:** Compiler is faster than the interpreter.
Extension name represents in which language user developed program.
Dot obj (.obj) is the extension.



What is object Code?

Reply: Object code is nothing but machine code equivalent of source code.

- **TC**/lib is the directory.
- (**.exe**) Dot Exe Code is Full-fledged machine code. It consists of the code of header file.
- (**.obj**) Dot file will simply converts the user code into machine code but not gives the actual code of header file.

What is difference between object code and executable code?

Reply: Executable code = object + Header File code

```
#include<stdio.h>                         1111 0000 stdio.h  ---->  .exe code
Main()                                    101010()                  execute entire code
{                                         {
     c=a+b;                                    1100 1100 1011
                                               1010 0000 0110
     Printf(c);                                0000 1100 (10)
}                                         }
```

Why microprocessor will not execute (.obj) file?

Reply: Because it will not contain entire code of Header File.

Microprocessors cannot be understand all the instructions of the another microprocessor. So, when the processors are different in different system, same **.exe** code will not be executed by all different systems. But users' code can be run on different system of different microprocessors.

Java is an OPENSOURCE from SUNMICRO SYSTEMS put the source code in internet. Microsoft download that and developed Microsoft Dot Net (**.Net**).

Every OS will store data and instruction in different format. So **.exe** file will not be run on different OS.

C, C++ programs are system dependent/plat form dependent. Because those programmers are executable only on that computer system where they are developed.

Why C, C++ are not suitable for internet?

Reply: Because C, C++ are platform Dependent they are not suitable for internet.

What are Byte Code instructions?

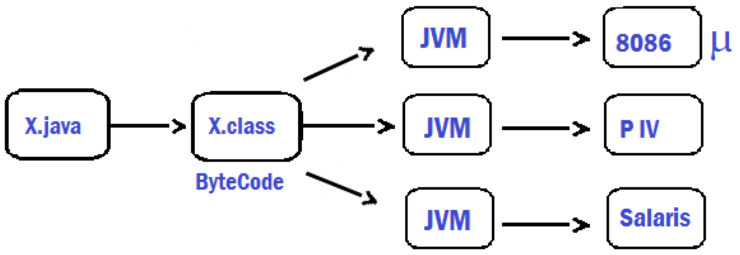
Reply: ***Byte code*** is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine* (***JVM***).

There are 200 instructions are create in java. The size of each instruction is 1- byte.

∴ These are called ***Byte code*** instructions.

Byte code instructions are a fixed set of instructions representing all operations. These instructions are developed by "*java Soft people*".
The size of each instructions is 1 – byte.

JVM is a program or it is software. It does not create **.exe** file



- Byte code is a system independent.it is a platform independent.
- JVM is system dependent, platform dependent, it is written in C Language.

iiQ **What is difference between .exe file and .Class file?**

Reply: **.exe** file contains machine code to understandable to the process.
**.class** file contains byte code to understandable to the **JVM**
**.exe** file is System dependent, platform dependent.
**.class** file is system independent, platform independent.

Java programs are executable on any computer system.it means java is system independent (platform independent).so java is highly suitable for internet.

**Security problems of Data on Internet:**

i. **Eaves Dropping**
ii. **Tampering**
iii. **Impersonation**
iv. **Virus**
v. **Digital Signature**

| | |
|---|---|
| **Eaves Dropping:** | Reading other data illegally. |
| **Tampering:** | Reading and modifying other data illegally. |
| **Impersonation:** | A person acting as another person or interact. |
| **Virus:** | It is a program that damages data, Software & Hardware of a PC |
| | **Exe:** images, audio, Videos etc. |
| **Digital Signature:** | It is a data file, it contains personal Id Information |

- By using java on internet we can create high level secured systems, By eliminating, "Eaves Dropping"
- This is the reason java is suitable for internet.

Features of Java:

1. Simple to learn
2. Object Oriented
3. Distributed
4. Robust - (Strong )
5. Secured
6. Architecture Neutral
7. Portable
8. Interpreted
9. High Performance
10. Multithreaded
11. Dynamic

## Simple to learn:

Java is a simple programming language to learning & practicing java is easy because of the resemblance with C & C++, also complex topic of c and C++ (like Pointers, Hieratical Inheritance & Operator Overloading) are eliminated in java.

iiQ Why Pointer are Eliminated in java?

Reply: Because

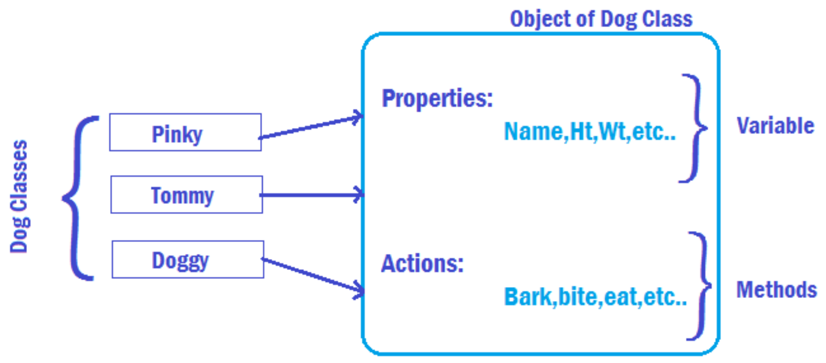
- Pointers leads to confusion for a programmer
- Pointer easily crash the program
- Pointers are security threat for data because of these problems pointer are eliminated.

## Object Oriented:

Unlike C++, java is a purely Objected Oriented Programming language, java Programs use **Objects** and **Classes**.

### Object:

- An object is anything that exist physical form or physically exist in the real world.
- An Object contains Properties and can perform Actions.
- **Properties** are represented by **variables.**
- **Actions** are represented by **Methods.**
- An Object contains variable and methods.



**Class:**

- A class is a group name; that specifies properties and actions of an **object.**
- A class also has properties and actions.
- A class also contains variable and methods.

**Note:**

Class does not exist physically, but an object can exists physically.

- An object is an **Instance** of a class ***i.e.*** A Physical Form.
- Class is a plan or model to create objects, a class can exist without an Object.
- An object cannot exist without a class.

## Distributed:

Information is distributed on various computers on a network, using java we can write program which capture information and distributes it to clients.

## Robust - (Strong):

Java programs will not crash easily because of its **exception handling** and its **memory management** feature.

iiQ

**What is Memory Management feature in java?**

Reply: In java there is a JVM (Java Virtual Machine)

- JVM's class loader subsystem memory De-allocation.
- JVM's garbage collection.

## Secured:

Java enables the construction of **virus free** & **Temper Free** system

## Architecture Neutral:

Java's byte code is not machine dependent; it can be run on any machine with any processors and with any OS.

## Portable:

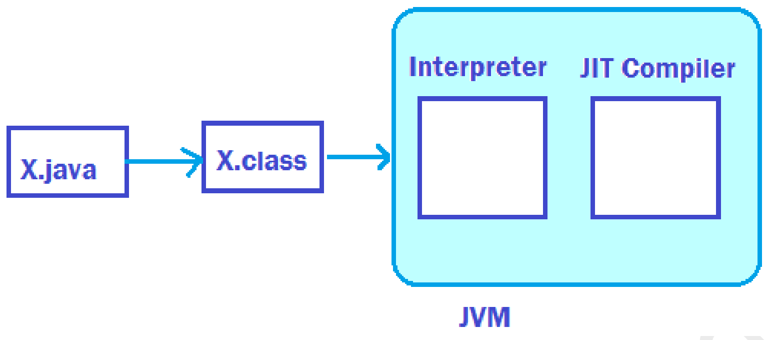
Java programs give the same result on any machine; **i.e.** its byte code will help the programmer to get exact output for a java program which is executed on any machine.

## Interpreted:

Java program are compiled to generate the byte code, this byte code can be interpreted by the interpreter in JVM.

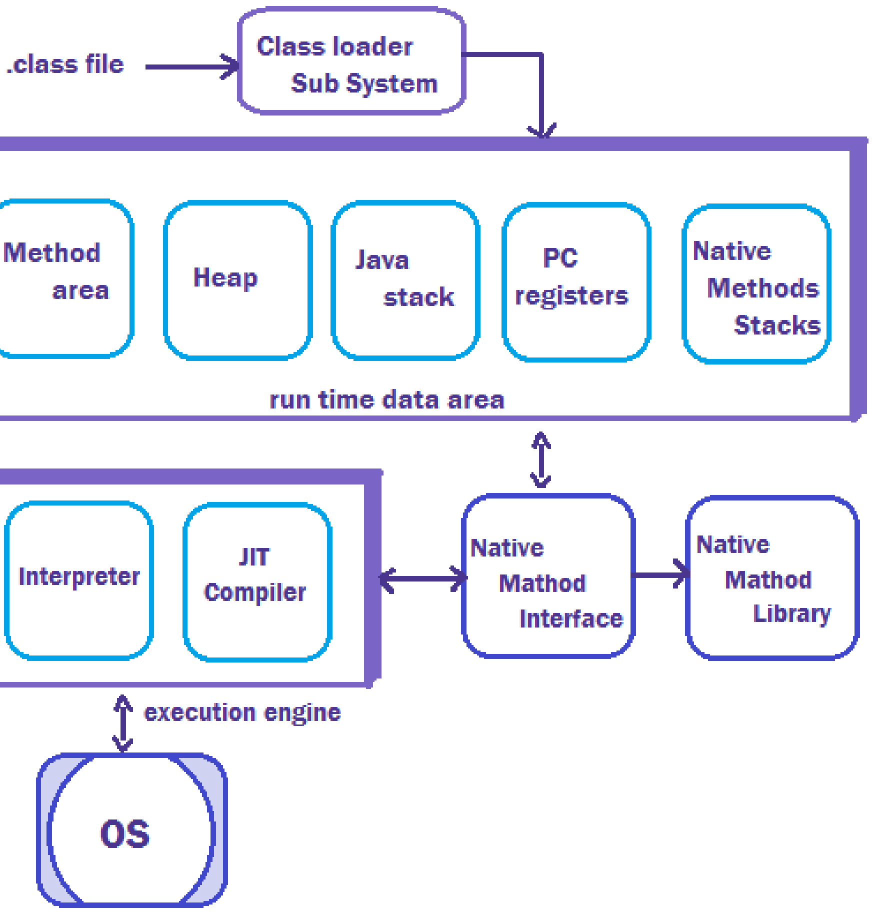
## High Performance:

Along with interpreter, there is JIT (Just in Time) compiler in JVM which enhance the speed of execution.



Java Virtual Machine Architecture (or)

JVM Block Diagram



JVM is a program which is written by the Java Soft People. In JVM, class loader subsystem does the following task (work):

- It loads the dot class (.class) file from the hard disk into the memory of the computer.
- It verifies the byte code instructions.
- Then it will allot the memory required by the java program.

The memory allots by the class loader subsystem is divided into Five parts, called **Run Time Data Areas.** They are...

1. Method Area
2. Heap Memory
3. Java Stacks
4. PC Registers
5. Native Method Stacks

**Method Area:**

Class code, method code and static variable are stored in method area.

**Heap Memory:**

Object are created in the heap memory, here we can create any no.of objects. Heap memory is a huge memory block in JVM.

**Java Stacks:**

These are the memory areas where java methods are executed.java stacks are divided into frames and on each frame a separate method is executed.

**PC registers** (Program Counters)**:**

These registers store the memory address of the next machine code instructions to be executed by the microprocessors.

**Native Method Area:**

These are the memory areas where native methods (C, C++ functions and programs) are executed.

**Note:**

Native method interface is a program that copies native method libraries Like C, C++ Header files into the JVM

**Multithreaded:**

We can create several processors in java, called **threads** this is an essential feature to design server side programs.

**Dynamic:**

We can develop programs in java which dynamically interact with the user on internet

Ex: Applets.

**Comments:**

A comment represents description about the features of programs. These are non-executable statements by the JVM. The advantage of writing comments is to increase readability and understandability of executable statements in the program by the programmer.

There are three types of comments...

1. Single Line comments

   Ex: // it represents the Single Line Comments.

2. Multi Line comments

   Ex:
```
/* it represents
      The Multi Line
Comments
*/
```

3. Java Documentation comments

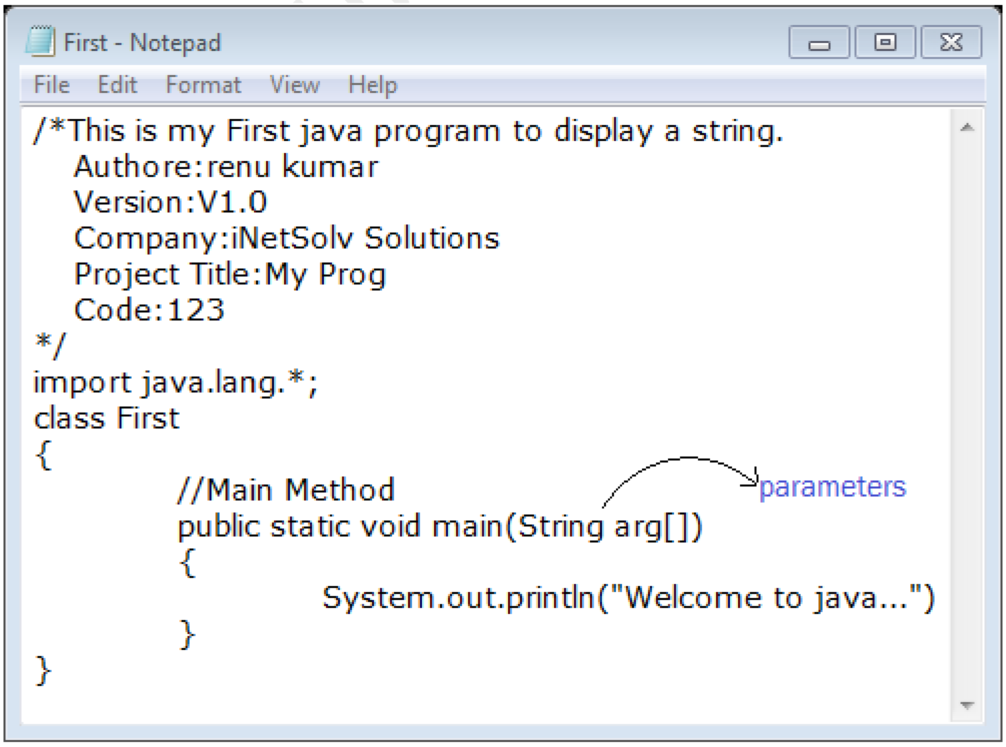
   Ex:
```
/** it represents  The Java Documentation Comments.
        Using these comments we can create API Documentation from a
Java programs
*/
```

iiQ

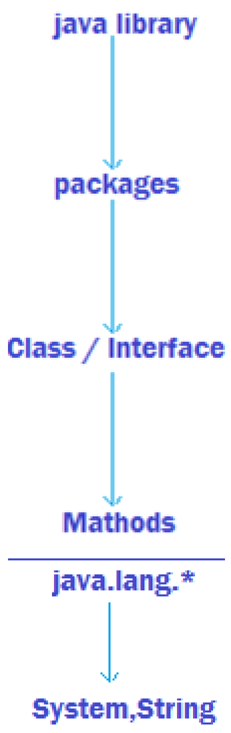
**What is API Document?**

Reply: **API (Application Programming Interface) documentation is a file that contains discretion of all the features of software's, a product, or a Technology.**
**API Document is created by using the java doc compiler.**

**Ex.prg: 1**



```
/*This is my First java program to display a string.
   Authore:renu kumar
   Version:V1.0
   Company:iNetSolv Solutions
   Project Title:My Prog
   Code:123
*/
import java.lang.*;
class First
{
          //Main Method
          public static void main(String arg[])
          {
                    System.out.println("Welcome to java...")
          }
}
```



Sava the java program as " First.java" (mention the extension as .java)

- Main Method is entry point to JVM to start execution of a java program.
- A **parameter** is a variable to receive data from outside, so **String arg []** is a main method parameters. JVM wants a main method argument as **Sstring** from only.
- **Arguments:** The data or values passed to any method.

**To call a method in java there are two ways:**

1. **Create an object to the class**

   **Class_name Obj = new class_ name ();**

2. **Then call method using object**

   **Obj.method ();**

## Static Methods:

A static method is a method that can be called without using any object.

These methods are called using **class_name.method_name.**

**Public:** It is available to outside programs or JVM.

**About** System.out.println ("………………………… /");

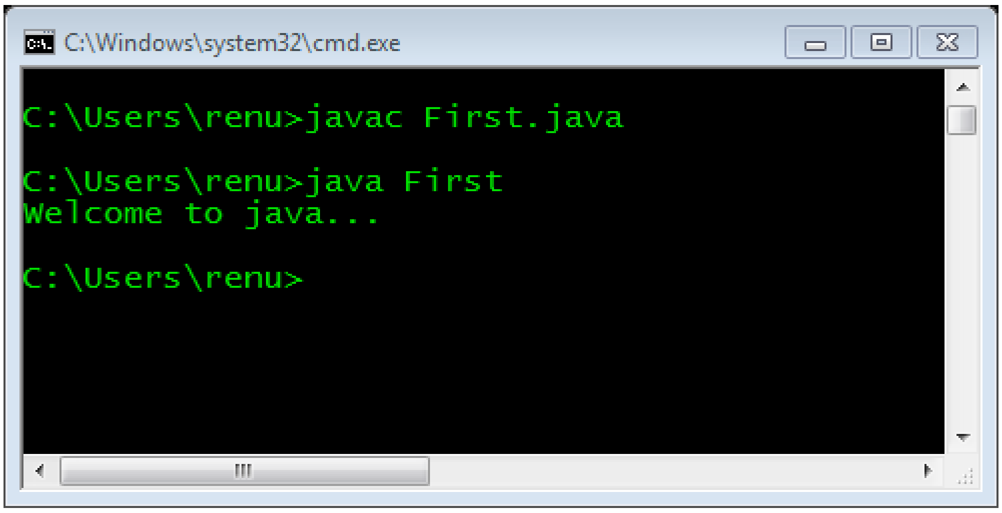
**printStream Obj = new printStream (System. out);**

**Obj.print ("………………………………/");**

- System. out returns printStream object default that object represents standard output device (Monitor). Here out is a Static Variable in System class.
- System.in returns the inputStream object by default it represents standard input device (Key Board). Here in is a Static Variable in System class.

## To Execute the First.java Program:

C:\User\renu\>javac First.java
C:\ User\renu\>java First



```
C:\Users\renu>javac First.java

C:\Users\renu>java First
Welcome to java...

C:\Users\renu>
```

iiQ What is JRE & JDK?

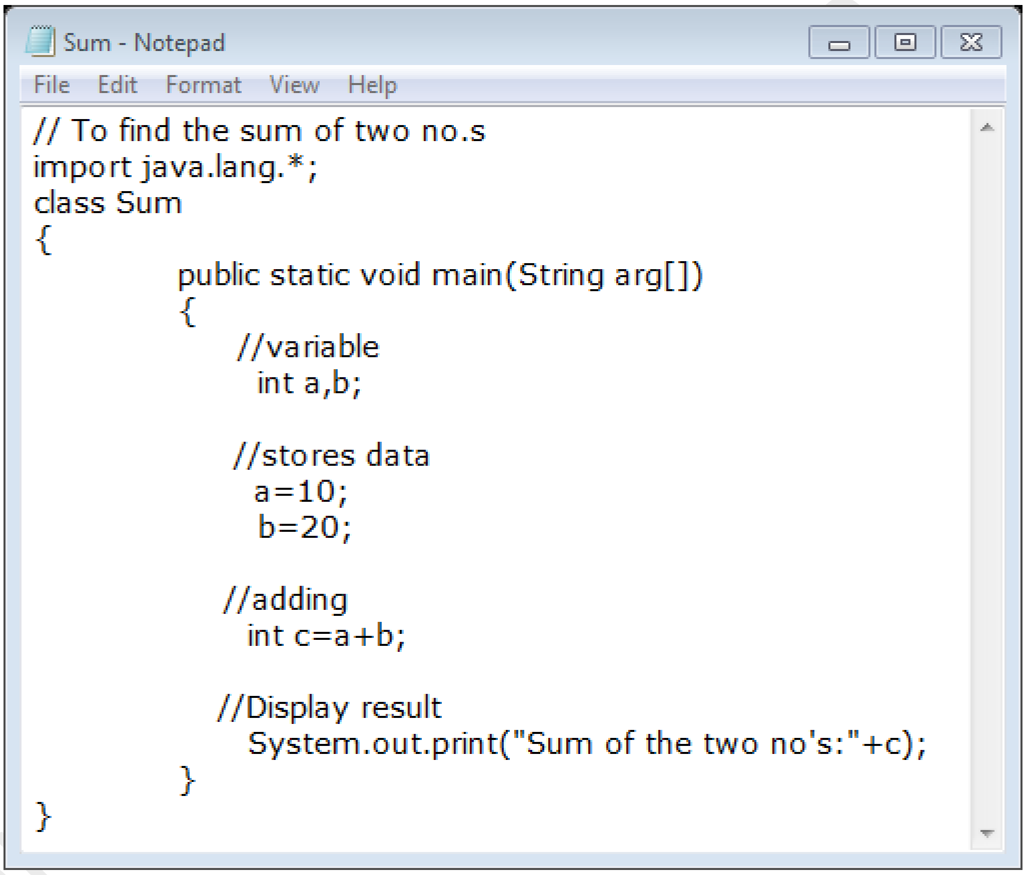
**Reply:** JRE – Java Runtime Environment.

JDK – Java Developer Kit

- JRE = JVM + java library.
- JDK = JRE + JVM +javac

Ex.prg: 2 To find the Sum of two no's.

- main (int arg[]) this main method does not treated as java main method by JVM, because JVM wants a main method arguments as **String** form only.



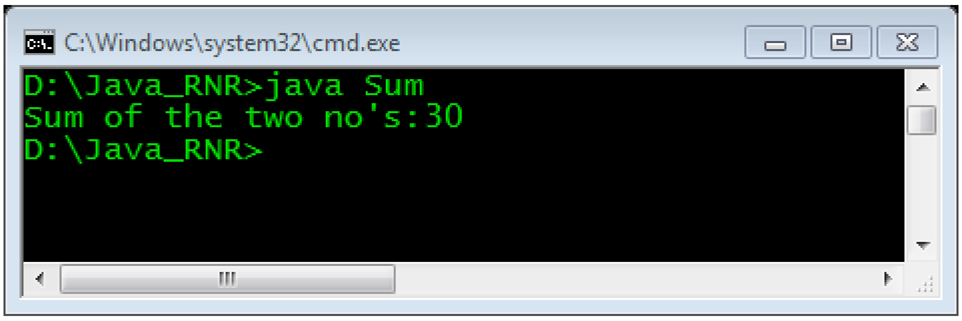
```
// To find the sum of two no.s
import java.lang.*;
class Sum
{
            public static void main(String arg[])
            {
                //variable
                 int a,b;

                //stores data
                 a=10;
                 b=20;

               //adding
                 int c=a+b;

              //Display result
                 System.out.print("Sum of the two no's:"+c);
            }
}
```

**Print () -:** Display the output and then keeps the cursor in the same line.

**Pritln () -:** Display the output and then throws the cursor in the next line.

**These both the methods are belong to the Printstream class.**

**Output:**



```
D:\Java_RNR>java Sum
Sum of the two no's:30
D:\Java_RNR>
```

**Naming conventions:** Java is a case sensitive language.

The rules to be followed by the programmers by writing class name, method name, and variable declaration etc. in the java programs.

There are six rules in the java naming conventions…

1. Package Names in java are written in all small letters.
   Ex: java.awt;
   java.io;
   javax.swing;
2. Class Names & Interface Names are start with capital letters.
   Ex: String
   DataInputStream
   ActionListener
3. Method Names start with a small letters, then each word start with capital letters.
   Ex: println ()
   readLine ()
   getNumberInstance ()
4. Variable also fallows the method Naming convention rule (i.e. Method name rules)
   Ex: empName
   emp_Net_Sal
   cityName
5. Constant variable name should be written using all capital letters
   Ex: PI
   MAX_VALUE
   Font.BOLD
6. All Key words should be written in all small latters
   Ex: public
   void
   import

**Some Important packages of Core Java:**

**Java. lang.*;**

This package got primary classes and interfaces essential for java program. It contains or consists of wrapper classes, String, Threads etc.

- **Wrapper Classes:**
  These are the classes useful to convert ordinary classes to objects
- **Threads:**
  Executing the classes is called a thread.

**Java.util.*;**

This package contains useful classes and interfaces like Stacks, LinkedList, HashTable, Arrays etc.

## Java.io.*; input - output

This package handles files and input – output related tasks or operations. We can read or write the java package using this package.

## Java.awt.*;

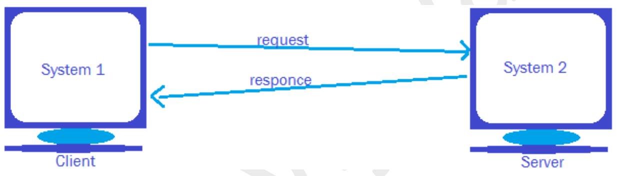
This packages helps to develop GUI (Graphical User Interfaces) applications. It consists of important sub-packages like **java.awt.event.*;**

## Javax.swing.*;

Javax is extends class to **java.awt.*;**
This package helps to develop GUI (Graphical User Interfaces) applications.

## Javax.net.*;

A **client** is a machine that receives services from a network. A **Server** is a machine that provides services to other computers in a network.



Using **java.net.*;**
Package we can write client – server programming can be done using these packages.

## Javax.applets.*;

Applets are programs which come from a server into a client and get executed on the client machine.

- An applet is dynamically interactive programs
- Applets are client side programs

## Javax.sql.*;

This package helps to connect to database like **oracle,** and utilizes them in java programs.

- This package supports **Structure Query Language** (SQL**).**

≈ ≈ ≈

# Data type & Literals

A **Data Type** represents the type of data stored into a variable or memory.

There are five data type as follows…

1. Integer Data Types
2. Float Data Types
3. Character Data Types
4. String Data Types
5. Boolean Data Types

## Integer Data Types:

These data types represent numbers without decimal point like 10, -20, 9600, 0 etc.

| Data Types | Memory Size | Min & Max Values |
|---|---|---|
| **Byte** | 1 byte | -128 to +127 |
| **Short** | 2 byte | -32768 to +32767 |
| **Int** | 4 byte | -2147483648 to +2147483647 |
| **Long** | 8 byte | -9223372036854775808 to +9223372036854775807 |
| | | $9.2x10^{18}$ to $9.2x10^{18}$ |

Ex: byet rno = 101;
long sal = 340000;
int x=20;

A **Literal** represents a value directly stored into a variable **i.e. roll_no=101.**

## Float Data Type: Single Point

These data types represent numbers with decimal point like 10.0, -20.12, 96.00, 0.01 etc.

| Data Types | Memory Size | Min & Max Values |
|---|---|---|
| **float** – (Single precision) | 4 byte | -1.4e-47 to +1.4e48 |
| **double** – (double precision) | 8 byte | -4.9e-324 to +1.8e308 |

Ex: float PI = 3.142; - (default float is double precision number's)
float PI = 3.142f; - (Now it treated as floating value)
double dist=1.98e8;

$1.2X10^{-2.5}$ ➔ $?x10^{?}$ = e/E (exponentiation) ➔ 1.2e-2.5

## Character Data Type:

These data types represent a single character like A, a, 1,*, %, “, -, ‘, @ etc.

| Data Types | Memory Size | Min & Max Values |
|---|---|---|
| Char | 2 byte | 0 to 65535 |

Ex: char ch = ‘X’;
 char at = ‘@’;

iiQ **What is Unicode System?**

Reply: **A 16-bit character set standard, designed to include characters appearing in most languages including Chinese, Japanese, etc.**

- Unicode is a standard to include alphabets from all human languages into java.
- Unicode system uses 2 bytes of memory to represents a single character.

## String Data Type:

A String represents group of characters like “renu kumar”, “AP 16X 5124”, “iNet” etc.

Ex: String name = “iNet Solv”;
 string str = new string (“Vijayawada”);

A string is a **data type** as well as **a class** because every class is a **user defined data type.**

## Boolean Data Type:

This data type represents two values, either **true** or **false** or binary values like 1 or 0.

Ex: boolean response = true;
 boolean request = false;

*Useful:*

**ASCII** -: American Standard Code for Information Interchange

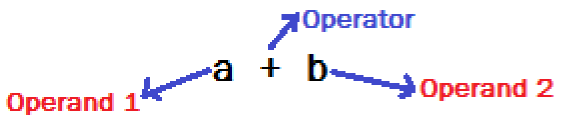
A code for information exchange between computers made by different companies;
A string of **7** binary digits represents each character; used in most microcomputers

≈ ≈ ≈

# Operators

An operator is a symbol that can perform an operation on operands (variable);
Operators are convenient to program (or) to do convenient to do programming.

- If an operator acts on a single variable (Operand) then it is called as **Unary operator.**
- If an operator acts on a two variables (Operands) then it is called as **Binary operator.**
- If an operator acts on a three variables (Operands) then it is called as **ternary operator.**



There are twelve Operators as follows…

1. Arithmetic Operators
2. Unary Operators
3. Incremental Operators
4. Decremented Operators
5. Assignment Operators
6. Relational Operators
7. Logical Operators
8. Boolean Operators
9. Bitwise Operators
10. Conditional Operators
11. Membership or (.)Dot Operators
12. Instanceof Operators
13. New Operators

## Arithmetic Operators:

These operators performs basic arithmetic operations etc.

| Operator | Operators Meaning | Usage |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a – b |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulus (remainder of Div.) | a % b |

Ex: int a=2, b=5;
a +b = 7;
a%b=2; a/b=0

## Unary Operators:

This operator negates the value of variable.

The negate (**~**) convert the '+Ve' value to '–Ve' value and vice versa.

There are five data type as follows...

1. Unary Incremental (++)
2. Decrement (––)
3. Unary Minus (–)

### Incremental (++):

This operator is increases the value of variable by 1.

```
Ex:     int i=2;
        ++i = 3;
        I++ = 4;        i=i+1➔4
```

- Writing ++ before the variable is called **Pre – Increment.**
- Writing ++ after the variable is called **Post – Increment.**
  - In **Pre – Increment.**

    Increment is done first, and then all other operations are done later.
  - In **Post – Increment.**

    All other operations are done first; and then increment is done at the end.

```
Ex:     int x=1;                    int x=1;
        System.out.println (x);     System.out.println (x);
        System.out.println (++x);   System.out.println (x++);
        System.out.println (x);     System.out.println (x);
O/P:    1, 2, 1, 1                  1, 1, 2, 1
```

### Decremented (––):

This operator is decreases the value of variable by 1.

```
Ex:     int i=3;
        ––i = 2;
        I–– = 1;        i=i–1➔0
```

- Writing –– before the variable is called **Pre – Decrement.**
- Writing ++ after the variable is called **Post – Decrement.**
  - In **Pre – Decrement.**

    Decrement is done first, and then all other operations are done later.
  - In **Post – Decrement.**

    All other operations are done first; and then Decrement is done at the end.

```
Ex:     int x=3;                    int x=3;
        System.out.println (x);     System.out.println (x);
        System.out.println (––x);   System.out.println (x––);
        System.out.println (x);     System.out.println (x);
O/P:    3, 2, 2                     3, 3, 2
```

### Unary Minus (−):

This operator is changes the sign value of variable.

```
Ex:     int x=2;
        System.out.println (x);
        System.out.println (−x);
        System.out.println (−(−x));
O/P:    3, −3, 3
```

### Assignment Operators:

This operators store a value into a variable.

```
Ex:     int x=2;
        Int y = x;
        Int z =x + y − 6;        ➔ Expression
```

| Expanded Notation | Compound Notation | Meaning |
|---|---|---|
| Sal=sal+500 | Sal+ =500 | Addition Assignment |
| num=num−12.5 | num− = 12.5 | Subtraction Assignment |
| X=x/2 | x/ = 2 | Division Assignment |
| p=p*k | p* = k | Multiplication Assignment |
| i=i%10 | i% = 10 | Module Assignment |

### Relational Operators:

These operators are supports to compare two quantities; they are used in construction of simple conditions.

| Operators | Meaning |
|---|---|
| > | Greater than |
| >= | Greater than or equals to |
| < | Less than |
| <= | Less than or equals to |
| == | Equals to |
| != | Not Equals to |

```
Ex:     if (x! = y)
        If (a >= (b + c))
        If (num == 1)
```

## Logical Operators:

These operators are used to construction to compound conditions. A compound condition is a combination of more than one simple condition

| Operators | Meaning |
|---|---|
| && | Logical AND |
| \| \| | Logical OR |
| ! | Logical NOT |

Ex: if ((a == 1) && (b! =1))
If ((x > y) | | (y >= z) | | (z < x))

## Boolean Operators:

These operators acts upon Boolean variables and they return Boolean value, or Boolean type result

| Operators | Meaning |
|---|---|
| & | boolean AND |
| \| | boolean OR |
| ! | boolean NOT |

| Ex: | boolean a = true; | boolean b = false; |
|---|---|---|
| | a \| a ➔ true | a & a ➔ true |
| | b \| b ➔ false | b & b ➔ false |
| | a \| b ➔ true | a & b ➔ false |
| | ! a ➔ false | ! b ➔ true |

## Bitwise Operators:

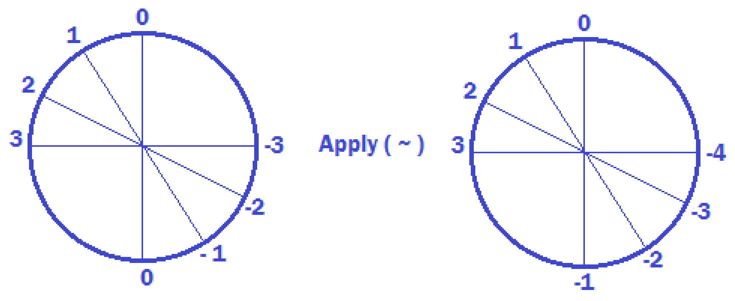
These operators act upon the individual bits (**0 & 1**) of data. These operators are used for testing, complementing or shifting bits to the right or left.

A list of seven Bitwise operators as follows…

1. Bitwise Complements (**~**)
2. Bitwise AND (**&**)
3. Bitwise OR (**|**)
4. Bitwise XOR (**^**)
5. Bitwise Left shift (**<<**)
6. Bitwise Right shift (**>>**)
7. Bitwise Zero filled Right shift (**>>>**)

## Bitwise Complement (~):

This Bitwise complementary operator (~) is a unary operator.it returns the value by complementing the each bit of operands.



1's complements

Ex: int x=10;

~ x= −11;

## Bitwise AND (&):

This operator performs ANDing operation on individual bits of numbers.

**Truth Table:**

- It gives the relations between input bits to output bits.

| X | Y | X & Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Ex: int x=10; int y=11;

X & Y = ?
X: 0000 1010
Y: 0000 1011
X&Y : 0000 1010 ➔result is 10

## Bitwise OR (|):

This operator performs ORing operation on individual bits of numbers.

**Truth Table:**

- It gives the relations between input bits to output bits.

| X | Y | X & Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

Ex: int x=10; int y=11;

X | Y = ?
X: 0000 1010
Y: 0000 1011
X|Y : 0000 1011 ➔result is 11

## Bitwise XOR (^):

This operator performs ORing operation on individual bits of numbers.

**Truth Table:**

- It gives the relations between input bits to output bits.

Ex:    int x=10; int y=11;

X ^ Y = ?
X: 0000 1010
Y: 0000 1011
X^Y : 0000 0001 ➔result is 1

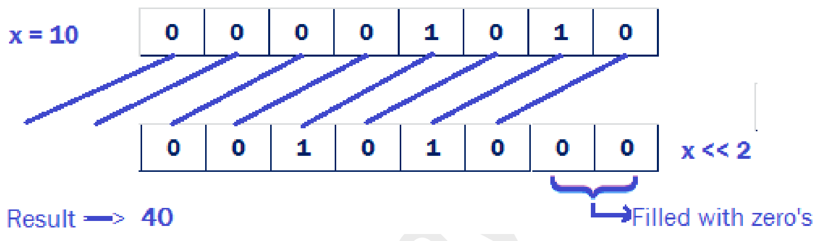
| X | Y | X & Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

## Bitwise Left shift (**<<**):

This Bitwise Left shift operator (**<<**) is a binary operator; it shifts the bits of a number towards left a specified no.of times.

X=**10**;

X<<**2**=?

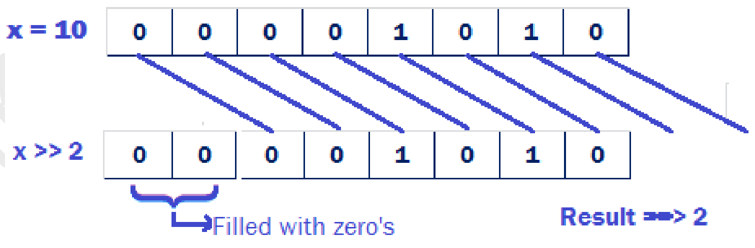


## Bitwise Right shift (**>>**):

This Bitwise Right shift operator (**<<**) is a binary operator; it shifts the bits of a number towards right a specified no.of times.

X=**10**;

X>>**2**=?

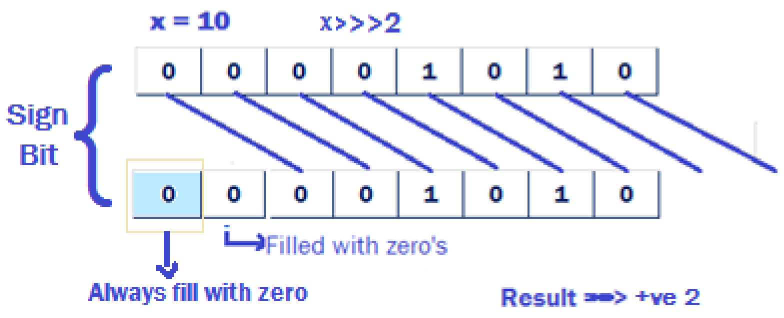


## Bitwise Zero Fill Right shift (**<<**):

This Bitwise Right shift operator (**<<**) is a binary operator; it shifts the bits of a number towards right a specified no.of times.

X=**10**;

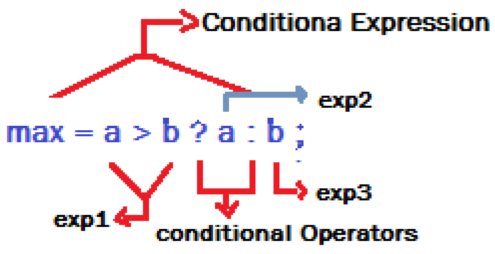
X>>>**2**=?



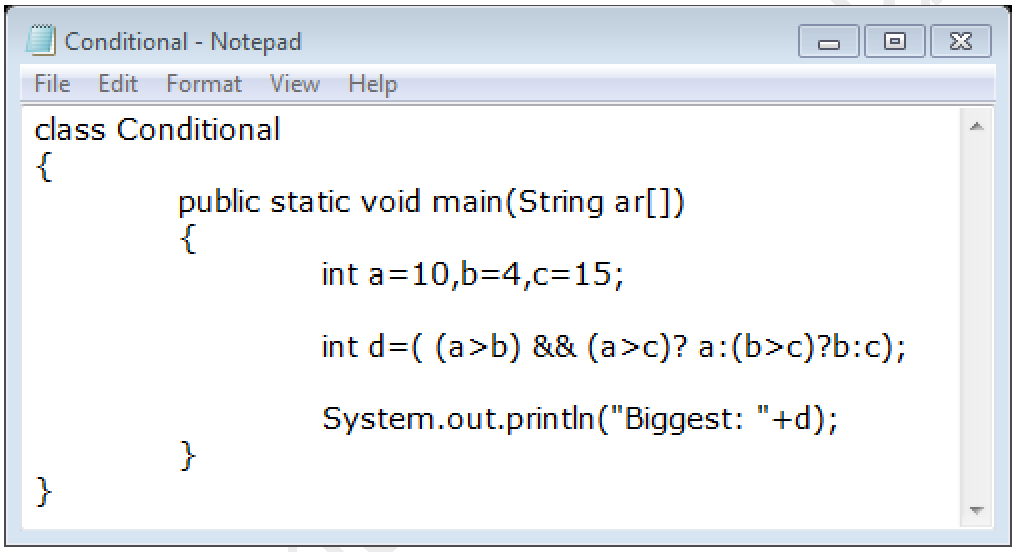
## Conditional Operators:

The conditional operator consist of two symbols (**?**) and (**:**). It acts upon three variables so it is called ternary Operators.

**Syntax:** Val = exp1? exp2: exp3;

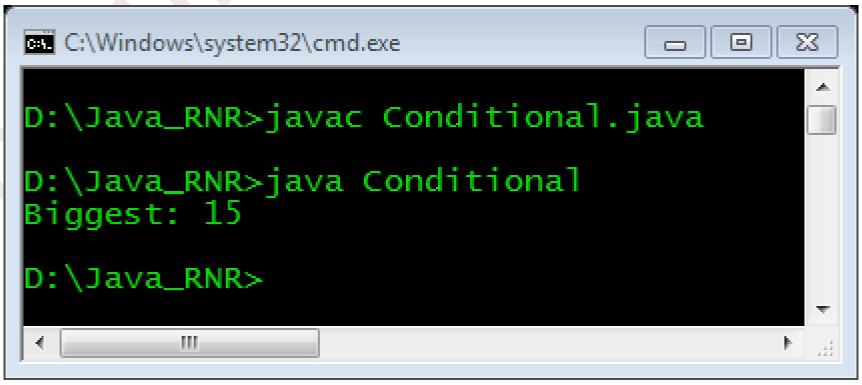


**Ex.prg: 3 To find the biggest of three no's using ternary operators.**



```
class Conditional
{
            public static void main(String ar[])
            {
                        int a=10,b=4,c=15;

                        int d=( (a>b) && (a>c)? a:(b>c)?b:c);

                        System.out.println("Biggest: "+d);
            }
}
```

**Output:**



```
D:\Java_RNR>javac Conditional.java

D:\Java_RNR>java Conditional
Biggest: 15

D:\Java_RNR>
```

## Membership or (.)Dot Operators:

This operator represents the membership. This (**.**) Dot operator can use as follows…

- To refer to class and interfaces of a packages.
  - Package**.**Class
    Ex: java.io.BufferedReader;
- To refer to variable of a class.
  - Class_name**.**variable_name; or object.variable_name;
    Ex: System.out or obj.sname;

- To refer to method of a class.
  - Class_name.method_name; or object.method_name;

    Ex: Math.sqrt(); or obj.getSal();

    **Sqrt() is a static method of the Math class.**

## Instanceof Operators:

This operator is useful to know whether an object belongs to a class or an object.

**Syntax:** Boolean val = object_name **instanceof** class_name;

Ex: Boolean x = emp instanceof Employee;

**emp will returns the true or false value to x to conform.**

## New Operators:

This operator is useful to create an object to a class or an object.

**Syntax:** class_name obj_name = **new** class_name();

Ex: Employee emp = new employee();

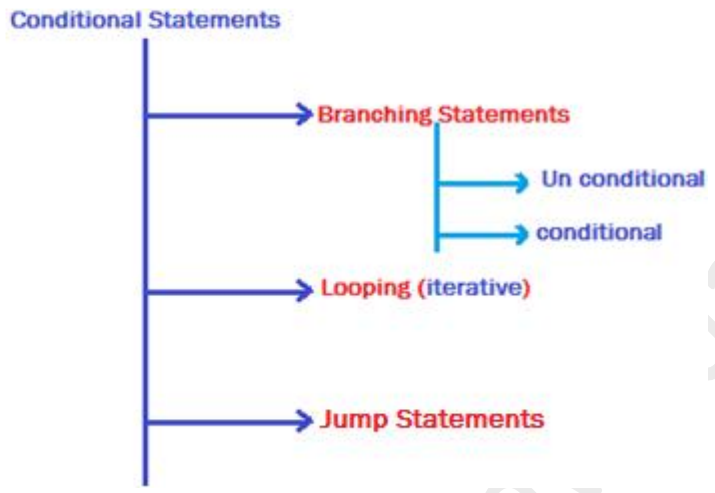
**Object are create at Heap Memory by JVM dynamically at run time.**

≈ ≈ ≈

# Control Statements

The Control statements are changes the flow of the execution of programing statements.

- Using control statements it is possible to create computers and difficult programs with complex or critical logics.



Control statements are divided into 3 parts, they are…

1. Branching Statements
2. Looping (Iterative) statements
3. Jump statements

## Branching Statements:

Branching conditional statements can be divided into two parts they are…

- Un Conditional
- Conditional

### Un Conditional:

- **goto:**

  The keywords **goto** are reserved but not used. In the early days of Java, it is a Un-Conditional Statement; Java does not allow the **goto** control statement because a mass of tangled jumps and conditional branches that makes a program virtually impossible to understand.

### Conditional:

- **if … else:**

  This conditional statements executes a task (one or more statements) depending upon whether a conditional is true or not.

**Syntax:**

```
if (Condition)
      Statements;
[else Statements ]
```

Ex.prg: 4 If… else

```
//check if a number is positive or not
Class IfElse
{
    Public static void main(String arg[])
    {
        Int num = 5;        //declaration & initialization of statement
        if (num == 0)
                System.out.println ("It is Zero.");
        else if (num > 0)
                System.out.println ("It is positive.");
        else if (num < 0)
                System.out.println ("It is nagative.");
    }
}
```

Output: 4

```
It is positive.
```

- **switch:**

  This conditional statements is useful to execute a particular task from among several tasks depending upon the value of a variable.

**Syntax:**

```
switch (variable)
{
    Case 1: Statements 1;
    .
    .
    Case n: Statements n;
    [default: default statements]
}
```

Ex.prg: 5 switch

```
//Selecting the colours
Class SwitchDemo
{
    Public static void main(String arg[])
    {
        Char colour = 'g';
        switch(g) {
            case r: System.out.println (" Red");break;
            case g: System.out.println (" Ggreen");break;
            case b: System.out.println ("Blue");break;
```

```
                    case w: System.out.println ("White");break;
                    default: System.out.println ("No Colour");break;
            }
        }
    }
```

**Output: 5**

```
Green.
```

**Note:**

If none of the statements are selected in the switch statement then default case will be executed by default.

- Switch Statement is most suitable to Menu **Driven** programs**.**

## Looping (Iterative) statements:

A loop (iterative) can execute several times but a statement can execute in one time there are four types of looping statements.

- while
- do….while
- for
- for each

### while loop: (entry Control Statements)

This loop executes a group of statements repeatedly as long as a condition is true.

**Syntax:**

```
while (condition)
{
    //body;
}
```

**Ex.prg: 6 while**

```
// To generate even no's upto 10
Class WhileDemo
{
    Public static void main(String arg[])
    {
        Int x = 2;          //declaration & initialization
        while (x <= 10) {
            System.out.print (x+",");
            x+=2;
        }
    }
}
```

**Output: 6**

```
2, 4, 6, 8, 10,
```

## do … while loop: (exit Control Statements)

This do …while loop is verifies a condition at the end of the statement hence the statement will be executed at least one's through the condition is false initially.

**Syntax:**

```
do
{
    //body;
} while (condition);
```

**Ex.prg: 7 do … while**

```
// To generate no's upto 10
Class DoWhileDemo
{
    Public static void main(String arg[])
    {
        Int x = 0;          //declaration & initialization
        do {
            System.out.print (x+",")
            x++;
        } while (x <= 10);
    }
}
```

**Output: 7**

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

## for loop:

The for loop operates as follows.

When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop.

- Initialization:

  It is important to understand that the initialization expression is only executed once. Next, condition is evaluated.

- condition:

  This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then body will executed. If it is false, the loop terminates.

Next, the iteration portion of the loop is executed.

- Iteration: ( Increment / Decrement)

  This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

  For loop is used to execute for a fixed no.of times.

**Syntax:**
```
for (initialization; condition; iteration)
{
    // body;

}
```

**Ex.prg: 8 for**

```
// To generate no's upto 10
class ForDemo
{
    public static void main(String arg[])
    {
        for (int x = 0;x <=10;i++)          //declaration & initialization
            System.out.print (x+",")
    }
}
```

**Output: 8**

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

**Note:**

We can write for loop by eliminating initialization; condition; iteration expressions or any two expressions or all of the three expressions.

- A loop that is executed for ever is called **infinite looping statement.**

```
for ( ; ; )                    for ( ; done ; )
{                              {
    //body                         //body
}                              }
```

- Infinite loops are draw back in a program
- To terminate or stop the continuous execution of infinite looping program it is needed to use a shortcut key on keyboard.i.e. press ctrl + c on keyboard.
- Nested Loops

  Java allows loops to be nested. That is, one loop may be inside another.

  For example, here is a syntax that nests **for** loops:

**Syntax:**
```
for (initialization; condition; iteration)
{
    for (initialization; condition; iteration)
    {
            // body;

     }

}
```

foreach loop:

This loop repeatedly executes group of statements **foreach** element of a collection

- A collection represents a group of elements or objects.
  Ex: Any Array []; & java.util classes are the examples of Collections
  String arg[]
- foreach statements is executed as many time as elements are exist in collections.
- Using foreach statement each element of an array arr[] will executes.

**Syntax:**
```
foreach (var: Collection_Name)
{
    // body;
}
```

**Ex.prg: 9 foreach**

```
//diplay the elements of a collection
class ForEach
{
    public static void main(String arg[])
    {
        int arr[] = {55, 66, 77, 88, 99 };      //array
        //read element by element from arr[]
        for(int x: arr)
        System.out.print ("Element :+x);
    }
}
```

**Output: 9**

```
55, 66, 77, 88, 99,
```

**Jump statements**:

A jump statement allows your program to execute in a non-leaner fashion. Java supports three jump statements: **break**, **continue**, and **return**. These statements transfer control to another part of your program.

Java supports three jump statements:

- **break**,
- **continue**, and
- **Return**.

## Break:

In Java, the **break** statement has three uses.

- First, as you have seen, it terminates a statement sequence in a **switch** statement.
- Second, it can be used to exit a loop.
- Third, it can be used as a "civilized" form of goto.

  The last two uses are explained here.

### Using break to exit a loop

**Ex.prg: 10 break**

```
// Using break to exit a loop.
class BreakLoop
{
      public static void main(String args[])
      {
                for(int i=0; i<100; i++)
                {
                     if(i == 10) break;           // terminate loop if i is 10
                     System.out.print(i+ ",");
                }
                System.out.println("Loop complete.");
      }
}
```

**Output: 10**

```
0,1,2,3,4,5,6,7,8,9,Loop complete.
```

### Using break as a Form of Goto:

The general form of the labeled break statement is shown here:

```
break label;                    //goto end of block
```

Here, label is the name of a label that identifies a block of code. When this form of break executes, control is transferred out of the named block of code. The labeled block of code must enclose the break statement, but it does not need to be the immediately enclosing block. This means that you can use a labeled break statement to exit from a set of nested blocks. But you cannot use break to transfer control to a block of code that does not enclose the break statement.

Ex.prg: 11 breakGoto

```
// Using break as a civilized form of goto.
class BreakGoto
{
    public static void main(String args[])
    {
        boolean t = true;
        first: {
                second: {
                        third: {
                                System.out.println("Before the break.");
                                if(t) break second;   // break out of second block
                                System.out.println("This won't execute");
                        }
                        System.out.println("This won't execute");
                }
                System.out.println("This is after second block.");
        }
    }
}
```

Output: 11

```
Before the break.
This is after second block.
```

Continue:

It is useful to force an early iteration of a loop. That is, you might want to continue running the loop, but stop processing the remainder of the code in its body for this particular iteration.

Here is an example program that uses **continue** to cause two numbers to be printed on each line:

Ex.prg: 12 continue

```
// Demonstrate continue.
class Continue
{
    public static void main(String args[])
    {
          for (int i=0; i<10; i++) {
             System.out.print(i + " ");
             if (i%2 == 0) continue;
               System.out.println("");
          }
    }
}
```

**Output: 12**

```
0 1

2 3

4 5

6 7

8 9
```

As with the break statement, continue may specify a label to describe which enclosing loop to continue. Here is an example program that uses continue to print a triangular multiplication table for 0 through 9.

**Ex.prg: 13 Continue label**

```
// Using continue with a label.
class ContinueLabel
{
    public static void main(String args[])
    {
          outer: for (int i=0; i<10; i++)
          {
               for(int j=0; j<10; j++)
              {
                    if(j > i)
                    {
                         System.out.println();
                         continue outer;
                    }
                    System.out.print(" " + (i * j));
              }
          }
          System.out.println();
    }
}
```

**Output: 13**

```
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

## Return:

The last control statement is **return**. The **return** statement is used to explicitly return from a method. That is,

- This **return** statement immediately terminates the method in which it is executed and come back to the calling method.
- This return statement can be return some resulting value to the calling method.

The following example illustrates this point:

**Ex.prg: 14 return**

```
// Demonstrate return.
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");
        if(t) return; // return to caller
        System.out.println("This won't execute.");
    }
}
```

**Output: 14**

```
Before the return.
```

**Note:**

Return statement in main method will terminate the application.

**iiQ** **What is difference between System.exit(0) & System.exit(1)?**

**Reply:** API (Application Programming Interface) documentation is a file that contains discretion of all the features of software's, a product, or a Technology.
API Document is created by using the java doc compiler.

- **System.exit(0) :** Represents normal termination.
- **System.exit(1) :** Represents termination due to an error.

≈ ≈ ≈

# IO Streams basics

A **Stream** represents the act of flow of data from one place to another place. Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information. A stream is linked to a physical device by the Java I/O system.

Ex: Play a video or audio file as it downloads from the Internet.

There are two types of streams…

1. **InputStreams: They are reading or receive the data.**
2. **OutputStreams: They are writing or send the data.**

   All streams are represented as classes in **java.io** packages.

**The Predefined Streams:**

All Java programs automatically import the **java.lang** package. This package defines a class called **System**, which encapsulates several aspects of the run-time environment.

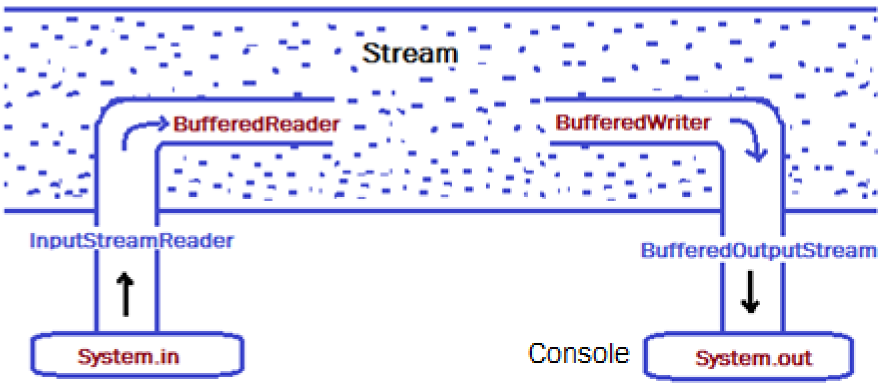
For example, using some of its methods, you can obtain the current time and the settings of various properties associated with the **system**.

**System** also contains three predefined stream variables, **in, out, and errs.** These fields are declared as **public** and **static** within System. This means that they can be used by any other part of your program and without reference to a specific System object.

- **System.out** refers to the standard output stream. By default, this is the console.
- **System.in** refers to standard input, which is the keyboard by default.
- **System.err** refers to the standard error stream, which also is the console by default.

However, these streams may be redirected to any compatible I/O device.

System.in is an object of type **InputStream**; System.out and System.err are objects of type **PrintStream**.These are **byte** streams, even though they typically are used to read and write characters from and to the console. Here we can wrap these within character-based streams.



**Reading Console Input:**

- Attach **InputeStreamReader** to the keyboard.

i.e. The object of InputStreamReader to the keyboard object.

**Syntax:** InputStreamReader isr = new InputStreamReader (System.in);

- Attach **InputeStreamReader** to the **BufferedReader**.

  i.e. The object of InputStreamReader to the BufferedReader object.

**Syntax:** BufferedReader br = new BufferedReader (isr);

- Read data at object br using **read()** or **readLine ()** methods
  - **read ()** : It reads a single line characters.
  - **readLine ()** : It reads a string or a group of characters.

The preferred method of reading console input for Java 2 is to use a character-oriented stream, which makes your program easier to internationalize and maintain.

**Note:**

Java does not have a generalized console input method that parallels the standard C function scanf ( ) or C++ input operators.

### Reading Characters:

To read a character from a BufferedReader, use **read( )**. The version of **read( )** that we will be using is int read( ) throws IOException Each time that read( ) is called, it reads a character from the input stream and returns it as an integer value. It returns –1 when the end of the stream is encountered. As you can see, it can throw an IOException.

The following program demonstrates read( ) by reading characters from the console until the user types a "q":

**Ex.prg: 15 read a character**

```
// Use a BufferedReader to read characters from the console.
import java.io.*;
class BRRead
{
    public static void main(String args[])throws IOException
    {
        char c;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter characters, 'q' to quit.");
        // read characters
        do
        {
            c = (char) br.read();   //type chasting
            System.out.println("U entered: "+c);
        }while(c != 'q');
    }
}
```

**Output: 15**

```
Enter characters, 'q' to quit.
renuq
U entered: r
U entered: e
U entered: n
U entered: u
U entered: q.
```

This output may look a little different from what you expected, because System.in is line buffered, by default. This means that no input is actually passed to the program until you press ENTER. As you can guess, this does not make read( ) particularly valuable for interactive, console input.

## Reading String:

To read a string from the keyboard, use the version of readLine( ) that is a member of the BufferedReader class. Its general form is shown here: String readLine( ) throws IOException As you can see, it returns a String object. The following program demonstrates BufferedReader and the readLine( ) method; the program reads and displays lines of text until you enter the word “stop”:

**Ex.prg: 16 read a String**

```
// Read a string from console using a BufferedReader.
import java.io.*;
class BRReadLines
{
    public static void main(String args[])throws IOException
    {
        // create a BufferedReader using System.in
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str;
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'stop' to quit.");
        do {
            str = br.readLine();
            System.out.println(str);
        } while(!str.equals("stop"));
    }
}
```

**Output: 16**

```
Enter lines of text.
Enter 'stop' to quit.
abcd
abcd
stop
```

The next example creates a tiny text editor. It creates an array of String objects and then reads in lines of text, storing each line in the array. It will read up to 100 lines or until you enter "stop". It uses a BufferedReader to read from the console.

**Reading Integer value:**

```
String str = br.readLine();
int n = Integer.parseInt(str);        or     int n = Integer.parseInt (br.readLine());
```

**Reading floating point number:**

```
float f = Float.parseFloat(br.read());     or   float n = Float.parseFloat (br.readLine());
```

- Float data type can represents up to 7 decimal digits it will display accurately after decimal point.

**Reading double number:**

```
double d = Double.parseDouble(br.readLine());
```

- Double data type can represents up to **15** decimal digits it will display accurately after decimal point

**Writing Console Output:**

Console output is most easily accomplished with print( ) and println( ), These methods are defined by the class PrintStream (which is the type of the object referenced by System.out). Even though System.out is a byte stream, using it for simple program output is still acceptable. However; a character-based alternative is described in the next section. Because PrintStream is an output stream derived from OutputStream, it also implements the low-level method write( ). Thus, write( ) can be used to write to the console.

The simplest form of write( ) defined by PrintStream is shown here:

```
void write(int byteval)
```

This method writes to the stream the byte specified by byteval. Although byteval is declared as an integer, only the low-order eight bits are written.
Here is a short example that uses write( ) to output the character "A" followed by a newline to the screen:

**Ex.prg: 17 Console Output**

```
// Demonstrate System.out.write().
class WriteDemo
{
    public static void main(String args[])
    {
        int b;
        b = 'A';
        System.out.write(b);
```

```
                System.out.write('\n');
        }
    }
```

Output: 17

```
A
```

You will not often use write( ) to perform console output (although doing so might be useful in some situations), because print( ) and println( ) are substantially easier to use.

## BufferedWriter:

A BufferedWriter is a Writer that adds a flush () method that can be used to ensure that data buffers are physically written to the actual output stream. Using a BufferedWriter can increase performance by reducing the number of times data is actually physically written to the output stream.

A BufferedWriter has these two constructors:

BufferedWriter(Writer outputStream)

BufferedWriter(Writer outputStream, int bufSize)

The first form creates a buffered stream using a buffer with a default size. In the second, the size of the buffer is passed in bufSize.

## BufferedOutputStream:

A BufferedOutputStream is similar to any OutputStream with the exception of an added flush () method that is used to ensure that data buffers are physically written to the actual output device. Since the point of a BufferedOutputStream is to improve performance by reducing the number of times the system actually writes data, you may need to call flush () to cause any data that is in the buffer to get written. Unlike buffered input, buffering output does not provide additional functionality. Buffers for output in Java are there to increase performance.
Here are the two available constructors:

BufferedOutputStream(OutputStream outputStream)

BufferedOutputStream(OutputStream outputStream, int bufSize)

The first form creates a buffered stream using a buffer of 512 bytes. In the second form, the size of the buffer is passed in bufSize.

## Type Conversion or Casting:

Converting one type of data type into another type i.e. assigns a value of one type to a variable of another type generally it is named as Type Casting. If the two types are compatible, then Java will perform the conversion automatically.

*For example:* it is always possible to assign an **int** value to a **long** variable. However, not all types are compatible, and thus, not all type conversions are **implicitly** allowed. For instance, there is no conversion defined from **double** to **byte**. Fortunately, it is still possible to obtain a conversion between incompatible types. To do so, you must use a **cast operator**, which performs an **explicit** conversion between incompatible types.

## Automatic Conversions:

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

When these two conditions are met, a ***widening conversion*** takes place. *For example*, the **int** type is always large enough to hold all valid **byte** values, so no explicit cast statement is required.
For **widening conversions**, the numeric types, including integer and floating-point types, are compatible with each other. However, the numeric types are not compatible with **char** or **boolean**. Also, char and boolean are not compatible with each other.

## Casting Incompatible Types:

This conversion will not be performed automatically, *For example*, we want to assign an **int** value to a **byte** variable? because a byte is **smaller than** an int. This kind of conversion is sometimes called a ***narrowing conversion***, since you are explicitly making the value narrower so that it will fit into the target type.

To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form:

**Syntax:** (target-type) value;

Here, target-type specifies the desired type to convert the specified value to.

*For example*: the following fragment casts an ***int*** to a ***byte***. If the integer's value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range.

```
int a;
byte b;
// …
b = (byte) a;
```

A different type of conversion will occur when a floating-point value is assigned to an integer type: ***truncation***. As you know, integers do not have fractional components. Thus, when a floating-point value is assigned to an integer type, the fractional component is lost.

*For example:* if the value **1.23** is assigned to an **integer**, the resulting value will simply be **1**. The **0.23** will have been ***truncated***. Of course, if the size of the whole number component is too large to fit into the target integer type, then that value will be reduced modulo the target type's range.

The following program demonstrates some type conversions that require casts:

**Ex.prg: 18 Conversion**

```
// Demonstrate casts.
class Conversion
{
    public static void main(String args[])
    {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("\nConversion of int to byte.");
        b = (byte) i;
        System.out.println("i and b " + i + " " + b);
        System.out.println("\nConversion of double to int.");
        i = (int) d;
        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte) d;
        System.out.println("d and b " + d + " " + b);
    }
}
```

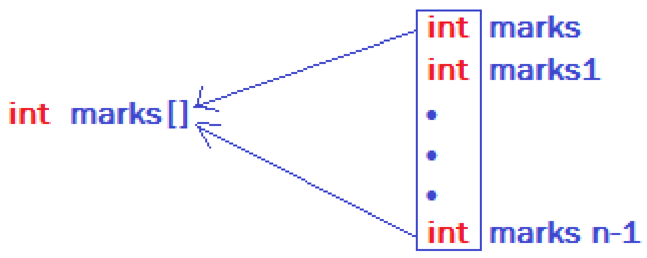
**Output: 18**

```
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
d and i 323.142 323
Conversion of double to byte.
d and b 323.142 67
```

≈≈≈

# Arrays

An **Array** represents a group of elements of same type. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.



Advantage:

- The main advantage of an array is to handle group of elements with the same name in a sequential memory space.

There are two types of Arrays...

1. **Single Dimension (1D)**
2. **Multi Dimension (2D, 3D)**

All Arrays can also be treated as a collection.

**Single Dimension:**

A single dimension (**1**D) array represents one several row or column. The general form of a one-dimensional array declaration is

**Syntax:** **type var-name[ ][ ];**

Here, type declares the base type for the array determines what type of data the array will hold. The base type determines the data type of each element that comprises the array.

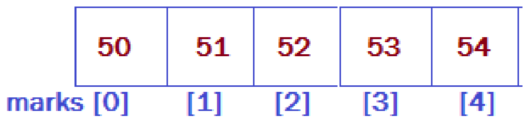
*For example:* the following declares an array named marks obtained by ***one student*** with the type "array of int":

```
int marks[];            //declaration
```

Fortunately no arrays exist like this, and the size for this array is set to null, which represents an array with no value. To allocate memory there are two ways in java.

- We can declare a **1**D array and initialize it with element

```
int marks[] = {50, 51, 52, 53, 54};   //initialization
```



A **1**D Array will have only one index that represents elements position number in the array.

```
float X[] = {1.1f, 2.2f, 3.3f, 4.4f, 5.5f};    //initialization
char ch[] = {'a', 'b', 'c', 'd', 'e'};          //initialization
```

- We can allote memory for a **1**D using new operator and letter storing the elements.

  int marks[] = new int[**5**] ➔ size of an array.

  marks[] = **50;**

  marks[] = **51;**

  marks[] = **52;** **etc. or**

  string str[] = new string[**20**];

  double d[] = new double[**100**]

***Note:***

Array name**.**length returns the size of the array, Here **.**length is a property, it will calculate size of an array.

Ex.prg: 19 Single Dimension

```
//Total marks and percentage of a student
import java.io.*;
class Arr
{
      public static void main(String arr[])throws IOException
      {
        BufferedReader  br = new BufferedReader(new InputStreamReader(System.in));

              System.out.print("How many SUbjects: ");
              int n=Integer.parseInt(br.readLine());
              int marks[] = new int[n];
              for(int i=o;i<n;i++)
              {
                    System.out.print("Enter Marks:");
                    marks[i]=Integer.parseInt(br.readLine());
              }
              int tot=o;
              for(int i=o;i<n;i++)
              {
                    System.out.println("Marks "+i +": "+marks[i]);
                    tot+=marks[i];
              }
          System.out.println("Total Marks: "+tot);
              float per=(float)(tot/n);//Type casting
          System.out.println("Percentage Marks: "+per);
      }
}
```

Output: 19

```
How many Subjects: 4
Enter Marks:50
Enter Marks: 51
Enter Marks: 52
Enter Marks: 53
Marks o: 50
Marks 1: 51
Marks 2: 52
Marks 3: 53
Total Marks: 206
Percentage Marks: 51.0
```

**Two or Multi Dimension:**

A two or Multi dimension (**2**D) array represents one row or one column. To declare a multidimensional array variable, specify each additional index using another set of square brackets. The following declares a two-dimensional array variable called 2D.

**Syntax:** type var-name[ ][ ];

Here, type declares the base type for the array determines what type of data the array will hold. The base type determines the data type of each element that comprises the array.

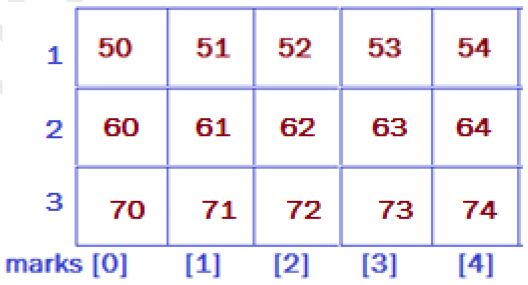
*For example:* the following declares an array named marks obtained by a ***group of student*** with the type "array of int":

```
int marks[][];          //declaration
```

Fortunately no arrays exist like this, and the size for this array is set to null, which represents an array with no value. To allocate memory there are two ways in java.

- We can declare a **2**D array and initialize it with element

```
int marks[][] = {{50, 51, 52, 53, 54},
                 {60, 61, 62, 63, 64},        //initialization
                 {70, 71, 72, 73, 74}};
```



A **2**D Array will have two indexes that represents elements position in row and column position numbers in the array.

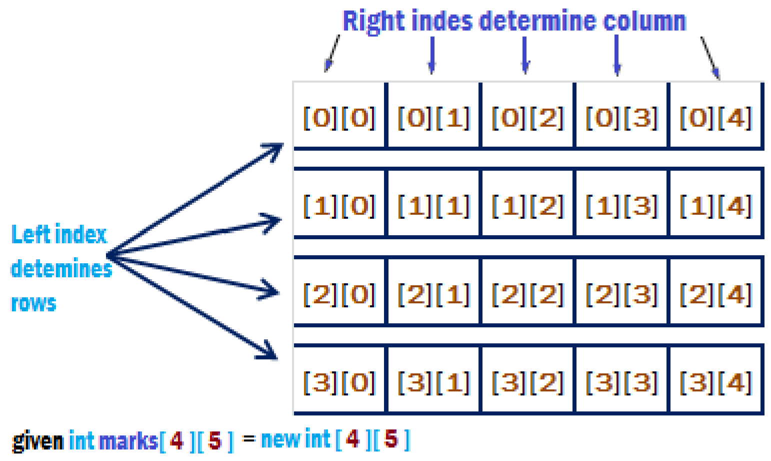
- i.e. TwoD array is a combination of several oneD arrays

```
float X[] = {{1.1f, 1.2f, 1.3f, 1.4f, 1.5f},      //initialization
             {2.1f, 2.2f, 2.3f, 2.4f, 2.5f},
             {3.1f, 3.2f, 3.3f, 3.4f, 3.5f}};
```

- We can allot memory for a **2**D using new operator and letter storing the elements.

```
int marks[ ][ ] = new int[ 4][5]        ➔ size of an array.
    byte b[ ][ ] = new byte[10][20];
```

This allocates a **4** by **5** array and assigns it to marks. Internally this matrix is implemented as an array of arrays of int. conceptually; this array will look like the one shown in below figure.



The following program numbers each element in the array from left to right, top to bottom, and then displays these values:

**Note:**

We can use array symbol before the array name also Ex: String [][] = new String[ **4**][**5** ]

**Ex.prg: 20 Two Dimensions**

```
//Two Dimensional Array
import java.io.*;
class Arr2
{
      public static void main(String arr[])throws IOException
      {
            float x[][] = {{1.1f, 1.2f, 1.3f, 1.4f, 1.5f},
                           {2.1f, 2.2f, 2.3f, 2.4f, 2.5f},
                           {3.1f, 3.2f, 3.3f, 3.4f, 3.5f}};
            for(int i=0;i<3;i++)
            {
                  for(int j=0;j<4;j++)
                  {
                  System.out.print(x[i][j]+"\t");
                  }
               System.out.println();
            }
      }
}
```

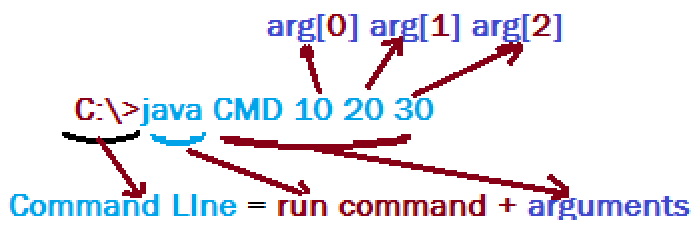
**Output: 20**

```
1.1   1.2   1.3   1.4
2.1   2.2   2.3   2.4
3.1   3.2   3.3   3.4
```

**iiQ** What is Command Line Arguments?

**Reply:** Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command-line arguments to ***main( ).*** A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to *main( )*.

**Syntax:**



For example, the following program displays all of the command-line arguments that it is called with:

Ex.prg: 21 Command Line Arguments

```
//Command Line Arguments
import java.io.*;
class CMD
{
    public static void main(String arg[])throws IOException
    {
        int n=arg.length;
        System.out.println("No.of arguments: "+n);
        System.out.println("Arguments");
        for(int i=o;i<n;i++)
        {
          System.out.println(arg[i]);
        }
    }
}
```

Output: 21

```
C:\>java CMD 10 20 30
No.of arguments: 3
10
20
30
```

≈ ≈ ≈

In Java a string is a group or sequence of characters, but, unlike

- C, C++ languages that implement strings as character type arrays, whereas
- In Java implements strings as objects of type String.

Implementing strings as built-in objects allows Java to provide a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string. Also, String objects can be constructed a number of ways, making it easy to obtain a string when needed.

## Creating a String:

There are three ways to creating the strings…

1. We can declare string type variable and initialize it with a group of characters.

   Ex:String str = 'Hello';

2. We can create a string class object and pass a group of characters into the objects.

   Ex:String s2 = new String ("Hay");

3. We can convert a characters type into class objects by passing the array to the objects.

   Ex:char ch[] = {'a', 'b', 'c', 'd'}

   String str1 = new String (ch);

   str2 = new String (ch,1,3)// bcd will be O/P

## String Manipulating Methods:

This manipulating methods (functions) are available in **java.long.String.*;**

Some Important string methods of Core Java:

### String concatenates (String str)

This method use for concatenate the strings when calling the string with str.

***Note:***

In Some case **+** will also work to do the concatenations.

### int length()

This method returns the length of the string. i.e. returns the no.of character in given string.

Ex: s1.length

```
String s1="Hydra", s2="Bad";
String x=s1.concate(s2);                         // s1+s2
x.length;
```

Char CharAt(int i)

This method returns the characters at the specified locations. Stating from o th locations.

int compareTo(String str)

int compareToIngnoreCase(String str)

This method returns

➔A negative value if the calling string comes before str in dictionary orders,

➔A positive value if the string comes after str or

➔Zero if the strings str equal.

Ex:
```
s1="Boy",s2="Box";
int n =s1.compareTo(s2);
if s1 > s2, n = +ve
if s1 < s2, n = +ve
if s1 == s2, n = 0
```

- CompareTo() ➔is a method which is compare the case sensitive.

boolean equals(String str)

boolean equalsIgnorecase(String str)

This method returns true if the calling string equals str.

boolean startsWith(String prefix)

This method returns true if the invoking string starts with *prefix*

boolean endsWith(String suffix)

This method returns true if the invoking string starts with *suffix*

**Note:**

Above two methods are case sensitive when comparison.

int indexsOf(String str)

This method returns the first occurrence of the str in the string.

Ex:
```
String s1="this is a box";
int n=str.indexOf("is");
```

int LastIndexOf(String str)

This method returns the last occurrence of the str in the string.

**Note:**

Both the above methods returns negative values, if str not found in the calling string counting starts from zero.

String replace(char oldChar,char newChar)

This method returns a new string that is obtained by replacing all characters old characters in the string with new characters.

String substring(int beginIndex)

This method returns a new string consists of all characters from *beginIndx* until *end* of the string.

String substring(int beginIndex,int endIndex)

This method returns a new string consists of all characters from *beginIndx* until *endIndex(exclusives)*

String toLowerCase()

Converts all characters into ***lower case*** latters.

String toUpperCase()

Converts all characters into ***UPPER CASE*** latters.

String trim()

Eliminates all loading and trailing spaces.

**Ex.prg: 21 Command Line Arguments**

```
//Understanding String Concept
import java.io.*;
class Str
{
    public static void main(String arg[])throws IOException
    {
        String s1 = "This is Java";          //Creating New String
        String s2 = new String("I Like it");
        Char ch[] = {"i". "N", "e", "t", "S", "o", "l", "v"};
        String s3 = new String(ch);

        System.out.println(" S1:"+ s1);      //Display the String

        System.out.println(" S2:"+ s2);

        System.out.println(" S3:"+ s3);

        //To Know length of the String
        System.out.println(" length of S1:"+ s1.length);

        //Check if s1 start with this or not.
```

```
            boolean X=s1.startsWith("This");
            if(X == true)
                System.out.println ("S1starts with "This"");
            Else
                System.out.println ("S1starts with "This"");
            //Retrieve a Sub string  from S2 S3
            String sub1 = s2.substring(0,6);
            String sub2 = s3.substring(0);
            System.out.println(sub1+sub2);

            //Converting the case of string S1
            System.out.println("Upper Case:S1:"+s1.toUpperCase());
            System.out.println("Lower Case:S2"+ s2.toLowerCase());
        }
    }
```

**Output: 21**

```
S1: This is Java
S2: I like it
S3: iNetSolv
Length of S1:12
S1starts with "this"
I like iNet Solv
Upper Case: S1: THIS IS JAVA
Lower Case: S2 i like it
```