

## Introduction to Java: -

- ✓ Java was originally developed by Sun Microsystems.
- ✓ Java is open source and has a worldwide community for its continued development and growth.
- ✓ It is a front end tool.
- ✓ java is the name of an island in Indonesia where first coffee was produced.
- ✓ Originally java was introduced by James Gosling and released in 1995.

## Note: -

There are 3 platforms for Java programming language:

### **1. Java Platform Standard Edition (J2SE)**

It is considered as foundation edition to develop standalone applications like swings, applets, etc.

### **2. Java Platform Enterprise Edition (J2EE)**

It contains additional frameworks and libraries to support distributed and Web development applications.

### **3. Java Platform Micro Edition (J2ME)**

It is a development and deployment platform for portable, embedded and mobile devices including sensors, routers, switches, gateways, iPhones, printers, TV set-top boxes, etc.

## Java IDE: -

Java Integrated Development Environment (Java IDE) includes text editor, debugger, compiler and some more tools that will help in automation, testing and analyzing the java application. IDE allows developers to convert their logical code into useful software application Following are the best Java IDEs

1. Eclipse
2. NetBeans
3. BLUEJ
4. JDeveloper
5. DrJava
6. Greenfoot
7. IntelliJ Idea
8. JGrasp
9. Android Studio
10. JCreator

## Output Method in Java: -

### **I) To print the output in the same line.**

```
System.out.print("Text Message");  
System.out.print("Text Message" + Variable);
```

### **II) To print the output in a separate line.**

```
System.out.println("Text Message");  
System.out.println("Text Message" + Variable);
```

Note: - Java is case sensitive. We should write the code in lower case letters.

## Structure of java program: -

```
documentation  
import package_name;  
class class_name  
{  
    public static void main(String args[])  
    {  
        variables declaration part;  
        executable statements;  
    }  
}
```

Example:-

```
/* Java program to print sum of two numbers */  
import java.io.*;  
class Prg1  
{  
    public static void main(String args[])  
    {  
        int a, b, c;  
        a=10;  
        b=15;  
        c=a+b;  
        System.out.println("Sum=" + c);  
    }  
}
```

### Note: -

The extension of java program should be ".java"  
Save the program with class name. (Ex:- Prg1.java)

How to Compile and Run Java program?

1. Open Command Prompt.
2. Change into the folder.
3. user javac command to compile.
4. use java command to run.

```
javac Prg1.java  
java Prg1
```

### Dynamic Input: -

- ✓ Input methods are used to read values into variables while executing a java program.
- ✓ The following are predefined input methods in Scanner class.
- ✓ Scanner class is available in java.util package.
- ✓ So we should import the util package using the following statement.

```
import java.util.Scanner;
```

1. nextByte() - accepts byte type value and returns
2. nextShort() - accepts short type value and returns
3. nextInt() - accepts an integer and returns
4. nextLong() - accepts long type value and returns
5. nextFloat - accepts numbers with decimals and returns
6. nextDouble() - accepts numbers with decimals and returns
7. nextBoolean() - accepts either true or false value.
8. nextCharAt(0) - accepts a single character and returns
9. next() - accepts a single word and returns
10. nextLine() - accepts a line of text and returns

```
/* Java program to read values into variables and print given values */  
import java.util.Scanner;  
class Prg2
```

```

{
public static void main(String args[])
{
    byte a;
    float b;
    boolean c;
    char d;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter byte, float, boolean and char type values\n");
    a=in.nextByte();
    b=in.nextFloat();
    c=in.nextBoolean();
    d=in.next().charAt(0);
    System.out.println("Byte Value = " + a);
    System.out.println("Float Value = " + b);
    System.out.println("Boolean Value = " + c);
    System.out.println("Char Value = " + d);
}
}

```

## Operators in Java

An operator is a symbol used to perform certain operation.

There are different kind of operators in Java.

### **I) Arithmetic operators: -**

The symbols +, -, \*, / and % are called arithmetic operators.

These are used to perform mathematical calculations.

**/\* Ex: - Perform all arithmetic operations on two given numbers \*/**

```

import java.util.*;
public class Prg3
{
    public static void main(String[] args)
    {
        int a,b;
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter two numbers\n");
        a = obj.nextInt();
        b = obj.nextInt();
    }
}

```

```

        System.out.println("Sum      = " + (a+b));
        System.out.println("Difference = " + (a-b));
        System.out.println("Product   = " + (a*b));
        System.out.println("Quotient  = " + (a/b));
        System.out.println("Remainder = " + (a%b));
    }
}

```

## II) Relational Operators: -

The symbols >, <, >=, <=, == and != are called relational (or) comparison operators.

These are used to write conditions.

Its output may be true or false, nothing else.

**/\* Ex: - Write java program to print values of relational expressions \*/**

```

import java.io.*;
public class Prg4
{
    public static void main(String[] args)
    {
        int a=12,b=6;
        System.out.println(a>=b);      // true
        System.out.println(a<=2*b);    // true
        System.out.println(a!=b);      // true
        System.out.println(b==a/2);    // true
        System.out.println(b>a);       // false
        System.out.println(a<b);       // false
    }
}

```

## III) Logical Operators: -

The symbols &&, || and ! are called logical operators.

These are used to combine two or more conditions.

**Truth Table for Logical and (&&): -**

Condition1	Condition-2	Condition-1 && Condition-2
True	True	True
True	False	False
False	True	False
False	False	False

### Truth Table for Logical or (||): -

Condition1	Condition-2	Condition-1    Condition-2
True	True	True
True	False	True
False	True	True
False	False	False

### Truth Table for NOT operator (!): -

Condition	!Condition
True	False
False	True

### **/\* Ex: - Print values of relational expressions using logical operators \*/**

```
import java.io.*;
public class Prg5
{
    public static void main(String[] args)
    {
        int a=12,b=6;
        boolean x=true,y=false;
        System.out.println(a>10 && b<10);           // true && true → true
        System.out.println(a==2*b && b!=a);         // true && true → true
        System.out.println(a>=b || a<b);           // true || false → true
        System.out.println(a<=b || a<b);           // false || false → false
        System.out.println(!x);                     // not true → false
        System.out.println(!y);                     // not false → true
    }
}
```

### **IV Assignment Operator: -**

"=" is called assignment operator, used to store some value into a variable.

Date Type	Assigning Value
int	int a=25;
Long	Long b=25000;
Float	float c=3.142f;
Char	char d='M';
String	String e="Engineering";

Note: - While assigning values, character should be enclosed in single quotations and string values should be enclosed in double quotations.

**/\* Example: - Write java program using assignment operator \*/**

```
import java.io.*;
public class Prg6
{
    public static void main(String[] args)
    {
        int a=25;
        long b=25000;
        float c=3.142f;
        char d='M';
        String e="Engineering";
        boolean f=true;
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);
    }
}
```

**Compound Assignment Operator: -** The assignment operator combine with another binary operator is known as compound assignment operator.

Compound assignment operators are listed below.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

**/\* Example: - Write java program using compound assignment operators \*/**

```

import java.io.*;
public class Prg7
{
    public static void main(String[] args)
    {
        int a=10, b=5, c=2;
        a+=b;
        System.out.println("a = " + a);
        b-=a/b;
        System.out.println("b = " + b);
        c*=a+b;
        System.out.println("c = " + c);
    }
}

```

Output: -

a = 15

b = 2

c = 34

### V) Ternary Operator: -

"? : " is called ternary operator. It is also known as conditional operator.

Syntax: -

**(condition) ? value1 : Value2;**

Working: -

If condition is TRUE then it returns value1. Otherwise it returns value2.

**/\* Ex: - Find biggest of two given numbers using ternary operator \*/**

```

import java.util.*;
public class Prg8
{
    public static void main(String[] args)
    {
        int a, b, x;
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter two numbers\n");
        a = obj.nextInt();
        b = obj.nextInt();
    }
}

```



```

x = (a>b) ? a : b;
System.out.println("Big Number = " + x);
}
}

```

**VI) Bitwise Operators: -**

The symbols &, |, ^, ~, << and >> are called bitwise operators.

Name	Symbol	Usage	What it does
Bitwise And	&	a&b	Returns 1 only if both the bits are 1
Bitwise Or		a b	Returns 1 if one of the bits is 1
Bitwise Not	~	~a	Returns the complement of a bit
Bitwise Xor	^	a^b	Returns 0 if both the bits are same else 1
Bitwise Left shift	<<	a<<n	Shifts a towards left by n digits
Bitwise Right shift	>>	a>>n	Shifts a towards right by n digits

These are used to perform the operations on the data at the bit-level. For handling electronics and IOT-related operations, programmers use bitwise operators.

AND		
A	B	Result
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	Result
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
A	B	Result
0	0	0
0	1	1
1	0	1
1	1	0

NOT	
A	Result
0	1
1	0

Example: -

Binary value of 7 is 111.

Binary value of 4 is 100.

Below is the process for calculating values of 7&4, 7|4 and 7^4.

**~a means – (a+1).**

$$\begin{array}{r} 7 = 0\ 1\ 1\ 1 \\ 4 = 0\ 1\ 0\ 0 \\ \hline 7 \text{ AND } 4 = 0\ 1\ 0\ 0 = 4 \end{array}$$

$$\begin{array}{r} 7 = 0\ 1\ 1\ 1 \\ 4 = 0\ 1\ 0\ 0 \\ \hline 7 \text{ OR } 4 = 0\ 1\ 1\ 1 = 7 \end{array}$$

$$\begin{array}{r} 7 = 0\ 1\ 1\ 1 \\ 4 = 0\ 1\ 0\ 0 \\ \hline 7 \text{ XOR } 4 = 0\ 0\ 1\ 1 = 3 \end{array}$$

**Bitwise Not Operator (~)** It is also known as bitwise complement operator.

Formula: -  $\sim a = -(a+1)$

Ex: -  $\sim 48$  gives -49

**Bitwise Left shift Operator**  $48 \ll 4 \Rightarrow 48 \times 2^4 \Rightarrow 48 \times 16 = 768$

**Bitwise Right shift Operator**  $48 \gg 4 \Rightarrow 48 / 2^4 \Rightarrow 48 / 16 = 3$

**/\* Write java program using bitwise operators \*/**

```
import java.util.*;
public class Prg9
{
    public static void main(String[] args)
    {
        int a=7, b=4, c=48;
        System.out.println("a&b = " + (a&b));
        System.out.println("a|b = " + (a|b));
        System.out.println("a^b = " + (a^b));
        System.out.println("~b = " + (~b));
        System.out.println("c<<b = " + (c<<b));
        System.out.println("c>>b = " + (c>>b));
    }
}
```

Output: -  
a&b = 3  
a|b = 7  
a^b = 3  
~b = -5  
c<<b = 768  
c>>b = 3

## VII) Unary Operators: -

++ and – are called unary operators.

Increment operator ++ is used to increase the value of a variable by 1.

Decrement operator -- is used to decrease the value of a variable by 1.

### Pre-increment Operator (++): -

**If you use ++ before the variable, then it is called pre-increment operator.**

Here first the variable is increased by 1, then the remaining process will be done,

Ex: -

```
public class Prg10
{
    public static void main(String[] args)
    {
        int a=10, b;
        System.out.println("Value of a = " + ++a);
        b=++a;
        System.out.println("Value of b = " +b);
    }
}
```

Output: -

Value of a = 11

Value of a = 12

### Post-increment Operator (++): -

**If you use ++ after the variable, then it is called pre-increment operator.**

Here first the process will be performed then the variable is increased by 1.

**/\* Ex: - Write java program using post increment operator \*/**

```
public class Prg11
{
    public static void main(String[] args)
    {
        int a=10, b;
        System.out.println("Value of a = " + a++);
        b=a++;
        System.out.println("Value of b = " +b);

        If(a++ ==13)
            System.out.println("Good");
        else
            System.out.println("Excellent ");
    }
}
```

**Output: -**

**Value of a = 10**

**Value of a = 11**

**Excellent**

### **TYPE CASTING (or) Type Conversion**

It is a process of converting one datatype to another datatype.

It can be done in 2 ways:

#### **1.Implicit or widening: -**

**Lower Data Type → Higher Data Type**

It is a process of converting data from lower sized datatype to higher sized datatype.

byte ->short -> int -> long -> float -> double

Here data is directly converted. So implicit type conversion is also known as automatic type conversion.

**/\* Example: - Write java program using implicit type conversion \*/**

```
public class Prg12
{
    public static void main(String[] args)
    {
        char x='A';
```

```
int y=x;
System.out.println("\nchar value = " + x);
System.out.println("\nint value = " + y);
}
}
```

Output: -  
char value = A  
int value = 65  
Excellent

```
public class Prg13
{
    public static void main(String[] args)
    {
        int x=97;
        char y=x;
        System.out.println("\nchar value = " + x);
        System.out.println("\nint value = " + y);
    }
}
```

Output: -  
char value = a  
int value = 97  
Excellent

```
public class Prg14
{
    public static void main(String[] args)
    {
        int a=17, b=5;
        float c;
        c = (float) a/b; // implicit type casting
        System.out.println("\nDivision = " + c);
    }
}
```

Output: -  
Division = 3.4

## 2. Explicit or Narrowing: -

### Higher Data Type → Lower Data Type

It is a process of converting data from higher sized datatype to lower sized datatype.

Double -> float -> long -> int -> short-> byte

**/\* Write java program using explicit type conversion \*/**

```
public class Prg15
{
    public static void main(String[] args)
    {
        long l = 23456L;
        int i = (int)l; // explicit type casting
        System.out.println("\nlong Value = " + l);
        System.out.println("\nint Value = " + i);
    }
}
```

Output: -

long Value = 23456

int Value = 23456

## Control Statements in Java

- control statements are used to control the flow of Java code.
- control statements are classified into three categories.
- These are used to perform three kinds of tasks.
- They are branching, looping and jumping.

### 1. Decision Making statements (for Branching)

- ✓ if statement
- ✓ switch statement

### 2. Loop statements

- ✓ while loop
- ✓ do while loop
- ✓ for loop
- ✓ for-each loop

### 3. Jump statements

- ✓ break statement
- ✓ continue statement

## TYPES OF IF STATEMENTS

### 1. Simple if statement: -

If statement without "else" is known as simple if statement.

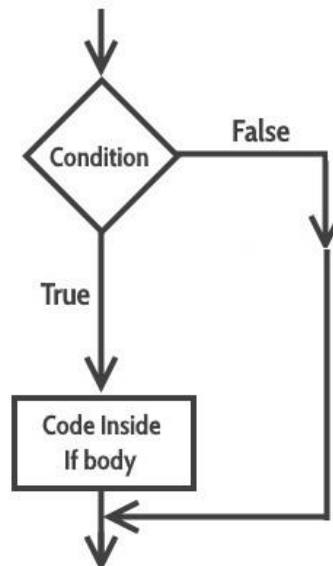
It is used to decide whether a particular code will be executed or not.

That means if certain condition is true, then the code will be executed otherwise not executed.

**Syntax: -**

```
if(condition)
{
    Statements;
}
```

**Working: -** If the condition is true then the statements within the curly braces are executed. Otherwise they are not executed.



Example: -

**/\* Write java program to convert given negative number as a positive number \*/**

```
import java.util.Scanner;
class Prg16
{
    public static void main(String args[])
    {
        int n;
        Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter a negative number ");
n = sc.nextInt();
if(n<0)
{
    n = -1 * n;
}
System.out.println("Positive Number = " + n);
}
}
```

## 2. if else statement: -

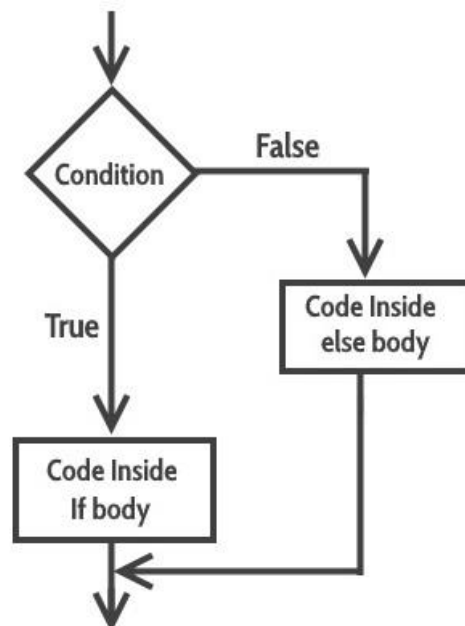
Syntax: -

```
if(condition)
{
    Statements Part - A;
}
else
{
    Statements Part - B;
}
```

Working: - If the condition is true then "Statements Part A" is executed.

If the condition is false, then "Statements Part B" is executed.

We should not use semicolon (;) after **if** or **else** keywords.





Example: -

**/\* Write java program to read two numbers and print whether they are equal or not \*/**

```
import java.util.Scanner;
class Prg17
{
    public static void main(String args[])
    {
        int a,b;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter two numbers ");
        a = sc.nextInt();
        b = sc.nextInt();
        if(a==b)
        {
            System.out.println("Given numbers are equal");
        }
        else
        {
            System.out.println("Given numbers are NOT equal");
        }
    }
}
```

### **3. Compound if statement: -**

An "if statement" with two or more conditions is known as a compound if statement. We should use logical operators (&& , ||) between the conditions.

Example: -

**/\* Write java program to check the given letter is a vowel or not \*/**

```
import java.util.Scanner;
class Prg18
{
    public static void main(String args[])
    {
        char L;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any letter ");
    }
}
```

```

L = sc.next().charAt(0);
if(L=='a' || L=='e' || L=='i' || L=='o' || L=='u')
{
    System.out.println("It is a vowel");
}
else
{
    System.out.println("It is NOT a vowel");
}
}
}

```

#### **4. else if statement: -**

It is used to execute one block of statements from several blocks of statements depending on a set of alternative conditions.

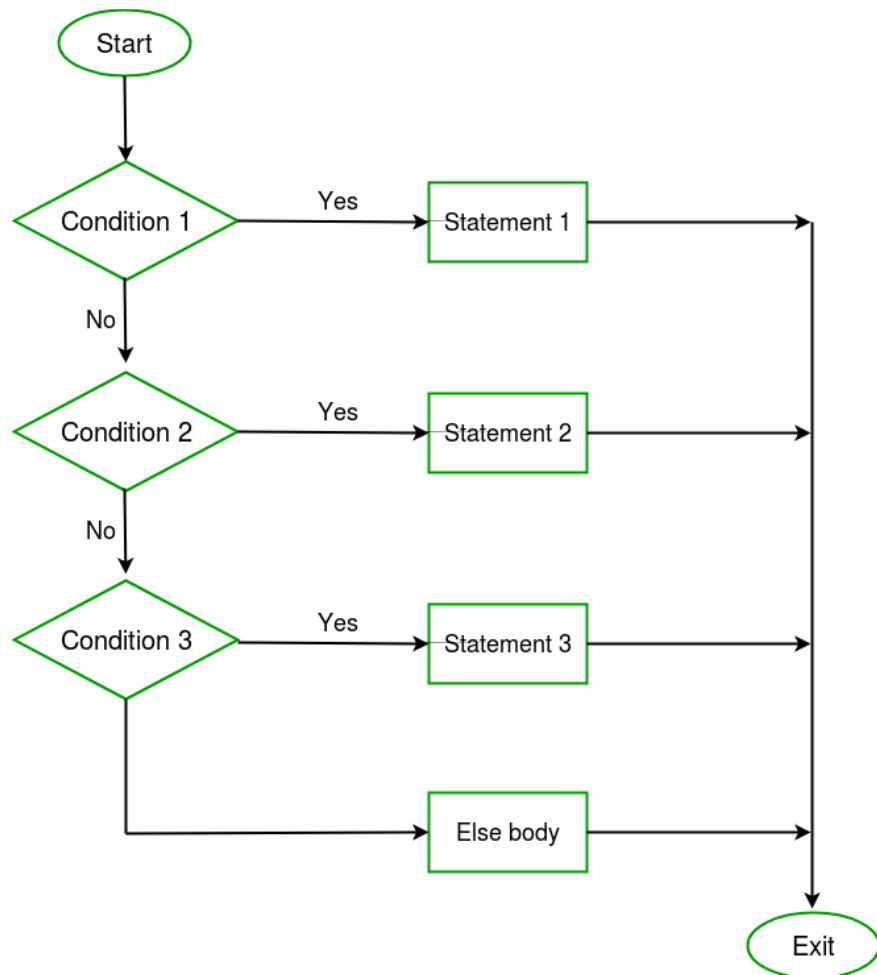
Syntax: -

```

if(condition-1)
{
    Statements Part - 1;
}
else if(condition-2)
{
    Statements Part - 2;
}
else if(condition-3)
{
    Statements Part - 3;
}
.....
.....
else if(condition-N)
{
    Statements Part - N;
}
else
{
    Statements Part - X;
}

```

Flow Chart: -



**/\* Write java program to read age of a person and print age group\*/**

```
import java.util.Scanner;
class Prg19
{
    public static void main(String args[])
    {
        int a;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter age of a person ");
        a=sc.nextInt();
        if(a>=60)
        {
            System.out.println("Old Age");
        }
        else if(a>=45)
```

```

    {
        System.out.println("Adult / Middle Age");
    }
else if(a>=20)
    {
        System.out.println("Young Age");
    }
else if(a>=13)
    {
        System.out.println("Teen Age");
    }
else
    {
        System.out.println("Child Age");
    }
}
}

```

### 5. Nested if statement: -

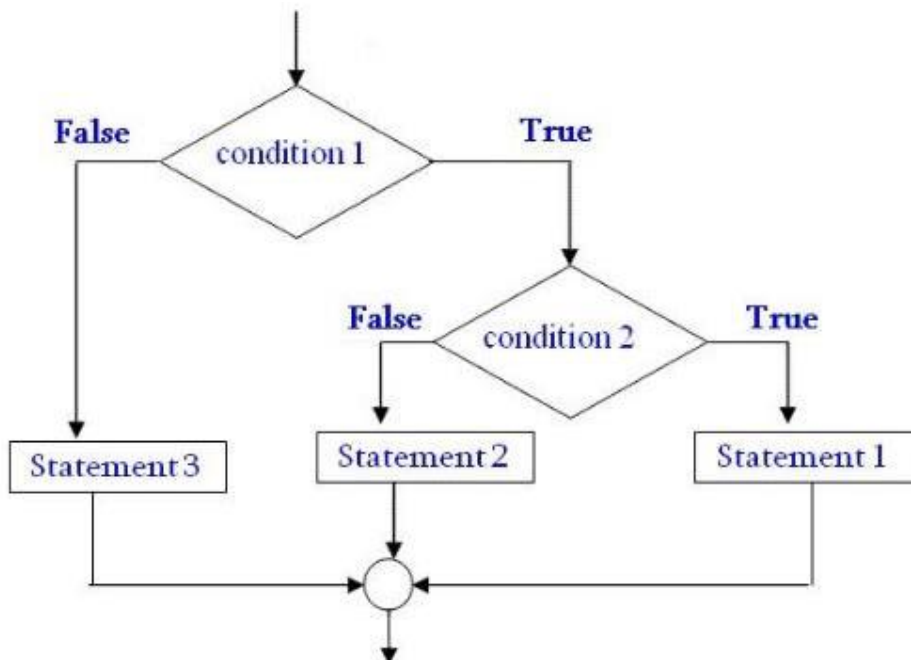
An "if statement" within another "if statement" is known as a nested if statement.

Syntax: -

```

if(condition-1)
{
    if(condition-2)
    {
        Statements Part - 1;
    }
    else
    {
        Statements Part - 2;
    }
}
else
{
    Statements Part - 3;
}

```



**/\* Write java program to read three numbers and print given numbers in ascending order \*/**

```

import java.util.Scanner;
class Prg20
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter three numbers");
        a=sc.nextInt();
        b=sc.nextInt();
        c=sc.nextInt();
        System.out.println("Given numbers in ascending order : ");
        if(a<b && a<c)
        {
            System.out.print(a + " ");
            if(b<c)
                System.out.println(b + " " + c);
            else
                System.out.println(c + " " + b);
        }
    }
}
  
```

```

else if(b<a && b<c)
{
    System.out.print(b + " ");
    if(a<c)
        System.out.println(a + " " + c);
    else
        System.out.println(c + " " + a);
}
else
{
    System.out.print(c + " ");
    if(a<b)
        System.out.println(a + " " + b);
    else
        System.out.println(b + " " + a);
}
}
}

```

### Switch Statement: -

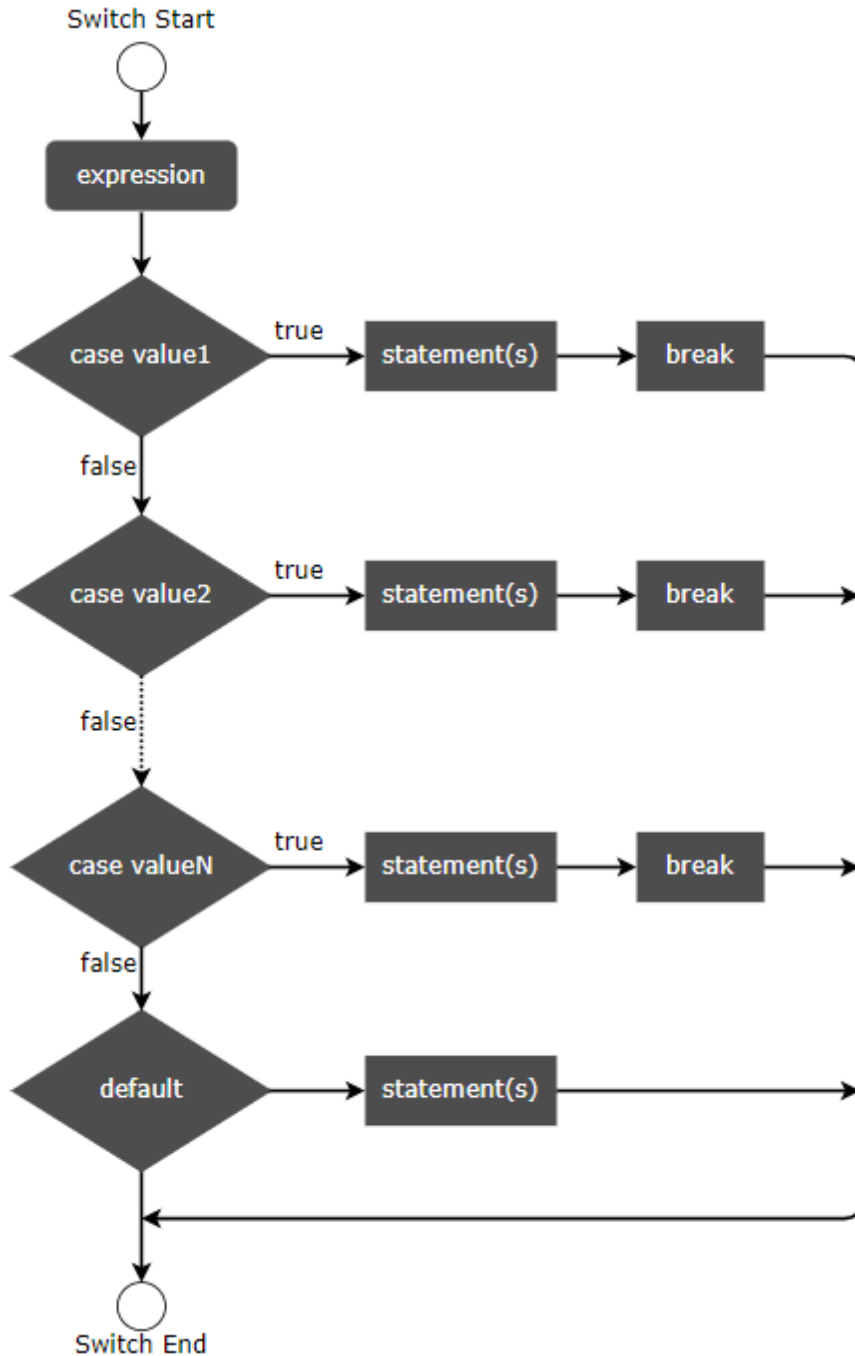
It is used to compare the value of a variable with a set of constant values in different cases sequentially for a matched one and executes the statements part of that particular case only. If all cases are mismatched, then the statements in the default section are executed.

Syntax: -

```

switch(variable/expression)
{
    case value_1:
        statements; break;
    case value_2:
        statements; break;
    case value_3:
        statements; break;
    .....
    case value_N:
        statements; break;
    default:
        statements;
}

```



**/\* Write java program to read two numbers, an arithmetic operator and perform the operation between given numbers and print the result \*/**

`import java.util.Scanner;`

`class Prg21`

`{`

`public static void main(String args[])`

`{`

```

int a, b;
char op;
Scanner sc = new Scanner(System.in);
System.out.println("Enter an arithmetic operator ");
op=sc.next().charAt(0);
System.out.println("Enter two numbers");
a=sc.nextInt();
b=sc.nextInt();
switch(op)
{
    case '+':
        System.out.println("Sum = " + (a+b)); break;
    case '-':
        System.out.println("Difference = " + (a-b)); break;
    case '*':
        System.out.println("Product = " + (a*b)); break;
    case '/':
        System.out.println("Quotient = " + (a/b)); break;
    case '%':
        System.out.println("Remainder = " + (a%b)); break;
    default:
        System.out.println("Invalid arithmetic operator");
}
}
}

```

## Loops (or) Iteration Statements

**Loop:** - A loop is used to execute one or more statements repeatedly until the condition becomes false.

There are 4 types of loops in java. They are ...

1. while loop
2. do while loop
3. for loop
4. for each loop

### while loop: -

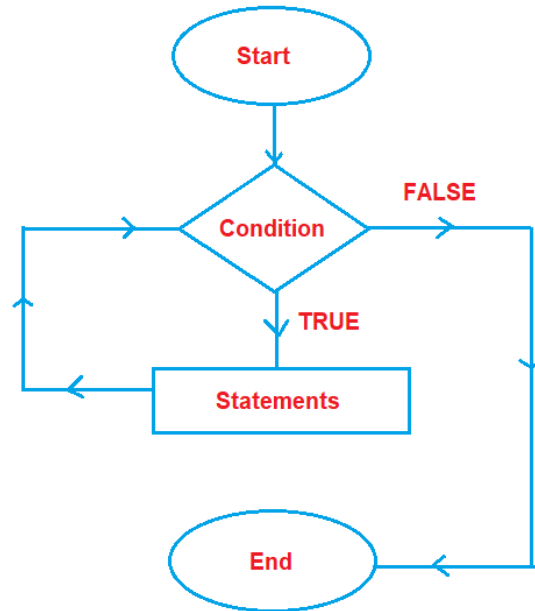
It is called pre-test loop or entry controlled loop because first it checks the loop condition. If the condition is true, the control enters into the body of the loop and executes all statements and return to the condition. Then the condition once



again verified. If it is true, then the same process is repeated. This process is repeated as long as the condition remains true. If condition becomes false, then the loop is stopped.

Syntax: -

```
while(condition)
{
    body of the loop;
}
```



**/\* Java program to print the place values of digits of given number \*/**

```
import java.util.*;
public class Prg22
{
    public static void main(String[] args)
    {
        int n,p=1;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number ");
        n = sc.nextInt();
        while(n>0)
        {
            System.out.println("Place value of " + (n%10) + " is " + p);
            p=p*10;
            n=n/10;
        }
    }
}
```

```
}  
}
```

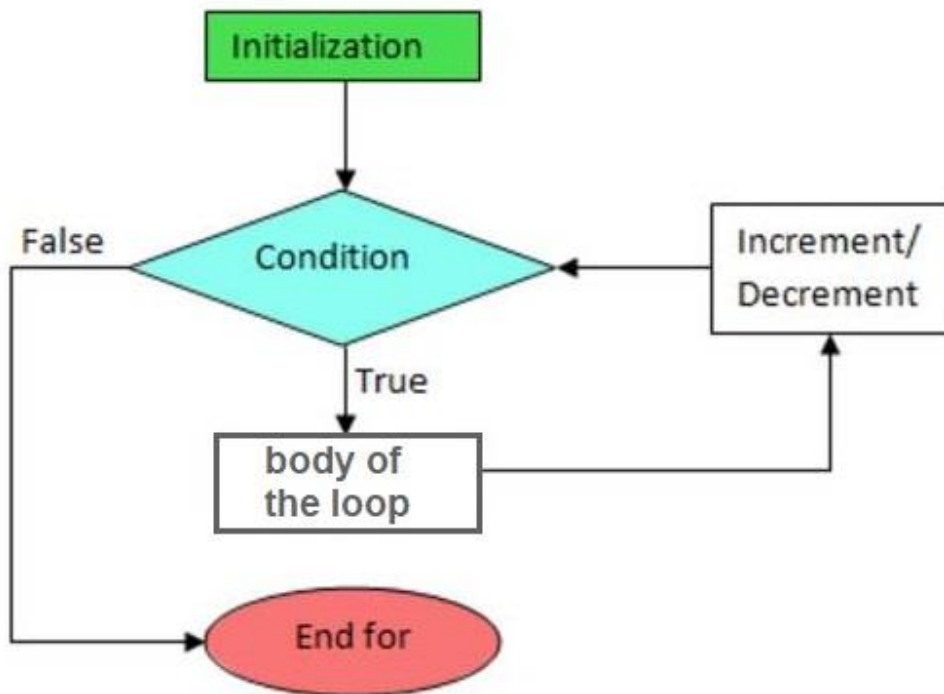
### for loop: -

It is used to execute the body of the loop for a fixed number of times.

#### **Syntax: -**

```
for(initialization ; condition ; increment/decrement)  
{  
    body of the loop;  
}
```

Flowchart: -



Example1: -

```
int a;  
for(a=1 ; a<10 ; a++)  
{  
    printf("%d\t" , a);  
}
```

Output: 1 2 3 4 5 6 7 8 9

Example2: -

```
for(int a=9 ; a>0 ; a--)  
{  
    printf("%d\t" , a);  
}
```

Output: 9 8 7 6 5 4 3 2 1

**Working of for loop:** - Whenever for loop found first the loop variable is initialized with the initial value. Then the condition is verified. If condition is true, the control enter into the body of the loop and executes all statements and return to increment/decrement part. Then the value of loop variable is updated. The new value of loop variable is again verified by the condition. If it is true, the control once again enters into the body of the loop and the same process is repeated. This process is repeated as long as the condition remains true. If condition becomes false, then the loop is terminated. In a for loop the one and only section which is executed only once is "initialization" section. For loop is considered as pre-test loop or entry controlled loop.

**/\* Write java program to print Nth table \*/**

```
import java.util.*;  
public class Prg23  
{  
    public static void main(String[] args)  
    {  
        int n,i;  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter N value ");  
        n = sc.nextInt();  
        for(i=1 ; i<=20 ; i++)  
        {  
            System.out.println(n + " X " + i + " = " + n*i);  
        }  
    }  
}
```

**do while loop: -**

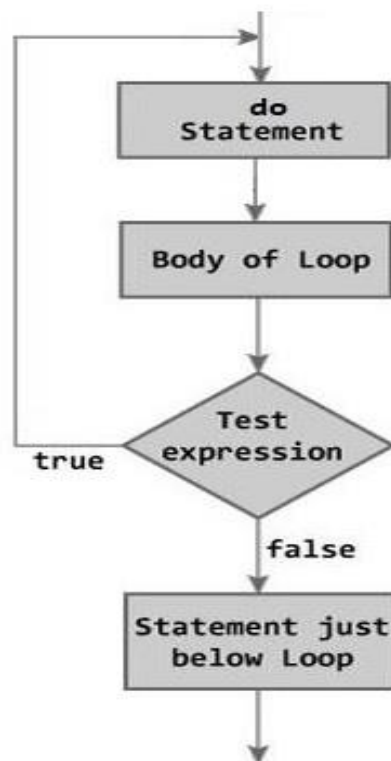
It is called post-test loop or exit controlled loop because it checks the loop condition after executing the body of the loop. Do while loop is used to execute

the body of the loop at least once irrespective of the loop condition. That means "do while loop" never becomes a NULL loop.

Syntax: -

```
do
{
    body of the loop;
}
while(condition);
```

Flowchart: -



**/\* Write java program to read given numbers upto zero is given \*/**

```
import java.util.*;
public class Prg24
{
    public static void main(String[] args)
    {
        int n;
        Scanner sc = new Scanner(System.in);
        do
        {
```

```
System.out.println("Enter N value ");
n = sc.nextInt();
} while(n != 0);
System.out.println("Zero is given");
}
}
```

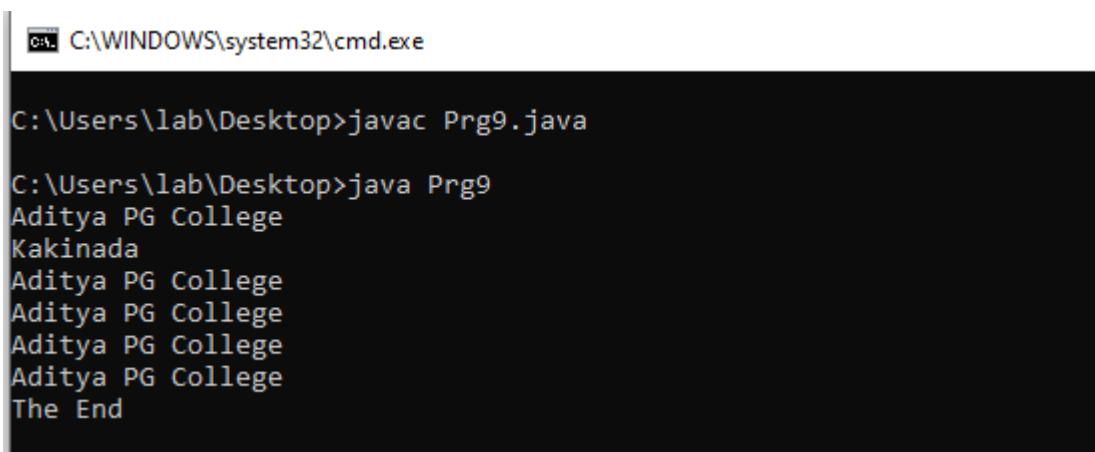
## JUMP Statements in Java

There are two types of jump statements in java. They are continue and break.

**1. continue statement:** - It is used to continue the next iteration of the loop without executing its following statements.

**/\* Write java program to explain the working of continue statement \*/**

```
import java.util.*;
public class Prg25
{
    public static void main(String[] args)
    {
        int i;
        for(i=1 ; i<=5 ; i++)
        {
            System.out.println("Aditya PG College");
            if(i>1)
                continue;
            System.out.println("Kakinada");
        }
        System.out.println("The End");
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\Users\lab\Desktop>javac Prg9.java
C:\Users\lab\Desktop>java Prg9
Aditya PG College
Kakinada
Aditya PG College
Aditya PG College
Aditya PG College
Aditya PG College
The End
```

**2. break statement:** - It is used to immediately exit from a loop. Similarly, it is also used to immediately quit from a case in switch statement.

**/\* Write java program to explain the working of break statement \*/**

```
import java.util.*;
public class Prg26
{
    public static void main(String[] args)
    {
        int i;
        for(i=1 ; i<=5 ; i++)
        {
            System.out.println("Aditya PG College");
            if(i>1)
                break;
            System.out.println("Kakinada");
        }
        System.out.println("The End");
    }
}
```

```
C:\WINDOWS\system32\cmd.exe

C:\Users\lab\Desktop>javac Prg9.java

C:\Users\lab\Desktop>java Prg9
Aditya PG College
Kakinada
Aditya PG College
The End
```

### Nested switch statement

If you write one or more switch statements within the body of another switch statement, then the outer switch statement is called nested switch statement. It is used to choose one option from several options inside a particular case of a switch statement.

**Syntax: -**

```
switch(expression1) → Outer switch statement
{
    case value1:
        statements;
        switch(expression2) → Inner switch statement
        {
            case value:
                statements; break;
            case value:
                statements; break;
            .....
        }
        break;
    case value2:
        statements; break;
    case value3:
        statements; break;
    .....
    default:
        statements;
}
```

**Example: -**

**/\* Write java program to read course code and branch code of a student and print fee details using the following data \*/**

Course	Branch	Fee
Inter	MEC	35000
	MPC	40000
	Bi.PC	45000
Degree	BBA	45000
	BCA	50000
	B.Sc	55000
PG	MBA	65000
	MCA	70000

```
import java.util.*;
public class Prg27
{
    public static void main(String[] args)
    {
        int bc,cc;
        Scanner sc = new Scanner(System.in);
        System.out.println("101) Inter\n201) Degree\n301) PG");
        System.out.println("\n Select your course code ");
        cc=sc.nextInt();
        switch(cc)
        {
            case 101:
                System.out.println("1. MEC\n2. MPC\n3. Bi.PC");
                System.out.println("\n Select your course code ");
                bc=sc.nextInt();
                switch(bc)
                {
                    case 1: System.out.println("MEC yearly fee = 35000"); break;
                    case 2: System.out.println("MPC yearly fee = 40000"); break;
                    case 3: System.out.println("Bi.PC yearly fee= 45000"); break;
                    default: System.out.println("Invalid branch code");
                }
                break;
            case 201:
                System.out.println("1. BBA\n2. BCA\n3. B.Sc");
                System.out.println("\n Select your course code ");
                bc=sc.nextInt();
                switch(bc)
                {
                    case 1: System.out.println("BBA yearly fee = 45000"); break;
                    case 2: System.out.println("BCA yearly fee = 50000"); break;
                    case 3: System.out.println("B.Sc yearly fee = 55000"); break;
                    default: System.out.println("Invalid branch code");
                }
                break;
            case 301:
                System.out.println("1. MBA\n2. MCA\n3. M.Sc");
                System.out.println("\n Select your course code ");
                bc=sc.nextInt();
                switch(bc)
                {
                    case 1: System.out.println("MBA yearly fee = 65000"); break;
                    case 2: System.out.println("MCA yearly fee = 70000"); break;
                    case 3: System.out.println("M.Sc yearly fee = 75000"); break;
                    default: System.out.println("Invalid branch code");
                }
                break;
            default:
                System.out.println("Invalid course code");
        }
    }
}
```



```
}  
}  
}
```

## NESTED FOR LOOP

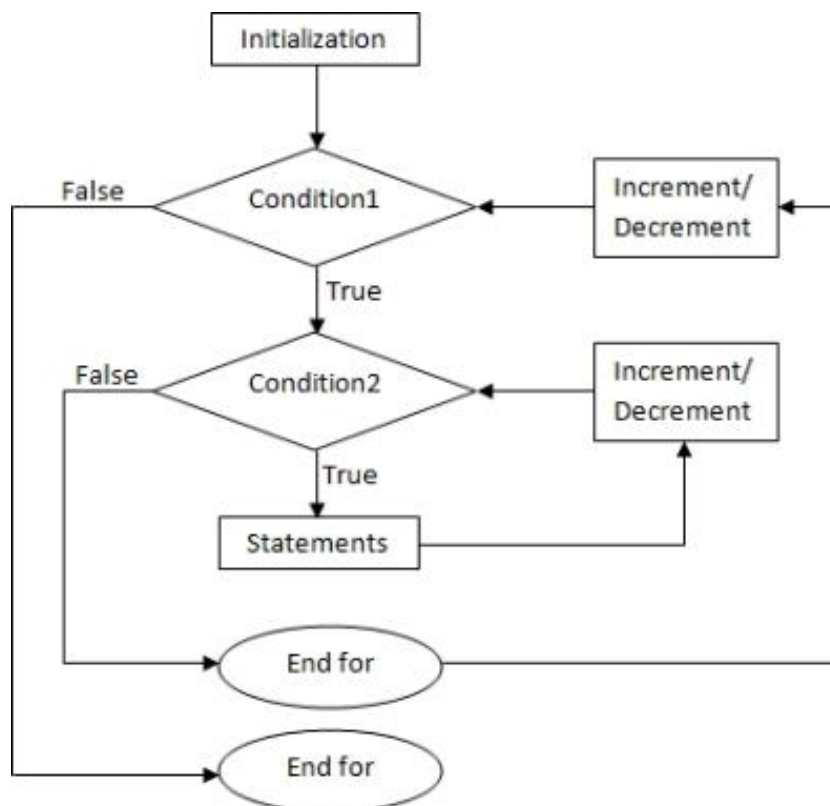
If a loop exists inside the body of another loop, then it is called a nested loop. Similarly, if you write one or more for loops within the body of another for loop then it is called a nested for loop.

### Syntax: -

```
for(initialization ; condition1 ; updation) → Outer for loop  
{  
    for(initialization ; condition2 ; updation) → Inner for loop  
    {  
        body of the nested for loop;  
    }  
}
```

**Working of nested for loop: -** For each value of outer for loop, the inner for loop is executed from the initial value to final value. That means if outer for loop is repeated for M times and inner for loop is repeated for N times then the statements in the nested for loop are executed MXN times. If the condition of outer for loop is false, then the execution of nested for loop is stopped.

### Flowchart: -



**/\* Write C program to print 4 weeks each week with 7 days \*/**

```
import java.util.*;  
public class Prg9  
{  
    public static void main(String[] args)  
    {  
        int i, j;  
        for(i=1 ; i<=4 ; i++)
```

```

{
    System.out.println("\nWeek: " + i);
    for(j=1; j<=7; j++)
    {
        System.out.print("\tDay: " + j);
    }
}
System.out.print("\nThe End");
}
}

```

## Features of Java

### **1. Simple:**

Because few concepts and syntax is based on 'C' language.

### **2. Object-Oriented:**

Java provides many OOPs concepts like data abstraction, data hiding, data encapsulation, data binding, polymorphism, inheritance, etc.

### **3. Platform Independent:**

Platform means software or hardware environment. Java bytecode is a platform independent code because it can be executed by the JRE rather than the OS.

### **4. Secured:**

No pointers concept in Java.

programs are executed in JVM (Java Virtual Machine)

### **5. Robust:**

Java uses very strong memory management and Java makes an effort to eliminate error-prone situations.

### **6. Architecture Neutral:**

It provides 8 data types with fixed size. So there is no implementation dependent features for both 32 and 64-bit architectures.

### **7. Portable:**

java files can be carried from one location to other location.

### **8. High Performance:**

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

### **9. Multithreaded:**

Thread is a part of program. Here, we can execute multiple threads at a time.

### **10. Distributed:**

In java we can create distributed applications on internet.

## Difference between JDK, JRE, and JVM

**JRE (Java Runtime Environment)** is a set of software tools consisting of set of libraries + other files that JVM uses at runtime.

**JVM (Java Virtual Machine)** is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is simply a specification that provides a runtime environment in which Java bytecode can be executed.

**JDK (Java Development Kit)** is a software development environment in which Java applications and applets are developed. It physically exists. It contains JRE + development tools. That means JDK contains JRE, an interpreter/loader, a compiler, an archiver (jar), a documentation generator (Javadoc) to accomplish the development of a java applications. JDK is an implementation of Oracle corporation. Oracle provides three types of platforms for JDK. They are J2SE, J2EE and J2ME.

## Java Tokens

These are building blocks of java program.

## **1. Identifiers:**

These are name of variables, methods, classes, etc.  
These can contain letters, numbers and underscores.  
It should not contain white spaces.  
It should start with character.  
Java is case sensitive.  
These cannot be keywords.

## **2. Keywords:**

These are reserve words which should be used as it is in a program like int, float, if, else, do, for, break, continue, class, etc.

## **3. Literals:**

These are constant values used in a program.

## **4. Comments:**

These are non-executable statements.

// single line comment

/\* ..... \*/ multiline comment

\*\* ..... \*/ documentation comment, every line should have \*

## **5. Datatypes:**

These are used to specify the type of data stored in a variable, along with memory allocation.

## **6. Operators:**

These are used to perform calculations.

1. Arithmetic Operators
2. Unary Operators
3. Assignment operator
4. Compound Assignment operator:
5. Relational operators
6. Logical operators
7. Bit wise operators
8. Ternary operator
9. Dot membership operator
10. *instanceof* operator.

## **Primitive Data Types**

In Java all variables must first be declared before they can be used. A primitive type is predefined by the language and is named by a keyword or reserved keyword. The eight primitive data types supported by the Java programming language are listed below.

**1. byte:** The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).

**2. short:** The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767

**3. int:** The int data type is a 32-bit signed two's complement integer. It has a min value of -2,147,483,648 and a max value of 2,147,483,647

**4. long:** The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807

**5. float:** The float data type is a single-precision 32-bit IEEE 754 floating point. Use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency.

**6. double:** The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

**7. boolean:** The boolean data type has only two possible values. They are true and false. This is used for simple flags that track true/false conditions.

**8. char:** The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (0) and a maximum value of '\uffff' (65,535)

### **Difference Between C and JAVA**

<b>C</b>	<b>JAVA</b>
1. C is a structured programming language.	1. Java is pure object oriented programming language.
2. C allows us to divide a problem in to a no. of functions.	2. Java allows us to divide a problem into no. of objects.
3. C uses header files using pre-processor directions. #include <stdio.h>.	3. Java does not use header files and pre-processor directions.
4. C uses pointers .	4. Java does not use pointers.
5. C uses goto, sizeof, typedef keywords.	5. Java does not use goto, sizeof, typedef keywords.
6. C uses structure and unions.	6. Java use classes and interfaces in place of structure and in unions.
7. In c data has no security.	7. In java data has security.
8. C does not have Inheritance and polymorphism	8. Java has more inheritance and polymorphism.
9. C follows top-down approach.	9. Java follows bottom-up approach.
10. C is partially platform independent language.	10. Java is completely platform independent language.

### **What is Object Oriented Programming? What are the characteristics and features of Object Oriented Programming language?**

The Object oriented programming is one of the newest and most powerful paradigms. The Object-Oriented Programming mentions to the programming methodology based on the objects in place of procedures and functions.

These objects are planned into classes. Modern programming languages like C++, Python, java, PHP are object-oriented languages.

The object is an instance or specific type, of the class. Each and every object has a structure related to other objects in the class, but it can be allocated individual features. The Object oriented programming makes it easier to the programmers to design and organize software programs. The Object oriented programming has ten basic principles. They are...

- ✓ Encapsulation
- ✓ Data Abstraction
- ✓ Inheritance
- ✓ Polymorphism

- ✓ Extensibility
- ✓ Persistence
- ✓ Generality
- ✓ Object Concurrency
- ✓ Event Handling
- ✓ Message Passing

According to the Object Oriented programming, Java allows us to working with classes and objects. For implementation of OOPs we have to follow the following four fundamental concepts.

- 1. Encapsulation** – Hide unnecessary details in classes and deliver a simple and clear interface for working.
- 2. Inheritance** – It explains how the class hierarchies develop code readability and support the reusability of functionality.
- 3. Polymorphism** – It explains how to work with different object in the same manner, which explain the specific implementation of some abstract behavior.
- 4. Data abstraction** - Abstraction is a concept which facilitates to extract the essential information of an object.

### **What is String? Explain String handling functions in Java?**

#### **Strings in Java**

String is basically an object that represents sequence of char values. An array of characters works same as string.

For example:

```
char[] ch={'j','a','v','a','l','a','n','g','u','a','g','e'};
String s=new String(ch);
```

It is same as

```
String s = "javalanguage";
```

**String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Note1: - The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

The *CharSequence* interface is used to represent the sequence of characters.

String, [StringBuffer](#) and [StringBuilder](#) classes implement it. It means, we can create strings in Java by using these three classes.

Note2: - String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use [StringBuffer](#) and [StringBuilder](#) classes.

There are two ways to create String object:

1. By String literal
2. By new keyword

Java String literal is created by using double quotes.

For Example, String s="welcome";

#### **String Methods in Java: -**

1. **length():** It returns the length of the string (No.of characters in the given string)
2. **charAt(Index):** It returns the particular character at the specified Index.

3. **Contains("Sub-String")**: It returns true or false after matching the sequence of char value.
4. **equals()**: It returns true if the strings are exactly same.
5. **equalsIgnoreCase()** : It compares the strings on the basis of the content irrespective of the case.
6. **isEmpty()**: It returns true if the string is empty.
7. **toLowerCase()**: It convert all letters of string into lower case.
8. **toUpperCase()**: It convert all letters of string into upper case.
9. **trim()** : It removes beginning and ending spaces in the given string.
10. **replaceAll(old\_char, new\_char)**: It replaces all occurrences of the old\_char with new\_char.
11. **substring(begin\_index, end\_index)**: It returns substring between begin index and end index.
12. **split(char)**: It split the string into several sub-strings by using the specified character as a separator.

**/\* Write Java program to Count number of words in the given text \*/**

```
import java.util.Scanner;
class No_Of_Words
{
    public static void main(String args[])
    {
        Scanner sc=new
        Scanner(System.in); String s1;
        System.out.println("Enter a line of text ");
        s1=sc.nextLine();
        String w[]=s1.split(" ");
        int n=w.length;
        System.out.println("The words in the given text: -");
        for(int i=0; i<n ; i++)
            System.out.println(w[i]);
        System.out.println("Number of words = " + n);
    }
}
```

**/\* Write Java program to check the given string is palindrome or not \*/**

```
import java.io.*;
import java.util.*; class
StringReverse
{
    public static void main(String args[])
    {
        String s1;
        String s2="";
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a string ");
        s1=sc.next();
        for(int i=s1.length()-1 ; i>=0 ; i--)
            s2=s2+s1.charAt(i); System.out.println("Given
        String = "+s1); System.out.println("Reverse
        String = "+s2); if(s1.equals(s2))
            System.out.println(s1 + " is palindrome"); else
            System.out.println(s1 + " is NOT palindrome");
    }
}
```