

## Unit—I-PART-IV

**Introduction to JAVA:** History of Java, Java buzzwords, data types, variables, scope and life time of variables, Type conversion and casting, arrays, operators, Operator Precedence, control statements.

### **Introduction:**

What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

Mobile applications (specially Android apps)

Desktop applications

Web applications

Web servers and application servers

Games

Database connection

And much, much more!

Why Use Java?

Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)

It is one of the most popular programming language in the world

It has a large demand in the current job market

It is easy to learn and simple to use

It is open-source and free

It is secure, fast and powerful

It has a huge community support (tens of millions of developers)

Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs

### **History of java:-**

Java is a computer programming language.

Java was created based on C and C++.

Java uses C syntax and many of the object-oriented features are taken from C++.

Before Java was invented there were other languages like COBOL, FORTRAN, C, C++, Small Talk, etc.

These languages had few disadvantages which were corrected in Java. Java was invented by a team of 13 employees of Sun Microsystems, Inc. which is lead by James Gosling, in 1991.

The team includes persons like Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan, etc., Java was developed as a part of the Green project. Initially, it was called Oak, later it was changed to Java in 1995.



**James Gosling**

Creator of Java

Born in Alberta, Canada on 19<sup>th</sup> May 1955.

- [www.btechsmarclass.com](http://www.btechsmarclass.com)

The C language developed in 1972 by Dennis Ritchie had taken a decade to become the most popular language.

In 1979, Bjarne Stroustrup developed C++, an enhancement to the C language with included OOP fundamentals and features.

A project named “Green” was initiated in December of 1990, whose aim was to create a programming tool that could render obsolete the C and C++ programming languages.

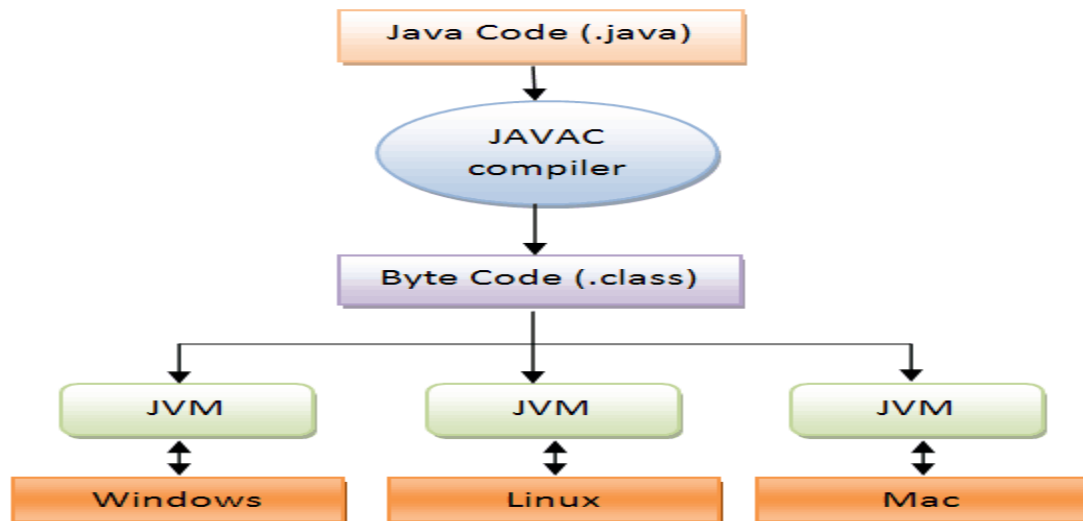
Finally in the year of 1991 the Green Team was created a new Programming language named “OAK”.

After some time they found that there is already a programming language with the name “OAK”.

So, the green team had a meeting to choose a new name. After so many discussions they want to have a coffee. They went to a Coffee Shop which is just outside of the Gosling’s office and there they have decided name as “JAVA”.

## JVM

- JVM means Java virtual Machine.
- It is a set of software and program components.
- As the name suggests, it is a virtual computer that resides in the real computer.
- Java can achieve its platform independence feature due to JVM.
- JVM creation and execution of java program.



When we want to write any java program ,we write the source code and store it in a file with extension .java

This .java file is compiled using javac and a class file gets created.

This class file is actually a byte code.

The name byte code is given because of the instruction set of java program.

The Java virtual Machine takes byte code as input, reads it, interprets it and then executes it.

The java virtual Machine can generate an output corresponding to the underlying platform(operating system).

This allows any java program to execute on any platform.

Hence java is known as platform independent language.

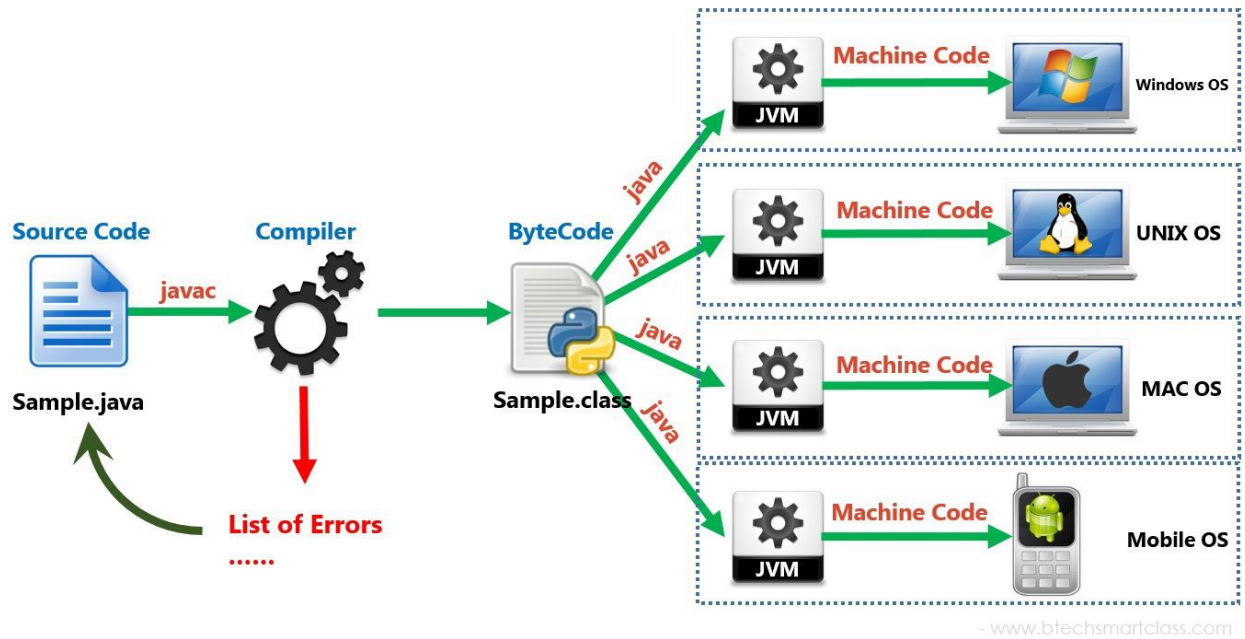
### Execution Process of Java Program

The following three steps are used to create and execute a java program.

Create a source code (.java file).

Compile the source code using javac command.

Run or execute .class file using java command.



## Java buzzwords or java Features:

Java is the most popular object-oriented programming language.

Java has many advanced features, a list of key features is known as Java Buzz Words. The java team has listed the following terms as java buzz words.

Simple

Secure

Portable

Object-oriented

Robust

Architecture-neutral (or) Platform Independent

Multi-threaded

Interpreted

High performance

Distributed

Dynamic

### **Simple:**

Java programming language is very simple and easy to learn, understand, and code.

Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++.

### **Secure:**

java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

### **Portable:**

Portability is one of the core features of java which enables the java programs to run on any computer or operating system.

### **Object-Oriented:**

Java is said to be a pure object-oriented programming language.

In java, everything is an object. It supports all the features of the object-oriented programming paradigm.

### **Robust:**

Java is more robust because the java code can be executed on a variety of environments.

java has a strong memory management mechanism (garbage collector).

### **Architecture-neutral (or) Platform Independent:**

Java has invented to archive "write once; run anywhere, any time, forever".

The JVM allows the java program created using one operating system can be executed on any other operating system.

### **Multi-threaded:**

Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

### **Interpreted:**

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode.

The byte code is interpreted to any machine code so that it runs on the native machine.

### **High performance:**

Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

### **Distributed:**

Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet.

### **Dynamic:**

Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

## Java Data Types:

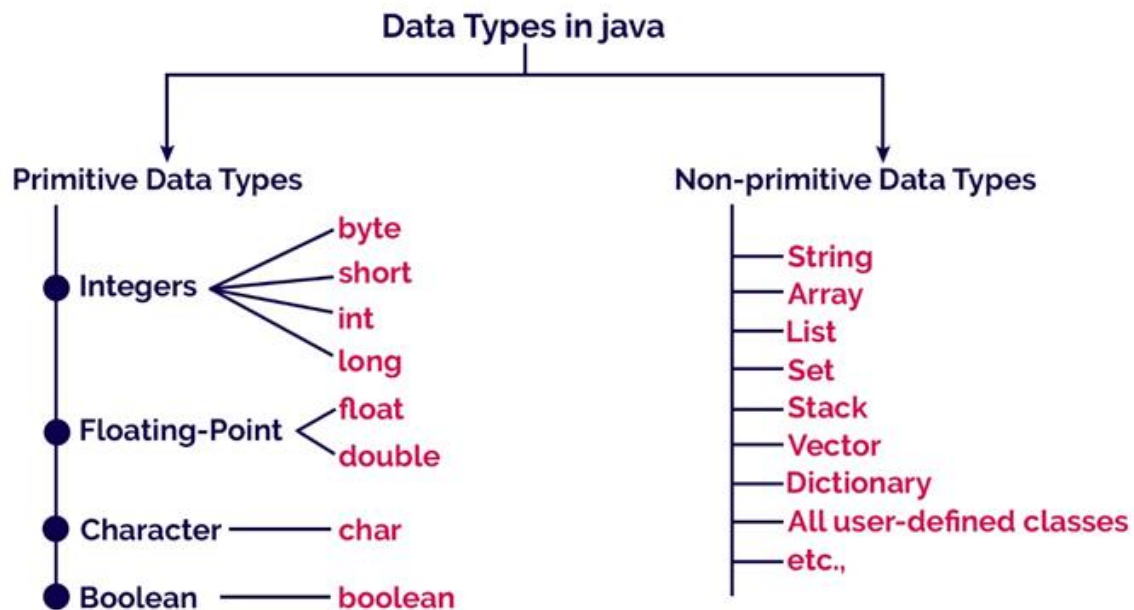
Java programming language has a rich set of data types.

The data type is a category of data stored in variables.

In java, data types are classified into two types and they are as follows.

Primitive Data Types

Non-primitive Data Types



## Primitive Data Types:

The primitive data types are built-in data types and they specify the type of value stored in a variable and the memory size.

In java, primitive data types includes **byte**, **short**, **int**, **long**, **float**, **double**, **char**, and **boolean**.

The following table provides more description of each primitive data type.

Data type	Meaning	Memory size	Range
byte	Whole numbers	1 byte	-128 to +127
short	Whole numbers	2 bytes	-32768 to +32767
int	Whole numbers	4 bytes	-2,147,483,648 to +2,147,483,647
long	Whole numbers	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

Data type	Meaning	Memory size	Range
float	Fractional numbers	4 bytes	-
double	Fractional numbers	8 bytes	-
char	Single character	2 bytes	0 to 65535
boolean	unsigned char	1 bit	0 or 1

### Non-primitive Data Types:

In java, non-primitive data types are the reference data types or user-created data types.

All non-primitive data types are implemented using object concepts.

Every variable of the non-primitive data type is an object.

The default value of non-primitive data type variable is null.

In java, examples of non-primitive data types

are **String**, **Array**, **List**, **Queue**, **Stack**, **Class**, **Interface**, etc.

#### Byte:-

byte is a keyword in java.

It occupies 1 byte of space in the memory.

The range of byte is -128 to 127.

#### Short:-

Short is a keyword in java.

It is also whole number.

It occupies 2 bytes of space in the memory.

The range of short is -32768 to +32767

### Java Variables:

A variable is a named memory location used to store a data value. A variable can be defined as a container that holds a data value.

In java, we use the following syntax to create variables.

#### **Syntax**

data\_type variable\_name;

(or)

data\_type variable\_name\_1, variable\_name\_2,...;

(or)

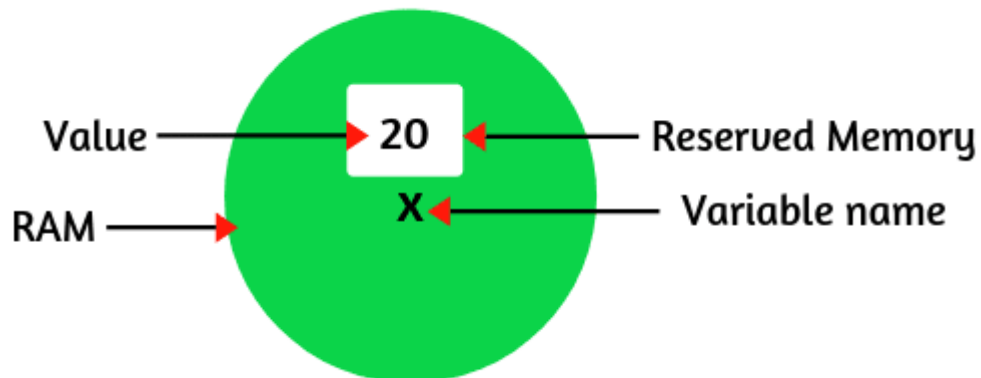


data\_type variable\_name = value;

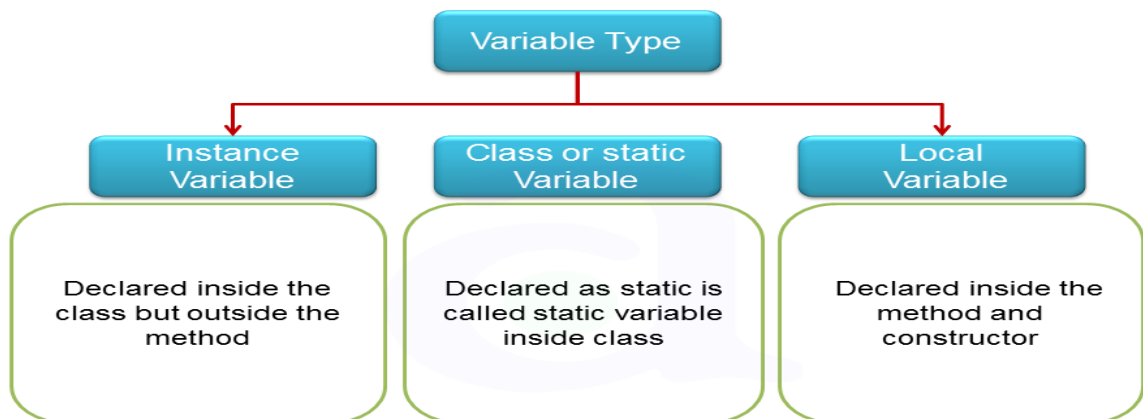
(or)

data\_type variable\_name\_1 = value, variable\_name\_2 = value,...;

Data type → **int x=20;**



### A Memory Representation of Variable



```
class ClassName {  
    int count = 100; ← Instance variable  
    static int a = 0; ← Static variable  
    void method() {  
        int b = 77; ← local variable  
    }  
}
```




### Local variables:

The variables declared inside a method or a block are known as local variables. A local variable is visible within the method in which it is declared.

Example

```
void method() {  
    int b = 77;  
}
```



### Instance variables or member variables or global variables:

The variables declared inside a class and outside any method, constructor or block are known as instance variables or member variables.

These variables are visible to all the methods of the class.

Ex:-

```
class ClassName {  
    int count = 100;  
}
```



### Static or class variable:-

A static variable is a variable that declared using **static** keyword.

The instance variables can be static variables but local variables can not.

Static variables are initialized only once, at the start of the program execution.

Example

```
static int a = 0;
```



Ex:-

```
public class A  
{  
    static int m=100;//static variable  
    void method()  
    {  
        int n=90;//local variable  
    }  
    public static void main(String args[])  
    {
```

```

    int data=50;//instance variable
}
} //end of class

```

### Scope and life time of variables :-

Scope of a variable determines the area or a region of code where a variable is available to use.

In Java, the scope of such a variable is from its declaration to the end of the method.

### Lifetime of a variable:

The lifetime of a variable is the time period in which the variable has valid memory.

In Java, the lifetime of a variable is the period of time beginning when the method is entered and ending when execution of the method terminates.

### Instance Variables:

A variable declared inside the class but outside the body of the method, is called an instance variable.

**Instance variables** are defined as outside of any method declaration

It is not declared as static.

General **scope of an instance variable** is throughout the class except in static methods.

**Lifetime of an instance variable** is until the object stays in memory.

```

class Sample
{

```

```

    int x, y; //instance variables
    static int result;
    void add(int a, int b) //a and b are local variables
    {
        x = a;
        y = b;
        int sum = x+y; //Sum
        System.out.println("Sum = "+sum);
    }

```

Scope of  
x and y

```

    public static void main(String[] args)
    {
        Sample obj = new Sample();
        obj.add(10,20);
    }

```

```

}

```

Act

### Class Variables or static variables:

A variable that is declared as static is called a static variable or class variable. It cannot be local.

You can create a single copy of the static variable and share it among all the instances of the class.

General **scope of a class variable** is throughout the class and the **lifetime of a class variable** is until the end of the program or as long as the class is loaded in memory.

```
class Sample
{
    int x, y;
    static int result; //Class variable
    void add(int a, int b)
    {
        x = a;
        y = b;
        int sum = x+y;
        System.out.println("Sum = "+sum);
    }
    public static void main(String[] args)
    {
        Sample obj = new Sample();
        obj.add(10,20);
    }
}
```

Scope of result

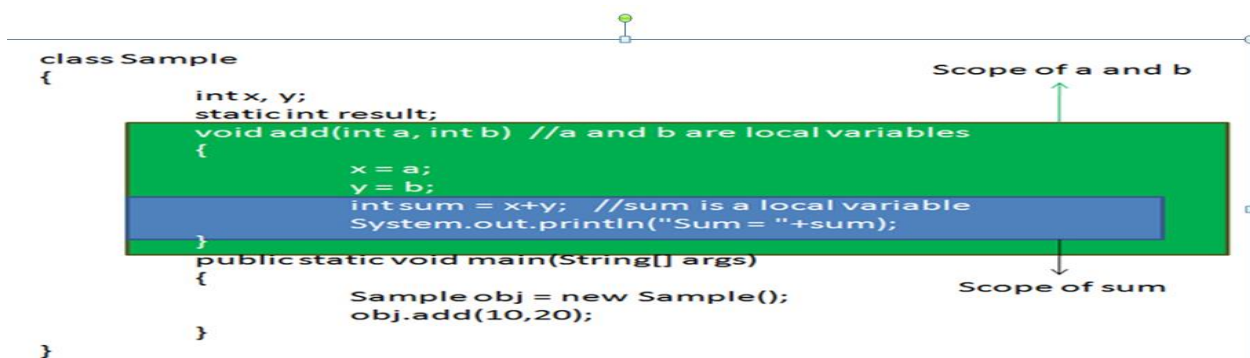
### Local Variables:

A variable declared inside the body of the method is called local variable.

We can use this variable only within that method.

A local variable cannot be defined with "static" keyword.

**Scope of a local variable** is within the block in which it is declared and the **lifetime of a local variable** is until the control leaves the block in which it is declared.

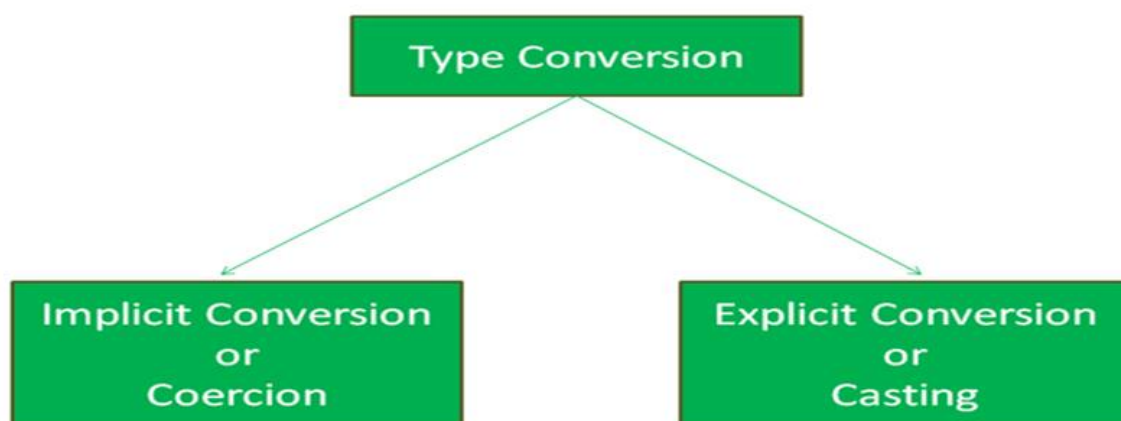


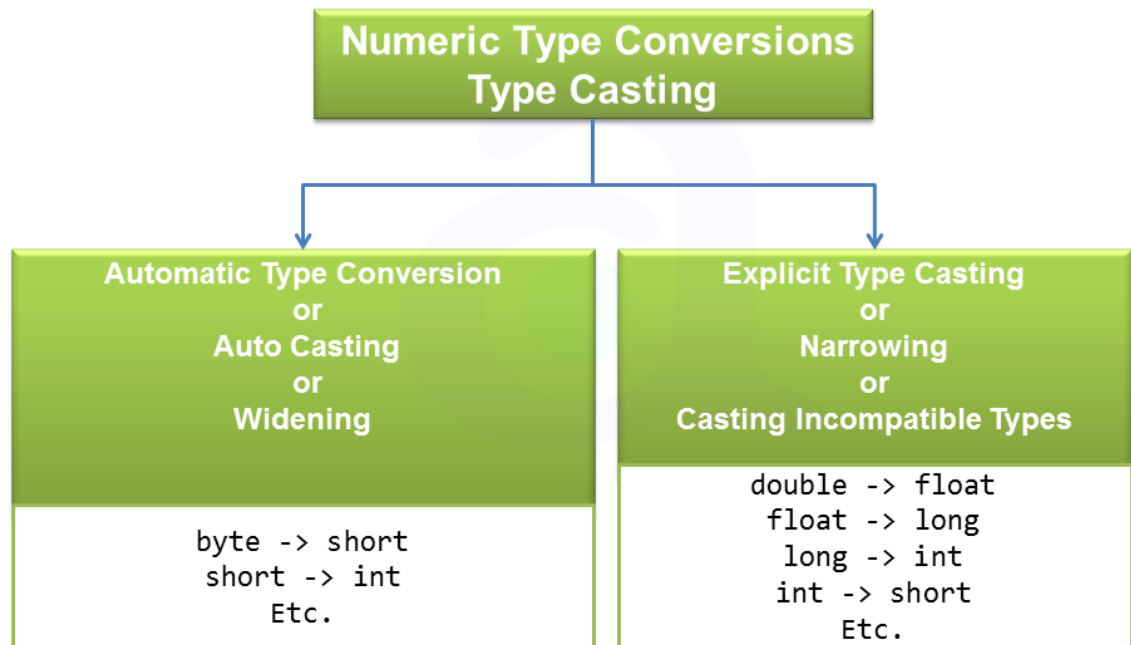
### Type conversion or type casting:-

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically.

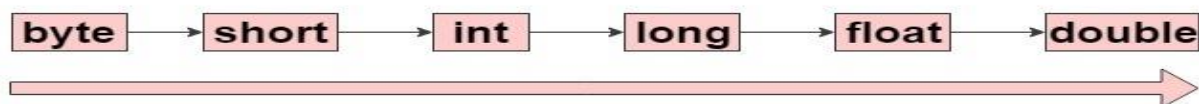
The automatic conversion is done by the compiler and manual conversion performed by the programmer.

Convert a value from one data type to another data type is known as **type casting**.

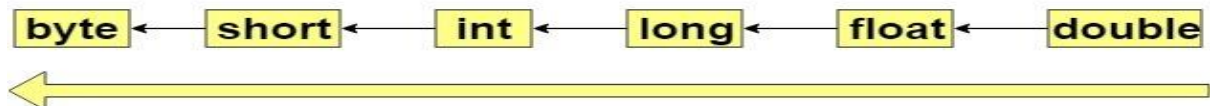




## Automatic Type Conversion (Widening - implicit)



## Narrowing (explicit)



### Widening Type Casting:

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically.

```

WideningTypeCastingExample.java
public class WideningTypeCastingExample
{
    public static void main(String[] args)
    {
        int x = 7;
        //automatically converts the integer type into long type
        long y = x;
        System.out.println("Before conversion, int value "+x);
    }
}
  
```

```

System.out.println("After conversion, long value "+y);
}
}

```

### Narrowing Type Casting:

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer.

Ex:-

NarrowingTypeCastingExample.java

```

public class NarrowingTypeCastingExample
{
    public static void main(String args[])
    {
        double d = 166.66;
        //converting double data type into long data type
        int f = (int)d;
        //converting long data type into int data type
        System.out.println("Before conversion: "+d);
        //fractional part lost
        System.out.println("After conversion of double type: "+f);
    }
}

```

## Java Arrays

An array is a collection of similar data values with a single name.

An array can also be defined as, a special type of variable that holds multiple values of the same data type at a time.

An array refers to a data structure that contains homogeneous elements. This means that all the elements in the array are of the same data type. Let's take an example:

40	55	23	87	21	11	94	Elements
0	1	2	3	4	5	6	Index

This is an array of seven elements. All the elements are integers and homogeneous. The green box below the array is called the index, which always starts from zero and goes up to n-1 elements. In this case, as there are seven



elements, the index is from zero to six. There are three main features of an array:

### **Advantages of Arrays in Java:**

- Java arrays enable you to access any element randomly with the help of indexes
- It is easy to store and manipulate large data sets
- Disadvantages of Arrays in Java
- The size of the array cannot be increased or decreased once it is declared—arrays have a fixed size
- Java cannot store heterogeneous data. It can only store a single type of primitives.
- Define an Array in Java
- Arrays in Java are easy to define and declare. First, we have to define the array. The syntax for it is:

```
type var-name[];  
OR  
type[] var-name;
```

Here, the type is int, String, double, or long. Var-name is the variable name of the array.

Example:-

```
class Testarray{  
public static void main(String args[]){  
int a[]=new int[5];  
a[0]=10;  
a[1]=20;  
a[2]=70;  
a[3]=40;  
a[4]=50;  
for(int i=0;i<a.length;i++){  
System.out.println(a[i]);  
}  
}
```

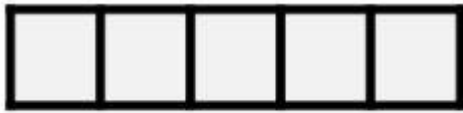
### **Types of Arrays:**

There are three types of arrays. We use these types of arrays as per the requirement of the program. These are:

#### **1. One-dimensional Array:**

Also known as a linear array, the elements are stored in a single row. For example:



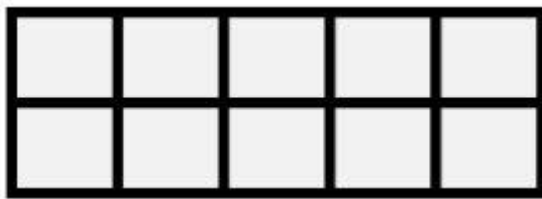


```
int Array = new int[5];
```

In this example, we have an array of five elements. They are stored in a single line or adjacent memory locations.

## 2. Two-dimensional Array:

Two-dimensional arrays store the data in rows and columns:



```
int[][] Array = new int[2][5];
```

In this, the array has two rows and five columns

The index starts from 0,0 in the left-upper corner to 1,4 in the right lower corner.

In this Java code, we have a two-dimensional array.

We have two rows and three columns. Brackets separate the rows, and the number of elements separates the columns.

For this, we use two for loops: one for rows and one for each element in the row. When we execute this program, the result will be as follows:

### Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

*//Java Program to illustrate the use of multidimensional array*

```
class Testarray3 {
public static void main(String args[]) {
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}}
```

## Java Operators

### Operators in Java:

Operator in [Java](#) is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in Java which are given below:

- 1) Unary Operator,
- 2) Arithmetic Operator,
- 3) Shift Operator,
- 4) Relational Operator,
- 5) Bitwise Operator,
- 6) Logical Operator,
- 7) Ternary Operator and
- 8) Assignment Operator.

### Java Unary Operator:

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

**Java Unary Operator Example: ++ and --**

```
public class OperatorExample
{
    public static void main(String args[])
    {
        int x=10;
        System.out.println(x++); //10 (11)
        System.out.println(++x); //12
        System.out.println(x--); //12 (11)
        System.out.println(--x); //10
    }
}
```

**Output:**

10  
12

### Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

**Java Arithmetic Operator Example**

```
public class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=5;
        System.out.println(a+b); //15
        System.out.println(a-b); //5
        System.out.println(a*b); //50
    }
}
```

```
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

### Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true. The bitwise & operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

### Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false. The bitwise | operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

### Java Ternary Operator:

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

### Java Ternary Operator Example

```
public class OperatorExample{
```

```

public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}

```

### Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

### Java Assignment Operator Example

```

public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}
}

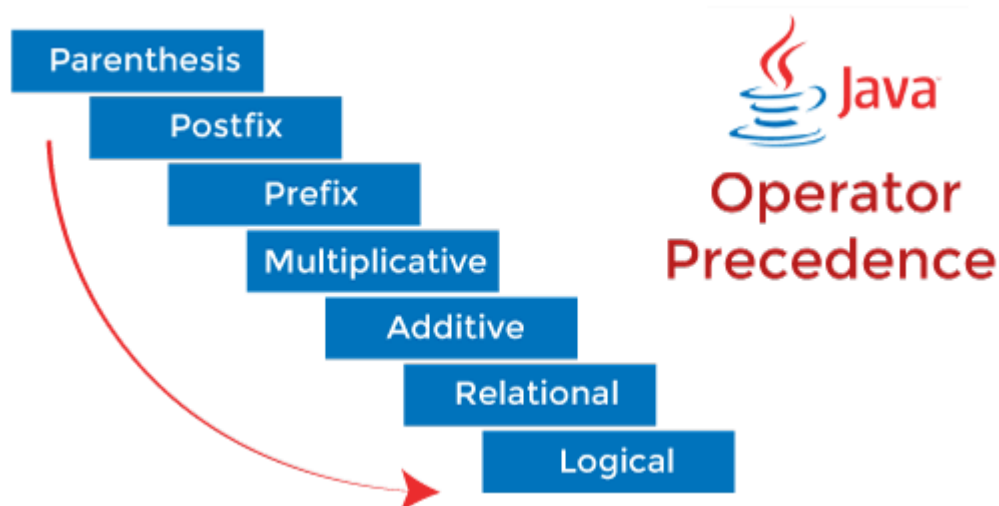
```

### Operator precedence:-

#### What is operator precedence?

The **operator precedence** represents how two expressions are bind together. In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate.

While solving an expression two things must be kept in mind the first is a **precedence** and the second is **associativity**.



### Precedence:

Precedence is the priority for grouping different types of operators with their operands.

It is meaningful only if an expression has more than one operator with higher or lower precedence. The operators having higher precedence are evaluated first.

### Associativity:

We must follow associativity if an expression has more than two operators of the same precedence. In such a case, an expression can be solved either **left-to-right** or **right-to-left**, accordingly.

### Java Operator Precedence Example:

Let's understand the operator precedence through an example. Consider the following expression and guess the answer.

**1 + 5 \* 3**

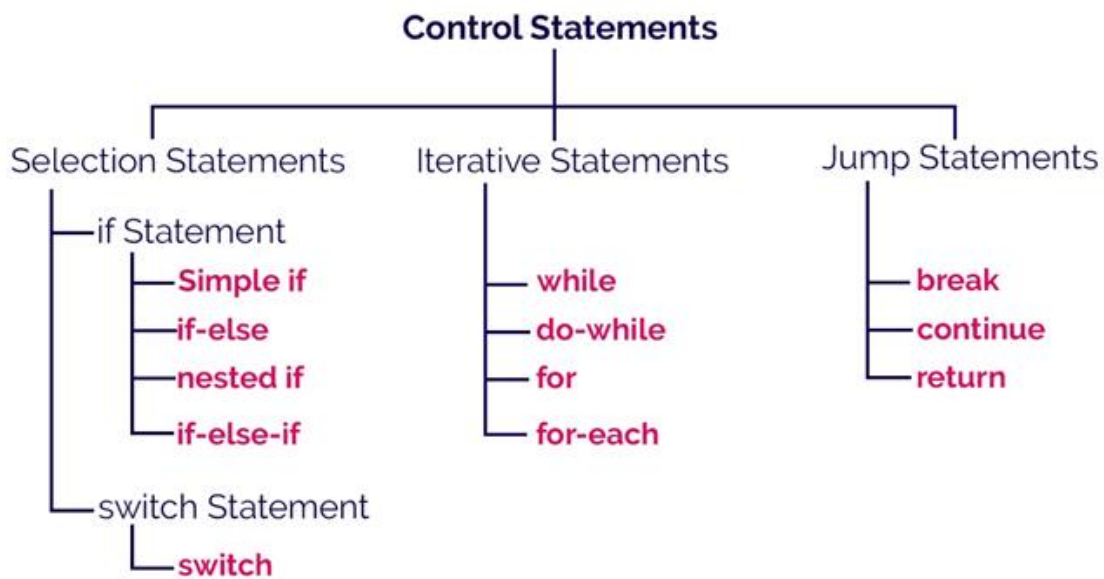
You might be thinking that the answer would be **18** but not so. Because the multiplication (\*) operator has higher precedence than the addition (+) operator. Hence, the expression first evaluates  $5 * 3$  and then evaluates the remaining expression i.e.  $1 + 15$ . Therefore, the answer will be **16**.

Let's see another example. Consider the following expression.

$x + y * z / k$

In the above expression, \* and / operations are performed before + because of precedence. y is multiplied by z before it is divided by k because of associativity.

Control statements



In java, the control statements are the statements which will tell us that in which order the instructions are getting executed. The control statements are used to control the order of execution according to our requirements.

**Java** provides statements that can be used to control the flow of Java code.

Such statements are called control flow statements.

Java provides three types of control flow statements.

**Decision Making statements**

1. if statements
2. switch statement
3. Loop statements
4. do while loop
5. while loop
6. for loop
7. for-each loop
8. Jump statements
9. break statement
10. continue statement

**Decision-Making statements:**

**1) If Statement:**

In Java, the "if" statement is used to evaluate a condition.

The condition of the If statement gives a Boolean value, either true or false.

In Java, there are four types of if-statements given below.

Simple if statement

if-else statement

if-else-if ladder  
Nested if-statement

### 1) Simple if statement:

It is the most basic statement among all control flow statements in Java.

It evaluates a Boolean expression

It enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition)
{
statement 1; //executes when condition is true
}
```

Example

```
public class Student
{
public static void main(String[] args)
{
int x = 10;
int y = 12;
if(x>y )
{
System.out.println("x is is greater than y");
}
}
}
```

### 2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.

The else block is executed if the condition of the if-block is evaluated as false.

**Syntax:**

```
if(condition)
{
statement 1; //executes when condition is true
}
Else
{
statement 2; //executes when condition is false
}
```

```

public class Student
{
    public static void main(String[] args)
    {
        int x = 10;
        int y = 12;
        if(x+y < 10) {
            System.out.println("x + y is less than 10");
        } else {
            System.out.println("x + y is greater than 20");
        }
    }
}

```

### 3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements.

where the program may enter in the block of code where the condition is true.

Syntax of if-else-if statement is given below.

```

if(condition 1)
{
    statement 1; //executes when condition 1 is true
}
else if(condition 2)
{
    statement 2; //executes when condition 2 is true
}
else {
    statement 2; //executes when all the conditions are false
}

```

```

public class Student
{
    public static void main(String[] args)
    {
        String city = "Delhi";
        if(city == "Meerut")
        {
            System.out.println("city is meerut");
        } else if (city == "Noida")
        {
            System.out.println("city is noida");
        }
    }
}

```



```

}else if(city == "Agra")
{
System.out.println("city is agra");
}else
{
System.out.println(city);
}
}
}
}

```

#### 4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```

if(condition 1)
{
statement 1; //executes when condition 1 is true
if(condition 2)
{
statement 2; //executes when condition 2 is true
}
Else
{
statement 2; //executes when condition 2 is false
}
}

```

#### Switch Statement:

In Java, Switch statements are similar to if-else-if statements.

The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

The syntax to use the switch statement is given below.

```

switch (expression){
    case value1:
        statement1;
        break;
    .
    .
    .
    case valueN:
        statementN;
        break;
}

```

```

    default:
        default statement;
}
public class Student
{
    public static void main(String[] args)
    {
        int num = 2;
        switch (num){
            case 0:
                System.out.println("number is 0");
                break;
            case 1:
                System.out.println("number is 1");
                break;
            default:
                System.out.println(num);
        }
    }
}

```

### Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.

In Java, we have three types of loops are executed.

for loop  
while loop  
do-while loop

### **Java for loop:**

In Java, **for loop** is similar to **C** and **C++**.

It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.

Syntax:-

```

for(initialization, condition, increment/decrement)
{
    //block of statements
}
public class Cal
{
    public static void main(String[] args)
    {

```

```

int j;
for(int j = 1; j<=10; j++)
{
System.out.println("The j value is " + j);
}
}

```

### Java while loop:

It is also known as the entry-controlled loop since the condition is checked at the start of the loop.

If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```

while(condition)
{
//looping statements
}

```

Consider the following example.

```

public class val
{
public static void main(String[] args) {
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10)
{
System.out.println(i);
}
}
}
}

```

### Java do-while loop:

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

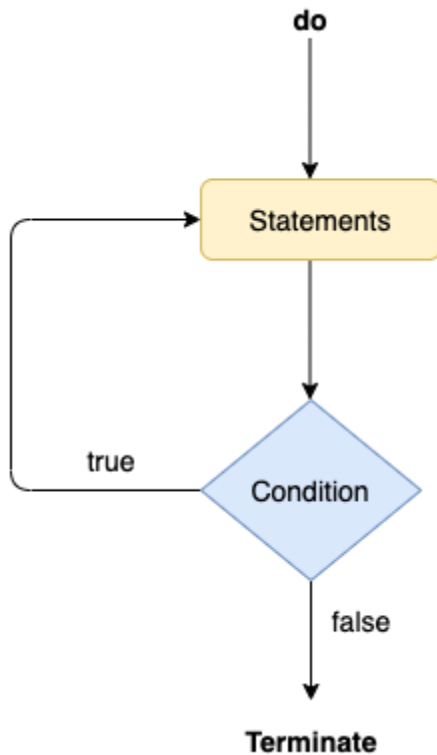
```

do
{

```

```
//statements  
} while (condition);
```

he flow chart of the do-while loop is given in the following image.



```
public class val  
{  
    public static void main(String[] args)  
    {  
        int i = 0;  
        do  
        {  
            System.out.println("the i value is "+i);  
            i++;  
        } while(i<=10);  
    }  
}
```

### Jump Statements:

Jump statements are used to transfer the control of the program to the specific statements.

In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

#### Java break statement:

As the name suggests, the **break statement** is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement

```
public class BreakExample {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        for(int i = 0; i<= 10; i++) {  
            System.out.println(i);  
            if(i==6) {  
                break;  
            }  
        }  
    }  
}
```

#### Java continue statement:

Unlike break statement, the **continue statement** doesn't break the loop whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

```
public class BreakExample  
{  
    public static void main(String[] args)  
    {  
        for(int i = 0; i<= 10; i++)  
        {  
            System.out.println(i);  
            if(i==6)  
            {  
                continue;  
            }  
        }  
    }  
}
```

OOPS THROUGH JAVA