**Java is a programming language**

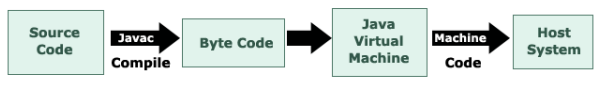
Java is a high level,object-oriented and secure programming language. Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. **James Gosling** is known as the father of Java

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

**Robust** Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. All of the above features makes the java language robust.

**What is Java Virtual Machine?**

Languages like C and Pascal converts the source code into machine code for one specific type of machine and the machine language vary from system to system . But Java compiler produce code for a virtual machine . JVM converts the byte code into machine code for the computer one wants to run.



The JVM is a part of JRE that is required by every operating system. Each operating system and CPU architecture requires a JRE. JRE consists of a set of base classes i.e. Java API as well as a JVM. JVM is java interpreter as it converts the byte code into machine code for the computer one wants to run. JRE consists of a number of classes based on JavaAPI and JVM, and without JRE, it is impossible to run Java.

**-----------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>**

**Application**

1. Desktop Applications

2. Web Applications

3. Enterprise Applications such as banking applications.

4. Mobile

5. Embedded System

6. Robotics

7. Games, etc

------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>>>>>>>>>>

To create a simple Java program, you need to create a class that contains the main method. Let's

understand the requirement first.

The requirement for Java Hello World Example

For executing any Java program, the following software or application must be properly installed.

o Install the JDK if you don't have installed it, download the JDK

and install it.

**Set path** of the jdk/bin directory

o Create the Java program

o Compile and run the Java program

------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>>>>>>>

Creating Hello World Example

**Let's create the hello java program:**

**Simple.java**

class Simple{

public static void main(String args[]){

System.out.println("Hello Java");

}

}

Save the above file as **Simple.java.**

To compile: **javac Simple.java**

To execute: **java Simple**

Output:

Hello Java

--------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>

**Parameters used in First Java Program**

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

o **class** keyword is used to declare a class in Java.

o **public** keyword is an access modifier that represents visibility. It means it is visible to all.

o **static** is a keyword. If we declare any method as static, it is known as the static method. The core

advantage of the static method is that there is no need to create an object to invoke the static

method. The main() method is executed by the JVM So, it saves memory.

o **void** is the return type of the method. It means it doesn't return any value.

o **main** represents the starting point of the program.

o **String[] args** or String args[] is used for command line argument

. We will discuss it in coming section.

o **System.out.println()** is used to print statement

---------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>

**Java OOPs** Concepts Object-Oriented Programming is a paradigm that provides many concepts, such as inheritance, data binding, polymorphism, etc

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented

Programming is a methodology or paradigm to design a program using classes and objects. It simplifies

software development and maintenance by providing some concepts:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

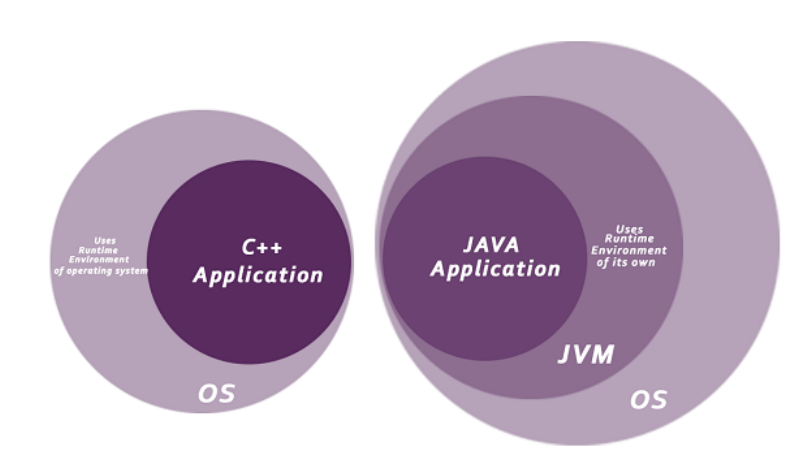
**-------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>>**

**Java is best known for its security.** With Java, we can develop virus-free systems. Java is secured

because:

o No explicit pointer

o Java Programs run inside a virtual machine sandbox



----------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>>>

**How to set path in Java**

The path is required to be set for using tools such as javac, java, etc.

There are two ways to set the path in Java:

1. Temporary

2. Permanent

**1)** How to set the Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow the following steps:

o Open the command prompt

o Copy the path of the JDK/bin directory

o Write in command prompt: set path=copied\_path

**2)** How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

o Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user

variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

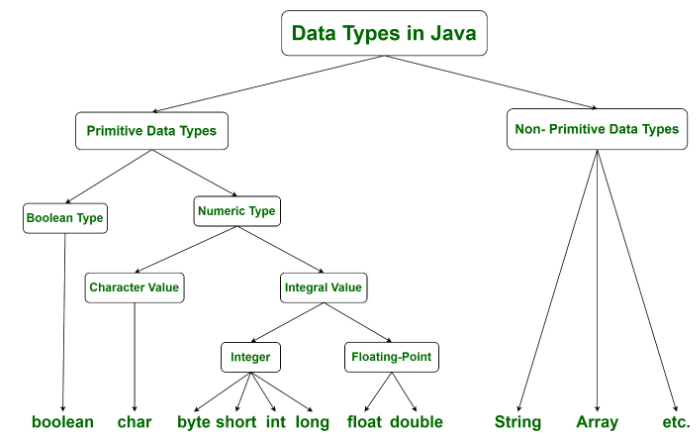
----------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>>

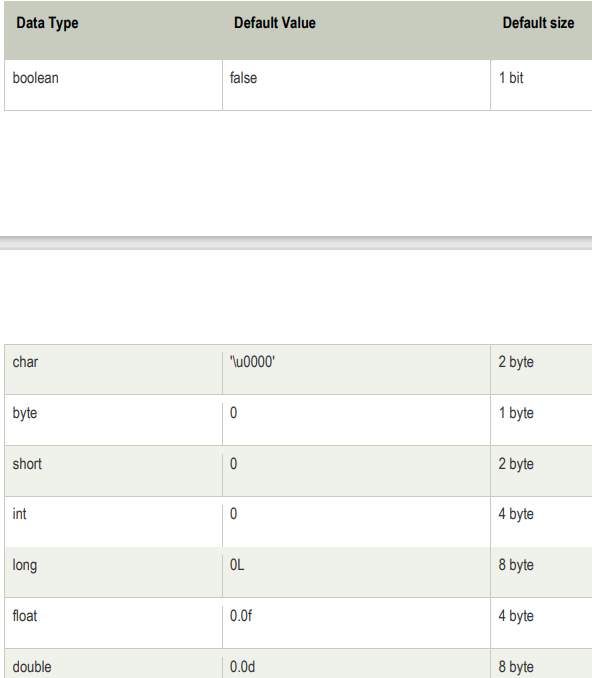
**Data Types in Java**

Data types specify the different sizes and values that can be stored in the variable. There are two types of

data types in Java:

1. **Primitive Data Type**: such as boolean, char, int, short, byte, long, float, and double
2. **Non-Primitive Data Type :** or Object Data type: such as String, Array, etc.





-----------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>

**Java Variables**

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type .Variable is a name of memory location.

**Declaring (Creating) Variables**

Syntax :

Data\_type variable\_Name = value;

Int varma=400;

**Rules to Declare a Variable**

1. A variable name can consist of Capital letters **A-Z**, lowercase letters **a-z** digits **0-9**, and two special characters such as **\_** underscore and **$** dollar sign.
2. The first character must not be a digit.
3. Blank spaces cannot be used in variable names.
4. Java keywords cannot be used as variable names.
5. Variable names are case-sensitive.
6. There is no limit on the length of a variable name but by convention, it should be between 4 to 15 chars.
7. Variable names always should exist on the left-hand side of assignment operators.



**Types of Variables**

There are three types of variables in Java:

o local variable

o instance variable

o static variable

1. **Local Variable** A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.

**Example:**

public class Test {

public void pupAge() {

int age = 0 // local variabe

age = age + 7;

System.out.println("Puppy age is : " + age)

}

public static void main(String args[]) {

Test t = new Test(); /\*classname objname=new classname();\*/

t.pupAge();

}

}

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

🡪Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

🡪 The instance variables are visible for all methods, constructors, and block in the class.

🡪 Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

**Example:**

import java.io.\*;

public class Employee {

public String name;

private double salary;

public Employee (String empName) {

name = empName;

}

public void setSalary(double empSal) {

salary = empSal;

}

public void printEmp() {

System.out.println("name : " + name );

System.out.println("salary :" + salary);

}

public static void main(String args[]) {

Employee empOne = new Employee("Ransika");

empOne.setSalary(1000);

empOne.printEmp();

}

}

1. **Static variable** A variable that is declared as static is called a static variable. Memory allocation for static variables happens only once when the class is loaded in the memory. Static variables are created when the program starts and destroyed when the program stops. Static variables can be accessed by calling with the class name **ClassName.VariableName.**

**Example:**

class Student{

   int rollno;

   String name;

   static String college ="SKBR";

   Student(int r, String n){

   rollno = r;

   name = n;

   }

   void display (){System.out.println(rollno+" "+name+" "+college);}

}

public class TestStaticVariable1{

 public static void main(String args[]){

 Student s1 = new Student(111,"varma");

 Student s2 = new Student(222,"rohit");

 s1.display();

 s2.display();

 }

}

**Example:**

**class** Test {

**static** **int** a = m1();

**static**

    {

        System.out.println("Inside static block");

    }

**static** **int** m1()

    {

        System.out.println("from m1");

**return** 20;

    }

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

    {

        System.out.println("Value of a : " + a);

        System.out.println("from main");

    }

}

------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>>

**Operator in Java**

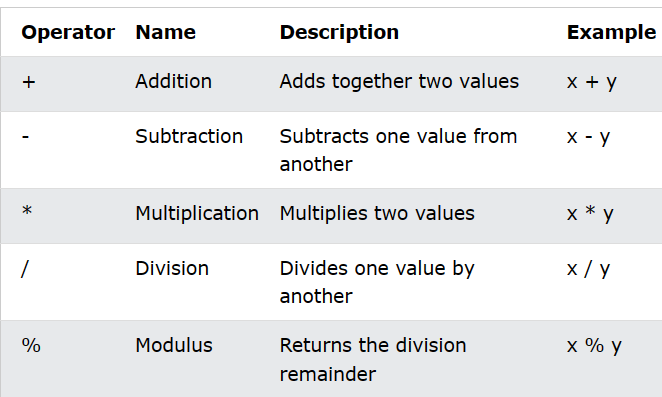
is a symbol that is used to perform operations on operands.(a+b)

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

1. Arithmetic Operator,
2. Comparision Operator,
3. Bitwise Operator,
4. Logical Operator,
5. Ternary Operator and
6. Assignment Operator.
7. Increment or Decrement Operator

**Arithmetic Operators**

Arithmetic operators are used to perform common mathematical operations.



**Example:**

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

}}

**Example:// BODMAS**

public class OperatorExample{

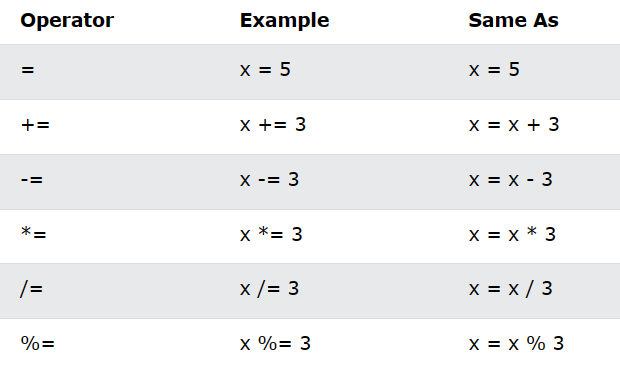
public static void main(String args[]){

System.out.println(10\*10/5+3-1\*4/2);

}}

**Java Assignment Operators**

Assignment operators are used to assign values to variables.In the example below, we use the assignment operator (=) to assign the value.



**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);

}}

**Example:**

import java.io.\*;

class Assignment {

public static void main(String[] args)

{

int num1 = 10, num2 = 20;

System.out.println("num1 = " + num1);

System.out.println("num2 = " + num2);

num1 += num2;

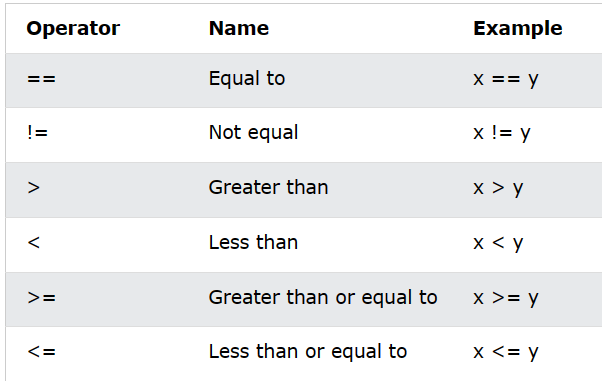
System.out.println("num1 = " + num1);

}

}

**Java Comparison Operators**

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.The return value of a comparison is either true or false. These values are known as Boolean values.



**Example:**

public class Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

System.out.println("a == b = " + (a == b) );

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 >= a = " + (b >= a) );

System.out.println("b <= a = " + (b <= a) );

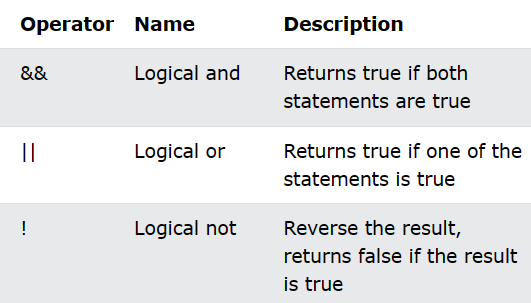
}

}

**Java Logical Operators**

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:



**Example:**

public class Test {

public static void main(String args[]) {

boolean a = true;

boolean b = false;

System.out.println("a && b = " + (a&&b)); //

System.out.println("a || b = " + (a||b) );

System.out.println("!(a && b) = " + !(a && b));

}

}

**Example:**

import java.io.\*;

class Logical {

public static void main(String[] args)

{

int a = 10, b = 20, c = 20, d = 0;

System.out.println("Var1 = " + a);

System.out.println("Var2 = " + b);

System.out.println("Var3 = " + c);

if ((a < b) && (b == c)) {

d = a + b + c;

System.out.println("The sum is: " + d);

}

else

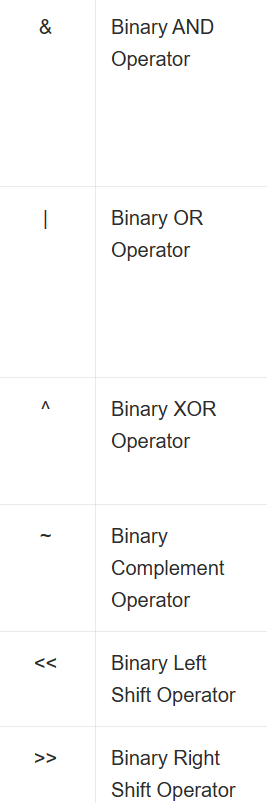
System.out.println("False conditions");

}

}

**Bitwise operators:**

The Java Bitwise Operators allow access and modification of a particular bit inside a section of the data.

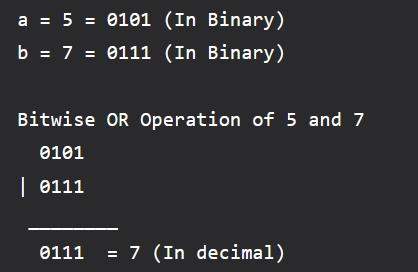


**Bitwise OR (|)**

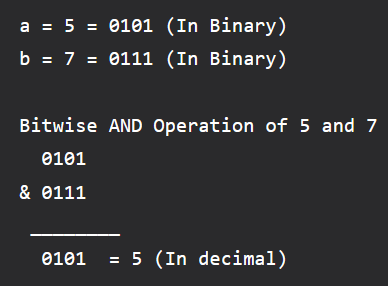
**16 8 4 2 1**

**1 0 1 0**

**0 1 0 1**

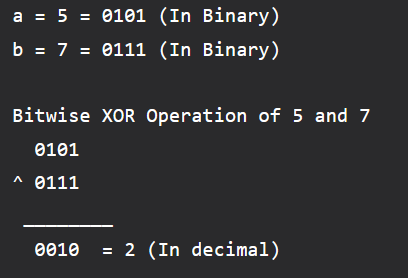


**Bitwise AND (&)**

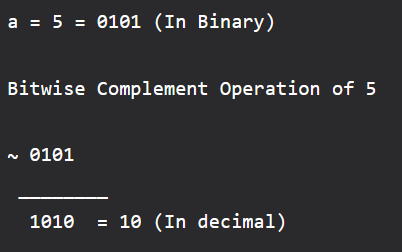


**Bitwise XOR (^)**

This operator is a binary operator, denoted by ‘^.’ It returns bit by bit XOR of input values, i.e., if corresponding bits are different, it gives 1, else it shows 0.



**Bitwise Complement (~) -(n+1)**



**Example:**

public class operators {

public static void main(String[] args)

{

int a = 5;

int b = 7;

System.out.println("a&b = " + (a & b));

System.out.println("a|b = " + (a | b));

System.out.println("a^b = " + (a ^ b));

System.out.println("~a = " + ~a);

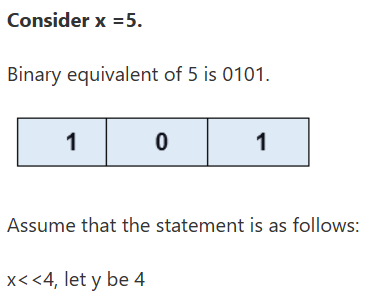
a &= b;

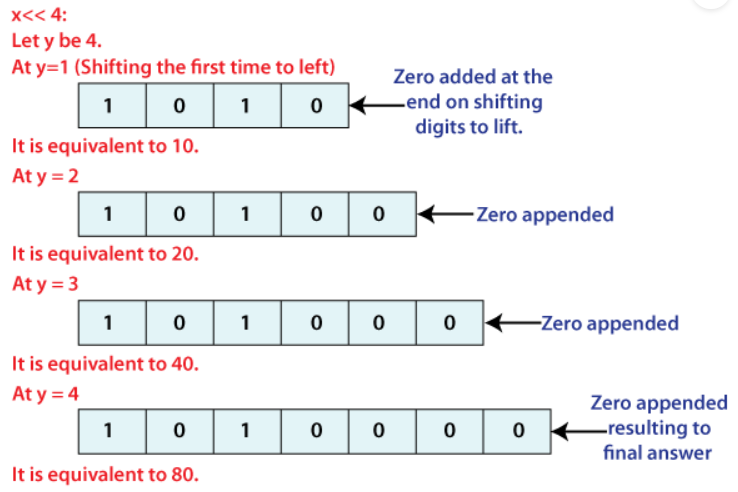
System.out.println("a= " + a);

}

}

**Left Shift Operator**

****

****

**Example:**

public class LeftShift {

    public static void main( String[ ] args ) {

        int x = 5 ;

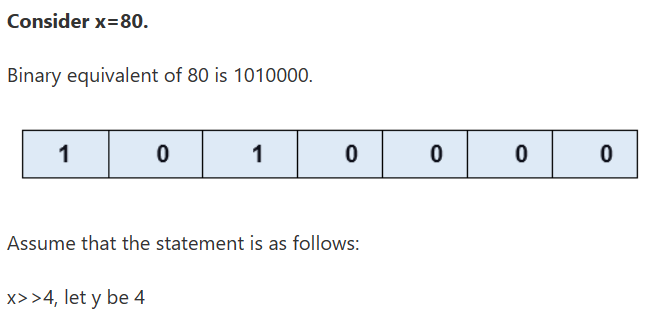
        int y = x << 4 ;

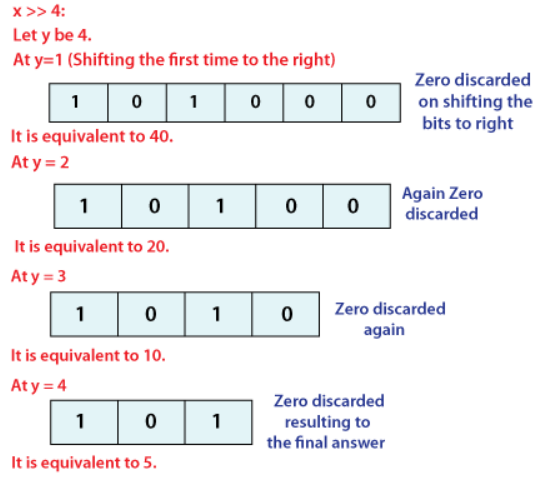
        System.out.println( " The value is: " +y) ;

    }

}

**Right Shift Operator**

****

****

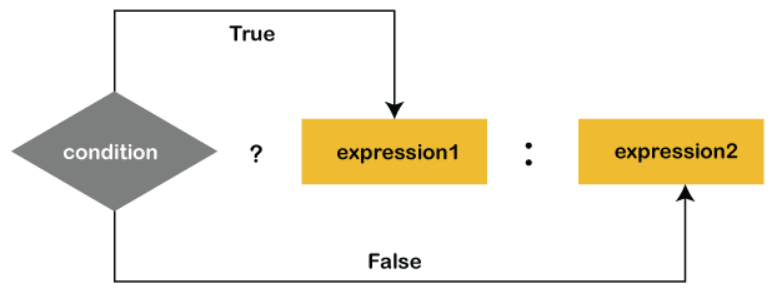
**Ternary Operator Java**

In Java, the ternary operator is a type of Java conditional operator.

The **ternary operator (? :)** consists of three operands.

**Syntax:**

variable = (condition) ? expression1 : expression2



**Example:**

public class TernaryOperatorExample

{

public static void main(String args[])

{

int x, y;

x = 20;

y = (x == 1) ? 61: 90;

System.out.println("Value of y is: " + y); //90

y = (x == 20) ? 61: 90;

System.out.println("Value of y is: " + y); //61

}

}

**Example:**

public class LargestNumberExample

{

public static void main(String args[])

{

int x=69;

int y=89;

int z=79;

int largestNumber= (x > y) ? (x > z ? x : z) : (y > z ? y : z);

System.out.println("The largest numbers is: "+largestNumber);

}

}

**Increment and Decrement Operators**

The decrement (--) and increment (++) operators are special types of operators used in programming languages to decrement and increment the value of the given variable by 1 (one), respectively>

Following types of increment and decrement operators are found in the C language:

1. Prefix Increment operator
2. Prefix Decrement operator
3. Postfix Increment operator
4. Postfix Decrement operator

**Example:**

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

}}

**Example:**

public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=10;

System.out.println(a++ + ++a);//10+12=22

System.out.println(b++ + b++);//10+11=21

}

}

-----------------------------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>

Type Casting in Java

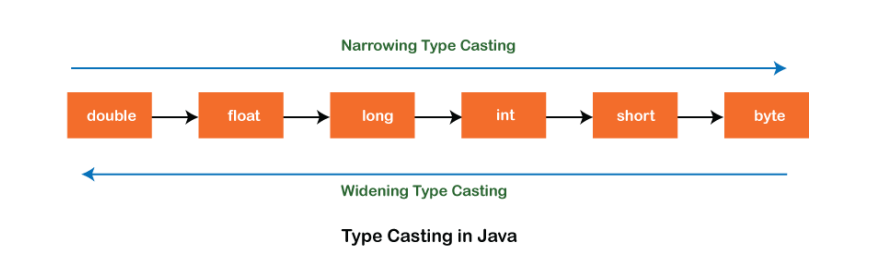
type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

**Types of Type Casting**

There are two types of type casting:

o **Widening** Type Casting

o **Narrowing** Type Casting



1. **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.

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

**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;

//automatically converts the long type into float type

float z = y;

System.out.println("Before conversion, int value "+x);

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

System.out.println("After conversion, float value "+z);

}

}

**Output:-**

Before conversion, the value is: 7

After conversion, the long value is: 7

After conversion, the float value is: 7.0

1. **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. If we do not perform casting then the compiler reports a compile-time error.

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

**NarrowingTypeCastingExample.java**

public class NarrowingTypeCastingExample

{

public static void main(String args[])

{

double d = 166.66;

//converting double data type into long data type

long l = (long)d;

//converting long data type into int data type

int i = (int)l;

System.out.println("Before conversion: "+d);

//fractional part lost

System.out.println("After conversion into long type: "+l);

//fractional part lost

System.out.println("After conversion into int type: "+i);

}

}

**Output:-**

Before conversion: 166.66

After conversion into long type: 166

After conversion into int type: 166

**Example:** Type conversion from int to String

class Main {

public static void main(String[] args) {

string num = 10;

System.out.println("The value is: " + num);

String data = String.valueOf(num);

System.out.println("The string value is: " + data);

System.out.println("The string value is: " + (num+data));

}

}

**Example**: Type conversion from String to int

class Main {

public static void main(String[] args) {

String data = "10";

System.out.println("The string value is: " + data);

int num = Integer.parseInt(data);

System.out.println("The integer value is: " + num);

}

}

**NOTE:** the **parseInt()** method of the Java Integer class to convert a string type variable into an int variable.

-----------------------------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>>

**Java Command Line Arguments**

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input.The users can pass the arguments during the execution bypassing the command-line arguments inside the main() method.We need to pass the arguments as space-separated values. We can pass both strings and primitive data types(int, double, float, char, etc) as command-line arguments. These arguments convert into a string array and are provided to the main() function as a string array argument.

**Example:**

class CommandLineExample{

public static void main(String args[]){

System.out.println("Your first argument is: "+args[0]);

}

}

**Example to print all the values:**

class A{

public static void main(String args[]){

for(int i=0;i<args.length;i++)   **(or)** System.out.println("args[" + x + "]: " + args[ x ]);

System.out.println(args[i]);

}

}

**Example to check condition while passing arguments :**

class Commandla {

public static void main(String[] args)

{

if (args.length > 0) {

System.out.println("The command line"

+ " arguments are:");

for (String val : args)

System.out.println(val);

}

else

System.out.println("No command line "

+ "arguments found.");

}

}

**Example: Numeric Command-Line Arguments**

class Main {

public static void main(String[] args) {

for(String str: args) {

// convert into integer type

int argument = Integer.parseInt(str);

System.out.println("Argument in integer form: " + argument);

}

}

}

------------------------------------------------------------------------🡪>>>>>>>>>>>>>>>

**Control Statements in Java**

**Control statements in java** are an essential part of any programming language and are used to control the flow of a program. They determine what action the program should take and what code should be executed based on certain conditions. Control statements in java are used to make decisions and execute different code blocks accordingly.

**Java provides three types of control flow statements**.

1. Selection statements (or) Decision Making (if, switch)
2. Iteration statements (while, do-while, for).
3. Jump statements (break, continue, return).

**Decision-Making statements:**

decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.

**---------------🡪**

**If Statement:**

* Simple if statement
* if-else statement
* if-else-if ladder
* Nested if-statement

**if :**

class IfDemo {

public static void main(String args[])

{

int i = 10;

if (i < 15){

System.out.println("10 is less than 15");

System.out.println("Outside if-block");

// both statements will be printed}

}

}

**if else:**

class IfElseDemo {

public static void main(String args[])

{

int i = 20;

if (i < 15)

System.out.println("i is smaller than 15");

else

System.out.println("i is greater than 15");

}}

**if-else-if ladder :**

import java.io.\*;

class Ifelseladder {

public static void main(String[] args)

{

// initializing expression

int i = 20;

// condition 1

if (i == 10)

System.out.println("i is 10\n");

// condition 2

else if (i == 15)

System.out.println("i is 15\n");

// condition 3

else if (i == 20)

System.out.println("i is 20\n");

else

System.out.println("i is not present\n");

System.out.println("Outside if-else-if");

}

}

**Nested-if:**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Nestedif

{

public static void main(String args[])

{

int a=10;

int b=20;

if(a==10){

if(b==20){

System.out.println("hello world");

} } } }

**-----------------🡪**

**switch Statement**

The switch statement is another multi-way decision making statement. You can consider the switch as an alternative for if-else-if ladder. that allows the programmer to check multiple conditions and execute a different code block for each condition. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

**Points to be noted about switch statement**

* The case variables can be int, short, byte, char, or enumeration.
* Cases cannot be duplicate
* Default statement is executed when any of the case doesn't match the value of expression. It is optional.
* Break statement terminates the switch block when the condition is satisfied.  
  It is optional, if not used, next case is executed.

**Example:**

Class SwitchExample{

public static void main(String args[ ])

{

int day = 4;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

case 4:

System.out.println("Thursday");

break;

case 5:

System.out.println("Friday");

break;

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;

}

}

**Iteration Statements**

While selection statements are used to select a set of statements based on a condition, iteration statements are used to repeat a set of statements again and again based on a condition for finite or infinite number of times.

1. for loop
2. while loop
3. do-while loop

-------------🡪

**for loop:**

**example:**

class ForEx {

public static void main(String[] args)

{

for (int i = 1; i <= 10; i++) {

System.out.println(i);

}

}

}

**Example:**

public class Calculattion {

public static void main(String[] args) {

// TODO Auto-generated method stub

int sum = 0;

for(int j = 1; j<=10; j++) {

sum = sum + j;

}

System.out.println("The sum of first 10 natural numbers is " + sum);

}

}

------------🡪

**while loop:**

**example:**

class whileLoopDemo {

public static void main(String args[])

{

// initialization expression

int i = 1;

// test expression

while (i < 6) {

System.out.println("Hello World");

// update expression

i++;

}

}

}

**Example:**

public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

while(i<=10) {

System.out.println(i);

i = i + 2;

}

}

}

-----------🡪

**do-while:**

**example:**

class DoWhileEx {

public static void main(String[] args)

{

int i = 0;

do {

System.out.println("Print statement");

i++;

}

while (i < 0);

}

}

**Example:**

public class Calculation {

public static void main(String[] args) {

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

do {

System.out.println(i);

i = i + 2;

}while(i<=10);

}

}

**Jump Statements**

As the name implies, jump statements are used to alter the sequential flow of the program. Jump statements supported by Java are: break, continue and return.  In other words, jump statements transfer the execution control to the other part of the program.

1. Break
2. Continue
3. Return

**break statement:**

**example:**

import java.io.\*;

class BreakEx {

public static void main (String[] args) {

int n = 1;

switch(n){

case 1:

System.out.println("GFG");

break;

case 2:

System.out.println("Second Case");

break;

default:

System.out.println("default case");

}

}

}

**Example:**

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) {

continue;

}

}

}

}

**Continue statement:**

**Example:**

import java.util.\*;

public class GFG {

public static void main(String args[])

{

for (int i = 0; i <= 15; i++) {

if (i == 10 || i == 12) {

continue;

}

System.out.print(i + " ");

}

}

}

**Example:**

public class ContinueExample {

public static void main(String[] args) {

for(int i = 0; i<= 2; i++) {

  for (int j = i; j<=5; j++) {

if(j == 4) {

continue;

}

System.out.println(j);

}

}

}

}

**return Statement**

The return statement can only be used inside methods which can return control or a value back to the calling method. The return statement can be written at any line inside the body of the method. Common convention is to write the return statement at the end of the method’s body. the return statement is used for returning a value when the execution of the block is completed. The return statement inside a loop will cause the loop to break and further statements will be ignored by the compiler.

**Following are the important points must remember while returning a value**:

* The return type of the method and type of data returned at the end of the method should be of the same type. For example, if a method is declared with the float return type, the value returned should be of float type only.
* The variable that stores the returned value after the method is called should be a similar data type otherwise, the data might get lost.

**Example:**

public class SampleReturn1

{

    public int CompareNum()

    {

        int x = 3;

        int y = 8;

        System.out.println("x = " + x + "\ny = " + y);

        if(x>y)

            return x;

        else

            return y;

    }

    public static void main(String ar[])

    {

        SampleReturn1 obj = new SampleReturn1();

        int result = obj.CompareNum();

        System.out.println("The greater number among x and y is: " + result);

    }

}

--- ---- --- --- --- ---- --- --- -- -- --- --- ---- --- --- -- -- -- -- --- --- ---- --- ---- --- ---- --- --- -- ---- --- --- --- ---- --- -- ------ --- -- --- 🡪

**Arrays in Java**

an array is a collection of similar type of elements which has contiguous memory location. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on



**Advantages**

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

**Disadvantages**

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

**Types of Array in java**

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

**Single Dimensional Array in Java**

**Syntax to Declare an Array in Java**

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

**Instantiation of an Array in Java**

1. arrayRefVar=**new** datatype[size];

**Example of Java Array**

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

**class** Testarray{

**public** **static** **void** main(String args[]){

**int** a[]=**new** **int**[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

**for**(**int** i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

**Output:**

10

20

70

40

50

**Declaration, Instantiation and Initialization of Java Array**

We can declare, instantiate and initialize the java array together by:

1. **int** a[]={33,3,4,5};//declaration, instantiation and initialization

**example**

//Java Program to illustrate the use of declaration, instantiation

//and initialization of Java array in a single line

**class** Testarray1{

**public** **static** **void** main(String args[]){

**int** a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array

**for**(**int** i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

**Output:**

33

3

4

5

**For-each Loop for Java Array**

We can also print the Java array using [**for-each loop**](https://www.javatpoint.com/for-each-loop). The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The **syntax** of the for-each loop is given below:

1. **for**(data\_type variable:array){
2. //body of the loop
3. }

**example** of array using the for-each loop.

//Java Program to print the array elements using for-each loop

**class** Testarray1{

**public** **static** **void** main(String args[]){

**int** arr[]={33,3,4,5};

//printing array using for-each loop

**for**(**int** i:arr)

System.out.println(i);

}}

**Output:**

33

3

4

5

**Passing Array to a Method in Java**

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get the minimum number of an array using a method.

//Java Program to demonstrate the way of passing an array

//to method.

**class** Testarray2{

//creating a method which receives an array as a parameter

**static** **void** min(**int** arr[]){

**int** min=arr[0];

**for**(**int** i=1;i<arr.length;i++)

**if**(min>arr[i])

  min=arr[i];

System.out.println(min);

}

**public** **static** **void** main(String args[]){

**int** a[]={33,3,4,5};//declaring and initializing an array

min(a);//passing array to method

}}

**Output:**

3

**Anonymous Array in Java**

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

//Java Program to demonstrate the way of passing an anonymous array

//to method.

**Example:**

**public** **class** TestAnonymousArray{

//creating a method which receives an array as a parameter

**static** **void** printArray(**int** arr[]){

**for**(**int** i=0;i<arr.length;i++)

System.out.println(arr[i]);

}

**public** **static** **void** main(String args[]){

printArray(**new** **int**[]{10,22,44,66});//passing anonymous array to method

}}

**Output:**

10

22

44

66

**Multidimensional Array in Java**

In such case, data is stored in row and column based index (also known as matrix form).

**Syntax** to Declare Multidimensional Array in Java

1. dataType[][] arrayRefVar; (or)

2. dataType [][]arrayRefVar; (or)

3. dataType arrayRefVar[][]; (or)

4. dataType []arrayRefVar[];

**Example** to instantiate Multidimensional Array in Java

* 1. int[][] arr=new int[3][3];//3 row and 3 column

**Example** to initialize Multidimensional Array in Java

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

**example:**

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 Program to demonstrate the addition of two matrices in Java**

class Testarray5{

public static void main(String args[]){

//creating two matrices

int a[][]={{1,3,4},{3,4,5}};

int b[][]={{1,3,4},{3,4,5}};

  //creating another matrix to store the sum of two matrices

int c[][]=new int[2][3];

//adding and printing addition of 2 matrices

for(int i=0;i<2;i++){

for(int j=0;j<3;j++){

c[i][j]=a[i][j]+b[i][j];

System.out.print(c[i][j]+" ");

}

System.out.println();//new line

}

}}

**EXAMPLE:**

public class MatrixMultiplicationExample{

public static void main(String args[]){

//creating two matrices

int a[][]={{1,1,1},{2,2,2},{3,3,3}};

int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multiplication of two matrices

int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

c[i][j]=0;

for(int k=0;k<3;k++)

{

c[i][j]+=a[i][k]\*b[k][j];

}//end of k loop

System.out.print(c[i][j]+" ");  //printing matrix element

}//end of j loop

System.out.println();//new line

}

}}

**Example:**

public class TwoDArray {

public static void main(String[] args) {

int rows = 4;

int columns = 4;

int[][] array = new int[rows][columns];

int value = 1;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

array[i][j] = value;

value++;

}

}

System.out.println("The 2D array is: ");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

System.out.print(array[i][j] + " ");

}

System.out.println();

}

}

}

**Output**

The 2D array is:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

### classes in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

**A class in Java can contain:**

* Methods
* Constructors
* Blocks
* Nested class and interface

**Syntax to declare a class:**

class <class\_name>{

//body }

### object in Java

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only. An entity that has state and behavior is known as an object

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**Object Definitions:**

* An object is a real-world entity.
* An object is a runtime entity.
* The object is an entity which has state and behavior.
* The object is an instance of a class.

**Syntax to declare a object :**

Class\_name object\_name = new Class\_name ( ) ;

### Object and Class Example:

**Example:- Main with in the class**

**class** Student{

**int** id=10;

 String name=”Raghu”;

**public** **static** **void** main(String args[]){

  Student s1=**new** Student();//creating an object of Student

  System.out.println(s1.id);//accessing member through reference variable

  System.out.println(s1.name);

 }

}

### Example: main outside the class

**class** Student{

**int** id;

 String name;

}

**class** TestStudent1{

**public** **static** **void** main(String args[]){

  Student s1=**new** Student();

  System.out.println(s1.id);

  System.out.println(s1.name);

 }

}

## **3 Ways to initialize object**

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

### 1)Initialization through reference :

Initializing an object means storing data into the object.

**EX:1**

**class** Student{

**int** id;

 String name;

}

**class** TestStudent2{

**public** **static** **void** main(String args[]){

  Student s1=**new** Student();

  s1.id=101;

  s1.name="Sonoo";

  System.out.println(s1.id+" "+s1.name);//printing members with a white space

 }

}

**EX:2**

**class** Student{

**int** id;

 String name;

}

**class** TestStudent3{

**public** **static** **void** main(String args[]){

  //Creating objects

  Student s1=**new** Student();

  Student s2=**new** Student();

  //Initializing objects

  s1.id=101;

  s1.name="Sonoo";

  s2.id=102;

  s2.name="Amit";

  //Printing data

  System.out.println(s1.id+" "+s1.name);

  System.out.println(s2.id+" "+s2.name);

 }

}

### 2) Initialization through method

 we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

**class** Student{

**int** rollno;

 String name;

**void** insertRecord(**int** r, String n){

  rollno=r;

  name=n;

 }

**void** displayInformation(){System.out.println(rollno+" "+name);}

}

**class** TestStudent4{

**public** **static** **void** main(String args[]){

  Student s1=**new** Student();

  Student s2=**new** Student();

  s1.insertRecord(111,"Karan");

  s2.insertRecord(222,"Aryan");

  s1.displayInformation();

  s2.displayInformation();

 }

}

### 3) Initialization through a constructor :

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created.

That means constructor method is automatically called when the object for the class is created.

**Ex:**

public class Main {

int x;

public Main() {

x = 5;

}

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

**Ex:**

public class Main {

int modelYear;

String modelName;

public Main(int year, String name) {

modelYear = year;

modelName = name;

}

public static void main(String[] args) {

Main myCar = new Main(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

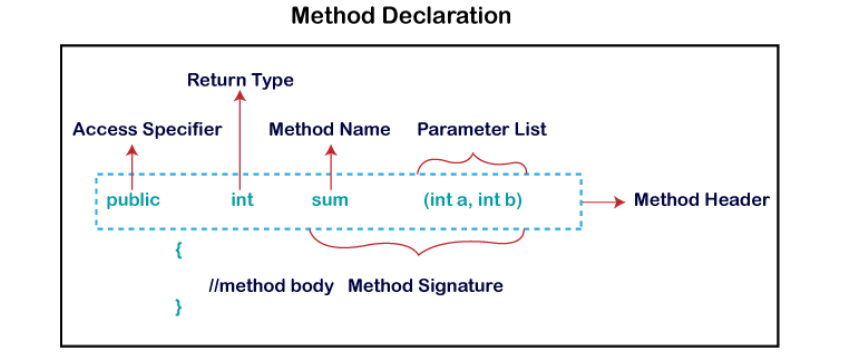
}

}

**Method in Java**

In general, a method is a way to perform some task. Similarly, the method in Java is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using methods

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

**Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of

the method. Java provides four types of access specifier:

**o Public:** The method is accessible by all classes when we use public specifier in our application.

**o Private:** When we use a private access specifier, the method is accessible only in the classes in

which it is defined.

**o Protected:** When we use protected access specifier, the method is accessible within the same

package or subclasses in a different package.

**o Default:** When we do not use any access specifier in the method declaration, Java uses default

access specifier by default. It is visible only from the same package only.

**Return Type**: Return type is a data type that the method returns. If the method does not return anything, we use void keyword.

**Parameter List**: It is the list of parameters separated by a comma and enclosed in the pair of parentheses.

It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

**Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is

enclosed within the pair of curly braces.

**Naming a Method:**

While defining a method, remember that the method name must be a verb and start with a lowercase letter.

**Single-word method name:** sum(), area()

**Multi-word method name:** areaOfCircle(), stringComparision()

**Types of Method**

There are two types of methods in Java:

o Predefined Method

o User-defined Method

**Predefined Method**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method.

Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc.

**EAMPLE:**

public class PredefinedMethodExample {

public static void main(String[] args) {

String str = "Hello World!";

int length = str.length(); // length() is a predefined method in the String class

System.out.println("The length of the string is: " + length);

}

}

**User-defined Method**

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

**EXAMPLE:**

import java.util.Scanner;

public class EvenOdd

{

public static void main (String args[])

{

Scanner scan=new Scanner(System.in);

System.out.print("Enter the number: ");

int num=scan.nextInt();

findEvenOdd(num);

}

public static void findEvenOdd(int num)

{

if(num%2==0)

System.out.println(num+" is even");

else

System.out.println(num+" is odd");

}

}

**Method Overloading**

In Java If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

**Advantage of method overloading**

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type

**Method Overloading: changing no. of arguments**

class Adder{

static int add(int a,int b){return a+b;}

static int add(int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

Output:

22

33

**Method Overloading: changing data type of arguments :**

class Adder{

static int add(int a, int b){return a+b;}

static double add(double a, double b){return a+b;}

}

class TestOverloading2{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

Output:

22

24.9

**Method Overriding in Java**

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

Rules for Java Method Overriding The method must have the same name as in the parent class The method must have the same parameter as in the parent class. There must be an IS-A relationship (inheritance).

**Example:**

class Parent {

private void m1() {

System.out.println("From parent m1()");

}

protected void m2(){

System.out.println("From parent m2()");

}}

class Child extends Parent {

private void m1(){

System.out.println("From child m1()");

}

public void m2(){

System.out.println("From child m2()");

}}

class Main {

public static void main(String[] args){

Parent obj1 = new Parent();

obj1.m2();

Parent obj2 = new Child();

obj2.m2(); }}

**EXAMPLE 2:**

class Parent {

void show() { System.out.println("Parent's show()"); }

}

class Child extends Parent {

{

super.show();

System.out.println("Child's show()");

}

}

class Main {

public static void main(String[] args)

{

Parent obj = new Child();

obj.show();

}

}

**Constructors in Java**

In Java

, a constructor is a block of codes similar to the method. It is called when an instance of the class

is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler

provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not

necessary to write a constructor for a class. It is because java compiler creates a default constructor if your

class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name.

2. A Constructor must have no explicit return type.

3. A Java constructor cannot be abstract, static, final, and synchronized.

**Types of Java constructors**

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)

2. Parameterized constructor

**Java Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. <class\_name>(){ }

**Example of default constructor**

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

//Java Program to create and call a default constructor

class Bike1{

//creating a default constructor

Bike1(){System.out.println("Bike is created");}

//main method

public static void main(String args[]){

//calling a default constructor

Bike1 b=new Bike1();

}

}

Output:

Bike is created

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the

type.

**Example of default constructor that displays the default values**

//Let us see another example of default constructor

//which displays the default values

class Student3{

int id;

String name;

//method to display the value of id and name

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

//creating objects

Student3 s1=new Student3();

Student3 s2=new Student3();

//displaying values of the object

s1.display();

s2.display();

}

}

Output:

0 null

0 null

**Java Parameterized Constructor**

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can

provide the same values also.

**Example of parameterized constructor**

In this example, we have created the constructor of Student class that have two parameters. We can have

any number of parameters in the constructor.

//Java Program to demonstrate the use of the parameterized constructor.

class Student4{

int id;

String name;

//creating a parameterized constructor

Student4(int i,String n){

id = i;

name = n;

}

//method to display the values

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

//creating objects and passing values

Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

//calling method to display the values of object

s1.display();

s2.display();

}

}

Output:

111 karan

222 aryan

**Constructor Overloading in Java**

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java

methods.

Constructor overloading in Java

is a technique of having more than one constructor with different parameter lists. They are arranged in a

way that each constructor performs a different task. They are differentiated by the compiler by the number

of parameters in the list and their types.

**Example of Constructor Overloading**

//Java program to overload constructors

class Student5{

int id;

String name;

int age;

//creating two arg constructor

Student5(int i,String n){

id = i;

name = n;

}

//creating three arg constructor

Student5(int i,String n,int a){

id = i;

name = n;

age=a;

}

void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){

Student5 s1 = new Student5(111,"Karan");

Student5 s2 = new Student5(222,"Aryan",25);

s1.display();

s2.display();

}

}

Output:

111 Karan 0

222 Aryan 2

**Java Inner Classes (or ) nested classes**

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

**Example:**

class OuterClass {

int x = 10;

class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass Outer = new OuterClass();

OuterClass.InnerClass Inner = Outer.new InnerClass();

System.out.println(Inner.y + Outer.x);

}

}

## Access Outer Class From Inner Class

One advantage of inner classes, is that they can access attributes and methods of the outer class

class OuterClass {

int x = 10;

class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.y + myOuter.x);

}

}

# **Java static keyword**

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly.

#### **Understanding the problem without static variable**

**class** Student{

**int** rollno;

     String name;

     String college="ITS";

}

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all [objects](https://www.javatpoint.com/object-and-class-in-java). If we make it static, this field will get the memory only once.

### Example of static variable

**class** Student{

**int** rollno;

   String name;

**static** String college ="ITS";//static variable

   //constructor

   Student(**int** r, String n){

   rollno = r;

   name = n;

   }

**void** display ()

{

System.out.println(rollno+" "+name+" "+college);}

}

**public** **class** TestStaticVariable1{

**public** **static** **void** main(String args[]){

 Student s1 = **new** Student(111,"Karan");

 Student s2 = **new** Student(222,"Aryan");

 s1.display();

 s2.display();

 }

}

Output:

111 Karan ITS

222 Aryan ITS

## Java static method

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

### Example of static method

**class** Student{

**int** rollno;

     String name;

**static** String college = "ITS";

**static** **void** change(){

     college = "BBDIT";

     }

     Student(**int** r, String n){

     rollno = r;

     name = n;

     }

**void** display(){System.out.println(rollno+" "+name+" "+college);}

}

**public** **class** TestStaticMethod{

**public** **static** **void** main(String args[]){

    Student.change();//calling change method

    //creating objects

    Student s1 = **new** Student(111,"Karan");

    Student s2 = **new** Student(222,"Aryan");

    Student s3 = **new** Student(333,"Sonoo");

    //calling display method

    s1.display();

    s2.display();

    s3.display();

    }

}

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

## Java static block

* Is used to initialize the static data member.
* It is executed before the main method at the time of classloading.

### Example of static block

**class** A2{

**static**{System.out.println("static block is invoked");}

**public** **static** **void** main(String args[]){

   System.out.println("Hello main");

  }

}

Output:static block is invoked

Hello main

**Final Keyword In Java**

The final keyword in java is used to restrict the user. The java final keyword can be used in many context.

**Final can be:**

* variable
* method
* class

The final keyword can be applied with the variables.

**1) Java final variable**

If you make any variable as final, you cannot change the value of final variable

final variable once assigned a value can never be changed.

**EXAMPLE:**

class Bike9{

final int x=”varma”;

void varma(){

x=”Raghu”;

}

public static void main(String args[]){

Bike9 obj=new Bike9();

obj.varma();

}

}

**Output:**

Compile time Error

**2) Java final method**

If you make any method as final, you cannot override it.

**Example :**

class Bike{

final void run(){System.out.println("running");}

}

class Honda extends Bike{

void run(){System.out.println("running safely with 100kmph");}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

}

Output:Compile Time Error

**3) Java final class**

If you make any class as final, you cannot extend it.

**Example :**

final class Bike{}

class Honda1 extends Bike{

void run(){System.out.println("running safely with 100kmph");}

public static void main(String args[]){

Honda1 honda= new Honda1();

honda.run(); \

}

}

Output:

Compile Time Error

**Super Keyword in Java**

Usage of Java super Keyword

* super can be used to refer immediate parent class instance variable.
* super can be used to invoke immediate parent class method.
* super() can be used to invoke immediate parent class constructor.

1. **super is used to refer immediate parent class instance variable.**

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

**EXAMPLE :**

class Animal{

String color="white";

}

class Dog extends Animal{

String color="black";

void printColor(){

System.out.println(color);//prints color of Dog class

System.out.println(super.color);//prints color of Animal class

}

}

class TestSuper1{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}

Output:

Black

White

**2) super can be used to invoke parent class method.**

The super keyword can also be used to invoke parent class method.

**EXAMPLE:**

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void eat(){System.out.println("eating bread...");}

void bark(){System.out.println("barking...");}

void work(){

super.eat();

bark();

}

}

class TestSuper2{

public static void main(String args[]){

Dog d=new Dog();

d.work();

}}

Output:

eating...

barking...

**3) super is used to invoke parent class constructor.**

The super keyword can also be used to invoke the parent class constructor.

**example:**

class Animal{

Animal(){System.out.println("animal is created");}

}

class Dog extends Animal{

Dog(){

super();

System.out.println("dog is created");

}

}

class TestSuper3{

public static void main(String args[]){

Dog d=new Dog();

}}

**Output:**

animal is created

dog is created

# Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.

### Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

1. **abstract** **void** printStatus();//no method body and abstract

Example of Abstract class that has an abstract method

**abstract** **class** Bike{

**abstract** **void** run();

}

**class** Honda4 **extends** Bike{

**void** run(){System.out.println("running safely");}

**public** **static** **void** main(String args[]){

 Bike obj = **new** Honda4();

 obj.run();

}

}

**Example:**

**abstract** **class** Bike{

   Bike(){System.out.println("bike is created");}

**abstract** **void** run();

**void** changeGear(){System.out.println("gear changed");}

 }

//Creating a Child class which inherits Abstract class

**class** Honda **extends** Bike{

**void** run(){System.out.println("running safely..");}

 }

//Creating a Test class which calls abstract and non-abstract methods

**class** TestAbstraction2{

**public** **static** **void** main(String args[]){

  Bike obj = **new** Honda();

  obj.run();

  obj.changeGear();

 }

}