Primitive data types in java

* **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large [arrays](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html), where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.
* **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.
* **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the [Integer](https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html) class to support the arithmetic operations for unsigned integers.
* **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 264-1. Use this data type when you need a range of values wider than those provided by int. The [Long](https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html) class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.
* **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the [java.math.BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html) class instead. [Numbers and Strings](https://docs.oracle.com/javase/tutorial/java/data/index.html) covers BigDecimal and other useful classes provided by the Java platform.
* **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
* **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
* **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the [java.lang.String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) class. Enclosing your character string within double quotes will automatically create a new String object; for example, String s = "this is a string";. String objects are *immutable*, which means that once created, their values cannot be changed. The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such. You'll learn more about the String class in [Simple Data Objects](https://docs.oracle.com/javase/tutorial/java/data/index.html)

Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable. If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it. Accessing an uninitialized local variable will result in a compile-time error.

### Literals

You may have noticed that the new keyword isn't used when initializing a variable of a primitive type. Primitive types are special data types built into the language; they are not objects created from a class. A literal is the source code representation of a fixed value; literals are represented directly in your code without requiring computation. As shown below, it's possible to assign a literal to a variable of a primitive type:

boolean result = true;

char capitalC = 'C';

byte b = 100;

short s = 10000;

int i = 100000;

#### Integer Literals

An integer literal is of type long if it ends with the letter L or l; otherwise it is of type int. It is recommended that you use the upper case letter L because the lower case letter l is hard to distinguish from the digit 1.

Values of the integral types byte, short, int, and long can be created from int literals. Values of type long that exceed the range of int can be created from long literals. Integer literals can be expressed by these number systems:

* Decimal: Base 10, whose digits consists of the numbers 0 through 9; this is the number system you use every day
* Hexadecimal: Base 16, whose digits consist of the numbers 0 through 9 and the letters A through F
* Binary: Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later)

For general-purpose programming, the decimal system is likely to be the only number system you'll ever use. However, if you need to use another number system, the following example shows the correct syntax. The prefix 0x indicates hexadecimal and 0b indicates binary:

// The number 26, in decimal

int decVal = 26;

// The number 26, in hexadecimal

int hexVal = 0x1a;

// The number 26, in binary

int binVal = 0b11010;

#### Floating-Point Literals

A floating-point literal is of type float if it ends with the letter F or f; otherwise its type is double and it can optionally end with the letter D or d.

The floating point types (float and double) can also be expressed using E or e (for scientific notation), F or f (32-bit float literal) and D or d (64-bit double literal; this is the default and by convention is omitted).

double d1 = 123.4;

// same value as d1, but in scientific notation

double d2 = 1.234e2;

float f1 = 123.4f;

#### Character and String Literals

Literals of types char and String may contain any Unicode (UTF-16) characters. If your editor and file system allow it, you can use such characters directly in your code. If not, you can use a "Unicode escape" such as '\u0108' (capital C with circumflex), or "S\u00ED Se\u00F1or" (Sí Señor in Spanish). Always use 'single quotes' for char literals and "double quotes" for String literals. Unicode escape sequences may be used elsewhere in a program (such as in field names, for example), not just in char or String literals.

The Java programming language also supports a few special escape sequences for char and String literals: \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), and \\ (backslash).

There's also a special null literal that can be used as a value for any reference type. null may be assigned to any variable, except variables of primitive types. There's little you can do with a null value beyond testing for its presence. Therefore, null is often used in programs as a marker to indicate that some object is unavailable.

Finally, there's also a special kind of literal called a class literal, formed by taking a type name and appending ".class"; for example, String.class. This refers to the object (of type Class) that represents the type itself.

## Using Underscore Characters in Numeric Literals

In Java SE 7 and later, any number of underscore characters (\_) can appear anywhere between digits in a numerical literal. This feature enables you, for example. to separate groups of digits in numeric literals, which can improve the readability of your code.

For instance, if your code contains numbers with many digits, you can use an underscore character to separate digits in groups of three, similar to how you would use a punctuation mark like a comma, or a space, as a separator.

The following example shows other ways you can use the underscore in numeric literals:

long creditCardNumber = 1234\_5678\_9012\_3456L;

long socialSecurityNumber = 999\_99\_9999L;

float pi = 3.14\_15F;

long hexBytes = 0xFF\_EC\_DE\_5E;

long hexWords = 0xCAFE\_BABE;

long maxLong = 0x7fff\_ffff\_ffff\_ffffL;

byte nybbles = 0b0010\_0101;

long bytes = 0b11010010\_01101001\_10010100\_10010010;

You can place underscores only between digits; you cannot place underscores in the following places:

* At the beginning or end of a number
* Adjacent to a decimal point in a floating point literal
* Prior to an F or L suffix
* In positions where a string of digits is expected

The following examples demonstrate valid and invalid underscore placements (which are highlighted) in numeric literals:

// **Invalid: cannot put underscores**

// **adjacent to a decimal point**

float pi1 = 3\_.1415F;

// **Invalid: cannot put underscores**

// **adjacent to a decimal point**

float pi2 = 3.\_1415F;

// **Invalid: cannot put underscores**

// **prior to an L suffix**

long socialSecurityNumber1 = 999\_99\_9999\_L;

// OK (decimal literal)

int x1 = 5\_2;

// **Invalid: cannot put underscores**

// **At the end of a literal**

int x2 = 52\_;

// OK (decimal literal)

int x3 = 5\_\_\_\_\_\_\_2;

// **Invalid: cannot put underscores**

// **in the 0x radix prefix**

int x4 = 0\_x52;

// **Invalid: cannot put underscores**

// **at the beginning of a number**

int x5 = 0x\_52;

// OK (hexadecimal literal)

int x6 = 0x5\_2;

// **Invalid: cannot put underscores**

// **at the end of a number**

int x7 = 0x52\_;

Certainly! **MS-DOS** (Microsoft Disk Operating System) commands are still valuable for computer professionals and provide the fundamentals of computing. These text-based commands allow users to perform various tasks, such as file management, system configuration, and program execution. Here’s a quick reference list of some essential MS-DOS commands along with their syntax:

1. **CD** (Change Directory):
   * Syntax: CD [drive:] [path]
   * Description: Changes the current directory to the specified folder.
2. **DIR** (Directory):
   * Syntax: DIR [drive:] [path]
   * Description: Displays a list of files and subdirectories in a directory.
3. **COPY**:
   * Syntax: COPY [source] [destination]
   * Description: Copies files from one location to another.
4. **DEL** (Delete):
   * Syntax: DEL [drive:] [path]
   * Description: Deletes one or more files.
5. **REN** (Rename):
   * Syntax: REN [drive:] [path] [filename1] [filename2]
   * Description: Renames a file or directory.
6. **MKDIR** (Make Directory):
   * Syntax: MKDIR [drive:] [path]
   * Description: Creates a new directory.
7. **RMDIR** (Remove Directory):
   * Syntax: RMDIR [drive:] [path]
   * Description: Removes an existing directory.
8. **TYPE**:
   * Syntax: TYPE [drive:] [path] filename
   * Description: Displays the contents of a text file.
9. **EDIT**:
   * Syntax: EDIT [drive:] [path] filename
   * Description: Opens the MS-DOS text editor for editing a specified file.
10. **CHKDSK** (Check Disk):
    * Syntax: CHKDSK [volume:] [/F] [/V] [/R] [/X] [/I] [/C] [/L [:size]] [/B]
    * Description: Scans and fixes errors on a disk.
11. **FORMAT**:
    * Syntax: FORMAT volume [/FS:file-system] [/V:label] [/Q] [/L [:size]] [/A:size] [/C] [/X]
    * Description: Prepares a storage medium for data storage.
12. **XCOPY**:
    * Syntax: XCOPY [source] [destination] [/E] [/C] [/H] [/R] [/Y]
    * Description: Copies files and directories, including subdirectories.
13. **TREE**:
    * Syntax: TREE [drive:] [path]
    * Description: Graphically displays the folder structure of a drive or path.
14. **DATE**:
    * Syntax: DATE [MM-DD-YYYY]
    * Description: Displays or sets the system date.
15. **TIME**:
    * Syntax: TIME [HH:MM:SS]
    * Description: Displays or sets the system time.
16. **HELP**:
    * Syntax: HELP [command]
    * Description: Provides help information for MS-DOS commands.
17. **EXIT**:
    * Syntax: EXIT
    * Description: Exits the MS-DOS command prompt or a batch file

Nested IF:

==========

One if works inside another if , in this situation both the condition must be true.

if(cond)

{

if(cond)

Statement;

}

Syntax: if else if

=========

if(cond)

Statement

else if (cond)

Statement

else if (cond)

statement

else

Statement'

Syntax – ===================

switch case:

The switch statement or switch case in java is a multi-way branch statement. Based on the value of the expression given, different parts of code can be executed quickly

Use of Switch Case

==================

The switch case in java is used to select one of many code blocks for execution.

Break keyword: As java reaches a break keyword, the control breaks out of the switch block. The execution of code stops on encountering this keyword, and the case testing inside the block ends as the match is found. A lot of execution time can be saved because it ignores the rest of the code's execution when there is a break.

Default keyword: The keyword is used to specify the code executed when the expression does not match any test case.

1: It uses int , char , string

2: break is important for it.

3: case must be uniue :

4: Logical operators can't work.

How Does the Switch Statement Work

===================================

The switch case in Java works like an if-else ladder, i.e., multiple conditions can be checked at once. Switch is provided with an expression that can be a constant or literal expression that can be evaluated. The value of the expression is matched with each test case till a match is found. If there is no match, the default keyword, if specified- the associated code executes. Otherwise, the code specified for the matched test case is executed.

switch(int, char, String)

{

case < case Identity> :

Statement;

break;

case < case Identity> :

Statement;

break;

case < case Identity> :

Statement;

break;

case < case Identity> :

Statement;

break;

default:

Statement;

Break;

}

=======================================================================================

Loop control Statement

Loop: cause a section of program to be executed rpeatedly while an expression is true.when the expression becomes false , the loop terminates and the control passed on to the satatement.

A loop consist of two segment

1: Control Statement

2 body of the loop.

There are four kinds of loop.

1: while

2:do while;

3: for

4:for each

We have to also work on Nested while, do while and for

Syntax of the loop;

1: Initialization expression:

the control variable must be initialized.

2: Test Expression

The value of the control variable is tested , and if found true , the body the loop executed. otherwise it gets teminated.

3: Update Expression

the control variable must be updated.

1: while(): it is kind of loop it has syntax similar to loop, initializaton, testexp, iteration or updation.

it first check the condition , if the condition is true it execute the statement under the loop. If the test expression false it exit from the loop

Syntax:

initialization

true

while(cond or test)

{

Statement;

updation or iteration;

}

//

++ = Increment operator Decrement operator --

post fix

a=1;

a++; or a=a+1;

a=2

a=4

a=3

a--; or a=a-1

y = a ++

y=1

a=2

prefix operations

case 2:

a=2

y=++a; or a=a+1

a=a+1;

y=3

a=3

Math class in java -----

[The **Java Math** class: provides methods for performing basic numeric operations, including elementary exponential, logarithm, square root, and trigonometric functions1](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html). Here are some commonly used methods from the Math class:

1. Math.max(x, y): Returns the highest value between x and y.

**Java**

int maxResult = Math.max(5, 10); // Result: 10

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. Math.min(x, y): Returns the lowest value between x and y.

**Java**

int minResult = Math.min(5, 10); // Result: 5

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. Math.sqrt(x): Returns the square root of x.

**Java**

double sqrtResult = Math.sqrt(64); // Result: 8.0

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. Math.abs(x): Returns the absolute (positive) value of x.

**Java**

double absResult = Math.abs(-4.7); // Result: 4.7

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. Math.random(): Generates a random number between 0.0 (inclusive) and 1.0 (exclusive).

**Java**

double randomNum = Math.random(); // Random value between 0.0 and 1.0

\*\*\*\* Loops in java ::

Certainly! In Java, there are three primary types of loops: **for loop**, **while loop**, and **do-while loop**. Let’s explore each of them:

1. **For Loop**:
   * The for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement combines initialization, condition checking, and increment/decrement in one line.
   * Syntax:

**Java**

for (initialization; testing condition; increment/decrement) {

// Statements

}

**While Loop**:

* A while loop is a control flow statement that repeatedly executes a set of instructions while a given Boolean condition remains true.
* Syntax:

**Java**

while (boolean condition) {

// Loop statements...

}

**Do-While Loop**:

* The do-while loop is similar to the while loop, but it checks the condition after executing the statements. It is an example of an exit-controlled loop.
* Syntax:

**Java**

do {

// Statements...

} while (condition);

# For-each loop in Java

Prerequisite: [Decision making in Java](https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/)  
For-each is another array traversing technique like for loop, while loop, do-while loop introduced in Java5. 

* It starts with the keyword **for** like a normal for-loop.
* Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
* In the loop body, you can use the loop variable you created rather than using an indexed array element.
* It’s commonly used to iterate over an array or a Collections class (eg, ArrayList)

**Syntax:**

for (type var : array)

{

statements using var;

}

**Simple program with for each loop:**

|  |
| --- |
| /\*package whatever //do not write package name here \*/    **import** java.io.\*;    **class** Easy    {    **public** **static** **void** main(String[] args)        {          // array declaration    **int** ar[] = { 10, 50, 60, 80, 90 };    **for** (**int** element : ar)                System.out.print(element + " ");      }  } |

**Output**

10 50 60 80 90

# Operators in Java

## What are the Java Operators?

Operators in Java are the symbols used for performing specific operations in Java. Operators make tasks like addition, multiplication, etc which look easy although the implementation of these tasks is quite complex.

## Types of Operators in Java

### **1. Arithmetic Operators**

They are used to perform simple arithmetic operations on primitive data types.

* **\* :**Multiplication
* **/ :**Division
* **% :**Modulo
* **+ :**Addition
* **– :**Subtraction

### **2. Unary Operators**

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

* **– :** **Unary minus**, used for negating the values.
* **+ :** **Unary plus** indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
* **++ :** **Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.   
  + **Post-Increment:**Value is first used for computing the result and then incremented.
  + **Pre-Increment:**Value is incremented first, and then the result is computed.
* **– –  : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.   
  + **Post-decrement:**Value is first used for computing the result and then decremented.
  + **Pre-Decrement: The value** is decremented first, and then the result is computed.
* **! : Logical not operator**, used for inverting a boolean value.

// Java Program to implement

// Uniary Operators

import java.io.\*;

// Driver Class

class GFG {

// main function

public static void main(String[] args)

{

// Interger declared

int a = 10;

int b = 10;

// Using uniary operators

System.out.println("Postincrement : " + (a++));

System.out.println("Preincrement : " + (++a));

System.out.println("Postdecrement : " + (b--));

System.out.println("Predecrement : " + (--b));

}

### } output :

Postincrement : 10

Preincrement : 12

Postdecrement : 10

Predecrement : 8

### **4. Relational Operators**

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable **relation\_operator** value

Some of the relational operators are-

* **==, Equal to**returns true if the left-hand side is equal to the right-hand side.
* **!=, Not Equal to**returns true if the left-hand side is not equal to the right-hand side.
* **<, less than:**returns true if the left-hand side is less than the right-hand side.
* **<=, less than or equal to**returns true if the left-hand side is less than or equal to the right-hand side.
* **>, Greater than:**returns true if the left-hand side is greater than the right-hand side.
* **>=, Greater than or equal to**returns true if the left-hand side is greater than or equal to the right-hand side.

### **3. Assignment Operator**

**‘=’** Assignment operator is used to assign a value to any variable. It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable **=** value;

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a **Compound Statement**. For example, instead of a **=** a+5, we can write a **+=** 5.

* **+=**, for adding the left operand with the right operand and then assigning it to the variable on the left.
* **-=**, for subtracting the right operand from the left operand and then assigning it to the variable on the left.
* **\*=**, for multiplying the left operand with the right operand and then assigning it to the variable on the left.
* **/=**, for dividing the left operand by the right operand and then assigning it to the variable on the left.
* **%=**, for assigning the modulo of the left operand by the right operand and then assigning it to the variable on the left.

### **5. Logical Operators**

These operators are used to perform “logical AND” and “logical OR” operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has “Logical NOT”, which returns true when the condition is false and vice-versa

Conditional operators are:

* **&&, Logical AND:**returns true when both conditions are true.
* **||, Logical OR:**returns true if at least one condition is true.
* **!, Logical NOT:**returns true when a condition is false and vice-versa

### **6. Ternary operator**

The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.

The general format is:

condition **?** if true **:** if false

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’

### **7. Bitwise Operators**

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

* **&, Bitwise AND operator:**returns bit by bit AND of input values.
* **|, Bitwise OR operator:**returns bit by bit OR of input values.
* **^, Bitwise XOR operator:**returns bit-by-bit XOR of input values.
* **~, Bitwise Complement Operator:**This is a unary operator which returns the one’s complement representation of the input value, i.e., with all bits inverted.

### **8. Shift Operators**

These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two. General format-

number **shift\_op** number\_of\_places\_to\_shift;

* **<<, Left shift operator:**shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two.
* **>>, Signed Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect to dividing the number with some power of two.
* **>>>, Unsigned Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

### **9. instanceof operator**

The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface. General format-

object **instance of** class/subclass/interface

// Java program to illustrate

// instance of operator

class operators {

public static void main(String[] args)

{

Person obj1 = new Person();

Person obj2 = new Boy();

// As obj is of type person, it is not an

// instance of Boy or interface

System.out.println("obj1 instanceof Person: "

+ (obj1 instanceof Person));

System.out.println("obj1 instanceof Boy: "

+ (obj1 instanceof Boy));

System.out.println("obj1 instanceof MyInterface: "

+ (obj1 instanceof MyInterface));

// Since obj2 is of type boy,

// whose parent class is person

// and it implements the interface Myinterface

// it is instance of all of these classes

System.out.println("obj2 instanceof Person: "

+ (obj2 instanceof Person));

System.out.println("obj2 instanceof Boy: "

+ (obj2 instanceof Boy));

System.out.println("obj2 instanceof MyInterface: "

+ (obj2 instanceof MyInterface));

}

}

class Person {

}

class Boy extends Person implements MyInterface {

}

interface MyInterface {

}

**Why did Java come to the market?**

Java programming language is mainly designed to overcome the disadvantages of c and C++ and to develop internet applications by achieving **platform independence.**  
  
  
**What is exactly Java?**

Java is a technology that provides two things

1. Programming language(JSE/core java3)
2. Platform (JVM)

By using the language we can develop java applications and also provides a runtime environment (JVM) where we execute java programs.  
  
  
As we know java comes in three different editions which allows us to develop three different types of applications, the three different editions are

1. JSE (java standard edition)(J2SE)
2. JEE (java enterprise edition)(J2EE)
3. JME (java micro edition)(J2ME)

These three different types of editions allow us to develop three different types of applications. They are as follows

1. Standalone/Desktop application
2. Enterprise / Distributed / Internet application
3. Device application  
     
     
   **What is a standalone application?**

An application installed in one system that can be executed from the same system but can’t be executed from another system is called a standalone application. The application developed for the context of one client is called a standalone application. JSE is used for developing standalone applications.

**What is an enterprise application?**

An application whose resources are shared among multiple clients is called distributed/ enterprise application. All internet applications are called distributed applications. These applications are present in the server system. JEE is used for developing internet applications.

There are two types of internet applications, such as

1. Web supportive application
2. Web application

An application that resides in the server system and that is downloaded and executed on the client computer by network call is called a web supportive application. An application that resides in the server system and that is executed directly in the server system via network call and send the response back to the

**What is the device application?**

JME means java mobile/micro edition. JME is used for developing device applications, such as mobile games tablets. JME is used for developing software for consumer electronics devices means embedded systems like mobile, it is popular for developing mobile gaming applications, a fully automated washing machine, robot

**What is a platform?**

A platform is an environment where we execute something. A platform is a hardware or software environment where a program runs. For example, a computer platform is( OS+ hardware devices)  
  
**What is Platform Dependent?**

The programs or applications complied on one platform (operating system) can be executed **on the same type of operating system**s but cannot be executed on other **types of operating systems is called platform-dependent applications.**

The programming languages which are used to develop such type of applications are called platform dependent programming language. C, C++ is called a platform-dependent programming language, because when C, C++ programs have complied, C, C++ compiler generates machine codes that are specific to the operating system.

**Drawbacks**: We need to develop the same application multiple times which increases the development cost and time.

**Solution**: Need to develop platform Independent application.  
  
  
**Platform Independent:**

If the applications complied code is able to run on different types of OS irrespective of the OS where it is compiled then such applications are called platform-independent applications.

The programming language which is used for developing such type of applications known as the platform-independent programming language.

Java is a platform-independent programming language because java program compiled code can execute (run) on any OS.

In this case, when the source code is compiled, the compiler generates intermediate code (byte code) rather than machine code, which is not specific to any OS. The intermediate code can be carried and executed on any machine provided software called JVM which is responsible for converting intermediate code into machine code  
  
  
**What is Byte Code?**

Java source code compiled ,code is called byte code. When the java code is compiled java compiler (javac) generates byte code (intermediate code).

1. Byte code is an
2. intermediate language (IL) code.
3. Byte code is a virtual machine code.
4. **Byte code is a platform-independent code.**
5. Byte code is the collections of mnemonics (MOVE, LOAD, ADD ETC), these byte codes are only understood by JVM.
6. Every byte code occupies 1 byte.

**What is thSimpleInte difference between byte code and machine code in Java?**

**Java Application Development Lifecycle  
  
  
Java Application Development Lifecycle:**

There are three core technology packages used in Java Programming: JDK, JVM, and JRE  
  
  
**JDK (Java Development Kit):**

The JDK (Java Development Kit) allows developers to create Java applications and applets that can be executed and run by the JVM and JRE. The JDK is a software package you download in order to create Java-based applications. The JDK is an implementation of the Java platform specification, including compiler and class libraries.

**The important features of JDK are:**

* It enables you to handle multiple extensions in a single catch block.
* JDK includes all features that JRE has.
* It contains development tools such as a compiler, debugger, etc.
* JDK provides the environment to develop and execute Java source code.
* It can be installed on Windows, Unix, and Mac operating systems.
* **JVM (Java Virtual Machine):**
* **JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn’t physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed**. It can also run those programs which are written in other languages and compiled to Java bytecode. It converts Java bytecode into machine language. JVM is a part of the Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE.
* The important features of JVM are :
* It enables you to run applications in a cloud environment or in your device.
* Java Virtual Machine converts byte code to machine-specific code.
* It provides basic java functions like memory management, security, garbage collection, and more.
* JVM runs the program by using libraries and files given by the Java Runtime Environment.
* JDK and JRE both contain Java Virtual Machine.
* It can execute the java program line by line hence it is also called an interpreter.
* JVM is easily customizable for example, you can allocate minimum and maximum memory to it.
* It is independent of hardware and the operating system. So, you can write a java program once and run it anywhere.  
    
    
  **Java Architecture**
* **Java Architecture** is a collection of components, i.e., **JVM, JRE,** and **JDK**. **It** integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. **Java Architecture** explains each and every step of how a program is compiled and executed.
* **Java Architecture** can be explained by using the following steps:
* There is a process of compilation and interpretation in Java.
* Java compiler converts the Java code into byte code.
* After that, the **JVM converts the byte code into machine code.**
* The machine code is then executed by the machine.

**Components of Java Architecture**

The Java architecture includes the three main components:

* Java Virtual Machine (JVM)
* Java Runtime Environment (JRE)
* Java Development Kit (JDK)

JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. **JVM is the one that actually calls the main method present in a java code**. JVM is a part of JRE(Java Runtime Environment).  
  
Java applications are called **WORA (Write Once Run Anywhere)**. This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM

**Class Loader Subsystem  
=====================**

It is mainly responsible for three activities.

**Loading**

**Linking**

**Initialization**

**Loading: The Class loader reads the “.class” file, generate the corresponding binary data and save it in the method area. For each “.class” file**

**, JVM stores the following information in the method area.**

The fully qualified name of the loaded class and its immediate parent class.

Whether the “.class” file is related to Class or Interface or Enum.

Modifier, Variables and Method information etc.

After loading the “.class” file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in java.lang package. These Class object can be used by the programmer for getting class level information   
For every loaded “*.class”* file, only **one** object of the class is created  
  
**Linking:** Performs verification, preparation, and (optionally) resolution. 

* *Verification*: **It ensures the correctness of the *.class* file i.e. it checks whether this file is properly formatted and generated by a valid compiler or not.** If verification fails, we get run-time exception *java.lang.VerifyError*. This activity is done by the component **ByteCodeVerifier**. Once this activity is completed then the class file is ready for compilation.
* *Preparation*: JVM allocates memory for class variables and initializing the memory to default values.
* *Resolution*: It is the process of replacing symbolic references from the type with direct references. It is done by searching into the method area to locate the referenced entity.
* **Initialization:** In this phase, **all static variables are assigned with their values defined in the code and static block(if any).** This is executed from top to bottom in a class and from parent to child in the class hierarchy.   
    
    
   **There are three class loaders**   
  *Bootstrap class loader*: Every JVM implementation must have a bootstrap class loader, capable of loading trusted classes. It loads core java API classes present in the “*JAVA\_HOME/jre/lib/rt.jar”* directory. This path is popularly known as the bootstrap path. It is implemented in native languages like C, C++.
* ***Extension class loader*:** It is a child of the bootstrap class loader. It loads the classes present in the extensions directories “*JAVA\_HOME/jre/lib/ext”*(Extension path) or any other directory specified by the java.ext.dirs system property. It is implemented in java by the *sun.misc.Launcher$ExtClassLoader* class.
* ***System/Application class loader***: It is a child of the extension class loader. It is responsible to load classes from the application classpath. It internally uses Environment Variable which mapped to java.class.path. It is also implemented in Java by the *sun.misc.Launcher$AppClassLoader* class.

: JVM follows the Delegation-Hierarchy principle to load classes. System class loader delegate load request to extension class loader and extension class loader delegate request to the bootstrap class loader. If a class found in the boot-strap path, the class is loaded otherwise request again transfers to the extension class loader and then to the system class loader. At last, if the system class loader fails to load class, then we get run-time exception java.lang.ClassNotFoundException.  
  
**JVM Memory**

1. **Method area:** In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
2. **Heap area:** Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.
3. **Stack area:** For every thread, JVM creates one run-time stack which is stored here. **Every block of this stack is called activation record/stack frame which stores methods calls**. **All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource**.
4. **PC Registers:** Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
5. **Native method stacks:** For every thread, a separate native stack is created. It stores native method information

**Execution Engine**

Execution engine executes the “.class” (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:

**Interpreter:** It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.

**Just-In-Time Compiler(JIT)** : It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.

Garbage Collector: It destroys un-referenced objects. For more on Garbage Collector, refer Garbage Collector.

Java Native Interface (JNI) :

It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

**Native Method Libraries :**

It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.