Date : 13th feb 2024

## **What is Java?**

* Developed By James Gosling at Sun Microsystems in 1995
* It is a Programming language and a Platform.
* Platform: any hardware or software env. Where program runs.
* Since Java has JRE (Java runtime env) and API . so it is a platform

##### **Types of Java Applications :**

1. **Standalone application :**

* Also known as Desktop app or window based app
* Traditional software that we need to install on our system
* Ex. : media player, antivirus

1. **Web application**

* Application that runs on server side and created dynamic page
* Ex. Servlets , JSP , Spring

1. **Enterprise Application**

* Application that is distributed in nature such as bank applications

1. **Mobile Application**

* Application created for mobile device
* Android and JAVA ME are used to create mobile applications

##### **Java Platforms / Editions :**

1. **Java SE (Java standard edition) :**

* Include java programming APIs such as java.lang, java.io, java.net , java.util etc
* Includes core topics like OOPs, String . Regex , Exception

1. **Java EE (Enterprise edition) :**

* Used to develop web and enterprise applications
* Build on top of Java SE platform
* Includes topics like Servlets , JSP, Web services etc.

1. **Java ME (Micro edition) :**

* Micro platform that is dedicated to mobile applications

1. **Java FX :**

* Used to develop rich internet applications
* Uses lightweight user interface API

## Features of Java :

1. **Simple :**

* Easy to learn
* Simple syntax . based on C++
* Clean and easy to understand
* Removed many complicated features like - explicit pointers and operator overloading
* Have automatic garbage collector

1. **Object-Oriented :**

* Object-oriented programming language i.e everything is an object
* Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.
* Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
* Basic concepts of OOPs are:
  + Object
  + Class
  + Inheritance
  + Polymorphism
  + Abstraction
  + Encapsulation

1. **Portable :**

* Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

1. **Platform independent :**

* Java is write once , run anywhere so it is not platform dependent
* C, C++ are complied into platform specific machines
* Java is software-based platform which runs on top of other hardware based platforms .
* Java has two components :
  + Runtime env.
  + API
* Java code is complied by compiler and converted into bytecode . this bytecode is platform-independent code

1. **Secured :**

* Java is best known for security
* Java can be used to developed virus free system
* Java is secured by bcz :
  + No explicit pointer
  + Java Programs run inside a virtual machine sandbox
  + **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
  + **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
  + **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

1. **Robust**

* Robust means strong
* It is robust bcz :
  + Uses string memory management
  + Lack of pointers
  + Automatic garbage collection which runs on JVM
  + Exception handling
  + Type checking mechanism

1. **Architecture neutral :**

* There is no implementation-dependent feature .
* Ex: size of primitive type is fixed
* In c int occupies 2 bytes incase of 32 bit architecture and 4 bytes in case of 64 bit architecture but Java occupies 4 bytes for both 32 and 64 bit

1. **Interpreted**
2. **High Performance :**

* Java is faster as java bytecode is close to native code
* Still little slower than compiler language like C++
* Since java is an interpreted language so it is slower than the compiler language

1. **Multithreaded :**

* Thread is separate program executing concurrently
* We can write Java programs that deal with many tasks at once by defining multiple threads.
* The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
* Threads are important for multi-media, Web applications, etc.

1. **Distributed :**

* Distributed because it facilitates user to create distributed applications
* RMI and EJB is used to crete distributed applications

1. **Dynamic :**

* Java supports dynamic loading of classes i.e clases are loaded on demand
* Java supports dynamic compilation and automatic memory management (garbage collection).

## **C++ VS Java**

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **C++** | **Java** |
| Platform independent | Platform dependent | Platform independent |
| Used for | For system programming | Application programming, window based , web-based, enterprise , mobile applications |
| Goto statement | Support Goto statement | Does not support Goto |
| Multiple inheritance | Supports Multiple inheritance | Does not support Multiple Inheritance |
| Operator overloading | Supports Operator overloading | Doesnot supports operator overloading |
| Pointers | Supports Pointers | Supports pointers internally . but cant write pointers in program explicitly Java restrict pointer support |
| Compiler & interpreter | Uses compiler only | Uses both compiler and interpreter |
| Flow | C++ is compiled and run using the compiler which converts source cod into machine code so it is platform dependent | Java source code is converted into bytecode at compilation time . The interpreter executes this bytecode at runtime and produces output. It is interpreted that is why it is platform-independent |
| Call by value and call by reference | Supports both | Supports only call by value |
| Structure and Union | Supports both | Supports NOne |
| Thread Support | Relies on Third-party libraries for thread support | Have build in thread support |
| Multi-comment / Documentation comment | Does not support | Support documentation comment (/\* \*/) |
| Virtual keyword | Supports virtual keyword so that we can decide whether or not to override a function | Has no virtual keyword . We can override all non static method by default , in other word , non-static methods are virtual by default |

14TH Feb 2024

## **First Java Program | Hello World Example**

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

##### **The requirement for Java Hello World Example :**

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

##### Install the JDK if you don't have installed it, download the JDK and install it.

##### Set path of the jdk/bin directory.

##### Create the Java program

##### Compile and run the Java program

Java Program :

**Class Test{**

**Public static void main (String argos[])**

**{**

**system.out.println(“Hello world”)**

**}**

**}**

##### 

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

**Parameters used in Java program :**

* **Class :** Keyword used to declare class
* **Public :** Access Modifier that represents visibility . it means it is visible to all
* **Static :** It is a keyword . If we declare any method as static then that method is called static method . Advantage if using static methos us that there is no need to create objects to invoke static method. Main() method is executed by JVM so it doesnot require creating an object to invoke the main method which saves memory
* **Void :** return type . thi doesnt return a value
* **Main :** represents starting point of the program
* **String args[] :** command line argument .\
* **System.out.println :** used to print statement in new line . System is class , out is object of Printstream class and println()is method os Printstream class .

**Valid Ways to write the parameters in program :**

**1) By changing the sequence of the modifiers, method prototype is not changed in Java.**

static public void main(String args[])

**2) The subscript notation in the Java array can be used after type, before the variable or after the variable.**

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

**3) You can provide var-args support to the main() method by passing 3 ellipses (dots)**

public static void main(String... args)

**NOTE :**

Variable Arguments (Varargs) in Java is a method that takes a variable number of arguments. Variable Arguments in Java simplifies the creation of methods that need to take a variable number of arguments.

Syntax of Varargs :

public static void fun(int ... a)

{

// method body

}

The … syntax tells the compiler that varargs have been used, and these arguments should be stored in the array referred to by a.

**4) Having a semicolon at the end of class is optional in Java.**

class A{

static public void main(String... args){

System.out.println("hello java4");

}

};

**5) Valid Java main() method signature**

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

public static void main(String... args)

static public void main(String[] args)

public static final void main(String[] args)

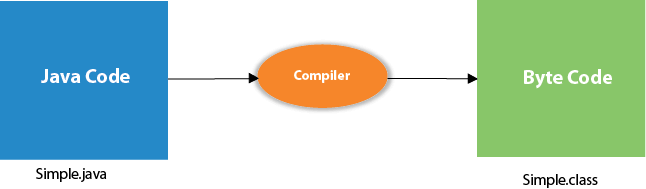
final public static void main(String[] args)

final strictfp public static void main(String[] args)

## **Internal Details of Hello Java Program**

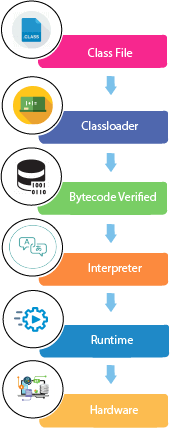
## What happens at compile time?

At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into bytecode.



## What happens at runtime?

At runtime, the following steps are performed:



**Classloader:** It is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.

**Interpreter:** Read bytecode stream then execute the instructions.

### Q) Can you save a Java source file by another name than the class name?

Yes, if the class is not public. It is explained in the figure given below:

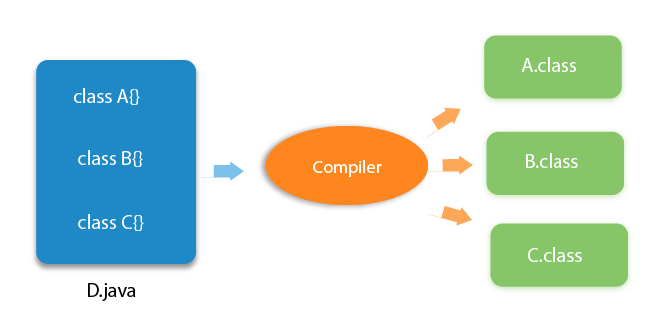


|  |  |
| --- | --- |
| **To compile:** | javac Hard.java |
| **To execute:** | java Simple |

Observe that, we have compiled the code with file name but running the program with class name. Therefore, we can save a Java program other than class name.

### Q) Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



# How to set path in Java

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

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

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:

* Open the command prompt
* Copy the path of the JDK/bin directory
* Write in command prompt: set path=copied\_path

### For Example:

set path=C:\Program Files\Java\jdk1.6.0\_23\bin

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

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

* 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

## **JDK, JRE, and JVM**

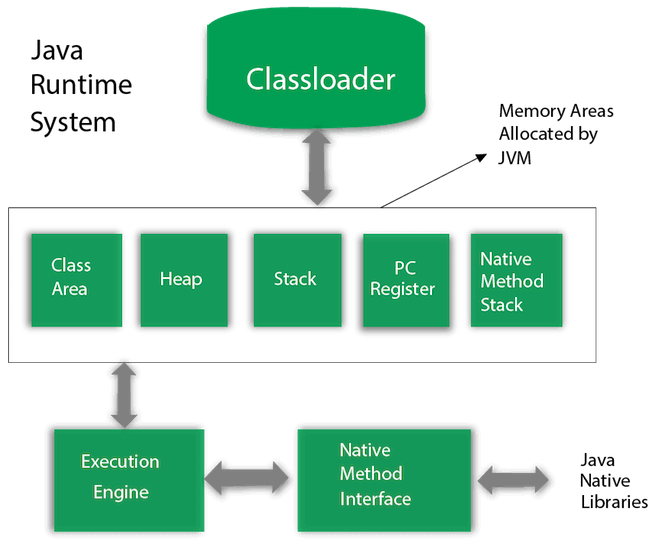
1. **JVM : Java Virtual Machine**

* Abstract machine
* Called virtual machine because it doesn't physically exist
* Provides runtime env. Where java bytecode is executed
* Runs those program also which are written in diff. Language and complied to Java bytecode
* JVM performs the following main task :
  + Loads the code
  + Verifies the code
  + Executes the code
  + Provides the runtime env.
* JVM is platform dependent
* It is:
  + A specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
  + An implementation Its implementation is known as JRE (Java Runtime Environment).
  + Runtime Instance Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.
* JVM provides the definition for the
  + Memory area
  + Class file format
  + Register set
  + Garbage collected Heap
  + Fatal error reporting

## 

## **JVM Architecture**

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



**A) Classloader**

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA\_HOME/jre/lib/ext directory.

System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

**B) Class(Method) Area**

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

**C) Heap**

It is the runtime data area in which objects are allocated.

**D) Stack**

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

**E) Program Counter Register**

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

**F) Native Method Stack**

It contains all the native methods used in the application.

**G) Execution Engine**

It contains:

A virtual processor

Interpreter: Read bytecode stream then execute the instructions.

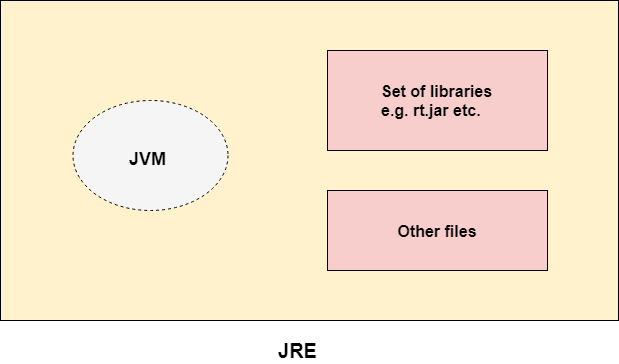
Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

**H) Java Native Interface**

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

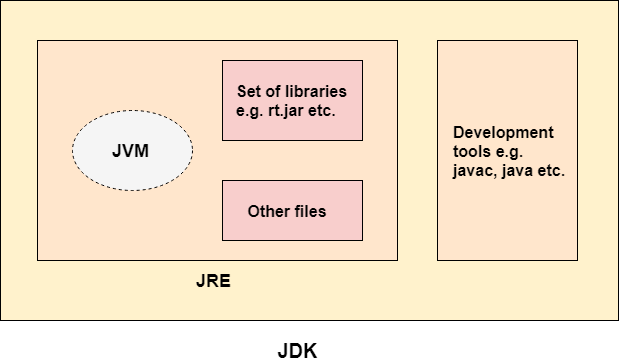
1. **JRE : Java Runtime Environment**

* Also written as Java RTE
* JRE is a set of software tools which are used for developing java applications
* It is the implementation of JVM , it physically exist
* Contains set of libraries + other files that JVM uses at runtime
* The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



1. **JDK : Java Development Kit**

* Software development env. Which is used to develop java applications and applets .
* Physically exist
* Contains JRE + development tools
* JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
  + Standard Edition Java Platform
  + Enterprise Edition Java Platform
  + Micro Edition Java Platform
* JDK contains private JVM and few other resources such as interpreter/loader(java) , compiler (javac) , archiver(jar) , document generator (javadoc) etc. to complete the development of java applications

****

## **Java Variable :**

* Variable is container that holds value while java program is executed
* Variable is assigned with Data type
* It is a name of memory location
* It is name of reserved area allocated in memory
* Ex : int a = 50 // here a is a variable

**Types of Variable :**

1. Local variables
2. Instance Variable
3. Static Variable
4. **Local variable :**

* Varibel declared inside the body of the method
* Can Use this variable within the method
* Other methods didnt know this variable exist int he method
* Cannot be defined as static

1. **Instance Variable:**

* Varibel declared inside the class but outside the body of the method
* Called instance variable bcz its value is instance- specific and is not shared among instances
* An object that is created using a class is said to be an instance of that class.
* Cannot declared Static

1. **Static variables :**

* Variable desclared as satic
* Cannot be local
* Can create single copy of static variable and share it among all the instance of class
* Momory allocations for static variables happens only once – when the class is loaded in memory

**EX :**

**public class A {**

**static int m=100;//static variable**

**void method()**

**{**

**int n=90;//local variable**

**}**

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

**{**

**int data=50;//instance variable**

**}**

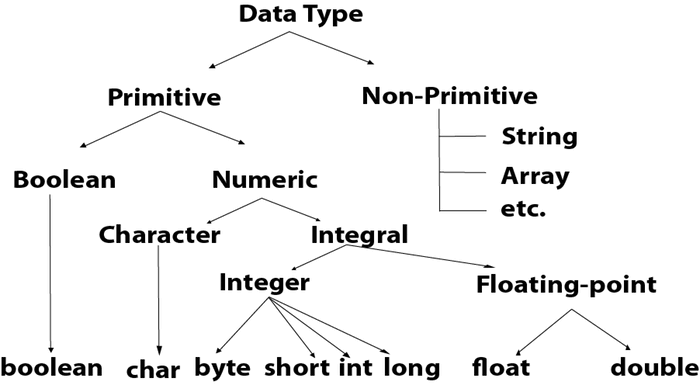
**}//end of class**

## **Data Types in Java :**

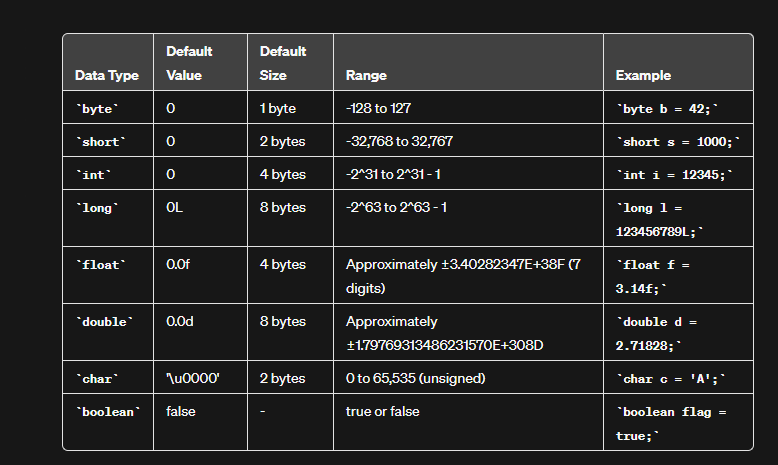
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

**NOTE : Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.**



1. **Primitive data types:**



Signed and unsigned are terms used to describe how numbers are represented in binary, indicating whether a number includes a sign (positive or negative) or is strictly positive.

**Signed:**In a signed number representation, a certain bit is used to represent the sign of the number (positive or negative).

For example, in a signed 8-bit representation, the range is typically from -128 to 127. The leftmost bit (most significant bit) is reserved for the sign (0 for positive, 1 for negative).

Signed 8-bit:

* 01111111 (127)
* 00000000 (0)
* 11111111 (-1)
* 10000000 (-128)

**Unsigned:**In an unsigned number representation, all bits are used to represent the magnitude of the number, and there is no dedicated sign bit.

For example, in an unsigned 8-bit representation, the range is from 0 to 255.

Unsigned 8-bit:

* 11111111 (255)
* 00000000 (0)

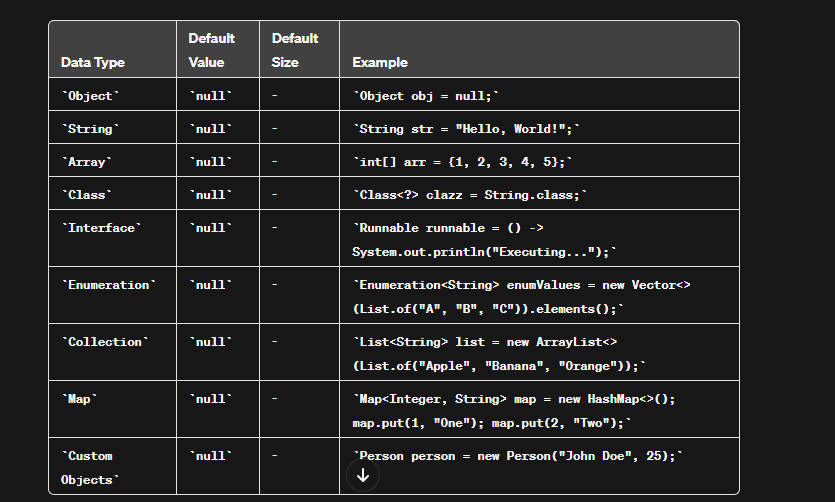
In Java, all primitive data types (except char) are signed. This means that the leftmost bit is used to represent the sign. For example, in a signed byte, the range is from -128 to 127. The char type, although 16 bits, is unsigned and represents Unicode characters ranging from 0 to 65,535.

### Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

1. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

* Java non-primitive data types, also known as reference types or objects,
* they have a default value of null



* In the above examples, the default value for all non-primitive types is null. The size of non-primitive types is not fixed and depends on the specific implementation and the data stored in them.

# Unicode System

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

### Why java uses Unicode System?

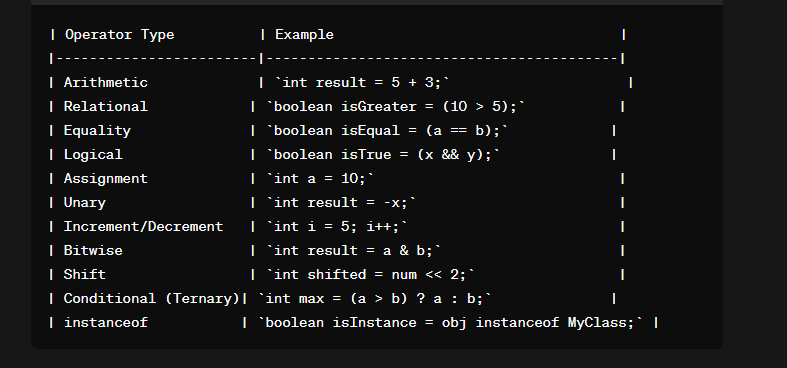
|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII** (American Standard Code for Information Interchange) for the United States. * **ISO 8859-1** for Western European Language. * **KOI-8** for Russian. * **GB18030 and BIG-5** for chinese, and so on. |

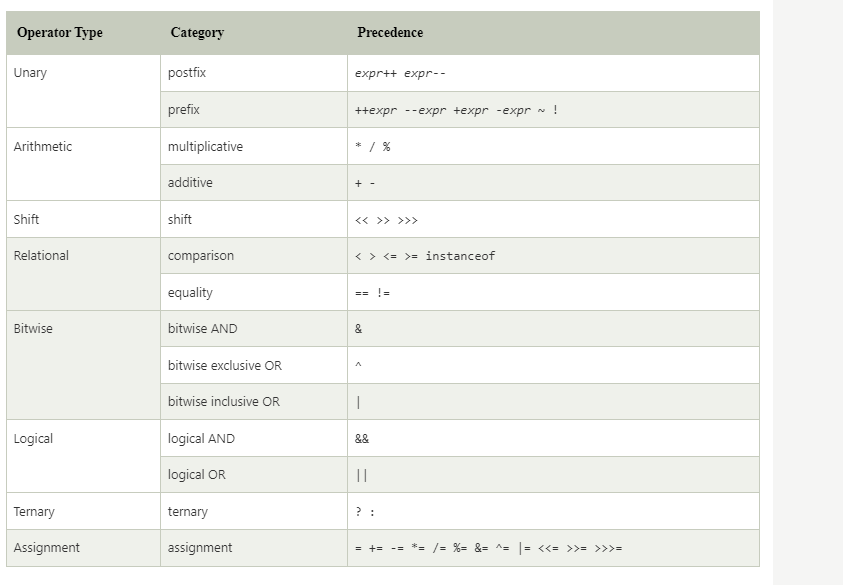
## Problem

|  |
| --- |
| **This caused two problems:**   1. A particular code value corresponds to different letters in the various language standards. 2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte. |

## Solution

|  |
| --- |
| To solve these problems, a new language standard was developed i.e. Unicode System. |
| In unicode, character holds 2 byte, so java also uses 2 byte for characters. |
| **lowest value:**\u0000 |
| **highest value:**\uFFFF Operators in Java **Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol that is used to perform operations. |





##### **Java Unary Operator**

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

* incrementing/decrementing a value by one
* negating an expression
* inverting the value of a boolean

Increment :

a = a+1 = a++ / ++a

Post and pre-increment:

Pre : ++a

* assign/change the value
* use the value

Ex : int a = 10 , b = 0 ;

B = ++a

sysout(a) // 11

sysout(b) // 11

Post : a++

* Use the value
* Change the value

Ex : int a = 10 , b = 0 ;

B = a++ // a = 11

sysout(a) // 11

sysout(b) // 10

Decrement :

A = a-1 = a - - /– - a

**Post and pre decrement :**

Pre : - - a

Int a = 10 , b = 0

B = - - a // a = 9 == b

Sysout (a) // 9

sysout(b) // 9

**Post : a - -**

* **Assing / use**
* **Change the value**

Int a = 10 , b = 0

B = a - - // b = 10 , a = 9

Sysout (a) // a = 9

sysout(b) // 10

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

**public** **class** OperatorExample{

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

**int** x=10;

System.out.println(x++);//10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}

**Output:**

10

12

12

10

### Java Unary Operator Example 2: ++ and --

1. **public** **class** OperatorExample{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. **int** b=10;
5. System.out.println(a++ + ++a);//10+12=22
6. System.out.println(b++ + b++);//10+11=21
8. }}

**Output:**

22

21

### Java Unary Operator Example: ~ and !

1. **public** **class** OperatorExample{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. **int** b=-10;
5. **boolean** c=**true**;
6. **boolean** d=**false**;
7. System.out.println(~a);//-11 (minus of total positive value which starts from 0)
8. System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);//false (opposite of boolean value)
10. System.out.println(!d);//true
11. }}

# Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

## List of Java Keywords

A list of Java keywords or reserved words are given below:

1. [**abstract**](https://www.javatpoint.com/abstract-keyword-in-java)**:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. [**boolean:**](https://www.javatpoint.com/boolean-keyword-in-java) Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. [**break**](https://www.javatpoint.com/java-break)**:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. [**byte**](https://www.javatpoint.com/byte-keyword-in-java)**:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. [**case**](https://www.javatpoint.com/case-keyword-in-java)**:** Java case keyword is used with the switch statements to mark blocks of text.
6. [**catch**](https://www.javatpoint.com/try-catch-block)**:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. [**char**](https://www.javatpoint.com/char-keyword-in-java)**:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. [**class**](https://www.javatpoint.com/class-keyword-in-java)**:** Java class keyword is used to declare a class.
9. [**continue**](https://www.javatpoint.com/java-continue)**:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. [**default**](https://www.javatpoint.com/default-keyword-in-java)**:** Java default keyword is used to specify the default block of code in a switch statement.
11. [**do**](https://www.javatpoint.com/java-do-while-loop)**:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. [**double**](https://www.javatpoint.com/double-keyword-in-java)**:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. [**else**](https://www.javatpoint.com/java-if-else)**:** Java else keyword is used to indicate the alternative branches in an if statement.
14. [**enum**](https://www.javatpoint.com/enum-in-java)**:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. [**extends**](https://www.javatpoint.com/inheritance-in-java)**:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. [**final**](https://www.javatpoint.com/final-keyword)**:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. [**finally**](https://www.javatpoint.com/finally-block-in-exception-handling)**:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. [**float**](https://www.javatpoint.com/float-keyword-in-java)**:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. [**for**](https://www.javatpoint.com/java-for-loop)**:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. [**if**](https://www.javatpoint.com/java-if-else)**:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. [**implements**](https://www.javatpoint.com/interface-in-java)**:** Java implements keyword is used to implement an interface.
22. [**import**](https://www.javatpoint.com/package)**:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. [**instanceof**](https://www.javatpoint.com/downcasting-with-instanceof-operator)**:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. [**int**](https://www.javatpoint.com/int-keyword-in-java)**:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. [**interface**](https://www.javatpoint.com/interface-in-java)**:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. [**long**](https://www.javatpoint.com/long-keyword-in-java)**:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. [**new**](https://www.javatpoint.com/new-keyword-in-java)**:** Java new keyword is used to create new objects.
29. [**null**](https://www.javatpoint.com/null-keyword-in-java)**:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. [**package**](https://www.javatpoint.com/package)**:** Java package keyword is used to declare a Java package that includes the classes.
31. [**private**](https://www.javatpoint.com/private-keyword-in-java)**:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. [**protected**](https://www.javatpoint.com/protected-keyword-in-java)**:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. [**public**](https://www.javatpoint.com/public-keyword-in-java)**:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. [**return**](https://www.javatpoint.com/return-keyword-in-java)**:** Java return keyword is used to return from a method when its execution is complete.
35. [**short**](https://www.javatpoint.com/short-keyword-in-java)**:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. [**static**](https://www.javatpoint.com/static-keyword-in-java)**:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. [**strictfp**](https://www.javatpoint.com/strictfp-keyword)**:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. [**super**](https://www.javatpoint.com/super-keyword)**:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. [**switch**](https://www.javatpoint.com/java-switch)**:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. [**synchronized**](https://www.javatpoint.com/synchronization-in-java)**:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. [**this**](https://www.javatpoint.com/this-keyword)**:** Java this keyword can be used to refer the current object in a method or constructor.
42. [**throw**](https://www.javatpoint.com/throw-keyword)**:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. [**throws**](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)**:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. [**try**](https://www.javatpoint.com/try-catch-block)**:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. [**volatile**](https://www.javatpoint.com/volatile-keyword-in-java)**:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. [**while**](https://www.javatpoint.com/java-while-loop)**:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

##### 