**What is Java ?**

Java is a programming language and a platform used to create software. It’s known for being high-level, strong, object-oriented, and secure.

Java was developed by Sun Microsystems in 1995, and James Gosling is often called the father of Java. Originally, it was named Oak, but they changed it to Java because Oak was already taken.

**Example of Hello Java**

class Simple{

public static void main(String args[]){

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

}

}

**Java can be used to create four main types of applications:**

Standalone Applications:

These are desktop programs that you install on your computer, like media players or antivirus software. Java uses AWT and Swing to create these applications.

Web Applications:

These run on a server and generate dynamic web pages. Technologies like Servlets, JSP, Struts, Spring, and Hibernate are commonly used for web apps in Java.

Enterprise Applications:

These are large, distributed systems, like banking software. They focus on high security, load balancing, and clustering. Java Enterprise Beans (EJB) is used to build these applications.

Mobile Applications:

These are designed for mobile devices. Java is used to create mobile apps mainly through Android and Java ME.

# **Java has four main platforms or editions:**

### Java SE (Standard Edition)

This is the basic Java platform that includes essential APIs for programming. It covers core concepts like object-oriented programming, strings, exceptions, multithreading, and more.

### Java EE (Enterprise Edition)

This platform is used for building large web and enterprise applications. It builds on Java SE and includes technologies like Servlets, JSP, and EJB for creating robust applications.

### Java ME (Micro Edition)

This edition is designed specifically for mobile applications, making it suitable for smaller devices.

### JavaFX

This platform is used to create rich internet applications with a focus on a modern and lightweight user interface.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

# **Features of Java**

Java was created to be portable, simple, and secure. In addition to these goals, it has several key features that contribute to its popularity, often referred to as "Java buzzwords." These features help make Java an attractive choice for developers.

A list of the most important features of the Java language is given below.



1. [Simple](https://www.javatpoint.com/features-of-java#Simple)
2. [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
3. [Portable](https://www.javatpoint.com/features-of-java#Portable)
4. [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
5. [Secured](https://www.javatpoint.com/features-of-java#Secured)
6. [Robust](https://www.javatpoint.com/features-of-java#Robust)
7. [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)
8. [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)
9. [High Performance](https://www.javatpoint.com/features-of-java#High-Performance)
10. [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)
11. [Distributed](https://www.javatpoint.com/features-of-java#Distributed)
12. [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)

### Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

### Object-oriented

Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java 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:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

### Platform Independent



Java is platform-independent, meaning you can write your code once and run it on any system without changes. Unlike languages like C or C++, which are compiled for specific machines, Java uses a "write once, run anywhere" approach.

A platform is the environment where a program runs, and there are two types: software-based and hardware-based. Java is a software-based platform that runs on top of various hardware platforms.

The Java platform consists of two main components:

1. Runtime Environment: This allows Java programs to run on different systems.

2. API (Application Programming Interface) : This provides a set of tools and libraries for developing Java applications.

When you write Java code, it gets compiled into bytecode, which is platform-independent. This bytecode can be executed on different operating systems like Windows, Linux, Solaris, and macOS, reinforcing the idea of "Write Once, Run Anywhere" (WORA).

### Secured

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

* **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.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

### Robust

The English mining of Robust is strong. Java is robust because:

* It uses strong memory management.
* There is a lack of pointers that avoids security problems.
* Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
* There are exception handling and the type checking mechanism in Java. All these points make Java robust.

### Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

### Portable

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

### High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

### Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

### Multi-threaded

A thread is like a 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.

### Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

### Creating Hello World Example

Let's create the hello java program:

1. **class** Simple{
2. **public** **static** **void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

Save the above file as Simple.java.

To compile: javac Simple.java

To execute: java Simple

**Output:**

*Hello Java*

**Compilation Flow:**

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.



## **Parameters used in First Java Program**

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

* **class** keyword is used to declare a class in Java.
* **public** keyword is an access modifier that represents visibility. It means it is visible to all.
* **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 doesn't require creating an object to invoke the main() method. So, it saves memory.
* **void** is the return type of the method. It means it doesn't return any value.
* **main** represents the starting point of the program.
* **String[] args** or **String args[]** is used for [command line argument](https://www.javatpoint.com/command-line-argument). We will discuss it in coming section.
* **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of [System.out.println()](https://www.javatpoint.com/system-out-println-in-java) statement in the coming section.

## **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.

### JVM

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.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](https://www.javatpoint.com/os-tutorial) is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a 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.



### JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It 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

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



# **JVM (Java Virtual Machine) Architecture**

1. [Java Virtual Machine](https://www.javatpoint.com/jvm-java-virtual-machine)
2. [Internal Architecture of JVM](https://www.javatpoint.com/jvm-java-virtual-machine#jvminternalarch)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

### What is JVM

It is:

1. **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.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

### What it does

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

## **JVM Architecture**

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



### 1) 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.

1. **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.
2. **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.
3. **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.
4. //Let's see an example to print the classloader name
5. **public** **class** ClassLoaderExample
6. {
7. **public** **static** **void** main(String[] args)
8. {
9. // Let's print the classloader name of current class.
10. //Application/System classloader will load this class
11. Class c=ClassLoaderExample.**class**;
12. System.out.println(c.getClassLoader());
13. //If we print the classloader name of String, it will print null because it is an
14. //in-built class which is found in rt.jar, so it is loaded by Bootstrap classloader
15. System.out.println(String.**class**.getClassLoader());
16. }
17. }

[Test it Now](javascript:void(0))

Output:

*sun.misc.Launcher$AppClassLoader@4e0e2f2a*

*null*

These are the internal classloaders provided by Java. If you want to create your own classloader, you need to extend the ClassLoader class.

### 2) Class(Method) Area

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

### 3) Heap

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

### 4) 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.

### 5) Program Counter Register

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

### 6) Native Method Stack

It contains all the native methods used in the application.

### 7) Execution Engine

It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **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.

### 8) 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.

# **Java Variables**

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](https://www.javatpoint.com/java-data-types): primitive and non-primitive.

## **Variable**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.



1. **int** data=50;//Here data is variable

### Types of Variables

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* 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.

#### **2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called an instance variable because its value is instance-specific and is not shared among instances.

#### **3) Static variable**

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

# **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).

## **Java Primitive Data Types**

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in [Java language](https://www.javatpoint.com/java-tutorial).

Java is a statically-typed programming language. It means, all [variables](https://www.javatpoint.com/java-variables) must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

* boolean data type
* byte data type
* char data type
* short data type
* int data type
* long data type
* float data type
* double data type



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

## **Boolean Data Type**

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:**

1. Boolean one = **false**

## **Byte Data Type**

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:**

1. **byte** a = 10, **byte** b = -20

## **Short Data Type**

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:**

1. **short** s = 10000, **short** r = -5000

## **Int Data Type**

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**Example:**

1. **int** a = 100000, **int** b = -200000

## **Long Data Type**

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

**Example:**

1. **long** a = 100000L, **long** b = -200000L

## **Float Data Type**

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:**

1. **float** f1 = 234.5f

## **Double Data Type**

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:**

1. **double** d1 = 12.3

## **Char Data Type**

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example:**

1. **char** letterA = 'A'

### 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.

# **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. |

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | expr++ expr-- |
| prefix | ++expr --expr +expr -expr ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

## **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. For example: +, -, \*, / etc.

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

* Unary Operator,
* Arithmetic Operator,
* Shift Operator,
* Relational Operator,
* Bitwise Operator,
* Logical Operator,
* Ternary Operator and
* Assignment Operator.

# **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.