**JAVA KEY CONCEPTS**

**History Of Java**

Java, developed by James Gosling and team at Sun Microsystems in 1991, was initially intended for digital devices. Java's original name was "Oak." It was named after an oak tree that stood outside James Gosling's office. However, due to trademark issues, the name was changed to "Java" in 1995.It was officially released in 1995 with the "write once, run anywhere" capability, quickly becoming popular for web applications. Key milestones include:

- 1995: Java 1.0 released.

- 1997: Introduction of Java 2 Platform (J2SE, J2EE, J2ME).

- 2004: Java 5 introduced major updates.

- 2006: Java became open-source.

- 2009: Oracle acquired Sun Microsystems.

- 2014: Java 8 introduced lambda expressions and the Stream API.

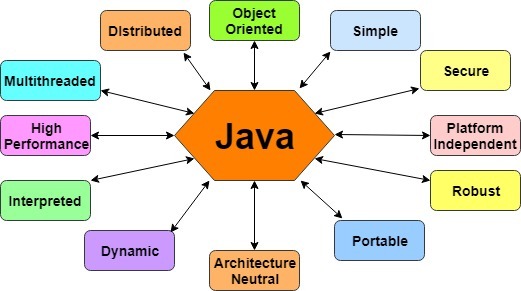
- 2018: Oracle began a six-month release cycle.

- 2021: Java 17 released as a long-term support version.

Java's robustness, cross-platform capability, and extensive ecosystem of libraries and frameworks have made it a cornerstone of modern software development, powering everything from mobile applications to large-scale enterprise systems. Its continuous evolution and adaptability to new technologies have ensured its relevance and dominance in the ever-changing landscape of programming languages.

**Features Of Java**

[Java programming](https://www.javatpoint.com/java-tutorial) language there are also some excellent features which play an important role in the popularity of this language.

****

* **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++ .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 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.

* **Platform Independent**

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

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

* **Robust**

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.

* **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.

* **Distributed**

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications.

* **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.

* **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 itsnative languages, i.e., C and C++.

**JVM Architecture**

Java Virtual Machine.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 . JVM (Java Virtual Machine) provides an environment to run Java code.

JRE deploys the all codes for creating a JVM. It will deploy the codes for creating JVM according to the operating system it is in. JVM alone can not execute Java byte code. For that, JVM required some runtime libraries. Along with JVM codes to deploy and those libraries, here comes JRE.



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.

* **Bootstrap ClassLoader**: It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, , java.io package classes, etc.
* **Extension 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.

**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: A virtual processor, Interpreter-Read bytecode stream then execute the instructions, compiler- It is used to improve the performance.

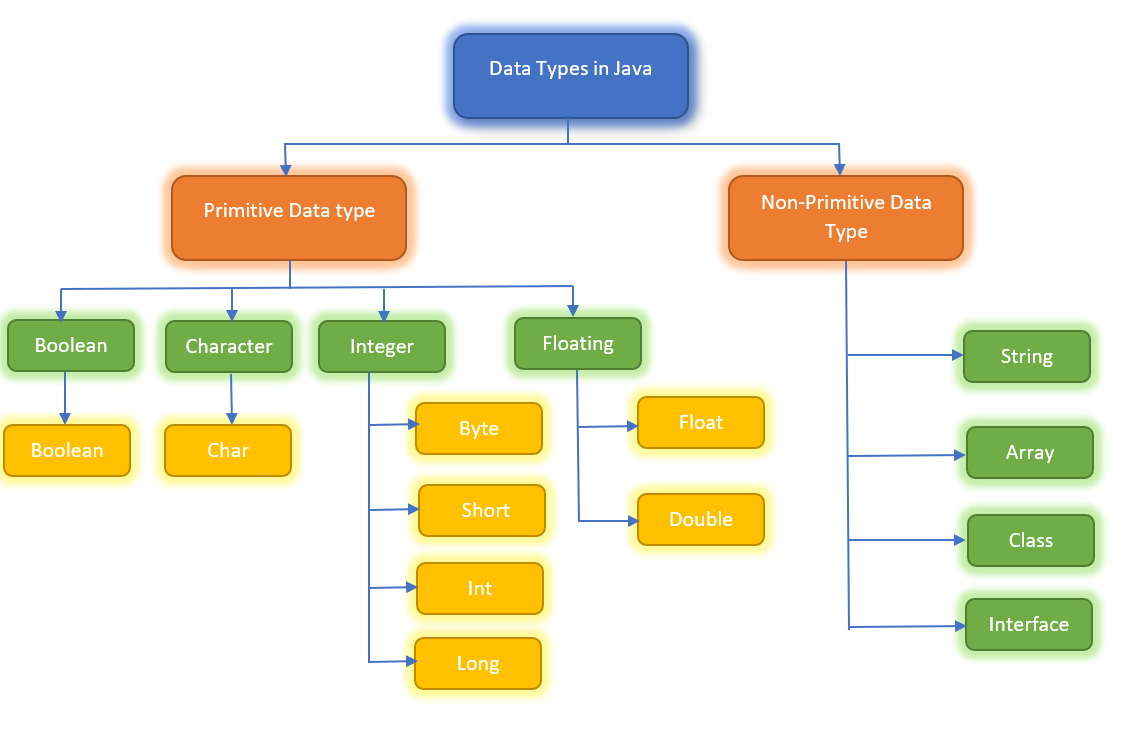
**8) Java Native Interface**

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.

**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:

* **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
* **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).



**1.Primitive data types**

**Boolean Data Type**

A Boolean data type comprises a single bit of information that can only store true or false values. True or false conditions are tracked using this data type, and Boolean data types are also used to store the result of various conditions.

**Byte Data Type**

The byte data type is an illustration of a primitive data type. It is a signed two's complement integer of 8 bits, and it stores whole numbers ranging from -128 to 127. A byte data type is useful for saving large amounts of memory.

**Char Data Type**

A single character is stored in this data type, and the character must be enclosed in single quotes, such as 'E' or 'e'. You can also use ASCII values to display specific characters.

**Short Data Type**

A short data type is larger than a byte but smaller than an integer, and it saves values ranging from -32,768 to 32767. This data type's default size is 2 bytes.

**Int Data Type**

This data type is capable of storing whole numbers ranging from -2147483648 to 2147483647. When creating variables with numeric values, int is generally the preferred data type.

**Long Data Type**

This data type is a two's complement 64-bit integer. A long data type's default size is 64 .

**Float Data Type**

It would be best to use a floating-point type when you need a number with a decimal, such as 8.88 or 3.14515.This data type supports fractional numbers ranging from 3.4e038 to 3.4e+038. It is important to note that the value should end with an "f."

**Double Data Type**

The double data type can store fractional numbers from 1.7e-308 to 1.7e+308. Note that you should end the value with a "d".

**2.Non-primitive**

* **Class and objects:**

A class in Java is a user defined data type i.e. it is created by the user. It acts a template to the data which consists of member variables and methods.

An object is the variable of the class, which can access the elements of class i.e. methods

* **Interface**

An interface is similar to a class however the only difference is that its methods are abstract by default i.e. they do not have body. An interface has only the final variables and method declarations. It is also called a fully abstract class.

* **String**

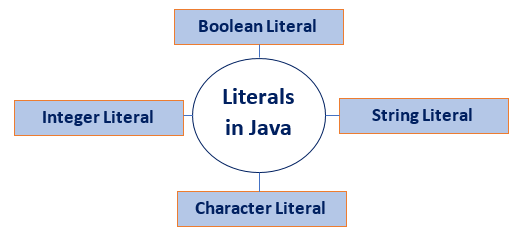
A string represents a sequence of characters for example "Javatpoint", "Hello world", etc. String is the class of Java.

* **Array**

An array is a data type which can store multiple homogenous variables i.e., variables of same type in a sequence. They are stored in an indexed manner starting with index 0. The variables can be either primitive or non-primitive data types.

**Literals in java**

literals refer to fixed values that are directly represented in the code. They can be of various types depending on the programming language and the context in which they are used. Here are some common types of literals:



1. **Integer literals:** These represent whole numbers. Examples:

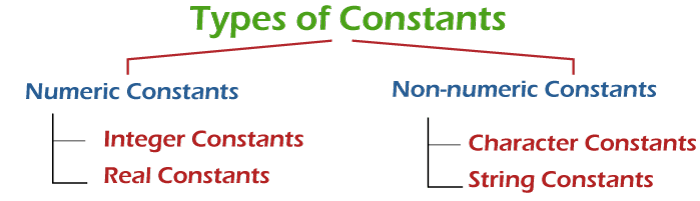
42 (decimal), 0x2A(hexadecimal), 052(octal), 0b101010 (binary)

2. **Character Literals:**A character literal is expressed as a character or an escape sequence, enclosed in a single quote ('') mark. It is always a type of char. For example, 'a', '%', '\u000d', etc.

1. **String Literals:** String literal is a sequence of characters that is enclosed between double quotes ("") marks. It may be alphabet, numbers, special characters, blank space, etc. For example, "Jack", "12345", "\n", etc.
2. **Floating Point Literals:** The vales that contain decimal are floating literals. In Java, float and double primitive types fall into floating-point literals. Keep in mind while dealing with floating-point literals.

**Constants in Java**

Like other programming language, Java also has some constants. In Java, constants are fixed values that do not change during the execution of a program. Constants are typically defined using the final keyword. Constants can be of any data type, including primitive types (such as int, float, char) and reference types (such as strings and objects).

****

**Non-numeric Constants**

A constant that does not contain digits is called non-numeric constants. There are the following two types of non-numeric constants:

**Character Constants:** A Character constant is a single alphabet, digit or any special symbol enclosed using single quotes. For example, 'Y', 'd', '6', '#', '&'.The maximum length of a character constant is 1 character long. It means that we cannot put more than one character inside single quotation marks.

**String Constants:** String constants consist of zero or more characters enclosed in double quotes (""). At the end of the string, the null character i.e '\0' is automatically placed by the compiler. For example, "hello", " " (denotes blank space), "111".

**Numeric Constants**

Numeric constants are the constants that contains numerals. It may also have a leading sign and decimal point.

**Integer Constants**

A constant that contains digits (0-9) and does not have decimal point is called integer constants. By default, it is type of int. examples –

Decimal Constants:It contains digits between 0 to 9. Note that must not start with 0. For example, 898, 67, 66.

Octal Constants**:** It contains digits between 0 to 7 and must begin with 0. For example, 012, 032, 067.

Hexadecimal Constants:It contains digits between 0 to 9 and letters a to f .It must begin with 0X or 0x. For example, 0x23, 0x76, 0X6A, 0XFF.

**Real Constants**

Numeric constants that have a decimal point are called real or floating-point constants. By default, the real constants are of double type. We can explicitly mention the type of a floating-point constant as a float by appending the letter f or F at the end of the constant. For example, 45f, -0.14f, 5.6F.

**Java Variables**

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.Variable is a name of memory location. There are three types of variables in java: local, instance and static.

**1) Local Variable**

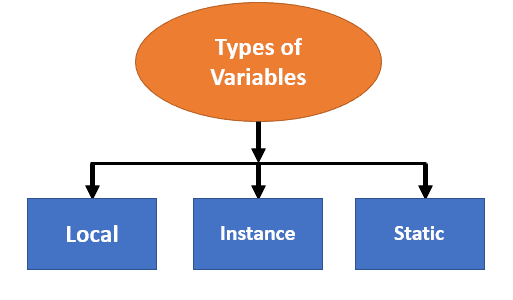
A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.

**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.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.



**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. Java contains a list of keywords They are listed below in the table with the primary action associated with them.

| **Keyword** | **Usage** |
| --- | --- |
| **abstract** | Specifies that a class or method will be implemented later, in a subclass |
| **assert** | Assert describes a predicate placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. |
| **boolean** | A data type that can hold True and False values only |
| **break** | A control statement for breaking out of loops. |
| **byte** | A data type that can hold 8-bit data values |
| **case** | Used in switch statements to mark blocks of text |
| **catch** | Catches exceptions generated by try statements |
| **char** | A data type that can hold unsigned 16-bit Unicode characters |
| **class** | Declares a new class |
| **continue** | Sends control back outside a loop |
| **default** | Specifies the default block of code in a switch statement |
| **do** | Starts a do-while loop |
| **double** | A data type that can hold 64-bit floating-point numbers |
| **else** | Indicates alternative branches in an if statement |
| **enum** | A Java keyword is used to declare an enumerated type. Enumerations extend the base class. |
| **extends** | Indicates that a class is derived from another class or interface |
| **final** | Indicates that a variable holds a constant value or that a method will not be overridden |
| **finally** | Indicates a block of code in a try-catch structure that will always be executed |
| **float** | A data type that holds a 32-bit floating-point number |
| **for** | Used to start a for loop |
| **if** | Tests a true/false expression and branches accordingly |
| **implements** | Specifies that a class implements an interface |
| **import** | References other classes |
| **instanceof** | Indicates whether an object is an instance of a specific class or implements an interface |
| **int** | A data type that can hold a 32-bit signed integer |
| **interface** | Declares an interface |
| **long** | A data type that holds a 64-bit integer |
| **native** | Specifies that a method is implemented with native (platform-specific) code |
| **new** | Creates new objects |
| **null** | This indicates that a reference does not refer to anything |
| **package** | Declares a Java package |
| **private** | An access specifier indicating that a method or variable may be accessed only in the class it’s declared in |
| **protected** | An access specifier indicating that a method or variable may only be accessed in the class it’s declared in (or a subclass of the class it’s declared in or other classes in the same package) |
| **public** | An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible) |
| **return** | Sends control and possibly a return value back from a called method |
| **short** | A data type that can hold a 16-bit integer |
| **static** | Indicates that a variable or method is a class method (rather than being limited to one particular object) |
| **strictfp** | A Java keyword is used to restrict the precision and rounding of floating-point calculations to ensure portability. |
| **super** | Refers to a class’s base class (used in a method or class constructor) |
| **switch** | A statement that executes code based on a test value |
| **synchronized** | Specifies critical sections or methods in multithreaded code |
| **this** | Refers to the current object in a method or constructor |
| **throw** | Creates an exception |
| **throws** | Indicates what exceptions may be thrown by a method |
| **transient** | Specifies that a variable is not part of an object’s persistent state |
| **try** | Starts a block of code that will be tested for exceptions |
| **void** | Specifies that a method does not have a return value |
| **volatile** | This indicates that a variable may change asynchronously |
| **while** | Starts a while loop |
| **sealed** | The sealed keyword is used to declare a class as “sealed,” meaning it restricts which classes can extend it. |
| **permits** | The permits keyword is used within a sealed class declaration to specify the subclasses that are permitted to extend it. |

**Java Comments**

The Java comments are the statements in a program that are not executed by the compiler and interpreter. Comments are used to make the program more readable by adding the details of the code. It makes easy to maintain the code and to find the errors easily. The comments can be used to provide information or explanation about the variable, method, class, or any statement.

**Types of Java Comments**

**1) Java Single Line Comment**

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.

Single line comments starts with two forward slashes (//). Any text in front of // is not executed by Java.

**2) Java Multi Line Comment**

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).

Multi-line comments are placed between /\* and \*/. Any text between /\* and \*/ is not executed by Java.

**3) Java Documentation Comment**

Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.

To create documentation API, we need to use the javadoc tool. The documentation comments are placed between /\*\* and \*/.