History of Java :

**Java** is an Object-Oriented programming language **developed by James Gosling in the early 1990s**. Java was created at [Sun Microsystems](https://www.britannica.com/money/Sun-Microsystems-Inc) .

Java was first released in 1995, and Java’s ability to provide interactivity and multimedia showed that it was particularly well suited for the Web.

[James Gosling](https://en.wikipedia.org/wiki/James_Gosling), Mike Sheridan, and [Patrick Naughton](https://en.wikipedia.org/wiki/Patrick_Naughton) initiated the Java language project in June 1991.

The language was initially called [*Oak*](https://en.wikipedia.org/wiki/Oak_(programming_language)) after an [oak](https://en.wikipedia.org/wiki/Oak) tree that stood outside Gosling's office. Later the project went by the name *Green* and was finally renamed *Java*, from [Java coffee](https://en.wikipedia.org/wiki/Java_coffee), a type of coffee from [Indonesia](https://en.wikipedia.org/wiki/Indonesia) .

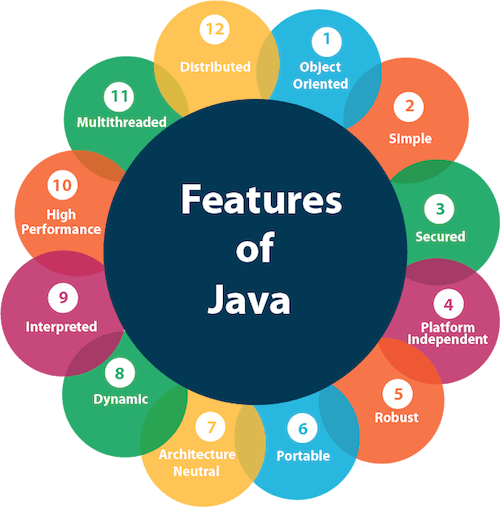
Sun Microsystems released the first public implementation as Java 1.0 in 1996.

The name Java originates from a sort of **espresso bean**, Java. Gosling came up with this name while having a coffee near his office. Java was created on the principles like **Robust, Portable, Platform Independent, High Performance, Multithread, etc.** and was called one of the **Ten Best Products of 1995** by the **TIME MAGAZINE**. Currently, Java is used in **internet programming, mobile devices, games, e-business solutions, etc**

Features :

Java is a feature rich language and with every new version, it is continously evolving. It is widely used across billions of devices. Following are the main features of the Java language -

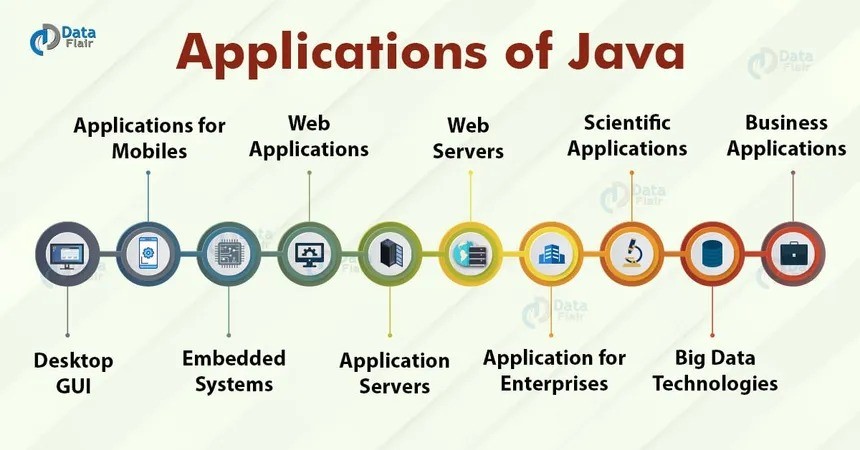
* [Object Oriented](https://www.tutorialspoint.com/java/java-features.htm#object-oriented)
* [Platform Independent](https://www.tutorialspoint.com/java/java-features.htm#platform-independent)
* [Simple](https://www.tutorialspoint.com/java/java-features.htm#simple)
* [Secure](https://www.tutorialspoint.com/java/java-features.htm#secure)
* [Architecture-neutral](https://www.tutorialspoint.com/java/java-features.htm#architecture-neutral)
* [Portable](https://www.tutorialspoint.com/java/java-features.htm#portable)
* [Robust](https://www.tutorialspoint.com/java/java-features.htm#robust)
* [Multithreaded](https://www.tutorialspoint.com/java/java-features.htm#multithreaded)
* [Interpreted](https://www.tutorialspoint.com/java/java-features.htm#interpreted)
* [High Performance](https://www.tutorialspoint.com/java/java-features.htm#high-performance)
* [Distributed](https://www.tutorialspoint.com/java/java-features.htm#distributed)
* [Dynamic](https://www.tutorialspoint.com/java/java-features.htm#dynamic)

[](https://www.tutorialspoint.com/java/java-features.htm#dynamic)

Applications:

**Top 10 Applications of Java** in the real world are :

* *Desktop GUI Applications*
* *Mobile Applications*
* *Artificial intelligence*
* *Web applications*
* *Big Data technology*
* *Gaming applications*
* *Business applications*
* *Embedded systems*
* *Cloud applications*
* *Scientific applications*



Comments in java :

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

* Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

// This is a comment   
System.out.println("Hello World");

Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

/\* The code below will print the words Hello World   
to the screen, and it is amazing \*/   
System.out.println("Hello World");

JVM (Java Virtual Machine) Architecture :

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

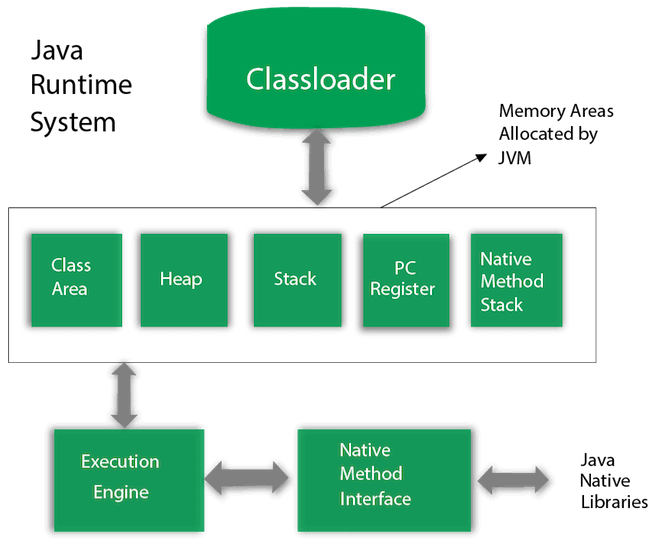
JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

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.

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

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

Literals in Java :

**Literal:** Any constant value which can be assigned to the variable is called literal/constant.

For Integral data types (byte, short, int, long), we can specify literals in 4 ways:-

**Decimal literals (Base 10):**In this form, the allowed digits are 0-9.

int x = 101;

**Octal literals (Base 8):**In this form, the allowed digits are 0-7.

// The octal number should be prefix with 0.   
int x = 0146;

**Hexa-decimal literals (Base 16):**In this form, the allowed digits are 0-9, and characters are a-f. We can use both uppercase and lowercase characters as we know that java is a case-sensitive programming language, but here java is not case-sensitive.

// The hexa-decimal number should be prefix   
// with 0X or 0x.   
int x = 0X123Face;

**Binary literals:**From 1.7 onward, we can specify literal value even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.

int x = 0b1111;

***Note :****By default, every literal is of int type, we can specify explicitly as long type by suffixed with l or L. There is no way to specify byte and short literals explicitly but indirectly we can specify. Whenever we are assigning integral literal to the byte variable and if the value is within the range of byte then the compiler treats it automatically as byte literals.*

Floating-Point literal

For Floating-point data types, we can specify literals in only decimal form, and we cant specify in octal and Hexadecimal forms.

**Decimal literals(Base 10):**In this form, the allowed digits are 0-9.

double d = 123.456;

***Note:****By default, every floating-point literal is of double type, and hence we cant assign directly to the float variable. But we can specify floating-point literal as float type by suffixed with f or F. We can specify explicitly floating-point literal as double type by suffixed with d or D. Of course this convention is not required.*

Char literals

For char data types, we can specify literals in 4 ways:

**Single quote:**We can specify literal to a char data type as a single character within the single quote.

char ch = 'a';

**Char literal as Integral literal:** we can specify char literal as integral literal, which represents the Unicode value of the character, and that integral literal can be specified either in Decimal, Octal, and Hexadecimal forms. But the allowed range is 0 to 65535.

char ch = 062;

**Unicode Representation:** We can specify char literals in Unicode representation ‘\uxxxx’. Here xxxx represents 4 hexadecimal numbers.

char ch = '\u0061';// Here /u0061 represent a.

**Escape Sequence:**Every escape character can be specified as char literals.

char ch = '\n';

String literals

Any sequence of characters within double quotes is treated as String literals.

String s = "Hello";

String literals may not contain unescaped newline or linefeed characters. However, the Java compiler will evaluate compile-time expressions, so the following String expression results in a string with three lines of text:

Boolean literals

Only two values are allowed for Boolean literals, i.e., true and false.

boolean b = true;

***Note:****When we are performing concatenation operations, then the values in brackets are concatenated first. Then the values are concatenated from the left to the right. We should be careful when we are mixing character literals and integers in String concatenation operations and this type of operation are known as****Mixed Mode operation****.* 