**History Of Java Programming Language**

Java is a programming language invented by James Gosling, Mike Sheridan, and Patrick Naughton, a team of Sun engineers known as the Green team in 1991. History of java versions starts with, The first public version, Java 1.0, was released in 1996 and could run on many devices for free.

* **1991**: Java is conceived by Sun Microsystems, originally named Oak.
* **1995**: Java 1.0 is released, introducing "Write Once, Run Anywhere" with the Java Virtual Machine (JVM).
* **1997-1999**: Java 2 introduces major features and rebrands into Java SE and Java EE.
* **2006**: Java 6 is released with performance improvements.
* **2008**: Oracle acquires Sun Microsystems.
* **2011-2014**: Java 7 and 8 introduce significant features like Lambdas and the new Date and Time API.
* **2017-2021**: Java 9-17 bring modularity, local variable type inference, and other enhancements. Java now follows a time-based release cycle with updates every six months.

# Java is a high-level, class-based, object-oriented programming language that was designed to have as few implementation dependencies as possible. Today, Java continues to be widely used in various domains, from web and mobile applications to enterprise systems and embedded devices.

# **Features of Java**

The primary objective of java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

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



### **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.

### **Platform Independent**

Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), 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**

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.

### **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++.

## Applications of Java

## What is Java Used For?

### **Mobile App Development**

The Java programming language can be considered as the official language for mobile application development. Most of the android applications build using Java. The most popular [android](https://www.javatpoint.com/android-tutorial) app development IDE [**Android Studio**](https://www.javatpoint.com/android-studio) also uses Java for developing android applications.

### **Desktop GUI Applications**

We can also develop a [GUI](https://www.javatpoint.com/gui-full-form) application using Java. Java provides [AWT](https://www.javatpoint.com/java-awt), [JavaFX](https://www.javatpoint.com/javafx-tutorial), and [Swing](https://www.javatpoint.com/java-swing) for developing the GUI based desktop application. The tools contain the pre-assembled components like list, menu, button.

### **Web-based Applications**

It is also used for developing the web-based application because it provides vast support for web development through [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), and [Struts](https://www.javatpoint.com/struts-2-tutorial). It is the reason that Java is also known as a server-side programming language. Using these technologies, we can develop a variety of applications. The most popular frameworks Spring, Hibernate, Spring Boot, used for developing web-based applications. **LinkedIn, AliExpress, web.archive.org, IRCTC,** etc. are the popular websites that are written using Java programming language.

### **Game Development**

Java is widely used by game development companies because it has the support of the open-source most powerful 3D engine. The engine provides unparalleled capacity when it comes to the context of the designing of 3D games. The most popular games developed in Java are [Minecraft](https://www.javatpoint.com/how-to-download-minecraft-java-edition), Mission Impossible III, etc. There are some popular Frameworks and Libraries available for Game Development, like - LibGDX and OpenGL.

### **Big Data Technology**

As many programming languages are available for Big Data Technology but still Java is the first choice for the same. The tool [Hadoop](https://www.javatpoint.com/hadoop-tutorial) [HDFS](https://www.javatpoint.com/hdfs) platform for processing and storing big data applications is written in Java. In big data, Java is widely used in [ETL](https://www.javatpoint.com/etl-testing) applications such as Apache Camel and Apache Kafka.

### **Distributed Applications**

The JINI (Java Intelligent Networking Infrastructure) provides the infrastructure to register and find distributed services based on its specification. It implements a mechanism that is known as JavaSpaces. It supports the distribution, persistence, and migration of objects in a network.

### **Cloud-Based Applications**

A cloud application is the on-demand availability of IT resources via the internet. The cloud-based application provides the service at a low cost. Java provides the environment to develop cloud-based applications. We can use Java to develop [SaaS (Software as a Service)](https://www.javatpoint.com/software-as-a-service), LaaS (Logging as a Service), and [PaaS (Platform as a Service)](https://www.javatpoint.com/platform-as-a-service).

### **IoT Application**

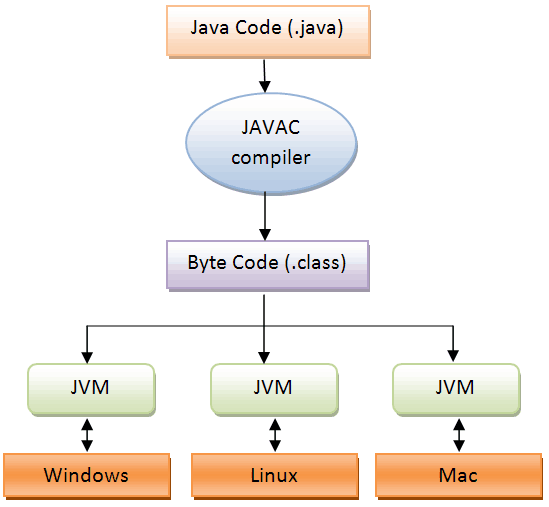
[IoT](https://www.javatpoint.com/iot-internet-of-things) is a technology that connects the devices in its network and communicates with them. IoT has found almost in all the small devices such as health gears, smartphones, wearables, smart lighting, TVs, etc. For developing the IoT application there is a lot of programming languages that can be used but Java offers an edge to developers that is unparalleled.

**Top of FormJVM (Java Virtual Machine) Architecture**

**Bottom of Form**

**Top of Form**

**Bottom of Form**



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).

Java is called platform independent because of JVM.

Components of JVM: Following are the components of JVM:

* Stack
* Heap
* Constant Pool
* Code segment
* Program counter

 **Stack**:

The JVM stack is responsible for storing frames, which hold local variables and partial results. Each thread has its own JVM stack, created simultaneously with the thread. This stack stores method calls and the data needed during method execution.

 **Heap**:

The heap is a shared runtime data area where objects are allocated. It is used for dynamic memory allocation for Java objects and class instances. The garbage collector manages the heap, reclaiming memory for objects that are no longer in use.

 **Constant Pool**:

The constant pool is a runtime storage area containing constants and symbolic references to classes, methods, and fields. Each class and interface has a constant pool, which stores literals and references used by the JVM during execution.

 **Code Segment**:

The code segment, also known as the method area, stores class structures such as the runtime constant pool, field and method data, and the code for methods and constructors. It is shared among all threads and is an essential part of the JVM's memory model.

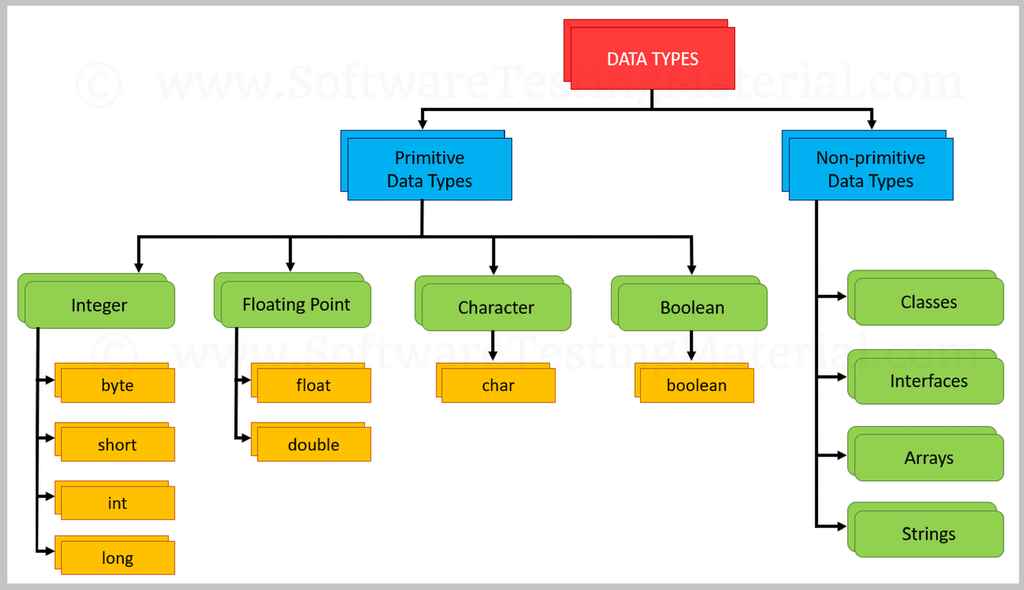
 **Program Counter (PC)**:

The program counter is a register in the JVM that keeps track of the address of the JVM instruction currently being executed. Each thread has its own program counter, which helps in multi-threaded execution by maintaining the current instruction's address separately for each thread.

# **Java Data Types**

Java is statically typed and also a strongly typed language because, in Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the Java data types.

here 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).

## Primitive Data Types in Java

Primitive data are only single values and have no special capabilities.  There are 8 primitive data types. They are depicted below in tabular format below as follows:

**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 is an 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:**

**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**

Long is a signed 64-bit type and is useful for those occasions where an int type is not large enough to hold the desired value. The range of a long is quite large. This makes it useful when big, whole numbers are needed.

**Example:**

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

**Float Data Type**

Float is specified as a single-precision value that uses 32 bits of storage. Single precision is faster and takes half as much space as double precision, but will become imprecise when the values are either very large or very small.

**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.

**Example:**

1. **double** d1 = 12.3

**Char Data Type**

Char in Java is not the same as char in C or C++. Java uses Unicode to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.

In Java, char is 16 bit and ranges from 0 to 65,536. There are no negative chars. The standard set of characters known as ASCII still ranges from 0 to 127 as always and the extended 8-bit character set, ISO-Latin-1, ranges from 0 to 255.

**Example:**

1. **char** letterA = 'A'

## Non-Primitive Data Type or Reference Data Types

The **Reference Data Types** will contain a memory address of variable values because the reference types won’t store the variable value directly in memory. They are strings, objects, arrays, etc.

### **1. Strings**

[Strings](https://www.geeksforgeeks.org/strings-in-java/) are defined as an array of characters. The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char-type entities. Unlike C/C++, Java strings are not terminated with a null character.

**Example:**   
String s1 = new String("GeeksforGeeks");

### **2. Class**

A [class](https://www.geeksforgeeks.org/classes-objects-java/) is a user-defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type.

### **3. Object**

An [Object](https://www.geeksforgeeks.org/classes-objects-java/) is a basic unit of Object-Oriented Programming and represents real-life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

### **4. Interface**

Like a class, an [interface](https://www.geeksforgeeks.org/interfaces-in-java/) can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

* Interfaces specify what a class must do and not how. It is the blueprint of the class.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

### **5. Array**

An [Array](https://www.geeksforgeeks.org/arrays-in-java/) is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. The following are some important points about Java arrays.

* In Java, all arrays are dynamically allocated. (discussed below)
* A Java array variable can also be declared like other variables with [] after the data type.
* The variables in the array are ordered and each has an index beginning with 0.

# **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.

Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold.

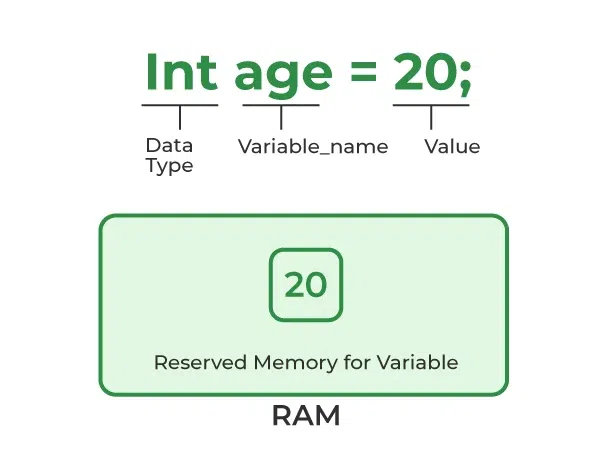
Variables in Java are only a name given to a memory location. All the operations done on the variable affect that memory location.

In Java, all variables must be declared before use.

**How to Initialize Variables in Java?**

It can be perceived with the help of 3 components that are as follows:

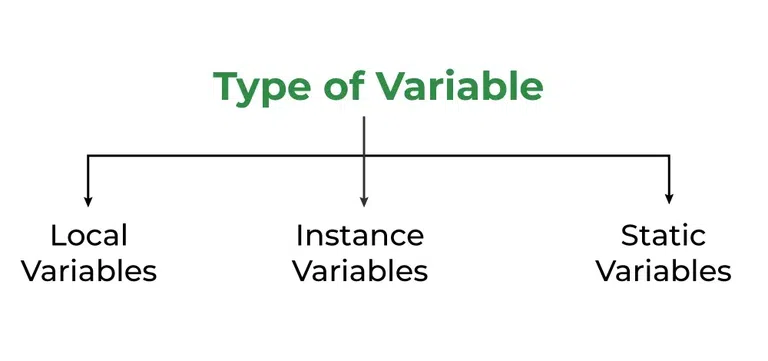
* **datatype**: Type of data that can be stored in this variable.
* **variable\_name**: Name given to the variable.
* **value**: It is the initial value stored in the variable.



### **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.

### **Example to understand the types of variables in java**

**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

# **Constants in Java**

A constant is a [**variable**](https://www.tutorialspoint.com/java/java_variable_types.htm) whose value **cannot change once it has been assigned**. Java doesn't have built-in support for constants.

A constant can make our program more easily read and understood by others. In addition, a constant is cached by the [**JVM**](https://www.tutorialspoint.com/What-is-Java-Virtual-Machine-JVM) as well as our application, so using a constant can improve performance.

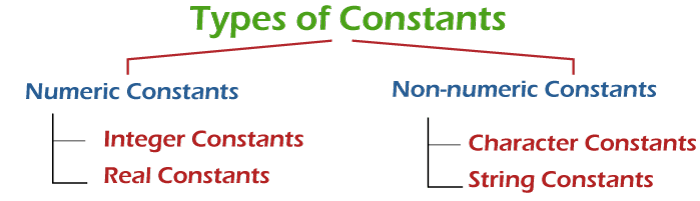
To [**define a variable as a constant**](https://www.tutorialspoint.com/what-is-a-constant-and-how-to-define-constants-in-java), we just need to add the keyword "**final**" in front of the variable declaration.

**Syntax:**

**static** **final** datatype identifier\_name=value;

For example:

**static** **final** **double** PI = 3.14;



### **Numeric Constants**

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

There are the following two types of numeric contestants:

### **1. 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**.

There are the following three types of integer constants:

* **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 (either in upper or lower case). It must begin with 0X or 0x. For example, **0x23, 0x76, 0X6A, 0XFF.**

### **2.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.

The real constants can be written in the following two forms:

* Fractional Form
* Exponential Form

### **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.

Java also supports backslash character constants. These are used in output methods. It is also known as **escape sequence**. For Example, \n, \t, \a, etc.

**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".**

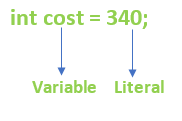
# **Literals in Java**

In [Java](https://www.javatpoint.com/java-tutorial), **literal** is a notation that represents a fixed value in the source code. In lexical analysis, literals of a given type are generally known as [**tokens**](https://www.javatpoint.com/java-tokens). In this section, we will discuss the term **literals in Java**.

## Literals

In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. Java has various types of literals.

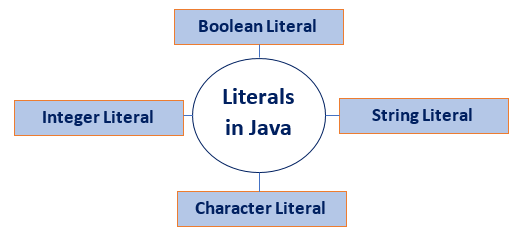
The following figure represents a literals.



**Types of Literals in Java**

There are the majorly **four** types of literals in Java:

1. Integer Literal
2. Character Literal
3. Boolean Literal
4. String Literal



### 1. Integer Literal

* **Definition**: Represents whole numbers without a fractional part.
* **Examples**: 42, 0, -15
* **Types**: Can be specified in decimal (base 10), octal (base 8, prefixed with 0), hexadecimal (base 16, prefixed with 0x or 0X), and binary (base 2, prefixed with 0b or 0B).
* **Usage**: Used for counting, indexing, and calculations that require whole numbers.

int decimal = 42; // decimal literal

int octal = 052; // octal literal

int hex = 0x2A; // hexadecimal literal

int binary = 0b101010; // binary literal

### **2. Character Literal**

* **Definition**: Represents a single Unicode character enclosed in single quotes.
* **Examples**: 'A', '1', '\n' (newline character), '\u0041' (Unicode representation for 'A')
* **Usage**: Used for characters in text processing and other operations where single characters are needed.

char ch1 = 'A';

char ch2 = '1';

char newline = '\n';

char unicodeChar = '\u0041'; // Unicode for 'A'

### **3. Boolean Literal**

* **Definition**: Represents one of two values: true or false.
* **Examples**: true, false
* **Usage**: Used in conditional statements, loops, and other control structures to manage program flow based on logical conditions.

boolean isJavaFun = true;

boolean isFishTasty = false;

### **4. String Literal**

* **Definition**: Represents a sequence of characters enclosed in double quotes.
* **Examples**: "Hello, World!", "12345", "@#&!"
* **Usage**: Used for text processing, displaying messages, and storing any kind of text data.

String greeting = "Hello, World!";

String number = "12345";

String specialChars = "@#&!";