Data Types

⇒ Java as a Strongly Typed Language:

In Java, every variable has a type, every expression has a type, and all types are strictly defined. Moreover, every assignment is checked by the compiler for type compatibility. For this reason, Java is considered a strongly typed programming lang

⇒ Data Type - Definition:

A data type in programming defines:

1) The kind of data a variable can store (eg. integer, character, floating-point number, object etc).
2) the range of values it can hold.
3) The operations that can be performed on it
4) The amount of memory allocated for storing that data.

In Java, data types are divided into:

* primitive types → predefined by the language (eg. int, double, boolean).
* Reference types → store references to objects (eg. strings, arrays, user-defined classes).

Primitive Data Types (8)

① Numerical Data Types
(to represent numbers)

- **Integral Types** (Whole Numbers)
  - byte
  - short
  - int
  - long
- **Floating Point Types** (Real Numbers)
  - float
  - double

② char Data Type
(to represent characters)

* char stores a single 16-bit Unicode character.

③ boolean DataType
(to represent logical values)

* boolean has only two possible values: true or false

* All numeric types in Java are signed, except char (which is unsigned) — because they can be represented as both "+ve" & "-ve" numbers.
* boolean is not numeric, so signed/unsigned doesn't apply.

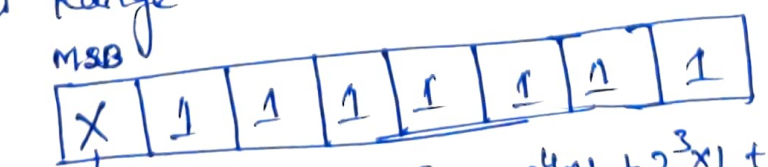
Integral Data Types

① Byte ⟹ size = 1byte (8 bits)

Max Value = +127
Min Value = -128
Range = -128 to 127 [-2^7 to 2^7-1]

The first bit (most significant bit) is the sign bit.

Sign Bit



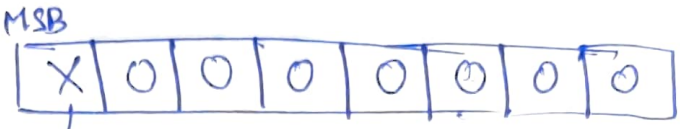
$= 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1$

$= 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= 127 \Rightarrow 01111111$

$\Rightarrow 0b1111111$

* 0 → positive no. (+ve)
* 1 → negative no. (stored in two's complement form)



Sign bit = 1 (negative)

$= 2^7 \times (-1) + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 0 \ldots + 2^0 \times 0$

$= -128 + 0 + 0 + 0 \ldots + 0$

$= -128 \Rightarrow 10000000$ (-0b10000000)

* Negative numbers are stored in two's complement form, which means
  1) Invert all bits of the positive value
  2) Add 1 to the result.

Eg: -5 in an 8 bit byte → start from 00000101 (5),
invert → 11111010, add 1 → 11111011.
So, if the sign bit is 1, the binary is interpreted as a negative value.

* Positive numbers are stored directly in memory.

Eg:

byte b = 10; → works fine.

byte b2 = 130; → Compile-time error: possible loss of precision
  ↳ found : int
  ↳ required : byte

byte b3 = 10.5; → C.E: possible loss of precision.

byte b4 = false; → CE: incompatible types

byte b5 = "Praveen"; → CE: incompatible types
  ↳ found : java.lang.String
  ↳ required : byte.

Best Case Use: The byte type is best when handling raw binary data, especially in:

* File operations (reading/writing bytes)
* Network data transfer (streams)

Analogy: Think of byte as the smallest box in which you can pack your data when shipping (file or network). If you use bigger boxes (int, long), you waste space & bandwidth.

- **short:**
  * size: 2 bytes (16 bit)
  * Range: -32768 to 32767 (-2^{15} to $2^{15}-1$)
  * Usage: Rarely used in modern Java programming

Eg:

```
short s = 130;     ⟶ works fine
short s2 = 32768;  ⟶ CE: possible loss of precision
short s3 = true;   ⟶ CE: incompatible types.
```

- **Why it's Rare:**
  * Originally useful for 16-bit processors (like Intel 8086) to save memory.
  * Modern processors are 32-bit or 64 bit, making short less relevent.
  * Now mostly replaced by int for general integer use.

Tip: Use short only when memory optimization is critical (eg: large arrays on embedded systems)

③ int :

* Size : 4 bytes (32 bits)
* Range : -2147483648 to 2147483647
  $(-2^{31}$ to $2^{31}-1)$
* Usage : Most commonly used integer type in Java.

Eg:

```
int i = 130;  ⟶ Works fine
int i2 = 10.5;  ⟶ CE: possible loss of precision
int i3 = true;  ⟶ CE: incompatible types.
```

- Why it's popular:
  * Default data type for integers (unless specified otherwise)
  * Offers a good balance b/w range & memory usage.
  * Efficiently handled by modern 32bit & 64bit processors.

Tip: Use int for most-number calculations unless:
↳ You need very large numbers (long).
↳ You need to save memory in huge array (byte or short).

④ long:

* size : 8 bytes (64 bits)
* Range : -2^{63} to $2^{63}-1$
* Usage : Used when int is not sufficient to store large integer values.

Eg: Counting characters in a large file

```
long l = f.length();  ⟶ f is a File Object
```

↳ File.length() returns long because very large files can exceed the range of int.

Key points:

* Ideal for very large numeric values.
* Commonly used in file sizes, timestamps, & counters that can exceed 2 billion.
* Still represents whole numbers only.

Note:

* All the above data types - byte, short, int & long
  ↳ store whole numbers (integers)
* For real numbers (decimal values), use floating-point data types (float, double).

Floating-Point Data Types

Feature	float	double
1) Accuracy	5-6 decimal places	14-15 decimal places
2) Size	4 bytes	8 bytes
3) Range	-3.4e38 to 3.4e38	-1.7e308 to 1.7e308
4) Precision	Single Precision	Double Precision
5) Usage	When memory is critical & moderate precision is enough	when high precision is required for calculations.

Eg: float f = 3.14f; → "f" suffix needed for float literals.
double d = 3.14159265; → double is default for decimal literals.

Note:

1) float is memory efficient but less precise → suitable for graphics, sensor data & scenarios where approximate values are fine.

2) double is default for decimal numbers in Java & used in financial, scientific, & mathematical computations requiring high precision.

boolean Data Type

* size: Not precisely defined in Java (JVM dependent), typically 1 bit conceptually but stored as 1 byte (or) more internally).
* Range: Not Applicable – can only be true (or) false.
* Usage: To store logical values and control program flow in conditional statements & loops.

Eg:

```
boolean b = true;      } valid
boolean b1 = false;

boolean b = TRUE;      → CE: cannot find symbol.
boolean b = "True";    → CE: incompatible types.
boolean b = 0;         ——"——
boolean b = 1;         ——"——
```

Eg 1: if condition

```
int x = 0;
if (x) {                → CE: incompatible types (int → boolean)
    S.O.P ("hello");
} else {
    S.O.P ("hi");
}
```

In Java, the condition inside if must be strictly boolean, unlike C/C++ where non-zero integers are treated as true.

Eg 2: while loop

```
while(1){ ⟶ // CE: incompatible types (int → boolean)
    S.O.P("hello");
}
```

// The condition must be:

```
while(true){
    S.O.P("hello");
}
```

⇒ Java enforces type safety → integers cannot be implicitly converted to booleans. This helps avoid many logical errors common in C/C++.

char Data Type:

* size: 2 bytes (16 bits)
* Range: 0 to 65535 (\u0000 to \uFFFF)
* Usage: stored a single unicode character.

Why size is 2 Byte in Java:

* In older languages like C & C++, characters were based on the ASCII system (≤ 256 characters), so 1 byte was enough.
* Java is Unicode-based, allowing characters from all world languages (more than 256 but ≤ 65,536 characters).
* Hence, Java's char requires 2 bytes to represent all possible Unicode characters.

Eg:

```
char ch1 = 97;      → valid, stores 'a' (ASCII 97)
char ch2 = 'A';     → valid
char ch3 = '\u0905'; → valid, store '3I' (Devanagiri letter A).
char ch4 = 65536;   → invalid
                      ↳ CE: possible loss of precision.
```

```
S.O.P (ch1);      → prints 'a'
S.O.P ((int) ch1); → prints 97.
```

⇩

char stores numeric Unicode values under the hood.

Summary of Data types:

data type	size	Default Value
byte	1 byte	0
short	2 bytes	0
int	4 bytes	0
long	8 bytes	0L
float	4 bytes	0.0f
double	8 bytes	0.0d
boolean	NA	false
char	2 bytes	0 or '\u0000' (represents blank space)

→ object references have a default value of null.
→ Numeric literals default to int (whole numbers) & double (decimal numbers) unless suffixed with L, f or d.
→ boolean doesn't store numbers like in c/c++
↳ only true or false.

Java Numeric Data Types Summery

Data Type (default Value)	Size	Precision/ Accuracy	Usage
1) byte (0)	1 byte (8 bits)	Whole no's only.	For raw binary data, file I/O, network streams.
2) short (0)	2 bytes (16 bits)	—"—	Rarely used; was for 16 bit processors.
3) int (0)	4 bytes (32 bits)	—"—	Most Common for integers.
4) long (0L)	8 bytes (64 bits)	—"—	For very large integers (file sizes, timestamps).
5) float (0.0f)	4 bytes (32 bits)	~5 to 6 decimal places	When memory is important, moderate decimal accuracy
6) double	8 bytes (64 bits)	~14 to 15 decimal places.	Default for decimal no's; high precision.