Java - Data Types

Java is a strongly typed programming language that requires you to declare the type of each variable before using it. Understanding the various data types available in Java is essential for efficient memory usage and proper program execution.

Java Data Types – Notes

What is a Data Type?

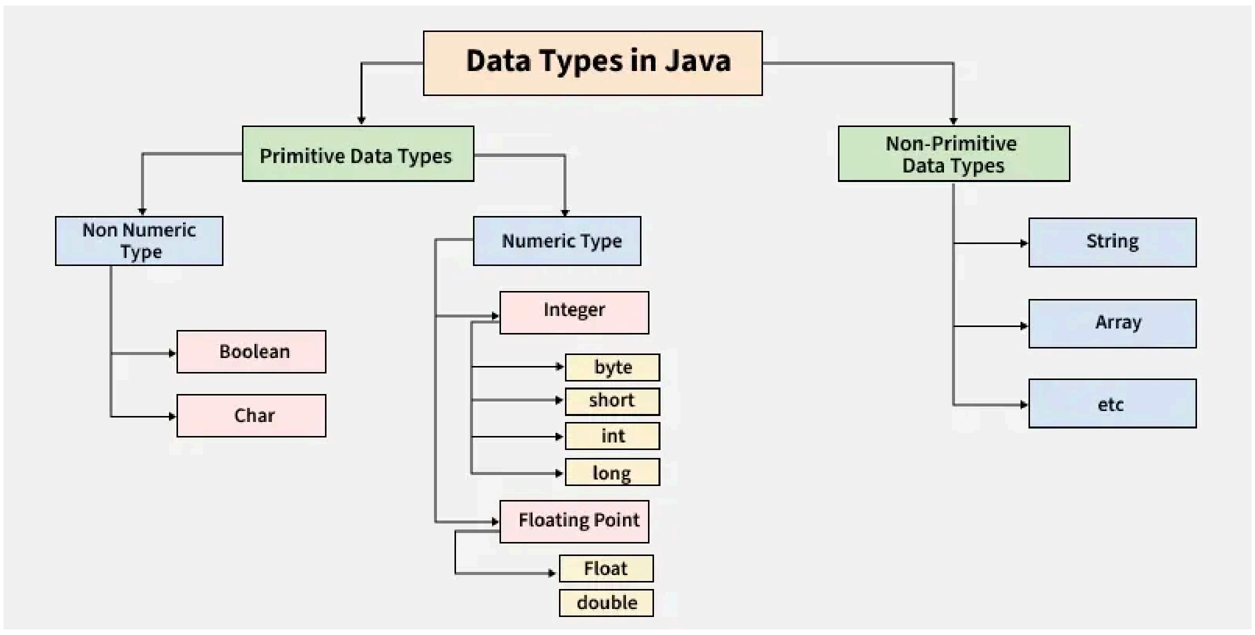
A data type is the classification of data that specifies the kind of value stored (like number, text, or true/false) and which operations are valid on it. It tells the computer how to interpret and handle the value.

A **data type** defines:

- What type of value a variable can store
- The size of memory required
- The range of values it can hold
- The operations allowed on that data

Java has **two categories of data types**:

1. **Primitive Data Types** (basic, built-in, 8 types)
2. **Non-Primitive Data Types** (reference types like String, Arrays, Classes, etc.)



1. Primitive Data Types (8 types)

A primitive data type is a basic, built-in type provided by a programming language that stores a single, simple value and is not composed of other types.

It directly represents simple values like numbers, characters, or true/false, typically with a fixed size and predefined operations.

Java provides 8 built-in primitive data types:

Data Type	Size (Bytes)	Range	Example
byte	1	-128 to 127	`byte b = 100;`
short	2	-32,768 to 32,767	`short s = 20000;`
int	4	-2,147,483,648 to 2,147,483,647	`int n = 100000;`
long	8	Very large values (~9 quintillion)	`long l = 9999999999L;`
float	4	6–7 decimal digits	`float f = 12.34f;`
double	8	15–16 decimal digits	`double d = 123.456789;`
char	2	Single 16-bit Unicode character	`char c = 'A';`
boolean	1 bit	true or false	`boolean flag = true;`

👉 **Quick Trick to Remember (8 types):**

big **s**mart **i**ntelligent **l**ions **f**ight **d**angerous **c**razy **b**ulls

(byte, short, int, long, float, double, char, boolean)

2. Non-Primitive (Reference) Data Types

- A non-primitive data type is a user-defined or built-in composite type whose variables hold references to objects/collections, enabling methods and richer structure.

- These types can be null, can vary in size, and typically provide operations via methods or interfaces.
- Created by **programmers** or provided by Java libraries.
- Store references, not direct values; memory is allocated on the heap for the actual object.
- Support methods/behaviors and can model complex entities, collections, and relationships.
- Size is not fixed like primitives; instances can grow/shrink (e.g., collections).

Examples:

1. **String** → `"Hello Java"`
2. **Array** → `{1, 2, 3, 4}`
3. **Class / Object** → custom user-defined types

In Java, a **Class** is a blueprint or template that defines the structure and behavior of objects. It contains:

- Fields (variables) that store data
- Methods that define behavior
- Constructors that initialize objects
- Other components like nested classes and interfaces

An **Object** is an instance of a class - a real-world entity created from the class blueprint. When you create an object, you're:

- Allocating memory for that specific instance
- Initializing its state (data)
- Allowing it to perform operations defined by its methods

Classes are fundamental to object-oriented programming in Java, allowing you to create custom user-defined types beyond the primitive data types. They enable you to model real-world entities and their relationships in your code.

4. **Interface**

An Interface in Java is a blueprint of a class that specifies what a class must do (but not how). It's a collection of abstract methods and constants. Some key characteristics of interfaces:

- They contain only method signatures and constants (no implementation)
- A class implements an interface to inherit its abstract methods
- Interfaces enable multiple inheritance in Java
- They help achieve abstraction and loose coupling.

🔷 Example Program (Primitive Data-Types)

```
public class DataTypeExample {
    public static void main(String[] args) {
        byte b = 100;
        short s = 20000;
        int i = 100000;
        long l = 10000000000L;
        float f = 12.5f;
        double d = 123.456;
        char c = 'A';
        boolean bool = true;

        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("char: " + c);
        System.out.println("boolean: " + bool);
    }
}
```

✅ Key Points

- Data Types = Define what kind of value can be stored.
- **Integer values** → `byte, short, int, long`
- **Decimal values** → `float, double`
- **Character** → `char`
- **True/False** → `boolean`
- **Reference/Object types** → `String, Array, Classes, Interface`

Difference Between Primitive and Non-Primitive Data Types in Java

Feature	Primitive Data Types	Non-Primitive Data Types
Definition	The most basic, built-in data types in Java.	Reference types (objects) created by programmers or provided by Java libraries.
Examples	**`byte, short, int, long, float, double, char, boolean`**	**`String, Array, Class, Interface`**

Feature	Primitive Data Types	Non-Primitive Data Types
Size	Fixed size (depends on type, e.g., int = 4 bytes, double = 8 bytes)	No fixed size (depends on object, may use more memory)
Stored Value	Stores the **actual value** (e.g., number 10, true/false)	Stores the **reference (memory address)** of the object
Default Value	`0` for numbers, `false` for boolean, `'\u0000'` for char	`null`
Operations	Can perform arithmetic directly (`+`, `-`, `*`)	Cannot do arithmetic directly; need methods/functions
Null Allowed?	❌ Cannot be null (they always have a value, default or assigned)	✅ Can be null (means no object is referenced)
Speed	Faster (works at low-level, directly with memory)	Slower (because object handling requires more steps)
Inheritance	Cannot inherit (not objects)	Can be used with inheritance & polymorphism
Example Code	`int age = 20;`	`String name = "Rahul";`

Memory Representation

- **Primitive:** Directly stores value in memory.

  Example:

  int x = 10;

  Memory: [10]

- **Non-Primitive:** Stores reference (address), which points to the actual object in memory.

  Example:

  String s = "Java";

  Memory: [address -⟶ "Java"]