Java Data Types

Java is a statically typed language, so every variable is explicitly declared with a type before being used in the program. The type system in Java ensures that each variable operates only on values of the declared data type, providing strict type checking and data type safety.

1. Built-in Data Types in Java

Java's data types are categorized into two main groups:

Category	Data Types
Primitive	byte, short, int, long, float, double, char, boolean
Non-Primitive	String, arrays, classes, interfaces, enums

2. Java Data Types: Use, Range, and Examples

a. Numeric Types

- **byte:** 8-bit signed integer. Range: -128 to 127.
  - Example: `byte x = 100;`
- **short:** 16-bit signed integer. Range: -32,768 to 32,767.
  - Example: `short y = 10000;`
- **int:** 32-bit signed integer. Range: -2,147,483,648 to 2,147,483,647.
  - Example: `int z = 123456;`
- **long:** 64-bit signed integer. Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
  - Example: `long l = 123456789L;`
- **float:** 32-bit floating-point. Approx. ±3.4 × 10^38 (about 6-7 decimal digits).
  - Example: `float f = 3.14f;`
- **double:** 64-bit floating-point. Approx. ±1.7 × 10^308 (about 15 decimal digits).
  - Example: `double d = 2.71828;`

b. Boolean Type

- **boolean:** Only two possible values: `true` or `false`.
  - Example: `boolean flag = true;`

c. Character Type

- **char:** 16-bit Unicode character. Range: 0 to 65,535.
  - Example: `char c = 'A';`

d. Non-Primitive Types

- **String:** Sequence of characters (object, not primitive).
  - Example: `String s = "Hello";`
- **Array:** Collection of variables of the same type.
  - Example: `int[] nums = {1, 2, 3};`
- **Classes, Interfaces, Enums:** Complex structures and blueprints for creating objects.

3. Accepting User Input for Specific Data Types

Java uses the `Scanner` class (from `java.util`) for runtime input:

a. Reading Basic Input

```
Scanner sc = new Scanner(System.in);
String name = sc.nextLine();  // accepts string
```

b. Accepting Integer Input

```
int age = sc.nextInt();
```

c. Accepting Float Input

```
float salary = sc.nextFloat();
```

d. Accepting Boolean Input

```
boolean isStudent = sc.nextBoolean();  // expects "true" or "false"
```

e. Accepting Array Input

```
int[] items = new int[5];
for (int i = 0; i < 5; i++) {
   items[i] = sc.nextInt();
}
```

f. Accepting Character Input

```
char ch = sc.next().charAt(0);
```

5. Mutability and Immutability

- **Primitive types** are immutable (changing the value actually replaces the value, it is like constant).
- **String** is immutable.
- Arrays and user-defined objects can be mutable or immutable, depending on their definition.

6. Practical Importance

- Correct data typing enforces program safety, predictability, and performance.
- Type mismatches result in compile-time errors, making Java programs more reliable.

7. Summary Table

Data Type	Example	Mutable?	Conversion Example	Typical Use
byte	100	No	Byte.parseByte("12")	Small-range numbers
short	10000	No	Short.parseShort("200")	Medium-range numbers
int	123456	No	Integer.parseInt("456")	General numeric variables
long	123456789L	No	Long.parseLong("700000")	Large-range numbers
float	3.14f	No	Float.parseFloat("2.5")	Decimal values (approximate)
double	2.71828	No	Double.parseDouble("1.618")	High-precision decimal values
char	'A'	No		Single characters
boolean	true	No	Boolean.parseBoolean("true")	Logical values
String	"hello"	No	String.valueOf(32)	Text/string data
int[]	{1,2,3}	Yes		Array of integers
Object	new Person()	Yes/No		Custom classes, complex data structures

Type Casting in Java

Type casting in Java is the process of converting a variable from one data type to another. It is used to enable assignments or operations between different data types, especially when working with primitive data types or objects of related types.

Types of Type Casting

1. **Widening Casting <u>(Implicit)</u>**

    - Converts a smaller data type to a larger data type (e.g., int to double).
    - Done automatically by the compiler.
    - Safe—no data loss.
    - Example:

    ```
    int num = 10;
    double d = num; // No explicit cast needed


    Hierarchy: byte → short → char → int → long → float → double.
    ```

2. **Narrowing Casting <u>(Explicit)</u>**

    - Converts a larger data type to a smaller one (e.g., double to int).
    - Must be done manually by the programmer.
    - Risk of data loss or loss of precision.
    - Example:

    ```
    double d = 9.78;
    int num = (int) d; // Explicit cast
    ```

    - Hierarchy: double → float → long → int → char → short → byte.

```java
// Simple Java program to demonstrate Type Casting
public class TypeCastingExample {
    public static void main(String[] args) {

        // Implicit Casting (smaller to larger type) - automatically
        int intNum = 100;
        double doubleNum = intNum; // int to double (implicit casting)

        System.out.println("Widening Casting:");
        System.out.println("Integer value: " + intNum);
        System.out.println("Converted to double: " + doubleNum);

        // Explicit Casting (larger to smaller type)
        double d = 9.78;
        int i = (int) d; // double to int (explicit casting)

        System.out.println("\nNarrowing Casting:");
        System.out.println("Double value: " + d);
        System.out.println("Converted to integer: " + i);

        // Example in calculation
        int score = 420, maxScore = 500;
        float percentage = (float) score / maxScore * 100;

        System.out.println("\nExample with calculation:");
        System.out.println("Percentage: " + percentage + "%");
    }
}
```