Java Data Types

Java is a **strongly typed language**, meaning:

- Every variable has a declared type
- Every expression has a type
- All assignments are checked **at compile time** for type compatibility

This design improves **program reliability, safety, and maintainability**.

Is Java a Pure Object-Oriented Language?

No. Java is not 100% pure object-oriented.

Reasons:

- Java uses **primitive data types** (`int`, `char`, `boolean`, etc.) which are **not objects**
- Java does **not support multiple inheritance of classes**
- Java does **not support operator overloading** (except `+` for `String`)

However, Java is **object-oriented in practice**, as most real-world programming is done using objects.

Java Primitive Data Types (Exam-Critical)

Java has **8 primitive data types**, grouped as:

1️⃣ Numeric Types

- Integral
- Floating-point

2️⃣ Non-Numeric Types

- `char`
- `boolean`

Integral Data Types (Whole Numbers)

Type	Size	Range
`byte`	1 byte	−128 to 127
`short`	2 bytes	−32,768 to 32,767
`int`	4 bytes	-2^{31} to $2^{31}-1$
`long`	8 bytes	-2^{63} to $2^{63}-1$

✔ Except `char` and `boolean`, **all numeric types are signed**

`byte` Data Type

- Used for **raw binary data**, streams, files, and network I/O
- Internally uses **2's complement** for negative numbers

```
byte b = 10;        // valid
byte b = 130;       // compile-time error (out of range)
byte b = true;      // invalid
```

`short` Data Type

- Rarely used in modern Java
- Historically relevant for 16-bit systems (now obsolete)

```
short s = 100;      // valid
```

```
short s = 32768;    // compile-time error
```

`int` Data Type

- **Default and most commonly used** integer type
- Used for loop counters, array indexing, calculations

```
int i = 100;        // valid
int i = 10.5;       // compile-time error
```

`long` Data Type

- Used when `int` is insufficient (large values, file sizes, timestamps)
- **Suffix `L` or `l` recommended (`L` preferred)**

```
long l = 10L;       // valid
long l = 10;        // valid
int x = 10L;        // compile-time error
```

Floating-Point Data Types (Real Numbers)

Type	Size	Precision
`float`	4 bytes	~6–7 decimal digits
`double`	8 bytes	~15–16 decimal digits

✔ **Default floating-point literal is `double`**

`float`

```
float f = 10.5f;    // valid
```

```
float f = 10.5;     // compile-time error
```

`double`

```
double d = 10.5;    // valid
double d = 10.5D;   // valid
```

Scientific (Exponential) Notation

```
double d = 10e2;     // 1000.0
float f = 10e2F;     // valid
```

`boolean` Data Type

- Only two valid values: `true` and `false` (lowercase)
- **No numeric conversion allowed** (unlike C/C++)

```
boolean b = true;    // valid
boolean b = 1;       // invalid
boolean b = "true";  // invalid
```

✔ Conditions in `if`, `while`, `for` **must be boolean**

`char` Data Type (Unicode-Based)

- Java uses **Unicode (UTF-16)**
- Size: **2 bytes**
- Range: `0` to `65535`

Valid Representations

1️⃣ **Character literal**

```
char ch = 'a';
```

2️⃣ Unicode value (decimal / hex)

```
char ch = 97;        // 'a'
char ch = 0x0061;    // 'a'
```

3️⃣ Unicode escape sequence

```
char ch = '\u0061';  // 'a'
```

Escape Characters (Exam Favorite)

Escape	Meaning
`\n`	New line
`\t`	Tab
`\r`	Carriage return
`\b`	Backspace
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash

String Literals

- Strings are **objects**, not primitives
- Written using **double quotes**

```
String s = "Java";
```

Numeric Literals (Java 7+ Enhancements)

Binary Literals

```
int x = 0b1010;   // 10
```

Underscore in Numeric Literals

Improves readability only; ignored by compiler.

```
int x = 1_000_000;
double d = 1_23_456.78;
```

❌ Invalid placements:

```
double d = _123.45;
double d = 123_.45;
```

Type Casting (High-Scoring Topic)

Implicit Casting (Widening)

- Done automatically by compiler
- No data loss

```
int x = 'a';      // char → int (97)
double d = 10;    // int → double
```

Order:

```
byte → short → int → long → float → double
```

Explicit Casting (Narrowing)

- Programmer responsibility
- Data loss possible

```
int x = 130;
byte b = (byte)x;   // -126
```

Floating → Integral Casting

- Decimal part is discarded

```
double d = 130.456;
int x = (int)d;     // 130
```

Default Values (Very Important for Exams)

Type	Default
Numeric	`0` / `0.0`
`boolean`	`false`
`char`	`'\u0000'`
Object reference	`null`

> ⚠️ Default values apply **only to instance variables**, not local variables

---

Certification Key Takeaways

- Java is **strongly typed**
- `int` and `double` are default numeric types
- `boolean` is **not numeric**
- `char` is **Unicode, not ASCII**
- Narrowing requires explicit cast
- Literal rules are heavily tested