Java Data Types and Literals Reference

Comprehensive Table of Java Data Types and Literals

Data Type	Size (bits)	Minimum Value	Maximum Value	Default Value	Literal Format	Example Literals
Integral Types						
`byte`	8	-128	127	0	Decimal, Hexadecimal, Binary	`100`, `0x64`, `0b1100100`
`short`	16	-32,768	32,767	0	Decimal, Hexadecimal, Binary	`30000`, `0x7530`, `0b111010110000000`
`int`	32	-2^31	2^31 - 1	0	Decimal, Hexadecimal, Binary	`2147483647`, `0x7FFFFFFF`, `0b01111111111111111111111111111111`
`long`	64	-2^63	2^63 - 1	0L	Decimal, Hexadecimal, Binary	`9223372036854775807L`, `0x7FFFFFFFFFFFFFFFL`, `0b0111111111111111111111111111111111111111111111111111111111111111L`
Floating-Point Types						
`float`	32	-3.4 × 10^38	3.4 × 10^38	0.0f	Decimal, Scientific Notation	`3.14f`, `2.5F`, `6.022e23f`, `-1.23e-5f`
`double`	64	-1.8 × 10^308	1.8 × 10^308	0.0d	Decimal, Scientific Notation	`3.14159`, `2.5D`, `6.022e23`, `-1.23e-5d`
Logical Type						
`boolean`	1	N/A	N/A	`false`	Literal keywords	`true`, `false`
Character Type						
`char`	16	'\u0000' (0)	'\uffff' (65,535)	'\u0000'	Character, Unicode, Escape Sequences	`'A'`, `'\u0041'`, `'\n'`, `'\t'`, `'\''`

Literal Examples and Special Notations

Integer Literal Variations

```
// Decimal (base 10)
int decimal = 42;

// Hexadecimal (base 16)
int hex = 0x2A;  // Equivalent to 42

// Binary (base 2)
int binary = 0b101010;  // Equivalent to 42

// Long literals (with L suffix)
long bigNumber = 9223372036854775807L;
```

Floating-Point Literal Variations

```java
// Standard decimal notation
double d1 = 3.14;

// Scientific notation
double scientific = 6.022e23;

// Float with explicit suffix
float f1 = 3.14f;

// Double with optional suffix
double d2 = 3.14D;
```

Character and String Literal Examples

```java
// Character literals
char letter = 'A';
char unicode = '\u0041';  // Unicode for 'A'
char escape = '\n';  // Newline
char specialChar = '\'';  // Single quote

// String literals
String greeting = "Hello, World!";
String multiline = "This is a\nmultiline string";
```

Literal Conversion and Type Inference

Automatic Type Promotion

```java
// Implicit conversion
int intValue = 100;
long longValue = intValue;  // Automatic widening
double doubleValue = intValue;  // Automatic widening

// Explicit casting required for narrowing
long bigLong = 100L;
int narrowedInt = (int)bigLong;  // Explicit cast needed
```

Best Practices

1. Use the smallest data type that can hold your value
2. Be aware of literal suffixes
3. Use underscores for readability in large numbers (Java 7+)
    - Example: `long million = 1_000_000L;`
4. Be cautious with floating-point precision
5. Use explicit casting when converting between types

Common Pitfalls

- Overflow in integer types
- Precision loss in floating-point conversions
- Unintended type conversion
- Forgetting literal suffixes for long and float types

By understanding these literal formats, you can write more precise and readable Java code.