1Z0-808 Exam Topic Reviewer

TopicId: 1004
Topic: Primitive Data Types and Literals

August 5, 2025

Topic 1004: Primitive Data Types and Literals

Thinking Like the Compiler: The Building Blocks

Welcome back. The exam will test your understanding of the absolute basics with surprising depth. Primitive types are Java's fundamental data building blocks. They are not objects, which means they don't have methods and are stored directly in memory (stack for local variables, heap for instance variables). Understanding their precise rules is non-negotiable.

The Eight Primitives: Your Core Toolkit

You must memorize these eight types, their sizes, and their default values. Default values apply to *instance* and *static* variables only. **Local variables are never given a default value.** This is a critical distinction for the exam.

Type	Size	Def
`boolean`	JVM Specific	`fal`
`char`	16-bit	`'\0`
`byte`	8-bit	`(by`
`short`	16-bit	`(sh`
`int`	32-bit	`0`
`long`	64-bit	`0L`
`float`	32-bit	`0.0`
`double`	64-bit	`0.0`

Literals: How You Write Values

A literal is a fixed value written directly in your code. The exam loves to trick you on the rules for writing them.

Integer Literals

By default, any whole number you type is an `int` literal.

- **Decimal (base 10):** `int x = 100;`
- **Octal (base 8):** Must start with a `0`. Valid digits are 0-7.

```
int octal = 017; // This is 15 in decimal
// int invalidOctal = 08; // COMPILE ERROR!
```

- **Hexadecimal (base 16):** Must start with `0x` or `0X`. Digits are 0-9 and A-F.

```
int hex = 0xFF; // This is 255 in decimal
```

- **Binary (base 2):** Must start with `0b` or `0B`. Digits are 0 and 1.

```
int binary = 0b1010; // This is 10 in decimal
```

- **Long Literals:** To specify a `long`, you must add an `L` or `l` suffix. The compiler will complain if a number is outside the `int` range and doesn't have an `L`.

```
long value1 = 2147483647; // Max int, OK
// long value2 = 2147483648; // COMPILE ERROR: Integer too large
long value3 = 2147483648L; // OK, it's a long literal
```

Floating-Point Literals

By default, any number with a decimal point is a `double`.

- **Double:** `double d = 99.5;` or `double d = 99.5d;`
- **Float:** You MUST use an `f` or `F` suffix.

```
// float f = 99.5; // COMPILE ERROR: Lossy conversion from double to float
float f = 99.5f; // OK, it's a float literal
```

- **Scientific Notation:** `double sci = 1.23e4; // 1.23 * 10`^{4}

Character and Boolean Literals

- **Character (`char`):** A single character in single quotes `' '`. A `char` is technically a 16-bit unsigned integer.

```
char letter = 'A';
char unicode = '\u0041'; // 'A' using unicode escape
char number = 65; // 'A' using its ASCII/Unicode value
```

- **Boolean (`boolean`):** Only two possible values: `true` and `false`.

Exam Trap: Underscores in Numeric Literals

Java 7 introduced the ability to use underscores to improve readability. The rules are strict and perfect for exam questions.

```
// VALID uses:
int million = 1_000_000;
long creditCard = 1234_5678_9012_3456L;
double pi = 3.14_159_265;
int binary = 0b1101_0101;

// INVALID uses (WILL CAUSE COMPILE ERRORS):
// float f1 = _100.0f;     // Cannot start with _
// float f2 = 100.0_f;     // Cannot be at the end
// float f3 = 100_.0f;     // Cannot be next to the decimal point
// long l1 = 9876543210_L; // Cannot be just before suffix
// int x = 0_x_123;       // Cannot be next to a radix prefix
```

Key Takeaways for the 1Z0-808 Exam

- **Local vs. Instance Variables:** Local variables have NO default value. Instance/static variables do. A program that tries to use an uninitialized local variable will not compile.
- **Number Systems:** Be comfortable reading octal (`0`), hex (`0x`), and binary (`0b`). Know that `08` is invalid.
- **Literal Suffixes:** An integer literal is an `int` unless it has an `L`. A floating-point literal is a `double` unless it has an `f`.

- **Underscore Rules:** Memorize where underscores CANNOT go: beginning, end, next to decimal point, or next to prefix/suffix.