

## Java Tip 0001 — The Type of Integer Literals

© 2009 Rex Jaeschke. All rights reserved.

**Issue:** I'm coming from a C/C++ environment where the type of an integer literal depends on the platform, with no special handling for the most negative value when two's-complement representations are used. How does this work in Java?

**Response:** Java requires the following sizes for integer type values: `byte` in 8 bits, `short` in 16 bits, `int` in 32 bits, and `long` in 64 bits, with all bits participating in the representation. (Note that JDK1.5 saw the addition of the field `SIZE` to the classes `Byte`, `Short`, `Integer`, and `Long`. This `int` field indicates the number of bits used to represent a value of the corresponding type in two's complement binary form.)

The general rule is that an integer literal not having an 'l' (ell) or 'L' suffix has type `int`, while one having such a suffix has type `long`, provided, of course, the value can actually be represented in that type. For example, `1234` has type `int` while `1234L` has type `long`. `4000000000` is invalid, because the largest `int` value is `2147483647` ( $2^{31}$ ), whereas `4000000000L` has type `long`. And `9223372036854775809L` is invalid as its value exceeds `Long.MAX_VALUE` ( $2^{63}$ ). There is no such thing as an integer literal of type `byte` or `short`. (That said, `Byte.MAX_VALUE` and `Short.MAX_VALUE`, and their minimum counterparts do have values of type `byte` and `short`, respectively.) An integer constant expression can involve a cast, so `(byte)3` is such an expression that has type `byte`.

In Java, all integer values are signed, and they have a two's-complement representation. This means that each integer type value range has one more negative value than it does positive, as follows: `byte` (-128 to 127), `short` (-32768 to 32767), `int` (-2147483648 to 2147483647), and `long` (-9223372036854775808L to 9223372036854775807L).

The expression `-2147483648` consists of two tokens: the unary minus operator and the integer literal `2147483648`. And even though the value of that literal *cannot* be represented in type `int` (the default type in the absence of a suffix), Java requires that this be accepted *provided* it is preceded by a unary minus. (A similar situation exists for the smallest negative value of type `long`.)

Consider the expression `-(2147483648)`. The JLS, §3.10.1, "Integer Literals", p. 23, states "... the literal `2147483648` may appear only as the operand of the unary negation operator `-`." Although one can argue that in this example the literal is indeed the operand of the unary minus operator, the presence of the redundant grouping parentheses causes the JDK1.6 compiler to diagnose this with the message, "integer number too large: 2147483648". The JLS, 3e, does not appear to address this situation.

Note, in C/C++, when targeting a two's-complement platform, the two expressions `-2147483648` and `-2147483647-1` can have different types.