61

Java supports eight basic data types known as *primitive types* as described in Table 2-1. The primitive types include a Boolean type, a character type, four integer types, and two floating-point types. The four integer types and the two floating-point types differ in the number of bits that represent them and therefore in the range of numbers they can represent.

Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	`\u0000`	16 bits	`\u0000` to `\uFFFF`
`byte`	Signed integer	0	8 bits	–128 to 127
`short`	Signed integer	0	16 bits	–32768 to 32767
`int`	Signed integer	0	32 bits	–2147483648 to 2147483647
`long`	Signed integer	0	64 bits	–9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	1.4E–45 to 3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	4.9E–324 to 1.7976931348623157E+308

The `boolean` type represents truth values. This type has only two possible values, representing the two Boolean states: on or off, yes or no, true or false. Java reserves the words `true` and `false` to represent these two Boolean values.

Excerpts from Flanagan, D. (2019). *Java in a Nutshell*. Sebastopol, CA. O'Reilly.

62

The `char` type represents Unicode characters. Java has a slightly unique approach to representing characters—`javac` accepts identifiers and literals as UTF-8 (a variable-width encoding) in input. However, internally, Java represents chars in a fixed-width encoding—either a 16-bit encoding (before Java 9) or as ISO-8859-1 (an 8-bit encoding, used for Western European languages, also called Latin-1) if possible (Java 9 and later).

This distinction between external and internal representation does not normally need to concern the developer. In most cases, all that is required is to remember the rule that to include a character literal in a Java program, simply place it between single quotes (apostrophes):

```
char c = 'A';
```

You can, of course, use any Unicode character as a character literal, and you can use the `\u` Unicode escape sequence. In addition, Java supports a number of other escape sequences that make it easy both to represent commonly used nonprinting ASCII characters, such as `newline`, and to escape certain punctuation characters that have special meaning in Java. For example:

```
char tab = '\t', nul = '\000', aleph = '\u05D0', slash = '\\';
```

`char` values can be converted to and from the various integral types, and the `char` data type is a 16-bit integral type. Unlike `byte`, `short`, `int`, and `long`, however, `char` is an unsigned type. The `Character` class defines a number of useful `static` methods for working with characters, including `isDigit()`, `isJavaLetter()`, `isLowerCase()`, and `toUpperCase()`.

In addition to the `char` type, Java also has a data type for working with strings of text (usually simply called *strings*). The `String` type is a class, however, and is not one of the primitive types of the language. Because strings are so commonly used, though, Java does have a syntax for including string values literally in a program. A `String` literal consists of arbitrary text within double quotes (as opposed to the single quotes for `char` literals). For example:

```
"Hello World"
"'This' is a string!"
```

63

The integer types in Java are `byte`, `short`, `int`, and `long`. As shown in Table 2-1, these four types differ only in the number of bits and, therefore, in the range of numbers each type can represent. All integral types represent signed numbers; there is no `unsigned` keyword as there is in C and C++.

Literals for each of these types are written exactly as you would expect: as a sequence of decimal digits, optionally preceded by a minus sign.[1] Here are some legal integer literals:

```
0
1
123
-42000
```

Integer literals are 32-bit values (and so are taken to be the Java type `int`) unless they end with the character `L` or `l`, in which case they are 64-bit values (and are understood to be the Java type `long`):

```
1234        // An int value
1234L       // A long value
0xffL       // Another long value
```

Integer literals can also be expressed in hexadecimal, binary, or octal notation. A literal that begins with `0x` or `0X` is taken as a hexadecimal number, using the letters `A` to `F` (or `a` to `f`) as the additional digits required for base-16 numbers.

Each integer type has a corresponding wrapper class: `Byte`, `Short`, `Integer`, and `Long`. Each of these classes defines `MIN_VALUE` and `MAX_VALUE` constants that describe the range of the type. The classes also define useful static methods, such as `Byte.parseByte()` and `Integer.parseInt()`, for converting strings to integer values.

Excerpts from Flanagan, D. (2019). *Java in a Nutshell*. Sebastopol, CA. O'Reilly.

64

Real numbers in Java are represented by the `float` and `double` data types. As shown in Table 2-1, `float` is a 32-bit, single-precision floating-point value, and `double` is a 64-bit, double-precision floating-point value. Both types adhere to the IEEE 754-1985 standard, which specifies both the format of the numbers and the behavior of arithmetic for the numbers.

Floating-point values can be included literally in a Java program as an optional string of digits, followed by a decimal point and another string of digits. Here are some examples:

```
123.45
0.0
.01
```

Floating-point literals can also use exponential, or scientific, notation, in which a number is followed by the letter `e` or `E` (for exponent) and another number. This second number represents the power of 10 by which the first number is multiplied. For example:

```
1.2345E02     // 1.2345 * 10^2 or 123.45
1e-6          // 1 * 10^-6 or 0.000001
6.02e23       // Avogadro's Number: 6.02 * 10^23
```

Floating-point literals are `double` values by default. To include a `float` value literally in a program, follow the number with `f` or `F`:

```
double d = 6.02E23;
float f = 6.02e23f;
```

Most real numbers, by their very nature, cannot be represented exactly in any finite number of bits. Thus, it is important to remember that `float` and `double` values are only approximations of the numbers they are meant to represent. A `float` is a 32-bit approximation, which results in at least six significant decimal digits, and a `double` is a 64-bit approximation, which results in at least 15 significant digits. In Chapter 9, we will cover floating-point representations in more detail.

In addition to representing ordinary numbers, the `float` and `double` types can also represent four special values: positive and negative infinity, zero, and NaN. The infinity values result when a floating-point computation produces a value that overflows the representable range of a `float` or `double`.

When a floating-point computation underflows the representable range of a `float` or a `double`, a zero value results.

The `float` and `double` primitive types have corresponding classes, named `Float` and `Double`. Each of these classes defines the following useful constants: `MIN_VALUE`, `MAX_VALUE`, `NEGATIVE_INFINITY`, `POSITIVE_INFINITY`, and `NaN`.