Primitive Data Types

| byte | short | int | long | float | double | char | boolean |

It would be awkward if every time you used data you had to invent your own scheme to represent it with bits. There are types of data that are so fundamental that ways to represent them are built into Java. These are the **primitive data types**. The eight primitive data types are: byte, short, int, long, float, double, char, and boolean.

Upper and lower case characters are important in these names. So "byte" is the name of a primitive data type, but "BYTE" is not. Computer languages where case is important are called **case sensitive**. Some languages are not case sensitive, especially old languages that were designed when data entry equipment did not have lower case characters.

In the phrase *primitive data type* the word **primitive** means "a fundamental component that is used to create other, larger parts." This word is used frequently in computer science. To solve a large problem, you look for the primitive operations that are needed, then use them to build the solution.

Objects

All data in Java falls into one of two categories: **primitive data** and **objects**. There are only eight primitive data types. However, Java has **many types of objects**, and you can invent as many others as you need. Any data type you invent will be a type of object.

Much more will be said about objects in future chapters (since Java is a **object oriented** programming language). The following is all you need to know, for now:

- A primitive data value uses a small, fixed number of bytes.
- There are only eight primitive data types.
- A programmer **cannot create new primitive data types**.
- An object is a big block of data. An object may use many bytes of memory.
- An object usually consists of many internal pieces.
- The data type of an object is called its **class**.
- Many classes are already defined in Java.
- A programmer **can invent new classes** to meet the particular needs of a program.

A (crude) analogy is that a primitive data value is like a nut or a bolt, but an object is like a whole machine.
**Numeric Primitive Data Types**

Numbers are so important in Java that six of the eight primitive data types are numeric types.

There are both integer and floating point primitive types. Integer types have no fractional part; floating point types have a fractional part. On paper, integers have no decimal point, and floating point types do. But in main memory, there are no decimal points. Even floating point values are represented with bit patterns. There is a fundamental difference between the method used to represent integers and the method used to represent floating point numbers.

Each primitive type uses a fixed number of bits. This means that if you are using a particular data type then the same number of bits will be used no matter what value is represented.

For example, all values represented using the short data type use 16 bits. The value zero (as a short) uses 16 bits and the value thirty thousand uses 16 bits.

All values represented using the long data type use 64 bits. The value zero (as a long) uses 64 bits, the value thirty thousand uses 64 bits, and the value eight trillion uses 64 bits.

Values that are large in magnitude (negative or positive) need more bits to be represented. This is similar to writing out numbers on paper: large numbers need more digits. If a value needs more bits than a particular data type uses, then it cannot be represented using that data type.

In the tables, E means "ten to the power of". So \(3.5E38\) means \(3.5 \times 10^{38}\)

### Integer Primitive Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128 to +127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2 billion to +2 billion (approximately)</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9E18 to +9E18 (approximately)</td>
</tr>
</tbody>
</table>
More Bits for More Range

Larger ranges of numeric values require more bits. The different sizes for integer data enable you to pick an appropriate size for the data you are working with. Usually you should pick a data type that has a range much greater than the range of numbers you expect to deal with. If a program uses only a few dozen variables it will run just as fast and take up about as much main memory no matter what size is used for its variables.

Why do the small sized data types exist, then? Well, many real-world programs deal with massive amounts of data (billions of data items) and then using the smaller sizes may save significant amounts of space and time. But we will not use that much data in these notes. Usually you should use int or double for your numeric data.

When you write a program you do not have to know how to represent a number in bits. You can type the number just as you would on a typewriter. This is called a literal. The word "literal" means that a value is explicitly shown in the program.

For example, 125 literally represents the value one hundred twenty five. Integer literals in a program are written as in a book, except there are no commas:

```
125
-32
16
0
-123987
```

All of the above examples are 32 bit int literals.
A 64 bit `long` literal has an upper case 'L' or lower case 'l' at the end. However, **NEVER** use the lower case 'l' because it is easily confused with a digit ‘1’.

```
125L
-32L
16L
0L
-123987L
```

The last two examples use lower case 'l' and are very confusing.

(The author once spent several hours debugging a program where this was the problem.)

### The `char` Primitive Data Type

Computer programs frequently work with character data. The **primitive data type for characters in Java is named `char`**. The `char` type represents a character using 16 bits. In many programming languages, only 8 bits are used for this purpose. Java uses 16 bits so that a very large number of characters can be represented, nearly all of the characters in all of the World’s languages. The method used is called **Unicode**.

For example, here is a 16 bit pattern:

```
0000000001100111
```

If you know that these 16 bits are of data type `char`, then you could look in a table and discover that they represent the character 'g'. If you have a really good memory, you might recall that the same 16 bits represent the integer 103 if they are regarded as data type `short`. **Knowing the data type of a pattern is necessary to make sense of it.**

**Upper and lower case** characters are represented by different patterns. Punctuation and special characters are also `char` data. There are also special characters, like the space character that separates words.

**Control characters** are bit patterns that show the end of a line or where to start pages. Other control characters represent the mechanical activities of old communications equipment (such as Teletypes) that are rarely used these days. Many of these control characters are no longer used for their original purpose.

Primitive type `char` represents a **SINGLE** character. It does not include any font information. When you want to deal with more than one character at a time (almost always), you need to use objects that have been built out of `char` data.
Character Literals

A character literal is a single character with an apostrophe on each side:

'm'   'y'   'A'

A control character is represented with a special sequence of characters:

'\n'   '\t'

Each of these represents a single control character. The first one represents the newline character and the second one represents the tabulation character. You will rarely use any control characters other than these two. Several others are listed in the Java documentation.

Warning: The following is not a character literal:

"Hello"

This is a String, which is not primitive data. It is, in fact, an object. Strings are surrounded by double quote marks "", not by apostrophes.

Primitive Data Type boolean

Another of the primitive data types is the type boolean. It is used to represent a single true/false value. A boolean value can have only one of two values:

true    false

In a Java program, the words true and false always mean these boolean values. The data type boolean is named after George Boole, a nineteenth century mathematician, who discovered that a great many things can be done with true/false values. Although the amount of information in a boolean primitive is logically one bit, for convenience Java uses more than that to represent it.