Special Topic 4.6

Formatting Numbers

The default format for printing numbers is not always what you would like. For example, consider the following code segment:

```
double total = 3.50;
final double TAX_RATE = 8.5; // Tax rate in percent
double tax = total * TAX_RATE / 100; // tax is 0.2975
System.out.println("Total: " + total);
System.out.println("Tax:   " + tax);
```

The output is

```
Total: 3.5
Tax:   0.2975
```

You may prefer the numbers to be printed with two digits after the decimal point, like this:

```
Total: 3.50
Tax:   0.30
```

You can achieve this with the `printf` method of the `PrintStream` class. (Recall that `System.out` is an instance of `PrintStream`.) The first parameter of the `printf` method is a *format string* that shows how the output should be formatted. The format string contains characters that are simply printed, and *format specifiers:* codes that start with a `%` character and end with a letter that indicates the format type. There are quite a few formats—Table 4 shows the most important ones. The remaining parameters of `printf` are the values to be formatted. For example,

```
System.out.printf("Total:%5.2f", total);
```

prints the string `Total:`, followed by a floating-point number with a *width* of 5 and a *precision* of 2. The width is the total number of characters to be printed: in our case, a space, the digit 3, a period, and two digits. If you increase the width, more spaces are added. The precision is the number of digits after the decimal point.

Table 4 Format Types

Code	Type	Example
`d`	Decimal integer	`123`
`x`	Hexadecimal integer	`7B`
`o`	Octal integer	`173`
`f`	Fixed floating-point	`12.30`
`e`	Exponential floating-point	`1.23e+1`
`g`	General floating-point (exponential notation is used for very large or very small values)	`12.3`
`s`	String	`Tax:`
`n`	Platform-independent line end	

Table 5 Format Flags

Flag	Meaning	Example
`-`	Left alignment	`1.23` followed by spaces
`0`	Show leading zeroes	`001.23`
`+`	Show a plus sign for positive numbers	`+1.23`
`(`	Enclose negative numbers in parentheses	`(1.23)`
`,`	Show decimal separators	`12,300`
`^`	Convert letters to uppercase	`1.23E+1`

This simple use of `printf` is sufficient for most formatting needs. Once in a while, you may see a more complex example, such as this one:

```
System.out.printf("%-6s%5.2f%n", "Tax:", total);
```

Here, we have three format specifiers. The first one is `%-6s`. The `s` indicates a string. The hyphen is a *flag*, modifying the format. (See Table 5 for the most common format flags. The flags immediately follow the `%` character.) The hyphen indicates left alignment. If the string to be formatted is shorter than the width, it is placed to the left, and spaces are added to the right. (The default is right alignment, with spaces added to the left.) Thus, `%-6s` denotes a left-aligned string of width 6.

You have already seen `%5.2f`: a floating-point number of width 5 and precision 2. The final specifier is `%n`, indicating a platform-independent line end. In Windows, lines need to be terminated by *two* characters: a carriage return `'\r'` and a newline `'\n'`. In other operating systems, a `'\n'` suffices. The `%n` format emits the appropriate line terminators.

Moreover, this call to `printf` has two parameters. You can supply any number of parameter values to the `printf` method. Of course, they must match the format specifiers in the format string.

The `format` method of the `String` class is similar to the `printf` method. However, it returns a string instead of producing output. For example, the call

```
String message = String.format("Total:%5.2f", total);
```

sets the `message` variable to the string `"Total: 3.50"`.