Chapter 2: Formatted Output

Formatted Output

We're going to look at one of the most often-used techniques for formatting output. It's a method that's been carried over from C and C++ into Java: the printf() method. Another name for it is the *formatted print method*. It's very flexible, and we'll take a look at some of its more commonly used features.

Java's a printf() method gives us more control over output format than the print() or println() methods do. Those simply send a string to the output device, and you have to do any formatting in the string itself. That's difficult to do when you're displaying text or numbers that have varying widths. The printf() method allows us to specify more details about what we want to display. For example, we can set the number of columns a field should take, how many decimal places to use for numbers, and so on.

Here's how it works. While print() and println() accept only one argument, a string, printf() accepts multiple arguments. Its first argument is a *format string* describing the output format. The format string contains *format specifiers* that are placeholders describing the output format for data values.

One or more data values always follow the format string. The printf() method formats the data values and puts them into the output according to the format specifiers in the format string. Let me give you an example, and then I'll explain how it works and go over some of the options we can use in the format specifiers.

Here's a short program that displays some average hourly wages in three columns for different people, some imaginary and some real:

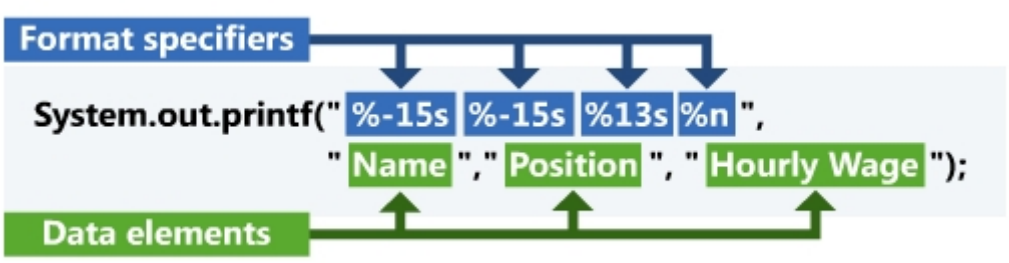
```
/**
 * FirstFormatExample provides a quick example of printf() formatting.
 *
 * @author Merrill Hall
 * @version 1.0
 */
public class FirstFormatExample {
    /**
     * The main() method displays several strings and numbers in columns
     */
    public static void main(String[] args) {
        System.out.printf("%-15s  %-15s  %13s%n","Name","Position",
                          "Hourly Wage");
        System.out.printf("%-15s  %-15s  %13s%n","---------------",
                          "---------------","-------------");
        System.out.printf("%-15s  %-15s  $%,12.2f%n","John Doe",
                          "Busboy",9.1);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Jane Doe",
                          "Cook",10.7);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Jack Doe",
                          "Architect",37.83);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Joan Doe",
                          "Electronic Engr",45.79);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Barack Obama",
                          "US President",192.31);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Tim Tebow",
                          "Quarterback",302.88);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","LeBron James",
                          "B-Ball Player",9168.27);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","David Beckham",
                          "Soccer Player",22115.38);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Warren Buffett",
                          "Investor",30218.76826923076923);
        System.out.printf("%-15s  %-15s  $%,12.2f%n","Bill Gates",
                          "Software Guru",1682692.30769230769231);
    }
}
```

This program produces the following output:

```
Name             Position           Hourly Wage
---------------  ---------------  -------------
John Doe         Busboy           $        9.10
Jane Doe         Cook             $       10.70
Jack Doe         Architect        $       37.83
Joan Doe         Electronic Engr  $       45.79
Barack Obama     US President     $      192.31
Tim Tebow        Quarterback      $      302.88
LeBron James     B-Ball Player    $    9,168.27
David Beckham    Soccer Player    $   22,115.38
Warren Buffett   Investor         $   30,218.77
Bill Gates       Software Guru    $1,682,692.31
```

The printf() method uses a *format string* with format specifiers (the characters following the percent signs) to describe the desired output. Each call to printf() above uses four format specifiers to lay out how to display the data. Following the string with the specifiers are three data elements that the specifiers format for display.

```
System.out.printf("%-15s %-15s %13s %n",
                                "Name","Position", "Hourly Wage");
```



The first two format specifiers in each string are "%-15s", which tell Java that the first two data elements should align with the left-hand margin (that's the "-"), will take up 15 output columns (that's the "15"), and will be string data (that's the "s"). When I say a data element will take up 15 output columns, I mean it'll be at least 15 spaces wide (and longer if it needs to be).

The third specifier in the first two calls is "%13s". It tells Java to expect a string for the third argument and to reserve 13 columns for it. These specifiers format the column headings.

The third specifier in all the remaining calls is "%,12.2f", which tells Java that the third data element is a floating-point number (that's the "f"), should display two decimal places (that's the ".2"), should use 12 output columns (that's the "12"), and should have comma separators between hundreds and thousands and millions and so on (that's the ",").

The fourth specifier in each call is "%n", which inserts a newline character into the output. That means the output will show each call on its own line.

Any text in the first string besides the specifiers displays as given. In this case, that includes spaces between columns and the dollar sign in front of the numbers.

Now You Try

Try running the program yourself, and look at the columns and the modifiers. Let me know if you have any questions.

How Format Specifiers Work

Here's the general format of the format specifiers, as described in the Formatter class documentation, with square brackets around optional parts:

%[argument_index$][flags][width][.precision]conversion

And here are the rules and meanings of the parts of the specifier format:

- Every specifier begins with a percent sign (%).
- Next an optional *argument index* tells Java which data element to get its data from, starting with 1$ for the first data element, 2$ for the second, and so on. This allows you to use data elements out of sequence or more than once. Without an index, printf() uses the data values in sequence—the first value with the first specifier, the second value with the second specifier, and so on.
- Then optional *flags* tell Java how to modify the output. Common flags are:
  - "," to insert separators in numbers
  - "−" to left-justify the data
  - "0" to pad a number to the left with leading zeros
  - "(" to enclose negative numbers in parentheses
- An optional *width* tells Java how many output columns to use for the data element as a minimum. If the data is wider, it'll use more columns.
- The optional *precision* tells Java how many decimal places numeric floating-point output should have. You'd use this to make your data line up properly even if you're working with widely varying numbers.
- The only required part of the specifier is the *conversion*, which tells Java what type of data element to convert to a string for output. An uppercase conversion code, where allowed, will convert output to all uppercase. The most common conversions are:
  - "s" or "S" for strings
  - "c" or "C" for Unicode characters
  - "d" for decimal integers

- "f" for floating-point numbers
- "t" or "T" for date and time conversions
- "n" to insert a newline character into the output
- "%" to put a percent sign in the output

These are just a few of the options available in format specifiers. You can look at more by checking out the Supplementary Material.

Let's look at a few more examples of format specifiers and their output. This example shows several ways to format one number:

```
/**
 * SecondFormatExample provides more examples of printf() formatting.
 *
 * @author Merrill Hall
 * @version 1.0
 */
public class SecondFormatExample {
    /**
     * The main() method displays several format options for numbers.
     */
    public static void main(String[] args) {
        System.out.printf("Format is %%10f:    %10f%n",123.45);
        System.out.printf("Format is %%10.0f:  %10.0f%n",123.45);
        System.out.printf("Format is %%10.1f:  %10.1f%n",123.45);
        System.out.printf("Format is %%10.2f:  %10.2f%n",123.45);
        System.out.printf("Format is %%10.3f:  %10.3f%n",123.45);
        System.out.printf("Format is %%010.3f: %010.3f%n",123.45);
    }
}
```

Its output is this:

```
Format is %10f:    123.450000
Format is %10.0f:         123
Format is %10.1f:       123.5
Format is %10.2f:      123.45
Format is %10.3f:     123.450
Format is %010.3f: 000123.450
```

That example takes one number and formats it six ways. Note that in the format string, a double percent sign shows up in the output as a single percent sign.

The first format, %10f, tells Java to use 10 positions to display the output. Java pads the number on the right with zeros to fill the 10 spaces.

The next four lines show from zero to three decimal places for the same number. Note, too, that Java rounds up when a digit of 5 or greater gets truncated.

The last line shows the use of a "0" in the format specifier to pad the number on the left with zeros.

Let me know if you have any questions about running or modifying these examples.

Date and Time Formatting With printf()

This example uses two classes we haven't discussed, Calendar and GregorianCalendar, to build an internal date/time object. I'm not going to go into the details of those classes here—you can read about them in the API if you're interested. But here's code that displays the day of the week, date, and time of day to the millisecond. It uses printf() to format the output nicely.

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import static java.util.Calendar.*;
/**
 * ThirdFormatExample provides more examples of printf() formatting.
 *
 * @author Merrill Hall
 * @version 1.0
 */
public class ThirdFormatExample {
    /**
     * The main() method displays several Date and time options
     */
    public static void main(String[] args) {
        Calendar c = new GregorianCalendar();
        System.out.printf("Today is %tA, %1$tB %1$te, %1$tY.%n", c);
        System.out.printf("The time is %tl:%1$tM:%1$tS.%1$tL %1$Tp.%n", c);
    }
}
```

And here's the output:

```
Today is Friday, July 19, 2013.
The time is 8:00:55.052 PM.
```

In each statement, each specifier after the first one uses "1$" to reuse the first data argument. The conversions begin with "t" or "T" to indicate a time/date conversion. The next character tells Java which format we want. In the first output statement, "A" is the conversion for fully spelled-out day of the week. "B" gives us the fully spelled-out month. "e" gives us a 1- or 2-position day of the month. "Y" displays a four-digit year.

In the next line, "l" (that's a lowercase "L") displays a 1- or 2-digit hour from 1 to 12. "M" gives us a 2-digit minute, "S" gets a 2-digit second, and "L" provides 3-digit milliseconds. Last, "p" gives us an a.m./p.m. indicator.

I hope these examples give you some idea of the formatting possibilities in printf(). You can find descriptions of many more by visiting the Supplementary Material. Play around with printf() a bit.

I've mentioned the API several times. In the next chapters, we'll look at the information we can get out of it.