**Formatting Print Output**

Earlier you saw the use of the print and println methods for printing strings to standard output (System.out). Since all numbers can be converted to strings (as you will see later in this lesson), you can use these methods to print out an arbitrary mixture of strings and numbers. The Java programming language has other methods, however, that allow you to exercise much more control over your print output when numbers are included.

**The printf and format Methods**

The java.io package includes a PrintStream class that has two formatting methods that you can use to replace print and println. These methods, format and printf, are equivalent to one another. The familiar System.out that you have been using happens to be a PrintStream object, so you can invoke PrintStream methods on System.out. Thus, you can use format or printf anywhere in your code where you have previously been using print or println. For example,

System.out.format(.....);

The syntax for these two [java.io.PrintStream](https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html" \t "_blank)methods is the same:

public PrintStream format(String format, Object... args)

where format is a string that specifies the formatting to be used and args is a list of the variables to be printed using that formatting. A simple example would be

System.out.format("The value of " + "the float variable is " +

"%f, while the value of the " + "integer variable is %d, " +

"and the string is %s", floatVar, intVar, stringVar);

The first parameter, format, is a format string specifying how the objects in the second parameter, args, are to be formatted. The format string contains plain text as well as *format specifiers*, which are special characters that format the arguments of Object... args. (The notation Object... args is called *varargs*, which means that the number of arguments may vary.)

Format specifiers begin with a percent sign (%) and end with a *converter*. The converter is a character indicating the type of argument to be formatted. In between the percent sign (%) and the converter you can have optional flags and specifiers. There are many converters, flags, and specifiers, which are documented in [java.util.Formatter](https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html" \t "_blank)

Here is a basic example:

int i = 461012;  
System.out.format("The value of i is: %d%n", i);

The %d specifies that the single variable is a decimal integer. The %n is a platform-independent newline character. The output is:

The value of i is: 461012

The printf and format methods are overloaded. Each has a version with the following syntax:

public PrintStream format(Locale l, String format, Object... args)

To print numbers in the French system (where a comma is used in place of the decimal place in the English representation of floating point numbers), for example, you would use:

System.out.format(Locale.FRANCE,

"The value of the float " + "variable is %f, while the " +

"value of the integer variable " + "is %d, and the string is %s%n", floatVar, intVar, stringVar);

**An Example**

The following table lists some of the converters and flags that are used in the sample program, TestFormat.java, that follows the table.

|  |  |  |
| --- | --- | --- |
| **Converters and Flags Used in TestFormat.java** | | |
| **Converter** | **Flag** | **Explanation** |
| d |  | A decimal integer. |
| f |  | A float. |
| n |  | A new line character appropriate to the platform running the application. You should always use %n, rather than \n. |
| tB |  | A date & time conversion—locale-specific full name of month. |
| td, te |  | A date & time conversion—2-digit day of month. td has leading zeroes as needed, te does not. |
| ty, tY |  | A date & time conversion—ty = 2-digit year, tY = 4-digit year. |
| tl |  | A date & time conversion—hour in 12-hour clock. |
| tM |  | A date & time conversion—minutes in 2 digits, with leading zeroes as necessary. |
| tp |  | A date & time conversion—locale-specific am/pm (lower case). |
| tm |  | A date & time conversion—months in 2 digits, with leading zeroes as necessary. |
| tD |  | A date & time conversion—date as %tm%td%ty |
|  | 08 | Eight characters in width, with leading zeroes as necessary. |
|  | + | Includes sign, whether positive or negative. |
|  | , | Includes locale-specific grouping characters. |
|  | - | Left-justified.. |
|  | .3 | Three places after decimal point. |
|  | 10.3 | Ten characters in width, right justified, with three places after decimal point. |

The following program shows some of the formatting that you can do with format. The output is shown within double quotes in the embedded comment:

import java.util.Calendar;  
import java.util.Locale;

public class TestFormat {

public static void main(String[] args) {  
 long n = 461012;  
 System.out.format("%d%n", n); // --> "461012"  
 System.out.format("%08d%n", n); // --> "00461012"  
 System.out.format("%+8d%n", n); // --> " +461012"  
 System.out.format("%,8d%n", n); // --> " 461,012"  
 System.out.format("%+,8d%n%n", n); // --> "+461,012"   
 double pi = Math.PI;  
 System.out.format("%f%n", pi); // --> "3.141593"  
 System.out.format("%.3f%n", pi); // --> "3.142"  
 System.out.format("%10.3f%n", pi); // --> " 3.142"  
 System.out.format("%-10.3f%n", pi); // --> "3.142"  
 System.out.format(Locale.FRANCE, "%-10.4f%n%n", pi); // --> "3,1416"  
 Calendar c = Calendar.getInstance();  
 System.out.format("%tB %te, %tY%n", c, c, c); // --> "May 29, 2006"  
 System.out.format("%tl:%tM %tp%n", c, c, c); // --> "2:34 am"  
 System.out.format("%tD%n", c); // --> "05/29/06"  
 }  
}

**Note:**  The discussion in this section covers just the basics of the format and printf methods. Further detail can be found in the [Basic I/O](https://docs.oracle.com/javase/tutorial/essential/io/formatting.html)section of the Essential trail, in the "Formatting" page.  
Using String.format to create strings is covered in [Strings](https://docs.oracle.com/javase/tutorial/java/data/strings.html).

**The DecimalFormat Class**

You can use the [java.text.DecimalFormat](https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html" \t "_blank)class to control the display of leading and trailing zeros, prefixes and suffixes, grouping (thousands) separators, and the decimal separator. DecimalFormat offers a great deal of flexibility in the formatting of numbers, but it can make your code more complex.

The example that follows creates a DecimalFormat object, myFormatter, by passing a pattern string to the DecimalFormat constructor. The format() method, which DecimalFormat inherits from NumberFormat, is then invoked by myFormatter—it accepts a double value as an argument and returns the formatted number in a string:

Here is a sample program that illustrates the use of DecimalFormat:

import java.text.\*;

public class DecimalFormatDemo {  
 static public void customFormat(String pattern, double value ) {  
 DecimalFormat myFormatter = new DecimalFormat(pattern);  
 String output = myFormatter.format(value);  
 System.out.println(value + " " + pattern + " " + output);  
 }  
 static public void main(String[] args) {

customFormat("###,###.###", 123456.789);  
 customFormat("###.##", 123456.789);  
 customFormat("000000.000", 123.78);  
 customFormat("$###,###.###", 12345.67);   
 }  
}

**The output is:**

123456.789 ###,###.### 123,456.789  
123456.789 ###.## 123456.79  
123.78 000000.000 000123.780  
12345.67 $###,###.### $12,345.67

The following table explains each line of output.

|  |  |  |  |
| --- | --- | --- | --- |
| **DecimalFormat.java Output** | | | |
| **Value** | **Pattern** | **Output** | **Explanation** |
| 123456.789 | ###,###.### | 123,456.789 | The pound sign (#) denotes a digit, the comma is a placeholder for the grouping separator, and the period is a placeholder for the decimal separator. |
| 123456.789 | ###.## | 123456.79 | The value has three digits to the right of the decimal point, but the pattern has only two. The format method handles this by rounding up. |
| 123.78 | 000000.000 | 000123.780 | The pattern specifies leading and trailing zeros, because the 0 character is used instead of the pound sign (#). |
| 12345.67 | $###,###.### | $12,345.67 | The first character in the pattern is the dollar sign ($). Note that it immediately precedes the leftmost digit in the formatted output. |