FORMATTING OUTPUT WITH THE NEW FORMATTER

Source http://java.sun.com/developer/JDCTechTips/2004/tt1005.html#1

J2SE 5.0 introduces a new way to format output that is similar to that of the C language's printf. In this approach, each argument to be formatted is described using a string that begins with % and ends with the formatted object's type. This tip introduces you to the new Formatter class and to the syntax of the formatting that you perform with the class.

It's likely that you will most often use the new formatting approach in a call similar to either of the following:

```
System.out.format("Pi is approximately  %f", Math.Pi);
System.out.printf("Pi is approximately  %f", Math.Pi);
```

The printf() and the format() methods perform the same function. System.out is an instance of java.io.PrintStream. PrintStream, java.io.PrintWriter, and java.lang.String each have four new public methods:

```
format( String format, Object... args);
printf( String format, Object... args);
format( Locale locale, String format, Object... args);
printf( Locale locale, String format, Object... args);
```

These correspond to the format() method in the underlying worker class java.util.Formatter class:

```
format(String format, Object... args)
format(Locale l, String format, Object... args)
```

Although you'll likely use these methods from String, PrintStream, and PrintWriter, you'll find the documentation for the various available formatting options in the documentation for the Formatter class.

Let's begin with the following example of the format() method:

```
formatter.format("Pi is approximately %1$f," +"and e is about %2$f", Math.PI,
Math.E);
```

The format() method requires some of the new language features introduced in J2SE 5.0. One is varargs, which simplifies the way that an arbitray number of arguments can be passed to a method. Notice that a variable number of Object instances can be entered for formatting. Also notice that the objects in the example are autoboxed and unboxed. Autoboxing/Unboxing is also a new feature in J2SE 5.0. Autoboxing eliminates the need for manually converting a primitive type (such as int) to a wrapper class (such as Integer),

unboxing automates the reverse process. In the example, Math.PI is a double which is autoboxed to a Double (to be treated as an Object). In addition, in the example the formatted output is written to java.lang.StringBuilder, yet another new feature introduced in J2SE 5.0.

The format itself is a String that includes zero or more formatting elements, each beginning with a %. Each formatting element is applied to one of the Objects passed in. Each formatting element has the general form:

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

The argument index is a positive integer that indicates the position of the argument in the argument list. The numbering begins with 1 for the first position, not with 0. So the first position in the previous code snippet is occupied by Math.PI, and is indicated by using 1$. The second position is occupied by Math.E, and is indicated by using 2$.

The width specifies the minimum number of characters to be written as output.

The precision is used to restrict the number of non-zero characters.

The conversion describes the type of the object being formatted. Much of this should be familiar to C programmers because this is a Java implementation of printf(). Common types include f for float, t for time, d for decimal, o for octal, x for hexadecimal, s for general, and c for a Unicode character. The following sample application, UsingFormatter, allows you to enter different formats from the command line and view the output. Notice that the application instantiates a destination -- in this example, a StringBuilder. It then instantiates a Formatter and associates it with the destination. The Formatter then formats some String and sends the output to the destination. The results of the conversion are then displayed to standard out.

```
package format;

import java.util.Formatter;

public class UsingFormatter {

  public static void main(String[] args) {
    if (args.length != 1) {
      System.err.println("usage: " +
        "java format/UsingFormatter ");
      System.exit(0);
    }
    String format = args[0];

    StringBuilder stringBuilder = new StringBuilder();
    Formatter formatter = new Formatter(stringBuilder);
    formatter.format("Pi is approximately " + format +
```

```
        ", and e is about " + format, Math.PI, Math.E);
      System.out.println(stringBuilder);
    }
  }
```

Compile and run this with the command line argument %f:

```
java format/UsingFormatter %f
```

You should get the result:

```
Pi is approximately 3.141593, and e is about 2.718282
```

Rerun this and set the precision to be two decimal places:

```
java format/UsingFormatter %.2f
```

You should see the following:

```
Pi is approximately 3.14, and e is about 2.72
```

Notice that the numbers are not just truncated. The value of e is rounded off to two decimal places. You can additionally specify the width by supplying the command line argument %6.2f. This time leading spaces are inserted because you specified that the number should use six characters, even though the precision restricts it to using only three characters and a decimal place. If you enter the command:

```
java format/UsingFormatter %6.2f
```

You should see this:

```
Pi is approximately   3.14, and e is about   2.72
```

The position can be used to specify which argument to format. Rerun UsingFormatter with the command line argument %1$.2f. This specifies that you want to use Math.PI twice. You should see the following output:

```
Pi is approximately 3.14, and e is about 3.14
```

You can change the Locale used to format the numbers by adding an import statement and calling a different constructor. Here's an example:

```
package format;

import java.util.Formatter;
```

```
import java.util.Locale;

public class UsingFormatter {

  public static void main(String[] args) {
    if (args.length != 1) {
      System.err.println("usage: " +
        "java format/UsingFormatter <format string>");
      System.exit(0);
    }
    String format = args[0];

    StringBuilder stringBuilder = new StringBuilder();
    Formatter formatter = new Formatter(stringBuilder,
                                Locale.FRANCE);
    formatter.format("Pi is approximately " + format +
      ", and e is about " + format, Math.PI, Math.E);
    System.out.println(stringBuilder);
  }
}
```

Compile and run this with the argument %.2f and you should see the decimal points changed to commas.

```
Pi is approximately 3,14, and e is about 2,72
```

As previously mentioned, you will typically not use the Formatter class explicitly. Instead, for example, you can directly use the printf() and format() methods in the PrintStream class. The following program, UsingSystemOut, is a rewritten version of the UsingFormatter program to use standard out:

```
package format;

public class UsingSystemOut {

  public static void main(String[] args) {
    if (args.length != 1) {
      System.err.println("usage: " +
        "java format/UsingSystemOut <format string>");
      System.exit(0);
    }
    String format = args[0];

    System.out.format("Pi is approximately " + format +
      ", and e is approximately " + format, Math.PI, Math.E);
```

```
  }
}
```

The behavior of UsingSystemOut is slightly different than that of UsingFormatter. The UsingFormatter program uses println(), the UsingSystemOut program does not. Because of that, if you run UsingSystemOut from the command line, you will notice that your next command prompt is on the same line as your output. You need to insert a new line. You can do this using formatted output by adding %n. Run UsingSystemOut with the command line argument %.2f%n:

```
java format/UsingSystemOut %.2f%n
```

You will see the following result:

```
Pi is approximately 3.14 , and e is about 2.72
```

You can replace the last method call with printf() if you prefer. There is no difference between System.out.format() and System.out.printf().

As a final example, let's take a look at how date and time objects can be formatted. These objects have the conversion type t or T. That letter is followed by a second letter that indicates which part of the time should be displayed and how it should be displayed. For example, you can display the hour in a variety of forms using tH, tI, tk, or tl, and the minute within the hour using tM. These can also be combined with tr, which displays tH:tM. Similarly, you can display the day of the week, the name of the month, and so on, using the format conversion keys detailed in the Formatter API.

Here is an example that displays the date by formatting the current time using tr for the hour and minute, tA for the day of the week, tB for the name of the month, te for the number of the day of the month, and tY for the year. These could all be preceded by 1$ to point to the first position. Instead, the %tr points to the first position. The < in the other format strings refers back to the position formatted previously.

```
package format;

import java.util.Calendar;

public class FormattingDates  {

  public static void main(String[] args) {
    System.out.printf("Right now it is %tr on " +
                      "%<tA, %<tB %<te, %<tY.%n",
                      Calendar.getInstance());
  }
}
```

Compile and run the FormattingDates program. You will see output that looks something like this:

```
Right now it is 01:55:19 PM on Wednesday, September 22, 2004.
```

This tip is intended to get you started using the new Formatter facility for formatting output. In some ways, the options should feel familiar to you from the old printf() days. Here as before, there are many possibilities available. You will only learn them by trying out different options and deciding which ones meet your needs.

~~~ End of Article ~~~