CISP21 – Programming in Java

Formatting data using printf and format methods

When you are displaying multiple lines of output, you usually want the output to line up evenly in columns. The **printf** method of the **PrintStream** class can help solve this problem. Remember that **System.out** is an instance of **PrintStream**, so it is available via the **System.out.printf** method. **Printf** is similar to the **print** method except that its argument list needs to be different.

The **printf** method is expecting a format string of how the data should be displayed, followed by the data to be displayed.

The format string can contain characters that are simply printed and can be combined with format specifiers, which are codes that start with a % and end with a letter that indicates the format type.

To help illustrate this method, here is an example.

System.out.printf("%-8s%7.2f%n", "Total:", totalDouble);
System.out.printf("%-8s%7.2f%n", "Tax:", taxDouble);
The output from these statements might be (vertical lines are there for explanation only):

Total:	100.00
Tax:	9.75

Explanation:

%-8s s indicates a String, the hyphen is a flag, modifying the format to left-alignment and the 8 is reserving a minimum of 8 characters for the string. The value used is the first one after the comma

%7.2f f indicates a floating-point number, the 7.2 indicates a total width of 7 with 2 decimal places (the decimal places are included in the count of 7) and any remaining space is filled on the left with spaces (right-aligned). The value used is the second one after the comma.

%n newline character similar to a \n escape sequence

Formatting using Strings within components:

The **format** method of the **String** class is similar to the **printf** method. However, it returns a String instead of directly producing output.

Because the format methods are basing position on font sizes, make sure that you have set a non-proportional font, such as *Courier* to be used within the component.

Reminder: to set a font, first create the font object and then it on the specific component (such as a text area or JOptionPane dialog box), as follows:

```
Font courierFont = new Font("Courier", Font.PLAIN, 12);
outputTextArea.setFont(courierFont) ;
```

This could be used to format a text area with fixed-width columns.

String totalString = String.format("%-8s%7.2f%n", "Total:", totalDouble);
String taxString = String.format("%-8s%7.2f%n", "Tax:", taxDouble);
outputTextArea.append(totalString);
outputTextArea.append(taxString);

See the next table on common format types and flags.

Common Format Types		
Code	Type	Example
d	Decimal integer	123
x	Hexadecimal integer	7B
o	Octal integer	173
f	Fixed floating-point	15.75
e	Exponential floating-point	1.23E+3
g	General floating-point (exponential notation used for very large or very small values)	123.5
s	String	Total:
n	Platform-independent line end (like a \n)	

Common Format Flags		
Flag	Meaning	Example
-	Left alignment (right-alignment is the default)	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	12.34
^	Convert letters to uppercase	1.23E+3

Currency display

If you want dollar signs ($)to display, you will need to first format the number into a formatted string via DecimalFormat or NumberFormat objects. Then add the formatted string object in the **format** method, remembering that a number that has been formatted is now a String object, not a number. An example follows:

DecimalFormat currencyFormat = new DecimalFormat ("$#0.00");
…
String formattedCurrency = currencyFormat.format(totalDouble);
String totalString = String.format("%-8s%7s%n", "Total:", formattedCurrency);