06.02 Virtual Lecture Notes (Part 1)

Formatting String Output

Most of the code in the FormattingStrings.java demo file has been commented out. As you move through the lesson, uncomment the appropriate lines to observe the subtleties of the **`printf( )`** method. The first example illustrates the bare minimum required to format a String with **`printf( )`**.

```
String arg1 = "Tallahassee";
String arg2 = "Florida";

System.out.printf("%s", arg1);
System.out.println();
System.out.printf("%S", arg1);
```

Notice that the format specifiers (%s and %S) are inserted in the **`String`** literal at the exact positions where the value of the variables will be printed. For Strings, the converter is either an uppercase or a lowercase "S." Carefully examine the source code in relation to the output.

- What is the result of using a lowercase "s" for the String converter?
- What is the result of using an uppercase "S" as the String converter?

Every format specifier begins with a percent sign (%) and ends with a type converter represented by a single character.

1. What happens if you delete the percent sign at the front of the format specifier? Try it.
2. What is the effect of changing "s" to "S" and vice versa, or removing it completely? You won't know unless you experiment.
3. Why is there a **`println( )`** statement after each **`printf( )`** statement? Comment them out and observe the results.
4. Could you obtain the same formatted output with either a **`print( )`** or a **`println( )`** statement?

The percent sign and the type converter (e.g., %s) are the bare minimum for a String format specifier; however, the optional components are what make printf( ) so useful.

Let us add two optional pieces to the format string: field width and escape characters. The number 15 in the following example specifies the minimum number of characters that will be written to the output field. In other words, the format string reserves a field 15 spaces wide to print the argument.

```
System.out.printf("%15s%n", arg1);
System.out.printf("%-15s%n", arg1);
```

Uncomment these statements in the demo program and observe the output. This example should raise more questions for you.

- Are you surprised by how the word is justified? Count the number of letters in Tallahassee. The field width was 15, but the word is smaller than the requested field width. When this happens, blank spaces are printed **in front** of the argument to fill up the field. How many blanks did it take to fill up the field in front of Tallahassee? Blank spaces are also characters, so the number of letters and spaces should add up to the field width of 15.
- Where is the blank **`println( )`** statement used in the previous examples? Earlier in the course, you printed escape characters to tab (\t), move to a new line (\n), etc. What purpose does %n serve in the format specifier? Take it out and observe the effect.
- What happens if you try to print something with more characters than the field width can hold? Change the **`arg1`** variable to a sentence with more than 15 characters and analyze the resulting output.

Obviously, there will be times when you need to left-justify a String. The **`printf( )`** method allows a variety of optional flags that give additional control over the output format.

- What is the function of a dash (-) in front of the field width? Delete it and observe the effect.

You now know the basics of **`String`** formatting, but further details about format specifier syntax can be found in the Java API.

Print