System.Out Methods & Concatenation

by Sophia

WHAT'S COVERED

In this lesson, you will learn about how to use a Java `Scanner` to read different types of input in Java. Specifically, this lesson covers:

1. Console Output

Console is the text-based interface that allows users to interact with a program by inputting and receiving text data. In the IDE, you will type directly into the console.

Throughout much of the Java code in previous tutorials, many of the lines of code that were used make use of `System.in` and `System.out`. `System.in` is called the "standard input stream." It is the channel through which console input is read.

`System.out` is the "standard output stream." They are both only to be used in console applications. This is due to the fact that graphical user interface (GUI) applications have their own approaches to input and output. `System.in` and `System.out` are special streams that are automatically opened by the system and are ready for use. They are special in that they should never be closed. In a future tutorial, you will learn about a few differences when using streams to read and write files. They will require action to explicitly open and close them when used for these purposes.

`System.out` includes two library methods that were used in a previous tutorial. Let's now take a closer look at `print()` and `println()`. The key difference between `print()` and `println()` is that `println()` automatically adds a new line character (`\n`) at the end of the output. However, `print()` does not.

Review the output of the following code that includes `println()`:

```
class ConsolePrint {
  public static void main(String[] args) {
```

```
    // println() adds \n at the end of each output
    System.out.println("Output 1");
    System.out.println("Output 2");
    // print() does not add \n
    System.out.print("Output 3");
    System.out.print("Output 4");
  }
}
```

The resulting output should look like this:

```
Output 1
Output 2
Output 3Output 4|
```

Since Output 1 and Output 2 are displayed using `println()`, a new line character (`\n`) is automatically added at the end of the output. This causes each to appear on its own line. Output 3 and Output 4 are sent to the console using plain `print()`, so no new line (or even space) is added at the end. In this case, note how the two `String` values are not only displayed on the same line but are also run together and you can see the cursor still appears directly after Output 4. When displaying text in an application, it is important to be mindful of the spaces needed to keep such hard-to-read chunks of text from occurring. In this case, putting a space after "Output 3" or at the start of the String "Output 4" (resulting in " Output 4") would help.

Note how the cursor right after the end of the program's output, which looks a bit odd.

It is possible to add a `\n` in the text to be printed out. This can be helpful when you want to double space lines of text. However, as a general rule, it is best to use `println()` for output that needs to appear on separate lines. When two or more pieces of text are output that need to be on the same line, it is important to add spaces where they are needed to keep the pieces of text from running together.

When displaying prompts for input, though, the usual practice is to use `System.out.print()` to display the prompt so that the user's input is on the same line as the prompt. Putting the cursor on the next line by using `System.out.println()` can give the appearance that the program is busy or frozen rather than waiting for user input.

---

2. Concatenation

In previous examples, you have seen the use of the plus sign (+) to concatenate pieces of text. When joining together a String value with other types of data (such as `int` or `double` values), these other types of values are usually automatically converted to `String` data.

In addition to the +, which does double duty as the concatenation operator as well as being the arithmetic addition operator, Java `Strings` have a `concat` method that can be used to explicitly join them. Here is an example:

```
class StringConcat {
  public static void main(String[] args) {
    String firstWord = "Hello, ";
    String greeting = firstWord.concat("world!");
    // Note how .concat() can be called on a literal String value
    String question = "How are ".concat("you?");
    System.out.println(greeting);
    System.out.println(question);
  }
}
```

The resulting output screen should look like this:

```
Hello, world!
How are you?
```

The same thing could be accomplished using the concatenation operator (+):

```
class StringConcat {
  public static void main(String[] args) {
    String firstWord = "Hello, ";
    String greeting = firstWord + "world!";
    /* There wouldn't really be a need to use + when
       the String could be written as "How are you?"
       but this is just a demonstration */
    String question = "How are " + "you?";
    System.out.println(greeting);
    System.out.println(question);
  }
}
```

The results in the console should be the same as the previous output.

When concatenating text, the `String` portion doesn't have to be first, but there has to be a `String` (either a literal `String` in double quotes or a `String` variable) in the pieces being joined:

```
class StringConcat {
  public static void main(String[] args) {
    int lineNumber = 1;
```

```
    // Note the period and space at start of appended text
    String lineOne = lineNumber + ". Sample text.";
    // Increase value of lineNumber by 1 using ++
    lineNumber++;
    String lineTwo = lineNumber + ". More sample text.";
    System.out.println(lineOne);
    System.out.println(lineTwo);
  }
}
```

The results should look like this:

```
1. Sample text.
2. More sample text.
```

The concatenation can also be carried out in the parentheses that pass the text to `print()` or `println()`:

```
class StringConcat {
  public static void main(String[] args) {
    int lineNumber = 1;
    // Note the period and space at start of appended text
    String lineOne = lineNumber + ". Sample text.";
    // Increase value of lineNumber by 1 using ++
    lineNumber++;
    String lineTwo = lineNumber + ". More sample text.";
    System.out.println(lineOne);
    System.out.println(lineTwo);
  }
}
```

There is an important rule to keep in mind about concatenation: At least one of the items being joined together must be a String (a variable or literal value). Concatenating a `char` or `int` with another `char` or `int` will have unexpected results, as these examples show.

Using a string when joining items together is demonstrated as follows:

```
class Concatenation {
  public static void main(String[] args) {
    char letter = 'A';
    // This line will not display A5
    System.out.println(letter + 5);
    // This line will not display A+
    System.out.println(letter + '+');
  }
```

```
}
```

One might expect to see the following:

```
A5
A+
```

However, that's not what happens! The resulting output looks like this:

```
70
108
```

Java works with `char` values behind the scenes as unsigned integers, so the + is interpreted as the arithmetic addition operator. If the `char` data can't be changed to a `String` value, adding an empty String ("" with no space between) into the concatenation will resolve the problem:

```
class Concatenation {
  public static void main(String[] args) {
    char letter = 'A';
    // This line will not display A5
    System.out.println("" + letter + 5);
    // This line will not display A+
    System.out.println("" + letter + '+');
  }
}
```

The output screen looks like this:

```
A5
A+
```

REFLECT

As seen above combining strings with other data types can be tricky. Why would Java not be able to determine that we wanted A5 and A+ from the first example? Remember in the end, all data that is processed in a computer is a number, and more specifically, a binary number. Our brains do sort of an analogous thing. Information is encoded in neurons, and we then interpret letters and numbers as separate entities based on context. We have to give Java a bit of context to make it know that we want a string for the A5 and A+ example and this is accomplished with the addition of the empty string.

SUMMARY

In this lesson, you explored the important parts of working with `System.out`'s `print()` and `println()` methods for displaying text in the **console output** window. You learned about prompts for

user input and displaying results from the program. You explored how to use these methods correctly and that they are an important part of Java programming. Finally, you learned that the **concatenation** of text and other values is also important for conveying information clearly to the user.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source **cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf**

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source **py4e.com/html3/**