

Introduction to String & Character Types

by Sophia



WHAT'S COVERED

In this lesson, you will learn about data string and character types that are used in Java. Specifically, this lesson covers:

Table of Contents

- [1. Character Data](#)
- [2. String Data](#)

1. Character Data

The various numeric values discussed previously are very important. However, modern computers do much more than work with numeric values. The concept of a "math machine" sounds interesting, but would probably not be useful.

In this tutorial, you will consider and use the Java data types used to represent text. These include numeric types like int, long, float, and double, representing **primitive data types**. These data types are built into the fabric of the Java language itself.

The character type (char) is another primitive type. It is used to hold a single character. The char type is used for working with values that are just a single character treated as a single character. You will see that it is also possible to have single-character strings.

A literal char value specified in code is enclosed in single quotes:

➞ EXAMPLE

```
char firstLetter = 'A';  
char punctuation = '?';
```

Note that individual digits can also be treated as characters to refer to the symbol on the screen or page (but not the numeric value):

➞ EXAMPLE

```
char luckyDigit = '7';
```

Keep in mind that just because an item on the screen consists of digits, does not mean it is a numeric value that could be used in arithmetic calculations. ZIP codes, phone numbers, and Social Security numbers are good examples of things that are made up of digits. They are even called numbers—but aren't really numbers. They are not necessarily useful in calculations. For this reason, single digits can be char data. And one or more digits can be a Java string.

Behind the scenes, Java treats a char value as a 16-bit int value. Since there are no "negative" characters in the way that there are both positive and negative numbers, the char type doesn't need a sign bit, so all 16 bits are used to represent a positive value. This means that char can hold values in the range from 0 to 65536. These underlying numeric values are used to map to Unicode characters. The Unicode character set is an international standard that allows computer languages to represent the letters, characters, and other signs used to write most of the human languages in use today, including but not limited to English.

Java includes some special characters that are specified using a backslash (\) followed by a letter. These help it to indicate a character that is not itself displayed, but that controls the display of information on the screen. These groups of characters that start with a backslash are known as **escape sequences**.

There are several escape sequences defined in Java. However, the most important escape sequences at this point are:

| Escape Sequence | Value / Use |
|-----------------|---|
| \n | New line (moves the cursor to the start of the following line) |
| \t | Tab (moves the cursor a set amount of space horizontally) |
| \" | Displays a double quote character as a char or part of a string |

➞ **EXAMPLE** Consider the following example that uses the \n escape sequence:

```
public class EscapeSequences {  
    public static void main(String[] args) {  
        char firstLetter = 'A';  
        char lastLetter = 'Z';  
        char newLine = '\n';  
        System.out.println(firstLetter);  
        System.out.print(lastLetter);  
    }  
}
```

The result of this code looks like this:

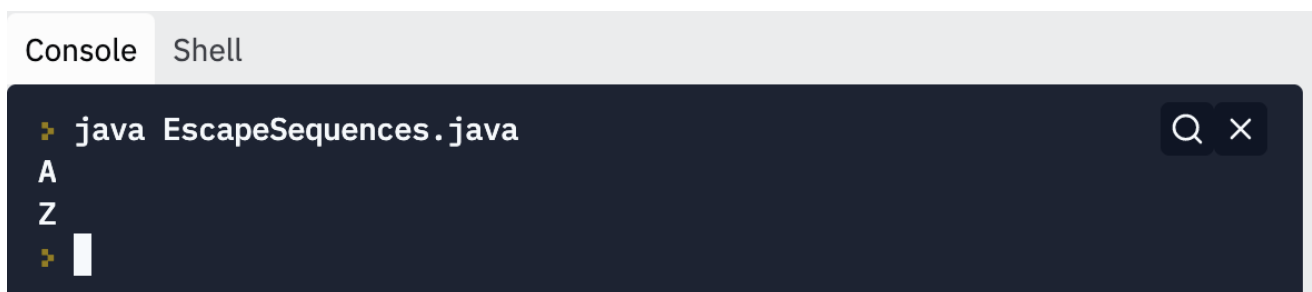
ConsoleShell

```
➤ java EscapeSequences.java  
AZ
```

➞ EXAMPLE Here is a sample of EscapeSequences.java A-Z:

```
public class EscapeSequences {  
    public static void main(String[] args) {  
        char firstLetter = 'A';  
        char lastLetter = 'Z';  
        char newLine = '\n';  
        /* System.out.print() doesn't automatically add  
        a new line character at the end the way that  
        System.out.println() does. */  
        System.out.print(firstLetter);  
        System.out.print(newLine); // Output new line  
        System.out.print(lastLetter);  
        System.out.print(newLine); // Output new line  
    }  
}
```

The result of this code looks like this:



```
Console Shell  
➞ java EscapeSequences.java  
A  
Z  
➞
```

 REFLECT

Note how the printing of the new line (`\n`) character after the A has the same effect as using `System.out.println()` in the previous code. Adding the new line character after the Z tidies the output up a bit by pushing the display of the system prompt down to the next line. The same effect could have been achieved using:

```
System.out.println(lastLetter);
```

➞ EXAMPLE EscapeSequences.java A-Z

```
public class EscapeSequences {  
    public static void main(String[] args) {  
        char firstLetter = 'A';  
        char lastLetter = 'Z';  
        char newLine = '\n';  
        /* System.out.print() doesn't automatically add  
        a new line character at the end the way that  
        System.out.println() does. */  
        System.out.print(firstLetter);  
        System.out.print('\t'); // Output tab
```

```

    System.out.print(lastLetter);
    System.out.print(newLine); // Output new line
}
}

```

The result of this code looks like this:



```

Console Shell
> java EscapeSequences.java
A Z
>

```

➞ EXAMPLE `EscapeSequences.java` Hello!

```

public class EscapeSequences {
    public static void main(String[] args) {
        System.out.println("The computer said, \"Hello!\");
    }
}

```

The result of this code looks like this:



```

Console Shell
> java EscapeSequences.java
The computer said, "Hello!"
>

```

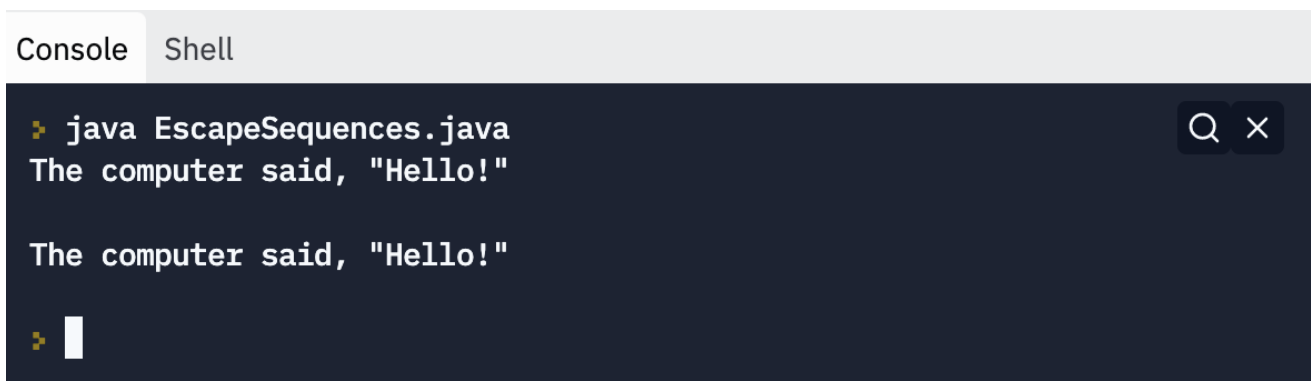
➞ EXAMPLE Replit `EscapeSequences.java` Hello! Hello!

```

public class EscapeSequences {
    public static void main(String[] args) {
        System.out.println("The computer said, \"Hello!\");
        // Added new line characters in output
        System.out.println("\nThe computer said,\"Hello!\");
    }
}

```

The result of this code looks like this:



```

Console Shell
> java EscapeSequences.java
The computer said, "Hello!"

The computer said, "Hello!"
>

```



Primitive Data Types

These data types are built into the Java programming language.

Escape Sequences

Special characters that are specified using a backslash (\) followed by a letter. These help Java indicate the character that controls the display of information on the screen.

2. String Data

Single characters are limited in what they can convey. Strings are a more flexible data type that incorporate much more information. A `String` in Java can be seen as a sequence of characters and may have a length of up to 2,147,483,647 characters.

String literals, which include string values specified in the code, are enclosed in double quotes that mark the beginning and end of the string. For instance, this line of code assigns the specified greeting text to a `String` variable:

```
String greeting = "Hello, world!";
```

It is important at this point to note that the `String` type is a class provided by the Java libraries. You will learn about Java classes in later tutorials. However, for now, remember that a `String` is a class that makes it possible to work with string data in programs. Since keywords, types, and identifiers in Java are always case sensitive, it is significant that the type name `String` begins with a capital letter. Whereas, `char` and the primitive numeric types are specified using lowercase letters, which include `int`, `long`, `float`, and `double`. When referring to the Java data type, this course will use `String` with a capital S. When talking about sequences of characters more generally, `string` will be written with a lowercase s.

Java operators will be covered in the next tutorial. For now, when working with `String` and `char` values, the `+` operator is useful for concatenating ("chaining together") values. Here is a short sample program.



TRY IT

Directions: Create a file named `Concatenation.java` and type in this code in Replit:

```
public class Concatenation {  
    public static void main(String[] args) {  
        // Joining literal String values  
        // Joining 3 pieces for demonstration purposes.  
        // This wouldn't happen in real-world code.  
        System.out.println("Hello" + ", " + "world! (1)");  
        // Joining Strings in variables  
        String hello = "Hello";  
        String comma = ", "; // Note space  
        String world = "world! (2)";  
        System.out.println(hello + comma + world);  
    }  
}
```

```
// Instead of using System.out.println() we
// could put in the new line \n.
// Note that we can change a values.
world = "world! (3)";
System.out.print(hello + comma + world + '\n');
}
}
```

The output from the program should look like this:

ConsoleShell

```

> java Concatenation.java
Hello, world! (1)
Hello, world! (2)
Hello, world! (3)
>

```

REFLECT

“Hello World” is almost always the first output produced by a programmer when learning any new programming language. It’s also great for trying out new concepts. In the code example above you’re seeing it produced by three different methods as a learning exercise. There are many other ways to produce this same output too. Soon, you’ll be using many other words and phrases in your programs particularly to capture input and to produce output which other humans will need to read, so in theory, strings will be one of the key Java primitives you will use. You can sort of think of it as “string theory” for defining the Java universe.

This example raises an important point about working with variables. The variable’s data type is only specified when the variable is declared, which may be in the first use.

The variable `world` was declared like this:

EXAMPLE

```
String world = "world! (2)";
```

When the variable’s value was changed, the data type is not included. Consider the following:

EXAMPLE

```
world = "world! (3)";
```

This is similar to the way that only the variable’s name (identifier) is used to access the data it holds (and the data type is not shown again):

EXAMPLE

```
System.out.print(hello + comma + world + '\n');
```



SUMMARY

In this lesson, you learned about using **character data** (char) and **String data** (String) in Java. You learned that strings play an important part in computer programs, including as data and as part of the user interface. You began to explore how the Java String class provides critical functionality which will be expanded further in upcoming tutorials.

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/



TERMS TO KNOW

Escape Sequences

Special characters that are specified using a backslash (\) followed by a letter. These help Java indicate the character that controls the display of information on the screen.

Primitive Data Types

These data types are built into the Java programming language.