#### The Java™ Tutorials

Hide TOC Search

Numbers and Strings

Numbers
The Numbers Classes
Formatting Numeric
Print Output
Beyond Basic
Arithmetic

Arithmetic
Summary of Numbers
Questions and Exercises
Characters
Strings

Converting Between
Numbers and Strings
Manipulating
Characters in a String
Comparing Strings and
Portions of Strings
The StringBuilder Class
Summary of Characters

The StringBuilder Class
Summary of Characters
and Strings
Autoboxing and Unboxing
Questions and Exercises

« Previous • Trail • Next »

Home Page > Learning the Java Language > Numbers and Strings

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.

See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

## **Strings**

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

#### **Creating Strings**

The most direct way to create a string is to write:

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

In this case, "Hello world!" is a *string literal*—a series of characters in your code that is enclosed in double quotes. Whenever it encounters a string literal in your code, the compiler creates a String object with its value—in this case, Hello world!.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has thirteen constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String helloString = new String(helloArray);
System.out.println(helloString);
```

The last line of this code snippet displays hello.

**Note:** The String class is immutable, so that once it is created a String object cannot be changed. The String class has a number of methods, some of which will be discussed below, that appear to modify strings. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.

### String Length

Methods used to obtain information about an object are known as *accessor methods*. One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object. After the following two lines of code have been executed, len equals 17:

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

A *palindrome* is a word or sentence that is symmetric—it is spelled the same forward and backward, ignoring case and punctuation. Here is a short and inefficient program to reverse a palindrome string. It invokes the String method charAt(i), which returns the i<sup>th</sup> character in the string, counting from 0.

```
public class StringDemo {
   public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];
        // put original string in an
        // array of chars
        for (int i = 0; i < len; i++) {
            tempCharArray[i] =
                palindrome.charAt(i);
        // reverse array of chars
        for (int j = 0; j < len; j++) {
            charArray[j] =
                tempCharArray[len - 1 - j];
        String reversePalindrome =
            new String(charArray);
        System.out.println(reversePalindrome);
}
```

Running the program produces this output:

```
doT saw I was toD
```

To accomplish the string reversal, the program had to convert the string to an array of characters (first for loop), reverse the array into a second array (second for loop), and then convert back to a string. The String class includes a method, getChars(), to convert a string, or a portion of a string, into an array of characters so we could replace the first for loop in the program above with

```
palindrome.getChars(0, len, tempCharArray, 0);
```

# Concatenating Strings

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end.

You can also use the concat() method with string literals, as in:

```
"My name is ".concat("Rumplestiltskin");
```

Strings are more commonly concatenated with the + operator, as in

```
"Hello," + " world" + "!"
which results in
```

The + operator is widely used in print statements. For example:

```
String string1 = "saw I was ";
System.out.println("Dot " + string1 + "Tod");
```

# which prints

. Dot saw I was Tod

"Hello, world!"

Such a concatenation can be a mixture of any objects. For each object that is not a String, its toString() method is called to convert it to a String.

Note: The Java programming language does not permit literal strings to span lines in source files, so you must use the + concatenation operator at the end of each line in a multi-line string. For example:

```
String quote =
   "Now is the time for all good " +
   "men to come to the aid of their country.";
```

Breaking strings between lines using the + concatenation operator is, once again, very common in print statements.

# Creating Format Strings

You have seen the use of the printf() and format() methods to print output with formatted numbers. The String class has an equivalent class method, format(), that returns a String object rather than a PrintStream object.

Using String's static format() method allows you to create a formatted string that you can reuse, as opposed to a one-time print statement. For example, instead of

« Previous • Trail • Next »

System.out.println(fs);