



Characters

Characters

Most of the time, if you are using a single character value, you will use the primitive `char` type. For example:

```
1 char ch = 'a';
2 // Unicode for uppercase Greek omega character
3 char uniChar = '\u03A9';
4 // an array of chars
5 char[] charArray = { 'a', 'b', 'c', 'd', 'e' };
```

There are times, however, when you need to use a `char` as an object—for example, as a method argument where an object is expected. The Java programming language provides a wrapper class that "wraps" the `char` in a [Character](#) object for this purpose. An object of type [Character](#) contains a single field, whose type is `char`. This [Character](#) class also offers a number of useful class (that is, static) methods for manipulating characters.

You can create a [Character](#) object with the [Character](#) constructor:

```
1 Character ch = new Character('a');
```

The Java compiler will also create a [Character](#) object for you under some circumstances. For example, if you pass a primitive `char` into a method that expects an object, the compiler automatically converts the `char` to a [Character](#) for you. This feature is called `_autoboxing_`—or *unboxing*, if the conversion goes the other way. For more information on autoboxing and unboxing, see the section Autoboxing and Unboxing.

Note: The [Character](#) class is immutable, so that once it is created, a [Character](#) object cannot be changed.

The following table lists some of the most useful methods in the [Character](#) class, but is not exhaustive. For a complete listing of all methods in this class (there are more than 50), refer to the [Character](#) API specification.

- [boolean isLetter\(char ch\)](#) and [boolean isDigit\(char ch\)](#): Determines whether the specified `char` value is a letter or a digit, respectively.
- [boolean isWhitespace\(char ch\)](#): Determines whether the specified `char` value is white space.
- [boolean isUpperCase\(char ch\)](#) and [boolean isLowerCase\(char ch\)](#): Determines whether the specified `char` value is uppercase or lowercase, respectively.
- [char toUpperCase\(char ch\)](#) and [char toLowerCase\(char ch\)](#): Returns the uppercase or lowercase form of the specified `char` value.
- [toString\(char ch\)](#): Returns a [String](#) object representing the specified character value — that is, a one-character string.

Characters and Code Points

The Java platform has supported Unicode Standard starting with JDK 1.0.2. Java SE 15 supports Unicode 13.0. The `char` data type and the [Character](#)

class are based on the original Unicode specification, which defined characters as fixed-width 16-bit entities. The Unicode Standard has since been changed to allow for characters whose representation requires more than 16 bits. The range of legal code points is now U+0000 to U+10FFFF, known as Unicode scalar value.

A `char` value is encoded with 16 bits. It can thus represent numbers from `0x0000` to `0xFFFF`. This set of characters is sometimes referred to as the *Basic Multilingual Plane (BMP)*. Characters whose code points are greater than `0xFFFF` (noted U+FFFF) are called *supplementary characters*.

A `char` value, therefore, represents Basic Multilingual Plane (BMP) code points. An `int` value represents all Unicode code points, including supplementary code points. Unless otherwise specified, the behavior with respect to supplementary characters and surrogate char values is as follows:

- The methods that only accept a `char` value cannot support supplementary characters. They treat `char` values from the surrogate ranges as undefined characters.
- The methods that accept an `int` value support all Unicode characters, including supplementary characters.

You can refer to the documentation of the [Character](#) class for more information.

Escape Sequence

A character preceded by a backslash (`\`) is an escape sequence and has special meaning to the compiler. The following table shows the Java escape sequences:

Escape Sequence	Description
<code>\t</code>	Insert a tab in the text at this point.
<code>\b</code>	Insert a backspace in the text at this point.
<code>\n</code>	Insert a newline in the text at this point.
<code>\r</code>	Insert a carriage return in the text at this point.
<code>\f</code>	Insert a form feed in the text at this point.
<code>\'</code>	Insert a single quote character in the text at this point.
<code>\"</code>	Insert a double quote character in the text at this point.
<code>\\</code>	Insert a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, `"`, on the interior quotes. To print the sentence

```
1 | She said "Hello!" to me.
```

you would write

```
1 | System.out.println("She said \"Hello!\" to me.");
```

Last update: September 14, 2021

[Numbers](#)



Characters



[Strings](#)

[Home](#) > [Tutorials](#) > [Numbers and Strings](#) > Characters