Documentation

The Java™ Tutorials

Trail: Learning the Java Language
Lesson: 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.

Characters

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

```
char ch = 'a';
// Unicode for uppercase Greek omega character
char uniChar = '\u03A9';
// an array of chars
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:

```
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 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 java.lang.Character API specification.

Useful Methods in the `Character` Class

Method	Description
`boolean isLetter(char ch)` `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)` `boolean isLowerCase(char ch)`	Determines whether the specified char value is uppercase or lowercase, respectively.
`char toUpperCase(char ch)` `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.

Escape Sequences

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 Sequences

Escape Sequence	Description
`\t`	Insert a tab in the text at this point.

`\b`	Insert a backspace in the text at this point.
`\n`	Insert a newline in the text at this point.
`\r`	Insert a carriage return in the text at this point.
`\f`	Insert a form feed in the text at this point.
`\'`	Insert a single quote character in the text at this point.
`\"`	Insert a double quote character in the text at this point.
`\\`	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

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

you would write

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


Previous page: Questions and Exercises: Numbers
Next page: Strings