

Chapter Two Bonus Lesson: JavaDoc

We've already talked about adding simple comments to your source code. The JDK actually supports more meaningful comments as well. If you add specially-formatted comments, you can then use a tool called **javadoc** to automatically create HTML files, turning your simple comments into flashy documentation!

To use JavaDoc, you will create a specific type of comment just before each class, field and method in your code. These comments will always begin with these characters: `/**` and end with these characters: `*/`. The text between these marks will contain text that describes the class, field or method in some detail. If you have multiple lines of text in your JavaDoc comment, each line will start with an asterisk `*`. This asterisk character and any whitespace before the first word in the line are ignored by the JavaDoc program when creating the HTML output. In this example we have described our main HelloWorld class using several lines of text.

```
/** TeenCoder: Java Programming<br />  
 * Chapter 02<br />  
 * The HelloWorld class will print "Hello, World" to the console<BR/>  
 */  
class HelloWorld
```

Since the text after the `*` will be output directly to HTML, we ended each line with a `
`. In HTML this marks the end of a line. You can use any HTML formatting you like within this block such as `` for bold or `<i>` for italics if you know more about HTML. In this class we'll simply use the `
` marker at the end of the line to ensure text is displayed on the HTML page the same way we see it in the source code.

JavaDoc Tags

Within a JavaDoc comment block you can add special tags that start with the symbol `@`. These tags will hold specific pieces of information that the HTML output will treat differently. One common tag is `@author`, which should be followed by a name. This tag will describe the author of the class. This tag is typically used once, at the top of your source code.

```
@author name
```

The `@version` tag is useful when you are revising a piece of code. In this case, you want to be able to give an indication that this is a newer version of the code. So, you can give it a version number, like 2.0 or 1.5 to indicate the version of the code. This tag is typically only used once at the top of a source file, just like the `@author` tag.

```
@version version
```

One of the most commonly used JavaDoc tags is the **@param** tag. This tag is used to describe a method parameter. We will discuss methods in further detail in a future lesson, but for now, you should understand that a method is a block of code that can take certain parameters as data. You can think of the parameters as the ingredients for the method. The **@param** tag will allow you to describe the name of the parameter and the data that should be contained within the parameters.

```
@param name description
```

You can use one **@param** tag for each parameter in the method that you are describing. The **@param** tag is followed by a name and a description. After any **@param** tags, the JavaDoc **@return** tag can be used to describe the data that a method (or a block of code) will be returning to the calling program.

```
@return description
```

You will only need one **@return** tag for any method that returns a value.

We have only described the most commonly used JavaDoc tags. There are many other tags you can use to increase the information available in your HTML output pages.

JavaDoc Comment Location

You must place your JavaDoc comments in particular spots, or they will be ignored. Your class-level comments should go directly above your “class” keyword and below any **import** statements. Your function-level comments should go directly above your function declaration.

```
import java.util.*; // any comments above your imports are ignored

/** class-level comments go here!
 */
class HelloWorld
{
    /** function-level comments go here!
     */
    public static void main(String args[]) // main entry point to the program
    {
    }
}
```

JavaDoc Example

If we were to add some JavaDoc comments to our “HelloWorld” program, it could look something like this:

```
import java.util.*; // any comments above your imports are ignored

/** TeenCoder: Java Programming<br />
 * Chapter 02<br />
 * The HelloWorld class will print "Hello, World" to the console<br />
 * @author Chris
 * @version 1.0 on March 30th, 2012
 */
class HelloWorld
{
    /**
     * The main function is run automatically when the program is started.
     * @param args The command-line arguments passed into the program
     */
    public static void main(String args[]) // main entry point to the program
    {
        System.out.println("Hello, World"); // a program statement!
    }
}
```

Let’s take a look at what we have done! First, notice that the top comment block is now enclosed in JavaDoc markers **/**** and ***/**. Each line within the block starts with an asterisk *****. We have also added the **@author** and **@version** tags to identify the programmer, version, and date.

Next we also added a JavaDoc comment in front of our **main()** function. The JavaDoc comment gives a brief description of the **main()** function and also identifies the parameters into the function. Since there is no data returned from the function, we have no **@return** tag.

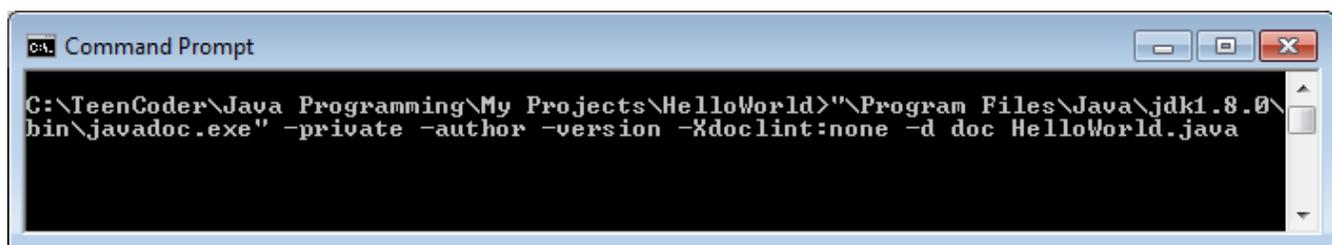
Using the JavaDoc Tool

Once we have JavaDoc comments in the source file, we can run the **javadoc** program to create the HTML output. Since the **javadoc** tool is part of the JDK like the **javac** compiler, you will run it from the command line just like you did **javac**. In a Mac OS Terminal you can type “javadoc” and the tool will be found automatically, but in a Windows Command Prompt you will use the fully qualified path to the JDK’s “bin” directory.

There are many parameters you can use to guide how Javadoc produces output. Just run **javadoc** with no parameters to see a complete list of options. The major ones we'll use are:

-private	This tells Javadoc to include all classes in your program, not just the ones declared for “public” use by other programmers.
-author	This tells Javadoc to use your <code>@author</code> tag to create an author section.
-version	This tells Javadoc to use your <code>@version</code> tag to create a version section.
-Xdoclint:none	This tells Javadoc to allow self-closing HTML tags such as <code>
</code> in your comments – only needed on JDK 8+.
-d <dir>	This tells Javadoc to send all of the HTML output files into the specified directory. If you don't use this, the Javadoc will be dumped right into your root source file directory, which can be confusing.
<source file>	At the end of all the other parameters, simply specify your source file name such as “HelloWorld.java”

On Windows we would run **javadoc** as shown below. Keep in mind your actual JDK version may be different, and the **-Xdoclint:none** parameter is only needed on JDK8 installations.

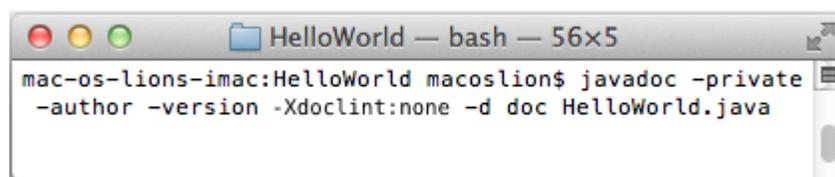


```

C:\TeenCoder\Java Programming\My Projects\HelloWorld>"\Program Files\Java\jdk1.8.0\
bin\javadoc.exe" -private -author -version -Xdoclint:none -d doc HelloWorld.java

```

The same example on Mac OS looks like this:

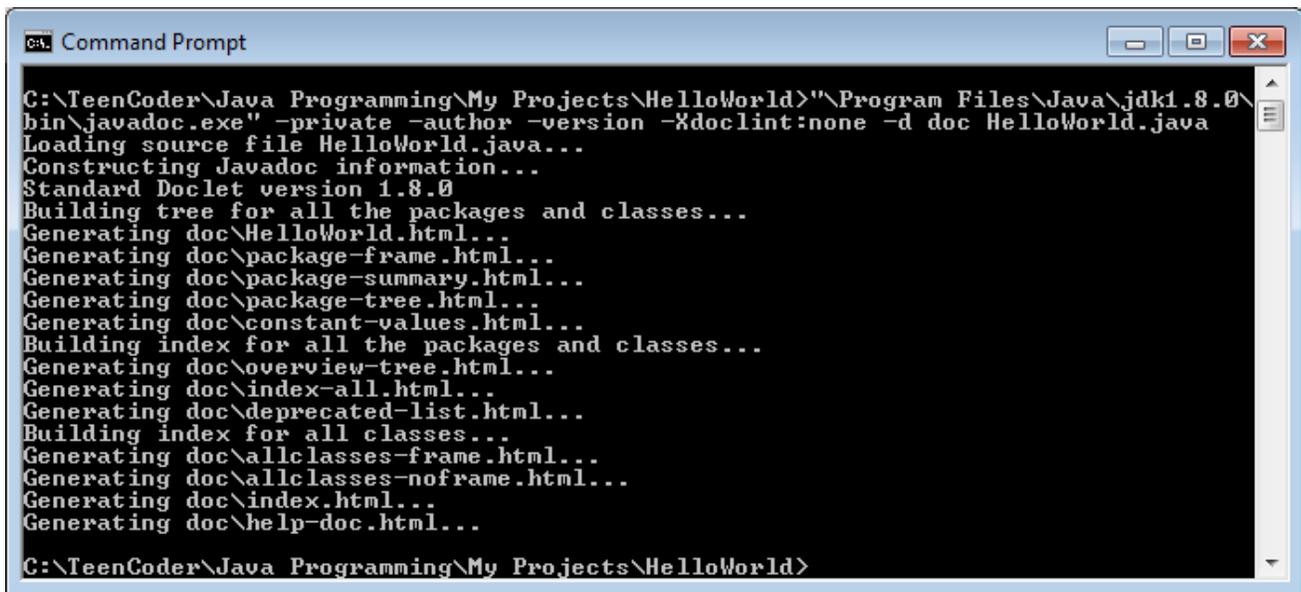


```

mac-os-lions-imac:HelloWorld macoslion$ javadoc -private
-author -version -Xdoclint:none -d doc HelloWorld.java

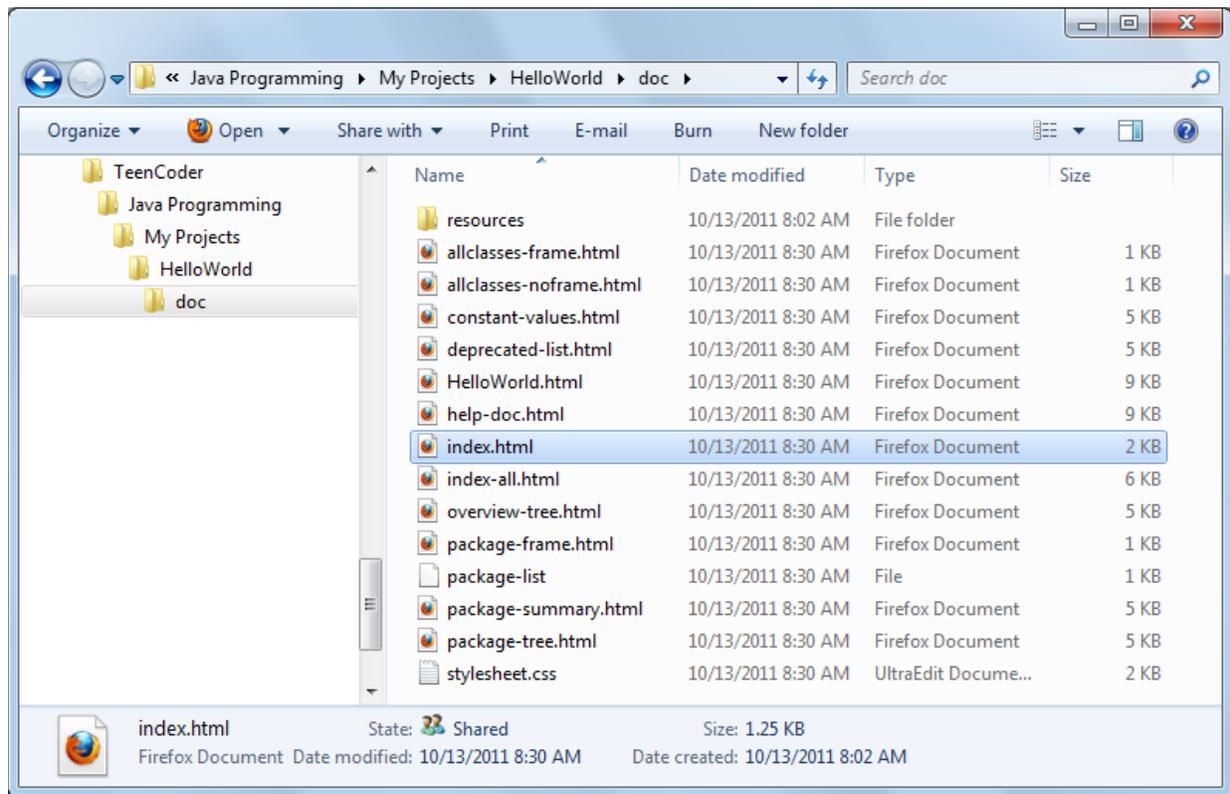
```

Once you run the **javadoc** program, you will see a bunch of output as individual HTML files are created.



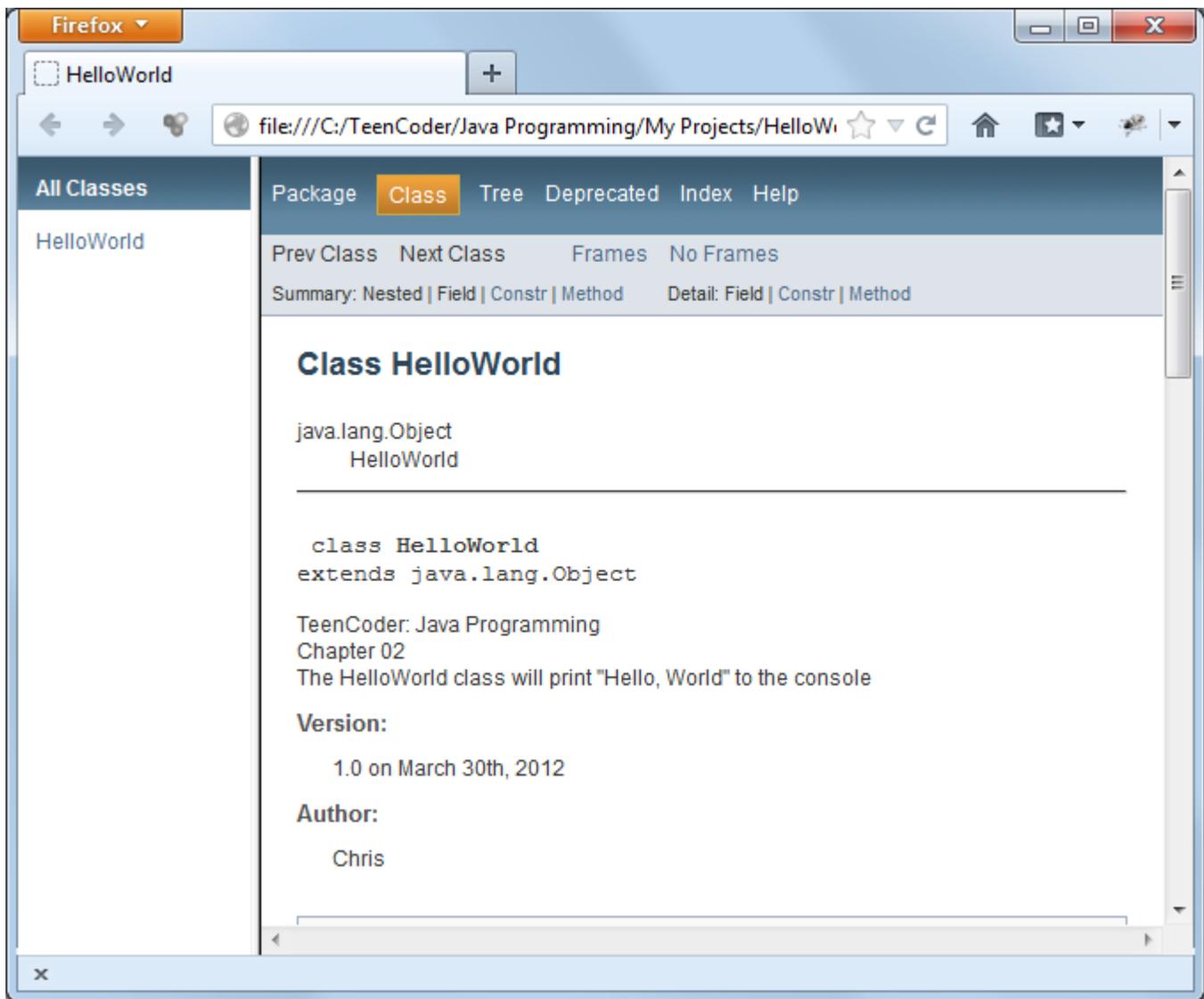
```
C:\TeenCoder\Java Programming\My Projects\HelloWorld>"\Program Files\Java\jdk1.8.0\bin\javadoc.exe" -private -author -version -Xdoclint:none -d doc HelloWorld.java
Loading source file HelloWorld.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0
Building tree for all the packages and classes...
Generating doc\HelloWorld.html...
Generating doc\package-frame.html...
Generating doc\package-summary.html...
Generating doc\package-tree.html...
Generating doc\constant-values.html...
Building index for all the packages and classes...
Generating doc\overview-tree.html...
Generating doc\index-all.html...
Generating doc\deprecated-list.html...
Building index for all classes...
Generating doc\allclasses-frame.html...
Generating doc\allclasses-noframe.html...
Generating doc\index.html...
Generating doc\help-doc.html...
C:\TeenCoder\Java Programming\My Projects\HelloWorld>
```

Or you may see descriptive error messages if it found something in your comments that did not meet the Javadoc standard. Notice we specified “doc” as our output directory, so after the program runs we will have a new “doc” subdirectory underneath the “HelloWorld” directory. The image below shows an example list of the output files in the “doc” directory as seen in Windows Explorer (details may vary):



That’s quite a lot of files to document one simple class! But you can simply pull up the “index.html” page in your favorite web browser to see the documentation for your program (no Internet connection required).

Here's what our "index.html" page looks like in a typical web browser:



The top part shows our "HelloWorld" class description, author, and version. Notice the overall framework is pretty fancy and can handle multiple classes listed on the left side, and has a number of tabs across the top as well. While this may be overkill for one simple class, if you are documenting many classes such as the Java class library, all the extra organization really comes in handy.

If you scroll down on the HTML page you will see additional information.

We haven't talked about "constructors" yet so you can ignore those sections, but notice that your `main()` function is clearly described and has a hyperlink that will take you right to the full definition.

Constructor Summary

Constructors

Constructor and Description
HelloWorld()

Method Summary

Methods

Modifier and Type	Method and Description
static void	main(java.lang.String[] args) The main function is run automatically when the program is started.

Method Detail

main

```
public static void main(java.lang.String[] args)
```

The main function is run automatically when the program is started.

Parameters:

`args` - The command-line arguments passed into the program

Finally if you keep scrolling towards the bottom you will find the documentation for each of your individual class functions. In our case we have only one function, `main()`, which appears with its descriptive method text and parameter description.

Package **Class** Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) Detail: [Field](#) | [Constr](#) | [Method](#)

As you can see, JavaDoc is a very powerful tool that can produce complete documentation on your program, all for the price of a few specially formatted comments in your code. Following the JavaDoc standard is especially important when you are writing classes that you expect other programmers to understand and use.