

Chapter 13

JavaDoc

Hello!

JavaDoc is a tool that helps us generate documentation for our code; specially formatted comments allow the javadoc tool to automatically generate good API documentation directly from the source files.

JavaDoc: Automated API Documentation Generation

Have you noticed how detailed and organized the API documentation is for the standard Java libraries? If you google any Java class, you see a nice web page with descriptions of the inheritance involved, all known fields/constructors/methods, and nice detailed descriptions of what each method does, down to the purpose of each parameter, links to other classes of interest, and maybe some good examples or discussion text. It probably took quite a bit of time for someone to make all those webpages, huh?

Wait a minute, though – Java was implemented by programmers. Programmers love writing programs to do menial tasks for them. Poke around a bit more, and you'll find that many other large-scale projects written in Java have quite similar-looking documentation, other than perhaps a more visually pleasing cosmetic touch. How did everybody get this cool documentation, and can we get in on it??

They used Javadoc, and we can most definitely get in on it! It turns out that we can write comments in a very specific style, and then a tool (available on the command line as `javadoc`) can scrape all of our source files and generate the entire batch of html files, with many different flags and options available to direct what we actually want included in our API webpages.

Javadoc comments are still just regular Java multi-line comments, but they will tend to look like this:

```
/**
 * Some short summary description goes here.
 *
 * @someAttribute    identifier    more info about this one.
 * @someAttribute    identifier    more info about this next one...
 */
```

Where can we add these Javadoc comments? Pretty much anywhere. We mostly want to add them at:

- the package level
- the class level
- the member level (both fields and methods)

After we've written some marvelous package in Java, and perhaps also added wonderfully descriptive javadoc comments all over the place, we can then invoke the `javadoc` tool to generate all our API documentation. Perhaps it would look like this example:

```
javadoc -d /Users/me/htmlDir -subpackages Foo
```

It requests all generated files to be placed at /Users/me/htmlDir, and is asking for all packages starting with Foo (and all contained subpackages) to be included in the documentation. There are many more options; this is a rather sparse usage to introduce the barest of basics.

Setup: Find and download `javadocExample.zip` (here: <http://muddsnyder.com/211/javadocExample.zip>). Open up all the .java files in it, and inspect them. Nice little package structure, doing a little math on the side. It's a bit of a silly example, but it's got all the pieces we need to try making javadoc documentation with it. You might want to draw out the package hierarchy as well as the classes hierarchy to get a feel for what this example contains.

Basic javadoc Generation

Javadoc will be generating documentation based on the structure of our code. We will first learn how to invoke the javadoc command, and then we will also learn how to write javadoc-style comments, so that the generated output has much more textual, descriptive information included in it.

Invoking javadoc: a few scenarios.

We will be asking the javadoc command to generate a significant number of files for us to implement all the HTML documents for us, so let's make a folder for them (which the lab will assume is named `htmlDir`). You might just put it adjacent to the package folder for now.

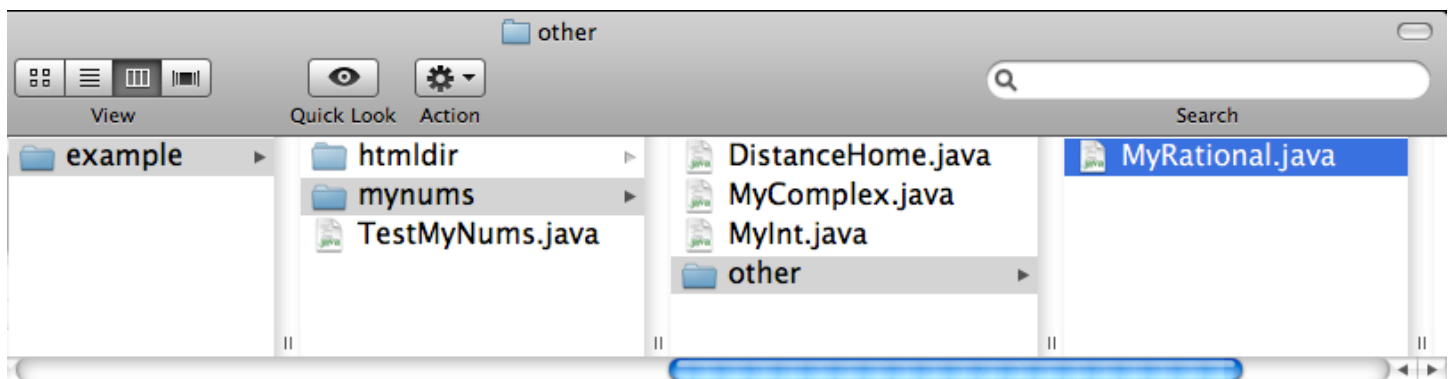


Image description: File directory found in the javadoc example zip file.

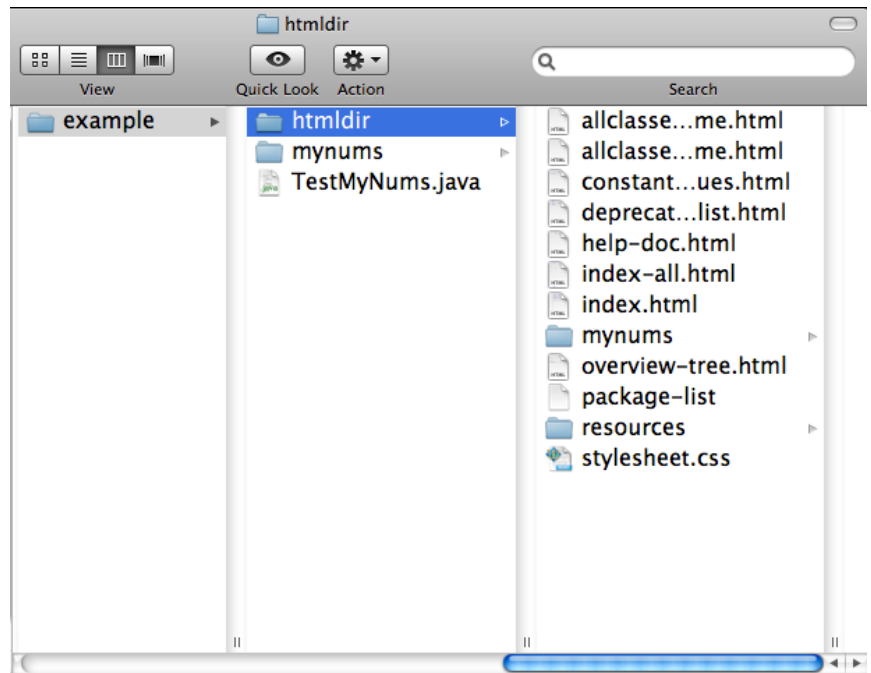
Assuming you have that directory structure, open up the terminal and navigate into the `example` folder. We will start off with almost the simplest invocation of javadoc:

```
javadoc -d htmlDir mynums
```

This (`-d htmlDir`) instructs javadoc to place all generated files in the directory `htmlDir`; and then we requested that just the `mynums` package is documented. If we look inside the `htmlDir` folder, we now see many generated files!

→ **Word of warning:** cut-pasting from this document might use weird characters, like a – instead of a -. Re-typing them is a good idea. Always double check that there are zero errors or warnings after regenerating the documentation whenever you invoke `javadoc!` ←

Image description: Contents of the `htmldir` directory.



Explore what we generated; variations on a theme

Under that `htmldir`, open `index.html`; this is the only file in there that we are interested in opening (the rest get used by this). You should see the familiar API layout as the regular Java class documentation, such as what you'd find as the top hits by googling "Java Scanner", "Java InputMismatchException", and so on.

We had a second package, though: `mynums.other` was also a (sub)package. Let's get that generated, too:

```
javadoc -d htmldir mynums mynums.other
```

This would get a bit tedious if we had many extra subpackages: we can also request all nested subpackages be included via the `-subpackages` option:

```
javadoc -d htmldir -subpackages mynums
```

Better! And if we have any extra java files we wanted to include (perhaps ones that are implicitly in the default package), you can just list them at the end:

```
javadoc -d htmldir -subpackages mynums Other.java
```

When you used the default package only, you might use this command (useful on some of our earlier projects, most likely):

```
javadoc -d htmldir *.java
```

Visibility-Level Inclusions

But now, take a closer look at the `MyRational` class's documentation. Where is our denominator? It was the default package-level visibility, but we don't see it here. It turns out that we can set a threshold of what level of visibility things to document:

Flag	What levels of visibility are documented? (everything up to the listed visibility)
------	---

-public	public
-protected	public, protected
-package	public, protected, package
-private	public, protected, package, private

-protected is the default option. Let's use -private to make everything visible (as backwards as that sounds: "-private, so show everything!" Think of it as, "show everything as sensitive as [private]").):

```
javadoc -d htmldir -private -subpackages mynums
```

Okay, now we re-visit the HTML document (just reload index.html), and see that `MyRational` now has its `(package private) int denominator` documented. Similarly, we can see `private int imag` in the `MyComplex` class. We actually have one of each level visibility in these packages (on purpose just for this section of the lab):

visibility:	package.Class:		member
public:	mynums.MyInt	→	public String garbage
protected:	mynums.MyInt	→	protected int val
package:	mynums.other.MyRational	→	int denominator
private:	mynums.MyComplex	→	private int imag

There are of course many more options we can use to change the generated documentation. But this is enough to get you started with the `javadoc` tool. We next need to learn how to add comments in a style that `javadoc` expects, so that it reads more like actual documentation (human-written, with sentences and links and things) instead of just exposing the structure of the code. In short, we want it to say not just *what* our code is, but also *why* it is doing things.

Adding Javadoc-style Comments

Even though `javadoc` can do a lot of useful work without any `javadoc` comments, the generated documentation is far richer when we embed descriptions throughout our code. These `javadoc` comments look like regular multi-line comments, and just so happen to have a double-star at the beginning:

```
/**
 * This is a javadoc comment. It basically always has a short
 * summary/description on the first few lines.
 */
```

The stars on the interior lines are not necessary any more. Just like your experience writing code so far, even though comment-writing isn't exactly the most mentally stimulating part of the coding process, it

still takes a significant amount of time. Your main interaction with Javadoc will be writing javadoc-style comments, and not the command-line documentation generation step that we just learned. But it helps to know why we're writing all of these specially formatted comments.

There are a number of places we will put these comments:

- **members** of classes (both fields and methods)
- **class** definitions themselves
- **packages** (in a special file named package-info.java, just inside the package folder).

Method Comments

Place the following comment in the `MyInt` class, immediately preceding the `addIn(MyInt other)` method.

```
/**
 * addIn accepts another MyInt object, whose value is
 * absorbed into this one.
 *
 * @param other the other number to get some value from.
 */
```

Note the empty line between the short description and the `@param` line; **it needs to be there**.

Regenerate the documentation, and inspect your Method Summary section for the `MyInt` class. The description was directly pulled into the documentation. Click through to the detailed method description, and you'll also see the **Parameters** section listing out our single parameter.

Let's inspect the `@return` tag next.

```
/**
 * returns the value contained in this MyInt.
 *
 * @return the current value in the instance variable val.
 */
```

Regenerate the documentation, and view this method's output as well. There are plenty of other `@tags` that we can use; we are only going to try out a few in this tutorial. You can get much more exhaustive coverage from Oracle here:

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#javadoctags>

Annotation	Identifier	Text
@param	identifierName	very short description of its purpose
@return	[none]	short description of what is returned
@throws	className	short description of when/why it would be thrown
@exception	className	actually just a synonym for the @throws tag
@see	relevantPlace	many allowed formats: for classes, members, strings, HTML, etc.
@author	name	
@version	someText	

We can of course have multiple `@param` lines, multiple `@see` lines, and multiple `@throws` (or `@exception`) lines. But there will be only zero or one line for any of `@author`, `@version`, `@return` (not used with `void`). It's even possible to create your own tags, a feature we won't explore here.

Your Turn!

You've been making a lot of modifications all along anyways, but now let's try some of these (and then regenerate the documentation each time to check our work):

- add a javadoc comment to the `addedTo` method of `MyInt`.
 - add a javadoc comment to the constructor for any of our three classes (`MyInt`, `MyComplex`, `MyRational`).
 - add a method `public double slope()` to the `MyRational` class; write a javadoc comment, and try out the `@throws` tag. This method returns `val/denominator`, so can you guess what exception might be thrown?
-

Field Comments

Class fields can also have javadoc comments. There aren't as many tags that we tend to use; in fact, we might not use any, opting just for a short description. (See the above link and scroll down a bit to see all the ones that are allowed for fields at all).

Change the plain old comment near the `garbage` field in `MyInt` to a javadoc comment:

```
/** this is just here so we have something public...
 * Basically a wall for people to write graffiti.
 */
```

We tend to just give a description for these. We might choose to include a few `@see` tags too.

Your Turn!

- add javadoc tags to the various fields in our three classes; regenerate the documentation and view them.
-

Class Comments

We can provide javadoc comments for an entire class. Try adding this to the `MyInt` class, directly above the `public class MyInt` line.

```
/**
 * the MyInt class is just a thin wrapper around an int,
 * plus some unnecessary extra functionality. It's really
 * just here for our javadoc tutorial.
 *
 * @since about two minutes ago, mynums v0.1
 * @author Yours Truly
 */
```

Regenerate the documentation, and find where this shows up. So far so good, except that our `author` tag wasn't used! We can request it by adding `-author` to our javadoc invocation:

```
javadoc -d htmldir -private -author -subpackages mynums
```

We might just have the description here, similar to fields. If anything, the `@since` and `@author` tags are likely to be used here. (Note: we can't split up `-d` and its directory, or `-subpackages` and its package; go ahead and put `-author` just after `javadoc`, or some other place where it doesn't disrupt these "pairs" of command-line arguments).

Your Turn!

- Document the other classes.
 - Document the interface. It works just like the classes.
-

Package Comments

There is one last place we are interested in adding `javadoc` comments: comments for an entire package. But we have one hurdle to overcome: there's no specific file that represents the entire package! Each class or enumeration or interface has its own file, but our packages are just a manifestation of the folder that contains various things that include package statements matching the name of the package in mind.

To solve this issue, we will add an almost-empty file, directly in the package folder. It must always be named `package-info.java`, and it needs to have a single package statement in it (identical to the package statements found in all other classes and whatnot that are also in this package).

Your Turn!

Add the following text to the file `package-info.java`, directly in the `mynums` folder.

```
/**
 * The mynums pacakage is just a silly package example
 * that we are using to learn about javadoc documentation.
 */
package mynums;
//nothing else needs to be in this file.
```

Regenerate the documentation, and you'll immediately see the documented `mynums` package.

Your Turn!

- Document the `mynums.other` package in the same fashion.
 - Try using `javadoc` on your most recent project, and also on any packages examples from class.
-

Going Further

There is plenty more that can be done with `javadoc`. At this point, it's probably best to just learn about features as you need them in your own uses of `javadoc`, perhaps on the job some day in the near future!

Description of the various `@tags`

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#javadoctags>

Much more info:

<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>

Options for the javadoc command:

http://www.java2s.com/Tutorial/Java/0020__Language/ThejavadocTool.htm

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#runningjavadoc>

Doclets can be written to generate other outputs such as pdfs, personalized/better(?) HTML, etc.

<http://docs.oracle.com/javase/1.4.2/docs/tooldocs/javadoc/overview.html>