Chapter 6: Documenting Java Code

Documentation for java code is often generated using javadoc. Javadoc was created by Sun Microsystems for the purpose of generating API documentation in HTML format from java source code. Using the HTML format gives the convenience of being able to hyperlink related documents together.

Section 6.1: Building Javadocs From the Command Line

Many IDEs provide support for generating HTML from Javadocs automatically; some build tools (Maven and Gradle, for example) also have plugins that can handle the HTML creation.

However, these tools are not required to generate the Javadoc HTML; this can be done using the command line `javadoc` tool.

The most basic usage of the tool is:

```
javadoc JavaFile.java
```

Which will generate HTML from the Javadoc comments in `JavaFile.java`.

A more practical use of the command line tool, which will recursively read all java files in `[source-directory]`, create documentation for `[package.name]` and all sub-packages, and place the generated HTML in the `[docs-directory]` is:

```
javadoc -d [docs-directory] -subpackages -sourcepath [source-directory] [package.name]
```

Section 6.2: Class Documentation

All Javadoc comments begin with a block comment followed by an asterisk (`/**`) and end when the block comment does (`*/`). Optionally, each line can begin with arbitrary whitespace and a single asterisk; these are ignored when the documentation files are generated.

```
/**
 * Brief summary of this class, ending with a period.
 *
 * It is common to leave a blank line between the summary and further details.
 * The summary (everything before the first period) is used in the class or package
 * overview section.
 *
 * The following inline tags can be used (not an exhaustive list):
 * {@link some.other.class.Documentation} for linking to other docs or symbols
 * {@link some.other.class.Documentation Some Display Name} the link's appearance can be
 * customized by adding a display name after the doc or symbol locator
 * {@code code goes here} for formatting as code
 * {@literal <>[]()foo} for interpreting literal text without converting to HTML markup
 * or other tags.
 *
 * Optionally, the following tags may be used at the end of class documentation
 * (not an exhaustive list):
 *
 * @author John Doe
 * @version 1.0
 * @since 5/10/15
 * @see some.other.class.Documentation
 * @deprecated This class has been replaced by some.other.package.BetterFileReader
 *
 * You can also have custom tags for displaying additional information.
 * Using the @custom.<NAME> tag and the -tag custom.<NAME>:htmltag:"context"
 * command line option, you can create a custom tag.
 *
 * Example custom tag and generation:
 * @custom.updated 2.0
 * Javadoc flag: -tag custom.updated:a:"Updated in version:"
 * The above flag will display the value of @custom.updated under "Updated in version:"
 *
 */
public class FileReader {
}
```

The same tags and format used for `Classes` can be used for `Enums` and `Interfaces` as well.

Section 6.3: Method Documentation

All Javadoc comments begin with a block comment followed by an asterisk (`/**`) and end when the block comment does (`*/`). Optionally, each line can begin with arbitrary whitespace and a single asterisk; these are ignored when the documentation files are generated.

```
/**
 * Brief summary of method, ending with a period.
 *
 * Further description of method and what it does, including as much detail as is
 * appropriate.  Inline tags such as
 * {@code code here}, {@link some.other.Docs}, and {@literal text here} can be used.
 *
 * If a method overrides a superclass method, {@inheritDoc} can be used to copy the
 * documentation
 * from the superclass method
 *
 * @param stream Describe this parameter.  Include as much detail as is appropriate
 *               Parameter docs are commonly aligned as here, but this is optional.
 *               As with other docs, the documentation before the first period is
 *               used as a summary.
 *
 * @return Describe the return values.  Include as much detail as is appropriate
 *         Return type docs are commonly aligned as here, but this is optional.
 *         As with other docs, the documentation before the first period is used as a
 *         summary.
 *
 * @throws IOException Describe when and why this exception can be thrown.
 *                     Exception docs are commonly aligned as here, but this is
 *                     optional.
 *                     As with other docs, the documentation before the first period
 *                     is used as a summary.
 *                     Instead of @throws, @exception can also be used.
 *
 * @since 2.1.0
 * @see some.other.class.Documentation
 * @deprecated  Describe why this method is outdated. A replacement can also be specified.
 */
public String[] read(InputStream stream) throws IOException {
    return null;
}
```

Section 6.4: Package Documentation

Version ≥ Java SE 5

It is possible to create package-level documentation in Javadocs using a file called `package-info.java`. This file must be formatted as below. Leading whitespace and asterisks optional, typically present in each line for formatting reason

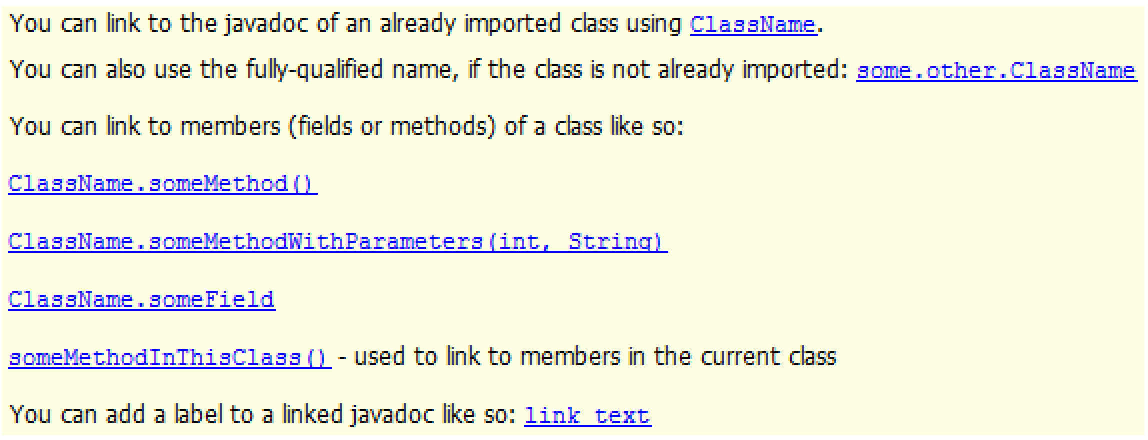
```
/**
 * Package documentation goes here; any documentation before the first period will
 * be used as a summary.
 *
 * It is common practice to leave a blank line between the summary and the rest
 * of the documentation; use this space to describe the package in as much detail
 * as is appropriate.
 *
 * Inline tags such as {@code code here}, {@link reference.to.other.Documentation},
 * and {@literal text here} can be used in this documentation.
 */
package com.example.foo;

// The rest of the file must be empty.
```

In the above case, you must put this file `package-info.java` inside the folder of the Java package `com.example.foo`.

Section 6.5: Links

Linking to other Javadocs is done with the `@link` tag:

```
/**
 * You can link to the javadoc of an already imported class using {@link ClassName}.
 *
 * You can also use the fully-qualified name, if the class is not already imported:
 *  {@link some.other.ClassName}
 *
 * You can link to members (fields or methods) of a class like so:
 *  {@link ClassName#someMethod()}
 *  {@link ClassName#someMethodWithParameters(int, String)}
 *  {@link ClassName#someField}
 *  {@link #someMethodInThisClass()} - used to link to members in the current class
 *
 * You can add a label to a linked javadoc like so:
 *  {@link ClassName#someMethod() link text}
 */
```


You can link to the javadoc of an already imported class using ClassName.

You can also use the fully-qualified name, if the class is not already imported: some.other.ClassName

You can link to members (fields or methods) of a class like so:

ClassName.someMethod()

ClassName.someMethodWithParameters(int, String)

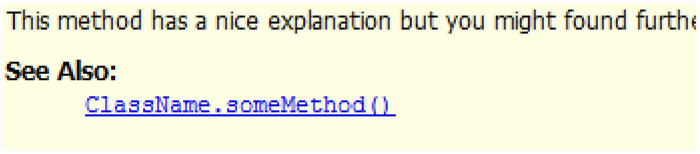
ClassName.someField

someMethodInThisClass() - used to link to members in the current class

You can add a label to a linked javadoc like so: link text

With the `@see` tag you can add elements to the *See also* section. Like `@param` or `@return` the place where they appear is not relevant. The spec says you should write it after `@return`.

```
/**
 * This method has a nice explanation but you might found further
 * information at the bottom.
 *
 * @see ClassName#someMethod()
 */
```



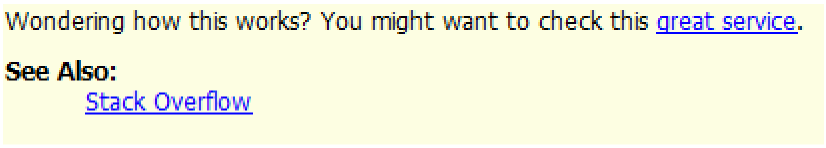
This method has a nice explanation but you might found furthe

See Also:

ClassName.someMethod()

If you want to add **links to external resources** you can just use the HTML `<a>` tag. You can use it inline anywhere or inside both `@link` and `@see` tags.

```
/**
 * Wondering how this works? You might want
 * to check this <a href="http://stackoverflow.com/">great service</a>.
 *
 * @see <a href="http://stackoverflow.com/">Stack Overflow</a>
 */
```



Wondering how this works? You might want to check this great service.

See Also:

Stack Overflow

Section 6.6: Code snippets inside documentation

The canonical way of writing code inside documentation is with the `{@code }` construct. If you have multiline code wrap inside `<pre></pre>`.

```
/**
 * The Class TestUtils.
 * <p>
 * This is an {@code inline("code example")}.
 * <p>
 * You should wrap it in pre tags when writing multiline code.
 * <pre>{@code
 *  Example example1 = new FirstLineExample();
 *  example1.butYouCanHaveMoreThanOneLine();
 * }</pre>
 * <p>
 * Thanks for reading.
 */
class TestUtils {
```

Sometimes you may need to put some complex code inside the javadoc comment. The `@` sign is specially problematic. The use of the old `<code>` tag alongside the `{@literal }` construct solves the problem.

```
/**
 * Usage:
 * <pre><code>
 * class SomethingTest {
 * {@literal @}Rule
 *  public SingleTestRule singleTestRule = new SingleTestRule("test1");
 *
 * {@literal @}Test
 *  public void test1() {
 *      // only this test will be executed
 *  }
 *
 *  ...
 * }
 * </code></pre>
 */
class SingleTestRule implements TestRule { }
```

Section 6.7: Field Documentation

All Javadoc comments begin with a block comment followed by an asterisk (`/**`) and end when the block comment does (`*/`). Optionally, each line can begin with arbitrary whitespace and a single asterisk; these are ignored when the documentation files are generated.

```
/**
 * Fields can be documented as well.
 *
 * As with other javadocs, the documentation before the first period is used as a
 * summary, and is usually separated from the rest of the documentation by a blank
 * line.
 *
 * Documentation for fields can use inline tags, such as:
 * {@code code here}
 * {@literal text here}
 * {@link other.docs.Here}
 *
 * Field documentation can also make use of the following tags:
 *
 * @since 2.1.0
 * @see some.other.class.Documentation
 * @deprecated Describe why this field is outdated
 */
public static final String CONSTANT_STRING = "foo";
```

Section 6.8: Inline Code Documentation

Apart from the Javadoc documentation code can be documented inline.

Single Line comments are started by `//` and may be positioned after a statement on the same line, but not before.

```
public void method() {

  //single line comment
  someMethodCall(); //single line comment after statement

}
```

Multi-Line comments are defined between `/*` and `*/`. They can span multiple lines and may even been positioned between statements.

```
public void method(Object object) {

  /*
```

```
    multi
    line
    comment
  */
  object/*inner-line-comment*/.method();
}
```

JavaDocs are a special form of multi-line comments, starting with `/**`.

As too many inline comments may decrease readability of code, they should be used sparsely in case the code isn't self-explanatory enough or the design decision isn't obvious.

An additional use case for single-line comments is the use of TAGs, which are short, convention driven keywords. Some development environments recognize certain conventions for such single-comments. Common examples are

- `//TODO`
- `//FIXME`

Or issue references, i.e. for Jira

- `//PRJ-1234`