Javadoc
The Java API Documentation Generator

⇒ a tool that

- parses the declarations and documentation comments in a set of source files and
- produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.
- produces one complete document each time it is run
- cannot modify or directly incorporate results from previous runs of Javadoc
- cannot do incremental builds
- can link to results from previous runs.
- rely on the compiler → the HTML output corresponds exactly with the actual implementation, which may rely on implicit, rather than explicit, source code.
- calls part of javac to compile the declarations, ignoring the member implementation
- will run on .java source files that are pure stub files with no method bodies → can write documentation comments and run Javadoc in the earliest stages of design while creating the API, before writing the implementation
- must be able to find all referenced classes, whether bootstrap classes, extensions, or user classes

Arguments

can be in any order

- options ⇒ Command-line options
- packagenames ⇒ A series of names of packages, separated by spaces
    - Javadoc uses -sourcepath to look for these package names.
    - Javadoc does not recursively traverse subpackages.
    - Wildcards such as asterisks (*) are not allowed
- sourcefiles ⇒ A series of source file names, separated by spaces
    - each of which can begin with a path and contain a wildcard such as asterisk (*)
    - The path that precedes the source file name determines where javadoc will look for it
        - Javadoc does not use -sourcepath to look for these source file names.
    - passing in fileName.java is identical to .\fileName.java
    - can also mix packagenames and sourcefiles
- @files ⇒ One or more files that contain packagenames and sourcefiles in any order, one name per line.

DOCUMENTATION COMMENTS for source code

- ahead of declarations for any entity (classes, interfaces, methods, constructors, or fields) → Javadoc comments
- consists of the characters between
    - the characters /** that begin the comment and
    - the characters */ that end it.
- can continue onto multiple lines.
- can put a comment on one line
- are recognized only when placed immediately before class, interface, constructor, method, or field declarations
- Documentation comments placed in the body of a method are ignored
- Only one documentation comment per declaration statement is recognized
- Don’t put an import statement between the class comment and the class declaration → Javadoc will ignore the class comment.
- comment ⇒ description followed by tags
- The text must be written in HTML
    - should use HTML entities
    - can use HTML tags
    - less-than (<) → &lt
    - greater-than (>) → &gt
    - ampersand (&)→ &amp
    - When writing documentation comments for members, it's best not to use HTML heading tags such as <H1> and <H2>, because Javadoc creates an entire structured document and these structural tags might interfere with the formatting of the generated document.
- **Leading asterisks** →
    - leading asterisk (*) characters on each line are discarded
    - blanks and tabs preceding the initial asterisk (*) characters are also discarded
    - If you omit the leading asterisk on a line, all leading white space is removed
- **First sentence** → a summary sentence, containing a concise but complete description of the declared entity
    - ends at
        - the first period that is followed by
            - a blank,
            - tab, or
            - line terminator, or
        - the first tag
    - Javadoc copies this first sentence to the member summary at the top of the HTML page
- A declaration with multiple fields
    - can have only one documentation comment which is copied for all fields
    - if you want individual documentation comments for each field, you must declare each field in a separate statement.
- **inheriting comments** ⇒ Automatic re-use of method comments
  If a method m1() in a class or interface has no doc comment or tags, Javadoc will instead use the comment and tags from method m2()
  it either overrides or implements, if any.
    - Javadoc will generate a subheading "Overrides" in the documentation for m(), with a link to the method it is

overriding
- When a method m() in a class overrides a method in a superclass
- When a method m() in an interface overrides a method in a superinterface

- Javadoc will generate a subheading "Specified by" in the documentation for m(), with a link to the method it is implementing.
  - When m(), a method in a class implements a method in an interface

description begins after the starting delimiter /** and continues until the tag section

- cannot continue after the tag section begins

Tag

⇒ a special keyword within a doc comment that Javadoc can process

- The tag section starts with the first character @ that begins a line (ignoring leading asterisks, white space and comment separator).
- can be any number of tags
- some types of tags can be repeated while others cannot.
- enable you to autogenerate a complete, well-formatted API
- start with an "at" sign (@)
- are case-sensitive
- must start at the beginning of a line (after any leading spaces and an optional asterisk) or it is treated as normal text
- convention: tags with the same name are grouped together

standard tags ⇒ @tag

- must appear at the beginning of a line, ignoring leading asterisks, white space and comment separator (/**)
  - you can use the @ character elsewhere in the text and it will not be interpreted as the start of a tag
  - If you want to start a line with the @ character and not have it be interpreted, use the HTML entity **&#064**

in-line tags ⇒ {@tag}

- allowed and interpreted anywhere that text is allowed.

@author	**@author** *name-text* • Adds when the -author option is used • may contain multiple @author tags • can specify o one name per @author tag or → Javadoc inserts a comma (,) and space between names o multiple names per tag → the entire text is simply copied to the generated document without being parsed
{@docRoot}	⇒ the relative path to the generated document's (destination) root directory from any generated page
@deprecated	**@deprecated** *deprecated-text* • The first sentence of *deprecated-text* should at least tell the user when the API was deprecated and what to use as a replacement. • You should include a {@link} tag (for Javadoc 1.2 or later) that points to the replacement API:
@exception	**@exception** *class-name description* • a synonym for @throws
{@ link}	**{@link** *package.class#member label***}** • Inserts an in-line link with visible text *label* • use &#125 for "}" inside the label • no limit to the number of {@link} tags allowed in a sentence • can use this tag o in the description part of a documentation comment o in the text portion of any tag
@param	**@param** *parameter-name description* • Adds a parameter to the "Parameters" section • description may be continued on the next line
@return	**@return** *description* • should describe the return type and permissible range of values
@see	**@see** *reference* • Adds a "See Also" heading with a link or text entry that points to *reference* • A doc comment may contain any number of @see tags, which are all grouped under the same heading. **@see** "*string*" **@see** <a href="*URL#value*">*label*</a> **@see** *package.class#member label*

- *label*
  - can contain whitespace
  - If *label* is omitted, then *package.class.member* will appear, suitably shortened relative to the current class and package
- ***package.class#member***
  - replace the dot ahead of the member name with a hash character (#)
  - If this name is in the documented classes, Javadoc will automatically create a link to it
  - To create links to external referenced classes, use the -link option
  - can be fully-qualified or partially-qualified
  - If less than fully-qualified, Javadoc uses the normal Java compiler search order to find it
  - can contain whitespace within parentheses, such as between method arguments.
- A space is the delimiter between *package.class#member* and *label*
- spaces may be used between parameters in a method
- different forms of the name

  Class → class or interface

  Type → class, interface, array, or primitive,

  method → method or constructor.
  - **Referencing a member of the current class**
    - @see *#field*
    - @see *#method(Type, Type,...)*
    - @see *#method(Type argname, Type argname,...)*
  - **Referencing another class in the current or imported packages**
    - @see *Class#field*
    - @see *Class#method(Type, Type,...)*
    - @see *Class#method(Type argname, Type argname,...)*
    - @see *Class*
  - **Referencing another package** (fully qualified)
    - @see *package.Class#field*
    - @see *package.Class#method(Type, Type,...)*
    - @see *package.Class#method(Type argname, Type argname,...)*
    - @see *package.Class*
    - @see *package*

@since	**@since** *since-text*
@serial	**@serial** *field-description*
@serialField	**@serialField** *field-name field-type field-description*
@serialData	**@serialData** *data-description*
@throws	**@throws** *class-name description* • @throws and @exception tags are synonyms
@version	**@version** *version-text* • normally refers to the version of the software (such as the Java 2 SDK) that contains this class or member.

WHERE TAGS CAN BE USED

All comments	• @see • @link • @since • @deprecated
Overview	• @see • {@link} • @since
Package	• @see • {@link} • @since • @deprecated
Class and Interface	• @see • {@link} • @since • @deprecated • @author • @version
Field	• @see • {@link} • @since • @deprecated • @serial • @serialField
Constructor and Method	• @see • {@link} • @since • @deprecated • @param • @return • @throws (@exception) • @serialData

OPTIONS for command line argument

- option names are case-insensitive, though their arguments can be case-sensitive

Javadoc Options

- **-overview** *path\filename*
- **-public**
  - Shows only public classes and members.
- **-protected**
  - Shows only protected and public classes and members.
  - This is the default
- **-package**
  - Shows only package, protected, and public classes and members
- **-private**
  - Shows all classes and members
- **-help**
  - Displays the online help, which lists javadoc and doclet command line options.
- **-doclet** *class*
  - If not used, javadoc uses the standard doclet for generating the default HTML format
- **-docletpath** *classpathlist*
- **-sourcepath** *sourcepathlist*
- **-classpath** *classpathlist*
- **-bootclasspath** *classpathlist*
- **-extdirs** *dirlist*
- **-verbose**
  - Provides more detailed messages while javadoc is running
  - causes the printing of additional messages specifying the number of milliseconds to parse each java source file
- **-locale** *language_country_variant*
  - must be placed *ahead* (to the left) of any options
  - the only command-line option that is order-dependent
- **-encoding** *name*
  - If not specified, the platform default converter is used.

Options Provided by the Standard Doclet

- **-d** *directory*
  - Specifies the destination directory where javadoc saves the generated HTML files
  - "d" means "destination."
  - Omitting this option causes the files to be saved to the current directory
  - value *directory* can be absolute or relative to the current working directory
- **-use**
- **-version**
  - Includes the @version text in the generated docs
  - This text is omitted by default
- **-author**
  - Includes the @author text in the generated docs.
- **-splitindex**
- **-windowtitle** *title*
- **-doctitle** *title*
- **-header** *header*
- **-footer** *footer*
- **-bottom** *text*
- **-link** *extdocURL*
- **-linkoffline** *extdocURL* *packagelistLoc*
- **-group** *groupheading* *packagepattern*:*packagepattern*:...
- **-nodeprecated**
- **-nodeprecatedlist**
- **-notree**
- **-nohelp**
- **-nonavbar**
- **-helpfile** *path\filename*
- **-stylesheetfile** *path\filename*
- **-serialwarn**
- **-charset** *name*
- **-docencoding** *name*

GENERATED FILES

- standard doclet generates HTML-formatted documentation
- generates files with two types of names:
  - those named after classes/interfaces
  - those that are not (such as package-summary.html)

  Files in the latter group contain hyphens to prevent filename conflicts with those in the former group.

Basic Content Pages

- **class or interface page** (*classname*.html) for each class or interface
- **package page** (package-summary.html) for each package
- **overview page** (overview-summary.html) for the entire set of packages.
  - created only if you pass into javadoc two or more package names

Cross-Reference Pages

- **class hierarchy page for the entire set of packages** (overview-

tree.html)
    - To view this, click on "Overview" in the navigation bar, then click on "Tree".
- **class hierarchy page for each package** (package-tree.html)
    - To view this, go to a particular package, class or interface page; click "Tree" to display the hierarchy for that package
- **"use" page** for each package (package-use.html) and a separate one for each class and interface (class-use/*classname*.html)
    - describes what packages, classes, methods, constructors and fields use any part of the given class, interface or package.
    - Given a class or interface A, its "use" page includes
        - subclasses of A,
        - fields declared as A,
        - methods that return A, and
        - methods and constructors with parameters of type A
    - access this page by first going to the package, class or interface, then clicking on the "Use" link in the navigation bar.
- **deprecated API page** (deprecated-list.html)
    - listing all deprecated names
    - deprecated name is not recommended for use, generally due to improvements, and a replacement name is usually given. Deprecated APIs may be removed in future implementations.
- **serialized form page** (serialized-form.html)
    - for information about serializable and externalizable classes.
    - get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description
- **index** (index-*.html) of all class, interface, constructor, field and method names, alphabetically arranged

Support Files

- **help page** (help-doc.html)
    - describes the navigation bar and the above pages
    - can provide your own custom help file to override the default using -helpfile
- **index.html file**
    - creates the HTML frames for display
    - This is the file you load to display the front page with frames.
    - This file itself contains no text content
- **frame files** (*-frame.html)
    - containing lists of packages, classes and interfaces,
    - used when HTML frames are being displayed.
- **package list** file (package-list)
    - a text file, not HTML
    - not reachable through any links.
- **style sheet** file (stylesheet.css)
- **doc-files** directory that holds any image, example, source code or other files that you want copied to the destination directory
    - not processed by Javadoc in any manner
    - not generated unless it exists in the source tree.

HTML Frames

- pass source files (*.java) or a single package name as arguments into the javadoc command → create only one frame in the left-hand column
- pass into javadoc two or more package names → creates a third frame listing all packages, as well as an overview page (Detail)

http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javadoc.html