What is Javadoc?

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.

Following is a simple example where the lines inside /\*….\*/ are Java multi-line comments. Similarly, the line which preceeds // is Java single-line comment.

Example

/\*\*

\* The HelloWorld program implements an application that

\* simply displays "Hello World!" to the standard output.

\*

\* @author Zara Ali

\* @version 1.0

\* @since 2014-03-31

\*/

public class HelloWorld {

public static void main(String[] args) {

// Prints Hello, World! on standard output.

System.out.println("Hello World!");

}

}

You can include required HTML tags inside the description part. For instance, the following example makes use of <h1>....</h1> for heading and <p> has been used for creating paragraph break –

Example

/\*\*

\* <h1>Hello, World!</h1>

\* The HelloWorld program implements an application that

\* simply displays "Hello World!" to the standard output.

\* <p>

\* Giving proper comments in your program makes it more

\* user friendly and it is assumed as a high quality code.

\*

\*

\* @author Zara Ali

\* @version 1.0

\* @since 2014-03-31

\*/

public class HelloWorld {

public static void main(String[] args) {

/\* Prints Hello, World! on standard output.

System.out.println("Hello World!");

}

}

The javadoc Tags

The javadoc tool recognizes the following tags −

|  |  |  |
| --- | --- | --- |
| **Tag** | **Description** | **Syntax** |
| @author | Adds the author of a class. | @author name-text |
| {@code} | Displays text in code font without interpreting the text as HTML markup or nested javadoc tags. | {@code text} |
| {@docRoot} | Represents the relative path to the generated document's root directory from any generated page. | {@docRoot} |
| @deprecated | Adds a comment indicating that this API should no longer be used. | @deprecated deprecatedtext |
| @exception | Adds a **Throws** subheading to the generated documentation, with the classname and description text. | @exception class-name description |
| {@inheritDoc} | Inherits a comment from the **nearest** inheritable class or implementable interface. | Inherits a comment from the immediate surperclass. |
| {@link} | Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class. | {@link package.class#member label} |
| {@linkplain} | Identical to {@link}, except the link's label is displayed in plain text than code font. | {@linkplain package.class#member label} |
| @param | Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section. | @param parameter-name description |
| @return | Adds a "Returns" section with the description text. | @return description |
| @see | Adds a "See Also" heading with a link or text entry that points to reference. | @see reference |
| @serial | Used in the doc comment for a default serializable field. | @serial field-description | include | exclude |
| @serialData | Documents the data written by the writeObject( ) or writeExternal( ) methods. | @serialData data-description |
| @serialField | Documents an ObjectStreamField component. | @serialField field-name field-type field-description |
| @since | Adds a "Since" heading with the specified since-text to the generated documentation. | @since release |
| @throws | The @throws and @exception tags are synonyms. | @throws class-name description |
| {@value} | When {@value} is used in the doc comment of a static field, it displays the value of that constant. | {@value package.class#field} |
| @version | Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used. | @version version-text |

Example

Following program uses few of the important tags available for documentation comments. You can make use of other tags based on your requirements.

The documentation about the AddNum class will be produced in HTML file AddNum.html but at the same time a master file with a name index.html will also be created.

import java.io.\*;

/\*\*

\* <h1>Add Two Numbers!</h1>

\* The AddNum program implements an application that

\* simply adds two given integer numbers and Prints

\* the output on the screen.

\* <p>

\* <b>Note:</b> Giving proper comments in your program makes it more

\* user friendly and it is assumed as a high quality code.

\*

\* @author Zara Ali

\* @version 1.0

\* @since 2014-03-31

\*/

public class AddNum {

/\*\*

\* This method is used to add two integers. This is

\* a the simplest form of a class method, just to

\* show the usage of various javadoc Tags.

\* @param numA This is the first paramter to addNum method

\* @param numB This is the second parameter to addNum method

\* @return int This returns sum of numA and numB.

\*/

public int addNum(int numA, int numB) {

return numA + numB;

}

/\*\*

\* This is the main method which makes use of addNum method.

\* @param args Unused.

\* @return Nothing.

\* @exception IOException On input error.

\* @see IOException

\*/

public static void main(String args[]) throws IOException {

AddNum obj = new AddNum();

int sum = obj.addNum(10, 20);

System.out.println("Sum of 10 and 20 is :" + sum);

}

}