

Chp-35

Java Documentation

The Java Language supports three types of comments:

Comment Description

`/* text */` The compiler ignores everything from `/*` to `*/`.

`// text` The compiler ignores everything from `//` to the end of the line.

`/**`

documentation `*/`

This is a documentation comment and in general its called **doc comment**. The **JDK**

javadoc tool uses *doc comments* when preparing automatically generated documentation.

This tutorial is all about explaining Javadoc. We will see how we can make use of Javadoc for generating useful documentation for our Java code.

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 has required documentation in a predefined format.

Following is a simple example where red part of the code represents Java comments:

```
/**
 * 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 example, below example makes use of `<h1>....</h1>` for heading and `<p>` has been used for creating paragraph break:

CHAPTER

TUTORIALS POINT

Simply

Easy Learning

```
/**
```

```
 * <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** deprecated-text
@exception Adds a **Throws** subheading to the generated documentation, with the class-name 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 superclass.
{@link}
Inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class. T
{@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 |

TUTORIALS POINT Simply Easy Learning

include | exclude
@serialData Documents the data written by the writeObject() or writeExternal() methods **@serialData** data-description
@serialField Documents an ObjectOutputStream component. **@serialField** field-name fieldType 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 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.
     */
    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);
    }
}
```

TUTORIALS POINT Simply Easy Learning

```
* @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);
}
}
```

Now, process above AddNum.java file using javadoc utility as follows:

```
$ javadoc AddNum.java
Loading source file AddNum.java...
Constructing Javadoc information...
Standard Doclet version 1.7.0_51
Building tree for all the packages and classes...
Generating /AddNum.html...
AddNum.java:36: warning - @return tag cannot be used in method with void return
type.
Generating /package-frame.html...
Generating /package-summary.html...
Generating /package-tree.html...
Generating /constant-values.html...
Building index for all the packages and classes...
Generating /overview-tree.html...
```

```
Generating /index-all.html...
Generating /deprecated-list.html...
Building index for all classes...
Generating /allclasses-frame.html...
Generating /allclasses-noframe.html...
Generating /index.html...
Generating /help-doc.html...
```

```
1 warning
```

```
$
```

You can check all the generated documentation here: [AddNum](#). If you are using JDK 1.7 then javadoc does not generate a great **stylesheet.css**, so I suggest to download and use standard stylesheet from <http://docs.oracle.com/javase/7/docs/api/stylesheet.css>

TUTORIALS POINT Simply Easy Learning

Chp-36

Java Library Classes

This tutorial would cover package **java.lang**, which provides classes that are fundamental to the design of the Java programming language. The most important classes are Object, which is the root of the class hierarchy, and Class, instances of which represent classes at run time. Here is the list of classes of package **java.lang**. These classes are very important to know for a Java programmer. Click a class link to know more detail about that class. For a further drill, you can refer standard Java documentation.

SN Methods with Description

1 Boolean

Boolean

2 Byte

The Byte class wraps a value of primitive type byte in an object.

3 Character

The Character class wraps a value of the primitive type char in an object.

4 Class

Instances of the class Class represent classes and interfaces in a running Java application.

5 ClassLoader

A class loader is an object that is responsible for loading classes.

6 Compiler

The Compiler class is provided to support Java-to-native-code compilers and related services.

7 Double

The Double class wraps a value of the primitive type double in an object.

8 Float

The Float class wraps a value of primitive type float in an object.

9 Integer

The Integer class wraps a value of the primitive type int in an object.

10 Long

The Long class wraps a value of the primitive type long in an object.

11

Math

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

CHAPTER

TUTORIALS POINT

Simply

Easy Learning

12

Number

The abstract class Number is the superclass of classes BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, and Short.

13 Object

Class Object is the root of the class hierarchy.

14 Package

Package objects contain version information about the implementation and specification of a Java package.

15

Process

The Runtime.exec methods create a native process and return an instance of a subclass of Process that can be used to control the process and obtain information about it.

16

Runtime

Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running.

17 RuntimePermission

This class is for runtime permissions.

18 SecurityManager

The security manager is a class that allows applications to implement a security policy.

19 Short

The Short class wraps a value of primitive type short in an object.

20 StackTraceElement

An element in a stack trace, as returned by Throwable.getStackTrace().

21

StrictMath

The class StrictMath contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

22 String

The String class represents character strings.

23 StringBuffer

A string buffer implements a mutable sequence of characters.

24 System

The System class contains several useful class fields and methods.

25 Thread

A thread is a thread of execution in a program.

26 ThreadGroup

A thread group represents a set of threads.

27 ThreadLocal

This class provides thread-local variables.

28 Throwable

The Throwable class is the superclass of all errors and exceptions in the Java language.

29

Void

The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the Java keyword void.