



Introducción

Documentar el código de un programa es añadir suficiente información como para explicar lo que hace, punto por punto, de forma que no sólo los ordenadores sepan qué hacer, sino que además los humanos entiendan qué están haciendo y por qué.

Porque entre lo que tiene que hacer un programa y cómo lo hace hay una distancia impresionante: todas las horas que el programador ha dedicado a pergeñar una solución y escribirla en el lenguaje que corresponda para que el ordenador la ejecute ciegamente.

Documentar un programa no es sólo un acto de buen hacer del programador por aquello de dejar la obra rematada. Es además una necesidad que sólo se aprecia en su debida magnitud cuando hay errores que reparar o hay que extender el programa con nuevas capacidades o adaptarlo a un nuevo escenario. Hay dos reglas que no se deben olvidar nunca:

1. Todos los programas tienen errores y descubrirlos sólo es cuestión de tiempo y de que el programa tenga éxito y se utilice frecuentemente.
2. Todos los programas sufren modificaciones a lo largo de su vida, al menos todos aquellos que tienen éxito.

Por una u otra razón, todo programa que tenga éxito será modificado en el futuro, bien por el programador original, bien por otro programador que le sustituya. Pensando en esta revisión de código es por lo que es importante que el programa se entienda: para poder repararlo y modificarlo.

¿Qué hay que documentar?

Hay que añadir explicaciones a todo lo que no es evidente.

No hay que repetir lo que se hace, sino explicar por qué se hace.

Y eso se traduce en:

- ¿de qué se encarga una clase? ¿un paquete?
- ¿qué hace un método?
- ¿cuál es el uso esperado de un método?
- ¿para qué se usa una variable?
- ¿cuál es el uso esperado de una variable?
- ¿qué algoritmo estamos usando? ¿de dónde lo hemos sacado?
- ¿qué limitaciones tiene el algoritmo? ¿... la implementación?
- ¿qué se debería mejorar ... si hubiera tiempo?

Tipos de comentarios

En Java disponemos de tres notaciones para introducir comentarios:

javadoc

Comienzan con los caracteres `/**`, se pueden prolongar a lo largo de varias líneas (que probablemente comiencen con el carácter `/*`) y terminan con los caracteres `*/`.

una línea

Comienzan con los caracteres `///` y terminan con la línea.

tipo C

Comienzan con los caracteres `/*`, se pueden prolongar a lo largo de varias líneas (que probablemente comiencen con el carácter `/*`) y terminan con los caracteres `*/`.

Cada tipo de comentario se debe adaptar a un propósito:

javadoc

Para generar documentación externa (ver comentarios javadoc más abajo)

una línea

Para documentar código que no necesitamos que aparezca en la documentación externa (que genere javadoc)

Este tipo de comentarios se usará incluso cuando el comentario ocupe varias líneas, cada una de las cuales comenzará con `///`

tipo C

Para eliminar código. Ocurre a menudo que código obsoleto no queremos que desaparezca, sino mantenerlo "por si acaso". Para que no se ejecute, se comenta.
(En inglés se suele denominar "comment out")

Javadoc, que veremos posteriormente, impone sus propias reglas prácticas.

¿Cuándo hay que poner un comentario?

Por obligación (**javadoc**):

1. al principio de cada clase
2. al principio de cada método
3. ante cada variable de clase

Por conveniencia (**una línea**):

4. al principio de fragmento de código no evidente
5. a lo largo de los bucles

Y por si acaso (**una línea**):

6. siempre que hagamos algo raro
7. siempre que el código no sea evidente

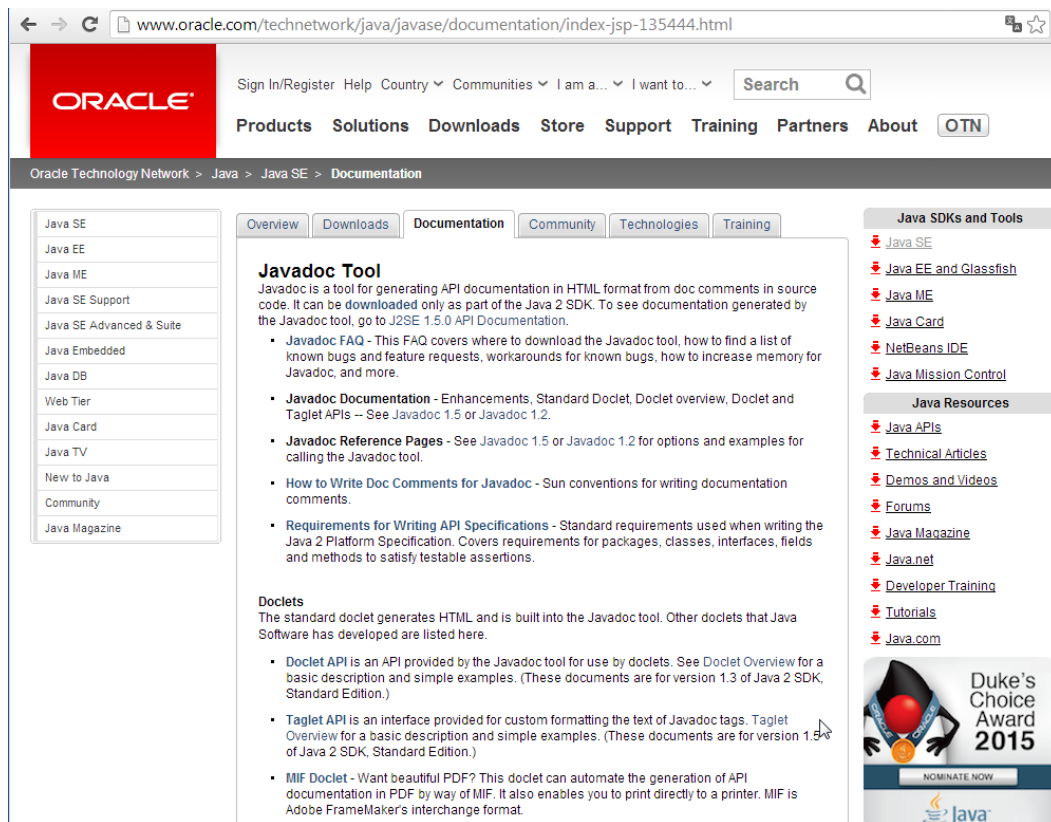
Es decir, que los comentarios vale más que sobren que falten.

Y una nota de cautela, cuando un programa se modifica, los comentarios deben modificarse al tiempo, no sea que los comentarios acaben refiriéndose a un algoritmo que ya no utilizamos.

¿Qué es Javadoc?

Javadoc es una utilidad que acompaña a todas las versiones del SDK jdk*, cuyo fin es fomentar una correcta documentación del código en Java. Esta utilidad es un programa que, si un conjunto de clases escritas en Java están adecuadamente comentadas, genera automáticamente un conjunto de páginas HTML similares a las que se usan como documentación de las clases estándar de Java. Constituye por tanto un formato de documentación estándar y familiar para cualquier programador en Java y fomenta asimismo la reutilización y compartición de código en Java.

Para más información vea la página oficial de javadoc:



The screenshot shows the Oracle Java SE Documentation page. The header includes the Oracle logo, navigation links (Sign In/Register, Help, Country, Communities, I am a..., I want to...), a search bar, and a menu (Products, Solutions, Downloads, Store, Support, Training, Partners, About, OTN). The breadcrumb trail is Oracle Technology Network > Java > Java SE > Documentation. The left sidebar lists various Java SE components. The main content area is titled 'Javadoc Tool' and provides an overview of the tool, its purpose, and links to various resources. The right sidebar lists 'Java SDKs and Tools' and 'Java Resources'.

Javadoc Tool

Javadoc is a tool for generating API documentation in HTML format from doc comments in source code. It can be downloaded only as part of the Java 2 SDK. To see documentation generated by the Javadoc tool, go to J2SE 1.5.0 API Documentation.

- **Javadoc FAQ** - This FAQ covers where to download the Javadoc tool, how to find a list of known bugs and feature requests, workarounds for known bugs, how to increase memory for Javadoc, and more.
- **Javadoc Documentation** - Enhancements, Standard Doclet, Doclet overview, Doclet and Taglet APIs -- See Javadoc 1.5 or Javadoc 1.2.
- **Javadoc Reference Pages** - See Javadoc 1.5 or Javadoc 1.2 for options and examples for calling the Javadoc tool.
- **How to Write Doc Comments for Javadoc** - Sun conventions for writing documentation comments.
- **Requirements for Writing API Specifications** - Standard requirements used when writing the Java 2 Platform Specification. Covers requirements for packages, classes, interfaces, fields and methods to satisfy testable assertions.

Doclets

The standard doclet generates HTML and is built into the Javadoc tool. Other doclets that Java Software has developed are listed here.

- **Doclet API** is an API provided by the Javadoc tool for use by doclets. See Doclet Overview for a basic description and simple examples. (These documents are for version 1.3 of Java 2 SDK, Standard Edition.)
- **Taglet API** is an interface provided for custom formatting the text of Javadoc tags. Taglet Overview for a basic description and simple examples. (These documents are for version 1.5 of Java 2 SDK, Standard Edition.)
- **MIF Doclet** - Want beautiful PDF? This doclet can automate the generation of API documentation in PDF by way of MIF. It also enables you to print directly to a printer. MIF is Adobe Framemaker's interchange format.

Java SDKs and Tools

- Java SE
- Java EE and Glassfish
- Java ME
- Java Card
- NetBeans IDE
- Java Mission Control

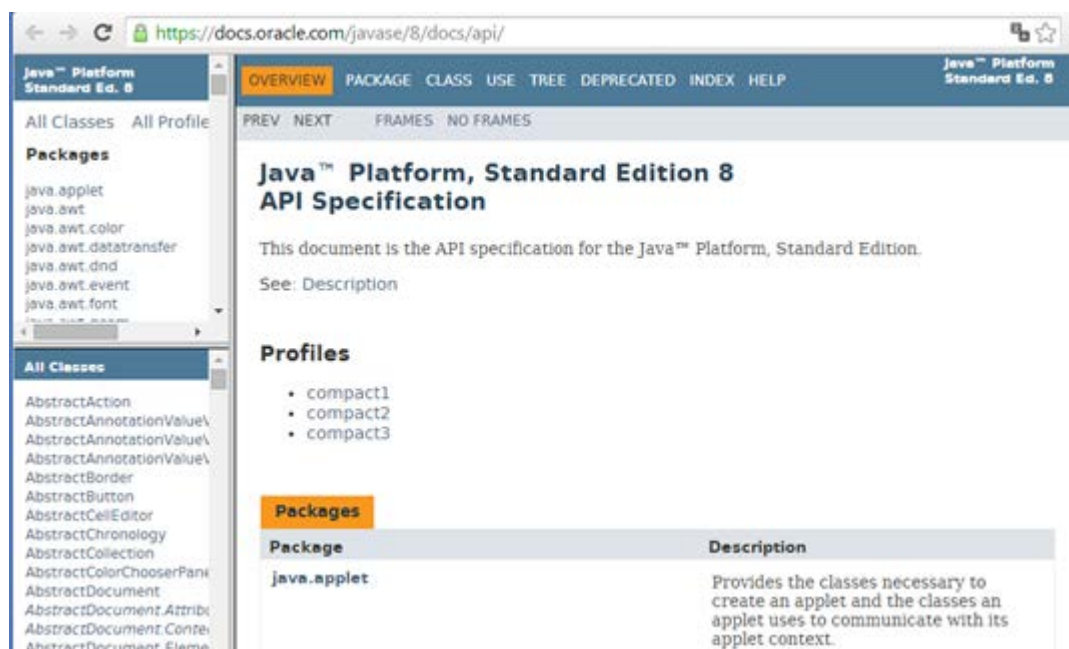
Java Resources

- Java APIs
- Technical Articles
- Demos and Videos
- Forums
- Java Magazine
- Java.net
- Developer Training
- Tutorials
- Java.com

Duke's Choice Award 2015

NOMINATE NOW

Java™ Platform, Standard Edition 8 API Specification. Generada con Javadoc



The screenshot shows the Java Platform, Standard Edition 8 API Specification page. The header includes the URL https://docs.oracle.com/javase/8/docs/api/. The left sidebar lists 'All Classes' and 'All Profile'. The main content area is titled 'Java™ Platform, Standard Edition 8 API Specification' and provides an overview of the specification. The right sidebar lists 'All Classes' and 'All Profile'.

Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.

Aunque javadoc no ayuda a la comprensión de los detalles de código, si ayuda a la comprensión de la arquitectura de la solución, lo que no es poco. Se dice que javadoc se centra en la interfaz (*API - Application Programming Interface*) de las clases y paquetes Java.

Documentación

En la documentación general de la implementación y especificación de un proyecto se deben contemplar dos aspectos:

- La documentación pública.
- La documentación privada.

La documentación pública se hace indispensable cuando ofrecemos un cierto conjunto de clases en algún dominio específico (un paquete). Al usuario de este paquete (quien a su vez es programador) le interesará saber qué clases están disponibles, qué hacen exactamente y bajo qué condiciones, es decir, la especificación completa de la interface correspondiente. Podemos así compartir la especificación pero con ocultamiento de información.

La documentación privada, por su parte, contempla no el uso, sino la misma implementación. En este caso se trata de describir cómo se hacen las cosas, qué estrategias se utilizan y, en general, a incluir toda anotación que ayude a explicar y clarificar una solución implementada.

Mantener presente estas dos perspectivas se constituye en una gran guía para el logro de una buena modularidad en el desarrollo de soluciones, tanto cuando se trata de paquetes, como en aplicaciones completas.

Documentación pública

La documentación pública –que se suele referenciar en inglés como API, Application Programming Interface– se realiza utilizando la herramienta javadoc (incluido en el kit de desarrollo para Java, JDK). Este programa produce páginas HTML permitiendo examinar la documentación de manera ágil mediante un navegador o visualizador. La información correspondiente se hace a través de comentarios reconocibles por javadoc dentro del código fuente.

Javadoc realza algunos comentarios, de los que exige una sintaxis especial. Deben comenzar por "/*" y terminar por "*/", incluyendo una descripción y algunas etiquetas especiales:

```
/**
 * Parte descriptiva.
 * Que puede consistir de varias frases o párrafos.
 *
 * @etiqueta texto específico de la etiqueta.
 */
```

Estos comentarios especiales deben aparecer justo antes de la declaración de una clase, una propiedad o un método en el mismo código fuente. En las siguientes secciones se detallan las etiquetas (tags) que javadoc sabe interpretar en cada uno de los casos.

Como regla general, hay que destacar que la primera frase (el texto hasta el primer punto) recibirá un tratamiento destacado, por lo que debe aportar una explicación concisa y contundente del elemento documentado. Las demás frases entrarán en detalles.

Documentación privada

En general, la documentación privada debe seguir las mismas pautas que para la documentación pública. Aunque los comentarios especiales serán ignorados por javadoc –ya que los elementos documentados no son públicos– se procura de esta manera un estilo unificado que favorece la legibilidad y el mantenimiento (un elemento no público podría dejar de serlo en algún momento). Se debe hacer buen uso de comentarios concisos que ayuden a clarificar secciones de código que de por sí no sean autoexplicativos. Los comentarios escritos en javadoc admiten dos características avanzadas:

1. La posibilidad de insertar etiquetas propias de HTML, como `` ó `<i></i>`.
2. La posibilidad de incluir etiquetas que documenten características específicas del elemento comentado. Por ejemplo, una clase puede tener un autor, una versión, etc., y un método unos parámetros.

Las características específicas se documentan con etiquetas de la forma “@etiqueta”, en formato de una por línea.

Algunas de las características que reconoce javadoc son:

Etiqueta	Sintaxis	Aplicable a	Efecto
@see	@see clase @see clase#método	Clase, método	Crea, en la página generada para la clase, un hipere enlace a la página de la clase o al método dentro de ella.
@version	@version texto	Clase, interface	Muestra el texto en el campo versión de la página generada para la clase o interface.
@author	@author texto	Clase, interface	Muestra el texto en el campo autor de la página generada para la clase o interface.
@param	@param variable texto	Método	Agrega una entrada a la sección “Parameters” de la sección del método en la página generada para la clase.
@return	@return texto	Método	Agrega una sección “Returns” de la sección del método en la página generada para la clase.
@throws	@throws clase texto	Método	Agrega una sección “Throws” de la sección del método en la página generada para la clase.
@deprecated	@deprecated texto	Clase, método, propiedad	Decirle a los programadores que la clase, método o propiedad está obsoleta y cuál usar en su lugar

Documentación de clases e interfaces

Deben usarse al menos las etiquetas:

- @author
- @version

La tabla muestra todas las etiquetas posibles y su interpretación:

@autor	Nombre del autor
@version	Identificación de la versión y fecha
@see	Referencia a otras clases y métodos

Documentación de constructores y métodos

Deben usarse al menos las etiquetas:

- @param
una por argumento de entrada
- @return
si el método no es void
- @exception ó @throws
una por tipo de Exception que se puede lanzar
(@exception y @throws se pueden usar indistintamente).

La tabla muestra todas las etiquetas posibles y su interpretación:

@param	Nombre del parámetro	Descripción de su significado y uso
@return		Descripción de lo que devuelve
@exception	Nombre de la excepción	Excepciones que pueden lanzarse
@throws	Nombre de la excepción	Excepciones que pueden lanzarse

@exception y @throws se pueden usar indistintamente.

Documentación de propiedades

Ninguna etiqueta es obligatoria.

Ejemplo

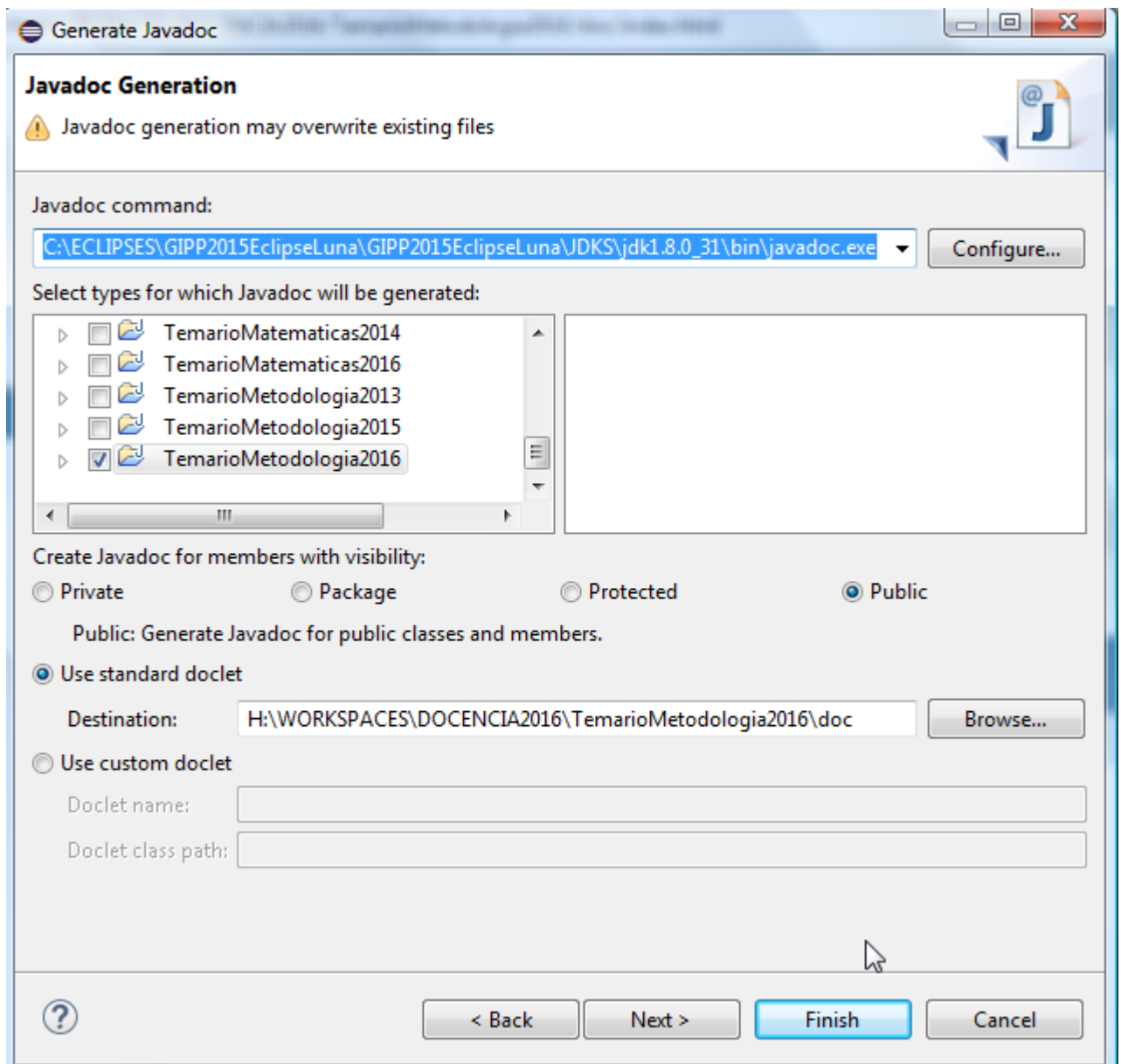
```
1  /**
2   * Ejemplo: Círculos.
3   *
4   * @author Mercedes MP2016
5   * @version 31.01.2016
6   */
7  public class Circulo {
8      private double centroX;
9      private double centroY;
10     private double radio;
11
12     /**
13      * Constructor.
14      *
15      * @param cx
16      *      centro: coordenada X.
17      * @param cy
18      *      centro: coordenada Y.
19      * @param r
20      *      radio.
21      */
22     public Circulo(double cx, double cy, double r) {
23         centroX = cx;
24         centroY = cy;
25         radio = r;
26     }
27
28     /**
29      * Getter.
30      *
31      * @return centro: coordenada X.
32      */
33     public double getCentroX() {
34         return centroX;
35     }
36
37     /**
38      * Calcula la longitud de la circunferencia (perímetro del círculo).
39      *
40      * @return circunferencia.
41      */
42
43
44
45
46     /**
47      * Desplaza el círculo a otro lugar.
48      *
49      * @param deltaX
50      *      movimiento en el eje X.
51      * @param deltaY
52      *      movimiento en el eje Y.
53      */
54     public void mueve(double deltaX, double deltaY) {
55         centroX = centroX + deltaX;
56         centroY = centroY + deltaY;
57     }
58
59     /**
60      * Escala el círculo (cambia su radio).
61      *
62      * @param s
63      *      factor de escala.
64      */
65     public void escala(double s) {
66         radio = radio * s;
67     }
68 }
```


Utilizar javadoc con Eclipse

1. Eclipse puede generar comentarios javadoc para clases y métodos.
 - 1.1 Situar el cursor en el texto de la declaración de clase o método.
 - 1.2 Clic Derecho → Source → Generate Element Comment.
 - 1.3 Javadoc comenta generando los tags apropiados pero deberemos escribir las descripciones.
2. Eclipse puede también compilar Javadocs para los proyectos/paquetes/clases en el espacio de trabajo.

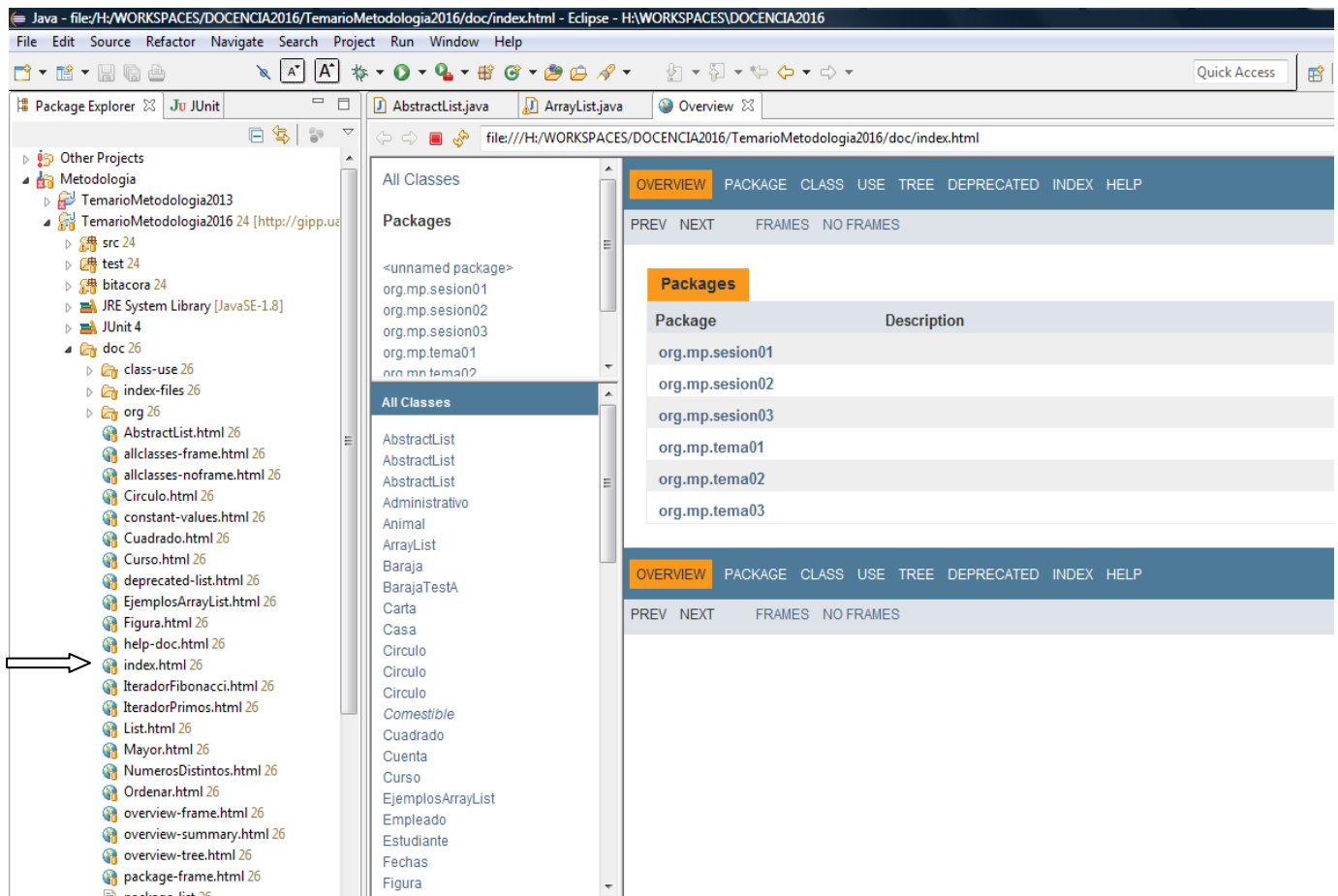
Poner la localización del archivo Javadoc y exportar el proyecto/paquete/clase como javadoc.

 - 2.1 Seleccionar el proyecto/paquete/clase del que queramos generar la documentación.
 - 2.2 File → Export
 - 2.3 Si fuese necesario, seleccionar Javadoc e introducir el path de **javadoc.exe**, por ejemplo:
C:\ECLIPSES\GIPP2015ECLIPSELUNA\GIPP2015ECLIPSELUNA\JDK\jdk1.8.0_31\bin
\javadoc.exe.



2.4 Seleccionar también el destino de la exportación y Next.

Se generará la documentación creando la carpeta doc. Se podrá consultar haciendo doble clic sobre index.html.

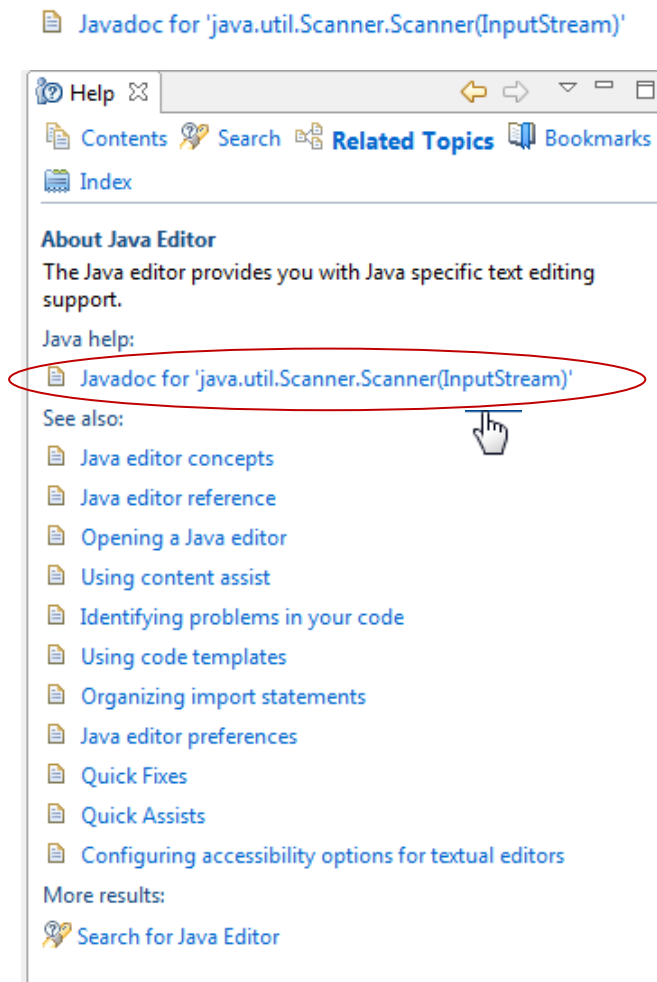


¿Cómo agregar la documentación de la API de Java en Eclipse?

Comprobaremos en primer lugar si la documentación es accesible. Para ello, sobre cualquier programa:

```
1 package org.mp.tema01;
2
3 import java.util.Scanner;
4
5
6 public class TestCirculoRectangulo {
7     private static Scanner entrada;
8
9     public static void main(String[] args) {
10
11         entrada = new Scanner(System.in);
12         int radio = entrada.nextInt();
13         Circulo circulo =
14             new Circulo(radio);
15         System.out.println("Circulo " + circulo.toString());
16         System.out.println("Su color es " + circulo.getColor());
17         System.out.println("El radio es " + circulo.getRadio());
18         System.out.println("El área es " + circulo.getArea());
19         System.out.println("El diámetro es " + circulo.getDiametro());
20
21         Rectangulo rectangulo =
22             new Rectangulo(2, 4);
23         System.out.println("\nRectángulo " + rectangulo.toString());
24         System.out.println("El área es " + rectangulo.getArea());
25         System.out.println("El perímetro es " +
26             rectangulo.getPerimetro());
27     }
28 }
```

Nos situamos sobre una clase de la API, por ejemplo Scanner y pulsamos F1. Aparecerá la ventana de ayuda que se muestra en la página siguiente y podemos seleccionar:



De esta manera, podremos acceder a la documentación y consultar cualquier duda sobre la clase.

Help

ContentsSearchRelated TopicsBookmarksIndex

OVERVIEWPACKAGECLASSUSE TREEDEPRECATEDINDEXHELP

PREV CLASSNEXT CLASSFRAMESNO FRAMESALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.util

Class Scanner

java.lang.Object
java.util.Scanner

All Implemented Interfaces:
Closeable, AutoCloseable, Iterator<String>

public final class **Scanner**
extends `Object`
implements `Iterator<String>`, `Closeable`

A simple text scanner which can parse primitive types and strings using regular expressions.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

For example, this code allows a user to read a number from `System.in`:

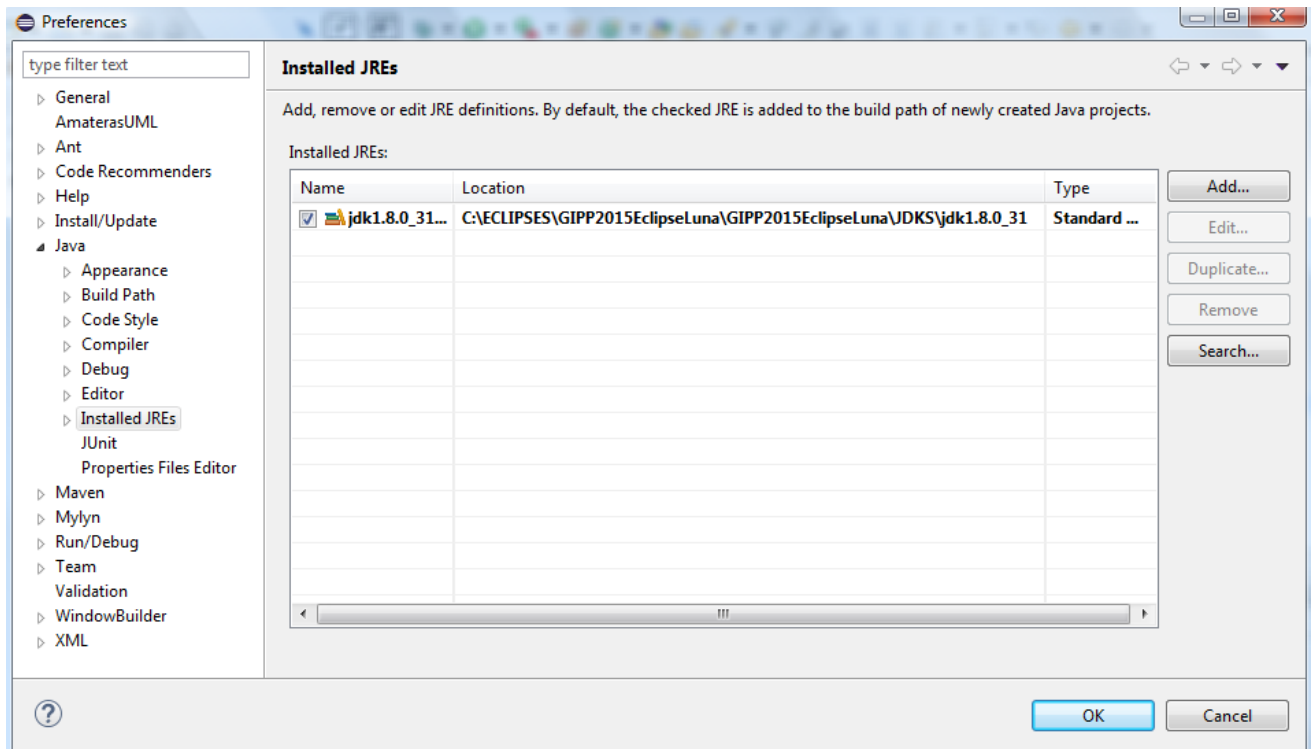
```
Scanner sc = new Scanner(System.in);  
int i = sc.nextInt();
```

Method Summary

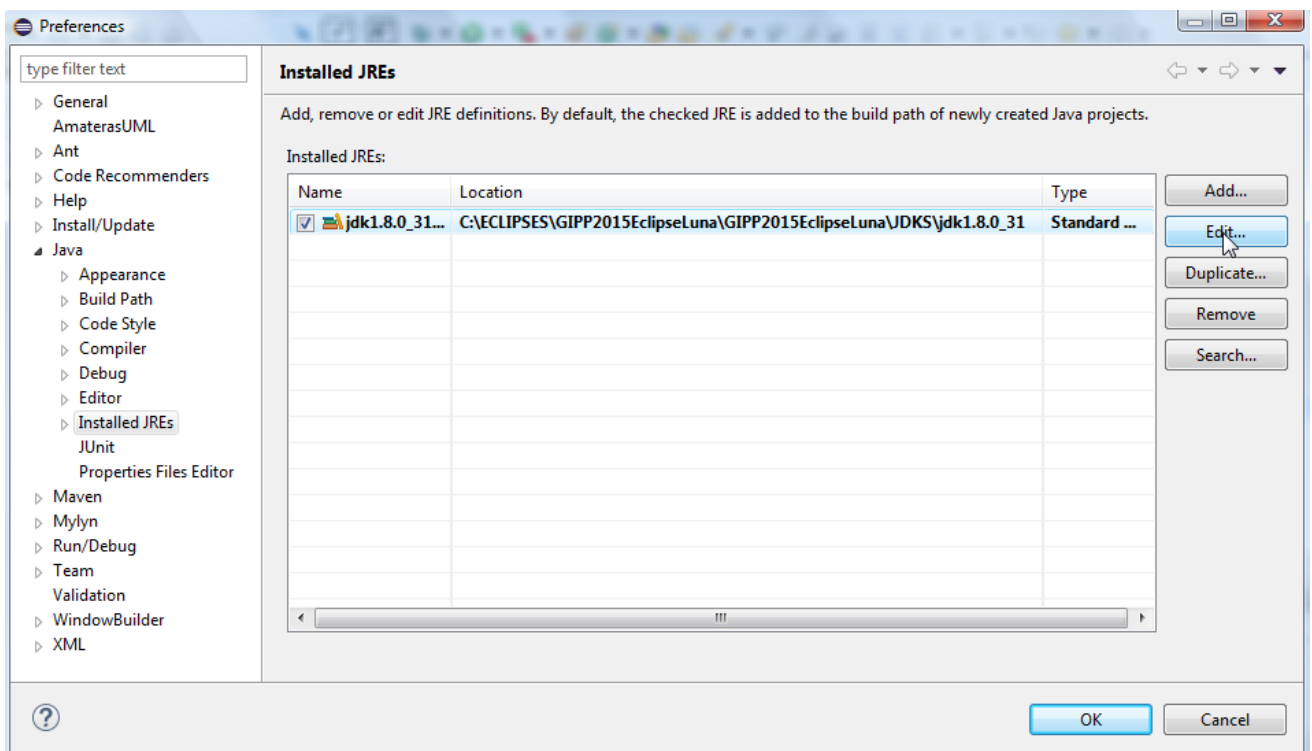
All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
void	close()	Closes this scanner.
Pattern	delimiter()	Returns the Pattern this Scanner is currently using to match delimiters.
String	findInLine(Pattern pattern)	Attempts to find the next occurrence of the specified pattern ignoring delimiters.
String	findInLine(String pattern)	Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
String	findWithinHorizon(Pattern pattern, int horizon)	Attempts to find the next occurrence of the specified pattern.
String	findWithinHorizon(String pattern, int horizon)	Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
boolean	hasNext()	Returns true if this scanner has another token in its input.
boolean	hasNext(Pattern pattern)	Returns true if the next complete token matches the specified pattern.

Si la documentación no estuviese accesible, tendríamos que:

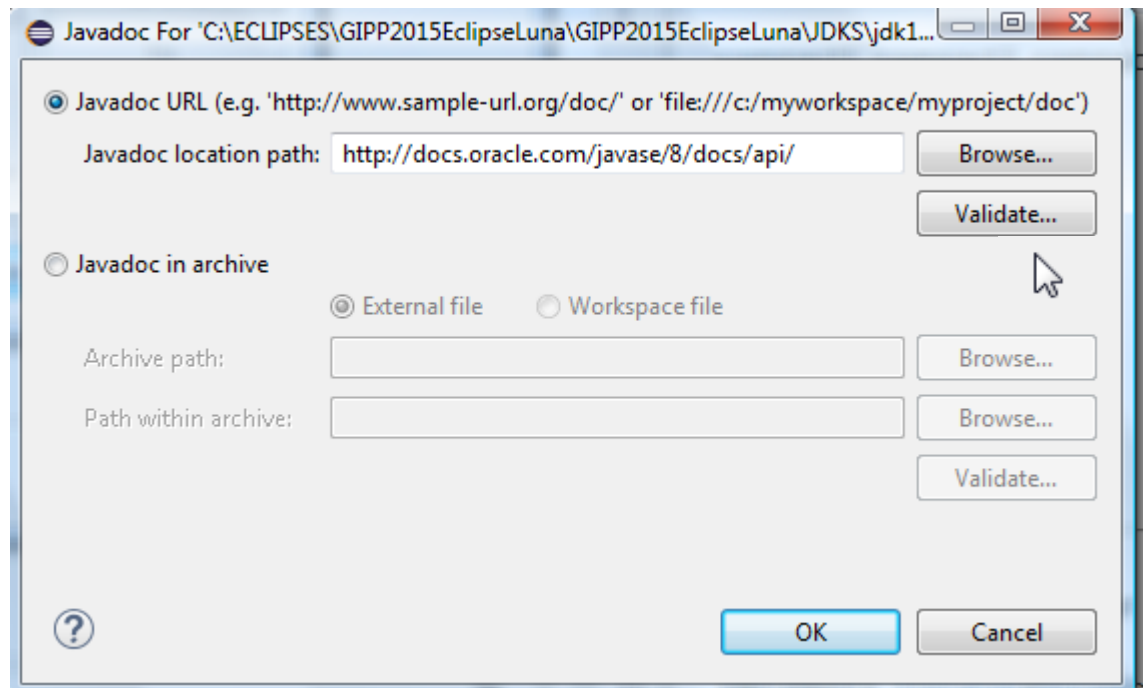
- Acceder a la página de Oracle de descargas de documentación, actualmente:
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Descargar el archivo en la carpeta de la distribución en uso de ECLIPSE.
- Abrir Eclipse y seleccionamos Windows → Preferences → Java → Installed JREs



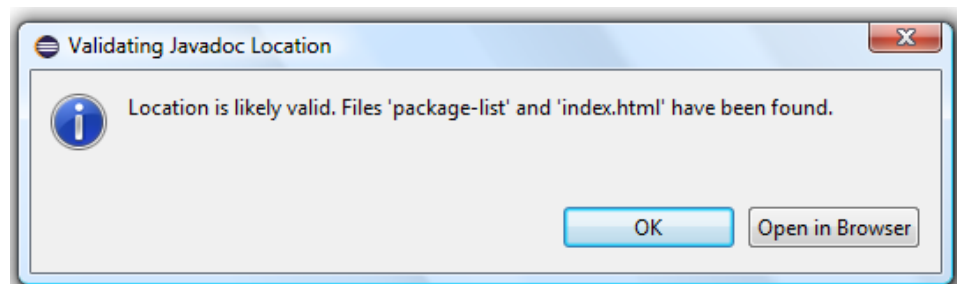
- Editar el JREs instalado



- Seleccionar Javadoc Location y la opción:



- Validar la localización.



Referencias

How to Write Doc Comments for the Javadoc Tool

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

<https://wiki.eclipse.org/Javadoc>