**JavaDoc**

Proyecto

Sistemas Informáticos

Tecnológico Espíritu Santo

Marcos Mora & Patricio Pihuave

Mayo 2017

**Dedicatoria**

Dedicado a Dios por darme las fuerzas para continuar en lo adverso, por guiarme con sabiduría en las situaciones difíciles. A mis padres por darme la vida, la oportunidad de estudiar y luchar día a día para que lograra escalar un peldaño más.

Y a mis amistades quienes me han demostrado su apoyo incondicional y sincero y a los maestros por sus conocimientos impartidos y guía.

**Agradecimientos**

A Dios por darme el aliento de vida y ayudarme a superar el miedo para poder continuar, a mis padres por darme el aliento a diario y a mis amigos por estar en las buenas y malas situaciones.

A mi maestra por su paciencia para conmigo, así como por sus conocimientos impartidos en clases ya que me ayudaron a poder pensar antes de cualquier situación a hacer una buena elección.

**Tabla de Contenidos**

[Capítulo 1 Introducción e información general 1](#_Capítulo_1_Java)

[Típos de comentarios en java 1](#_Tipos_de_Comentarios)

[Etiquetas de java 1](#_Etiquetas_de_Javadoc)

[Donde utilizar javadoc. 1](#_¿Dónde_utilizar_javadoc?)

[Ejemplos de internet. 1](#_Ejemplos_del_Internet)

[Conclusion 6](#_Conclusión)

[Referencias 8](#_Lista_de_referencias)

## **Capítulo 1** Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

**JavaDoc**

Javadoc es una utilidad de Oracle para la generación de documentación de APIs Application Programming Interface (Interfaz de programación de aplicaciones) en formato HTML a partir de código fuente Java. Javadoc es el estándar para documentar clases de Java. La mayoría de los IDEs Integrated Drive Electronics (entorno de desarrollo integrado) utilizan javadoc para generar de forma automática documentación de clases.

Javadoc es la herramienta de java para generar documentación básica para el programador a partir del código fuente. Se intenta evitar que la documentación que se genera mediante un editor de texto se quede rápidamente obsoleta cuando el programa continúa su desarrollo y no se tiene la disciplina/tiempo para mantener la documentación al día.

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.

Documentar un proyecto es algo fundamental de cara a su futuro mantenimiento. Cuando programamos una clase, debemos generar documentación lo suficientemente detallada sobre ella como para que otros programadores sean capaces de usarla sólo con su interfaz. No debe existir necesidad de leer o estudiar su implementación, lo mismo que nosotros para usar una clase del API (Application Programming Interface - Interfaz de programación de aplicaciones) Java no leemos ni estudiamos su código fuente.

## Tipos de Comentarios en Java

Java admite tres tipos de comentarios. Los dos primeros son el lineal que se representa con “ **//** ” y el multilinea “ **/ \* \* /** “.

El tercer tipo se llama Javadoc o comentario de documentacion, comienzan con los caracteres "**/\*\***", se pueden prolongar a lo largo de varias líneas (que probablemente comiencen con el carácter "**\***") y terminan con los caracteres "**\*/**". Los comentarios de la documentación le permiten integrar información sobre su programa en el propio programa. A continuación, puede utilizar el programa de utilidad javadoc (suministrado con el JDK) para extraer la información y ponerla en un archivo HTML.

## Etiquetas de Javadoc

La utilidad javadoc reconoce las siguientes etiquetas:

**@author**

La etiqueta @author documenta al autor de una clase o interfaz, tiene la siguiente sintaxis:

@author descripción

Aquí, la descripción ejecutando javadoc para que el campo @author se incluya en la documentación HTML.

**@Version**

La etiqueta de versión especifica la versión de una clase o interfaz. Tiene la siguiente sintaxis:

@Version info

Aquí, info es una cadena que contiene información de versión, normalmente una versión número, como 2.2. Deberá especificar la opción de versión al ejecutar javadoc para que el campo de versión se incluya en la documentación HTML.

**@param**

La etiqueta @param documenta un parámetro. Tiene la siguiente sintaxis:

@param parameter-name explanation

Aquí, parameter-name especifica el nombre de un parámetro. El significado de ese parámetro se describe mediante una explicación. La etiqueta @param se puede utilizar sólo en la documentación de un método o constructor, o una clase genérica o interfaz.

**@return**

La etiqueta de @return describe el valor devuelto de un método. Tiene la siguiente sintaxis:

@return explicacion

Aquí, la explicación describe el tipo y el significado del valor devuelto por un método. La etiqueta de @return (retorno) sólo se puede utilizar en la documentación de un método.

**@see**

La etiqueta @see Proporciona una referencia a información adicional. Dos formas de uso común se muestran aquí:

@see anchor

@see pkg.class#member text

En la primera forma ancla es un enlace a una URL absoluta o relativa. En la segunda forma ver pkg.Class#member texto especifica el nombre del elemento y el texto es el texto que se muestra para ese elemento. El parámetro de texto es opcional y, si no se utiliza, se muestra el elemento especificado por pkg.Class#member. El nombre de miembro también es opcional. Por lo tanto, puede especificar una referencia a un paquete, clase o interfaz además de una referencia a un método o campo específico. El nombre puede ser totalmente calificado o parcialmente calificado.

**@Thows**

La etiqueta @Throws (lanza) tiene el mismo significado que la etiqueta de excepción.

**@deprecated**

La etiqueta @deprecated especifica que los elementos de un programa están obsoletos. Se recomienda incluir etiquetas @see o {@link} para informar al programador sobre las alternativas disponibles. La sintaxis es la siguiente:

@deprecated description

Aquí, la descripción es el mensaje que describe la depreciación. La etiqueta @deprecated se puede utilizar en la documentación de campos, métodos, constructores, clases e interfaces.

**@exception**

La etiqueta @exception describe una excepción, describe una excepción a un método. Tiene la siguiente sintaxis:

@exception exception-name explanation

Aquí, el nombre completo de la excepción se especifica por el nombre de la excepción, y la explicación es una cadena que describe cómo puede producirse la excepción. La etiqueta @exception sólo se puede utilizar en la documentación de un método o constructor.

**@Since**

La etiqueta @Since establece que un elemento fue introducido en una versión específica. Tiene la siguiente sintaxis:

@Since release

Aquí, release es una cadena que designa el lanzamiento o versión en la que esta característica se hizo disponible.

**@Value**

La etiqueta @Value tiene dos formas, la primera muestra el valor de la constante que precede, que debe ser un campo estático.

**@serial**

La etiqueta @serial define el comentario para un campo serializable predeterminado. Tiene la siguiente sintaxis:

@serial descripcion

Aquí, la descripción es el comentario para ese campo.

@serialData (serie de datos)

La etiqueta @serialData documenta los datos escritos por los métodos Escribir Objeto () y escribir Métodos externos. (). Tiene la siguiente sintaxis:

@SerialData descripción

Aquí, la descripción es el comentario para esos datos.

**@code**

La etiqueta @code le permite incrustar texto, como un fragmento de código, en un comentario. Ese texto se muestra como-es en la fuente de código, sin ningún procesamiento posterior, como la representación HTML.

**{@docRoot}**

Especifica la ruta al directorio raíz de la documentación actual.

**{@inheritDoc}**

Esta etiqueta hereda un comentario de la superclase inmediata

## ¿Dónde utilizar javadoc?

1. Documentación de clases e interfaces

Deben usarse al menos las etiquetas:

@author: nombre del autor.

@version: identificación de la versión y fecha.

@see: referencia a otras clases y métodos

1. 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 o @throws:

Una por tipo de Exception que se puede lanzar (@exception y @throws se pueden usar indistintamente).

Las etiquetas @author y @version se usan para documentar clases e interfaces. Por tanto no son válidas en cabecera de constructores ni métodos. La etiqueta @param se usa para documentar constructores y métodos. La etiqueta @return se usa solo en métodos de tipo función.

Dentro de los comentarios se admiten etiquetas HTML, por ejemplo con @see se puede referenciar una página web como link para recomendar su visita de cara a ampliar información.

1. Para indicarle a Javadoc que queremos incluir documentación, debemos comenzar los comentarios de la siguiente manera:

/\*\*

\* Esto para Javadoc

\*/

1. Creamos una clase llamada "Clase\_java" y en el encabezado colocamos el autor de la clase, la instrucción @see con un enlace a una página web, también podemos agregar la versión de la clase:

/\*\*

\* @author Patricio

\* @see <a href="http://www.google.comt">http://www.google.com</a>

\* @version 1.2 17 de Mayo de 2017

\*/

public class Clase\_Java {

...

}

1. También podemos documentar las variables que hacen parte de nuestro código.

/\*\*

\* Variable privada: Nombre del autor

\*/

private final String Autor = "Patricio";

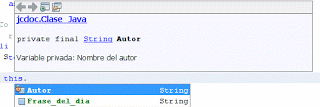
/\*\*

\* Variable publica: Una frase para reflexionar

\*/

public String Frase\_del\_dia = "Carpe diem";

Cuando se haga uso de estas variables comentadas, en el editor de Netbeans podremos ver algo como esto:



1. No debemos olvidar también comentar e constructor de clase.

/\*\*

\* Constructor de clase

\*/

public Clase\_Java(){

...

}

## Ejemplos del Internet

**Primero**

Cuando creamos un método es recomendable documentar los parámetros de entrada, si es que los tuviera, y si este método retorna algún resultado, utilizar la marca @return, si por el contrario el método es de la forma VOID, no se usa nada. Así también para las excepciones que puedan ocurrir se usa @exception. En la descripción del método, se puede incluir un ejemplo de uso encerrado en las etiquetas PRE, por ejemplo:

/\*\*

\* Método que convierte una cadena de la forma "yyyy-MM-dd HH:mm" a Date

\* <PRE> Clase\_Java cj = new Clase\_Java();

\* Date date = cj.StringToDate("2012-10-01 12:00")</PRE>

\* @param value String que debe tener la forma "yyyy-MM-dd HH:mm"

\* @return Date Un objeto Date con la fecha parseada

\* @exception ParseException Error de parseo, ocurre cuando no se puede convertir un String a Date

\* @since incluido desde la version 1.0

\*/

public Date StringToDate( String value )

{

Date date = new Date();

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm");

try {

date = (java.util.Date) formatter.parse( value );

} catch (ParseException ex) {

System.err.println( ex.getMessage() );

}

 return date;

}

Cuando utilicemos este método, podremos ver que Netbeans nos despliega toda la información en cuando se hace referencia al nombre StringToDate.

**Segundo**

Primer ejemplo: Utiliza comentarios de documentación A continuación se muestra un programa de ejemplo que utiliza la documentación comments.Notice la forma en que cada comentario precede inmediatamente el elemento que describe. Después de ser procesado por javadoc, la documentación sobre la clase SquareNum se encontrará en SquareNum.html.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

import java.io.\*;

/\*\*

\*Esta clase demuestra comentarios de documentación

\* @author Patricio

\* @version 1.2

\*/

public class SquareNum {

/\*\*

\* Este método devuelve el cuadrado de numero.

\* Esta es una descripción multilínea. Puedes usar

\* Tantas líneas como quieras.

\* @param num el valor a ser cuadrado.

\* @return num squared.

\*/

public double square(double num){

return num \* num;

}

/\*\*

\* Este método introduce un número del usuario.

\* @return la entrada de valor como un doble.

\* @exception IOException En error de entrada.

\* @see IOException

\*/

public double getNum() throws IOException {

// Crear un lector de Buffered usando el System.in

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader inData = new BufferedReader (isr);

String str;

str = inData.readLine();

return (new Double(str)).doubleValue();

}

/\*\*

\* Este método demuestra cuadrado ().

\* @param args no utilizado.

\* @exception IOException en error de entrada.

\* @see IOException

\*/

public static void main (String args[])

throws IOException

{

SquareNum ob = new SquareNum();

double val;

System.out.println("Introduzca el valor a cuadrar:");

val = ob.getNum ();

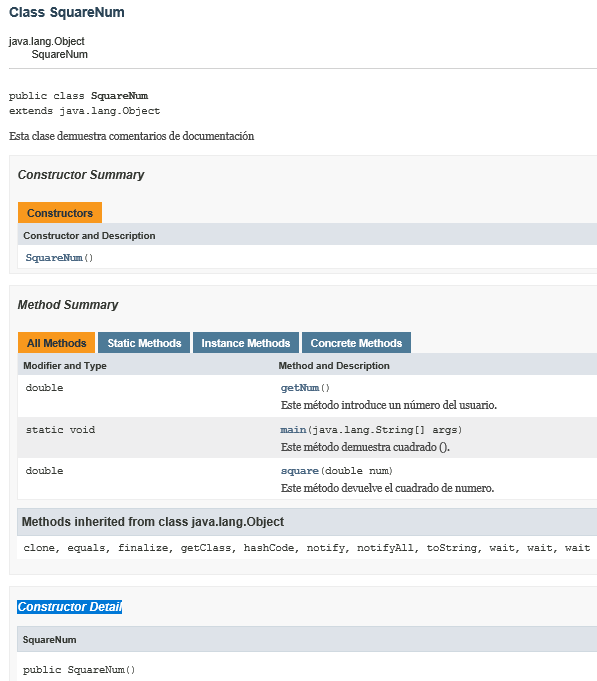
val = ob.square (val);

System.out.println("El valor cuadrado es" + val);

}

}

**Javadoc**



# 

**Ejemplos nuestros**

Primero Patricio

import java.util.ArrayList;

import java.util.Random;

/\*\*

\*Esta clase define objetos que contienen tantos enteros aleatorios entre 0 y 1000 como se le definen al crear un objeto.

\* @author Patricio

\* @version 1.0

\*

\*/

public final class SerieDeAleatoriosD

{

//Campos de la clase

private final ArrayList<Integer> serieAleatoria;

/\*\*

\* Constructor para la serie de números aleatorios

\* @param numeroItems El parámetro numeroItems define el número de elementos que va a tener la serie aleatoria

\*/

public SerieDeAleatoriosD (int numeroItems) {

serieAleatoria = new ArrayList<> ();

for (int i=0; i<numeroItems; i++) { serieAleatoria.add(0); }

System.out.println ("Serie inicializada. El número de elementos en la serie es: " + getNumeroItems() );

} //Cierre del constructor

/\*\*

\* Método que devuelve el número de ítems (números aleatorios) existentes en la serie

\* @return El número de ítems (números aleatorios) de que consta la serie

\*/

public int getNumeroItems() { return serieAleatoria.size(); }

/\*\*

\* Método que genera la serie de números aleatorios

\*/

public void generarSerieDeAleatorios () {

Random numAleatorio;

numAleatorio = new Random ();

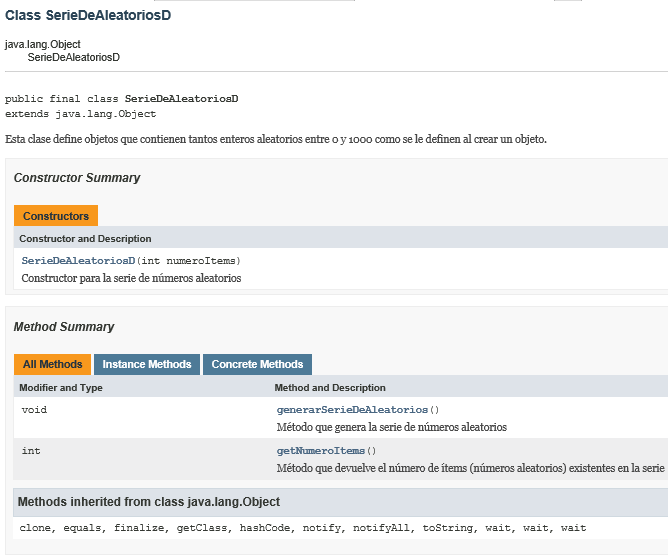
for (int i=0; i < serieAleatoria.size(); i++) { serieAleatoria.set(i, numAleatorio.nextInt(1000) ); }

System.out.print ("Serie generada! ");

} //Cierre del método

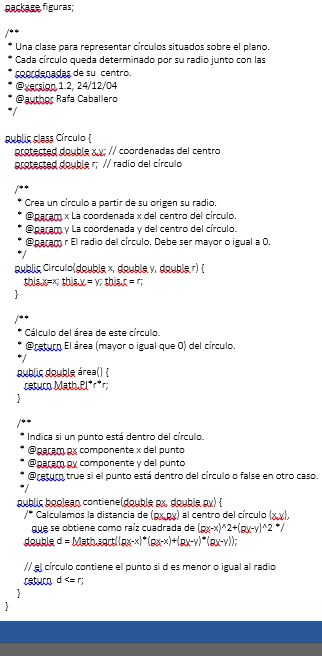
} //Cierre de la clase y del ejemplo

**JavaDoc**

****

****

**Ejercicio Marcos**



# Conclusión

Habiéndose dado argumentos que el Javadoc realmente es útil y necesario para documentar el código para nosotros mismo en futuras revisiones y/o modificaciones y como para los demás desarrolladores, es importante recalcar que todo desarrollador debería como regla general incluir en sus proyectos en java los comentarios de documentación.

# Lista de referencias

Java concepto tomado de A Beginner’s Guide Sixth Edition Create, Compile, and Run Java Programs Today, Autor Herbert Schildt. Part V Applying Java.

Ejemplos de internet, ejemplo 1 tomado de:

<http://jc-mouse.blogspot.com/2012/07/creacion-y-uso-de-javadoc-con-netbeans.html>

Ejemplo 2 tomado de A Beginner’s Guide Sixth Edition Create, Compile, and Run Java Programs Today, Autor Herbert Schildt. Part V Applying Java.

Ejemplos nuestros, ejemplo 1 Patricio Pihuave.