

Módulo profesional entornos de
desarrollo

UD 6–
Documentac
ión

contenido

- Documentar
- Tipos
- Formatos

documentación

Un sistema software que sea:

- difícil de comprender
- sólo utilizable por sus realizadores
- difícil de modificar

NO ES VÁLIDO

documentación

En todo proyecto de desarrollo de software es muy importante la documentación, no solo para el usuarios finales sino para los propios desarrolladores, tanto para uno mismo como para el resto del equipo de desarrolladores y los futuros miembros del mismo.

Ya veremos como ayuda la **refactorización** en el entendimiento del código, pero además es también comprensible con una buena **documentación**.

Tipos de documentación

Tipos de documentación

Documentación interna: va vinculada con el código

- Documentación de código (para los desarrolladores)

Documentación externa: localizada aparte del programa en el sentido que no forma parte del código.

- Documentación técnica (para los administradores)
- Documentación de usuario (para los usuarios)

Documentación de código

Un complemento muy útil para los desarrolladores es la documentación del código fuente que se está implementando.

Existen muchas formas diferentes de documentar con muchos niveles diferentes de profundización. La opción más sencilla es la primera que se aprende cuando se empieza a programar, que son los comentarios a lo largo del código fuente.

Documentar el código de un programa es dotar al software de toda la información que sea necesaria para explicar lo que hace. Los desarrolladores que llevan a cabo el software (y el resto del equipo de trabajo) deben entender qué está haciendo el código y el porqué.

Documentación de código

Hemos hablado antes de los comentarios y señalado que si un código necesita ser comentado es porque no es suficientemente descriptivo por si solo y necesita ser **refactorizado**. Aunque esto sea efectivamente así, el uso de comentarios sigue siendo una buena ayuda al entendimiento del código.

Además se utiliza de manera recurrente con el fin de omitir algunas instrucciones para que no se ejecuten, muy importante en un proceso de depuración.

Documentación de código

Los comentarios se encuentran intercalados con las sentencias de programación, de hecho se consideran parte del código fuente. Son pequeñas frases que explican pequeñas partes de las sentencias de programación implementadas.

Los comentarios no son tenidos en cuenta por parte de los compiladores a la hora de convertir el código fuente en código objeto y, posteriormente, en código ejecutable. Esto da libertad al programador para poder escribir cualquier cosa en ese espacio que él habrá indicado que es un comentario, sin tener que cumplir ninguna sintaxis específica.

Documentación de código

Existen diferentes formas de implementar comentarios, irá en función del lenguaje de programación que se utilice. A continuación se muestra la forma de implementar comentarios con Java.

En el lenguaje **Java** existen **tres formas** de poner comentarios.

- la forma de mostrar comentarios intercalados con el código fuente es añadiendo los caracteres `//`. Esto indicará que a partir de ese punto todo lo que se escriba hasta el final de la línea estará considerado en comentario y el compilador no lo tendrá en cuenta.

```
1. // Comentario de una línea
```

Documentación de código

- Otra forma de mostrar comentarios es añadiendo los caracteres `/*...*/`, Donde el comentario se encontraría ubicado en el lugar de los puntos suspensivos y, a diferencia de la anterior, se pueden escribir comentarios de más de una línea.

```
1. /* Comentario  
2. de varias  
3. líneas */
```

- El último caso son los comentarios para la herramienta de documentación **JavaDoc**. En este caso, antes del comentario pondremos una barra inclinada y dos asteriscos (`/**`) y finaliza con un asterisco y una barra

```
1. /** Comentario para JavaDoc */
```

¿Qué hay que documentar?

- Añadir explicaciones a todo lo que **NO** es evidente.
- ¿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?
- ¿qué se debería mejorar ... si hubiera tiempo?

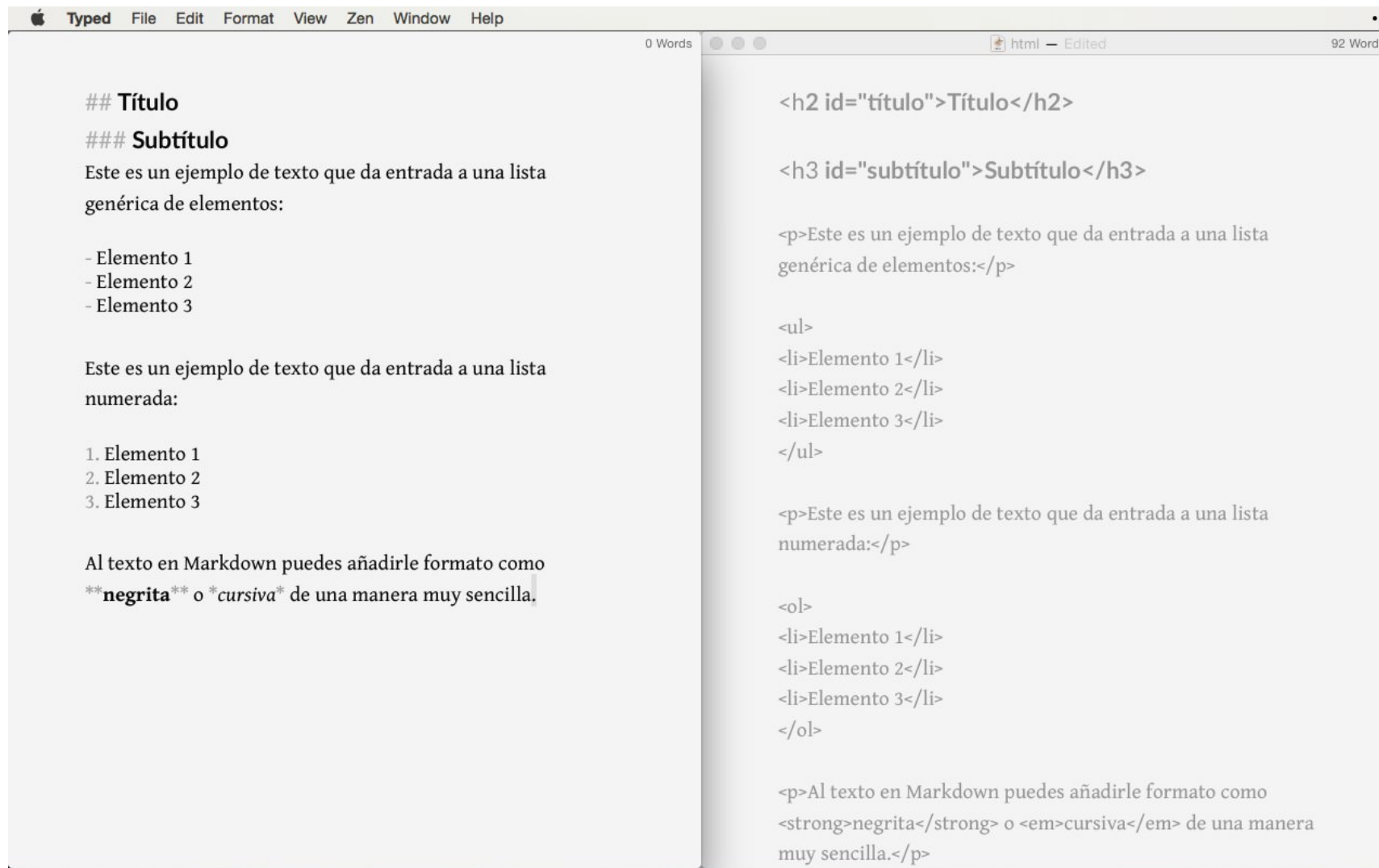
actividad

1. Comenta el código fuente de del examen de la primera evaluación.
Utiliza comentarios de una línea y de varias líneas

Formatos de documentación

Formatos de documentación

- **HTML** (p. ej. Javadoc)
- **Markdown** (p. ej. Gitbook) : es un lenguaje de marcado ligero. Una cantidad de sitios como GitHub, Reddit, Diaspora, Stack Exchange, OpenStreetMap, y SourceForge usan algunas variantes de Markdown para facilitar la discusión entre usuarios. Markdown convierte el texto marcado en documentos XHTML utilizando html2text.
- **reStructuredText** (p. ej. Readthedocs) lenguaje de marcas ligero creado para escribir textos con formato definido de manera cómoda y rápida. Es parte del proyecto Docutils dentro de la comunidad de Python. Tiene la principal ventaja de que ese texto puede usarse para generar documentos equivalentes en HTML, LaTeX, docbook, etc
- **asciidoc**: formato de documento de texto para escribir notas,



Observa la diferencia entre sintaxis **Markdown** (izquierda) y sintaxis **HTML** (derecha), ¿con cuál te quedas?

javadoc

Este tipo de utilidad de creación de documentación a partir de comentarios ha sido desarrollada por Oracle.

Javadoc permite la creación de documentación de API en formato HTML.

API

Del inglés Application Programming Interface (Interfaz de programación de aplicaciones). En la programación orientada a objetos, las API ofrecen funciones y procedimientos para ser utilizados por otras aplicaciones.

javadoc

Actualmente, es el estándar para crear comentarios de clases desarrolladas en Java. La sintaxis utilizada para este tipo de comentarios es empezar por `/**` y finalizar con `*/`, donde se incorporará el carácter `*` para cada línea, tal como se muestra a continuación:

En el siguiente ejemplo se efectúa una propuesta de los comentarios tipo Javadoc para la clase factorial, clase que calcula el factorial de un

```
/**
 * Clase que calcula el factorial de un número.
 * @Author IOC
 * @Version 2012
 */
public class Factorial {

    /**
     * Calcula el factorial de n.
     *  $N! = N * (n-1) * (n-2) * (n-3) * \dots * 1$ 
     * @Param n es el número al que se calculará el factorial.
     * @Return n! es el resultado del factorial de n
     */
    public static double factorial (double n) {

        if (n == 0)
            return 1;
        else
        {
            double resultado = n * factorial (n-1);
            return resultado;
        }
    }
}
```


javadoc

- La diferencia entre este tipo de comentarios y el resto de los comentarios es que los comentarios Javadoc sí tienen una estructura específica a seguir para ser escritos. En cambio, los comentarios internos dan completa libertad para ser implementados.
- Otra diferencia significativa es el objetivo de los comentarios. Los **comentarios internos** se hacen en todo el código fuente, mientras que los comentarios **Javadoc** están pensados para ser utilizados al principio de cada clase y de cada método.
- Finalmente, los comentarios Javadoc generan de forma automática la documentación técnica del software, en formato HTML .

Javadoc: documentación de apis

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

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.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interfaces relating to fonts.

Javadoc: documentación de apis

La documentación Javadoc es una colección de páginas HTML de todas las clases, métodos, parámetros y retornos junto con la información y especificaciones que quiera incluir el desarrollador de la API que en el caso de las clases de JDK incluye abundantes e interesantes detalles de implementación a tener en cuenta al usar las clases.

Javadoc: documentación de apis

Se genera a partir del propio código fuente de las clases con los comentarios incluidos que siguen cierto formato precediendo la definición de las clases y métodos.

Al estar código y documentación en el propio archivo de código fuente es más fácil mantener sincronizados el código y su documentación

Javadoc: documentación de apis

La documentación en el código fuente se incluye en comentarios que preceden una clase o método, además, con anotaciones se pueden documentar los parámetros y el valor de retorno.

Se pueden incluir etiquetas HTML junto con algunas “**anotaciones Javadoc**”.

Algunas anotaciones Javadoc incluidas en el JDK son las que veremos a continuación pero también se pueden desarrollar anotaciones propias (taglet/doclet) o personalizar los estilos de la documentación para cambiar el contenido, información incluida o adaptar los estilos según los colores de la organización.

Javadoc: documentación de apis

Los comentarios tipo Javadoc siguen unos estándares en su creación:

```
/**
```

```
* Texto con alguna descripción (posibilidad de usar HTML)
```

```
*
```

```
* Etiquetas javadoc (comienzan por @)
```

```
*
```

```
*/
```

Javadoc: documentación de apis

Hay dos tipos de **etiqueta**

- En **bloque**

- @tag texto

- En **línea**

- {@tag texto}

Una etiqueta en **bloque** debe aparecer al inicio de la línea de comentario.

El texto asociado con una etiqueta

```
/**
 * Clase que calcula el factorial de un número.
 * @Author IOC
 * @Version 2012
 */
public class Factorial {

    /**
     * Calcula el factorial de n.
     * N! = N * (n-1) * (n-2) * (n-3) * ... * 1
     * @Param n es el número al que se calculará el factorial.
     * @Return n! es el resultado del factorial de n
     */
    public static double factorial (double n) {

        if (n == 0)
            return 1;
        else
        {
            double resultado = n * factorial (n-1);
            return resultado;
        }
    }
}
```

```
/**
 * An example of inline tag. It computes {@code n1 + n2}.
 */
```

Javadoc: documentación de apis

Las etiquetas de **Javadoc** van precedidas por @, estos son los mas usados:

ETIQUETA	DESCRIPCIÓN
@author	Autor de la clase. Solo para las clases.
@version	Versión de la clase. Solo para clases.
@see	Referencia a otra clase, ya sea del API, del mismo proyecto o de otro. Por ejemplo: @see cadena @see paquete.clase#miembro @see enlace
@param	Descripción parámetro. Una etiqueta por cada parámetro.
@return	Descripción de lo que devuelve. Solo si no es void. Podrá describir valores de retorno especiales según las condiciones que se den, dependiendo del tipo de dato
@throws	Descripción de la excepción que puede propagar. Habrá una etiqueta throws por cada tipo de excepción.
@deprecated	Marca el método como obsoleto. Solo se mantiene por compatibilidad.
@since	Indica el nº de versión desde la que existe el método.

Javadoc: documentación de apis

Documentación de clases e interfaces

@author Nombre del autor de la clase o interfaz.

@version Versión de la clase y fecha

```
/ **
 * Clase que calcula el factorial de un número.
 * @Author IOC
 * @Version 2012
 * /
public class Factorial {
```

Javadoc: documentación de apis

Documentación de constructores y métodos

@param	nombre del parámetro	descripción de su significado y uso
@return		descripción de lo que se devuelve
@exception	nombre de la excepción	excepciones que pueden lanzarse
@throws	nombre de la excepción	excepciones que pueden lanzarse
@deprecated		Indica que el uso del elemento está desaconsejado

Javadoc: documentación de apis

@param nombre descripción

```
/**
 * Removes from this List all of the elements whose index is between
 * fromIndex, inclusive and toIndex, exclusive. Shifts any succeeding
 * elements to the left (reduces their index).
 * This call shortens the ArrayList by (toIndex - fromIndex) elements. (If
 * toIndex==fromIndex, this operation has no effect.)
 *
 * @param fromIndex index of first element to be removed
 * @param toIndex index after last element to be removed
 */
protected void removeRange(int fromIndex, int toIndex) {
    ...
}
```

Javadoc: documentación de apis

Documentación de constructores y métodos

@return descripción

```
/**
 * Tests if this vector has no components.
 *
 * @return <code>true</code> if and only if this vector has
 *         no components, that is, its size is zero;
 *         <code>false</code> otherwise.
 */
public boolean isEmpty() {
    return elementCount == 0;
}
```

Javadoc: documentación de apis

@throws tipo descripción

- Aplicable a constructores y métodos
- Describe las posibles excepciones del constructor/método
- Un @throws por cada posible excepción S
- Si es de a

```
/**
 * Parses the string argument as a signed decimal
 * <code>long</code>. The characters in the string must all be
 * decimal digits, except that...
 *
 * @param      s a <code>String</code> containing the <code>long</code>
 *              representation to be parsed
 * @return     the <code>long</code> represented by the argument in
 *              decimal.
 * @exception  NumberFormatException if the string does not contain a
 *              parsable <code>long</code>.
 */
public static long parseLong(String s)
    throws NumberFormatException {
    ....
}
```

Javadoc: documentación de apis

@see referencia

- Aplicable a clases, interfaces, constructores, métodos, atributos y paquetes
- Añade enlaces de referencia a otras partes de la documentación
- Variantes,
 - @see string
 - @see label
 - @see package.class#member label

```
package com.jdojo.utility;

/**
 * A dummy class.
 *
 * @see "Online Java Tutorial"
 * @see <a href="http://www.oracle.com">Oracle Website</a>
 * @see com.jdojo.utility.Calc Calculator
 * @see com.jdojo.utility.Calc#add(int, int) Add Method
 * @see com.jdojo.utility.Calc#multiply(int, int)
 */
public class Dummy {
}
```

Javadoc: documentación de apis

@deprecated descripción

```
/**
 * @deprecated explanation of why it was deprecated
 */
@Deprecated
public void deprecatedMethod() {
    ...
}
```

Javadoc: documentación de apis

- **{@inheritDoc}**: hereda el comentario Javadoc de la clase o método superior en la jerarquía de clases.
- Implícito (automático). La herramienta de generación de la documentación toma la descripción o el tag de la clase o interfaz de nivel superior.
- Explícito: Copia la documentación del elemento de nivel superior

```
/**
 * (superclase) ...
 *
 * @throws NullPointerException if <code>dst</code> is <code>>null</code>
 */
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    ....
}
```

```
/**
 * (subclase) ...
 *
 * @throws NullPointerException {@inheritDoc}
 */
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
```


Javadoc: documentación de apis

Reglas elementales

- La primera frase de cada comentario Javadoc debe ser una frase resumen con una **descripción concisa y completa**, terminada en punto, y seguida de un espacio, tabulador o retorno de carro
- Usar la **etiqueta** `<code>` para palabras clave y nombres
- Preferible el uso de la **tercera persona**

* Devuelve el índice del primer elemento...

* Devolvemos el índice del primer elemento... ⇐ Evitar

- Empezar con un **verbo** la descripción de los métodos
- **Omitir el sujeto** cuando es obvio

* @param peer nombre del peer

* @param peer parámetro con el nombre del peer ⇐ Evitar

Javadoc: documentación de apis

La etiqueta HTML `<code>` te permite darle un formato específico al texto como si fuera código de ordenador.

Esto ofrece un determinado tamaño y una distancia específica determinada para el código del ordenador.

HTML

```
<code>Este texto fue formateado con la etiqueta code.</code>
```

Demo

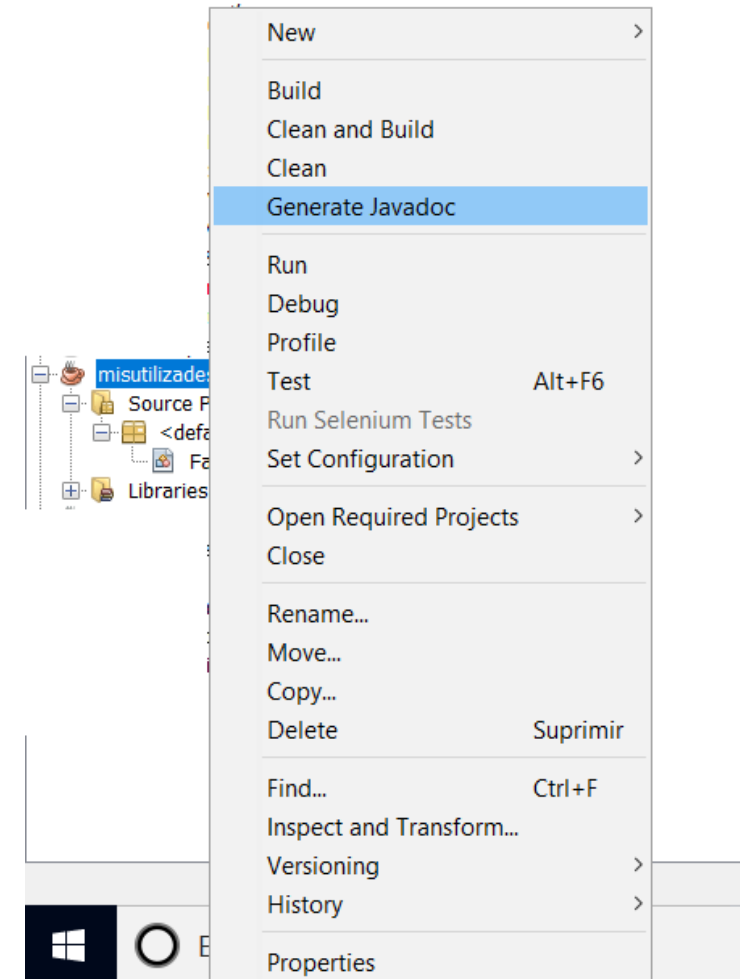
```
Este texto fue formateado con la etiqueta code.
```

```
/**
 * A utility class to perform basic calculations on numbers. All methods in this class are
 * <code>static</code>. It provides methods to perform addition, subtraction, multiplication,
 * and division.
 *
 * @author Kishori Sharan
 * @since Version 1.0
 */
public final class Calc {
    // More code goes here
}
```

Utilización de javadoc con netbeans

Muchos IDE que permiten programar en Java ofrecen funcionalidades para poder crear documentación a partir de Javadoc. Netbeans no es una excepción y también ofrece esta posibilidad.

Una vez desarrollado un proyecto en Java y añadidos los comentarios con las estructuras y estándares Javadoc, se podrá generar la documentación de forma automática utilizando la funcionalidad *Generar Javadoc* del menú *Proyecto*



Api clase factorial con javadoc

Se abre el navegador

The screenshot shows a web browser window displaying the Javadoc API for the `Factorial` class. The browser's address bar shows the URL `Factorial`. The page has a dark blue header with navigation links: `PACKAGE`, `CLASS` (highlighted), `USE`, `TREE`, `DEPRECATED`, `INDEX`, and `HELP`. Below the header, there are links for `PREV CLASS`, `NEXT CLASS`, `FRAMES`, and `NO FRAMES`. The main content area is titled **Class Factorial** and shows the class hierarchy: `java.lang.Object` and `Factorial`. The class is defined as `public class Factorial extends java.lang.Object`. A description in Spanish states: "Clase que calcula el factorial d'un nombre." Below this, there is a section for **Constructor Summary** with a sub-section for **Constructors**. It lists the constructor `Factorial()` with the description "Constructor and Description". Below this, there is a section for **Method Summary** with sub-sections for **All Methods** (highlighted), **Static Methods**, and **Concrete Methods**. It lists the static method `factorial(double n)` with the description "Calcula el factorial de n." At the bottom, there is a section for **Methods inherited from class java.lang.Object** listing methods like `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, and `wait`.

Factorial

All Classes

Factorial

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

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

Class Factorial

java.lang.Object
Factorial

public class Factorial
extends java.lang.Object

Clase que calcula el factorial d'un nombre.

Constructor Summary

Constructors

Constructor and Description

Factorial()

Method Summary

All Methods Static Methods Concrete Methods

Modifier and Type Method and Description

static double factorial(double n)
Calcula el factorial de n.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

ejemplo






<http://gpd.sip.ucm.es/rafa/docencia/programacion/tema1/javadoc.html>

Actividad guiada - individual

2. Incluye anotaciones Javadoc en el control de clases y objetos de la primera evaluación y genera la documentación
 - a. Documentar todas las **clases** indicando lo que es/hace
 - El **autor**
 - La **versión** por su fecha
 - b. Documentar todos y cada uno de los **métodos**
 - i. Lo que **hace**
 - ii. **Parámetros**. Para cada parámetro hay que escribir
 - @param
 - el nombre del parámetro
 - una somera descripción de lo que se espera en el parámetro
 - iii. El **resultado** que devuelve
 - @return
 - una somera descripción de lo que devuelve




Actividad autónoma

3. Actividad **en parejas**

-  Cada alumno pensará en una **clase**, y la **desarrollará con algunos métodos**.
-  Una vez realizado este primer paso, **generará la documentación javadoc** utilizando las anotaciones vistas en clase.
-  Cuando se obtengan los **ficheros html correspondientes a la documentación, se pasarán al compañero, el cual tendrá que desarrollar una pequeña clase donde se haga uso de la clase desarrollada por su compañero**.
-  El último paso será pasarle este **fichero java (donde se prueba la clase) al compañero** para ver que se ha realizado de manera correcta teniendo en cuenta únicamente la documentación
-  Escribe las conclusiones oportunas del proceso realizado en

Actividad autónoma individual

4. Alternativas a Javadoc

-  Investiga y enumera varias alternativas a Javadoc (al menos tres), comentando brevemente sus principales características.
-  Elige una de estas herramientas y desarrolla la documentación de alguna clase Java.
-  Envía dicho ejercicio en formato pdf en la fecha establecida por el profesor

Programar y documentar en inglés

Razones:

- **No es capricho mío**, la sintaxis del lenguaje de programación está en inglés así que nuestros programas se estarán acercando más a la naturaleza del lenguaje.
- Es habitual que ciertas empresas te obliguen a hacerlo.
- En cuanto te habitúes a hacerlo verás que la mayor parte de código que encuentres disponible está en inglés
- Tú mismo querrás compartir mañana tu código en repositorios como Github o Bitbucket y si están escritos en inglés tendrás mayor aceptación y mayor número de descargas.