

Javadoc. Herramienta de documentación

Contenido

1	Introducción.....	2
1.1	Historia.....	3
2	Conceptos de API java, JRE y java.base.	3
2.1	Las API's de java.	3
2.2	Java.base, java.lang y javax	4
2.3	Librerías básicas java, ampliando conocimientos.....	5
2.4	Organización de la documentación de la API Java, desarrollada en Javadoc. ..	9
2.5	Ejemplo de librería Javadoc. La API String	12
3	Herramienta Javadoc. Javadoc.exe	18
3.1	Principios y recomendaciones de Oracle para documentación Java.....	18
3.1.1	Escribir especificaciones API.....	19
3.1.2	Documentación de la Guía de Programación	20
3.1.3	Fuentes de documentacion.....	20
3.2	Como documentar en Javadoc	21
3.2.1	Documentación a nivel de módulo	25
3.2.2	Documentación a nivel de paquete	26
3.2.3	Documentación a nivel de interfaz	27
3.2.4	Documentación a nivel de clase	28
3.2.5	Documentación a nivel de método	31
3.2.6	La anotación @deprecated	34
3.3	Generando la documentación Javadoc.....	35

1 Introducción

Javadoc es una **utilidad de Oracle para la generación de documentación de APIs** en formato **HTML** a partir de código fuente Java. **Javadoc es el estándar para documentar clases de Java**. La mayoría **de los IDEs utilizan Javadoc para generar de forma automática documentación** de clases. **Eclipse también utiliza Javadoc y permite la generación automática de documentación, y visionarla bien de forma completa para todas las clases de un proyecto, o bien de forma particular para una de las clases de un proyecto.**

El formato de comentarios utilizado por Javadoc es el estándar de facto de la **industria para documentar clases Java**. Algunos IDEs, como **IntelliJ IDEA, NetBeans y Eclipse, generan automáticamente Javadoc HTML**. Muchos editores de archivos ayudan al usuario a producir origen Javadoc y utilizan la información de Javadoc como referencias internas para el programador.

Javadoc también proporciona **una API para crear doclets y taglets**, que permite a los usuarios **analizar la estructura de una aplicación Java**. Así es como **JDiff puede generar informes de los cambios entre dos versiones de una API**.

Javadoc **no afecta al rendimiento en Java, ya que todos los comentarios se eliminan en tiempo de compilación**. Escribir comentarios y Javadoc es para comprender mejor el código y, por lo tanto, mantenerlo mejor.

Veamos en primer lugar qué se debe incluir al documentar una clase:

- a) **Documentación de módulo:** una descripción general.
- b) **Documentación a nivel de paquete:** también descriptivo.
- c) **Nombre de la clase, descripción general, número de versión, nombre de autores.**
- d) **Documentación de cada constructor o método** (especialmente los públicos) incluyendo: **nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general**, descripción de parámetros (si los hay), descripción del valor que devuelve.

1.1 Historia

Javadoc ha sido el generador de documentación de lenguaje Java desde el inicio de Java. Antes del uso de generadores de documentación era costumbre utilizar procesadores de texto que normalmente escribirían sólo documentación independiente para el software, pero era mucho más difícil mantener esta documentación en sincronía con el software en sí.

Javadoc ha sido utilizado por Java desde la primera versión, y por lo general se actualiza en cada nueva versión del Java Development Kit.

La sintaxis `@field` de Javadoc ha sido emulada por sistemas de documentación para otros lenguajes, incluido el Doxygen entre lenguajes, el sistema JSDoc para JavaScript y el HeaderDoc de Apple.

2 Conceptos de API java, JRE y java.base.

Empezamos introduciendo la API de java. Veréis como está toda la documentación de la api de java definida en Javadoc.

2.1 Las API's de java.

La **API Java** es una **interfaz de programación de aplicaciones** (API, por sus siglas del inglés: **Application Programming Interface**) proporcionada por los **creadores del lenguaje de programación Java**, que **da a los programadores los recursos para desarrollar aplicaciones** Java.

Como **el lenguaje Java es un lenguaje orientado a objetos**, la **API de Java provee de un conjunto de clases** utilitarias para efectuar toda clase de tareas necesarias dentro de un programa.

La **API Java está organizada en paquetes lógicos**, donde cada paquete contiene un **conjunto de clases relacionadas** semánticamente.

Existen numerosas **API de Java para realizar todo tipo de operaciones**,¹ algunas de las más conocidas son:

1. **JAXP**: para procesar **XML**.
2. **Servlets**: para facilitar la implementación de **soluciones web**.
3. **Hibernate**: para facilitar la **implementación de persistencia de datos**.

La **función de las API's de java** es **ofrecernos herramientas a los desarrolladores** para crear aplicaciones. Nos hacen la vida más fácil, **permitiéndonos reutilizar código**, en lugar de tener que programarlo nosotros. Utilidades como ordenar, tener clasificado, hacer **operaciones matemáticas**, hacer **búsquedas de texto**, pedir **datos al usuario** y **muchos más procesos** son cuestiones **que se repiten con frecuencia** en programación, y por tanto se encontrarán resueltas en el API de Java. Por supuesto que **podemos crear algoritmos propios para ordenar listas**, pero lo **más rápido y eficiente en general será usar las herramientas del API** disponibles porque están desarrolladas por profesionales y han sido depuradas y optimizadas a lo largo de los años y versiones del lenguaje.

2.2 Java.base, java.lang y javax

Java.base define **las API fundamentales de java SE Platform**, y por tanto la base para desarrollar aplicaciones que corran sobre la **JRE**, que contiene la **JVM**. Proveedores: la **JDK** es un módulo **que proporciona una implementación del proveedor del sistema de archivos jrt para enumerar y leer los archivos de clase y recursos** en una imagen en **tiempo de ejecución**. Es la que nos **permite crear y gestionar ficheros de clases y recursos, compilar, ejecutar** nuestros programas. Es la base sobre la que se desarrollan más API's y aplicaciones.

java.lang. Proporciona **clases que son fundamentales para el diseño del lenguaje de programación Java**. Las clases más importantes son **Object**, que es la raíz de la jerarquía de **clases y Class**, instancias de las cuales **representan clases en tiempo de ejecución**. Nos permite **crear nuevas clases y objetos, herencia y polimorfismo**.

Estas **dos librerías no hace falta importarlas**, ni requerirlas en módulos, son **importadas de manera implícita**. En cualquier caso, **todas las librerías java van organizadas en paquetes** al igual que **debemos organizar nuestros componentes de nuestras aplicaciones en paquetes**. Por ejemplo, para la librería de ficheros, en la documentación se referenciarán como **API java.io** o **package java.io**.

El lenguaje de programación Java utiliza el **prefijo javax** para un **paquete de extensiones Java estándar**. Estos incluyen **extensiones como javax.servlet**, que se ocupa de la **ejecución de servlets**. La **API jcr**, que se ocupa de la **biblioteca de contenido Java y acceso a repositorios como CMS**. La **API swing** que se ocupan de **los interfaces de escritorio**, etc. En resumen, **se usa para extensiones** como **Java Enterprise**, desarrollo para aplicaciones empresariales.

2.3 Librerías básicas java, ampliando conocimientos

Cuando tengamos **experiencia como programadores Java**, posiblemente **dispongamos de clases desarrolladas por nosotros mismos** que utilicemos en distintos proyectos. En **empresas grandes**, es **frecuente disponer de clases desarrolladas por compañeros** de la empresa que **usaremos de forma parecida a como se usa el API de Java**: **conociendo su interfaz** pero no su implementación. Trabajar **con una clase sin ver su código fuente requiere que exista una buena documentación** que nos sirva de guía. Hablaremos de la **documentación de las clases y proyectos en Java** un poco más adelante. De momento, vamos a **aprender a usar la documentación del API de Java**.

En primer lugar, debemos **tener una idea de cómo se organizan las clases del API**. Esta **organización es en forma de árbol jerárquico**, como se ve en la figura “Esquema orientativo de la organización de librerías en el API de Java”. Esta **figura trata de mostrar la organización del API de Java**, pero **no recoge todos los paquetes ni clases existentes** que son muchos más y no cabrían ni en una ni en varias hojas.

Los **nombres de las librerías** responden a este esquema jerárquico y **se basan en la notación de punto**. Por ejemplo, el **nombre completo para la clase ArrayList** sería **java.util.ArrayList**. Se permite el **uso de * para nombrar a un conjunto de clases**. Por ejemplo **java.util.*** hace referencia **al conjunto de clases dentro del paquete java.util**, donde tenemos ArrayList, LinkedList y otras clases.

Para **utilizar las librerías del API**, existen **dos situaciones**:

- a) Hay **librerías o clases que se usan siempre** pues constituyen **elementos fundamentales del lenguaje Java** como la clase **String**. Esta clase, perteneciente al **paquete java.lang**, se puede utilizar directamente en cualquier programa Java ya que **se carga automáticamente**.
- b) Hay **librerías o clases que no siempre se usan**. Para **usarlas dentro de nuestro código hemos de indicar que requerimos su carga mediante** una sentencia **import incluida en cabecera de clase**. Por ejemplo `import java.util.ArrayList;` es una sentencia que incluida en cabecera de una clase nos permite usar la clase ArrayList del API de Java.

Escribir `import java.util.*;` nos permitiría cargar todas las clases del paquete `java.util`. Algunos paquetes tienen decenas o cientos de clases. Por ello nosotros preferiremos en general especificar las clases antes que usar asteriscos ya que evita la carga en memoria de clases que no vamos a usar. Una **clase importada se puede usar de la misma manera que si fuera una clase generada por nosotros**: podemos crear objetos de esa clase y llamar a métodos para operar sobre esos objetos. Además, cada clase tendrá uno o varios constructores.

Encontrar un listado de librerías o clases más usadas es una tarea casi imposible. Cada programador, dependiendo de su actividad, utiliza ciertas librerías que posiblemente no usen otros programadores. Los **programadores centrados en programación de escritorio usarán clases diferentes** a las que usan **programadores web** o de gestión de bases de datos. Las clases y las librerías básicas deberás ir conociéndolas mediante **cursos o lecciones de formación en Java**. Las **clases y librerías más avanzadas deberás utilizarlas y estudiarlas a medida** que te vayan siendo necesarias para el desarrollo de aplicaciones, ya que su estudio completo es prácticamente imposible. Podemos citar clases de uso amplio.

En el **paquete `java.io`**: clases `File`, `Filewriter`, `FileReader`, etc. En el **paquete `java.lang`**: clases `System`, `String`, `Thread`, etc. En el paquete **`java.security`**: clases que permiten implementar **encriptación y seguridad**. En el **paquete `java.util`**: clases `ArrayList`, `LinkedList`, `HashMap`, `HashSet`, `TreeSet`, `Date`, `Calendar`, `StringTokenizer`, `Random`, etc. En los paquetes `java.awt` y `javax.swing` una biblioteca gráfica: desarrollo de **interfaces gráficas** de usuario con ventanas, botones, etc.

Insistimos en una idea: **no trateis de memorizar la organización detallada del API de Java** ni un **listado de clases más usadas** porque esto **tiene poco sentido**. Lo importante es **que conozcais la forma de organización, cómo se estructuran y utilizan las clases** y que aprendas a buscar información para encontrarla rápidamente cuando te sea necesaria.

Para programar en Java tendremos que recurrir continuamente a consultar la documentación del API de Java. Esta **documentación está disponible en cds de libros y revistas especializadas o en internet** tecleando en un buscador como yahoo, google o bing el texto **"api java 8"** (o "api java 6", "api java 10", etc.) según la versión que estés utilizando. La **documentación del API de Java en general es correcta y completa**. Sin embargo, en casos excepcionales puede estar incompleta o contener erratas.

Mucha de la **documentación la encontrareis en la página oficial de Oracle**. Para documentar **las API's se usa habitualmente la aplicación Javadoc de la JDK que nos permite generar documentación** a partir de **etiquetas, comentarios y ficheros de texto**, genera de manera automática la documentación.

El sitio oficial de Oracle para java es:

<https://www.oracle.com/java/>

Podeis **encontrar información sobre descargas, APT's y más**. Es **nuestra primera fuente de documentación como programadores**, y es de obligada consulta, pues es la documentación oficial. **Java JSE** es la **versión oficial de Java** desarrollada por **Oracle**, que nos ofrece entre otras cosas la **JRE y la JDK**, kit de desarrollo para Java de Oracle

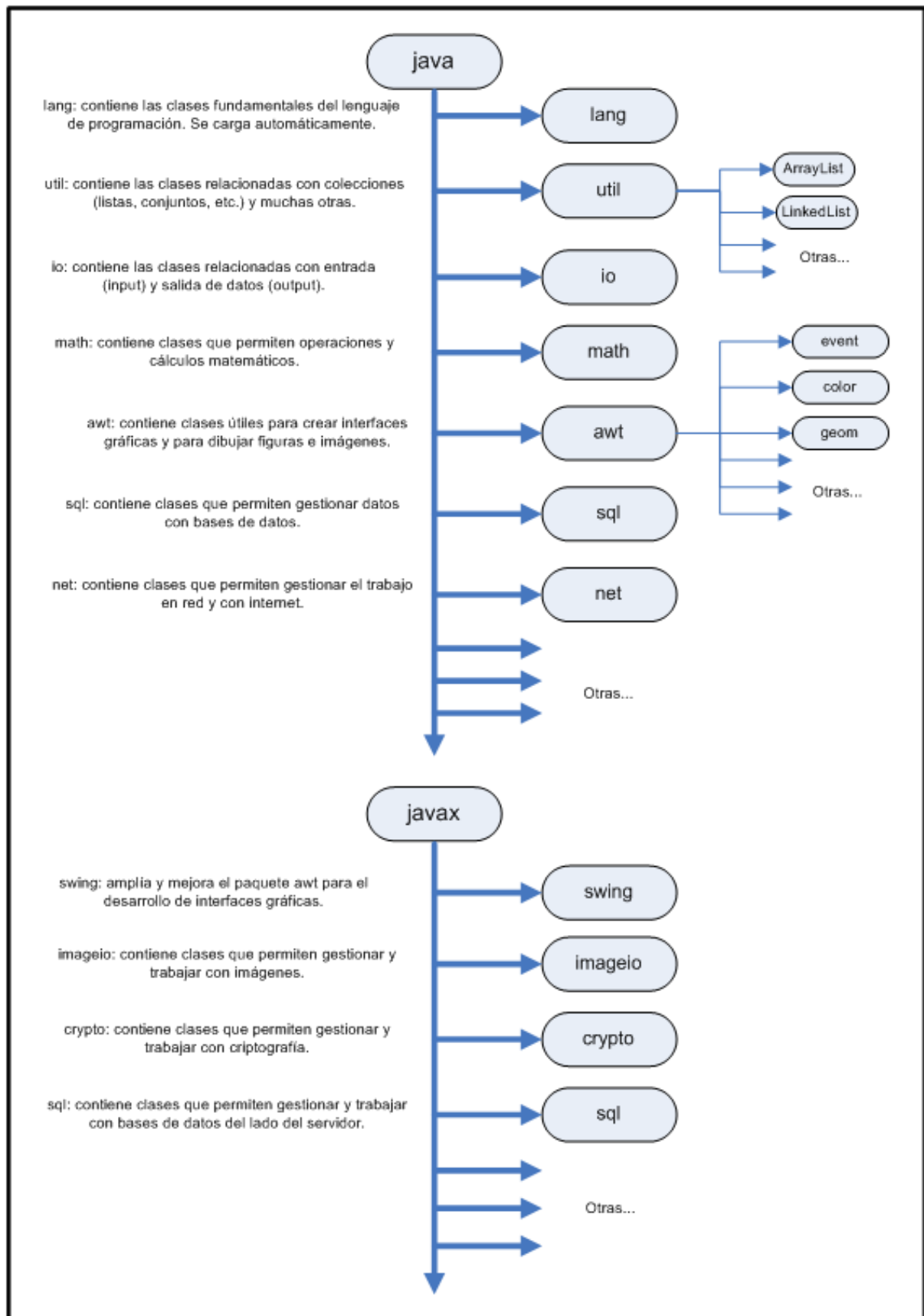
<https://docs.oracle.com/en/java/javase/14/docs/api/index.html>

<https://www.oracle.com/java/technologies/javase-downloads.html>

Encontrareis **información extra** en **openJDK**. Es una **plataforma abierta para desarrolladores de java**. Ofrece una **implementación alternativa a la versión oficial** de java que ofrece en la JSE

<https://openjdk.java.net/>

En la siguiente imagen **tenéis un gráfico resumen de las librerías** más utilizadas.



2.4 Organización de la documentación de la API Java, desarrollada en Javadoc.

Cómo se organiza este documento de API

La **documentación de la API** (Application Programming Interface) **tiene páginas web correspondientes a los elementos de la barra de navegación**, que se describen de la siguiente manera.

Overview (Visión general)

La página **Overview** es la primera página de la documentación de la API y proporciona una lista de todos los módulos con un resumen para cada uno. Esta página también puede contener una descripción general del conjunto de módulos.

Module (Modulo)

Cada **módulo** tiene una página que **contiene una lista de sus paquetes**, dependencias en otros módulos y servicios, con un resumen para cada uno. Esta página puede contener tres categorías:

- Módulos
- servicios
- Paquetes

Cada **paquete tiene una página que contiene una lista de sus clases e interfaces**, con un resumen para cada uno. Estas páginas pueden contener seis categorías:

- Interfaces
- Clases
- Enumeraciones
- Excepciones
- Errores
- Tipos de anotación
- Clase o interfaz

Cada **clase, interfaz, clase anidada e interfaz anidada** tiene **su propia página independiente**. Cada una de estas páginas tiene tres secciones que constan de una descripción de clase/interfaz, tablas de resumen y descripciones detalladas de miembros:

- Diagrama de herencia de clases
 - Subclases directas
 - Todas las subinterfaces conocidas
 - Todas las clases de implementación conocidas
 - Declaración de clase o interfaz
 - Descripción de clase o interfaz
-
- Resumen de la clase anidada
 - Resumen de campo
 - Resumen de la propiedad
 - Resumen del constructor
 - Resumen del método
 - Detalles del campo
 - Detalles de la propiedad
 - Detalles del constructor
 - Detalles del método

Cada **summary o resumen** contiene **la primera frase de la descripción detallada de ese elemento**. Las **entradas de resumen (summary)** son alfabéticas, mientras **que las descripciones detalladas están en el orden en que aparecen en el código fuente**. Esto conserva las agrupaciones lógicas establecidas por el programador.

Annotation Type

Cada tipo de anotación tiene su propia página independiente con las siguientes secciones:

- Annotation Type Declaration
- Annotation Type Description
- Required Element Summary
- Optional Element Summary
- Element Details
- Enum

Cada **enumeración** tiene **su propia página independiente** con las siguientes secciones:

- Declaración de Enum
- Descripción de Enum
- Resumen constante de Enum
- Detalles constantes de Enum
- Uso

Cada paquete documentado, clase e interfaz tiene su propia **página Use**. En esta página se describe qué paquetes, clases, métodos, constructores y campos usan cualquier parte de la clase o paquete dado. Dada una clase o interfaz A, su página “Use” incluye subclases de A, campos declarados como A, métodos que devuelven A y métodos y constructores con parámetros de tipo A. Puede acceder a esta página primero yendo al paquete, clase o interfaz, luego haciendo clic en el enlace “Usar” en la barra de navegación.

Tree (Class Hierarchy)

Hay una **página jerarquía de clases** para todos los paquetes, además de una jerarquía para cada paquete. Cada **página de jerarquía contiene una lista de clases y una lista de interfaces**. Las clases se organizan mediante la estructura de herencia a partir de **java.lang.Object**. Las interfaces no heredan de java.lang.Object.

Al ver **la página Información general**, al hacer clic en **“Tree”** se muestra la jerarquía para todos los paquetes.

Al ver un paquete, una clase o una página de interfaz determinados, al hacer clic en “Árbol” (Tree) se muestra la jerarquía solo para ese paquete.

Deprecated API

La **entrada Deprecated API** enumera todas las API que han quedado en desuso. No se recomienda el uso de una API en desuso, generalmente debido a mejoras y normalmente se administra una API de reemplazo. Las API en desuso se pueden quitar en futuras implementaciones.

Index

El índice contiene un índice alfabético de todas las clases, interfaces, constructores, métodos y campos, así como listas de todos los paquetes y todas las clases.

Serialized Form

Cada clase **serializable o externalizable** tiene una **descripción de sus campos y métodos de serialización**. Esta información es de interés para los desarrolladores que mantienen la API, no para los desarrolladores que usan la API. Aunque no hay ningún vínculo en la barra de navegación, puedes llegar a esta información yendo a cualquier clase serializada y haciendo clic en "Formulario serializado" (Serialize form) en la sección "Ver también" (See also) de la descripción de la clase.

Constant Field Values

Enumera **los campos finales estáticos y sus valores**.

2.5 Ejemplo de librería Javadoc. La API String

En este apartado vamos a describir una documentación de Javadoc ya colgada en el sitio web del proyecto y que contiene la descripción y documentación completa de la API de Java referenciada anteriormente. Veremos también dentro una API llamada API String, la clase String, una librería de java que nos permite manejo de cadenas, objetos String en Java. En estos apuntes vamos a aprender a generar esta documentación, y sitios web similares con la documentación de nuestras APIs y programas. Recordar la fase de maven site, donde indicábamos el sitio web de nuestro proyecto para su documentación.

Acceder a este enlace e iremos desgranando que elementos y como organiza Javadoc la documentación. En este enlace tenéis la documentación de todas las librerías de Java, y Javax para la versión 14. Entrad.

<https://download.java.net/java/GA/jdk14/docs/api/index.html>

Lo primero a tener en cuenta es que tenemos todos los módulos, luego una pestaña con los pertenecientes a la JRE, otra con los módulos o librerías pertenecientes a la JDK, y luego una aparte con un *sólo modulo java.cardio*.

All Modules	Java SE	JDK	Other Modules
Module	Description		
java.base	Defines the foundational APIs of the Java SE Platform.		
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		
java.datatransfer	Defines the API for transferring data between and within applications.		
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, in		
java.instrument	Defines services that allow agents to instrument programs running on the JVM.		

Igualmente, en la parte superior nos encontramos la estructura general de cualquier sitio Javadoc que se genere con la herramienta Javadoc, y que describimos anteriormente. La revisaremos a posteriori, pero podéis ver como Java organiza sus proyectos



Java® Platform, Standard Edition & Java Development Kit Version 14 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-pur

IDK

Un módulo contiene paquetes, un paquete contiene clases. Vamos **a abrir el módulo java.base que es el módulo fundacional o Java core de java**. Ya hemos dicho que **no hace falta importarlo en nuestros proyectos**, esta importado de manera implícita, haced click en java.base.

OVERVIEW
MODULE
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

MODULE: DESCRIPTION | MODULES | PACKAGES | SERVICES

Module java.base

Defines the foundational APIs of the Java SE Platform.

Providers:
The JDK implementation of this module provides an implementation created by calling `FileSystems.newFileSystem(URI.create("jrt"))`.

Module Graph:

Tool Guides:
java launcher, keytool

Since:
9

Packages

Ahora vamos a **pinchar en el paquete java.lang**.

OVERVIEW
MODULE
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

Module java.base

Package java.lang

Provides classes that are fundamental to the design of the Java programming language and represent classes at run time.

Frequently it is necessary to represent a value of primitive type as if it were an object. `Double`, for example, contains a field whose type is `double`, representing that value in methods for converting among primitive values, as well as supporting such standard methods for representing the type `void`.

The class `Math` provides commonly used mathematical functions such as sine, cosine, and character strings.

Classes `ClassLoader`, `Process`, `ProcessBuilder`, `Runtime`, `SecurityManager`, and `System` environment inquiries such as the time of day, and enforcement of security policies.

Class `Throwable` encompasses objects that may be thrown by the `throw` statement. See [Throwable](#) for more details.

Character Encodings

Hacemos click **en la clase String** y podemos ver toda la documentación para esta clase y hemos recorrido la organización de la documentación java, Javadoc para la API de java.

OVERVIEW
MODULE
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

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

Module java.base

Package java.lang

Class String

java.lang.Object
java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>, Constable, ConstantPoolEntry

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable
```

The `String` class represents character strings. All string literals in Java programs are represented by instances of this class.

Strings are constant; their values cannot be changed after they are created.

```
String str = "abc";
```

is equivalent to:

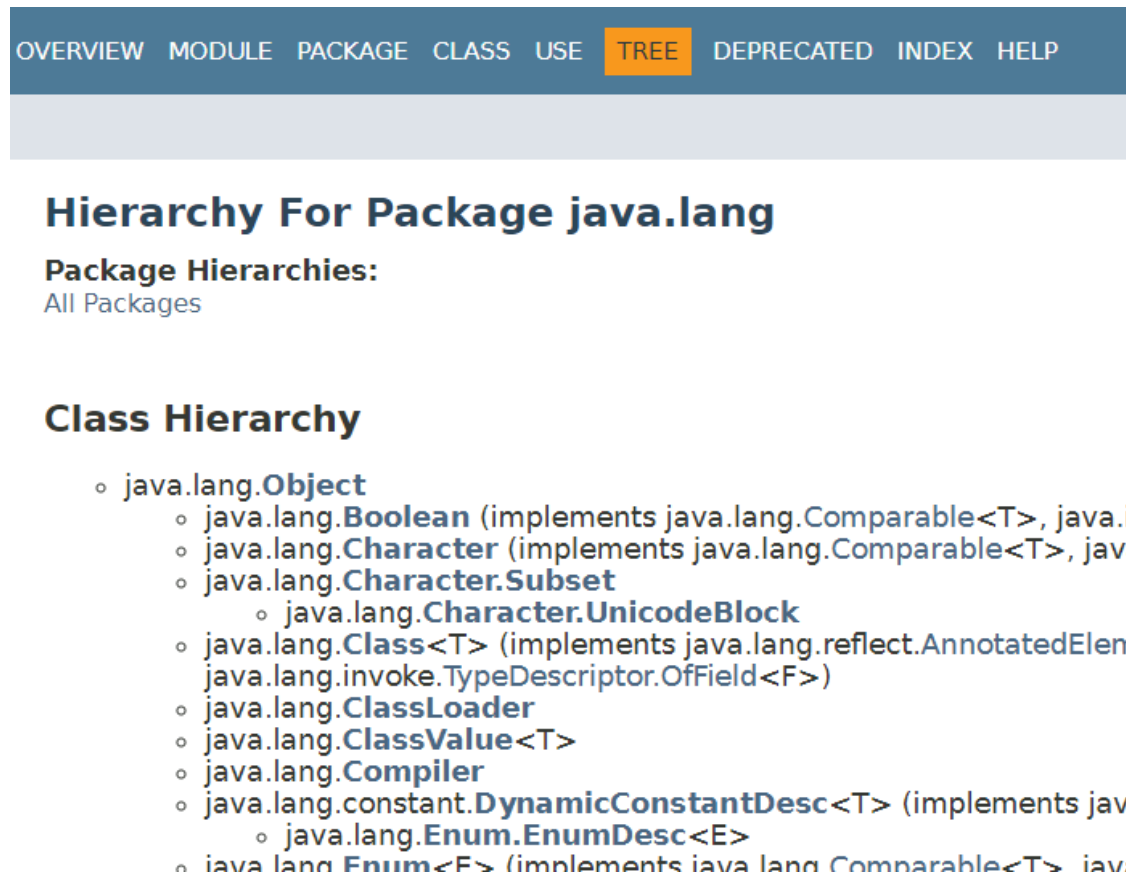
Si pinchamos en **USE**, veremos todos los paquetes que dependen de esta clase, que usan esta clase.

OVERVIEW
MODULE
PACKAGE
CLASS
USE
TREE
DEPRECATED
INDEX
HELP

Uses of Class java.lang.String

Packages that use String	
Package	Description
com.sun.jarsigner	This package comprises the interfaces and classes for digital signatures.
com.sun.java.accessibility.util	Provides a collection of interfaces and classes for accessibility.
com.sun.jdi	This is the core package of the Java Debug Interface.
com.sun.jdi.connect	This package defines connections between the JVM and the debugger.
com.sun.jdi.connect.spi	This package comprises the interfaces and classes for the SPI of the JDI.
com.sun.jdi.event	This package defines JDI events and event monitors.
com.sun.jdi.request	This package is used to request that a JDI event monitor be notified when a certain event occurs.

En **Tree**, el árbol de clases y subclases de **java.lang**



OVERVIEW MODULE PACKAGE CLASS USE **TREE** DEPRECATED INDEX HELP

Hierarchy For Package java.lang

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.**Object**
 - java.lang.**Boolean** (implements java.lang.Comparable<T>, java.i
 - java.lang.**Character** (implements java.lang.Comparable<T>, java
 - java.lang.**Character.Subset**
 - java.lang.**Character.UnicodeBlock**
 - java.lang.**Class**<T> (implements java.lang.reflect.AnnotatedElem
 - java.lang.**ClassLoader**
 - java.lang.**ClassValue**<T>
 - java.lang.**Compiler**
 - java.lang.constant.**DynamicConstantDesc**<T> (implements jav
 - java.lang.**Enum.EnumDesc**<E>
 - java.lang.**Enum**<E> (implements java.lang.Comparable<T>, java

En deprecated encontraremos todas las clases y método deprecated.

OVERVIEW	MODULE	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP
----------	--------	---------	-------	-----	------	------------	-------	------

Deprecated API

Contents

[For Removal](#)
[Modules](#)
[Interfaces](#)
[Classes](#)
[Enums](#)
[Exceptions](#)
[Fields](#)
[Methods](#)
[Constructors](#)
[Enum Constants](#)

For Removal	
Element	Description
<code>com.sun.java.accessibility.util.AWTEventMonitor.actionListener</code>	<i>This field is</i>
<code>com.sun.java.accessibility.util.AWTEventMonitor.adjustmentListener</code>	<i>This field is</i>

En index, un índice con todas las clases y métodos ordenados alfabéticamente

OVERVIEW	MODULE	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP
----------	--------	---------	-------	-----	------	------------	-------	------

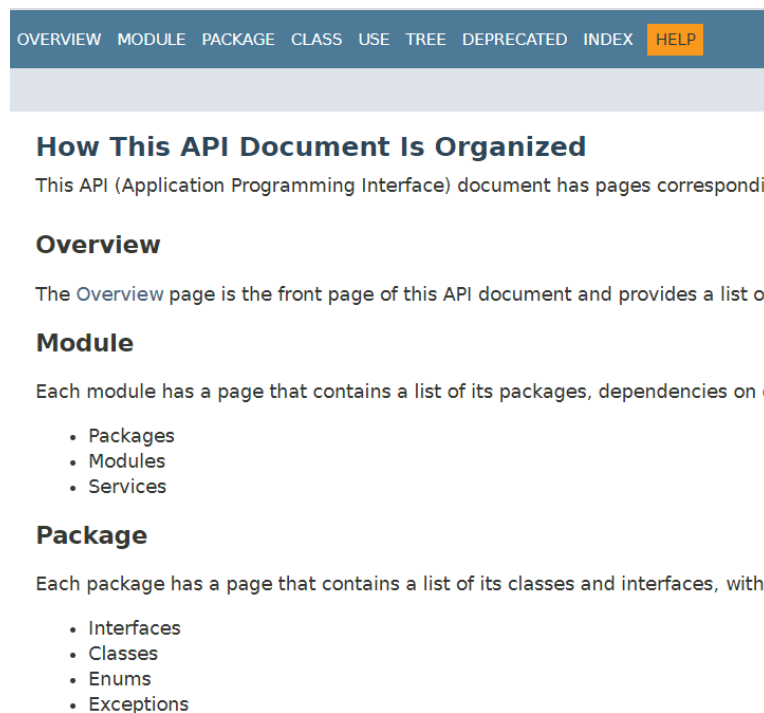
Index

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#) [_](#)
[All Classes](#) | [All Packages](#) | [System Properties](#)

A

- a** - Variable in class `java.awt.AWTEventMulticaster`
A variable in the event chain (listener-a)
- A** - Static variable in class `java.awt.PageAttributes.MediaType`
The MediaType instance for Engineering A, 8 1/2 x 11 in.
- A** - Static variable in class `javax.print.attribute.standard.MediaSize.Engineering`
Specifies the engineering A size, 8.5 inch by 11 inch.
- A** - Static variable in class `javax.print.attribute.standard.MediaSizeName`
A size.
- A** - Static variable in class `javax.swing.text.html.HTML.Tag`
Tag <a>
- A0** - Static variable in class `java.awt.PageAttributes.MediaType`
An alias for ISO_A0.
- A0** - Static variable in class `javax.print.attribute.standard.MediaSize.ISO`
Specifies the ISO A0 size, 841 mm by 1189 mm.
- A1** - Static variable in class `java.awt.PageAttributes.MediaType`
An alias for ISO A1.

En Help nos indica como la documentación está organizada, lo que hemos recorrido paso a paso.



3 Herramienta Javadoc. Javadoc.exe

Como ya **hemos indicado anteriormente la JDK nos proporciona la herramienta ,Javadoc, Javadoc.exe en Windows para generar nuestra documentación.** Se basa en etiquetas y anotaciones como ya hemos visto.

3.1 Principios y recomendaciones de Oracle para documentación Java

En **Java Software de Oracle**, tienen **varias directrices que podrían hacer que nuestros comentarios de documentación sean diferentes a los de desarrolladores de terceros.** Los comentarios de documentación como ya hemos visto definen la especificación oficial de la API de la plataforma Java y la documentación de guía de programación de la API.

Para poder **seguir la manera de documentar y trabajar en Java**, debemos realizar una documentación de **especificación de nuestros programas**, y una **guía de programa**, más completa, con por ejemplo, **manuales de uso y tutoriales**.

Por lo tanto, **habitualmente hay dos maneras diferentes de escribir comentarios de documentos**, como **especificaciones de la API o como documentación de la guía de programación**. Estos **dos objetivos se describen en las secciones siguientes**. Un personal con recursos puede permitirse el lujo de integrar ambos en la misma documentación (debidamente "fragmentado"); sin embargo, **las prioridades dictan que o demos más importancias la escritura de especificaciones de una Api** en comentarios de software que es de lo que trata este tema. O En su caso la documentación de programas

Por ejemplo, en el siguiente ejemplo podéis ver la diferencia entre :

- Documentación de API: <https://openjfx.io/javadoc/18/>

Vs

- Documentación de guía de referencia para programación: <https://openjfx.io/javadoc/18/javafx.graphics/javafx/scene/doc-files/cssref.html>

3.1.1 Escribir especificaciones API

En teoría, **la especificación de la API de Java comprende todas las explicaciones necesarias para hacer una implementación limpia** de la una **plataforma o framework** Java para "escribir una vez, ejecutarse en cualquier lugar"

La **especificación de la API de la plataforma Java** se define **mediante los comentarios de documentación** en el **código fuente** y los **documentos marcados** como **especificaciones con etiquetas** a partir de esos comentarios. De todas maneras, **cuando la API es muy compleja se puede desarrollar un sitio web o incluir enlaces a sitios web** en los que se desarrolla más en detalle la documentación.

La **especificación describe todos los aspectos del comportamiento de cada método para que un programador** pueda perfectamente **entender y usar la API**. La especificación debe describir (textualmente) **si una clase o método ofrece la funcionalidad Thread-safe, clase segura cuando la manejan hilos**.

La **especificación de API** es la **documentación mínima**, para uso de las **clases y métodos de la API**.

Ejemplo. La documentación de API de la librería java.nio

<https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/package-summary.html>

3.1.2 Documentación de la Guía de Programación

Lo que separa **las especificaciones de la API de una guía de programación son ejemplos, definiciones de términos de programación comunes, ciertas vistas generales conceptuales (como analogías) y descripciones de errores** de implementación y soluciones alternativas. Sin lugar a dudas toda la documentación **contribuye a la comprensión de la API por parte del desarrollador y ayudan** a este a **escribir aplicaciones robustas** más rápidamente.

En resumen, **añadimos más cosas como ejemplos, guías paso a paso , tutoriales, recomendaciones de uso**, etc. Detalles extra que se pueden incluir en la documentación de tu programa

- Se puede **incluir cualquiera o toda esta información en los comentarios de documentación** (y puede incluir etiquetas personalizadas, manejadas por un doclet personalizado, para facilitarla). Se puede **incluir enlaces a la documentación de la API de java o de otras APIs** que usemos en nuestros programas.
- Es útil **profundizar en cómo documentar errores y soluciones alternativas**. A veces **hay una discrepancia entre cómo debe funcionar el código y cómo funciona realmente**.

Ejemplo la API de la librería Turtle para Java

<http://sites.asmsa.org/java-turtle/documentation>

3.1.3 Fuentes de documentacion

Javadoc es la herramienta JDK que genera documentación de la API a partir de comentarios de documentación en código.

Archivos de origen

La **herramienta Javadoc** genera resultados a partir de cuatro tipos diferentes de archivos "fuente":

Javadoc puede generar resultados a partir de cinco tipos de ficheros fuente:

- Archivos **de código fuente para clases Java (.java)**: contienen comentarios de clase, interfaz, campo, constructor y método.
- Archivos de **comentario a nivel de módulo**.
- **Archivos de comentarios de paquetes**: contienen comentarios de paquetes
- **Descripción general de los archivos de comentarios** - estos contienen comentarios sobre el conjunto de paquetes
- **Archivos varios no procesados**: estos incluyen **imágenes, código fuente de ejemplo, archivos de clase, applets, archivos HTML** y cualquier otro elemento a la que desee hacer referencia desde los archivos anteriores.

3.2 Como documentar en Javadoc

Como ya hemos dicho, **Javadoc** es una **utilidad de Oracle** para la generación de **documentación de APIs en formato HTML** a partir de código fuente Java. De hecho, **Javadoc es el estándar para documentar clases de Java**. La mayoría de los **IDEs utilizan javadoc** para generar de forma automática documentación de clases. **Eclipse también utiliza javadoc y permite la generación automática de documentación**, y visionarla bien de forma completa para todas las clases de un proyecto, o bien de forma particular para una de las clases de un proyecto.

En **Javadoc** podemos documentar a nivel de modulo, de paquete, de clase y de método. Vamos a verlo en detalle en estos apuntes. Lo primero **introducir como realizar un comentario Javadoc y las etiquetas**. Un comentario es de tipo Javadoc si comienza de la siguiente manera con **/****, barra y **doble asterisco**. De esta manera la herramienta Javadoc **parsea el comentario**. Seguido siempre rellenamos **una descripción descriptiva de la función la clase**, modulo, paquete o método documentado. Después vienen las etiquetas con funciones específicas que ya veremos más adelante.

```

/**
 * Esta clase implementará o abstraerá un tipo particular de trabajador
 * en nuestra aplicación
 * llamado conserje
 *
 * @author Carlos Cano
 *
 */
public class Conserje extends Trabajador implements SueldoTrabajadores {

```

En la siguiente tabla tenéis la lista resumen de etiquetas Javadoc.

@author: Nombre del desarrollador.
@deprecated: métodos, clases y paquetes obsoletos
@param: definición de un parámetro en un método de clase.
@return: descripción del valor devuelto por un método
@see: enlace o asociación con otra clase y/o método.
@serial indica el id de un objeto serializable
@author: Nombre del desarrollador.
@deprecated: métodos, clases y paquetes obsoletos
@param: definición de un parámetro en un método de clase.
@version: versión del método o clase. Se suele usar de acuerdo con la liberación de versiones y la herramienta de versionado.
@throws: documenta una posible excepción que puede ser lanzada por el método.
@exception: es sinónima de throws.
@since: versión desde que lleva el método o clase utilizándose.
{@link}: incluye un enlace a otra sección de la documentación, método o clase.
{@linkplain}: incluye un enlace a otra sección de la documentación, método o clase, con formato de link estandar
@version: versión del método o clase. Se suele usar de acuerdo con la liberación de versiones y la herramienta de versionado.
@throws: documenta una posible excepción que puede ser lanzada por el método.
@exception: es sinónima de throws.

@since: versión desde que lleva el método o clase utilizándose.
@summary aparecerá en el área de resumen del elemento
@provides: indica que servicios provee el modulo, esta asociado con la orden Provides de module-info.java
@uses indica que servicios usa el módulo, relacionado con la orden uses de module-info.java
@code nos permite insertar código para explicaciones dentro de javadoc
@literal lo que escribimos dentro de esta etiqueta se interpreta literalmente no como javadoc
@docRoot

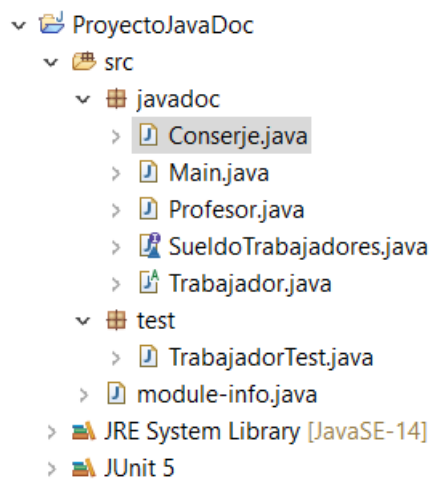
En Javadoc **podemos usar etiquetas HTML dentro de nuestros comentarios**. Las siguientes **etiquetas están permitidas a partir de Java 9**. Veremos sobre ejemplos como introducir HTML en nuestro Javadoc. **Javadoc genera páginas HTML con lo sí usamos etiquetas HTML** se **embeben en el HTML** de Javadoc

Etiquetas HTML permitidas en Javadoc			
A	EM	INPUT // sólo a partir de JDK 9	SUB
BIG // sólo a partir de JDK 8	FONT	LI	SUP // sólo a partir de JDK 8
B // sólo a partir de JDK 8	FOOTER // sólo a partir de JDK 9	LINK	TABLE
BLOCKQUOTE	FRAME // sólo a partir de JDK 8	LISTING // sólo a partir de JDK 9	TBODY
BODY	FRAMESET // sólo a partir de JDK 8	MAIN // sólo a partir de JDK 9	TD
BR	H1	MENU	TFOOT // sólo a partir de JDK 8
CAPTION	H2	META	TH
CENTER	H3	NAV // sólo a partir de JDK 9	

CITE // sólo a partir de JDK 8	H4	NOFRAMES // sólo a partir de JDK 8	
CODE	H5	NOSCRIP	
DD	H6	OL	
DFN // sólo a partir de JDK 8	HEAD	P	
DIR // sólo a partir de JDK 9	HEADER // sólo a partir de JDK 9	PRE	
DIV	HR	SCRIPT	
DL	HTML	SECTION // JDK 9 en adelante	
DT	I	SMALL	
DFN // JDK 8 en adelante	IFRAME // JDK 9 en adelante	SPAN	

Como ya hemos comentado, con **Javadoc** podemos **comentar módulos, paquetes** (packages), **clases y métodos**.

Haremos una introducción de como comentar paquetes, clases y métodos. Explicaremos las etiquetas, y entraremos en detalle en cada una de los elementos comentados. Para ello, **usaremos el siguiente proyecto** que se proporcionará con los apuntes en el fichero **ProyectoJávaDoc.zip**.



3.2.1 Documentación a nivel de módulo

Lo primero a comentar en un proyecto a partir de Java 9 es el **módulo**. El fichero **module-info.java**. En la etiqueta **@author** indicamos el autor, podemos incluir varias de ellas.

La etiqueta **@version** indica la versión actual de nuestro código. Debe ir en consonancia con el gestor de versiones, GIT en nuestro caso. La etiqueta **@since** indica **desde que versión de nuestra aplicación está disponible el módulo**, paquete o clase. Consta de dos números, por ejemplo, 1.2. El **segundo número** se usa para indicar los **cambios funcionales y de mantenimiento** de pequeño tamaño para una **versión**. Por ejemplo, la versión de **Android de lollipop 5.0**. Los diferentes **cambios de lollipop** pasa por la **5.1**, **5.2**, etc. Usamos la etiqueta **@version** y **@since**.

```
/**
 * Este módulo está desarrollado para ser un programa principal, no una API
 * En principio no será exportado por nadie. Maneja trabajadores y hace
 * calculo de sueldos e impuestos para estos trabajadores
 *
 * @author Carlos Cano
 * @author Serena Lopez
 * @version 1.2
 * @since 1.0
 */

module ProyectoJavaDoc {
    requires org.junit.jupiter.api;
}
```

En el ejemplo anterior, el método tiene de autor **Carlos Cano y Serena López @author Serena Lopez**, que se incorporó al **desarrollo después** y cambió este nuevo método. La **mayoría** de los **métodos** anteriores fueron creados por **otro autor**, el de la clase.

La versión es la 1.2, **@versión 1.2**. El método lleva **desarrollado** desde la 1.0, **@since 1.0**.

A nivel de modulo podemos usar las siguientes etiquetas

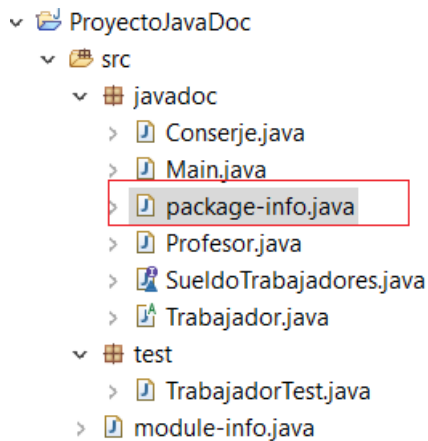
Module Declaration

```
{@author}, {@deprecated}, {@provides}, {@see}, {@since}, {@serialField},
{@uses}, {@version}

{@code}, {@docRoot}, {@index}, {@link}, {@linkplain}, {@literal}, {@summary}
```

3.2.2 Documentación a nivel de paquete

Para esta **documentación** deberemos crear un fichero **package-info.java** en la raíz del paquete.



Se incluirá una descripción del paquete, y las etiquetas que consideremos necesarias que luego especificaremos.

```
/**
 * El paquete incluira todas las clases de modelo de nuestro programa más la
 * clase principal.
 * Contiene un Interfaz SueldoTrabajadores, una clase abstracta Trabajador, y
 * dos clases que
 * heredan de ella e implementan el interfaz Profesor y Conserje
 *
 * @author Carlos Cano
 * @version 1.2
 * @since 1.0
 */

package javadoc;
```

Podemos **usar las siguientes etiquetas a nivel de paquete**

{ @author }, { @see }, { @since }, { @serialField }, { @version }

{ @code }, { @docRoot }, { @index }, { @link }, { @linkplain }, { @literal }, { @summary }

3.2.3 Documentación a nivel de interfaz

Veamos en primer lugar qué se debe incluir al documentar una clase:

a) **Nombre del interfaz, descripción general, número de versión, nombre de autores.**

b) **Documentación de cada método** incluyendo: nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general, descripción de parámetros (si los hay), **descripción del valor que devuelve.**

```
/**
 * Este interfaz define el comportamiento de la jerarquia de clases de
 * nuestro modelo. Será implementado por la clase abstracta
 * trabajador, que cederá la implementación a Profesor
 * y Conserje las clases que heredan de el
 * Obliga a implementar calculaSueldo() y calculaImpuesto a las subclases
 * Profesor y Conserje
 * @author Carlos Cano
 * @author Serena Lopez
 * @version 1.2
 * @since 1.0
 */
```

```
interface SueldoTrabajadores {
```

A nivel de **método observar como usamos la etiqueta @return** para indicar que parámetros **devuelve el método, un double en este caso**, Muy importante. **Podemos usar más etiquetas para métodos** que introduciremos más adelante.

```
/**
 * Este método calcula el sueldo para el trabajador específico. No
 * recibe parámetros
 * @see javadoc.Profesor#calculaSueldo()
 * @return double
 */
double calculaSueldo();
```

Importante resaltar aquí la etiqueta **@see** que nos generará un enlace **en la documentación a la clase Profesor**, método calculoSuelto. Es **una de las dos etiquetas que nos permite generar enlaces en java doc**. La explicaremos en detalle junto a link en el siguiente apartado.

Declaración Javadoc de tipos

Las declaraciones de tipo incluyen las **declaraciones de clases, interfaces, tipos de enumeración y tipos de anotación**.

{@author}, {@deprecated}, {@hidden}, {@param}, {@see}, {@since},
{@serialField}, {@version}

{@code}, {@docRoot}, {@index}, {@link}, {@linkplain}, {@literal}, {@summary}

3.2.4 Documentación a nivel de clase

Veamos en primer lugar qué se debe incluir al documentar una clase:

- a) **Nombre de la clase**, descripción general, número de versión, nombre de autores.
- b) **Documentación de cada constructor o método** (especialmente los públicos) incluyendo: nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general, descripción de parámetros (si los hay), descripción del valor que devuelve.

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**. La versión más reciente de **javadoc es la 8**.

Con **javadoc, documentamos un proyecto**, Aplicación ejecutable o API. Una **API no es sólo un ejecutable en si**, es **una librería que ofrece clases (objetos) y métodos, que son utilizados por otros programas JAVA**. Por ejemplo, **java.útil**, es un paquete de librerías, **una API**, que nos ofrece clases como **ArrayList** para ser usadas en nuestro programa.

Comenzamos con la cabecera de la clase

Lo primero a la hora de **comentar una clase es comentar su cabecera**. En la cabecera daremos una descripción general de la clase. Incluiremos detalles como:

1. La **descripción**,
2. Su **funcionalidad** que ofrece a otras clases
3. **Como se integra** o que labor hace dentro de **nuestro modelo de clases**.
4. Si **algún método u algoritmo de la clase es relevante**.

```

**
*
* <p>
* Este clase es la clase abstracta trabajador que define la base de la
* jerarquia de nuestro
* modelo. Heredarán de ella Profesor y Conserje
* Obliga a implementar funcion Trabajador: {@link
* javadoc.Trabajador#funcionTrabajador}
*
* </p>
*
* <p>
*
* Usamos en esta clase la API String para algunos de nuestros métodos
* <a
* href="https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang
* /String.html"> API String</a>
*
* </p>
*
* * Las subclases del modelo que heredan de trabajador son
* <ul>
* <li>
* <u>Profesor
* </li>
* <li>
* <u>Conserje
* </li>
* </ul>
*
* </p>
* @author Carlos Cano
* @author Serena Lopez
* @version 1.2
* @since 1.0
* @see javadoc.SueldoTrabajadores {@link javadoc.SueldoTrabajadores}
* /

```

```
public abstract class Trabajador implements SueldoTrabajadores {  
    protected int id;  
    protected String nombre;  
    protected double sueldo;  
}
```

Vamos a ver una a una las nuevas etiquetas que hemos introducido en estos comentarios:

Por **un lado como podéis ver con la etiqueta `@Link`** puedo **añadir enlaces a mi documentación Javadoc en cualquier parte de los comentarios**. Lo encontrareis en la imagen posterior.

```
{@link javadoc.Trabajador#funcionTrabajador}
```

Como **habéis visto con la etiqueta `<a>` de HTML** hemos añadido un enlace a la **documentación externa de la Api Stream**.

```
* <a  
* href="https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang  
* /String.html"> API String</a>
```

Igualmente fijaos como podemos construir **párrafos con la etiqueta `<p>` y una lista con ``**

```
* </p>  
*  
* * Las subclases del modelo que heredan de trabajador son  
* <ul>  
*     <li>  
*         Profesor  
*     </li>  
*     <li>  
*         Conserje  
*     </li>  
* </ul>  
*  
* </p>
```

```

Este clase es la clase abstracta trabajador que define la base de la jerarquia de nuestro modelo. Heredarán de ella Profesor y Conserje Obliga a implementar funcion Trabajador
Usamos en esta clase la API String para algunos de nuestros métodos
* Las subclases del modelo que heredan de trabajador son
    • Profesor
    • Conserje
Since:
1.0
Version:
1.2
Author:
Carlos Cano, Serena Lopez
See Also:
SuelidoTrabajadores

```

funcionTrabajador()

Enlace con @Link

Enlace con <a>

Enlace con @see

La diferencia entre @link y @linkplain es que produce el resultado en formato diferente

{@link #method()} – formato monoespaciado (código)

{@linkplain #method()} formato estandar.

3.2.5 Documentación a nivel de método

Documentaremos cada método publico cuando estemos generando documentación tipo de **especificación de API para nuestro programa**. Si la documentación es para uso **interno, para que los programadores, para mantenimiento o nuevas versiones de desarrollo**, comentaremos también los **métodos privados**.

Los **comentarios a nivel de método son muy similares a los anteriores**. Podemos usar @see @link, y todas las etiquetas que hemos introducido anteriormente. Hay dos etiquetas destacables propias de los métodos, y obligatorias. **@param y @return, que explicaremos sobre los ejemplos**

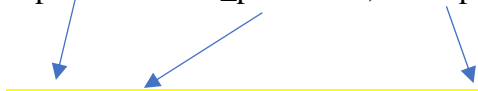
Comentando el constructor

Comenzaremos indicando como comentar un constructor de una clase. La etiqueta **@return no es necesaria**, la etiqueta, **@param si es obligatoria siempre y cuando el constructor tenga parámetros**:

En la **etiqueta @param se indica el nombre del parámetro, seguido de una descripción del mismo**:

Lo podéis ver en el ejemplo de constructor de la clase trabajador.

@param nombre_parametro, descripción


*@param id, introducimos el identificador, un entero de Trabajador

```
/**
 * Este método será reutilizado por las subclases para crear objetos
 * con la llamada
 * a super
 *
 *
 * @param id, introducimos el identificador, un entero de Trabajador
 * @param nombre, el nombre del trabajador creado
 * @param sueldo, su sueldo en decimal
 * @see Profesor#funcionTrabajador()
 javadoc.Conserje#funcionTrabajador()
 *
 */

public Trabajador(int id, String nombre, double sueldo) {

    this.id=id;
    this.nombre = nombre;
    this.sueldo = sueldo;
}
```

Ejercicio

1. Añadir una lista HTML en el Javadoc del constructor con los tipos de trabajadores que usaran este constructor.

2. Añadir un enlace externo a la API String de java versión 14.

Comentario de un método. Vamos a explicar cómo hemos comentado el método `iniciarConMayusculas` de la clase `formatos`. Es un método estático que recibe un parámetro `Nombre` y devuelve un `String`. En este caso usamos la etiqueta `@param` y `@return`.

```
* @param Nombre Un nombre con el nombre y apellidos separados por
* blancos
```

```
* @return String , el nombre y apellidos con las Iniciales en mayuscula
```

Como veis la etiqueta `@return` se indica el tipo de vuelto y una descripción de que se devuelve.

`@return` Tipo_Devuelto , descripción.

```
* @return String , el nombre y apellidos con las Iniciales en mayuscula
```

```
/**
 *
 * <p>
 *     * Este método recorre el nombre con los apellidos escribe * la
 * primera letra con
 * mayuscula para cada nombre y apellido
 * </p>
 *
 * <p>
 *     usamos la API String{@linkplain String}
 *     mirar el método trim{@link String#trim}
 *
 * </p>
 * @param Nombre Un nombre con el nombre y apellidos separados por
 * blancos
 *
 * @return String el nombre y apellidos con las Iniciales en mayuscula
 *
 * @see java.lang.String
 */
```

```
public static String inicialConMayusculas(String Nombre) {
```

```

        String trimNombre = Nombre.trim();

        String tratarNombre[] =trimNombre.split(" ");
        String Resultado ="";

        for (String s : tratarNombre) {

            Resultado = Resultado +
s.trim().substring(0,1).toUpperCase() + s.trim().substring(1) +" ";
        }

        return Resultado.trim();

    }
}

```

3.2.6 La anotación @deprecated

Por **último**, vamos a explicar como utilizar la anotación **deprecated** con un método. Se puede hacer a nivel de clase también, no lo olvidéis..

En Javadoc usamos la etiqueta **@deprecated** seguido de una descripción indicando , cuando y el porque. En el ejemplo indicamos desde que versión está deprecated y añadimos un link al método nuevo.

```

    *@deprecated desde version 1.1 porque
    * hay un método nuevo {@link Trabajador#calculaSueldoBase()}
    */

```

Este trozo de código que veis no es Javadoc. Estamos usando las anotaciones Java para indicar que el método ha sido deprecated. Los parámetros interiores no son necesarios, pero si aconsejables:

1. **Since:** para indicar desde que versión ha sido deprecated, una cadena.
2. **forRemoval:** si se va a borrar en el futuro, un booleano.

```

@Deprecated(since = "1.1", forRemoval = true)

```

```

/**
 * Este método será reutilizado por las con la llamada a super
 * subclases para crear objetos
 *
 * @return double, sueldo del trabajador
 * @see Profesor#funcionTrabajador()
 * javadoc.Conserje#funcionTrabajador()
 * @version 1.1
 * @since 1.0

```

```

*@deprecated desde version 1.1 porque
*    hay un método nuevo {@link Trabajador#calculaSueldoBase()}
*/

@Deprecated(since = "1.1", forRemoval = true)
public double viejoCalculoSueldo() {

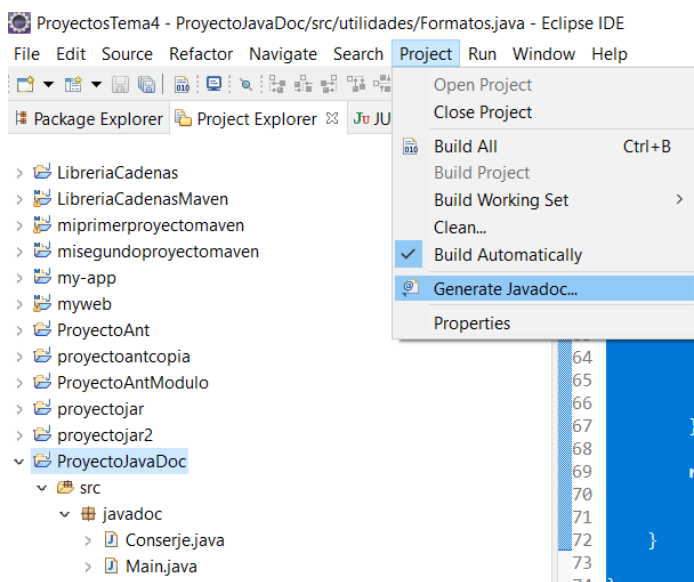
    return sueldo*0.12;
}

```

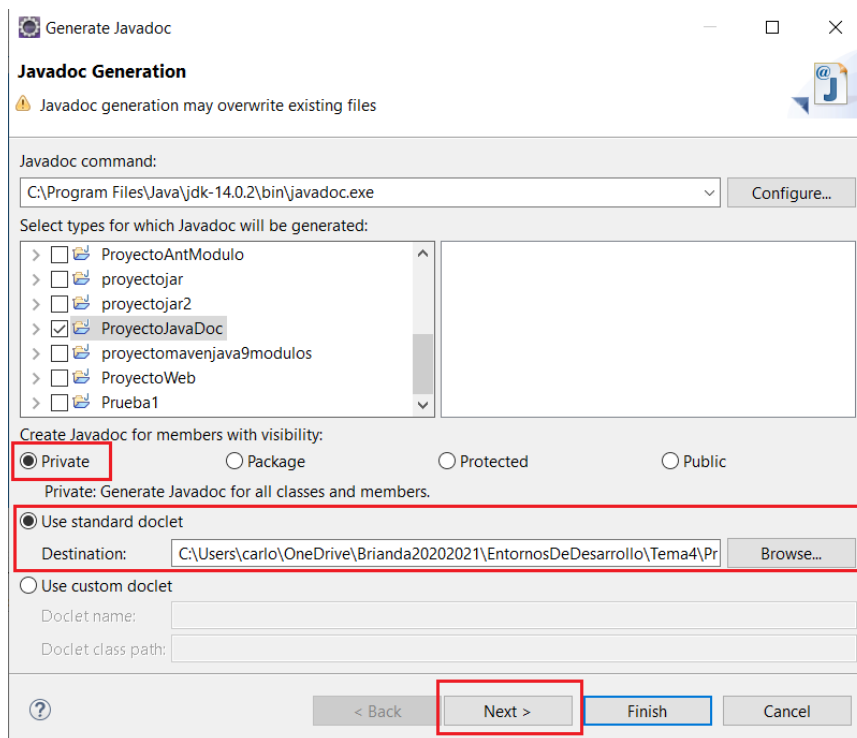
3.3 Generando la documentación Javadoc

Una vez que **tenemos documentadas todas las clases y métodos del proyecto** el paso final es **generar la documentación Javadoc desde Eclipse**. **Antes de generar el Javadoc, realizar un clean y un build del proyecto para garantizarnos que todo este correcto.**

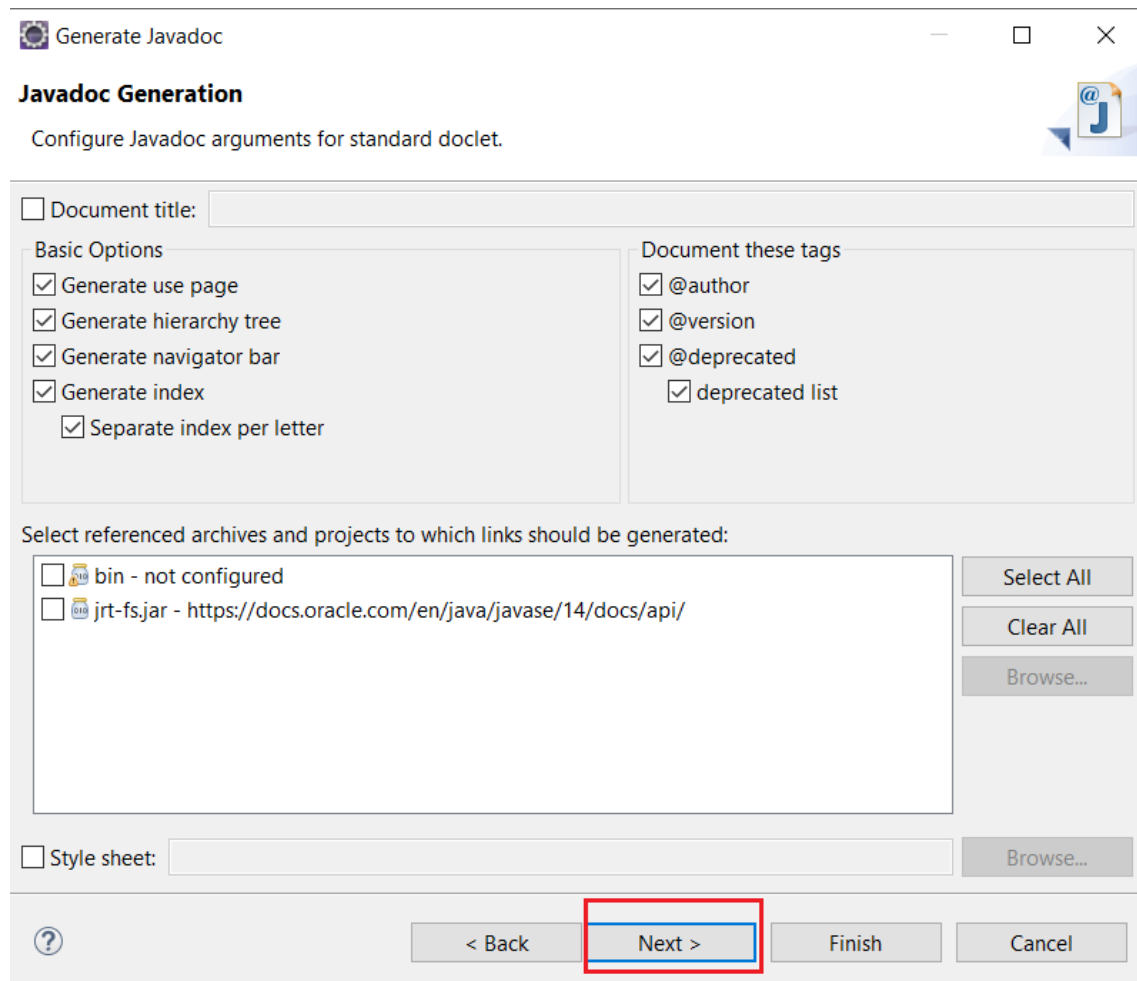
1. Para ellos seleccionamos el proyecto y en el menú Project seleccionamos la opción generate Javadoc. **Seleccionar el proyecto entero sino la documentación se generará incompleta.**



2. **Seleccionamos private en package visibility**, la salida que **nos da por defecto el Javadoc**, y pulsamos next.

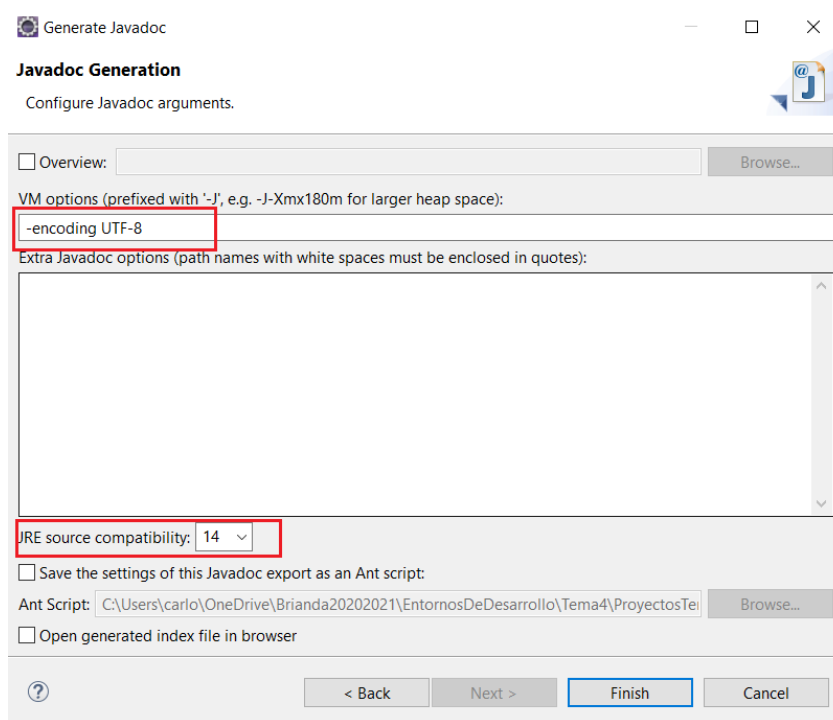


3. Volvemos a pulsar en siguiente con las opciones mostradas en la imagen.

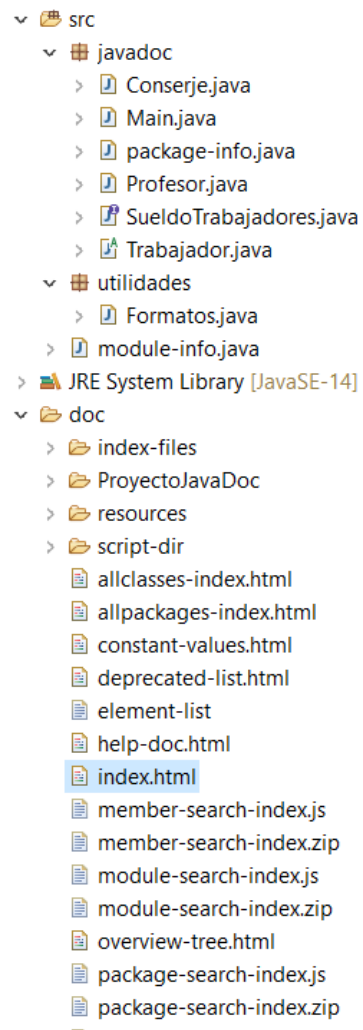


4. Para **finalizar** añadimos como parámetro a la máquina virtual el encoding **UTF-8** y la compatibilidad para versión **14** de JRE.

-encoding UTF-8



Como **resultado obtendréis dentro de la nueva carpeta doc** del propio proyecto **vuestra documentación javadoc.**



Abrid index.html desde vuestro browser y comprobar que se ha generado toda la documentación.

Haced click en el nombre del paquete para revisar las clases:

Module ProyectoJavaDoc

Este modulo esta desarrollado para ser un programa principal, no una API En principio no será exportado por nadie. Maneja trabajadores y hace calculos de sueldo

Since: 1.0
Version: 1.2
Author: Carlos Cano, Serena Lopez

Packages

Concealed

Package	Description
javadoc	El paquete incluirá todas las clases de modelo de nuestro programa más la clase principal. Contiene un Interfaz SueldoTrabajadores, una clase abstracta Profesor y Conserje

utilidades

Indirect Exports

From	Packages
java.base	com.sun.security.ntlm java.io java.lang java.lang.annotation java.lang.constant java.lang.invoke java.lang.module java.lang.ref java.lang.reflect java.lang.runti
java.nio.channels.spi	java.nio.charset java.nio.charset.spi java.nio.file java.nio.file.attribute java.nio.file.spi java.security java.security.cert java.security.interfac

Podéis **navegar clase a clase** si queréis.

[MODULE](#) [PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

Module ProyectoJavaDoc

Package javadoc

El paquete incluirá todas las clases de modelo de nuestro programa más la clase principal. Contiene un Interfaz SueldoTrabajadores, una clase abstracta Trabajador, y Conserje

Since: 1.0
Version: 1.2
Author: Carlos Cano

Interface Summary

Interface	Description
SueldoTrabajadores	Este interfaz define el comportamiento de la jerarquía de clases de nuestro modelo.

Class Summary

Class	Description
Conserje	Esta clase implementará o abstrará un tipo particular de trabajador en nuestra aplicación llamado conserje
Main	
Profesor	
Trabajador	Esta clase es la clase abstracta trabajador que define la base de la jerarquía de nuestro modelo.

Ejercicio

Estudiar el código y comentar el Javadoc para todas las clases y métodos que se han quedado incompletos en el proyecto.