Documentación y JavaDoc.

Entornos de desarrollo

ÍNDICE DE CONTENIDOS

- Introducción.
- Comentarios.
- JavaDoc.
- Generación de documentación: Tags.
- Generación en Eclipse.

Introducción

Hay muchas formas de programar, pero no siempre el código desarrollado por un programador tiene que ser comprendido por otro. Por ello, especialmente en entornos de desarrollo y proyectos que involucran gran cantidad de personas, se suele elaborar una **guía de estilo de programación** que recoge **reglas de codificación** tales como:

- ✓ Separar el código en ficheros y directorios.
- ✓ Elegir a nombres para variables y funciones.
- ✓ Alinear la sangría de un bloque, etc.

Introducción

Hay muchas versiones y ninguna es perfecta. Una guía de estilo de programación sirve para **unificar** la manera de crear código entre grupos de trabajo o de forma reconocida de forma generalizada. Un código es más fácil de entender si las mismas cosas se hacen de la misma manera. De tal forma que una persona que se integre en el proyecto tarda menos tiempo en entenderlo.

En general, podríamos estableces las siguientes **reglas** que debería cumplir el código de java. Estas reglas, suelen ser implementadas automáticamente por IDEs como Eclipse.

- ✓ Se define una clase por fichero y este fichero se llama igual que la clase
- ✓ Se define donde guardar los ficheros fuente y donde los ficheros compilados para poder usar un control de versiones.
- ✓ Usar la notación de Kernighan and Ritchie con la llave { en la misma línea.
- ✓ Se define la sangría como 4 espacios y se descarta tabuladores.
- ✓ Se puede exigir poner comentarios en estilo de JavaDoc u otras herramientas de documentación automatizada.
- ✓ linear la sangría de un bloque, etc.

Introducción

Si queremos de instaurar estas reglas en nuestra empresa, hay que tener en cuenta, que al principio solo complican el trabajo. En todo caso, un estilo debe ser fijado al **principio del proyecto**. Para una compañía pequeña puede ser una buena idea adherirse a un estilo popular como el de Sun Para Java o el de la biblioteca STL para C++.

Comentarios

Programar y desarrollar requiere además de poseer diversos conocimientos disponer de una buena documentación de consulta y referencia. Es muy importante insertar comentarios en el código para documentar las tareas que realiza y los pasos que se sigue para ello.

Por otro lado, en muchos casos es necesario aportar al destinatario del software documentación técnica que indique la funcionalidad del software. Hay dos tipos de comentarios:

- ✓ **Documentación de uso interno**: La persona que desarrolla el código inserta comentarios para detallar aspectos del código generado. Estos comentarios sirven para que, en un futuro otra persona o ella misma pueda comprender que tarea desarrolla el código de una forma rápida.

- ✓ **Documentación de uso externo**: Documentación pensada para que otras personas comprender y utilicen el código generado en desarrollos posteriores. Un claro caso es la documentación de las API y los Framework de programación donde las funciones tienen perfectamente detallada su funcionalidad y uso.

Comentarios

Existen herramientas que generan de forma automatizada la documentación externa. No obstante, muchos lenguajes de programación aportan formas sencillas para el programador de añadir comentarios dentro del código. Posteriormente, a partir de dichos comentarios se genera la documentación externa.

JavaDoc

Por defecto, Oracle suministra con Java una herramienta que se denomina JavaDoc existiendo otras en el mercado. JavaDoc genera documentación de APIs en formato HTML a partir de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java. La mayoría de los IDEs los generan automáticamente basándose en el estándar JavaDoc.

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

La fuente para JavaDoc 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.

Generación de documentación: Básica

La documentación para JavaDoc ha de incluirse entre símbolos de comentario que han de empezar con una barra y doble asterisco (/**), y terminar con un asterisco y barra simple (*/).

```
/**
 *  Esto es un comentario con Javadoc
 */
```

La ubicación le define a JavaDoc qué representa el comentario. Si está incluido justo antes de la declaración de clase se considerará un comentario de clase, y si está incluido justo antes de la signatura de un constructor o método se considerará un comentario de ese constructor o método.

```java
/**
 * Definida como elemento básico a la hora de crear a las distintas
 * personas representadas en el programa.
 * @author Fran
 */
public class Persona {
   private String nombre;
   private int edad;
   /**
    * Constructor de la clase Persona. Se crea un objeto Persona
    * dado por su nombre y su edad.
    * @param nombre Nombre de la Persona.
    * @param edad Edad de la Persona.
    */
   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }
```

Generación de documentación: Uso de Tags

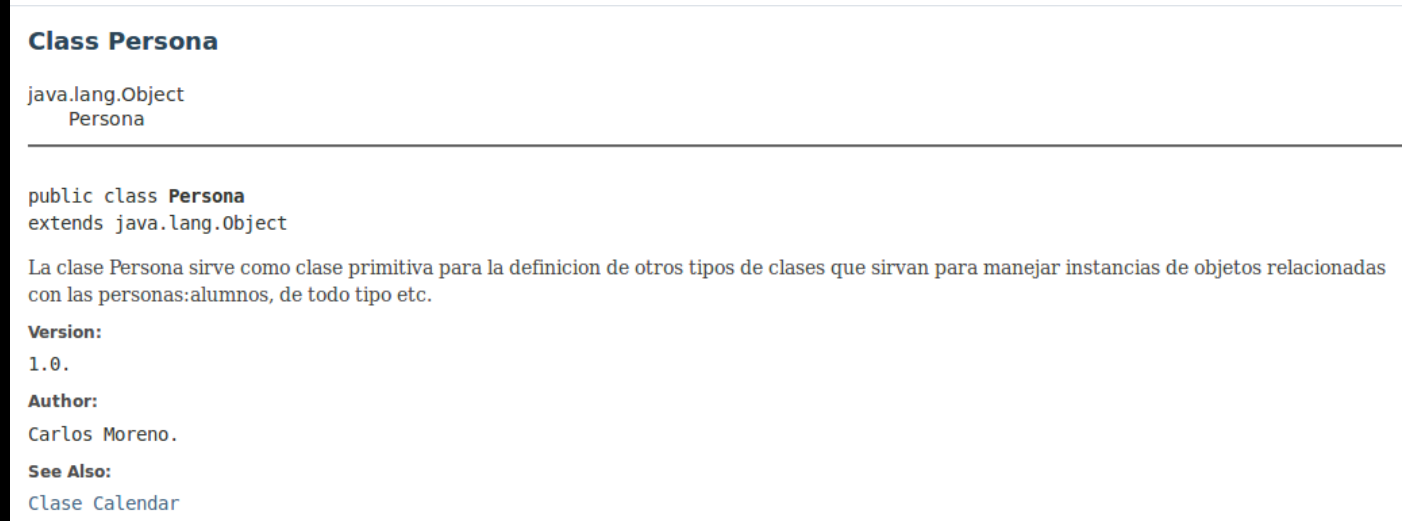
JavaDoc usa ciertas palabras reservadas (tags) precedidas por el carácter "@". Si no existe al menos una línea que comience con @ no se reconocerá el comentario para la documentación de la clase.
Hay tags que solo se usan en la documentación de clases mientras que otras se utilizan en la documentación de los métodos. En la siguiente tabla se muestra una serie de tags usados en JavaDoc.

Tag	Descripción
@author	Nombre del desarrollador.
@version	Versión del método o clase.
@param	Definición de un parámetro de un método. Debe de haber un @param para todos los parámetros del método.
@return	Informa del tipo de dato que devuelve el método. No se puede usar en constructores o métodos void ya que no tiene sentido.
@throws	Excepción lanzada por el método, posee un sinónimo de nombre @exception con lo cual se pueden usar ambas.
@see	Asocia con otro método o clase. También puede incluir una dirección que referencie una página WEB.
@since	Especifica la versión del producto
@serial	Describe el significado del campo y sus valores aceptables. Otras formas validas son @serialField y @serialData
@deprecated	Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores.

Tags de Clase

Los Tags de clase aportan información sobre las características de la clase en cuestión. Se indican antes de la definición de la cabecera de la clase. Entre ellas están las siguientes: **@author**, **@see**, **@version**.

```
/**
 * La clase Persona sirve como clase primitiva para la definicion de otros clases
 * tipos de que sirvan para manejar instancias de objetos
 * relacionadas con las
 * personas:alumnos, de todo tipo etc.
 *
 * @version 1.0.
 * @author Fran.
 * @see <a href="https://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html">Calendar</a>
```



Class Persona

java.lang.Object
    Persona

public class **Persona**
extends java.lang.Object

La clase Persona sirve como clase primitiva para la definicion de otros tipos de clases que sirvan para manejar instancias de objetos relacionadas con las personas:alumnos, de todo tipo etc.

Version:

1.0.

Author:

Carlos Moreno.

See Also:

Clase Calendar

Tags de Método

Los Tags de método aportan información sobre las características del método en cuestión. Se indican antes de la definición de la cabecera de la clase.

```
/**
    * Retorna true si la persona es mayor que la
edad pasada como
    * parametro. Retorna la excepcion EdadIlegal si
es menor que cero
    * la edad introducida.
    *
    * @param edad Edad con la que se quiere
comprobar.
    * @return true si la persona es mayor que esa
edad, false en caso contrario.
    * @throws EdadIlegal.
    */
   public boolean isMayorEdad(int edad) {}
```

isMayorEdad

```
public boolean isMayorEdad(int edad)
```

Retorna true si la persona es mayor que la edad pasada como parametro. Retorna la excepcion EdadIlegal si es menor que cero la edad introducida.

Parameters:

edad - Edad con la que se quiere comprobar.

Returns:

true si la persona es mayor que esa edad, false en caso contrario.

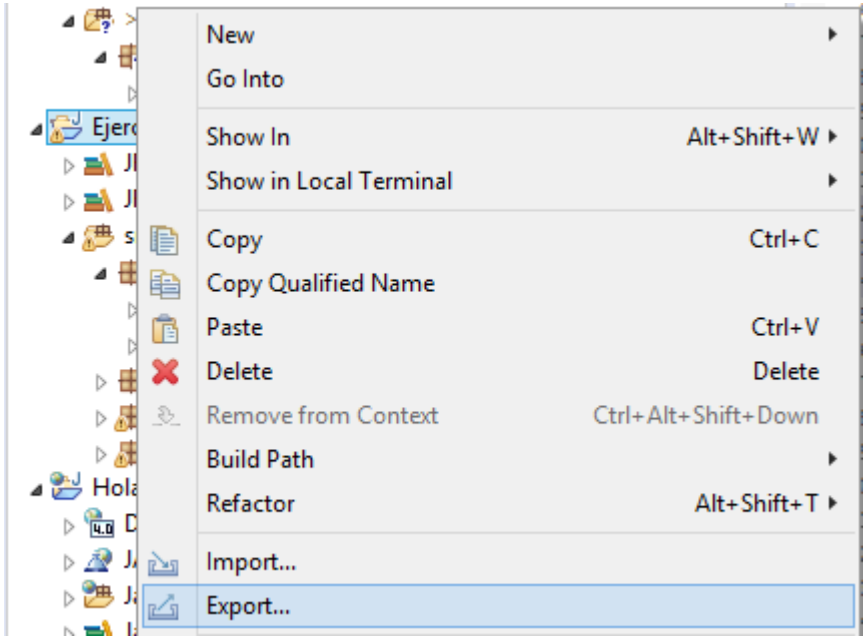
Throws:

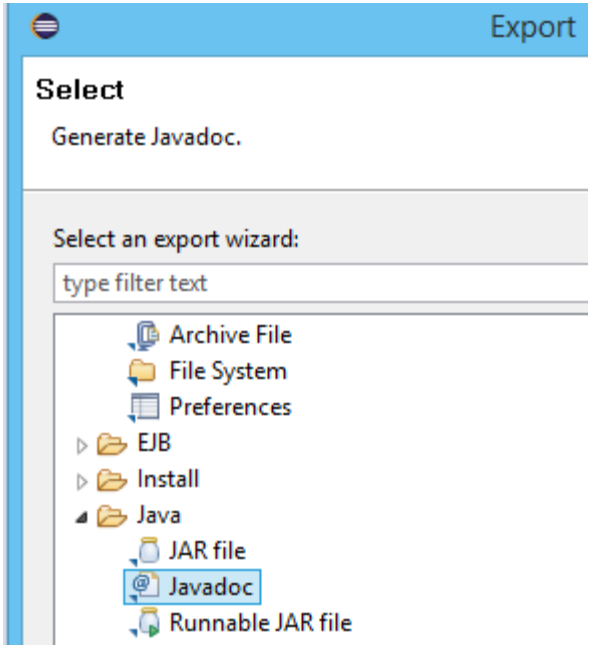
EdadIlegal.

Generación

- ✓ Una vez añadido los comentarios,completado todo, vamos a generar el JavaDoc, para ellos:
  Hacemos click derecho sobre nuestro proyecto > Export > Java > Javadoc

- ✓ Para generar el bloque de estándar de documentación, pulsamos el atajo de teclado: Alt + Shit + J






```
/**
 *
 */
public Alumno()
{
    asignaturas = new String[11];
```