UT4 – OPTIMIZACIÓN Y DOCUMENTACIÓN – PARTE I - DOCUMENTACIÓN

RESULTADOS DE APRENDIZAJE ASOCIADOS
4.- Optimiza código empleando las herramientas disponibles en el entorno de desarrollo.
CRITERIOS DE EVALUACIÓN
a) Se han identificado los patrones de refactorización más usuales.
b) Se han elaborado las pruebas asociadas a la refactorización.
c) Se ha revisado el código fuente usando un analizador de código.
d) Se han identificado las posibilidades de configuración de un analizador de código.
e) Se han aplicado patrones de refactorización con las herramientas que proporciona el entorno de desarrollo.
f) Se ha realizado el control de versiones integrado en el entorno de desarrollo.
g) Se han utilizado herramientas del entorno de desarrollo para documentar las clases.

UT4 – OPTIMIZACIÓN Y DOCUMENTACIÓN – PARTE I - DOCUMENTACIÓN

Índice de contenido

1.- Contenidos 3
2.- Importancia de la documentación 3
2.1.- Creación de documentación de calidad 4
2.2.- Diversas perspectivas de la documentación 5
2.3.- Documentación automática 6
3.- Documentación del código fuente 7
3.1.- Uso de JAVADOC en Eclipse 8
3.1.1- Uso de etiquetas de documentación 9
3.1.2- Generar la documentación 12
4.- Referencias bibliográficas 17

1.- Contenidos

En estos apuntes, comenzamos destacando la importancia de la documentación de clases en el entorno de desarrollo, esencial para la comprensión y mantenimiento del código. Seguidamente, nos enfocamos en la refactorización, identificando y aplicando patrones comunes, y resaltando la relevancia de las pruebas asociadas para asegurar la funcionalidad del código. Finalmente, abordamos el control de versiones integrado en el entorno de desarrollo, un paso clave para gestionar cambios y mantener la coherencia y trazabilidad del proceso de desarrollo, garantizando así la calidad y sostenibilidad del software.

2.- Importancia de la documentación

En el ámbito del desarrollo de software, la creación de documentación es una actividad constante. Los documentos generados por un amplio espectro de profesionales, como programadores, analistas y jefes de proyecto demuestran que es preferible no producir documentación a elaborar una que no aporte valor.

Se ha observado que, tanto durante como antes y después del proyecto, se produce una considerable cantidad de documentación destinada a diferentes perfiles, como programadores, usuarios finales y *testers*.

> **La clave de la documentación radica en su calidad.**

En ocasiones, la documentación en proyectos de TI adolece de esta calidad, especialmente visible en manuales de usuario que se centran en explicar lo obvio y dejan de lado información realmente útil.

Esta deficiencia de calidad suele deberse a la limitada dedicación de tiempo a la documentación o a la falta de interés de los programadores por esta tarea, resultando en un producto final deficiente.

2.1.- Creación de documentación de calidad

Se ofrecen aquí algunas directrices para elaborar documentación de alta calidad:

1. Antes de escribir, es recomendable **realizar esquemas**, que pueden ser incluidos en los manuales.
2. **Organizar la información por su relevancia**. Es preferible contar con un documento principal conciso y varios anexos que un único documento extenso.
3. **Elaborar resúmenes** y documentos maestros es fundamental **para sintetizar y explicar el conjunto de la documentación**.
4. Es aconsejable **utilizar estándares y herramientas de documentación** específicos de cada lenguaje de programación, como ***Javadoc*** para Java.
5. Al escribir documentación para usuarios, es esencial anticipar y responder a sus posibles preguntas.
6. La **claridad** es crucial en la documentación dirigida a usuarios, eligiendo adecuadamente entre un tono más directo (segunda persona) o uno más formal (tercera persona).
7. **Elegir** correctamente la **herramienta de documentación**. En algunos casos, es más efectivo utilizar herramientas hipertextuales o colaborativas en lugar de procesadores de texto tradicionales.

Dediquemos un minuto a buscar…

¿Qué son las herramientas hipertextuales?

8. Generalmente, se elaboran dos tipos de documentos para los usuarios: **una guía de usuario sencilla y un manual de referencia más completo**.
9. Considerar el nivel de conocimiento del usuario al redactar la documentación, detallando claramente lo que no deben o pueden hacer, así como las consecuencias de ciertas acciones.

2.2.- Diversas perspectivas de la documentación

El software puede ser examinado desde un enfoque técnico (componentes, clases, ficheros, interfaces, bases de datos, etc.) y un enfoque funcional (funcionamiento del sistema, acciones de los componentes, interacción con otro software, etc.). La documentación debe abarcar ambos aspectos, orientando el técnico hacia los desarrolladores y el funcional hacia los usuarios.

La documentación es una tarea continua que comienza con el proyecto y lo acompaña hasta su finalización.

A) Fase Inicial: Se establecen los cimientos del proyecto, incluyendo la planificación y las estimaciones de viabilidad.

B) Análisis: Se definen y analizan los requisitos del cliente, documentando todas las interacciones y acuerdos.

C) Diseño: Se determina la arquitectura del software, con documentos de diseño general y detallado elaborados por los analistas.

D)Codificación: Se documenta en detalle el código, incluyendo comentarios y especificaciones técnicas.

E) Pruebas: Se llevan a cabo pruebas funcionales y técnicas, documentando meticulosamente los resultados.

F) Explotación: Se registra cualquier incidencia o fallo para su posterior análisis y solución.

G) Mantenimiento: Se documentan todas las acciones de mantenimiento, con detalles de las operaciones realizadas.

> **La calidad de la documentación es crucial.** Sin ella, un software carece del soporte necesario para su mantenimiento y evolución.

2.3.- Documentación automática

La documentación automática es un complemento valioso a la práctica de documentar el código y tiene un papel significativo en el desarrollo de software. Vincular la importancia de la documentación del código con la documentación automática nos lleva a considerar cómo esta última puede mejorar y facilitar el proceso de documentación.

- **Eficiencia en la Creación de Documentación:** La documentación automática utiliza herramientas que generan documentación directamente a partir del código fuente. Esto ahorra tiempo y esfuerzo a los desarrolladores, quienes de otro modo tendrían que escribir esta documentación manualmente.
- **Consistencia y Estándar:** Las herramientas de documentación automática ayudan a mantener un formato estándar y consistente en toda la documentación. Esto es crucial cuando varios desarrolladores trabajan en el mismo proyecto, asegurando que todos sigan los mismos estándares y formatos.
- **Actualización en Tiempo Real:** Una gran ventaja de la documentación automática es su capacidad para actualizar la documentación en tiempo real a medida que el código cambia. Esto asegura que la documentación siempre esté sincronizada con la versión más reciente del código.
- **Enfoque en el Código:** Al automatizar la generación de documentación, los desarrolladores pueden centrarse más en escribir y mejorar el código, en lugar de dedicar tiempo a actualizar manualmente la documentación.
- **Facilita la Revisión del Código:** La documentación automática puede incluir detalles como quién escribió una parte específica del código, cuándo y por qué se hicieron cambios. Esto es útil para la revisión del código y para entender la evolución del proyecto.
- **Integración con Herramientas de Desarrollo:** Las herramientas de documentación automática suelen integrarse bien con otros entornos y herramientas de desarrollo, lo que facilita un flujo de trabajo más fluido.
- **Mejora la Calidad del Código:** Al igual que la documentación manual, la automática fomenta la escritura de código limpio y estructurado.

Los desarrolladores son más conscientes de cómo su código será interpretado y presentado en la documentación.

- **Accesibilidad y Distribución:** La documentación generada automáticamente puede ser fácilmente accesible para los miembros del equipo y otras partes interesadas, a menudo a través de plataformas en línea o intranets, lo que facilita el acceso y la distribución de la información.
- **Cumplimiento de Normativas:** En industrias donde el cumplimiento de normativas es crítico, la documentación automática asegura que todos los cambios y revisiones estén debidamente documentados y sean fácilmente rastreables.

3.- Documentación del código fuente

Es **fundamental documentar detalladamente el código fuente** para esclarecer las funciones del programa.

Esto asegura que todo el equipo de desarrollo comprenda las acciones realizadas y sus razones.

Un código con buena documentación facilita enormemente la corrección de fallos y la incorporación de nuevas características, adaptándolo así a distintas situaciones. Esto contrasta con un programa sin documentación.

Existen dos principios fundamentales que siempre deben recordarse:

- Todo software contiene errores, y su detección es solo cuestión de tiempo y uso frecuente, especialmente si el programa es exitoso.
- Todos los programas exitosos pasan por modificaciones a lo largo de su existencia.

Al documentar, es crucial detallar el propósito y funcionamiento de una clase, un método, y las razones detrás de su implementación. Esto incluye explicar las responsabilidades de una clase, un paquete, un método o una variable, su uso previsto, y los algoritmos empleados para solucionar problemas específicos, entre otros aspectos.

Para la documentación de proyectos, existen múltiples herramientas, dependiendo del lenguaje de programación. Ejemplos incluyen ***PHPDoc*** y ***phpDocumentor*** para PHP, ***Javadoc*** para Java, ***y JSDoc*** para JavaScript.

En esta parte del tema, nos centraremos en ***Javadoc***, la herramienta específica para Java.

3.1.- Uso de JAVADOC en Eclipse

Javadoc es la utilidad de Java para extraer y generar documentación directamente del código en formato HTML. Para que la documentación sea en verdad útil, deberemos escribir los comentarios de acuerdo con la documentación y recomendaciones de Javadoc.

La documentación y el código se incluirán dentro del mismo fichero.

Veamos a continuación las recomendaciones sobre los comentarios y la documentación del código fuente.

Los tipos de comentarios para generar documentación son:

- ***Comentarios de línea:*** comienzan con los caracteres "//" y termina con la línea.
- ***Comentarios de tipo C:*** comienzan con los caracteres "/*", y terminan con los caracteres "*/". Pueden agrupar varias líneas. Se les suele llamar también ***Comentarios de bloque.***
- ***Comentarios de documentación "Javadoc"***: estos comentarios se colocan entre los delimitadores /** … */, agrupan varias líneas, y cada línea irá precedida por un *, y lo más importante es que éstos ***deben colocarse antes de la declaración de una clase, un campo, un método o un constructor***.

Los comentarios de ***Javadoc*** están formados por dos partes:

- Una descripción
- Bloque de ***tags***

3.1.1- Uso de etiquetas de documentación

Se pueden usar tags para documentar aspectos concretos, como la versión de la clase, el autor, los parámetros utilizados, o los valores devueltos.

Las etiquetas de ***Javadoc*** van precedidas por **@.**

Las etiquetas más usadas, son las siguientes:

Etiqueta	Descripción
@param	Describe un parámetro de un método o constructor.
@return	Explica lo que devuelve el método.
@throws	Descripción de la excepción que puede propagar. Habrá una etiqueta @throws por cada tipo de excepción.
@see	Referencia a otra clase, ya sea del mismo proyecto o de otro. Por ejemplo: • @see cadena • @see paquete.clase#miembro • @see enlace
@since	Indica desde qué versión está disponible el elemento.
@deprecated	Marca el método o clase como obsoleto. Solo se mantiene por compatibilidad.
@author	Identifica al autor del código.
@version	Versión de la clase. Solo para clases.
@link	Inserta un enlace a otro elemento en la documentación.
@code	Para incluir fragmentos de código en la descripción.

Ejemplo:

```java
/**
 * Clase Empleado que representa la información y las operaciones de un empleado.
 * <p>
 * Esta clase proporciona métodos para establecer y obtener los detalles de un empleado
 * como su nombre, identificación y salario.
 * </p>
 *
 * @author Nombre del autor
 * @version 1.0
 */
public class Empleado {

    /**
     * Identificación única del empleado, del tipo 'int'.
     */
    private int id;

    /**
     * Nombre del empleado, del tipo 'String'.
     */
    private String nombre;

    /**
     * Salario del empleado, del tipo 'double'.
     */
    private double salario;

    /**
     * Constructor para crear un nuevo empleado con id, nombre y salario.
     *
     * @param id     Identificación del empleado, del tipo 'int'.
     * @param nombre Nombre del empleado, del tipo 'String'.
     * @param salario Salario del empleado, del tipo 'double'.
     */
    public Empleado(int id, String nombre, double salario) {
        this.id = id;
        this.nombre = nombre;
        this.salario = salario;
    }

    /**
     * Obtiene la identificación del empleado.
     *
     * @return El ID del empleado, del tipo 'int'.
     */
    public int getId() {
        return id;
    }

    /**
     * Establece la identificación del empleado.
     *
     * @param id El ID del empleado, del tipo 'int'.
```

```
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * Obtiene el nombre del empleado.
     *
     * @return El nombre del empleado, del tipo 'String'.
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Establece el nombre del empleado.
     *
     * @param nombre El nombre del empleado, del tipo 'String'.
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Obtiene el salario del empleado.
     *
     * @return El salario del empleado, del tipo 'double'.
     */
    public double getSalario() {
        return salario;
    }

    /**
     * Establece el salario del empleado.
     *
     * @param salario El salario del empleado, del tipo 'double'.
     */
    public void setSalario(double salario) {
        this.salario = salario;
    }

    /**
     * Método para imprimir los detalles del empleado.
     *
     * @return Detalles del empleado en formato String, del tipo
'String'.
     */
    @Override
    public String toString() {
        return "Empleado{" +
                "id=" + id +
                ", nombre='" + nombre + '\'' +
                ", salario=" + salario +
                '}';
    }
}
```

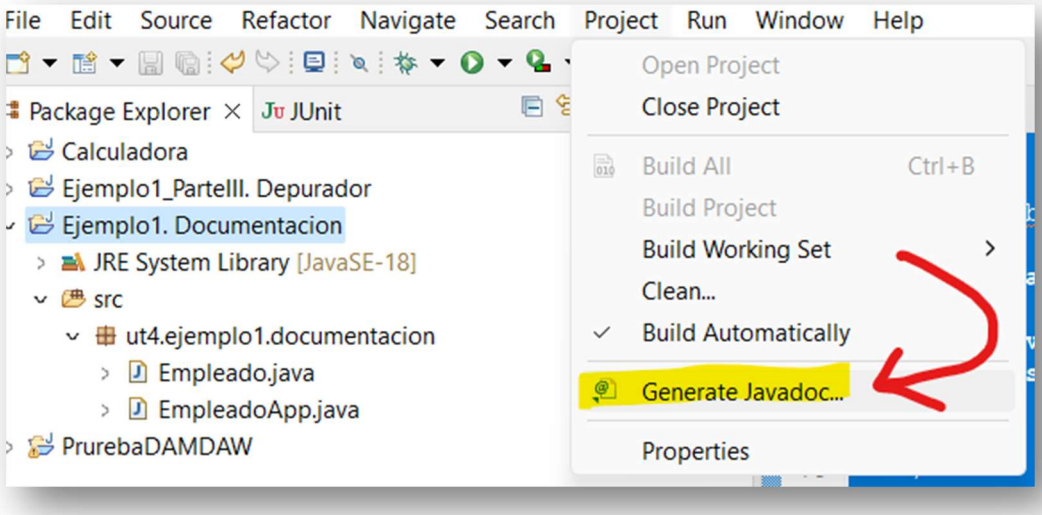
Mira el ejemplo que se proporciona en el aula.

3.1.2- Generar la documentación

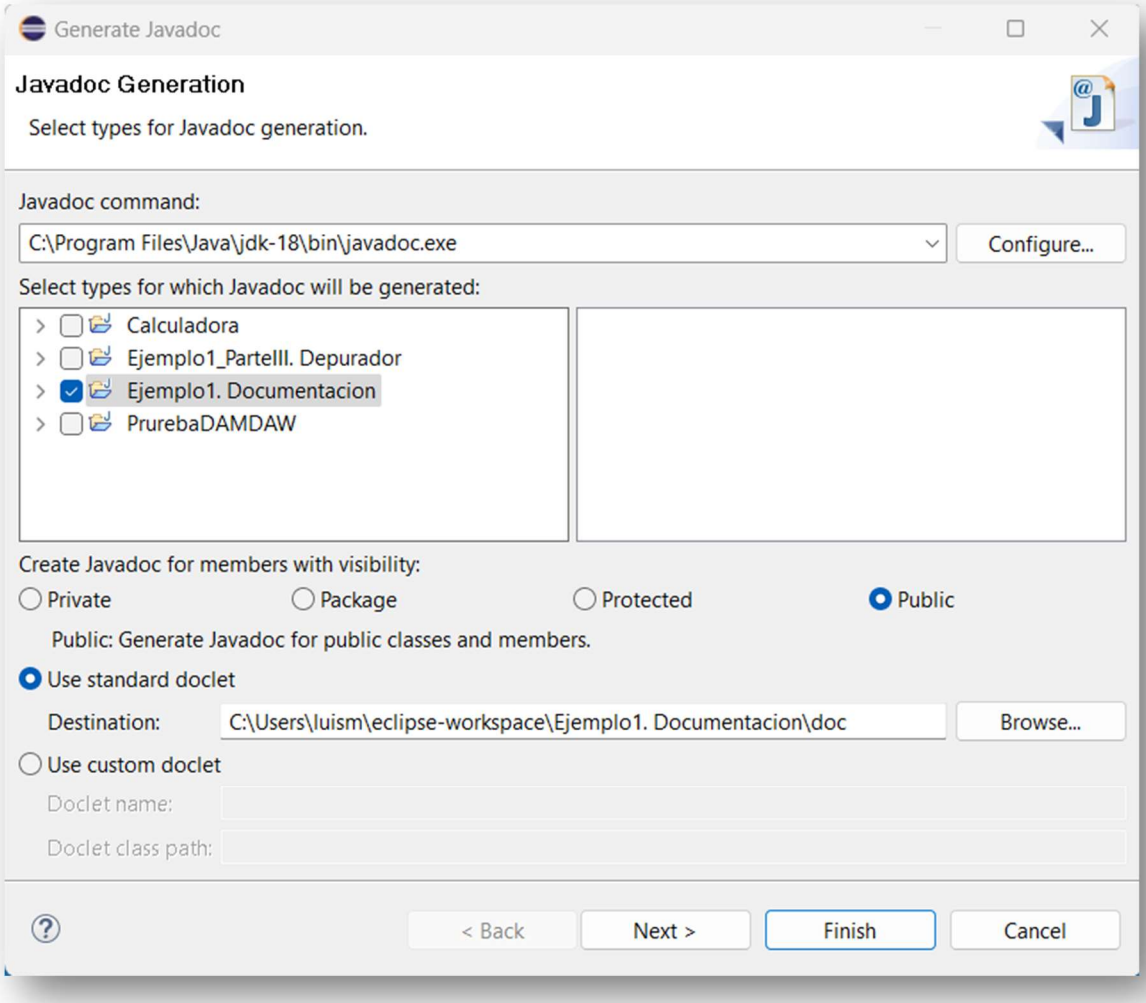
La mayor parte de los entornos de desarrollo incluyen un botón o un enlace para configurar y ejecutar *Javadoc.*

Para ejecutar *Javadoc* desde eclipse, seguiremos los siguientes pasos:

1) Se abre el menú ***Project*** *y se elige* ***Generate Javadoc***:



2) Después de hacer clic, nos aparecerá la siguiente ventana emergente:

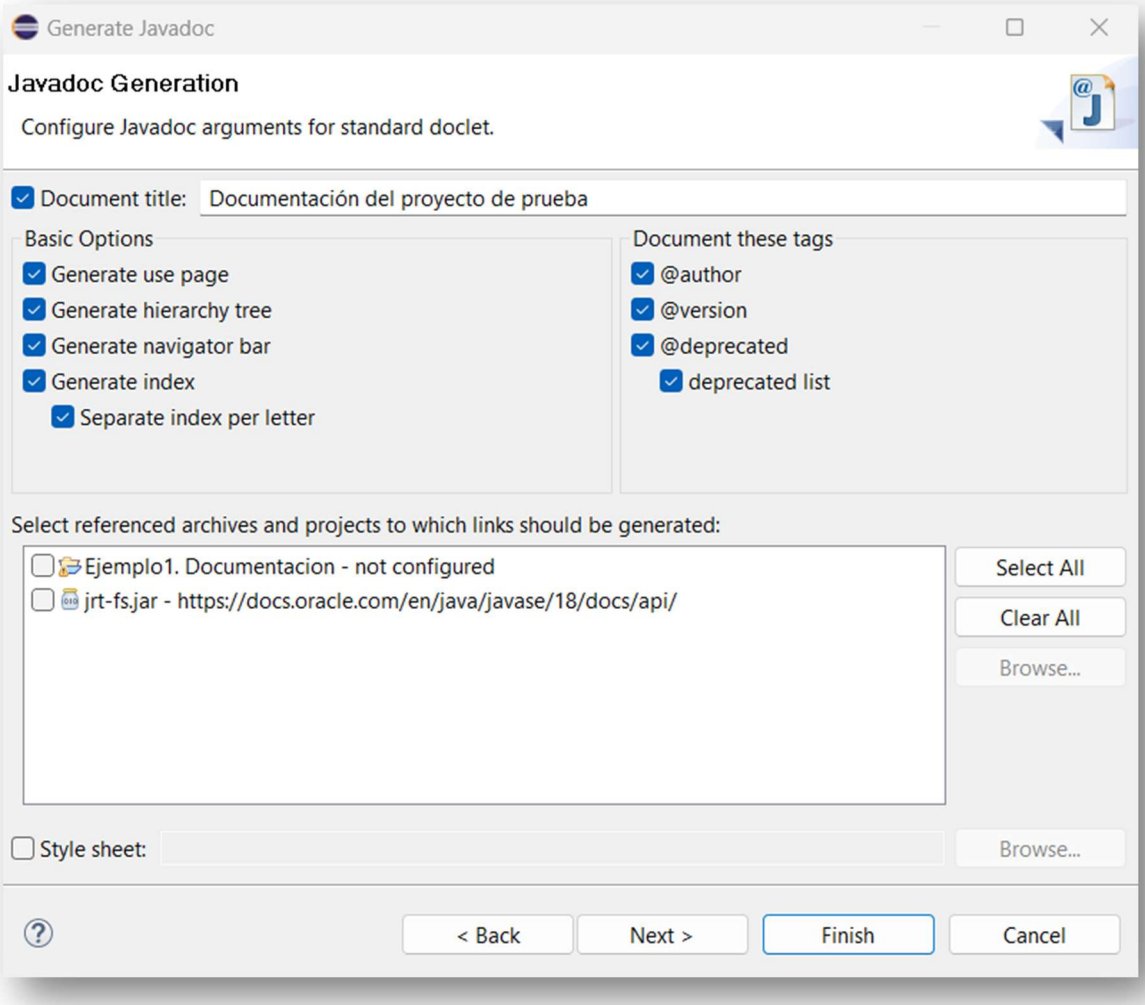


En la ventana se muestra la siguiente información:

- **Javadoc command:** Ruta al ejecutable javadoc.exe del JDK instalado, que se usará para generar la documentación. En este caso, es C:\Program Files\Java\jdk-18\bin\javadoc.exe.
- **Select types for Javadoc generation**: Una lista de paquetes y clases de Java disponibles en el proyecto actual para los cuales se puede generar la documentación. Hay varias clases y paquetes listados, pero no están seleccionados en la captura.
- **Create Javadoc for members with visibility**: Opciones para seleccionar qué miembros (métodos, variables, etc.) de las clases se incluirán en la documentación basados en su visibilidad: Private, Package, Protected, Public. La opción "Public" está seleccionada, lo que significa que solo los miembros públicos serán documentados.

- **Public: Generate Javadoc for public classes and members**: Un radio botón que está seleccionado para indicar que se generará Javadoc solo para las clases y miembros públicos.
- **Use standard doclet**: Esta opción está seleccionada e indica que se usará el doclet estándar para generar la documentación. Un doclet es un programa que especifica la estructura y contenido de la documentación.
- **Destination**: La ruta del directorio donde se guardarán los archivos de documentación generados. Aquí se muestra C:\Users\luism\eclipse-workspace\Ejemplo1. Documentacion\doc.
- **Use custom doclet**: Opciones para usar un doclet personalizado, no están seleccionadas.
- **Doclet name**: Un campo para ingresar el nombre del doclet personalizado si se seleccionara la opción anterior.
- **Doclet class path:** Un campo para ingresar la ruta de clases del doclet personalizado si se seleccionara la opción anterior.

3) Hacemos clic en “siguiente” y nos aparecerá la siguiente pantalla:

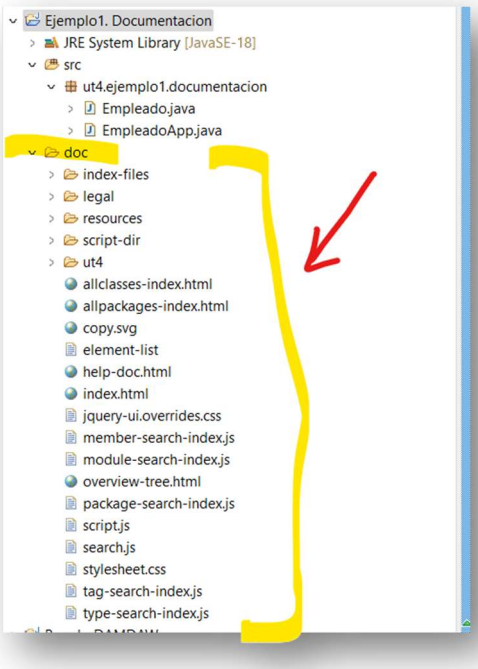


Esta ventana específica permite configurar opciones adicionales para la generación del *Javadoc*. Los elementos que se muestran son:

- **Document title:** Campo vacío donde se puede especificar el título de la documentación que se generará.
- Basic Options:
- **Generate use page:** Si está marcado, se generarán páginas que muestran dónde se utilizan las clases y miembros del proyecto.
- **Generate hierarchy tree:** Si está marcado, se generará un árbol de herencia para las clases e interfaces.
- **Generate navigator bar:** Si está marcado, se incluirá una barra de navegación en la documentación para facilitar el movimiento entre las páginas del Javadoc.

- **Generate index:** Si está marcado, se creará un índice para la documentación.
- **Separate index per letter:** Si está marcado, el índice se dividirá en secciones separadas por la letra inicial.
- **Document these tags:** Especifica qué etiquetas Javadoc se deben documentar.
    - @author
    - @version
    - @deprecated y una opción para incluir una lista de elementos deprecados.
- **Select referenced archives and projects to which links should be generated**: Permite seleccionar archivos de referencia y proyectos para generar enlaces en la documentación. Muestra un archivo no configurado llamado Ejemplo1. Documentacion y otro archivo llamado jrt-fs.jar con un enlace a la documentación de la API de Java en docs.oracle.com.
- **Style sheet:** Campo vacío donde se puede especificar la hoja de estilos CSS para personalizar la apariencia de la documentación generada.

4) Al hacer clic en “Finish”, se crea la documentación dentro del proyecto:



Ejercicio de clase:

Ahora genera tú la documentación para el Ejemplo 1 de clase.

4.- Referencias bibliográficas

- Moreno Pérez, J.C. *Entornos de desarrollo.* Editorial Síntesis.
- Ramos Martín, A. & Ramos Martín, M.J. *Entornos de desarrollo*. Grupo editorial Garceta.