SANTA ANA
Y SAN RAFAEL
Madrid

COLEGIOS
MARIANISTAS

OPTIMIZACIÓN Y DOCUMENTACIÓN

UD 4 Presentación

ENTORNOS DE DESARROLLO – 1ero DAM

OPTIMIZACIÓN Y DOCUMENTACIÓN

OBJETIVOS

- ✓ Aprender a comentar y **documentar** nuestro software.
- ✓ Comprender el concepto de **refactorización** y sus implementaciones más usuales.
- ✓ Aprender a reconocer **antipatrones** en nuestro software.
- ✓ Conocer el uso de un **control de versiones** y su aplicación en el desarrollo colaborativo.

OPTIMIZACIÓN Y DOCUMENTACIÓN

DOCUMENTACIÓN

- ✓ En todo proyecto software es fundamental la **documentación**, no solo para el usuario final, sino para los propios desarrolladores.
- ✓ La documentación es vital para la **fase de mantenimiento** del software.
- ✓ La documentación del código puede explicar, por ejemplo, el patrón o algoritmo utilizado.
- ✓ Los **comentarios de documentación** del código, aportan información sobre las clases y métodos, información específica y estructurada sobre su función y sus datos de E/S.
- ✓ Los comentarios en el código, sirven a los IDE's para generar **documentación de manera automática**.

OPTIMIZACIÓN Y DOCUMENTACIÓN

JAVADOC

- ✓ **Utilidad** de **Oracle** para la generación de documentación en formato HTML a partir de código fuente Java.
- ✓ **Javadoc** es el **estándar para documentar clases de Java**, ya que genera una documentación igual a la API de Java. La mayoría de los IDEs utilizan **javadoc** para generar de forma automática documentación de clases.
- ✓ **Eclipse** nos permite generar automáticamente la documentación de una clase, pero para eso debemos **introducir en el código, los comentarios** necesarios para que se genere esa documentación:
  - 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.

OPTIMIZACIÓN Y DOCUMENTACIÓN

CREACIÓN DEL JAVADOC

Para generar la documentación de un proyecto automáticamente hemos de seguir unas normas a la hora de realizar los comentarios dentro del mismo.

- ✓ a) Todo lo que queramos que se incluya en la documentación de **javadoc**, ha de ponerse en el .java entre símbolos de comentario que han de empezar con una barra y doble asterisco, y terminar con un asterisco y barra simple.
- ✓ b) Dónde colocamos el comentario, 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 un constructor o método se considerará un comentario de ese constructor o método.
- ✓ c) Para alimentar **javadoc** se usan ciertas palabras reservadas (**tags**) precedidas por el carácter "@", dentro de los símbolos de comentario **javadoc**. **Si no existe al menos una línea que comience con @ no se reconocerá el comentario para la documentación de la clase.**

OPTIMIZACIÓN Y DOCUMENTACIÓN

TAGS DE JAVADOC

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre autor o autores
@deprecated	Indica que el método o clase es obsoleto (propio de versiones anteriores) y que no se recomienda su uso.	Descripción
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	Nombre de parámetro y descripción
@return	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor de retorno
@see	Asocia con otro método o clase.	Referencia cruzada referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()).
@version	Versión del método o clase.	Versión
@throws	Descripción de la excepción que puede propagar. Habrá una etiqueta throws por cada tipo de excepción.	

OPTIMIZACIÓN Y DOCUMENTACIÓN

TAGS DE JAVADOC

- ✓ Las etiquetas **@author** y **@version** se usan para documentar **clases** e **interfaces**. Por tanto no son válidas en cabecera de constructores ni métodos.

  Una **interfaz** en Java es una colección de métodos abstractos y propiedades constantes.

- ✓ La etiqueta **@param** se usa para documentar constructores y métodos. La etiqueta **@return** se usa solo en métodos que devuelven un valor.
- ✓ Dentro de los comentarios se admiten **etiquetas HTML**, por ejemplo con **@see** se puede referenciar una página web como link para recomendar su visita de cara a ampliar información.

OPTIMIZACIÓN Y DOCUMENTACIÓN

EJEMPLO CLASE COMENTARIOS JAVADOC

```
package prueba01;

import java.util.ArrayList;
import java.util.Random;

/**
 * Esta clase define objetos que contienen tantos enteros aleatorios
 * entre 0 y 1000 como se le definen al crear un objeto
 * @author: Mario R. Rancel
 * @version: 22/09/2016/A
 * @see <a href = "http://www.aprenderaprogramar.com"/> aprenderaprogramar.com con Didactica en programacion </a>
*/
```

```
public class SerieDeAleatoriosD {
  //Campos de la clase

  private ArrayList<Integer> serieAleatoria;

  /**
   * Constructor para la serie de numeros aleatorios
   * @param numeroItems El parametro numeroItems define el numero de elementos que
   * va a tener la serie aleatoria
   * @see #getNumeroItems() llamamos a getNumeroItems() para recuperar el valor
   */
  public SerieDeAleatoriosD (int numeroItems) {
     serieAleatoria = new ArrayList<Integer> ();
     for (int i=0; i<numeroItems; i++) {  serieAleatoria.add(0);  }
     System.out.println ("Serie inicializada. El número de elementos en la serie es: " + getNumeroItems() );
   } //Cierre del constructor
```

OPTIMIZACIÓN Y DOCUMENTACIÓN

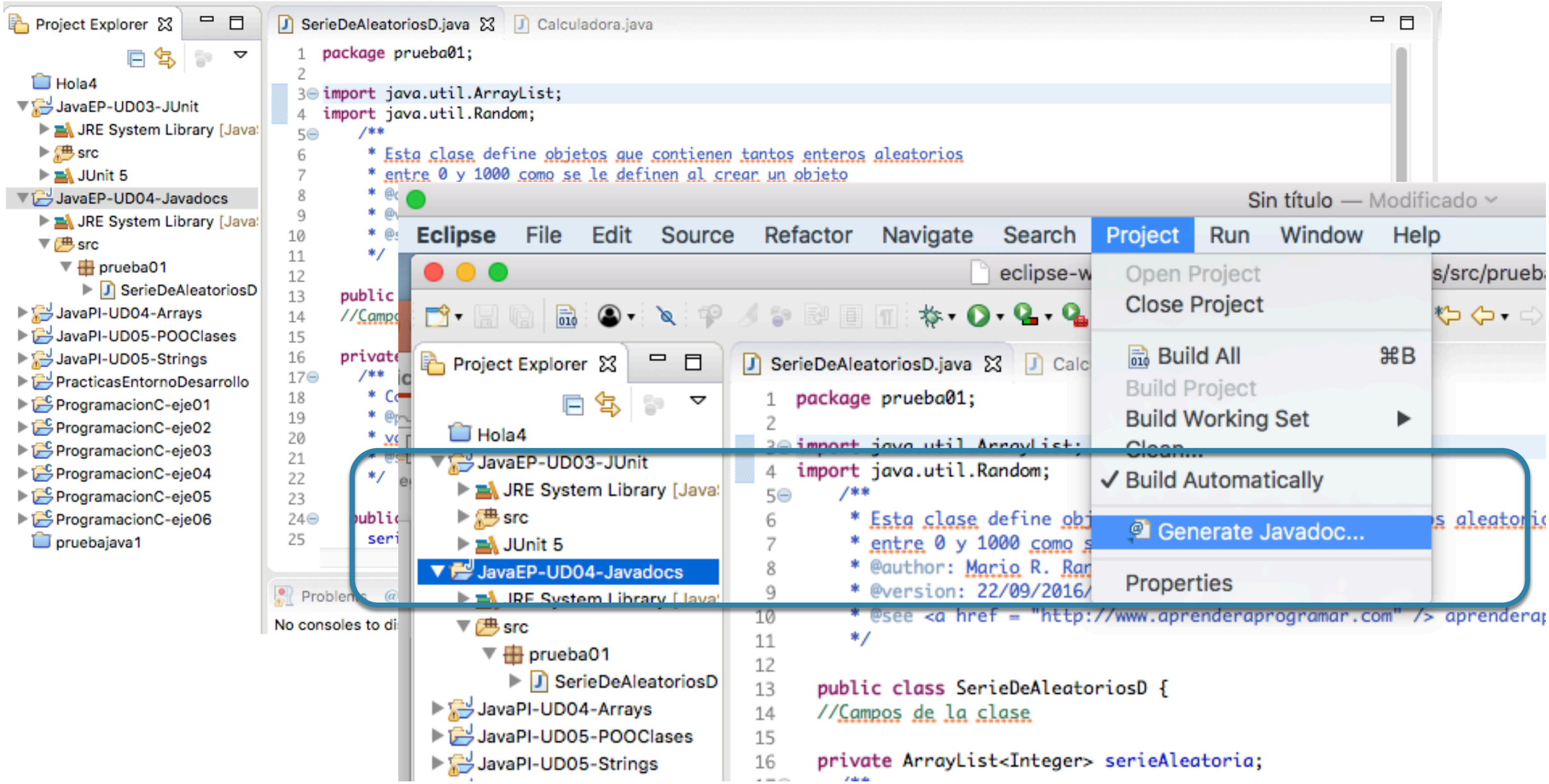
EJEMPLO CLASE COMENTARIOS JAVADOC (CONTINUACIÓN)

```
/**
 * Metodo que devuelve el numero de items (numeros aleatorios) existentes en la serie
 * @return <ul>
 *                  <li> El numero de items (numeros aleatorios) de que consta la serie
 *         </ul>
 */
 public int getNumeroItems() { return serieAleatoria.size(); }


/**
 * Metodo que genera la serie de numeros aleatorios
 */
 public void generarSerieDeAleatorios () {
     Random numAleatorio;
     numAleatorio = new Random ();
     for (int i=0; i < serieAleatoria.size(); i++) {
       serieAleatoria.set(i,       numAleatorio.nextInt(1000) );
     }
     System.out.print ("Serie generada! ");
   } //Cierre del metodo


} //Cierre de la clase y del ejemplo
```

IMPORTANTE: NO poner acentos ni eñes.

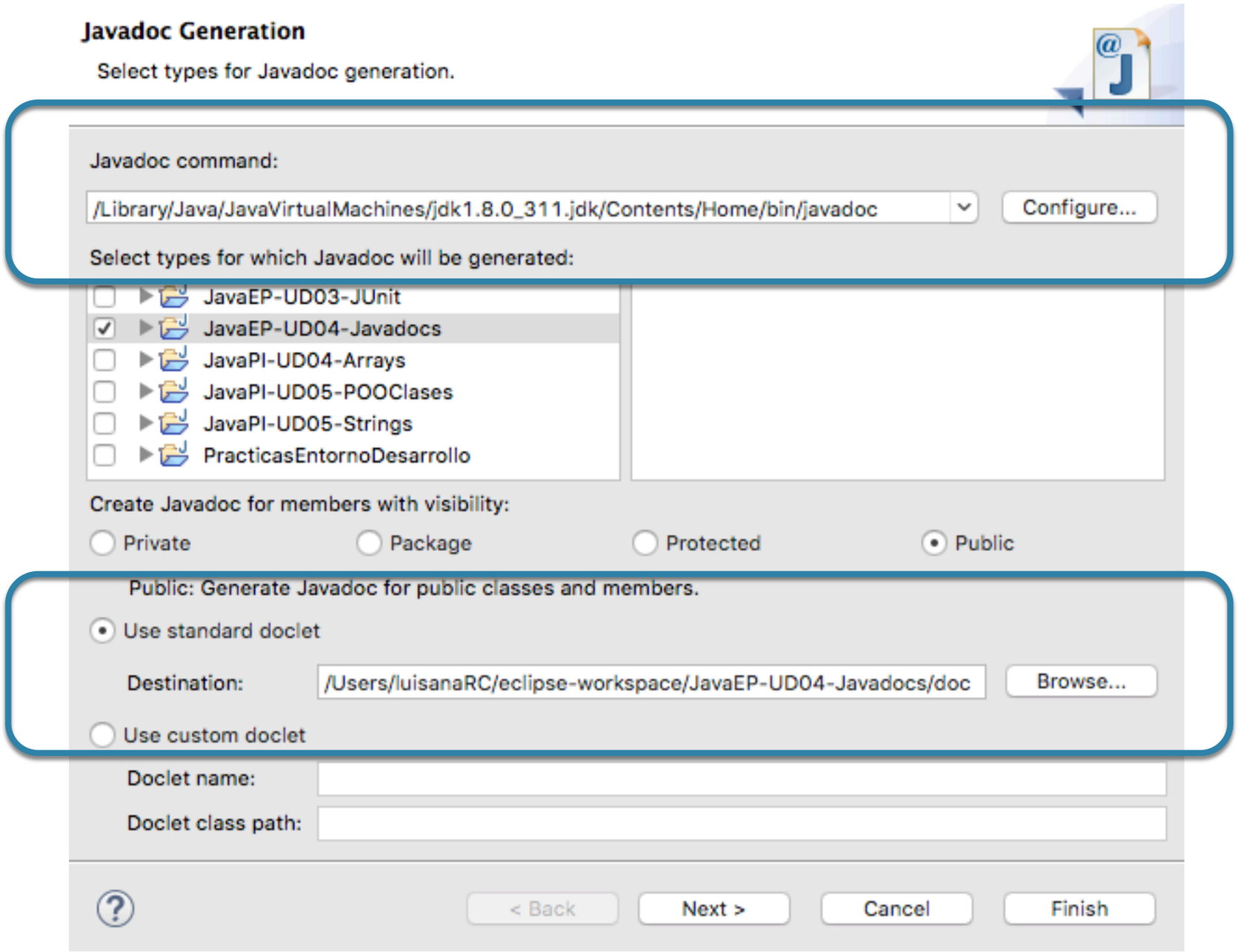
OPTIMIZACIÓN Y DOCUMENTACIÓN

GENERAR JAVADOC EN ECLIPSE



OPTIMIZACIÓN Y DOCUMENTACIÓN

GENERAR JAVADOC EN ECLIPSE



OPTIMIZACIÓN Y DOCUMENTACIÓN

GENERAR JAVADOC EN ECLIPSE

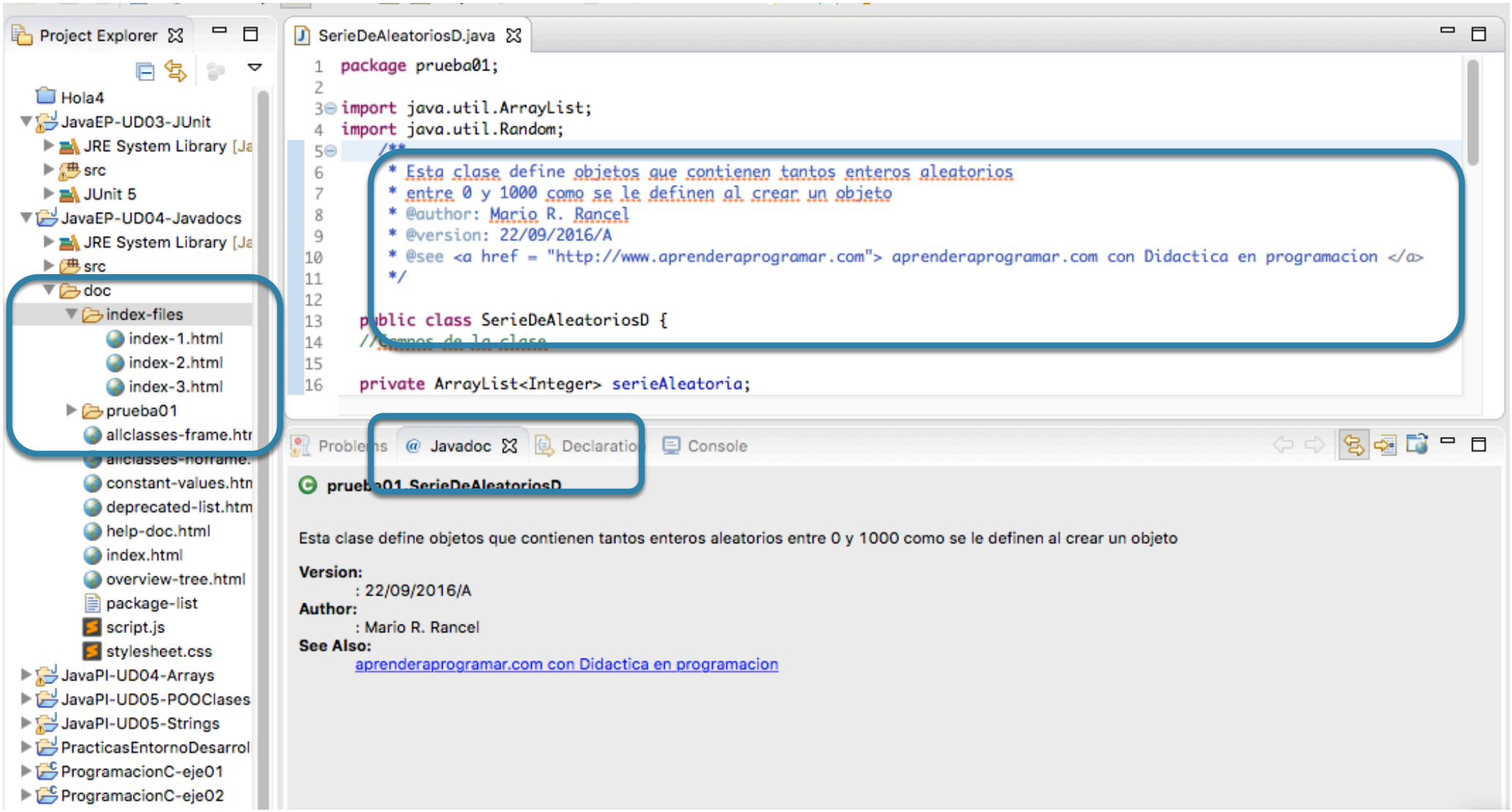
Salida por la **consola**:

```
Loading source files for package prueba01...
Constructing Javadoc information...
Standard Doclet version 1.8.0_311
Building tree for all the packages and classes...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/SerieDeAleatoriosD.html...
/Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/src/prueba01/SerieDeAleatoriosD.java:10: error: self-closing element not
        * @see <a href = "http://www.aprenderaprogramar.com" /> aprenderaprogramar.com - Didáctica en programación </a>
               ^
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/package-frame.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/package-summary.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/package-tree.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/constant-values.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/class-use/SerieDeAleatoriosD.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/prueba01/package-use.html...
Building index for all the packages and classes...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/overview-tree.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/index-files/index-1.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/index-files/index-2.html...
Generating /Users/luisanaRC/eclipse-workspace/JavaEP-UD04-Javadocs/doc/index-files/index-3.html...
```

Resolvemos el error y volvemos a generar la documentación

OPTIMIZACIÓN Y DOCUMENTACIÓN

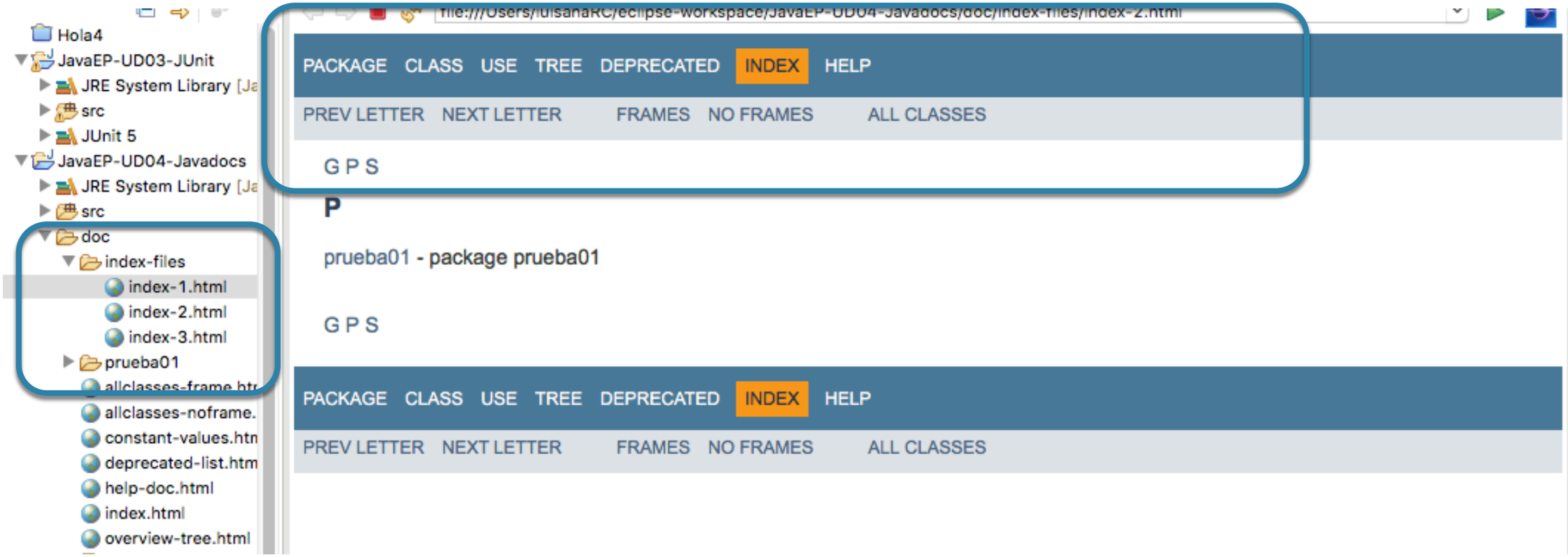
GENERAR JAVADOC EN ECLIPSE



En la pestaña “Javadoc” podemos ver cómo quedará en la documentación, los comentarios con tags, del código

OPTIMIZACIÓN Y DOCUMENTACIÓN

VISUALIZAR LA DOCUMENTACIÓN



Desde la carpeta Windows, pulsar un *html y se abre una ventana de explorador. La documentación está escrita en **HTML**

OPTIMIZACIÓN Y DOCUMENTACIÓN

REFACTORIZAR

- ✓ “Consiste en realizar una transformación al software preservando su comportamiento, y modificando su estructura interna para mejorarlo” (Opdyke, 1992).
- ✓ Informalmente es lo que se conoce como “**limpieza de código**”.
- ✓ La piedra angular de la refactorización se encuentra en: “No cambiar la funcionalidad del código ni el comportamiento del programa, el programa deberá comportarse de la misma manera y forma antes y después de la efectuar la refactorización”.

OPTIMIZACIÓN Y DOCUMENTACIÓN

MALOS OLORES

- ✓ Los "**malos olores**" son una relación de **malas prácticas de desarrollo**, indicadores de que nuestro código podría necesitar ser **refactorizado**.
- ✓ Son **indicadores**, no quiere decir que necesariamente sean un problema.
  - **Método largo**. Los programas que viven más y mejor son aquellos con métodos cortos, que son más reutilizables y aportan mayor semántica.
  - **Clase grande**. Clases que hacen demasiado y por lo general con una baja cohesión, son muy vulnerables al cambio.

OPTIMIZACIÓN Y DOCUMENTACIÓN

MALOS OLORES

- **Lista de parámetros larga**. Los métodos con muchos parámetros elevan el acoplamiento, son difíciles de comprender y cambian con frecuencia.
- **Obsesión primitiva**. Uso excesivo de tipos primitivos. A veces es mejor modelar objetos, incluso para pequeñas tareas como dinero, rangos, números de teléfono …
- **Clase de datos**. Clases que solo tienen atributos y métodos tipo *get* y *set*. Las clases siempre tienen que tener algún comportamiento no trivial.
- **Atributo temporal**. Algunos objetos tienen atributos que se usan solo en ciertas circunstancias. Lo esperado es que un objeto use todas sus variables.