Índice

1. Generación automática de documentación (javadoc) 1
1.1 Introducción 1
1.2 La herramienta Javadoc 1
1.3 Comentando el código Java 1
2 Guía de estilo de Java 5
2.1 Clases 6
2.2 Constructores 6
2.3 Métodos 6
2.4 Atributos 7
2.5 Comentarios a bloques de código 7
3 Desarrollar y ejecutar una aplicación sencilla 7
4 Cuestiones 7

1. Generación automática de documentación (javadoc)

1.1 Introducción

Javadoc es una herramienta para generar documentación de código Java a partir de comentarios (escritos según ciertas reglas) que se intercalan en el código fuente. La documentación se genera en formato HTML y, por tanto, se puede consultar fácilmente con un navegador.
Un ejemplo muy ilustrativo de documentación generada con javadoc es la especificación de la API (=Application Program Interface) para la plataforma J2SE (http://java.sun.com/j2se/1.4.2/docs/api/index.html).
Una explicación exhaustiva de la herramienta javadoc y de la forma de intercalar los comentarios en el código fuente para que puedan ser interpretados por la herramienta la podemos encontrar en la Javadoc Home Page (http://java.sun.com/j2se/javadoc/index.jsp).
En esta práctica se van a presentar sólo los aspectos más relevantes.

1.2 La herramienta Javadoc

La sintaxis con la que tiene que ser invocada la herramienta es:
javadoc [opciones] Clase
Aunque existe una gran variedad de opciones, estas son algunas de las que nos pueden resultar útiles en el desarrollo de las prácticas:

- javadoc -public Clase : muestra sólo las clases y los miembros públicos
- javadoc -protected Clase : muestra las clases y los miembros protegidos (protected) y públicos
- javadoc -package Clase : muestra las clases y los miembros con acceso de paquete, los protegidos y los públicos
- javadoc -private Clase : muestra todas las clases y todos los miembros, incluyendo los privados

1.3 Comentando el código Java

Se pueden escribir comentarios para documentación para cualquier paquete, clase, interfaz, constructor, método y atributo. Los comentarios a partir de los cuales se genera la documentación son llamados informalmente "comentarios Javadoc" ("Javadoc comments"). Un comentario Javadoc consta de un texto escrito con una cierta sintaxis entre los caracteres **/**** (que marca el principio del comentario) y ***/** (que

marca el final).
El texto del comentario puede extenderse a lo largo de varias líneas.
Ejemplo:

```
/**
 * Este es el típico formato de un comentario simple
 * que se extiende a lo largo de dos líneas
 */
```

Aunque para ahorrar espacio se puede poner un comentario en una sola línea

```
/** Este comentario ocupa sólo una línea */
```

Ubicación de los comentarios
Los comentarios para la documentación son reconocidos como tales sólo cuando se escriben inmediatamente delante de la declaración de una clase, interfaz, constructor, método o atributo. Los comentarios que se escriben en el cuerpo de un método no sirven para generar documentación. La herramienta Javadoc sólo reconoce un comentario para documentación para cada declaración.
Un error muy habitual es escribir una sentencia import entre el comentario de la clase y la declaración de la misma: si se hace así la herramienta Javadoc ignora el comentario y no genera la documentación para la clase.
Ejemplo:

```
/**
 * Este es el comentario para la documentación para la clase Whatever.
 */
import com.sun;   // ERROR - Es importante no intercalar una sentencia import aquí
public class Whatever
{ ...
```

Un comentario Javadoc se compone de una descripción principal seguida de una sección de etiquetas
Un comentario está compuesto por una descripción principal seguida de una sección de etiquetas (tags). La descripción principal empieza después del marcador de principio de comentario /** y sigue hasta la sección de etiquetas. El principio de esta sección está marcado por el primer carácter @. La descripción principal no puede continuar una vez iniciada la sección de etiquetas. Es posible tener un comentario sólo con sección de etiquetas, sin descripción principal.
Ejemplo:

```
/**
 * Esta frase sería la descripción principal del elemento cuya
 * declaración se iniciaría justo después de este comentario
 * @author laboratorio de LP
 */
```

Los comentarios se escriben en HTML
El texto de los comentarios se puede escribir utilizando la potencia del lenguaje HTML. Así, el tipo de letra negrita se especifica mediante la etiqueta <b>
Ejemplo:

```
/**
 * Esto es un comentario <b>Javadoc</b>
 * @author laboratorio de LP
 */
```

En cualquier caso no es imprescindible saber HTML para poder escribir y generar documentación Javadoc sencilla.

Primera frase de un comentario
La primera frase de un comentario Javadoc siempre debería ser una frase-resumen que contenga una descripción concisa pero completa de la entidad declarada. Esta frase acaba en el primer punto que es seguido por un espacio en blanco, un tabulador o una marca de final de línea; también se interpreta que termina con la primera aparición del símbolo @ que marca el principio de la sección de etiquetas. La herramienta Javadoc copia esta primera frase en el resumen de la entidad, en la parte superior de la página HTML de la documentación que genera.

Declaración con múltiples atributos
Java permite declarar más de un atributo en una sola sentencia, pero esta sentencia puede tener sólo un comentario para documentación que se copia para todos los atributos declarados en la sentencia. Por tanto, si se desea tener comentarios para la documentación individuales para cada atributo, se debe declarar cada campo en una sentencia diferente.
Por ejemplo, el siguiente comentario para documentación no tiene sentido para una declaración de atributos en una sola sentencia:

```
/**
 * Las distancias horizontal y vertical del punto (x,y)
 */
public int x, y;      // Evitar esto!!
```

La herramienta Javadoc generaría la siguiente documentación:

```
public int x
        The horizontal and vertical distances of point (x,y)
public int y
        The horizontal and vertical distances of point (x,y)
```

Tendría más sentido escribir dos sentencias de declaración (una para cada uno de los atributos), acompañadas de sus correspondientes comentarios para documentación (habría que ajustar el texto también).

Etiquetas Javadoc
La herramienta Javadoc "escanea" (to parse) las etiquetas cuando están incrustadas en un comentario Javadoc. Estas etiquetas para documentación permiten generar una API completa a partir del código fuente con los comentarios. Las etiquetas comienzan con el símbolo "at" (@) y son sensibles a mayúsculas-minúsculas ("case sensitive").
Una etiqueta se sitúa siempre al principio de una línea, o al menos sólo precedida por espacio(s) y asterisco(s), o será tratada como texto normal por la herramienta Javadoc. Por convenio, si una etiqueta se utiliza más de una vez, todas las apariciones se deben escribir seguidas; es decir, no se recomienda intercalar etiquetas.

Hay dos tipos de etiquetas:

- Etiquetas de bloque: sólo se pueden utilizar en la sección de etiquetas que sigue a la descripción principal. Son de la forma: @etiqueta
- Etiquetas inline: se pueden utilizar tanto en la descripción principal como en la sección de etiquetas. Son de la forma: {@tag}, es decir, se escriben entre los símbolos de llaves.

A continuación se presentan algunas etiquetas

@author nombre_del_autor
Para especificar el autor de la entidad. Si se desea que este comentario aparezca en la documentación (y no sólo al nivel del código fuente), en el Javadoc se tiene que utilizar explícitamente la opción -author. Esta etiqueta sólo es válida en los comentarios para documentación de paquetes y clases.

{@link package.class#member etiqueta}
Crea un enlace con el texto "etiqueta" que apunta a la documentación del paquete, la clase o el miembro especificado.

@param nombre_del_parámetro descripción
Añade un parámetro en la sección "Parámetros". La descripción se puede escribir en más de una línea. Esta etiqueta sólo es válida en los comentarios para documentación de métodos y constructores.

@return descripción
Añade una sección "Returns" con el texto "descripción". Este texto debería describir el tipo devuelto y el rango de posibles valores. Esta etiqueta sólo es válida en los comentarios para documentación de métodos.

@see referencia
Añade una cabecera "See Also" con un enlace que apunta a una referencia. Un comentario para documentación puede contener más de una de estas etiquetas; todos los enlaces de este tipo se agrupan bajo la misma cabecera.

@since texto
Añade en la documentación una cabecera "Since" con el "texto" especificado. Sirve para especificar la versión del software a partir de la cual ha estado disponible la entidad que se declara.

@throws nombre_de_la_clase descripción
@exception nombre_de_la_clase descripción
Para especificar el nombre de la clase (y acompañarlo con una descripción) de la clase lanzada cuando ocurre una excepción. Esta etiqueta sólo es válida en los comentarios para documentación de métodos y constructores.

{@value}
Cuando se usa en un comentario para documentación de un atributo estático, se muestra el valor de la constante. Este valor se muestra en la página de Valores de Atributos Constantes ("the Constant Field Values page"). Esta etiqueta sólo es válida en los comentarios para documentación de atributos.

A modo de resumen, se muestran a continuación las etiquetas descritas (que han sido sólo una parte de todas las existentes) agrupadas según las entidades en las que se pueden utilizar:

- Etiquetas para la documentación de Paquetes

@see @since @author {@link}

- Etiquetas para la documentación de Clases e Interfaces

@see @since @author {@link}

Ejemplo:
```
/**
 * A class representing a window on the screen.
 * For example:
 * <pre>
 *    Window win = new Window(parent);
 *    win.show();
 * </pre>
 *
 * @author  Sami Shaio
 * @see     java.awt.BaseWindow
 * @see     java.awt.Button
 */
class Window extends BaseWindow
{ ...
```

- Etiquetas para la documentación de Atributos

@see @since {@link} {@value}
Ejemplo:
```
/**
 * The X-coordinate of the component.
 *
 * @see #getLocation()
 */
int x = 1263732;
```

- Etiquetas para la documentación de Constructores y Métodos

@see @since @param @return(métodos; no constructores) @throws @exception {@link}
Ejemplo:
```
/**
 * Returns the character at the specified index. An index
 * ranges from <code>0</code> to <code>length() - 1</code>.
 *
 * @param     index  the index of the desired character.
 * @return    the desired character.
 * @exception  StringIndexOutOfRangeException
 *             if the index is not in the range <code>0</code>
 *             to <code>length()-1</code>.
 * @see       java.lang.Character#charValue()
 */
public char charAt(int index)
{ ...
```

2 Guía de estilo de Java

La guía de estilo recomienda una serie de convenciones para la escritura de programas en Java. Se trata de un asunto importante ya que:

- Durante su vida, el software está sujeto a mantenimiento.
- En entornos reales casi nunca una sola persona mantiene el software.

- Las convenciones mejoran la legibilidad del código.
- El software es un producto y como tal debe cumplir unos mínimos de calidad.

2.1 Clases

Nombre de una clase
Una clase es la abstracción de una entidad y, por tanto, el nombre que se le asigne debe estar relacionados con la entidad a la que representa.
En la API (Application Program Interface) de Java nos podemos encontrar nombres de clases del estilo de: Vector, Array, System, URL, Socket, SocketSecurityException, etc.
Como se puede ver el nombre de una clase comienza con mayúscula y, si está formado por más de una palabra, el inicio de cada una de ellas está en mayúscula.

Estructura de una clase

Parte	Descripción
Sentencia class	Define que se trata de una clase, por supuesto irá precedida de los modificadores pertinentes
Variables de clase static	Primero las declaradas como public, seguidas de las protected, las de acceso de package y finalmente las private
Variables de instancia	Primero las declaradas como public, seguidas de las protected, las de acceso de package y finalmente las private
Constructores	
Métodos	Agrupados por funcionalidad, si un método llama a otro lo mejor es que estén lo más cerca posible para que sea más legible

2.2 Constructores

El nombre del constructor ha de ser el mismo que el nombre de la clase.

2.3 Métodos

Los métodos representan una acción. Cuando un objeto recibe un mensaje, se le está pidiendo que realice una determinada tarea. Por lo tanto, usualmente el nombre de los métodos comienza por (o al menos incluye) un verbo.
En la API (Application Program Interface) de Java nos podemos encontrar nombres de métodos del estilo de (métodos correspondientes a la clase URL): getContent(), getDefaultPort(), getProtocol(), openConnection(), set(), etc.
Como se puede observar la primera letra es minúscula y, si el nombre está formado por más de una palabra, el inicio de las siguientes palabras comienza con mayúscula.

2.4 Atributos

Los atributos almacenan información relevante para la clase o para los objetos; por tanto, el nombre que se asigna a un atributo debe ser representativo de la información que almacena. De otro modo será muy difícil saber que tipo de información representa. La primera letra del nombre de un atributo es minúscula.

2.5 Comentarios a bloques de código

Se pueden realizar comentarios a líneas o bloques de código siempre que esas líneas tengan una cierta complejidad.
Por ejemplo no se deben utilizar este tipo de comentarios:

```
// Asignamos a representaPuesto el String ”DIRECCION”
representaPuesto = ”DIRECCION” ;
```

este otro seguramente tiene más sentido:

```
/* Puesto que se tiene puesto representado mediante un int y se desea una
representación en forma de String hay que realizar una trans formación */
switch ( puesto )
{ …
```

3 Desarrollar y ejecutar una aplicación sencilla

Se pide:

- Completar los archivos de código LanzaPrac1.java y ComplexNumber.java con ayuda de un proyecto en Eclipse
- Compilar y ejecutar desde una ventana de DOS y desde Eclipse la aplicación LanzaPrac1
- Escribir los comentarios Javadoc para la clase ComplexNumber y para todos sus miembros
- Generar la documentación de la clase ComplexNumber con la herramienta Javadoc

4 Cuestiones

1. ¿Cuál es la diferencia entre declarar la clase ComplexNumber como pública y no hacerlo? (public class ComplexNumber OR class ComplexNumber). Probarlo y pensar la respuesta.

2. ¿Puede una clase tener dos métodos constructores con el mismo nombre? Probarlo y pensar la respuesta.

3. ¿Cuál es la diferencia entre declarar los atributos de clase de la clase ComplexNumber (real, imag) como públicos (public float), como privados (private float) o sin calificador de visibilidad (float)? Probarlo y pensar la respuesta.

4. ¿Qué sucede si no sobreescribimos el método toString() dentro de la clase ComplexNumber? ¿Compila el código? ¿Hay error en tiempo de ejecución? ¿Nos

muestra la ejecución de la aplicación el resultado deseado? Probarlo y pensar la respuesta.

5. ¿Se podría escribir el código de las dos clases (LanzaPrac1 y ComplexNumber) en un mismo fichero LanzaPrac1.java?
   Es decir, ¿se puede eliminar el archivo ComplexNumber.java del proyecto pero incluir (Copy&Paste) el código de la clase en ComplexNumber en el archivo LanzaPrac1.java? Probarlo y pensar la respuesta.