Programación Orientada a Objetos
Sesión 8: JavaDoc y Entrada/Salida en Java
Curso 2017-18

Félix Gómez Mármol, Santiago Paredes Moreno y Gregorio Martínez Pérez
{felixgm, chapu, gregorio}@um.es

Índice

Parte I
Teoría

1. JavaDoc

Al escribir código conviene documentarlo adecuadamente, es decir hacer comentarios que ayuden a comprender, modificar y utilizar el código. Para documentar una clase hay que decir cuál es su utilidad y cómo se utiliza (qué hace y cómo). **JavaDoc** es una herramienta que nos permite generar la documentación de una clase a partir de los comentarios que pongamos en el código fuente. Los comentarios deben seguir un formato concreto para que JavaDoc los entienda y genere la documentación. En concreto, lo que JavaDoc genera es una o varias páginas Web con la descripción de las clases.

1.1. Formato JavaDoc

JavaDoc lee el código fuente y trabaja con los comentarios encerrados entre /** y */ que haya justo antes de la definición de clases y métodos. Dentro de esos comentarios podemos utilizar algunas etiquetas especiales para indicar el nombre del autor, los parámetros de los métodos, el valor que devuelven, etc.

A continuación vamos a estudiar el formato con un pequeño ejemplo. Como se puede ver en el listado 1, justo antes de la definición de la clase `Prueba` aparece el comentario principal que describe de forma general la utilidad de la clase. En este comentario la etiqueta más habitual es `@author` que se utiliza para indicar el nombre del autor del código.

```
1  /**
2   * Esta clase es un ejemplo del uso de JavaDoc.
3   *
4   * @author Luis Daniel Hernandez Molinero, Santiago Paredes Moreno y Gregorio Martinez
        Perez
5   */
6  public class Prueba {
7     private int miembro;
8
9     /**
10     * Constructor: Crea un objeto y le da un valor a su miembro privado.
11     *
12     * @param v Este es el valor que almacena el objeto
13     */
14    public Prueba(int v) {
15      miembro = v;
16    }
17
18    /**
19     * Se utiliza para obtener el valor del miembro
20     *
21     * @return Valor del miembro <code>miembro</code>
22     */
23     public getMiembro() {
24       return miembro;
25     }
26 }
```

Listado 1: Ejemplo de uso de JavaDoc

Para cada método, incluidos los constructores, se crea un comentario en el que primero se describe lo que hace el método y después, siempre que sea necesario, se usan las etiquetas `@param` y/o `@return` para indicar para qué sirven los parámetros y el valor que devuelve el método en cada caso.

JavaDoc genera una página web. Es por este motivo que podemos incluir código HTML en los comentarios. HTML es el lenguaje que se utiliza para escribir páginas web y, básicamente, consiste en una serie de etiquetas (también llamadas marcas) encerradas entre los símbolos "<" y ">" que modifican la apariencia y formato del texto que escribimos. A modo de ejemplo, para que el texto aparezca en negrita lo encerraremos entre las marcas `<strong>` y `</strong>`. Para usar una fuente de paso fijo útil para escribir código, nombres de variables, parámetros, etc, podemos usar las marcas `<code>` y `</code>` cuando es una palabra, o `<pre>` y `</pre>` para varios renglones.

1.2. Usando JavaDoc

Una vez que tenemos un proyecto con los comentarios adecuados en formato JavaDoc, para generar las páginas web asociadas a dicho JavaDoc, pincharemos con el botón derecho del ratón sobre el nombre del proyecto y seleccionaremos "Generate Javadoc". Si todo va bien, se abrirá un navegador mostrando las páginas web generadas.

2. Introducción a la Entrada/Salida en Java

Las **operaciones de entrada/salida** son las que permiten que el usuario introduzca datos al programa (entrada) y facilitan la presentación de resultados al mismo (salida). Se realizan a través de los denominados **dispositivos de entrada/salida** entre los que se encuentran el teclado y el ratón como elementos más representativos de la entrada, y la pantalla o la impresora como ejemplos de salida.

Hay un tercer elemento denominado dispositivo de almacenamiento de datos que sirve al mismo tiempo como entrada y como salida. Un disco duro o un disco USB son ejemplos de dispositivos de almacenamiento de datos. En ellos los datos se guardan dentro de unidades llamadas **ficheros** que se organizan en **carpetas**. Los ficheros pueden contener datos de cualquier tipo y suelen tener un formato reconocible por su nombre. Por ejemplo, los ficheros cuyo nombre acaba en ".tex" son documentos LaTeX. Los ficheros con nombre acabado en ".txt" suelen ser ficheros con texto legible y modificable con editores de texto básicos como el Bloc de Notas.

En este apartado vamos a estudiar la entrada y salida de datos con formato de texto ya que nos será útil para mostrar información sobre los resultados de nuestros programas y para pasar datos a los mismos con los que puedan trabajar.

3. Entrada por teclado y salida por pantalla básicas

La clase `System` cuenta con dos miembros públicos estáticos (o de clase) denominados `in` y `out` que permiten realizar entrada y salida a través del teclado y la pantalla respectivamente. Mostrar datos por la pantalla es sencillo gracias al uso de la variable `System.out` y los métodos `print` y `println`. Sin embargo, aunque es posible utilizar directamente `System.in` para introducir datos por teclado a nuestra aplicación, es más sencillo utilizarla junto con la clase `Scanner` del paquete `java.util`.

En el listado 2 se puede ver un ejemplo de uso tanto de `System.out` como de la clase `Scanner` en combinación con `System.in` para hacer lectura del teclado. Es necesario crear un objeto de tipo `Scanner` pasándole como parámetro en el constructor el objeto `System.in`. A partir de ese momento se pueden utilizar los distintos métodos de esta clase [1] para ir leyendo datos de tipos simples y cadenas de caracteres.

```
1  import java.util.Scanner;

3  public static void main(String[] args) {
4    Scanner scanner = new Scanner(System.in);
5    System.out.print("Introduce un numero entero: ");
6    int numeroEntero = scanner.nextInt();
7    System.out.println("El entero leido ha sido: " + numeroEntero);
8    scanner.close();
9  }
```

Listado 2: Ejemplo de uso de la clase `Scanner`

En el ejemplo, `scanner.nextInt()` lee lo que se introduzca por teclado saltándose los primeros espacios en blanco y tratando de interpretar como un número entero lo que haya a continuación hasta el siguiente espacio en blanco o hasta que encuentre un retorno de línea (asociado con la pulsación de la tecla `Intro`por parte del usuario). Como se puede ver, la última línea del programa es la llamada al método `close()` del objeto de tipo `Scanner`. Es conveniente hacer esto siempre que hayamos terminado de usar la entrada.

Para leer un entero usaremos el método `nextInt()`, para leer un valor real `nextDouble()`, para un valor booleano `nextBoolean()`. El método `next()` devuelve la siguiente palabra que encuentre entre espacios en blanco, y para leer todo lo que haya escrito hasta el final de la línea usaremos `nextLine()`.

También es posible determinar si vamos a poder ejecutar correctamente uno de los métodos anteriores, o no, usando su equivalente de tipo `hasNext*()`. Es decir, el método `hasNextInt()` devuelve verdadero (true) si lo siguiente que hay es un entero. También existen los métodos `hasNextDouble()`, `hasNextLine()` y `hasNextBoolean()` con el mismo propósito, pero aplicable a distintos tipos de datos.

[1] El JavaDoc de esta clase está disponible en https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Los objetos de la clase `Scanner` analizan carácter a carácter el objeto que se les pasa como parámetro en el constructor y responden a los distintos métodos saltando cuando es necesario los espacios en blanco.

```
1  import java.util.Scanner;
2
3  public static void main(String[] args) {
4    Scanner scanner = new Scanner(System.in);
5
6    System.out.print("Introduce un numero entero: ");
7    int numeroEntero = 0;
8    if (scanner.hasNextInt()) {
9      numeroEntero = scanner.nextInt();
10     System.out.println("El entero leido ha sido: " + numeroEntero);
11   } else {
12     System.out.println("No se ha introducido un entero valido");
13   }
14
15   System.out.print("Introduce tu color favorito: ");
16   String color = scanner.next();
17   System.out.print("Introduce tu nombre completo: ");
18   String nombre = scanner.next();
19   System.out.println("Te llamas " + nombre + " y tu color favorito es el " + color);
20   scanner.close();
21 }
```

Listado 3: Ejemplo avanzado de uso de la clase `Scanner`

4. Entrada por teclado y salida por pantalla usando ventanas

La salida por pantalla básica usando el objeto `System.out` se usa frecuentemente, pero muchas aplicaciones no utilizan la entrada de teclado básica. En su lugar utilizan ventanas con formularios para introducir los datos usando el teclado y el ratón para pulsar sobre botones. Cuando necesitamos hacerle al usuario una pregunta concreta, por ejemplo, pedirle que confirme que realmente quiere hacer una operación, puede ser más adecuado usar una ventana. A estas ventanas con un mensaje y una pregunta concreta se les llama **cuadros de diálogo**. Para generarlas podemos recurrir a la clase `JOptionPane` que incluye varios métodos estáticos útiles para este fin de los cuáles a continuación mostramos algunos:

- `void JOptionPane.showMessageDialog(null, String mensaje)`. Este método está sobrecargado para admitir diferentes conjuntos de parámetros. Esta es la versión más sencilla que nos permite mostrar un mensaje de información al usuario.

Figura 1: Ejemplo de cuadro de diálogo generado con el método `showMessageDialog()`

- `int JOptionPane.showConfirmDialog(null, String mensaje)`. Esta versión simple del método muestra un mensaje y permite al usuario pulsar "Si", "No" o "Cancelar". Según lo que se haya pulsado se devuelve un valor entero que coincidirá con alguna de las constantes definidas en la clase `JOptionPane` para gestionar dichas acciones. Algunas de estas constantes son `YES_OPTION`, `NO_OPTION`, `CANCEL_OPTION`, `OK_OPTION`, `CLOSED_OPTION`, etc.

Figura 2: Ejemplo de cuadro de diálogo generado con el método `showConfirmDialog()`

- `String JOptionPane.showInputDialog(null, String mensaje)`. Esta versión, también simplificada, muestra un mensaje al usuario pidiendo que introduzca una cadena que es devuelta como salida del método al cerrarse el diálogo pulsando el botón `Aceptar` por parte del usuario.

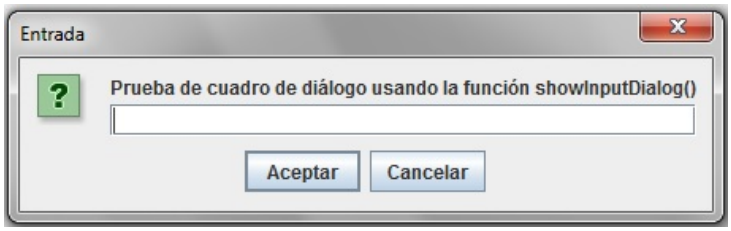

Figura 3: Ejemplo de cuadro de diálogo generado con el método `showInputDialog()`

Para hacer uso de esta clase es necesario importarla antes, para lo que utilizaremos la instrucción `import javax.swing.JOptionPane;` En cualquiera de los métodos anteriores el programa queda detenido hasta que el cuadro de diálogo se cierra ya que, lo normal, es que la respuesta del usuario sea necesaria para poder continuar con el programa. Como hemos comentado existen versiones alternativas de los métodos que permiten personalizar el cuadro de diálogo variando, por ejemplo, el icono que aparece. Para saber más se recomienda consultar la documentación de Java en la dirección https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html.

5. Control de errores en entrada/salida

Cuando trabajamos con dispositivos de entrada/salida se pueden producir multitud de errores durante la ejecución de nuestros programas. A estos errores se les denomina Excepciones, tal y como ya estudiamos en la sesión anterior. En Java existen muchas clases que representan los distintos tipos de Excepciones.

Normalmente, sabemos a priori en qué parte del código se puede producir cada tipo de Excepción porque en la documentación (JavaDoc) de las clases que usamos especifica para cada método qué Excepciones puede llegar a generar (apartado `throws`).

Como explicamos en la sesión anterior, Java dispone de un mecanismo para detectar cuándo se produce una Excepción y hacer algo al respecto. Consiste, por una parte, en encerrar el código en el que se puede producir el error dentro de un bloque especial denominado `try`, y por otra, el código que queramos que se ejecute en caso de que se produzca una excepción lo pondremos a continuación en un bloque denominado `catch`. Podemos verlo como que intentamos (`try`) ejecutar un bloque de código y si se produce una excepción la capturaremos (`catch`) y ejecutamos otro bloque de código alternativo. Para cada bloque `try` podemos asociar uno o más bloques `catch` ligados a las distintas excepciones que se pueden producir.

Por ejemplo, mirando el apartado `throws` de la documentación en JavaDoc de la clase `Scanner`, sabemos que el método nextInt puede generar tres Excepciones distintas, a saber: `InputMismatchException`, `NoSuchElementException`, y `IllegalStateException`. El listado 4 muestra el primer ejemplo de `Scanner` pero con el código necesario para controlar todos los posibles errores de introducción de datos. Fíjate que el bloque `try` contiene tanto el intento de lectura del teclado como la línea siguiente que es la que

```
1  public static void main(String[] args) {
2    Scanner s = new Scanner(System.in);
3
4    System.out.println("Introduce un numero entero");
5    try {
6      int i = s.nextInt();
7      System.out.println("El entero leido ha sido: " + i);
8    } catch (InputMismatchException error1) {
9      System.out.println("No has introducido un numero correcto");
10   } catch (NoSuchElementException error2) {
11     System.out.println("No hay cifras");
12   } catch (IllegalStateException error3) {
13     System.out.println("Se ha cerrado la entrada");
14   }
15   s.close();
16 }
```

Listado 4: Uso de try-catch con `Scanner`

utiliza el dato leído. Sin embargo, si el dato no llega a leerse porque se produce un error a mitad, la línea 7 no llegará a ejecutarse, en su lugar se ejecutará la línea 9, la 11 o la 13 dependiendo del error que se haya producido.

Si no nos interesa distinguir el tipo de error podemos usar un único bloque `catch` con la clase `Exception` que sirve para capturar cualquier tipo de Excepción, ya que `Exception` es una clase anterior en la jerarquía a las tres indicadas antes.

```
1  public static void main(String[] args) {
2    Scanner s = new Scanner(System.in);
3
4    System.out.println("Introduce un numero entero");
5    try {
6      int i = s.nextInt();
7      System.out.println("El entero leido ha sido: " + i);
8    } catch (Exception error) {
9      System.out.println("Se ha producido un error");
10   }
11   s.close();
12 }
```

Listado 5: Uso de try-catch con un único bloque catch genérico

Finalmente, NetBeans es capaz de detectar cuando es necesario rodear una invocación a un método que puede generar excepciones. En esos casos aparece un símbolo en el lateral y al pinchar sobre él nos permite seleccionar que automáticamente se incluya el bloque `try-catch` adecuado.

6. Entrada y salida con ficheros

Además de pedir al usuario que introduzca información y mostrarle resultados por pantalla es útil en algunas ocasiones utilizar ficheros para almacenar y leer datos. Por ejemplo, en un programa que tenga que hacer cálculos con multitud de datos, es mejor tener los datos en un fichero que pedírselos al usuario cada vez que lancemos el programa. Por otro lado, si queremos que entre dos ejecuciones distintas del mismo programa se conserve algún dato necesitaremos guardarlo en un fichero.

6.1. Lectura de ficheros (entrada)

En ocasiones no interesa pedir al usuario los datos de entrada, por ejemplo, cuando son muchos, o siempre son los mismos. En estos casos conviene tener los datos en un fichero de texto y hacer que el programa lea dicho fichero y obtenga de él los datos necesarios. Para hacer esto podemos utilizar la clase `Scanner` sin más que pasar al constructor un objeto de la clase `File` que se encuentra en el paquete `java.io`.

```
1  public static void main(String[] args) {
2    Scanner s = null;
3    try {
4      s = new Scanner(new File("datos.txt"));
5    } catch (Exception error) {
6      System.out.println("No existe el fichero");
7    }
8    int suma = 0;
9    while (s.hasNextInt()) {
10     int i = s.nextInt();
11     suma += i;
12   }
13   System.out.println("La suma total es " + suma);
14   s.close();
15 }
```

Listado 6: Ejemplo de uso de la clase `Scanner` para leer de un fichero

El listado 6 muestra un ejemplo de lectura de un fichero llamado "datos.txt" que contiene lo siguiente

```
1 2 3 4 5
6 7 8 9 10
```

Dicho fichero debe encontrarse en la carpeta del proyecto para que el programa funcione correctamente y, como se puede ver, durante la creación del objeto de tipo `Scanner` ha sido necesario utilizar un bloque `try-catch` ya que, al manejar ficheros de datos, también se pueden producir errores durante su creación, por ejemplo, que el fichero no exista o no se encuentre en la carpeta adecuada. El programa continúa leyendo todos los enteros que encuentra y sumándolos para, finalmente, mostrar el resultado total. Como se puede ver, se continúa leyendo del fichero mientras haya un entero sin procesar. Otra forma de detectar que ya se ha llegado al final del fichero es con el método `hasNext()`. Finalmente, hay que destacar que aunque el fichero de entrada tiene los distintos números separados por espacios en blanco y retornos de carro, el programa funciona correctamente gracias al uso de la clase `Scanner` que salta todos estos caracteres de separación de cadenas.

Finalmente, es posible modificar el comportamiento del objeto de tipo `Scanner` haciendo que en lugar de espacios en blanco utilice como separador otra cadena. Para hacer esto se usa el método `useDelimiter(String cadena)`.

6.2. Escritura de ficheros (salida)

Para escribir en un fichero existente, o crear uno nuevo en el que poder escribir, se pueden utilizar multitud de clases en Java. La opción que presentamos aquí es una de las más sencillas de todas. Se trata de utilizar un objeto de la clase `PrintWriter` a cuyo constructor se le pasa otro de la clase `FileWritter`, ambas pertenecientes al paquete `java.io`.

Durante la creación de estos objetos se pueden producir errores debido, por ejemplo, a que no hay espacio suficiente en el disco o que no es posible escribir en la ubicación elegida por falta de permisos. Es por ello que es necesario ejecutarlos siempre dentro de bloques `try-catch`.

Una vez que el objeto de tipo `PrintWritter` está creado podremos aplicar sobre él métodos conocidos como `print` y `println` que nos permitirán escribir adecuadamente en el fichero. En el listado 7 se puede ver un ejemplo completo en el que se pide al usuario su nombre y edad usando `System.out`, se recogen los datos introducidos usando un `Scanner` y se guardan los mismos en un fichero llamado "datos.txt" usando un objeto de la clase `PrintWriter`.

```
1  public static void main(String[] args) {
2    Scanner entrada = new Scanner(System.in);
3
4    try {
5      PrintWritter salida = new PrintWriter(new FileWriter("datos.txt"));
6      System.out.println("Escribe tu nombre");
7      String nombre = entrada.nextLine();
8      System.out.println("Escribe tu edad");
9      int edad = entrada.nextInt();
10
11
12     salida.println("Nombre : " + nombre);
13     salida.println("Edad : " + edad);
14   } catch (Exception error) {
15    System.out.println("Error al escribir en el fichero");
16   }
17   if (salida != null) salida.close();
18 }
```

Listado 7: Ejemplo de uso de la clase `PrintWriter` para escribir en un fichero

Parte II
Prácticas

7. Ejercicios sobre entrada/salida en Java tanto por pantalla como por fichero

En esta sesión de prácticas, para la que debes haber leído y entendido en detalle la parte teórica de esta sesión, vamos a trabajar sobre los conceptos relativos a la entrada/salida en Java tanto por pantalla como por fichero.

EJERCICIO 1. *Crea un programa con el código del listado 3, ejecútalo y cuando te haga la primera pregunta introduce* `  23  `*, es decir el número 23 precedido de varios espacios y seguido de varios espacios, pulsa el retorno de línea y contesta las siguientes preguntas. Vuelve a ejecutarlo y contesta a la primera pregunta con la cadena "23 Azul MiNombre" y pulsa retorno de línea. Fíjate cómo las dos siguientes preguntas aparecen respondidas directamente. Explica por qué ocurre esto.*

EJERCICIO 2. *Crea un programa en el que, usando el método adecuado de* `JOptionPane`*, se le pregunte al usuario su edad y determine el año en el que nació. Para ello puedes utiliza el método* `parseInt` *de la clase Integer, y las clases* `java.util.Date` *y* `java.util.Calendar`*. Primero debes obtener el objeto que representa el calendario del ordenador con el método estático* `getInstance` *de la clase* `Calendar`*. Utilizando este objeto serás capaz de obtener la fecha actual en un objeto de tipo Date y con él calcular el año de nacimiento del usuario. El programa debe estar preparado para detectar si el texto introducido por el usuario no es un número capturando las excepciones que puedan generarse al usar el método* `parseInt`*.*

EJERCICIO 3. *Escribe un programa que lea un fichero de texto y cuente el número de frases, es decir, el número de secuencias de palabras separadas por el caracter punto ".".*

EJERCICIO 4. *El programa del listado 7 crea un fichero con los datos de una persona. Escribe un programa capaz de leer un fichero como el creado anteriormente y mostrar sus datos por la pantalla.*