Programació Bàsica

UF3 | RA1 i RAE

CAS 6

CFGS-ASIX/DAW | 2020-2021

Profesora María Merino | **Grupo 5** Daniela Gallardo | **Fecha** 26/05/2021

ÍNDICE

INTRODUCTION

From a programming perspective, a fully functional file management system has become essential in our digital era. It has an enormous impact in terms of data storage, access, manipulation and, safety[1].

Many programming languages provide libraries that allow us, through pre-defined tools, to build up programs. Those programs, for instance, can perform tasks for accessing different systems for future users to interact with their components (files).

The Java language is no exception offering specific libraries for file management; with the java.io package import, if we do the correct method call (defined in this package), we can perform any task within a particular file system. This java package contains nearly every class we might need to use as input source and output sources while defining a destination (I/O) in Java file management.

But how do we build up a java program for learning how to access and manipulate files? We need to go through its development step by step, understanding more concepts and processes, which we will perform and clear up in the following pages.

[1] Data integrity: due to its accuracy and consistency.

0. ENUNCIADO

Continuando con la tarea encargada por el departamento de informática del Instituto Montsià para realizar una aplicación para la gestión y asignación de los portátiles y discos duros que se hacen a los alumnos, se quiere disponer de un primer volcado de datos. En concreto sería una aplicación ejecutable desde terminal que proporcione la siguiente funcionalidad:

- A partir de la información recogida con los pedidos dmidecode y lshw, en varios ficheros, disponer de un archivo único (archivo resultado) que contenga de forma diferenciable, los siguientes datos de cada PC:
    - Aula
    - PC
    - Marca
    - Modelo
    - Número de serie
    - Mac Wifi
    - Mac ethernet

- Los ficheros proporcionados por dmidecode y lshw deberán archivar de forma que permita hacer una comprobación de qué archivos se han procesado y cuáles no pudimos por algún motivo.

- En concreto la aplicación debería permitir:

    - **Procesar todos los ficheros de un directorio**. La aplicación, debe pedir en qué directorio están los archivos dmidecode y lshw. Y añadir las líneas adecuadas en el archivo resultado.

    - **Introducir manualmente la información de un PC.** La aplicación, debe pedir los datos de 1 PC, y añadir las líneas adecuadas en el archivo resultado.

    - **Modificar la información de un PC.** La aplicación, debe mostrar los datos de todos los PCs, ordenadas por aula y número, y permitir la introducción de los datos de modificación.

    - **Borrar la información de un PC.** La aplicación, debe mostrar los datos de todos los PCs, ordenadas por aula y número, y permitir borrar la información de un PC, a partir de cualquiera de los datos que lo identifiquen de forma única.

    - **Consultar los PCs de un aula.** La aplicación, debe pedir al usuario qué aula quiere ver, y el nombre del archivo donde guardar esta información. Mostrará por pantalla el listado de esta aula, e informará de la ruta completa en la que se ha guardado el archivo con la información de los PCs del aula.

- Se debe modularizar correctamente el programa.

- La entrega del trabajo debe incluir: el código fuente, el javadocs, explicación de las decisiones tomadas para implementar el programa y aprendizajes ejemplificados. La introducción y la conclusión (como mínimo) deberán estar redactadas en inglés.

- En la presentación se mostrará el funcionamiento de la aplicación además de hacer la presentación de diapositivas habitual. Todos los miembros del equipo deberán hacer una intervención en inglés.

1. LA CLASE FILE

La clase File pertenece al paquete *java.io*[2]. Podríamos resumir sobre la clase File en Java que:

- Es una representación abstracta[3] de nombres de rutas (*paths* en inglés) de archivos y directorios.
- Nos permite manejar archivos y ficheros.
- Contiene varios métodos para trabajar mediante la ruta de un archivo/fichero, así podremos:
    - Eliminar y renombrarlos.
    - Crear nuevos directorios.
    - Enumerar el contenido de un directorio.
    - Determinar varios atributos comunes de archivos y directorios.

Básicos a tener en cuenta
• El nombre de una ruta, ya sea abstracta o en forma de string puede ser absoluta o relativa. • El padre del nombre de una ruta abstracta la podemos obtener invocando al método getParent() de su clase. • Primero que nada, deberíamos crear el objeto de la clase File pasando el nombre del archivo o directorio a éste. Un sistema de ficheros debe implementar restricciones a determinadas operaciones en el objeto del sistema de archivos (permisos de lectura, escritura y ejecución) conocidas como “permisos de acceso” • Las instancias de la clase File son inmutables; esto quiere decir que una vez que son creadas, el nombre abstracto de una ruta representado en un objeto de la clase File, nunca cambiará.

1.2 ¿Cómo crear un objeto de la clase File? (Sintáxis)

Un objeto de la clase File se crea pasando una Cadena que representa el nombre de un archivo con la siguiente sintáxis:

```
File f = new File(<nombre_del_fichero[4]>);
```

[2] Esta clase fue fundamental para el manejo de ficheros y directorios hasta Java SE 6. A partir de Java SE 7 tenemos a nuestra disposición las clases y métodos de Java NIO.2

[3] El término abstracto, en este caso, se refiere al hecho que hasta el momento que no hayamos creado un objeto, la classe en sí no "hace" nada, sólo es nuestro molde para crear objetos; una vez que los creémos podrán o no comportarse de diferentes maneras, según programemos.

[4] Puede ser una ruta absoluta o una relativa.

```
f.createNewFile();
```

A tener en cuenta

- Es posible que el archivo con el que queremos trabajar exista o no.
- Tenemos que pensar que el nombre del archivo puede ir con una ruta, absoluta o relativa. También hay ciertas diferencias respecto de Unix y Windows.
- Cuando un programa se ejecuta, ya sea en Java o en cualquier otro lenguaje de programación, por defecto se le asigna una carpeta de trabajo. Esta carpeta suele ser, por defecto, la carpeta desde donde se encuentra el programa, si bien hay sistemas operativos que permiten especificar explícitamente mediante alguna opción sobre sus atajos o archivos ejecutables. En el caso de un programa en Java ejecutado a través de un IDE, la carpeta de trabajo suele ser la misma carpeta donde se ha elegido guardar los archivos del proyecto.

1.3 ¿Cómo leer un archivo de texto? (Sintáxis)

1.3.1 Opción 1: La clase Scanner

1. Importamos la clase File y java.util.Scanner (explicación en el paso 3)
2. Creamos un objeto de la clase File.
3. La clase que permite llevar a cabo la lectura de datos desde un archivo orientado a carácter es exactamente la misma que permite leer datos desde el teclado.
   Los valores almacenados en los archivos de este tipo se encuentran exactamente en el mismo formato que ha usado hasta ahora para entrar información en sus programas: una secuencia de cadenas de texto. La única diferencia es que estos valores no se piden al usuario durante la ejecución, sino que se leen desde el fichero.

```
import java.io.File;
import java.util.Scanner;
```
1

```
...
```

```
    // comment
    File fitxer = new File("Document.txt");
```
2

```
    Scanner lectorFitxer = new Scanner(fitxer);
```
3

```
...
```

prueba con filtros

```
46    }
47
48    public static void leer(File f2){
49        try {
50            Scanner sc = new Scanner(f2);
51            while (sc.hasNext()) {
52                String s= sc.nextLine();
53                if(s.contains("Product Name: HP")){
54                    System.out.println(s);
55                }
56            }
57        }catch (Exception e) {
58            System.out.println(e.toString());
59        }
60    }
61
62
63
```

Problems @ Javadoc Declaration Console

```
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (19-05-2021 10:27:57
Escriu la ruta de l'arxiu que vols processar
/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/dmidecode/22Aula_PC7 - Laia Girbes Capafons.txt
        Product Name: HP ProBook 450 G6
```

Prueba método leer()
bloque try

```
    public static void leer(File f2){
        try {
            nomfitxer=f2.getName();
            // prints
            System.out.println(nomfitxer);
            Scanner sc = new Scanner(f2);
            while (sc.hasNext()){
                String s= sc.nextLine();
                //imprimir los campos de la marca y modelo
                if((s.contains("Product Name"))||
                  (s.contains("Manufacturer"))||
                  (s.contains("Serial Number"))
                  ){
                    System.out.println(s);
                //evitar que imprima información de placa base y posteriores
                }else if(s.contentEquals("Base Board Information")){
                    System.exit(0);
                }
            }
```

Problems @ Javadoc Declaration Console

```
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (19-05-2021 12:14:30 – 12:14:36
Escriu la ruta de l'arxiu que vols processar
/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/dmidecode/22Aula_PC7 - Laia Girbes Capafons.txt
22Aula_PC7 - Laia Girbes Capafons.txt
        Manufacturer: HP
        Product Name: HP ProBook 450 G6
        Serial Number: 5CD9339X8L
```

Detalle prueba método leer()
bloques try y catch

```
 */
public static void leer(File f2) {
    try {
        nomfitxer = f2.getName();
        // prints
        System.out.println(nomfitxer);
        Scanner sc = new Scanner(f2);
        while (sc.hasNext()) {
            String s = sc.nextLine();
            // imprimir los campos de la marca y modelo
            if ((s.contains("Product Name")) || //dmidecode
                    (s.contains("producto")) ||//lshw
                    (s.contains("vendor")) || //lshw
                    (s.contains("Manufacturer")) || //dmidecode
                    (s.contains("serial")) ||//lshw
                    (s.contains("Serial Number"))) { //dmidecode
                System.out.println(s);
                // evitar que imprima información de placa base y posteriores
            }else if(s.contentEquals("Base Board Information")){
                System.exit(0);
            }
        }
    } catch (Exception e) {
        System.out.println(e.toString());
    }
  }
}
```

1.3.2 Opción 2: FileReader i BufferedReader

Podemos abrir un archivo de texto para leer utilizando la clase **FileReader**. Esta clase tiene métodos que nos permiten leer caracteres. Sin embargo, suele ser habitual querer las líneas completas, bien porque nos interesa la línea completa, bien para poder analizarla después y extraer campos de ella.

FileReader no tiene métodos que nos permitan leer líneas completas, pero sí BufferedReader. Afortunadamente, podemos construir un BufferedReader a partir de FileReader de la sigüiente manera:

1. I

`import java.io.File;`
`import java.io.FileReader;`
`import java.util.Scanner;`
	1

`...`	
`File fitxer = new File("Document.txt");`	**2**
`FileReader fr = new FileReader (fitxer);`	**3**
`BufferedReader br = new BufferedReader(fr);`	
`...`	
`String linea = br.readLine();`	

La apertura del archivo y su posterior lectura pueden lanzar excepciones que debemos capturar. Por ello, la apertura del archivo y la lectura se debe meter en un bloque try-catch.

Además, se debe cerrar el archivo una vez que terminamos de usarlo, tanto si todo ha ido bien como si ha habido algún error en la lectura después de haberlo abierto. Por esto, se suele poner además de los bloques al try-catch un bloque finally y dentro de él, el cierre/close () del archivo.

2. EXCEPCIONES

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. [5]
Una Excepción es:

- Una situación Excepcional.
- Altera la ejecución normal de un programa.
- El método dónde sucede crea un objeto de excepción y se lo pasa alguien para que pueda tratarlo.

Sin embargo, el uso de excepciones tiene varias ventajas:

- Permiten separar el código de tratamiento de errores del código normal.
- Evitan que haya errores inadvertidos.
- Permiten la propagación de los errores.
- Permiten agrupar en un lugar común el tratamiento de errores.

2.1 El tratamiento de Excepciones

Para aplicar el tratamiento de excepciones, utilizamos la siguiente sintaxis:

Importante: el uso de **`finally`** no es obligatorio, y podemos incluir más de un bloque **`catch`**.

```
try {
    instrucciones;
} catch (Exception e) {
    instrucciones;
} finally {
    instrucciones
}
```

1. Bloque try (código propio de la aplicación) debe envolver las sentencias que son susceptibles de provocar uno o varios tipos de excepción. Debemos agrupar las sentencias que vayan a tener un tratamiento idéntico de la situación excepcional.
2. Bloque catch (código que trata la situación excepcional), estos bloques sirven como manejadores de las situaciones excepcionales. Puede haber más de uno. Cada bloque puede manejar uno o más tipos de excepciones.
3. Bloque finally (código que se ejecuta siempre), tanto si hemos terminado correctamente el bloque try como el bloque catch. Se suele utilizar como código que asegura el cierre de recursos abiertos (ficheros, bases de datos, etc).

[5] Definición de ***excepción*** encontrada en la documentación de JAVA por ORACLE
What Is an Exception? (The Java™ Tutorials > Essential Classes > Exceptions).

`try {` `  instrucciones;`	**1**
`} catch (Exception e) {` `  instrucciones;`	**2**

`finally {`
`  instrucciones`
`}`
	3

3. CONSTRUCTORES DE LA CLASE FILE

File(File parent, String child)	Crea una nueva instancia de la clase File a partir de un nombre de ruta abstracto padre y una cadena de nombre de ruta hijo.
File(String pathname)	Crea una nueva instancia de la clase File convirtiendo el String pathname dado en un nombre de ruta abstracto.
File(String parent, String child)	Crea una nueva instancia de la clase File a partir de un String pathname abstracto padre y un String pathname de ruta hijo.
File(URI uri)	Crea una nueva instancia de File convirtiendo el file: URI en un nombre de ruta abstracto.

4. MÉTODOS DE LA CLASE FILE

Algunos métodos de la clase File son:

boolean **canExecute()**	Comprueba si la aplicación puede ejecutar el archivo indicado por esta ruta abstracta.
boolean **canRead()**	Comprueba si la aplicación puede leer el archivo indicado por esta ruta abstracta.
boolean **canWrite()**	Indica si la aplicación puede modificar el archivo indicado por este nombre de ruta abstracto.
int **compareTo(File pathname**)	Compara dos nombres de ruta abstractos lexicográficamente.
boolean **createNewFile()**	Crea atómicamente un nuevo archivo vacío con el nombre de esta ruta abstracta.
static **File createTempFile(String prefix,**	Crea un archivo vacío en el directorio de

String suffix)	archivos temporales por defecto.
boolean **delete()**	Elimina el archivo o directorio indicado por esta ruta abstracta.
boolean **equals(Object obj)**	Comprueba la igualdad de este nombre de ruta abstracto con el objeto dado.
boolean **exists()**	Comprueba si el archivo o directorio indicado por esta ruta abstracta existe.
String **getAbsolutePath()**	Devuelve la cadena de ruta absoluta de este nombre de ruta abstracto.
long **getFreeSpace()**	Devuelve el número de bytes no asignados en la partición
String **getName()**	Devuelve el nombre del archivo o directorio denotado por este nombre de ruta abstracto.
String **getParent()**	Devuelve la cadena de nombre de ruta del padre de este nombre de ruta abstracto.
File **getParentFile()**	Devuelve el nombre abstracto del padre de este nombre abstracto.
String **getPath()**	Convierte este nombre de ruta abstracto en una cadena de ruta.
boolean **isDirectory()**	Comprueba si el archivo indicado por esta ruta es un directorio.
boolean **isFile()**	Comprueba si el archivo indicado por esta ruta abstracta es un archivo normal.
boolean **isHidden()**	Comprueba si el archivo nombrado por esta ruta abstracta es un archivo oculto.
long **length()**	Devuelve la longitud del archivo denotado por este nombre de ruta abstracto.
String[ ] **list()**	Devuelve un array de Strings que nombran los archivos y directorios del directorio .
File[ ] **listFiles()**	Devuelve un array de pathnames abstractos que denotan los archivos del directorio.
boolean **mkdir()**	Crea el directorio nombrado por este nombre de ruta abstracta.
boolean **renameTo(File dest)**	Cambia el nombre del archivo indicado por esta ruta abstracta.

boolean **setExecutable(boolean executable)**	Un método conveniente para establecer el permiso de ejecución del propietario.
boolean **setReadable(boolean readable)**	Un método conveniente para establecer el permiso de lectura del propietario.
boolean **setReadable(boolean readable, boolean ownerOnly)**	Establece el permiso de lectura del propietario o de todos.
boolean **setReadOnly()**	Marca el archivo o directorio nombrado para que sólo se permitan operaciones de lectura.
boolean **setWritable(boolean writable)**	Un método conveniente para establecer el permiso de escritura del propietario.
String **toString()**	Devuelve la cadena de ruta de este nombre de ruta abstracto.
URI **toURI()**	Construye un URI de archivo que representa este nombre de ruta abstracto.

5. ARRAY VS. ARRAYLIST

En un principio nos planteamos trabajar con un Array de objetos para guardarnos la información de los campos que definen un PC, pero desde que abordamos el desarrollo del programa objetos de la clase PC que contienen la información extraída de los ficheros .txt dmidecode y lshw [consultar apartado 7.3.1], se hace más fácil y óptima la manipulación de objetos tipo PC con Arraylist.

A continuación detallamos una lista comparativa de los porqués de nuestra elección:

Un Array podríamos considerarlo como funcionalidad básica para guardar un grupo de datos de un mismo tipo proporcionada por Java. Mientras que ArrayList es parte de la "collection framework" en Java. Por lo tanto:

Array	ArrayList
● Los elementos que me guardo en un Array se acceden usando [ ]	● ArrayList tiene un conjunto de métodos para acceder a los elementos y modificarlos
● Array es una estructura de datos de tamaño fijo.	● ArrayList no posee un tamaño fijo. ● No es necesario mencionar el tamaño de Arraylist al crear su objeto. ● Incluso si especificamos alguna capacidad inicial, podemos añadir más elementos.
● Los arrays pueden contener tanto tipos de datos primitivos como objetos de una clase, dependiendo de la definición del array. ● En los arrays, depende de si los arrays son de tipo primitivo o de tipo objeto. ○ En el caso de los tipos primitivos, los valores reales son ubicaciones contiguas. ○ pero en el caso de los objetos, la asignación es similar a la de ArrayList.	● ArrayList no puede crearse para tipos de datos primitivos, los elementos del ArrayList son siempre referencias a objetos en diferentes ubicaciones de memoria. ● Por lo tanto, dentro de un ArrayList, los objetos reales nunca se almacenan en ubicaciones contiguas; las referencias de los objetos reales se almacenan en ubicaciones contiguas.
● Las funciones soportadas por ArrayList no son del todo compatibles con Arrays.	● Java ArrayList soporta muchas operaciones adicionales como indexOf(), remove(), etc.

[consultar métodos en Java ArrayList]

6. REGEX: Regular Expressions

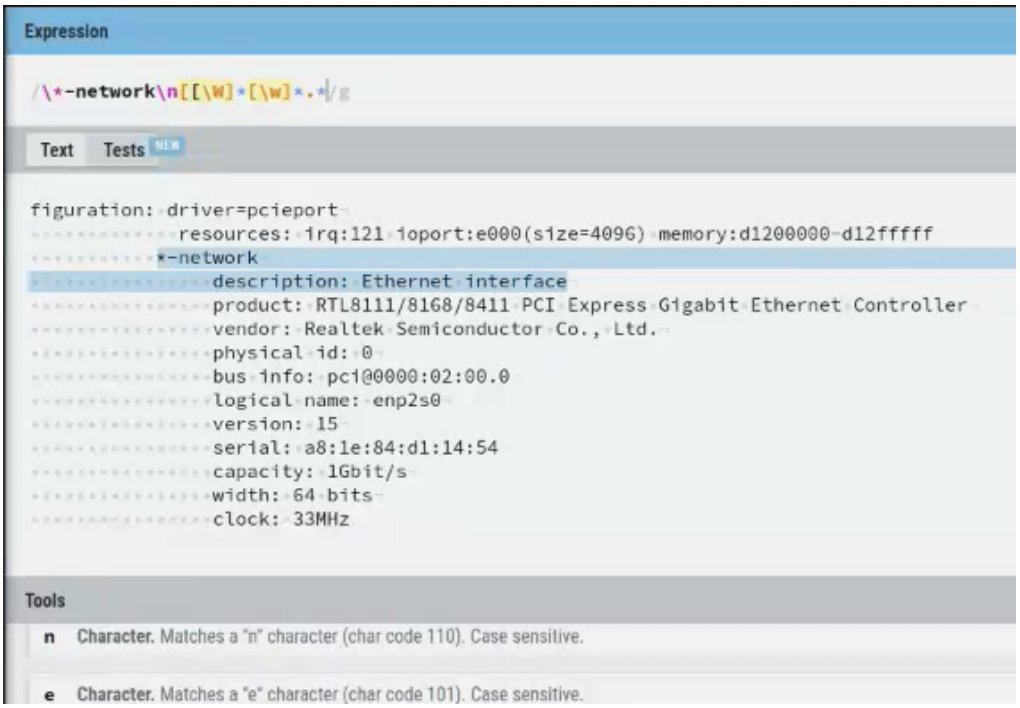
Utilizar REGEX en este caso ha sido bastante más efectivo que continuos bloques condicionales que no siempre funcionaron al momento de filtrar campos específicos.

Una Regular Expression es un patrón textual utilizado para buscar en el texto. Lo hace "comparando" la expresión regular con el texto. El resultado de comparar y examinar una expresión regular con un texto es:

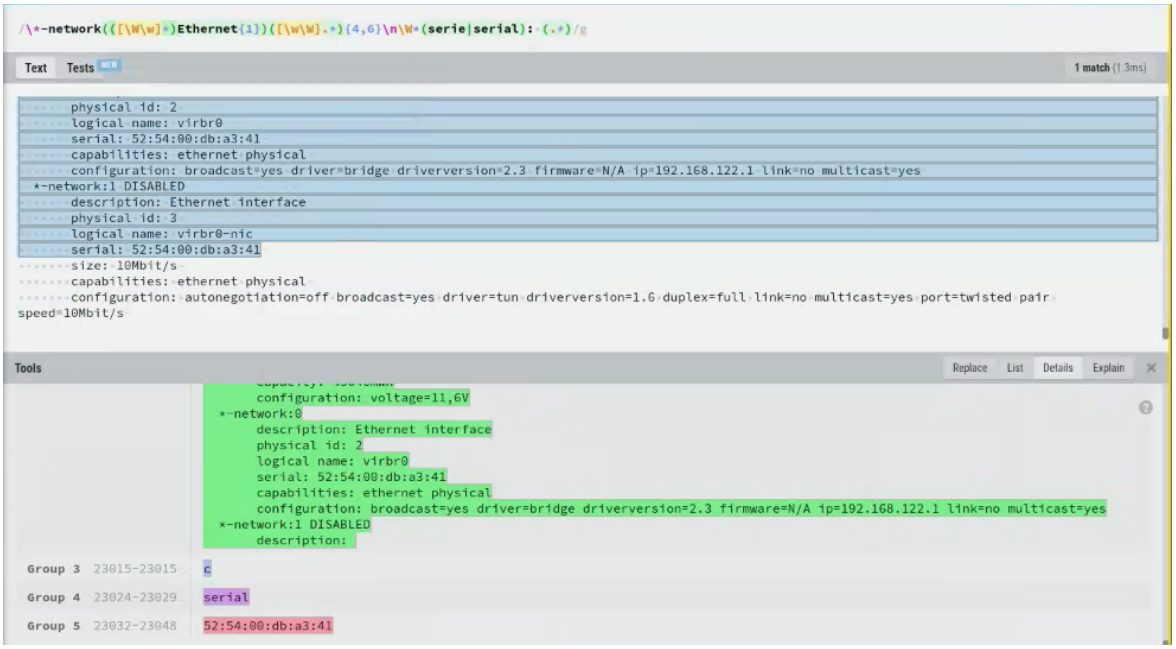
- Un verdadero / falso que especifica si la expresión regular coincidió con el texto.
- Un conjunto de coincidencias - una coincidencia por cada ocurrencia de la expresión regular encontrada en el texto.

Por ejemplo, puede utilizar una expresión regular para buscar en una cadena Java direcciones de correo electrónico, URLs, números de teléfono, fechas, etc. Esto se haría comparando diferentes expresiones regulares con la cadena. El resultado de comparar cada expresión regular con la cadena sería un conjunto de coincidencias - un conjunto de coincidencias para cada expresión regular (cada expresión regular puede coincidir más de una vez).

Para este caso se utilizó como herramienta inicial el sitio web regexr.com, realizamos pruebas como la que se muestran a continuación para ver si en cada fichero .txt podríamos filtrar información de los ordenadores según parámetros establecidos por un patrón con Expresiones Regulares :

prueba con RegEx 1


prueba con RegEx 2



6.2 Clases básicas de Java Regex

La API regex de Java consta de dos clases principales. Éstas son

- Pattern (java.util.regex.Pattern)
- Matcher (java.util.regex.Matcher)

La clase Pattern se utiliza para crear patrones (expresiones regulares). Un patrón es una expresión regular precompilada en forma de objeto (como una instancia de Pattern), capaz de coincidir con un texto.

La clase Matcher se utiliza para comparar una expresión regular dada (instancia de Patrón) con un texto varias veces. En otras palabras, para buscar múltiples ocurrencias de la expresión regular en el texto. El emparejador le dirá en qué parte del texto (índice de caracteres) encontró las ocurrencias. Puede obtener una instancia de Matcher a partir de una instancia de Pattern.

En el Cas 6 se utilizó RegEx creando una clase específica para su uso, en la clase creamos un método que nos convirtiera en string un pattern que hiciera match dentro de un fichero .txt para extraer datos específicos utilizando distintos patterns de entrada según el atributo (tipo String) del objeto PC que quisiéramos definir para pasarle un valor específico.

Método principal:

```java
/**
 * Método String query().
 * Convierte en String lo que se ha filtrado utilizando parámetros RegEx.
 * @param pattern
 * @param s
 * @param grup
 * @return matcher.group(grup) || mensaje no encontrado;
 */
public static  String query(String pattern, String s, int grup){
   Pattern patt = Pattern.compile(pattern);
   Matcher matcher = patt.matcher(s);
   if (matcher.find()) {
       return matcher.group(grup);
   } else {
       return "No trobat";
   }
}
```

Métodos que entregan el patron, string y group (ejemplo para ficheros .txt en lshw)

```java
/**
 * Método nomPCLS()
 * Devuelve el nombre de un PC en función de un nombre de archivo (ls)
 * @param String nomfitxer
 * @return String utilizando para filtrar método query
 **/
private static String nomPCLS(String nomfitxer){
    String firstCapture;
    int convertedInt;

    firstCapture=RegEx.query("ls_.*?(.*)_(.*?(\\d*)) -", nomfitxer, 3);
    convertedInt=Integer.parseInt(firstCapture);
    return String.format("%02d", convertedInt);

}
```

```java
/**
 * Método String nomAulaLS().
 * Devuelve el nombre del aula basado en el nombre estandarizado del archivo ls
 * @param String nomfitxer
 * @return String con el aula obtenido con el método query
 **/
private static String nomAulaLS(String nomfitxer){
   return RegEx.query("ls_.*?(.+)_", nomfitxer, 1).toUpperCase();
   /*Si quisiéramos COGER solo el número del aula comentaríamos lo de arriba y descomentaríamos lo siguiente
   (por ahora no lo pongo ya que hay un aula E1 y puede ser importante la letra):*/
   //return RegEx.query("ls_.*?(\\d+)_", nomfitxer, 1);
}
```

```java
/**
 * Método String obtenirModelLS().
 * Extrae modelo desde la descripción devuelta por ls.
 * @param String descripción
 * @return String con la información del producto obtenido con método query.
 **/
   private static String obtenirModelLS(String descripcion){
   return RegEx.query("(product)o?: (.*)([\\w\\W].*){4,6}\\n\\W*(anchura: | width: ){1}",descripcion,2);
}
```

7. PROCESAR TODOS LOS FICHEROS DE UN DIRECTORIO

Para procesar todos los ficheros de un directorio creamos la clase ProcessarFitxers.java (luego la hemos eliminado pero fue la manera inicial de abordar este caso) y comenzamos a escalar su complejidad como se presenta a continuación:

7.1 Procesar sólo un fichero .txt

Primero hice pruebas con los atributos para trabajar con un sólo objeto tipo File para:

1. Poder leer un archivo con extensión .txt
2. Sacar información específica (Aula, Marca, Modelo, Nro. de Serie)
3. Mostrar la información específica por pantalla

→→

```
public class ProcessarFitxers {
    //atributos para trabajar con un objeto File tipo Directorio
    String dirPath;//string con la ruta
    File dir=new File(dirPath);//cuando sea un directorio se crea un obj
    File[] fitxerTxt=dir.listFiles();//array de fitxers .txt

    //atributos para trabajar con objeto File tipo txt uno a uno
    private static File f;
    private static String nomfitxer;
```

Trabajé con un main para hacer pruebas

1. Para dar un valor al atributo de tipo File f:
    a. llamamos a un método que nos permita obtenerlo (archivo tipo .txt)

2. Para obtener datos específicos del atributo:
    a. llamamos a un método que filtra los campos que necesitamos rescatar del archivo tipo .txt siempre y cuando entreguemos un archivo como parámetro de entrada para que pueda procesarlo.

```
public static void main(String[] args) {
    f = obtenerArchivo();
    leerFiltrar(f);
}
```

El método obtenerArchivo() opera de la siguiente manera:

1. Pide ingresar por teclado la ruta del archivo
2. Retorna un objeto File que es el archivo que se procesará

```java
public static void main(String[] args) {
    f = obtenerArchivo();//valor nuevoArchivo
    leerFiltrar(f);//parámetro de entrada f=nuevoArchivo
}

/**
 * Method obtenerArchivo. Para obtener el archivo. Pide ingresar por teclado la
 * ruta del objeto File al que se desea acceder.
 *
 * @param empty.
 * @return a File object.
 */
public static File obtenerArchivo() {
    File nuevoArchivo;
    String path;
    Scanner sc;
    System.out.println("Escriu la ruta de l'arxiu que vols processar");
    sc = new Scanner(System.in);
    path = sc.nextLine();
    nuevoArchivo = new File(path);
    sc.close();
    return nuevoArchivo;
}
```

El método leerFiltrar() opera de la siguiente manera:

1. Recibe el cómo parámetro de entrada un objeto de tipo File.
2. Para obtener el sólo el nro de Aula y nro de PC:
    a. Al atributo de tipo String nomfitxer le damos el valor del nombre del archivo como parámetro de entrada utilizando el método .getName().
    b. Llamámos al método nomAulaPc() y le entregamos como parámetro de entrada nomfitxer así podremos extraer sólo lo que necesitamos.
3. Creamos una variable de tipo Scanner para poder leer el contenido del archivo .txt .
4. utilzamos una condición para filtrar las dos fuentes de archivos .txt con la ayuda del método .getParent()
    a. Directorio lshw
    b. Directorio dmidecode
5. Y también utilizamos otras condiciones para filtrar campos específicos de estos archivos .txt

```java
    /**
     * Method leerFiltrar.
     * Lee un archivo.
     * Filtra Aula, nro. PC, Marca, Modelo, nro. de Serie.
     * Imprime campos: Aula, nro. PC, Marca, Modelo, nro. de Serie.
     * @param f2 input tipo file para ser procesado.
     * @return void.
     */
    public static void leerFiltrar(File f2) {
        try {
            nomfitxer = f2.getName();
            // función para obtener número de Aula y número de PC
            nomAulaPC(nomfitxer);
            // procesar texto
            Scanner sc = new Scanner(f2);
            // si el padre del archivo contiene lshw
            if ((f2.getParent()).contains("lshw")) {
                while (sc.hasNext()) {
                    String lector = sc.nextLine();
                    // condiciones para rescatar modelo, marca y nro. de serie

                                        ...

                    // si la carpeta es dmidecode
                } else if ((f2.getParent()).contains("dmidecode")) {
                    while (sc.hasNext()) {
                        String lector = sc.nextLine();
                        // condiciones para rescatar modelo, marca y nro. de serie

                                        ...

            }//fin del bloque try
            sc.close();// cerramos scanner
        }catch (IOException e) {
            System.out.println(e.toString());
        }
```

Prueba de ejecución con ruta a fichero .txt en directorio lshw

```
Problems  Javadoc  Declaration  Console
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java  (23-05-2021
Escriu la ruta de l'arxiu que vols processar
/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/lshw/ls_a22_PC15 - Aleix Samper Colomo.txt
AULA 22 | PC15
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XW0
```

Prueba de ejecución con ruta a fichero .txt en directorio dmidecode

```
Problems  Javadoc  Declaration  Console
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java  (23-05-2021 16:43:
Escriu la ruta de l'arxiu que vols processar
/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/dmidecode/a22_pc03 - Àlex Pasalamar Serrano.txt
AULA 22 | pc03
Marca:  HP
Modelo:  HP ProBook 450 G6
Nro. de Serie:  5CD9339XVM
```

7.2 Procesar todos los ficheros .txt de un Directorio

Ahora que hemos conseguido 1) procesar un fichero ingresando su ruta, 2) obtener y 3) imprimir determinados campos, necesitamos **ejecutar esta misma acción varias veces**, pero **ahora debemos ingresar sólo la ruta del directorio padre** de varios tipos de ficheros .txt.

Para tener una idea de lo que teníamos que hacer definimos los siguientes atributos para trabajar con un objeto File que fuese un Directorio:

1. Uno de tipo String para guardarnos la ruta del directorio: **`dirPath`**
2. Uno de tipo File para crear el atributo del directorio: **`dir`**
3. Un Array de tipo File para guardarnos la lista de ficheros txt que encontremos dentro del directorio **`dir`**: **`fitxerTxt`**

La idea es la siguiente:

```
→→  public class ProcessarFitxers {
        //atributos para trabajar con un objeto File tipo Directorio
        String dirPath;//string con la ruta
        File dir=new File(dirPath);//cuando sea un directorio se crea un obj
        File[] fitxerTxt=dir.listFiles();//array de fitxers .txt
```

Pero en la práctica debemos declararlos así:

```
21  public class ProcessarFitxers {
22      // atributos para trabajar con un objeto File tipo Directorio
23      private static String dirPath;// string con la ruta
24      private static File dir;// cuando sea un directorio se crea un obj.
25      private static File[] fitxerTxt;// array de fitxers .txt
26
27      //atributo para trabajar con objeto file
28      private static String nomfitxer;
```

7.2.1 Registro Errores

A continuación se presentan errores en la concepci´n

Prueba en main, utilizamos una sentencia try-catch; en la sentencia try enumeramos una serie de tareas:

1. Asignamos el valor al atributo **`dirPath`** llamándo al método **`demanarRuta()`**
2. Asignamos el valor al atributo objeto tipo File **`dir`** con el método **`crearDirectorio()`** cuyo parámetro de entrada es el atributo **`dirPath`**.
3. Asignamos los valores al array **`fitxerTxt`** entregando la lista de elementos del directorio **`dir`** utilizando el método **`.listfiles()`**.

```java
public static void main(String[] args) {
    try {
    dirPath = demanarRuta();// 1. método para ingresar path por teclado
    dir = crearDirectorio(dirPath);// 2. inicializar valor de objeto File cuando es un directorio
    fitxerTxt = dir.listFiles();//3. llenamos el array con la lista de ficheros en el directorio

    //4. ordenamos alfabéticamente los archivos
    Arrays.sort(fitxerTxt);
    //5.informamos por pantalla que sucederá
    System.out.println("Directori "+dir.getName()+". Llistant fitxers...");
    System.out.println(" ");

    //6. Imprimir la lista de los ficheros dentro del directorio dado
    for (int i = 0; i < fitxerTxt.length; i++) {
        System.out.println("["+(i+1)+"] "+fitxerTxt[i].getName());
    }
    System.out.println("...\nfin lista.\n");


    //7. llamar método processarDirectori()
    processarDirectori(fitxerTxt);
    //el método debería imprimir los campos específicos por cada fichero en el array

    }catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

El método demanarRuta() opera de la siguiente manera:

1. Pide ingresar por teclado la ruta del archivo
2. Retorna un String con la ruta que se guardará en el atributo **`dirPath`**

```java
private static String demanarRuta() {
    // scanner para ingresar la ruta del directorio a escanear
    Scanner sc = new Scanner(System.in);
    String path;
    System.out.println("Escriu la ruta del directori que vols processar");
    path = sc.nextLine();
    sc.close();
    return path;
}
```

El método crearDirectorio() opera de la siguiente manera:

1. Necesita como parámetro de entrada un String cuyo valor es el del atributo **`dirPath`**
2. Crea un nuevo objeto tipo File con la ruta entregada por el parámetro de entrada.
3. Revisa si la ruta es un directorio o fichero.
4. Si es un directorio retorna un objeto tipo File que se guardará en el atributo **`dir`**

```java
private static File crearDirectorio(String dirPath2) {
    File nouDir = new File(dirPath2);
    if(nouDir.isFile()) {
        System.out.println("Ruta no vàlida. Ha de ser la ruta d'un directori");
    }else if (nouDir.isDirectory()) {
        System.out.println("procesando...\n ");
        System.out.println(nouDir.toString() + " es un directorio\n ");
    }
    return nouDir;
}
```

en el main...

```java
dir = crearDirectorio(dirPath);// 2. inicializar valor de objeto File cuando es un directorio
fitxerTxt = dir.listFiles();//3. llenamos el array con la lista de ficheros en el directorio
```

y luego llenamos el array **`fitxerTxt`** con los ficheros del directorio **`dir`**

El Método **`processarDirectori()`** funciona de la siguiente manera:

1. Tiene como parámetro de entrada un array de tipo File que contiene los ficheros .txt de un directorio
2. Recorre el array de tipo File que contiene los ficheros .txt
3. Imprime en la posición del archivo dentro del array.
4. Llama al método **`leerFiltrar()`** que extrae e imprime información específica.

```
private static void processarDirectori(File[] fitxerTxt2) {
    System.out.println("Extraient dades...");
    for (int j = 0; j < fitxerTxt2.length; j++)
    {
            File nuevoArchivo=fitxerTxt2[j];
            System.out.println("\n[Fitxer "+(j+1)+"]");
            leerFiltrar(nuevoArchivo);
            }
    }
```

Prueba de ejecución

```
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Hon
Escriu la ruta del directori que vols processar
/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/lshw
procesando...

/Users/Daniela/eclipse-workspace/comprova_portatils_feb2021/lshw es un directorio

Directori lshw. Llistant fitxers...

[1] ls_22Aula_PC7 - Laia Girbes Capafons.txt
[2] ls_A22_PC02 - Sergi Garcia Porres.txt
[3] ls_A22_PC05 - Oriol Román Masdeu.txt
[4] ls_A22_PC09 - Jeremie Montealegre Montealegre.txt
[5] ls_A22_PC10 - Adam Rkouni Mata.txt
[6] ls_A22_PC13 - Oriol Román Masdeu.txt
[7] ls_A22_PC14 - Sergi Garcia Porres.txt
[8] ls_A22_PC16 - Jeremie Montealegre Montealegre.txt
[9] ls_A25_PC01 - Oriol Román Masdeu.txt
[10] ls_A25_PC07 - Jeremie Montealegre Montealegre.txt
[11] ls_A25_PC08 - Hatim El Boukhari.txt
[12] ls_A25_PC10 - Laia Girbes Capafons.txt
[13] ls_A25_PC11 - Sergi Garcia Porres.txt
[14] ls_A25_PC12 - Oriol Román Masdeu.txt
[15] ls_A25_PC15 - Laia Girbes Capafons.txt
[16] ls_A25_PC16 - Hatim El Boukhari.txt
[17] ls_A25_PC6 - Laia Girbes Capafons.txt
[18] ls_A25_PC9 - Adam Rkouni Mata.txt
[19] ls_E1_PC05 - Sergi Garcia Porres.txt
[20] ls_a22_PC06 - Aleix Samper Colomo.txt
[21] ls_a22_PC15 - Aleix Samper Colomo.txt
[22] ls_a22_pc03 - Maria Merino Sanjuán.txt
[23] ls_a22_pc03 - Àlex Pasalamar Serrano.txt
[24] ls_a22_pc11 - Àlex Pasalamar Serrano.txt
[25] ls_a22_pc12 - Adam Rkouni Mata(1).txt
[26] ls_a22_pc12 - Adam Rkouni Mata(2).txt
[27] ls_a22_pc12 - Adam Rkouni Mata(3).txt
[28] ls_a22_pc12 - Adam Rkouni Mata.txt
[29] ls_a25_PC03 - Aleix Samper Colomo.txt
[30] ls_a25_PC13 - Aleix Samper Colomo.txt
[31] ls_a25_pc02 - Àlex Pasalamar Serrano.txt
[32] ls_a25_pc04 - Àlex Pasalamar Serrano.txt
[33] ls_a25_pc14 - Àlex Pasalamar Serrano.txt

...fin lista.
```

```
<terminated> ProcessarFitxers [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/

Extraient dades...

[Fitxer 1]
AULA
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339X8L

[Fitxer 2]
AULA 22 | pc02
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XW1

[Fitxer 3]
AULA 22 | pc05
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XTS

[Fitxer 4]
AULA 22 | pc09
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XVS

[Fitxer 5]
AULA 22 | pc10
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XTK

[Fitxer 6]
AULA 22 | pc13
Modelo:  HP ProBook 450 G6 (8AB02ES#ABE)
Marca:  HP
Nro. de Serie:  5CD9339XVD
```

7.3 Crear un nuevo fichero .txt con la lista de archivos procesados.

Ahora que ya conseguimos:

1. Procesar los ficheros dentro de un directorio.
2. Filtrar información específica.
3. Imprimir por pantalla la información específica.

Necesitamos:

1. Crear un nuevo fichero .txt
2. Poner la información específica en el fichero que se ha creado.

7.3.1 Consideraciones

Aunque se ha conseguido procesar, leer y filtrar se debe reestructurar la lógica de clases y funciones para:

1. **Crear un objeto de cada PC** a partir de la información específica extraída de los ficheros .txt dentro de los directorios dmidecode y lshw.
2. **Manipular el/los objeto/s creado/s** y sus atributos través de otros métodos, como por ejemplo:
    - **Escribir la información** de cada objeto dentro de un fichero .txt
    - **Crear un nuevo objeto** con los campos ya definidos e introducirlo al fichero .txt
    - **Modificar la información** de un objeto existente y reemplazar dicha información en el fichero .txt que hemos escrito esta info.
    - **Borrar un objeto** y actualizar dicha información en el fichero .txt que hemos escrito.
    - **Ordenar/filtrar + reescribir información** en un nuevo fichero .txt según atributos específicos de los objetos creados.

En base a lo anterior, la estructura inicial del programa será:

```
├── Cas6
    └── src
        └── pkgGestioFitxersInstitut
            ├── Principal.java (main)
            ├── Menus.java
            ├── ConversorPC.java
            ├── RegEx.java
            ├── PC.java
            └── Output.java
```

7.3.2 Clase Principal.java

[ver Clase Principal.java en github]

Esta clase funciona como main, desplegará el menú y además contiene métodos que llaman y ejecutan otros métodos.

Método **`procesarDirectoriosLsDm()`**.

1. Lee los subdirectorios del directorio indicado.
2. Pide ingresar la ruta del directorio padre y lo procesa con el método **`crearDirectorio()`**.
3. Procesa los subdirectorios llamando al método **`procesaDirectorio()`**.
4. Guarda información extraída con el método **`guardarArraylist()`**.

* @param empty
* @return void

```
public static void procesarDirectoriosLsDm() {
    String path;
    Scanner sc = new Scanner(System.in);

    System.out.println("Introdueix el directori que conté les carpetes lshw i dmidecode");
    path = sc.nextLine();

    File parent = crearDirectorio(path);

    File fileLS = new File(parent, "lshw");
    File fileDm = new File(parent, "dmidecode");

    procesaDirectorio(fileLS);
    procesaDirectorio(fileDm);

    Output.guardarArraylist(listaPCsLS, listaPCsDM);
}
```

Método **`crearDirectorio()`**

1. Crea un atributo de tipo File que es un Directorio.
2. Revisa si la ruta es un directorio o fichero.
3. Si es un directorio lo retorna como un atributo tipo File.

* @param dirPath string con la ruta del directorio.
* @return nouDir atributo objeto de File con la ruta del directorio a procesar.

```
public static File crearDirectorio(String dirPath) {
    File nouDir = new File(dirPath);
    System.out.println("comprovant que la ruta sigui un directori ...\n ");
    if (nouDir.isFile()) {
        System.out.println("Ruta no vàlida. Ha de ser la ruta d'un directori");
    } else if (nouDir.isDirectory()) {
        System.out.println("\n"+nouDir.toString() +" és un directori.");
    }
    return nouDir;
}
```

Método **`procesarDirectorio()`** [ver clase ConversorPC.java]

1. Al pasarle un directorio como argumento añade al ArrayList correspondiente (según criterio) los objetos PC descritos en los archivos.
   @param File directori
   @return void

```
public static void procesaDirectorio(File directori) {
    //guaradamos en un array los ficheros del directorio a procesar
    File[] fitxers = directori.listFiles();
    //recorre el arrayList idea bucle for each
    for (File f : fitxers) {
        //para lshw
        if (f.getName().startsWith("ls_")) {
            listaPCsLS.add(ConversorPC.convertirArchivoLS(f));
        //para dmidecode
        } else {
            listaPCsDM.add(ConversorPC.convertirArchivoDM(f));
        }
    }
}
```

7.3.3 Clase Menus.java

[ver Clase Menus.java en github]

Clase para desplegar el menú de opciones del programa. Se llama desde la clase principal.

7.3.4 Clase PC.java

[ver Clase PC.java en github]

Clase para construir objetos PC a partir del procesamiento de un Directorio y sus Ficheros.

La Clase ConversorPC() contiene los métodos que convierten texto en objetos tipo PC según los atributos establecidos en esta clase.

7.3.5 Clase ConversorPC.java

[ver Clase ConversorPC.java en github]

Clase para convertir un archivo de texto en un objeto tipo PC. Sigue la siguiente lógica:

1. Desde la clase Principal carga el directorio.
2. Itera en todos los ficheros tipo lshw/dmidecode añadiéndolos al arraylist.
3. Y aquella arraylist es la se exportará/escribirá en un fichero de texto.
4. Contiene métodos que convierten atributos de tipo String en objetos tipo PC.

7.3.6 Clase RegEx.java

[ver Clase RegEx.java en github]

Clase para convertir cada uno de los atributos que formarán un objeto PC (desde su único método) a String. Funciona de la siguiente manera:

1. Desde la clase ConversorPC cargamos el String filtrado por los parámetros query que queremos convertir a String.
2. Esta clase devolverá una variable de tipo String que servirá para asignar valores a los atributos de los Objetos de la clase PC.

7.3.7 Clase Output.java

[ver Clase Output.java en github]

Clase para crear, escribir y guardar un archivo de texto. Desde la clase Principal cargamos un arraylist de objetos PC. Desde esta clase creamos un fichero txt y escribimos los datos del arraylist.

CONCLUSION

At the end of this work I can say that in approaching case of this type I should have had more considerations regarding:

- What type of information I needed to obtain as an output and how I would manage to store that information while task planning the development of this case, it would have saved me time.
- Take into account the way I approached the previous case by creating a program structure and ordering the tasks in a better way, that would have prevented multiple failures and a better perspective by the time to create classes and methods.

However, I would say I have succeeded somehow in developing each of the tasks but the possibility of rethinking code structures and procedures to better modularize a program for file manipulation remains open.

WEBGRAFÍA

1_tractament_basic_fitxers
https://docs.google.com/document/d/1gq_NTEQn8Uf5nhG2UZ2hD6jzR6ujndd2Ar6lqyWP7D8/edit
[Consulta: 24 de abril 2021]

Programació bàsica (ASX) \ Programació (DAM i DAW)
https://ioc.xtec.cat/materials/FP/Materials/IC_S_INF/INF_IC_S_M03/web/html/WebContent/u6/a1/continguts.html
[Consulta: 24 de abril 2021]

File (Java Platform SE 7)
https://docs.oracle.com/javase/7/docs/api/java/io/File.html
[Consulta: 24 de abril 2021]

Lesson: Exceptions (The Java™ Tutorials > Essential Classes)
https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html
[Consulta: 24 de abril 2021]

Reading, Writing, and Creating Files (The Java™ Tutorials > Essential Classes > Basic I/O)
https://docs.oracle.com/javase/tutorial/essential/io/file.html
[Consulta: 24 de abril 2021]

Curso Open Webinars: Java 8 desde Cero
https://openwebinars.net/academia/aprende/java/
[Consulta: todo el tiempo de desarrollo de todo el cas6]

Curso Open Webinars: Java 8 para programadores Java
https://openwebinars.net/academia/aprende/java/
[Consulta: todo el tiempo de desarrollo de todo el cas6]

Top 20 Hilarious Code Comments Written by Coders
https://medium.datadriveninvestor.com/top-20-hilarious-code-comments-written-by-coders-86ba3631b77e
[Consulta: 1 de mayo 2021]

Classes and Objects in Java
https://www.geeksforgeeks.org/classes-objects-java/
[Consulta: 3 de mayo 2021]

Java.io.File Class in Java
https://www.geeksforgeeks.org/file-class-in-java/
[Consulta: 3 de mayo 2021]

Java Programming: Reading from a File - dummies

https://www.dummies.com/programming/java/java-programming-reading-from-a-file/
[Consulta: 3 de mayo 2021]

Java: Interacting with the File System
https://www.dummies.com/programming/java/java-interacting-with-the-file-system/?keyword=java%20file%20class%20&index=3&isSearch=1
[Consulta: 3 de mayo 2021]

Try Statements in Java
https://www.dummies.com/programming/java/try-statements-in-java/
[Consulta: 3 de mayo 2021]

How to Diagram Java Classes with UML
https://www.dummies.com/programming/java/diagram-java-classes-uml/
[Consulta: 3 de mayo 2021]

Gestión de excepciones en Java. Captura con bloques try catch finally.
https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=678:gestion-de-excepciones-en-java-captura-con-bloques-try-catch-finally-ejemplos-resueltos-sencillos-cu00927c&catid=58&Itemid=180
[Consulta: 3 de mayo 2021]

How To Work With Files In Java - Dec 09, 2020
https://www.marcobehler.com/guides/java-files
[Consulta: 3 de mayo 2021]

TutorialsPoint Java - Files and I/O
https://www.tutorialspoint.com/java/java_files_io.htm
[Consulta: 3 de mayo 2021]

Code Conventions for the Java Programming Language: 5. Comments
https://www.oracle.com/java/technologies/javase/codeconventions-comments.html
[Consulta: 17 de mayo 2021]

Bucle for-each en Java con ejemplos | Java desde Cero
https://javadesdecero.es/arrays/bucle-for-each/
[Consulta: 26 de mayo 2021]

Array vs ArrayList in Java
https://www.geeksforgeeks.org/array-vs-arraylist-in-java/
[Consulta: 26 de mayo 2021]

RegExr: Learn, Build, & Test RegEx
https://regexr.com/
[Consulta: 26 de mayo 2021]

Java Regex - Java Regular Expressions
http://tutorials.jenkov.com/java-regex/index.html
[Consulta: 26 de mayo 2021]