

## TRACTAMENT DE FITXERS EN JAVA

Tota la **UF3** està dedicada al tractament de fitxers, que porta implícita la gestió dels errors en temps d'execució, perquè és en l'accés a fitxers on podem trobar-nos més tipus d'errors impredecibles que haurem de saber gestionar. Si no ho has fet encara, revisa la documentació sobre el tractament d'excepcions en **java**.

Podem classificar els fitxers atenent al seu contingut (fitxers de text i fitxers binaris) i al seu mode d'accés (fitxers seqüencials i fitxers d'accés directe).

### ELS FITXERS CLASSIFICATS PEL SEU CONTINGUT

Els fitxers, referint-se al seu contingut i/o codificació, des de sempre s'han categoritzat en dos tipus, els fitxer de **text** i els fitxers **binaris**.

#### Fitxers de text

Els fitxers de text són fitxers que contenen informació textual que l'usuari pot llegir i interpretar. Son aquells que podrem entendre si fem un **cat** (o un **type** en Windows) del fitxer. Estan distribuïts en línies, separades per un caràcter de final de línia que pot ser un codi **13 (CR)** o retorn de carro), un codi **10 (LF)** o alimentació de línia) o els **2**, com passa al S.O. Windows, que el final de línia s'indica pels codis **13,10**. Un fitxer **HTML** o **XML** per exemple és un fitxer de text.

Els bytes que formen els fitxers de text no poden contenir qualsevol valor (de **0** a **255**), i per tant, normalment disposaran d'un caràcter especial per indicar que s'acaba el fitxer (**EOF – End Of File**) que ajuda a la seva gestió, tot i que aquest caràcter podria no existir en alguns fitxers.

#### Fitxers binaris

Els fitxers binaris són aquells en els que els seus bytes poden contenir qualsevol codi de caràcter (de **0** a **255**), i per tant no disposen de cap **EOF** que anunciï el final del fitxer, ja que aquest codi o caràcter especial podria formar part del mateix fitxer. Aquests fitxers poden ser executables, imatges, fitxers de so o vídeo, o en general qualsevol fitxer que no entri a la definició de fitxer de text comentada anteriorment.

Els llenguatges de programació diferencien entre aquests dos tipus de fitxers, disposant d'ordres diferents per cada tipus. Així, obrirem un fitxer en mode **text** o be en mode **binari**. Un fitxer de text, també es pot obrir en mode binari, i tractar els seus caràcters com si fossin **bytes**, però si obrim un fitxer binari en mode **text**, provablement obtindrem un resultat no desitjat.

### ELS FITXERS CLASSIFICATS PEL SEU MODE D'ACCÉS

#### Fitxers seqüencials

Son aquells que per accedir a una determinada posició del fitxer, hem de travessar totes les anteriors. Podem assimilar aquest tipus de fitxers a les antigues cintes de cassette, on per escoltar una determinada cançó, hem de travessar totes les que trobem abans d'aquesta. Aquest mode d'accés, que podem utilitzar tant en fitxers de text o binaris, és útil quan volem per exemple traslladar tot el contingut del fitxer des del disc a la memòria.

#### Fitxers d'accés directe

Son els que podem accedir a qualsevol contingut del fitxer a partir de la seva posició, de forma semblant a com fariem amb un vector. Aquest mode d'accés és útil sobre tot en fitxers binaris, ja que en fitxers de text, degut a la codificació diferent dels caràcters no anglesos, pot resultar complicat accedir a un caràcter determinat dintre del fitxer.

En aquest document es pretén donar una visió detallada de com treballar amb fitxers en **java**. Comprendre el funcionament dels fitxers i els diferents modes de treball dels mateixos és fonamental per poder treballar amb arxius des dels nostres programes. El tractament dels fitxers al **java** és semblant al que fan altres llenguatges. El primer que haurem de fer amb un fitxer serà obrir-lo, treballarem amb ell, i finalment el tancarem.

Les classes que farem servir per treballar amb fitxers estan ubicades al paquet **java.io**, de forma que les haurem d'importar des d'allà. Una forma fàcil d'importar totes les classes d'aquest paquet amb una sola instrucció és: **import java.io.\*;**

## Llegir un fitxer de text caràcter a caràcter

La classe que ens permet llegir fitxers de text caràcter a caràcter és **FileReader**. Més informació a la pàgina: <https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html>

### Class FileReader

```
java.lang.Object
java.io.Reader
java.io.InputStreamReader
java.io.FileReader
```

#### Constructor and Description

```
FileReader(File file)
Creates a new FileReader, given the File to read from.

FileReader(FileDescriptor fd)
Creates a new FileReader, given the FileDescriptor to read from.

FileReader(String fileName)
Creates a new FileReader, given the name of the file to read from.
```

Tot i que hi ha tres constructors, el més normal és que al crear un objecte de la classe, li passem com a paràmetre el **nom del fitxer** que volem obrir (indicant el **path** absolut o relatiu). Un cop obert, el fitxer es referenciarà al nostre programa a partir de l'**objecte de la classe**. El constructor d'aquesta classe assumeix que la codificació per defecte per als caràcters i la grandària per defecte per al buffer son apropiades. Per indicar un valor diferent de la grandària del buffer, i optimitzar així el rendiment, podem fer servir un **BufferedReader** al que passarem un objecte de la classe **FileReader** (es veuran exemples més endavant).

A continuació es mostra un programa en **java** que llegeix un fitxer de **text** caràcter a caràcter i compta els caràcters 'l' (ela minúscula) que conté. El programa fa la lectura sense utilitzar cap buffer (fa servir el buffer per defecte del S.O. equivalent a un cluster), i escriu el temps que triga en comptar totes les eles del fitxer.

```
7 import java.io.*; // Importem les classes que necessitem per treballar amb fitxers
8 public class LlegirCaraCar {
9     public static void main( String args[] ) {
10         long intervalo = System.currentTimeMillis(); // Per comptar el temps d'execució
11         FileReader fReader = null; // Representarà el fitxer dintre del nostre programa
12         int numLs=0; // Nombre de caràcters l (minúscula) al fitxer
13         String nomFitxer = "palabras.txt"; // Nom del fitxer que obrirem des del programa
14         int c; // Caràcter llegit, és de tipus int perquè pot ser -1
15
16         try {
17             // Referenciem el fitxer en disc a partir de l'objecte fReader creat.
18             fReader = new FileReader(nomFitxer);
19             while((c = fReader.read()) != -1 ) {
20                 // Aquí dintre fem el procés amb els caràcters del fitxer
21                 if ((char)c == 'l')
22                     numLs++; // Comptem una 'l' minúscula més al fitxer
23             }
24         } catch( Exception e ) {
25             System.out.println("S'ha produït un error en accedir al fitxer: " + nomFitxer);
26             e.printStackTrace(); // Escrivim el registre del stack on s'ha iniciat l'excepció
27         } finally {
28             System.out.println("Aquest codi s'executa sempre !");
29             try {
30                 if( fReader != null ) // Al tancar el fitxer, també es pot produir un error
31                     fReader.close();
32             } catch( Exception e ) {
33                 e.printStackTrace(); // Escrivim el registre del stack on s'ha iniciat l'excepció
34             }
35         }
36         System.out.println("Temps: " + (System.currentTimeMillis()-intervalo)+ " ms ");
37         System.out.println("Trobades " + numLs + " lletres 'l' en el fitxer " + nomFitxer);
38     }
39 }
```

Més endavant aprendrem a fer **servir try-with-resources** per no haver de tancar els fitxers explícitament

```
Símbolo del sistema
C:\Cole\Curs 2019-20\M03-Programació bàsica\M03-Carles ASIX\UF3\LeerFicheros\LlegirCaracters>java LlegirCaraCar
Aquest codi s'executa sempre !
Temps: 156 ms
Trobades 34036 lletres 'l' en el fitxer palabras.txt
```

El fitxer **palabras.txt** que obre i analitza el programa té una grandària de **775 KBytes**:

```

C:\Cole\Curs 2019-20\M03-Programació bàsica\M03-Carles ASIX\UF3\LeerFicheros\LlegirCaracters>dir palabras.txt
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 7EEE-00B1

Directorio de C:\Cole\Curs 2019-20\M03-Programació bàsica\M03-Carles ASIX\UF3\LeerFicheros\LlegirCaracters
08/03/2018  13:18                793.589 palabras.txt
               1 archivos                793.589 bytes
  
```

Aquest programa s'ha executat sobre un portàtil amb **SSD**, i per tant la lectura del disc és força ràpida. Com ja sabeu, la grandària mínima d'un arxiu al disc és d'un **clúster**, que és la unitat d'espai de disc més petita que el sistema operatiu permet gestionar. El nostre programa no fa servir cap buffer intern per agilitzar l'accés al fitxer, i per tant, aquest es llegeix des del disc en blocs de **1 clúster**. En aquest portàtil, la grandària del clúster és de **1 Kbyte**, de forma que el contingut sencer d'aquest arxiu, el recollim des del programa en **775** accessos al disc. Un buffer intern ens ajudaria a reduir el temps d'execució del nostre programa per a fitxers grans, ja que tot i ser un **SSD**, l'accés al disc és més lent que a la memòria. Amb un buffer, l'accés al disc es fa en blocs de la grandària del buffer, i podem recollir varis clústers del disc simultàniament (típicament tots ells, per a fitxers no exageradament grans).

### Class **BufferedReader**

java.lang.Object  
java.io.Reader  
java.io.BufferedReader

#### Constructor and Description

**BufferedReader**(Reader in)

Creates a buffering character-input stream that uses a default-sized input buffer.

**BufferedReader**(Reader in, int sz)

Creates a buffering character-input stream that uses an input buffer of the specified size.

La classe **BufferedReader** té dos constructors. En un d'ells, no cal indicar-li la grandària del buffer, i farà servir la grandària més convenient. Aquest constructor és el que utilitzarem d'entrada. El codi anterior canviarà lleugerament, s'indica la nova versió:

```

7  import java.io.*; // Importem les classes que necessitem per treballar amb fitxers
8  public class LlegirCaraCarBuffer {
9      public static void main( String args[] ) {
10         long intervalo = System.currentTimeMillis(); // Per comptar el temps d'execució
11         BufferedReader bReader = null; // CANVIAT - Farem servir un objecte de la classe BufferedReader
12         int numLs=0; // Nombre de caràcters l (minúscula) al fitxer
13         String nomFitxer = "palabras.txt"; // Nom del fitxer que obrirem des del programa
14         int c; // Caràcter llegit, és de tipus int perquè pot ser -1
15         try {
16             // CANVIAT - Creem el buffer i l'associem al fitxer
17             bReader = new BufferedReader( new FileReader("palabras.txt") );
18             while((c = bReader.read()) != -1) {
19                 // Aquí dintre fem el procés amb els caràcters del fitxer
20                 if ((char)c == 'l')
21                     numLs++; // Comptem una 'l' minúscula més al fitxer
22             }
23         } catch( Exception e ) {
24             System.out.println("S'ha produït un error en accedir al fitxer: " + nomFitxer);
25             e.printStackTrace(); // Escriuim el registre del stack on s'ha iniciat l'excepció
26         } finally {
27             System.out.println("Aquest codi s'executa sempre !");
28             try {
29                 if( bReader != null ) // Al tancar el fitxer, també es pot produir un error
30                     bReader.close();
31             } catch( Exception e ) {
32                 e.printStackTrace(); // Escriuim el registre del stack on s'ha iniciat l'excepció
33             }
34         }
35         System.out.println("Temps: " + (System.currentTimeMillis()-intervalo)+ " ms ");
36         System.out.println("Trobades " + numLs + " lletres 'l' en el fitxer " + nomFitxer);
37     }
38 }
  
```

```

C:\Cole\Curs 2019-20\M03-Programació bàsica\M03-Carles ASIX\UF3\LeerFicheros\LlegirCaracters>java LlegirCaraCarBuffer
Aquest codi s'executa sempre !
Temps: 47 ms
Trobades 34036 lletres 'l' en el fitxer palabras.txt
  
```

Fixeu-vos que el temps d'execució ha passat de **156 ms** a **47 ms**, es a dir, el programa s'executa tres vegades més ràpid. Si no indiquem la grandària del buffer, provablement java escull la grandària del buffer ajustada a la grandària del fitxer per optimitzar l'accés al mateix. Ho podem constatar modificant la línia on creem l'objecte de la classe **BufferedReader** de la forma:

```
// CANVIAT - Creem el buffer i l'associem al fitxer
bReader = new BufferedReader( new FileReader("palabras.txt"), 800000 );
while ((c = bReader.read()) != -1) {
```

Es a dir, fem que tot el contingut del fitxer es llegeixi d'un cop a la memòria. En aquest cas, el temps d'execució resulta ser el mateix que quan no especifiquem cap grandària per aquest buffer.

Si escollim una grandària del buffer diferent, per exemple **100.000 bytes** (necessitarem **8** accessos al fitxer per a llegir-ho completament). El temps d'execució s'incrementa de la forma:

Com a conclusió, podem extreure que quan no especifiquem la grandària del buffer, la classe escollirà la que permeti una optimització dels recursos, tant pel que respecte al temps d'execució del programa com de l'espai de memòria ocupada. Com a conseqüència, deixarem a **java** que s'encarregui d'escollir la grandària del buffer més convenient en cada moment.

Els fitxers, internament tenen una referència o **apuntador** que apunta al següent caràcter que es llegirà o s'escriurà al fitxer, i aquest, en cada operació de lectura o escriptura s'actualitza convenientment de forma totalment transparent a l'usuari.

## Altres mètodes de la classe **FileReader**

Aquests mètodes estan definits a la classe **InputStreamReader** que és la classe pare de **FileReader** (**FileReader** hereta d'aquesta).

- **String getEncoding()**

Retorna el nom del sistema de codificació de caràcters que s'ha fet servir al fitxer. Si executem el codi següent pel fitxer que ens ocupa, obtindrem el resultat que s'indica a sota:

```
fReader = new FileReader(nomFitxer);
System.out.println("Joc de caracters del fitxer: " + fReader.getEncoding());
```

Més informació a: <https://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html>

- **int read(char[] buf, int offset, int length)**

Aquest mètode llegeix com a màxim **length** caràcters de l'arxiu (a partir de l'apuntador al següent caràcter a llegir en aquest) i els col·loca al vector de caràcters **buf** a partir de la posició **offset** del mateix. Retorna el nombre de caràcters llegit (normalment **length**, excepte provablement a l'última lectura) o **-1** indicant que intentem llegir més enllà del final del fitxer.

- **int read(char[] buf)**

Igual que l'anterior, però utilitzant **0** com a **offset** i **buf.length** com a **length**.

## Llegir un fitxer de text línia a línia

De vegades, més que els caràcters individuals, ens interessen les línies o paràgrafs d'un fitxer de text. Els paràgrafs acaben amb un retorn de carro (codi **13**), alimentació de línia (codi **10**) o be amb els dos (codis **13** i **10**) segons el sistema operatiu. El Windows fa servir els codis **CR** i **LF** (retorn de carro i alimentació de línia).

```

7 import java.io.*; // Importem les classes que necessitem per treballar amb fitxers
8 public class LlegirLiniaLiniaBuffer {
9     public static void main( String args[] ) {
10         long intervalo = System.currentTimeMillis(); // Per comptar el temps d'execució
11         BufferedReader bReader = null; // Farem servir un objecte de la classe BufferedReader
12         int numLines=0; // Nombre de línies al fitxer
13         String nomFitxer = "palabras.txt"; // Nom del fitxer que obrirem des del programa
14         String linia; // Línia llegida del fitxer
15         try {
16             // Creem el buffer i l'associem al fitxer
17             bReader = new BufferedReader( new FileReader("palabras.txt") );
18             // Llegim les línies del fitxer
19             while((linia = bReader.readLine()) != null )
20                 // Aquí dintre fem el tractament a les línies del fitxer
21                 numLines++; // Comptem una línia més al fitxer
22         } // Podem indicar més excepcions, però per últim indiquem la genèrica
23         catch (FileNotFoundException e) {
24             System.out.println("Error: No s'ha trobat el fitxer: " + nomFitxer);
25             System.out.println(e.getMessage()); // Escriu el missatge en anglès
26         } catch (Exception e) {
27             System.out.println("S'ha produït un error en llegir el fitxer: " + nomFitxer);
28             System.out.println(e.getMessage()); // Escriu el missatge en anglès
29         } finally {
30             System.out.println("Aquest codi s'executa sempre !");
31             try {
32                 if ( bReader != null ) // Al tancar el fitxer, també es pot produir un error
33                     bReader.close();
34             } catch (Exception e) {
35                 System.out.println(e.getMessage()); // Escriu el missatge en anglès
36             }
37         }
38         System.out.println("Temps: " + (System.currentTimeMillis()-intervalo)+ " ms ");
39         System.out.println("El fitxer " + nomFitxer + " te un nombre de línies igual a " + numLines);
40     }
41 }

```

Si executem aquest codi, el resultat serà el que s'indica:

```

C:\WINDOWS\SYSTEM32\cmd.exe
Aquest codi s'executa sempre !
Temps: 63 ms
El fitxer palabras.txt te un nombre de línies igual a 80383

```

En aquest cas, només hem comptat les línies del fitxer, però fixeu-vos que dintre del **while**, podríem haver fet qualsevol altre tipus de tractament. Fixeu-vos també que el temps que triga el programa a executar-se en aquest cas és superior al del cas anterior, tot i que teòricament fa menys coses (l'altre exemple, comptava les lletres 'l' del fitxer, amb lo qual hem de llegir tots els caràcters del fitxer i comparar cada lletra amb una 'l', i si és aquest caràcter, comptar-ho. A priori, sembla que hauria de ser més ràpid llegir les línies, però provablement el que passa és que la funció **readLine()** de la classe **BufferedReader**, que fa servir la classe **FileReader** per llegir el fitxer caràcter a caràcter, per a cada línia, ha d'ajuntar tots els caràcters i retornar una cadena de text, la qual cosa té un cost important de temps.

Hem vist els mètodes més bàsics per llegir d'un fitxer de text, **caràcter a caràcter** i **línia a línia**. No obstant, un fitxer també es pot llegir utilitzant la classe **Scanner** (que ja hem fet servir per llegir del teclat). La classe **Scanner** permet fer una lectura d'alt nivell d'un fitxer, la qual cosa ens permet llegir fitxers amb format molt fàcilment. Anem a veure-ho.



## La classe Scanner per llegir fitxers de text

A l'exemple que s'indica, llegirem un fitxer de text línia a línia, i l'escriurem a la pantalla

```

1  /**
2   * Llegeix un fitxer de text línia a línia fent servir la classe Scanner
3   * Controla l'excepció: FileNotFoundException i altres que es produeixin
4   */
5
6  import java.io.File;                // Importem el que necessitem únicament
7  import java.io.FileNotFoundException;
8  import java.util.Scanner;
9
10 public class ScannerText_1 {
11     public static void main(String[] args) {
12         // Fitxer a llegir, trajectòria absoluta o relativa...
13         // File file = new File("G:\\NetBeansProjects\\classeScanner\\data-ISO.txt");
14         File file = new File("data-ISO.txt"); // Al directori per defecte
15         int n=0; // Indica la línia llegida del fitxer
16         try {
17             // El fitxer data.txt està format UTF-8
18             // Scanner sc = new Scanner(file, "UTF-8");
19             // El fitxer data-ISO.txt està en format ISO-8859-15, no ho indiquem
20             Scanner sc = new Scanner(file); // , "ISO-8859-15");
21             // Mentre quedin coses per llegir dins del fitxer
22             while (sc.hasNextLine()) {
23                 n++;
24                 String line = sc.nextLine();
25                 System.out.println("Linea: " + n + " = " + line);
26             }
27             sc.close();
28         } catch (FileNotFoundException ex) {
29             System.out.println("No s'ha pogut trobar el fitxer: data-ISO.txt");
30             System.out.println(ex);
31         } catch (Exception e) { // Qualsevol altre error
32             System.out.println(e); // Escrivim el missatge d'error en anglès
33         }
34     }
35 }

```

S'indica el resultat en executar-ho pels fitxers **data-ISO.txt** i **data.txt** (per obrir aquest segon fitxer, hem de canviar el nom de l'arxiu a la línia 14, evidentment).

```

C:\WINDOWS\SYSTEM32\cmd.exe
Linea: 1 = Primer fitxer
Linea: 2 = amb el que treballarem.
Linea: 3 = Anirem llegint
Linea: 4 = línia a línia
Linea: 5 = i mostrarem
Linea: 6 = el resultat

```

```

data-ISO.txt: Bloc d...
Archivo Edición Formato Ver Ayuda
Primer fitxer
amb el que treballarem.
Anirem llegint
línia a línia
i mostrarem
el resultat

```

Si al **data.txt** no indiquem la codificació que presenta (**UTF-8**), el resultat és el que s'indica a sota a la esquerra. Indicant la codificació el resultat apareix a la dreta. El fitxer té el mateix contingut que el **data-ISO.txt**, però amb una codificació diferent pels caràcters.

```

C:\WINDOWS\SYSTEM32\cmd.exe
Linea: 1 = Primer fitxer
Linea: 2 = amb el que treballarem.
Linea: 3 = Anirem llegint
Linea: 4 = lÃ- nia a lÃ- nia
Linea: 5 = i mostrarem
Linea: 6 = el resultat

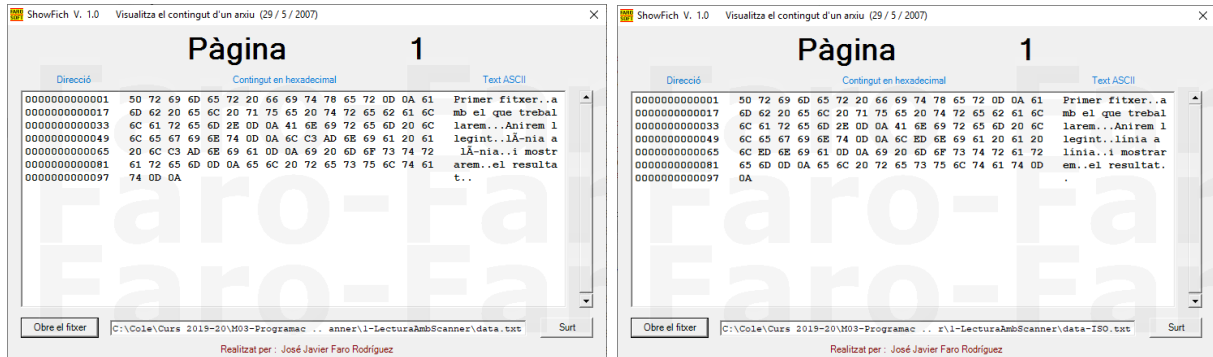
```

```

C:\WINDOWS\SYSTEM32\cmd.exe
Linea: 1 = Primer fitxer
Linea: 2 = amb el que treballarem.
Linea: 3 = Anirem llegint
Linea: 4 = línia a línia
Linea: 5 = i mostrarem
Linea: 6 = el resultat

```

A continuació es mostren les diferents codificacions dels dos fitxers (**data.txt** a l'esquerra i **data-ISO.txt** a la dreta). Fixeu-vos també que entre línia i línia, apareixen els caràcters **0D 0A**, que son els codis **13** i **10** (expressats en hexadecimal), i que corresponen als codis **CR-Carriage Return** (retorn de carro) i **LF-Line Feed** (alimentació de línia). Com a conclusió, aquests son fitxers que han estat creats per al S.O. **Windows**.



Fixeu-vos també que els codis de la majoria dels caràcters son iguals en els dos sistemes de codificació analitzats. Només canvien els codis dels caràcters que no existeixen a l'alfabet anglès (lletres accentuades, ñ, Ñ, ç, Ç, etc...).

## Temps d'execució del programa

Com a complement, anem a fer un programa per llegir amb la classe **Scanner** les línies del fitxer **paraules.txt**. El programa simplement comptarà les línies del fitxer. Després analitzarem el temps que triga a executar-se i el compararem amb l'exemple vist.

```

7 import java.io.*;           // Importem les classes que necessitem per treballar amb fitxers
8 import java.util.Scanner;   // Importem la classe Scanner
9 public class LlegirLiniaScanner {
10     public static void main( String args[] ) {
11         long intervalo = System.currentTimeMillis(); // Per comptar el temps d'execució
12         int numLines=0;      // Nombre de línies al fitxer
13         String linia;        // Línia llegida del fitxer
14         String nomFitxer = "palabras.txt"; // Nom del fitxer que llegirem
15         try {
16             Scanner sc = new Scanner(new File(nomFitxer),"UTF-8"); // Indiquem la codificació
17             // Llegim les línies del fitxer
18             while (sc.hasNextLine()) {
19                 // Aquí dintre fem el tractament a les línies del fitxer
20                 numLines++; // Comptem una línia més al fitxer
21                 linia = sc.nextLine();
22             }
23             sc.close(); // Tanquem el fitxer
24         } catch (Exception e) {
25             System.out.println("S'ha produït un error en llegir el fitxer: " + nomFitxer);
26             System.out.println(e.getMessage()); // Escriu el missatge en anglès
27         }
28         System.out.println("Temps: " + (System.currentTimeMillis()-intervalo)+ " ms");
29         System.out.println("El fitxer " + nomFitxer + " te un nombre de línies igual a " + numLines);
30     }
31 }

```

I si executem el programa veurem:

C:\WINDOWS\SYSTEM32\cmd.exe

```

Temps: 415 ms
El fitxer palabras.txt te un nombre de línies igual a 80383

```

Com es pot apreciar, l'execució del programa triga molt més que el programa equivalent amb el mètode **readLine()** de la classe **BufferedReader**, unes sis vegades més.

Un altre detall, és que si no indiquem el sistema de codificació, en executar obtenim:

```

Temps: 209 ms
El fitxer palabras.txt te un nombre de línies igual a 9970

```

S'ha trobat un EOF  
incorrecte al fitxer

## Missatges d'error en l'accés als fitxers

Hem comentat que el tractament estructurat dels errors esdevé inevitable quan accedim a fitxers. Aquí comentarem les diferències entre els possibles missatges d'error que podem utilitzar als nostres programes (utilitzant els missatges d'error per defecte del sistema). Alguns d'aquests s'han vist als exemples anteriors. Els programes s'han executat en un S.O. Windows 10, en Linux o MAC, els resultats podrien ser diferents, evidentment. El missatge d'error en tots els exemples és que el fitxer no s'ha trobat.

- Escriure directament l'objecte de la classe **Exception** (equivalent a **e.toString()**).

```

} catch( Exception e ) {
    System.out.println(e); // Escriu el missatge d'error
run:
java.io.FileNotFoundException: numlong.txt (El sistema no puede encontrar el archivo especificado)
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Fer servir la funció de la classe **getMessage()**.

```

} catch( Exception e ) {
    System.out.println(e.getMessage()); // Escriu el missatge d'error
run:
numlong.txt (El sistema no puede encontrar el archivo especificado)
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Escriure directament la situació del **Stack** a partir de l'objecte de la classe **Exception**.

```

} catch( Exception e ) {
    e.printStackTrace(); // Escriu el missatge d'error
run:
java.io.FileNotFoundException: numlong.txt (El sistema no puede encontrar el archivo especificado)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.util.Scanner.<init>(Scanner.java:611)
    at LlegirLongsScanner.main(LlegirLongsScanner.java:12)
BUILD SUCCESSFUL (total time: 0 seconds)

```

Com es pot apreciar, aquest últim exemple, pot ser útil quan estem depurant el programa, però no quan donem el nostre programa per acabat. Hi ha altres formes de visualitzar l'error que animo a provar a l'alumne. En tot cas, sempre podem capturar l'error i propagar un missatge d'error personalitzat.

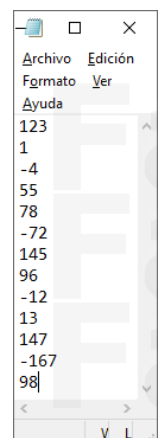
## Llegir valors numèrics d'un fitxer de text

La classe **Scanner** és molt útil per llegir fitxers de la mateixa forma que fem per llegir dades des del teclat. Suposem un fitxer de nom **numlong.txt** que conté valors numèrics enters (positius i negatius), dels quals volem fer la **mitja** i escriure el **nombre de valors**, el **major** i el **menor** d'ells.

A la dreta es mostra un exemple, on els valors estarien cadascú en una línia diferent, però aquests també podrien estar tots a la mateixa línia (separat per espais o tabuladors. O fins i tot aquests valors podrien estar combinats, es a dir, uns quants per línia, separats per un nombre indeterminat d'espais i/o tabuladors. Ja vam comentar que els delimitadors per defecte de la classe **Scanner** són l'espai, el tabulador i el retorn de carro, i quan llegim un enter, un long, un double, etc, saltarem qualsevol nombre d'aquests separadors fins que trobem el text del valor llegit. Si aquest no pot ser convertit al tipus desitjat, es provocarà evidentment una excepció que hauríem de controlar.

Si volem obrir un fitxer des de **NetBeans**, l'hauréu d'indicar el camí on trobar-lo (en altres ID's o des del sistema operatiu el posaríem a la carpeta de l'executable **.class**).

```
String nomFitxer = "C:\\Java\\Fitxers\\numlong.txt";
```





El codi del programa que realitzaria la funció descrita es mostra a continuació:

```

1 import java.io.*;           // Importem les classes necessàries
2 import java.util.Scanner;    // Importem la classe Scanner
3 public class LlegirLongsScanner {
4     public static void main( String args[] ) {
5         int numValors=0;      // Nombre de valors al fitxer
6         long l;              // Valor long llegit del fitxer
7         long suma=0;         // Suma dels valors presents al fitxer. Suposem que no desborda
8         long min=0, max=0;    // Valors min i max llegits del fitxer
9         String nomFitxer = "C:\\Java\\Fitxers\\numlong.txt"; // Nom del fitxer que llegirem
10        try {
11            Scanner sc = new Scanner(new File(nomFitxer));
12            // Llegim el primer valor del fitxer separatament
13            if (sc.hasNextLong()){
14                min = sc.nextLong();
15                max = min;    // el primer valor serà el màxim i el mínim
16                suma = min;
17                numValors++;  // Un valor llegit del fitxer
18            }
19            while (sc.hasNextLong()) {
20                // Aquí dintre fem el tractament als valors long llegits del fitxer
21                numValors++;  // Comptem un valor més al fitxer
22                l = sc.nextLong();
23                if (l>max)
24                    max = l;
25                else if (l<min)
26                    min = l;
27                suma += l;
28            }
29            sc.close();      // Tanquem el fitxer
30            System.out.println("Analitzat el fitxer " + nomFitxer);
31            System.out.println("Valor maxím: " + max + " Valor mínim: " + min);
32            System.out.println("Número de valors: " + numValors);
33            System.out.println("Mitja dels valors: " + ((double) suma / numValors));
34        } catch( Exception e) {
35            System.out.println(e.getMessage()); // Escriu el missatge d'error
36        }
37    }
38 }

```

Si executem aquest codi, i tenim el fitxer ubicat a la carpeta indicada, el resultat serà el següent:

```

run:
Analitzat el fitxer C:\Java\Fitxers\numlong.txt
Valor maxím: 147 Valor mínim: -167
Número de valors: 13
Mitja dels valors: 38.53846153846154
BUILD SUCCESSFUL (total time: 0 seconds)

```

## Llegir fitxers amb format amb la classe Scanner

I si necessitem llegir un fitxer on les dades d'interès no estan separades per espais, retorns de carro o tabuladors?. Llavors disposem del mètode **useDelimiter("Cadena de format")** per especificar quin és el format de les dades que volem llegir. A la cadena de format, indiquem que les dades estan separades per una coma, posant una coma per exemple. Qualsevol literal de la cadena de format farà que aquest sigui retirat de la cadena en la propera lectura sense que intervingui en la informació recollida per aquesta.

**useDelimiter** fa servir les expressions regulars:

[https://en.wikipedia.org/wiki/Regular\\_expression](https://en.wikipedia.org/wiki/Regular_expression) <https://regexone.com/>

Per exemple, un delimitador molt interessant és `"\\s+"` que vol dir qualsevol nombre de repeticions (fins i tot 0) de separadors (espais, tabuladors o retorns de carro).

A sota s'ha indicat un programa d'exemple, i a la dreta un llistat de les expressions regulars més utilitzades.

```
String input = "1 fish 2 fish red fish blue fish";

// \s* means 0 or more repetitions of any whitespace character
// fish is the pattern to find
Scanner s = new Scanner(input).useDelimiter("\\s*fish\\s*");

System.out.println(s.nextInt()); // prints: 1
System.out.println(s.nextInt()); // prints: 2
System.out.println(s.next());    // prints: red
System.out.println(s.next());    // prints: blue

// don't forget to close the scanner!!
s.close();
```

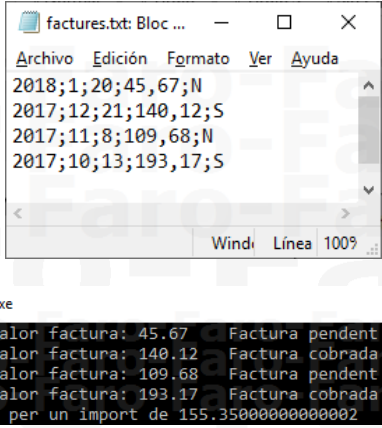
abc...	Letters
123...	Digits
\\d	Any Digit
\\D	Any Non-digit character
.	Any Character
\\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	Characters a to z
[0-9]	Numbers 0 to 9
\\w	Any Alphanumeric character
\\W	Any Non-alphanumeric character
{m}	m Repetitions
{m,n}	m to n Repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional character
\\s	Any Whitespace
\\S	Any Non-whitespace character
^..\$	Starts and ends
(..)	Capture Group
(a(bc))	Capture Sub-group
(.*)	Capture all
(ab cd)	Matches ab or cd

### Exemple d'ús del mètode useDelimiter de la classe Scanner

A continuació, es mostra un programa exemple que llegeix la facturació d'una empresa en un format particular i escriu aquestes dades a la consola de forma més entenedora.

Al fitxer **"factures.txt"**, els diferents camps, estan separats per punt i coma ','. Els tres primers camps son la **data**, que s'indica com **any;mes;dia**. A continuació, s'indica l'import de la factura. Fixeu-vos que **el separador decimal és una coma**. Per últim, figura un camp que indica si la factura ha estat cobrada o no. Si la factura està cobrada, figura una **'S'** en aquest camp, mentre que si no ho està figura una **'N'**.

```
1 import java.io.File;
2 import java.io.FileNotFoundException;
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class LecturaFactures {
7     public static void main(String[] args) {
8         int any,mes,dia;           // Data de la factura
9         double valor;             // valor de la factura
10        int pendants=0;             // Número de factures pendants
11        double sumaPendants=0;      // Import a les factures pendants
12        try {
13            Scanner s = new Scanner(new File("factures.txt"));
14            while (s.hasNextLine()) {
15                String linea = s.nextLine();
16                Scanner sl = new Scanner(linea);
17                sl.useDelimiter("\\s*;\\s*");
18                // Llegim l'any
19                any=sl.nextInt();
20                mes=sl.nextInt();
21                dia=sl.nextInt();
22                System.out.print("Data: "+dia+"/"+mes+"/"+any);
23                valor = sl.nextDouble(); // Llegim l'import
24                System.out.print(" \tValor factura: " + valor);
25                // Llegim uns S o una N
26                char c = sl.next().toUpperCase().charAt(0);
27                if (c == 'S')
28                    System.out.println("\tFactura cobrada");
29                else {
30                    System.out.println("\tFactura pendent");
31                    pendants++;
32                    sumaPendants += valor;
33                }
34            }
35            s.close();
36            System.out.println("Tenim " + pendants + " factures pendants per un import de " + sumaPendants);
37        } catch (FileNotFoundException e) {
38            System.out.println("S'ha trobat l'error següent: " + e.toString());
39        } catch (InputMismatchException e) {
40            System.out.println("Errors en el fitxer.");
41        } catch (Exception e) {
42            System.out.println("Excepció inesperada. L'error reportat pel sistema és:");
43            System.out.println(e);
44        }
45    }
46 }
```



The screenshot shows a text editor window titled 'factures.txt: Bloc ...' containing the following text:

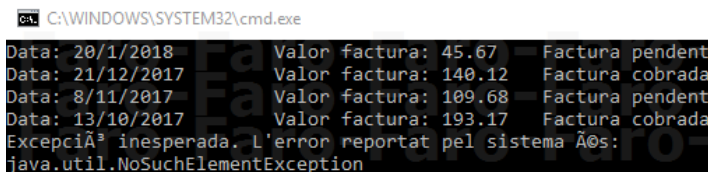
```
2018;1;20;45,67;N
2017;12;21;140,12;S
2017;11;8;109,68;N
2017;10;13;193,17;S
```

Below the text editor is a terminal window titled 'C:\WINDOWS\SYSTEM32\cmd.exe' showing the output of the Java program:

```
Data: 20/1/2018      Valor factura: 45.67      Factura pendent
Data: 21/12/2017     Valor factura: 140.12     Factura cobrada
Data: 8/11/2017      Valor factura: 109.68     Factura pendent
Data: 13/10/2017     Valor factura: 193.17     Factura cobrada
Tenim 2 factures pendants per un import de 155.35000000000002
```

La lectura de fitxers es basa en un coneixement previ de l'estructura de les dades dintre d'aquests. Si el fitxer no presenta el format adequat, es poden produir errors en temps d'execució que seran difícils de recuperar. Per exemple, si el fitxer té un error de sintaxis en un punt mig, la lectura no podrà continuar, i el que és pitjor, podria conduir a interpretacions incorrectes de la informació llegida de l'arxiu.

Podem comprovar que passa a aquest últim programa d'exemple, on vam llegir del fitxer **factures.txt**, si afegim al final del fitxer un parell de línies en blanc. En aquest cas, es produirà una excepció que farà que el programa no pugui acabar correctament:



```
C:\WINDOWS\SYSTEM32\cmd.exe
Data: 20/1/2018 Valor factura: 45.67 Factura pendent
Data: 21/12/2017 Valor factura: 140.12 Factura cobrada
Data: 8/11/2017 Valor factura: 109.68 Factura pendent
Data: 13/10/2017 Valor factura: 193.17 Factura cobrada
Excepció inesperada. L'error reportat pel sistema AOS:
java.util.NoSuchElementException
```

En aquest context, la forma més convenient de llegir d'un fitxer de text, on no tinguem un format absolutament estricte, seria fer-ho caràcter a caràcter o línia a línia. D'aquesta forma, disposem de més eines per detectar possibles errors en el format del fitxer per procedir, per exemple, a descartar les línies amb format inadequat després de proporcionar l'error convenient a l'usuari. Com a criteri general, si detectem un error de format en un fitxer la conseqüència en la majoria dels casos serà que haurem de descartar el fitxer. Pels fitxers importants pels nostres programes, per exemple fitxers de configuració, no seria mala idea disposar d'una còpia de seguretat, generada automàticament, que el programa pogués gestionar de manera transparent a l'usuari si el fitxer principal ha vist compromesa la seva integritat.

Una cosa que no podem mai aconseguir en llegir les dades d'un fitxer, és tornar a demanar les dades vàlides, com podríem fer per exemple, quan llegim dades introduïdes per l'usuari des del teclat. Per tant, hem de tenir una estratègia robusta que pugui fer front a qualsevol error de format que ens puguem trobar (un fitxer es pot corrompre, per exemple).

## Paràmetres passats als constructors

En la majoria dels constructors de les classes que permeten treballar amb fitxers als exemples comentats, hem vist que podem passar tres tipus d'objectes:

- Una cadena amb el nom que té el fitxer que volem obrir al disc. Opcionalment, podem indicar també el camí per arribar al fitxer (**path**). Recordeu que si volem escriure la barra "**\**" en java dintre d'una cadena, l'hem de duplicar. Podem indicar "**/**" al Windows també:

```
FileReader fReader = new FileReader("C:\\Java\\UF3\\exemple.txt");
```

- Un objecte de la classe **File**, que representa el fitxer dintre del nostre programa, i que construïm a partir de la cadena anterior amb el nom de l'arxiu. Això té més avantatges.

```
Scanner s = new Scanner(new File("factures.txt"));
```

- Un descriptor de fitxer o **FileDescriptor**. Els descriptors de fitxers no s'utilitzen directament, sinó mitjançant els fluxos d'entrada **System.in**, de sortida **System.out** i el de sortida estàndard d'error **System.err**. Com ja sabem, el sistema operatiu ens permet redirigir aquests fluxos cap a un fitxer, socket, port, o qualsevol altre canal.

Ja hem utilitzat a la **UF1** un objecte de la classe **Scanner** per llegir del teclat (dispositiu estàndard d'entrada), fent servir el flux **System.in** de la forma:

```
Scanner sc = new Scanner(System.in);
```

## Els mètodes de la classe File

Els objectes de la classe **File**, apart de poder ser utilitzats pels constructors de les classes que fem servir al tractament de fitxers, ens permeten també obtenir informació valuosa dels fitxers. Més informació a: <http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/archivos/file.htm>