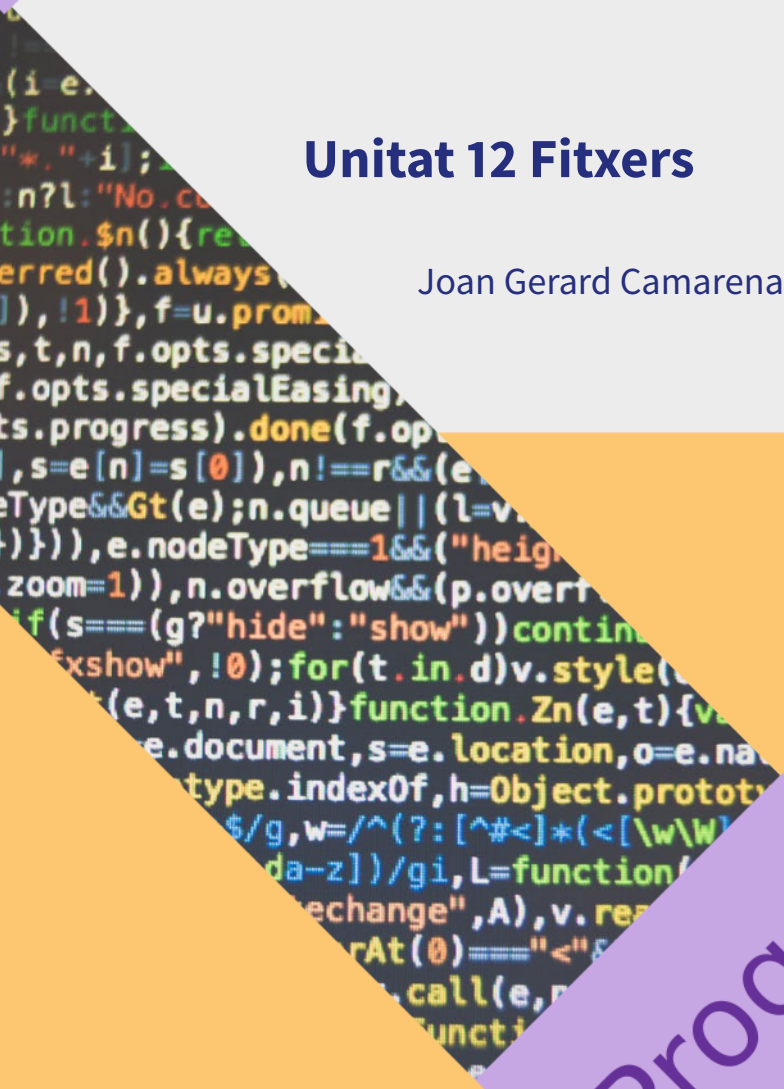
Curs 2019-2020

# Unitat 12 Fitxers

Joan Gerard Camarena Estruch



Programació

IES Jaume II El Just
Tavernes de la Valldigna

# 1. Introducció

Tots els llenguatges de programació tenen alguna forma d'interatuar amb els fitxers. Els algorismes que treballen amb fitxers solen tindre esta forma:

```
    Obrir el fitxer
    Decidir com hem de llegar/escriure les dades
    Llegir/escriure dades
    Tancar el fitxer
```

A banda d'això hem de decidir si hem de crear el fitxer, sobreescriure'l, esborrar-lo, on el creem i demes temes..

## 2. Sistema de fitxers

Abans de vore les operacions de lectura o escriptura sobre fitxers, anem a vore com podem saber les característiques d'un fitxer existent, com per exemple, el directori on està, la grandària del fitxer, si té permisos de lectura, etc i tot això sense tenir ni que llegir ni escriure amb el mateix (com si ferem servir l'explorador 'arxius)

Per a fer això existeix la classe `File` (pròpia de la llibreria de Java). Ens hem de crear un objecte d'eixa classe construint-lo a partir del nom del fitxer. Després, accedirem a les propietats del fitxer amb els mètodes (ja creats) d'eixa classe.

```
// Constructor 1. Passem la ruta absoluta com un String
File f = new File("c:\\kk\\f1.txt");

// Constructor 2. Passe el directori i el nom del fitxer com a String
File f = new File("c:\\kk", "f1.txt");

//Constructor 3
// a) Creem una referéncia a un directori
File d = new File("c:\\kk");
// b) Fem la referéncia al fitxer a partir del directori
File f = new File(d, "f1.txt");

// algunes propietats del fitxer
f.canWrite();
f.exists();
f.length();

// algunes propietats de directoris
f.isDirectory();
f.list();       // torna el llistat de fitxers com array de Strings
f.listFiles()   // torna el llistat de fitxers com array de File
```

> Sobre les rutes: Per a indicar un camí, com que el SO són distints ho farem: sd - LINUX/MAC/ → /home/user/ - WINDOWS → C:\usuarios\andres
>
> Amb `System.out.println(FileSystems.getDefault().getSeparator());` podem saber el tipus de *barra* o *contrabarra* a fer servir com a separador.

## CLASSES IMPLICADES EN LA LECTURA/ESCRIPTURA DE FITXERS DE TEXT

## 3.1 Streams d'entrada

En un fitxer de text pot haver des de dades fins a complexes configuracions com puguen ser els típics fitxers de configuració de Linux. Internament aquestes dades están guardades com a bytes (com tota la información existent als fitxers). És per això que hem de decidir com volem llegar eixa información (per bytes, per caracters o per línies). Segons la nostra desició haurem de escollir l'objecte d'acces a fitxers adequat.

Per a poder llegir les dades d'un fitxer de text, hem de fer 2 coses:

### 3.1.1 Lectura caràcter a caràcter

Executarem el mètode `read()` de `FileReader` tantes voltes com caràcters vulgam llegir. Este mètode retorna un enter, que és el codi del caràcter llegit. O bé, retorna -1 si no hem pogut llegir del fitxer.

Després, el que farem serà promocionar eixe **`int`** a **`char`** per a treballar amb ell.

Exemple: llegim d'un fitxer de text i ho mostrem per pantalla:

```
1 File f = new File("fitxer.txt");
2 FileReader fr = new FileReader(f);
3
4 int c;
5 while ((c = fr.read()) != -1) {
6     System.out.print((char) c);
7 }
8 fr.close();
```

> Nota: la majoria de mètodes de les classes de lectura/escriptura de fitxers (inclosos els constructors) poden donar errors. Per exemple, si no existeix el fitxer, si no té permisos, etc. Per tant, per a evitar possibles finalitzacions incorrectes del programa, haurem de tractar totes les possibles excepcions

### 3.1.2 Lectura línia a línia

Depenent de l'estructura d'un fitxer, a voltes voldrem llegir les seues dades línia a línia (i no caràcter a caràcter). Per a això, hem de crear un objecte de la classe `BufferedReader` associat a un objecte de la classe `FileReader`.

Exemple: llegim d'un fitxer de text, línia a línia, i ho mostrem per pantalla:

```
1 FileReader fr = new FileReader("proves.txt");
2 BufferedReader br = new BufferedReader(fr);
3 String s="";
4 while (  br.ready()  ) {
5     s = br.readLine();
```

```
6       System.out.println(  s );
7   }
8   br.close();
9   fr.close();
```

> Notes: `ready()` ens informa que encara queden dades per llegir al fitxer.

## 3.2 Streams de sortida

Igual que per a la lectura de fitxers, tenim classes anàlogues per a l'escriptura a fitxers

### 3.2.1 Escriptura caràcter a caràcter

Exemple: còpia d'un fitxer origen a un fitxer destí:

```
1  FileReader fr = null;
2  FileWriter fw = null;
3  try {
4      fr = new FileReader("entrada.txt");
5      fw = new FileWriter("sortida.txt");
6      int c;
7      while ( (c=fr.read() )  !=  -1 ) {
8          fw.write( (char)c );
9      }
10 }
```

Problema que pot presentar-se sempre que escrivim fitxers: esta forma d'obrir el fitxer és destructiva. És a dir: si el fitxer *sortida.txt* ja existia abans, l'esborrarà i sobreescriurà damunt.

Solucions:

1. Abans d'obrir-lo per a escriptura, comprovar l'existència del fitxer amb un objecte de la clase `File`.
2. Afegir la nova informació al final del fitxer. Per a fer això cal cridar al constructor del ‘

FileWriter‘ d'una altra forma, passant un boolea com a segon argument.

```
1  fw = new FileWriter(“sortida.txt”,true);
```

El segon argument indica que, si està a true, afegirà dades al fitxer. I si està a false, el fitxer es sobreescriurà.

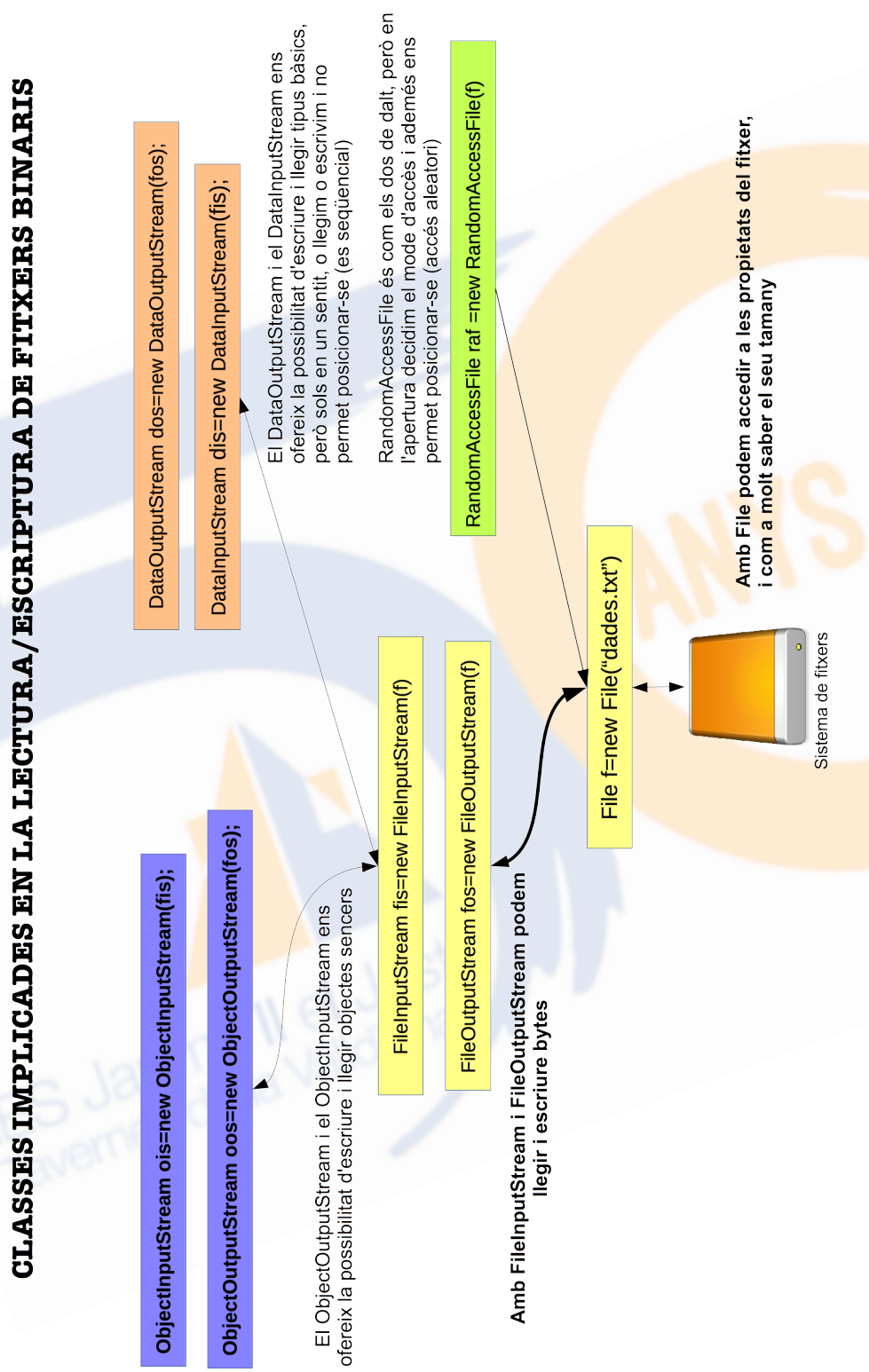
### 3.2.2 Escriptura línia a línia

Exercici Resolt: escriure una poesia a un fitxer de text. Amb `newLine()` escriurem un bot de linia:

```java
FileWriter fw = null;
BufferedWriter bw = null;
try {
    fw = new FileWriter("poesia.txt");
    bw = new BufferedWriter(fw);
    bw.write("No hi havia a València dos amants com nosaltres.");
    bw.newLine();
    bw.write("Feroçment ens amàvem des del matí a la nit.");
    bw.newLine();
    bw.write("Tot ho recorde mentre vas estenent la roba.");
    bw.newLine();
    bw.write("Han passat anys, molts anys; han passat moltes coses.");
    bw.newLine();
    bw.close();
    fw.close();
} catch (IOException ex) {

}
```

## 4. Fitxers binaris



A l'apartat anterior estudiarem com accedir a les dades que existeixen a un sistema de fitxers. Sols estudiarem com accedir a fitxers de text mitjançant la lectura dels caràcters un a un o en blocs. Ara vorem com accedir a dades de qualssevol naturalesa, ja que poden tindre guardats números enters, números amb decimals, etc. Esta informació es guarda als fitxers en format binari. És a dir, abans de guardar-se una dada, es transforma a la seua representació binària.

Per accedir a dades binàries en disc, farem servir les classes `DataInputStream` i `DataOutputStream`, creades a partir de `FileInputStream` i `FileOutputStream`, respectivament. Aquestes classes ens permenten llegir/escriure dades de qualsevol naturalesa, però amb la salvetat que és un accés seqüencial.

És a dir: quan obrim un fitxer, el cursor (punt de lectura o escriptura) se situa al principi del fitxer i, conforme llegim o escrivim, anem avançant. Aquest concepte és l'equivalent a les antigues cintes de audio o video. Quan arribem al final ja no podem retornar (rebobinar, si fos una cinta). El que si ens perment (sols al llegir) es botar-se un determinat nombre de bytes.

| Mètode | Descripció |
|---|---|
| `void close()` | Tanca el fitxer (si estava obert). |
| `long flush()` | Provoca l'escriptura de les dades des de l'stream al fitxer. Sols output. |
| `long size()` | Torna la quantitat de bytes escrits al fitxers. Sols output. |
| `void seek(long pos)` | Posiciona el cursor a la posició `pos` (des de l'inici del fitxer). Sols output. |
| `int skipBytes(int n)` | Bota els següents n bytes del fitxer. Sols input. |
| `readTIPUS` | Llig del fitxer el tipus `tipus`. Sols input. |
| `writeTIPUS` | Escriu al fitxer el tipus `tipus`. Sols output. |
| `mark(int n)` i `reset()` | Serveixen per posar un marcador a la posició n-ésima del fitxer, de manera que quan fem un reset torna el cursor a dita posició. Emula un rebobinat. Sols input.<br>Nota: No tots els `DataInputSream` ho supporten, per això preguntar-ho amb `markSupported()` |
| `int available()` | Torna una previsió dels bytes que queden per llegir, ja que podem botar-nos bytes. Sols input. |

> Notes:
>
> - El `writeUTF` i `readUTF` són per a `Strings`. `WriteUTF` afig 2 bytes abans del text indicant la longitud de la cadena que va a escriure's (així `readUTF`sap el tamany que ha de llegir).
> - La lectura de dades s'haurà de fer amb el mateix ordre que es va fer l'escriptura. Si no, es poden produir inconsistències.

### 4.2 Accès aleatori

L'accés aleatori o directe consistix en què podem accedir a una posició determinada del fitxer sense la necessitat de passar per les anteriors (els vectors també tenen accés directe). La classe `RandomAccessFile` ens permetrà eixe tipus d'accés:

```
1 RandomAccessFile(File file, String mode)
2 RandomAccessFile(String name, String mode)
```

El primer argument és el fitxer, i el segon el mode d'accés, que serà un string amb una combinació de lletres que indiquen:

- `r` → Només lectura
- `rw` → Lectura i escriptura

Si el fitxer ja existeix l'obri, i sino el crea. Per tant no es sobreescriu.

| Mètode | Descripció |
|---|---|
| `void close()` | Tanca el fitxer (si estava obert). |
| `long length()` | Torna la grandària del fitxer. |
| `long getFilePointer()` | Torna la posició del cursor del fitxer. |
| `void seek(long pos)` | Posiciona el cursor a la posició pos (des de l'inici del fitxer). |
| `int skipBytes(int n)` | Bota els següents n bytes del fitxer. |
| `readTIPUS` | Llig del fitxer el tipus TIPUS. |
| `writeTIPUS` | Escriu al fitxer el tipus TIPUS. |

## 5. Fitxers d'objectes

L'inconvenient de guardar la informació en mode binari com hem fet abans és que cal llegir en el mateix ordre en què hem escrit les dades. Ademés, nosaltres ja estem treballant en els objectes com a unitat. Llavors el que ens interesarà és escriure al fitxer els distints objectes i després llegirem objectes.

Per a fer això, els objectes necessiten ser *serialitzats* (ho podem entendre com posar totes les dades d'un objecte en binari i en série, unes darrere d'altres). Simplement, hem d'indicar-ho en la definició de la classe. Així:

```
1 class NomClasse implements java.io.Serializable {
2     //
3 }
```

I en la taula següent tenim les classes que cal utilitzar per a utilitzar els mètodes de lectura i escriptura sobre fitxers d'objectes:

| Classe o Mètode | Descripció |
|---|---|
| `ObjectOutputStream` | Incorpora els mètodes per a escriure objectes a un fitxer. La classe deu implementar `Serializable` |
| `writeObject(Object o)` | Escriure l'objecte `o` |
| `close()` | Tancar el fitxer |
| `writeTipus(Tipus t)` | Similar als fitxers binaris |
| `ObjectInputStream` | Incorpora els mètodes per a llegir objectes d'un fitxer. La classe deu implementar `Serializable` |
| `(ObjecteDesti)readObject()` | Llig un objecte. Hem de fer un càsting a l'objecte de destí (Persona, Alumne, Cotxe, etc.) |
| `Tipus readTipus()` | Similar als fitxers binaris |
| `skipBytes()` | Bota una certa quantitat de bytes |

Per a escriure objectes al fitxer cridarem al mètode `writeObject()`, passant-li com a paràmetre l'objecte que volem escriure. L'escriptura és seqüencial I destructiva (cada volta es crearà un nou fitxer).

Per a llegir del fitxer d'objectes cridarem al mètode `readObject()`, que ens retorna un objecte però

de la classe `Object`. Per tant, haurem de fer-li un casting per a convertir-lo al tipus d'objecte que estem llegint. Ara bé, abans de llegir objectes, caldrà assegurar-se que en queden per llegir. Per a això, cridarem al mètode `available()` (de la classe `FileInputStream`), el qual retorna la quantitat de bytes que falten per llegir al fitxer.

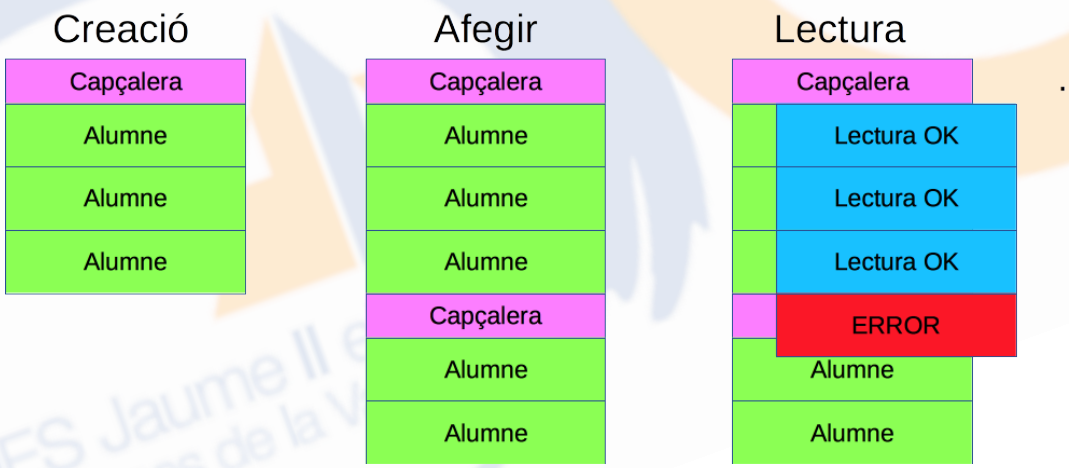
Si el tipus d'Objecte que intentem guardar no implementa la interfície `Serializable`, botarà l'excepció `java.io.NotSerializableException`.

### 5.1 Afegir objectes. Problemàtica

Si volem afegir objectes a un fitxer ja donat (amb certs objectes) hem de fer unes xicotets modificacions, degut al següent problema.

Quan escrivim objectes amb `writeObject()`, la classe `ObjectOutputStream` escriu una capçalera amb metainformació del objecte que va a escriure a continuació. Si volem afegir objectes, l'habitual és obrir el fitxer de nou, amb l'opció `append` a **`true`** i situar-se al final del fitxer i escriure els nous objectes.

Després si intentem llegir el fitxer al qual hem guardat la informació ens trobarem que tenim diverses capçaleres enmig del fitxer i com no sabem a priori la quantitat d'objectes tenim, llavors ens donarà errors el nostre codi, ja que intentem llegir un objecte i estarem llegint una capçalera.




La solució passa per evitar que s'escriga la capçalera dels objectes en futures addicions. Per això hem de fer el següent

```
1  public class MiObjectOutputStream extends ObjectOutputStream
2  {
3      /** Constructor */
4      public MiObjectOutputStream(OutputStream out) throws IOException
5      {
```

```
6           super(out);
7       }
8
9       /** Constructor */
10      protected MiObjectOutputStream() throws IOException,
            SecurityException
11      {
12          super();
13      }
14
15      /** Métode que esciur la capçalera. El redefinim i deixem en blanc
            */
16      protected void writeStreamHeader() throws IOException
17      {
18      }
19  }
```

Ens implementem un ObjectOutputStream propi de manera que redefinim el mètode encarregat d'escriure la capçalera que no faça res. Llavors:

1. Quan creem el fitxer farem servir `ObjectOutputStream`.
2. Quan afegim dades al fitxer farem servir `MiObjectOutputStream`.

### 5.2 Manera de treballar

Amb programes amb gran volum d'informació, les dades inicialment estaran al disc dur dins d'un fitxer. El nostre programa, per a utilitzar i manipular eixes dades, les portarà a memòria principal volcant els objectes del fitxer en un vector d'objectes.

A continuació, les dades podran ser consultades, esborrades, introduir nous objectes, etc.

I, quan volem acabar, volcarem el vector al disc dur, escrivint tots els objectes del vector en el fitxer que teníem.

Com altetrnativa, podem llegir i escriure tots els objectes d'una tacada, donat que, com que un `ArrayList` o vector és un objecte també, podem escriure'l tot sencer amb `writeObject`, en compte del típic bucle que va recorreguent l'array i processant-lo un a un.