



Unió Europea

Fons Social Europeu

L'FSE inverteix en el teu futur



GENERALITAT
VALENCIANA



UF11 - FITXERS

- Teoria -

PROGRAMACIÓ
CFGS DAM

Autor:

Àngel Olmos Giner

segons el material de Carlos Cacho i Raquel Torres

a.olmosginer@edu.gva.es

2022/2023

FITXERS

ÍNDEX DE CONTINGUTS

- 1. INTRODUCCIÓ**
- 2. GESTIÓ DE FITXERS**
 - 1. La classe *File***
 - 2. Rutes absolutes i relatives**
 - 3. Mètodes de la classe *File***
- 3. LECTURA I ESCRIPTURA DE FITXERS**
 - 1. Fitxers orientats a caràcter**
 - 2. Lectura de fitxers (classe *Scanner*)**
 - 3. Escriptura en fitxers (classe *FileWriter*)**
- 4. BONUS TRACK: Creació d'arxius de projecte .JAR**

1. INTRODUCCIÓ



- La principal funció d'una aplicació informàtica és la **manipulació i transformació de dades** (fins ara, mitjançant variables en memòria)
- Una vegada el **programa finalitza, les dades desapareixen**
- Imagineu-vos haver d'introduir les notes de tots els estudiants cada vegada que s'executa el programa per a gestionar-les. No té cap sentit
- Apareix la necessitat de poder registrar les dades en algun suport de memòria externa
- La manera més senzilla d'aconseguir aquest objectiu és **emmagatzemar la informació aprofitant el sistema d'arxius** que ofereix el sistema operatiu
- En aquesta unitat s'expliquen diferents classes de Java que ens permeten **crear, llegir, escriure i eliminar fitxers i directoris**

1. INTRODUCCIÓ



- Normalment, dins d'un sistema operatiu s'espera disposar d'alguna mena d'interfície o explorador per a poder gestionar arxius
- La **manera de gestionar el sistema d'arxius** sol ser molt similar en la immensa majoria dels SOs: **estructura jeràrquica amb carpetes i fitxers**
- La capacitat d'operar amb el sistema d'arxius no és exclusiva de la interfície oferida pel sistema operatiu i **molts llenguatges de programació proporcionen biblioteques que permeten accedir directament als mecanismes interns de gestió del sistema d'arxius**
- És possible crear codi font per a realitzar operacions típiques d'un explorador d'arxius
- Java inclou un conjunt de classes dins del **package *java.io*** mitjançant les quals és possible dur a terme pràcticament qualsevol tasca sobre el sistema d'arxius

FITXERS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. **GESTIÓ DE FITXERS**
 1. **La classe *File***
 2. Rutes absolutes i relatives
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

2. GESTIÓ DE FITXERS

La classe *File*



- ❑ La peça més bàsica per a poder operar amb arxius és la classe *File*
- ❑ No es refereix exactament a un arxiu. La classe *File* **representa una ruta dins del sistema d'arxius**
- ❑ Serveix per a realitzar operacions tant sobre rutes al sistema d'arxius que ja existisquen com no existents
- ❑ Es pot usar **tant per a manipular arxius com directoris** (tant fitxers com carpetes)
- ❑ *File* rep com a argument un *String* corresponent a la ruta sobre la qual es volen dur a terme les operacions

```
File f = new File (String ruta);
```



El format dependrà del Sistema Operatiu

2. GESTIÓ DE FITXERS

La classe *File*



```
File f = new File (String ruta);
```

- ❑ Els sistemes basats en UNIX usen la barra "/" mentre que Windows usa la inversa "\"
 - Exemple de ruta en UNIX: "/usr/bin"
 - Exemple de ruta en Windows: "C:\Windows\System32"
- ❑ **Java permet utilitzar la barra de UNIX "/" per a representar rutes en sistemes Windows → C:/Windows/System32**
- ❑ **Una ruta** és la forma general d'un nom d'arxiu o carpeta, per la qual cosa **identifica únicament la seua localització en el sistema d'arxius**
- ❑ Cadascun dels elements de **la ruta poden existir realment o no, però això no impedeix poder inicialitzar *File***
- ❑ No és fins que es criden els diferents mètodes definits en *File*, o fins que s'escriuen o es lliguen dades, que realment s'accedeix al sistema de fitxers

2. GESTIÓ DE FITXERS

La classe *File*



```
File f = new File (String ruta);
```

- ❑ És important entendre que un objecte representa una única ruta del sistema de fitxers
- ❑ **Per a operar amb diferents carpetes i arxius a la vegada, caldrà crear i manipular diversos objectes**

```
File carpetaFotos = new File("C:/Fotos");  
File unaFoto = new File("C:/Fotos/Foto1.png");  
File otraFoto = new File("C:/Fotos/Foto2.png");
```

← Directori

← Arxius

FITXERS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. **GESTIÓ DE FITXERS**
 1. La classe *File*
 2. **Rutes absolutes i relatives**
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

2. GESTIÓ DE FITXERS

Rutes absolutes i relatives



Una **ruta absoluta** és aquella que es refereix a un element a partir de l'arrel del sistema de fitxers. Per exemple "C:/Fotos/Foto1.png"

- Les **rutes absolutes** es distingeixen fàcilment, ja que el text que les representa comença d'una manera molt característica depenent del sistema operatiu de l'ordinador
 - Windows → nom de la unitat ("C:", "D:", etc.)
 - UNIX → barra "/"

C:\Fotos\Viatges (ruta a una carpeta)

M:\Documents\Unitat11\apartat1 (ruta a una carpeta)

N:\Documents\Unitat11\apartat1\Activitats.txt (ruta a un arxiu)

/Fotos/Viatges (ruta a una carpeta)

/Documents/Unitat11/apartat1 (ruta a una carpeta)

/Documents/Unitat11/apartat1/Activitats.txt (ruta a un arxiu)

2. GESTIÓ DE FITXERS

Rutes absolutes i relatives



- Quin problema veieu a l'ús de rutes absolutes?
- Els programes implementats mitjançant rutes absolutes ... Seran portables?
- Perquè un programa que interacciona amb el sistema de fitxers i utilitza rutes absolutes funcione en qualsevol dispositiu, **serà necessari que en el dispositiu hi haja exactament les mateixes carpetes**, tal com estan escrites en el codi font del programa
- En cas contrari, no funcionarà, ja que les carpetes i fitxers esperats no existiran, i per tant no es trobaran
- **Usar rutes absolutes fa que un programa sempre haja de treballar amb una estructura fixa del sistema d'arxius, exactament igual allà on s'execute**

2. GESTIÓ DE FITXERS

Rutes absolutes i relatives



Una **ruta relativa** és aquella que **no inclou l'arrel** i per això es considera que **part des del directori de treball** de l'aplicació. Aquesta carpeta pot ser diferent cada vegada que s'executa el programa.

- Quan un programa s'executa, per defecte se li assigna una carpeta de treball que sol ser la carpeta des d'on es llança el programa
- **Una ruta relativa** mai indica l'arrel del sistema de fitxers. Directament **comença pel primer element triat dins de la ruta**

```
Viatges  
Unitat11\apartat1  
Unitat11\apartat1\Activitats.txt
```

- Una ruta relativa sempre inclou el directori de treball de l'aplicació com a part inicial malgrat no haver-se escrit
- **El tret distintiu és que el directori de treball pot variar**

2. GESTIÓ DE FITXERS

Rutes absolutes i relatives



File f = new File ("Unitat11/apartat1/Activitats.txt")

Directori de treball	Ruta real
C:/Projectes/Java	C:/Projectes/Java/Unitat11/apartat1/Activitats.txt
X:/Unitats	X:/Unitats/Unitat11/apartat1/Activitats.txt
/Programes	/Programes/Unitat11/apartat1/Activitats.txt

- Aquest mecanisme **permet facilitar la portabilitat del codi entre diferents dispositius**, ja que només és necessari que els arxius i carpetes estiguen en la mateixa ruta relativa al directori de treball
- **Les rutes relatives permeten crear codi independent del SO**, ja que no és necessari especificar un format d'arrel lligada a un sistema d'arxius concret ("C:", "D:", "/", etc.)

FITXERS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. **GESTIÓ DE FITXERS**
 1. La classe *File*
 2. Rutes absolutes i relatives
 3. **Mètodes de la classe *File***
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



OBTENCIÓ DE LA RUTA

- ❑ **String getParent()** retorna la ruta de la carpeta de l'element referit
 - El *String* resultant és idèntic a la ruta original, eliminant l'últim element
 - Retorna null si la ruta tractada es refereix a la carpeta arrel d'un sistema d'arxius ("C:\", "/")

- ❑ **String getName()** retorna el nom de l'element que representa la ruta, ja siga una carpeta o un arxiu. És el cas invers del mètode getParent()

- ❑ **String getAbsolutePath() ... ??**

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



OBTENCIÓ DE LA RUTA

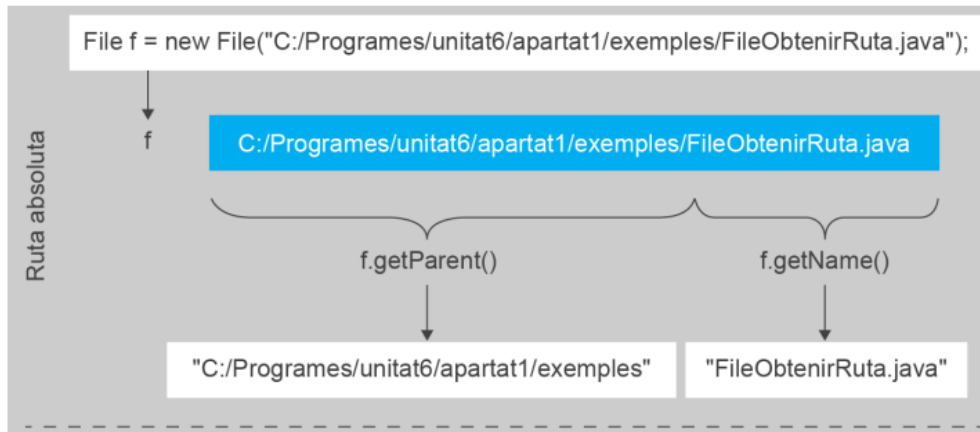
- ❑ **String getParent()** retorna la ruta de la carpeta de l'element referit
 - El *String* resultant és idèntic a la ruta original, eliminant l'últim element
 - Retorna null si la ruta tractada es refereix a la carpeta arrel d'un sistema d'arxius ("C:\\", "/"")

- ❑ **String getName()** retorna el nom de l'element que representa la ruta, ja siga una carpeta o un arxiu. És el cas invers del mètode getParent()

- ❑ **String getAbsolutePath() ... ??** retorna la ruta absoluta

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



OBTENCIÓ DE LA RUTA

Exemple 1:

- Crea 2 objectes *File* utilitzant rutes absolutes: a un directori i a un fitxer
- Crea 2 objectes *File* utilitzant rutes relatives: a un directori i a un fitxer
- Mostra en línies separades la següent informació per cada objecte creat:
 1. La ruta a l'element (directori o fitxer)
 2. El nom de l'element
 3. La seua ruta absoluta

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



OBTENCIÓ DE LA RUTA

Exemple 1:

```
public static void main(String[] args) {
    // Dues rutes absolutes
    File carpetaAbs = new File("/home/aolmos/fotos");
    File arxiuAbs = new File("/home/aolmos/fotos/albanial.jpg");
    // Dues rutes relatives
    File carpetaRel = new File("treballs");
    File fitxerRel = new File("treballs/document.txt");
    // Mostrem les seues rutes
    mostrarRutes(carpetaAbs);
    mostrarRutes(arxiuAbs);
    mostrarRutes(carpetaRel);
    mostrarRutes(fitxerRel);
}

public static void mostrarRutes(File f) {
    System.out.println("getParent() : " + f.getParent());
    System.out.println("getName() : " + f.getName());
    System.out.println("getAbsolutePath(): " + f.getAbsolutePath());
    System.out.println("");
}
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



COMPROVACIONS D'ESTAT

Donada la ruta emprada per a inicialitzar una variable de tipus *File*, **aquesta pot ser que realment existisca dins del sistema de fitxers o no**, ja siga en forma d'arxiu o carpeta:

- ☐ **boolean exists()** comprova si la ruta existeix dins del sistema de fitxers, però no si es tracta d'un fitxer o d'un directori (els fitxers sense extensió es poden confondre amb directoris)
- ☐ **boolean isFile()** comprova la ruta i retorna *true* si existeix i és un fitxer. Retornarà *false* si no existeix, o si existeix però no és un fitxer
- ☐ **boolean isDirectory()** ...

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



COMPROVACIONS D'ESTAT

Exemple 2:

- Crea la carpeta "Temporal" en l'arrel del disc
- Dins, un arxiu anomenat "Document.txt" (pot estar buit) i una carpeta anomenada "Fotos"
- Implementa el següent mètode en Java i utilitza'l per a comprovar l'estat de les carpetes i arxius creats
- Finalment esborra algun dels elements creats i torna a executar el codi Java

```
public static void mostrarEstat(File f) {  
    System.out.println("L'arxiu " + f.getAbsolutePath() + "' existeix? " + f.isFile());  
    System.out.println("La carpeta " + f.getAbsolutePath() + "' existeix? " + f.isDirectory());  
    System.out.println("");  
}
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



COMPROVACIONS D'ESTAT

Exemple 2:

```
public static void main(String[] args) {
    File temp = new File("C:/Temporal");
    File fotos = new File("C:/Temporal/Fotos");
    File document = new File("C:/Temporal/Document.txt");
    System.out.println("'" + temp.getAbsolutePath() + "' existeix? " + temp.exists() + "\n");
    mostrarEstat(fotos);
    mostrarEstat(document);
}

public static void mostrarEstat(File f) {
    System.out.println("L'arxiu '" + f.getAbsolutePath() + "' existeix? " + f.isFile());
    System.out.println("La carpeta '" + f.getAbsolutePath() + "' existeix? " + f.isDirectory());
    System.out.println("");
}
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



PROPIETATS DE FITXERS

El sistema de fitxers d'un sistema operatiu emmagatzema diversitat d'informació sobre els arxius i carpetes que pot resultar útil conèixer: els seus atributs d'accés, la seua grandària, la data de modificació, etc:

- ❑ **long length()** retorna la grandària d'un arxiu en bytes. Aquest mètode només pot ser utilitzat sobre una ruta que represente un arxiu, en cas contrari no es pot garantir que el resultat siga vàlid
- ❑ **long lastModified()** retorna l'última data d'edició de l'element. El resultat es codifica en un únic nombre enter el valor del qual és el nombre de mil·lisegons que han passat des de l'1 de gener de 1970

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



PROPIETATS DE FITXERS

Exemple 3:

- Crea un programa que mostre l'última modificació d'un arxiu i la seua grandària
- Per a fer-ho, utilitza els mètodes adients i la classe *Date*
- Executa el codi sobre un arxiu del teu ordinador
- Crea un nou arxiu buit i executa el codi sobre eixe arxiu
- Canvia el contingut de l'arxiu i torna a executar el codi

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



PROPIETATS DE FITXERS

Exemple 3:

```
public static void main(String[] args) {  
    File document = new File("C:/Temp/Document.txt");  
    System.out.println(document.getAbsolutePath());  
    long milisegons = document.lastModified();  
    Date data = new Date(milisegons);  
  
    System.out.println("Última modificació (ms) : " + milisegons);  
    System.out.println("Última modificació (data): " + data);  
    System.out.println("Grandària de l'arxiu: " + document.length());  
}
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



GESTIÓ DE FITXERS

El conjunt d'operacions més habituals en accedir a un sistema de fitxers d'un ordinador són les vinculades a la seua gestió directa: canviar de nom arxius, esborrar-los, copiar-los o moure'ls:

- ☐ **boolean mkdir()** permet crear la carpeta indicada en la ruta. Retorna *false* si per exemple la ruta és incorrecta, la carpeta ja existeix o l'usuari no té permisos d'escriptura
- ☐ **boolean delete()** esborra l'arxiu o carpeta indicada en la ruta **només si està buida**

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



GESTIÓ DE FITXERS

Exemple 4:

- Analitza el següent programa ... **què passarà? (pensa-ho sense executar-lo)**

```
public static void main(String[] args) {  
    File fotos = new File("C:/Temporal/Fotos");  
    File doc = new File("C:/Temporal/Document.txt");  
  
    boolean mkdirFot = fotos.mkdir();  
  
    if (mkdirFot) {  
        System.out.println("Creada carpeta " + fotos.getName() + "? "  
            + mkdirFot);  
    } else {  
        boolean delCa = fotos.delete();  
        System.out.println("Esborrada carpeta " + fotos.getName() + "? "  
            + delCa);  
        boolean delAr = doc.delete();  
        System.out.println("Esborrat arxiu " + doc.getName() + "? " + delAr);  
    }  
}
```

Assumim que en l'arrel de la unitat no hi ha cap carpeta anomenada "Temporal"

2. GESTIÓ DE FITXERS

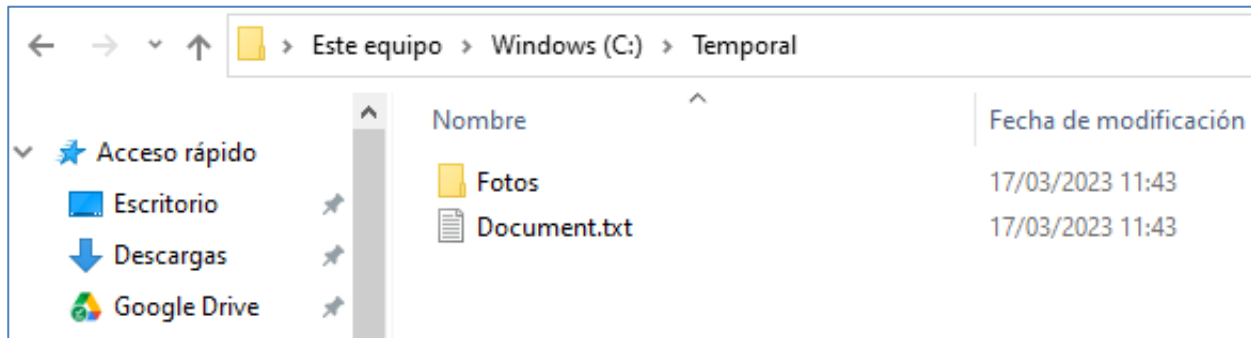
Mètodes de la classe *File*



GESTIÓ DE FITXERS

Exemple 4:

- Crea la carpeta “C:\Temporal” o “/Temporal” i ... **què passarà?**
- Executa el programa vàries vegades fins que compregues el que està passant
- Crea un arxiu “Document.txt” dins la carpeta Temporal ... **què passarà?**



2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



GESTIÓ DE FITXERS

Des del punt de vista d'un sistema operatiu, l'operació de **move** un arxiu o carpeta **no és més que canviar el seu nom des de la seua ruta original fins a una nova ruta destí**

- ❑ **boolean renameTo(File destí)** la seua funció real no és simplement canviar el nom d'un arxiu o carpeta, sinó canviar la ubicació completa. Se li dona com a argument un altre objecte *File* amb la ruta destí
 - Si l'element final de les rutes origen i destinació són diferents, el nom de l'element, siga arxiu o carpeta, canviarà
 - Retorna *false* si alguna ruta és incorrecta, no existeix l'arxiu, etc...
 - **En el cas de carpetes, és possible move-les encara que continguen arxius**

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



GESTIÓ DE FITXERS

Exemple 5:

- Crea la carpeta "C:\Temporal\" o "/Temporal/" si no existeix
- Dins de la carpeta Temporal, crea una carpeta anomenada "Backup" i una altra anomenada "Fotos"
- Dins de la carpeta "Fotos" crea dos documents anomenats "Pic1.png" i "Pic2.png"
- Crea un programa que:
 - Moga la carpeta "C:\Temporal\Fotos\" i el seu contingut a la carpeta "C:\Temporal\Backup\Fotografies\"
 - Moga el fitxer "Pic1.png" de la nova carpeta fins a "C:\Temporal\"

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



GESTIÓ DE FITXERS

Exemple 5:

```
public static void main(String[] args) {  
    File origenD = new File("C:/Temporal/Fotos");  
    File destiD = new File("C:/Temporal/Backup/Fotografies");  
    File origenF = new File("C:/Temporal/Backup/Fotografies/Pic1.png");  
    File destiF = new File("C:/Temporal/Pic1.png");  
  
    boolean res = origenD.renameTo(destiD);  
    System.out.println("S'ha mogut i canviat de nom la carpeta? " + res);  
    res = origenF.renameTo(destiF);  
    System.out.println("S'ha mogut la imatge? " + res);  
}
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



LLISTAT D'ARXIUS

Només en el cas de les carpetes, és possible consultar quin és el llistat d'arxius i carpetes que conté

❑ **File[] listfiles()** retorna un vector de tipus *File* amb tots els elements continguts en la carpeta (representats per objectes *File*)

- La grandària del vector serà igual al nombre d'elements que conté la carpeta
- Si la carpeta no existeix, el valor retornat serà *null*
- L'ordre dels elements mostrats és aleatori (no importa grandària, data, tipus ...)

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



LLISTAT D'ARXIUS

Exemple 6:

- Implementa un programa que mostre per pantalla el contingut del directori "C:\Temporal", especificant si l'element mostrat es un arxiu o d'una carpeta
- Fes proves amb la carpeta plena, buida i si la carpeta no existeix
- Els resultats han de ser com aquests ...

```
run:
Contingut de C:\Temporal és:
La carpeta no existeix
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
Contingut de C:\Temporal és:
La carpeta conté un total de 0 elements
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
Contingut de C:\Temporal és:
La carpeta conté un total de 7 elements
[DIR] Alumnat
[DIR] Backup
[DIR] Examen
[ARX] Examen.txt
[DIR] Exercicis
[ARX] Notes1DAM.xlsx
[ARX] Solucions.zip
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. GESTIÓ DE FITXERS

Mètodes de la classe *File*



LLISTAT D'ARXIUS

Exemple 6:

```
public static void main(String[] args) {
    File dir = new File("C:/Temporal");
    File[] llista = dir.listFiles();
    System.out.println("Contingut de " + dir.getAbsolutePath() + " és:");

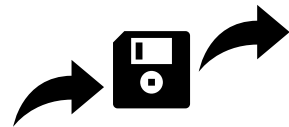
    if (llista == null) {
        System.out.println("La carpeta no existeix");
    } else {
        // Recorrem l'array i mostrem el nom de cada element
        System.out.println("La carpeta conté un total de " + llista.length + " elements");
        for (File f : llista) {
            System.out.println((f.isDirectory()? "[DIR] " : "[ARX] ") + f.getName());
        }
    }
}
```

Fer Exercicis butlletí A

1. INTRODUCCIÓ
2. GESTIÓ DE FITXERS
 1. La classe *File*
 2. Rutes absolutes i relatives
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

3. LECTURA I ESCRIPTURA DE FITXERS

Fitxers orientats a caràcter



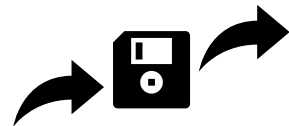
- L'objectiu principal d'usar fitxers és **poder emmagatzemar dades** de manera que entre diferents execucions del programa **siga possible recuperar-los**
- Per a saber com tractar les dades d'un fitxer, cal tindre molt clar com s'estructuren
- Els valors s'emmagatzemen en forma de seqüència, un darrere l'altre
- Per tant **la forma més habitual de tractar fitxers és seqüencialment**, de manera semblant a com es fa per a:
 - Llegir dades per teclat
 - mostrar-les per pantalla
 - recórrer les posicions d'un *array*
 - ...



Es denomina **accés seqüencial** al processament d'un conjunt d'elements de manera que només és possible accedir a ella d'acord amb la seua ordre d'aparició. Per a processar un element és necessari processar primer tots els elements anteriors.

3. LECTURA I ESCRIPTURA DE FITXERS

Fitxers orientats a caràcter



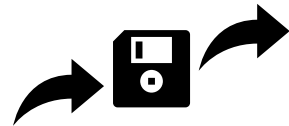
En els **fitxers orientats a caràcter**, les dades es representen com una seqüència de cadenes de text, on cada valor es diferencia de l'altre usant un delimitador. En canvi, en els **fitxers orientats a byte**, les dades es representen directament d'acord amb el seu format en binari, sense cap separació.

No els anem a tractar
en aquest mòdul

- Els valors estan emmagatzemats segons la seua representació en cadena de text, exactament en el mateix format que has usat fins ara per a entrar dades des del teclat
- Els diferents valors es distingeixen en estar **separats entre ells amb un delimitador**, que per defecte és qualsevol conjunt d'espais en blanc o salt de línia
- Es pot considerar que els valors estan organitzats un darrere l'altre, seqüencialment, com les paraules en la pàgina d'un llibre

3. LECTURA I ESCRIPTURA DE FITXERS

Fitxers orientats a caràcter



```
1,5 0,75 -2,35 18 9,4 3,1416 -15,785  
-200,4 2,56 9,3785
```

Quants valors diries que hi ha
emmagatzemats en aquest arxiu?
I de quin tipus són?

```
Hi havia una vegada...
```

I en aquest altre? Quantitat i tipus?

En un fitxer orientat a caràcter **és possible emmagatzemar qualsevol combinació de dades de qualsevol tipus** (*int, double, boolean, String*, etc.) ... la dificultat vindrà a l'hora de llegir-les

```
7 10 20,5 16,99  
Hi havia una vegada...  
true false 2020 0,1234
```

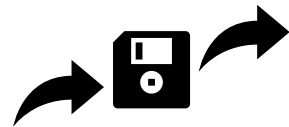
FITXERS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. GESTIÓ DE FITXERS
 1. La classe *File*
 2. Rutes absolutes i relatives
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



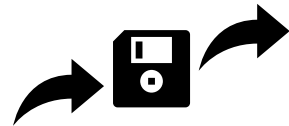
- La classe que permet dur a terme la **lectura de dades des d'un fitxer** orientat a caràcter és exactament la mateixa que permet llegir dades des del teclat: ***Scanner***
- Al cap i a la fi, els valors emmagatzemats es troben en el mateix format que has usat fins ara per a entrar informació en els programes: una seqüència de cadenes de text (*String*)
- Per a processar dades des d'un arxiu, **el constructor de la classe *Scanner* permet com a argument un objecte de tipus *File*** que continga la ruta a un arxiu

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

...
File f = new File("C:\\Programes\\Unitat11\\Document.txt");
Scanner lectorArchivo = new Scanner(f);
...
```

3. LECTURA I ESCRIPTURA DE FITXERS

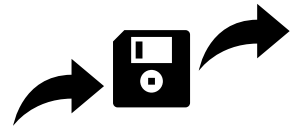
Lectura de fitxers (classe *Scanner*)



- Una vegada instanciat l'objecte *Scanner* **podem utilitzar els seus mètodes exactament igual que si llegírem de teclat: *hasNext()*, *next()*, *nextLine()*, *nextInt()*, *nextDouble()*, *nextBoolean()*, etc...**
- L'objecte *Scanner* gestiona internament **un punter que indica sobre quin valor actuaran les operacions de lectura**
- Inicialment el punter es troba en el primer valor dins de l'arxiu. Cada vegada que es fa una lectura **el punter avança automàticament fins al següent valor dins de l'arxiu i no hi ha cap manera de fer-lo retrocedir**
- A mesura que invoquem mètodes de lectura el punter continua avançant **fins que hagem llegit tantes dades com vulguem, o fins que no puguem continuar llegint perquè hem arribat al final del fitxer**

3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



Fitxer a llegir

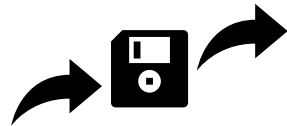
2 10

8 6 23 -15 0 -2 5

Exemple de
lectura sobre un
arxiu que conté
valors de tipus
sencer

3. LECTURA I ESCRIPTURA DE FITXERS

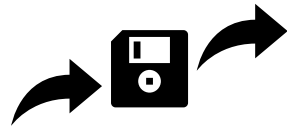
Lectura de fitxers (classe *Scanner*)



- És important recordar la **diferència entre el mètode *next()* i *nextLine()* ... ?**
- ***next()* només llig una paraula individual** (conjunts de caràcters, inclosos dígit, que no estan separats per espais o salts de línia, com per exemple "casa", "hola", "2", "3,14", "1024", etc.)
- ***nextLine()* llig tot el text que trobe (espais inclosos) fins al següent salt de línia**
- Una vegada s'ha finalitzat la lectura de l'arxiu, ja siguin totes o només una part, **és imprescindible executar un mètode especial anomenat *close()***
- Aquest mètode indica al sistema operatiu que l'arxiu ja no està sent utilitzat pel programa
- Això és molt important ja que mentre un arxiu es considera en ús, el seu accés pot veure's limitat

3. LECTURA I ESCRIPTURA DE FITXERS

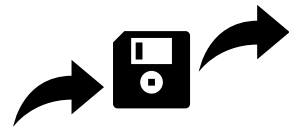
Lectura de fitxers (classe *Scanner*)



- És important saber que al instanciar l'objecte *Scanner* es llançarà una excepció de tipus *java.io.FileNotFoundException* si el fitxer no existeix
- *Scanner* també **pot llançar altres excepcions**, per exemple:
 - si s'intenta llegir el tipus de dada incorrecta (cridem a *nextInt()* quan no hi ha un enter)
 - si hem arribat al final del fitxer i intentem continuar llegint
- Sempre **caldrà manejar aquestes excepcions** mitjançant un *try-catch*

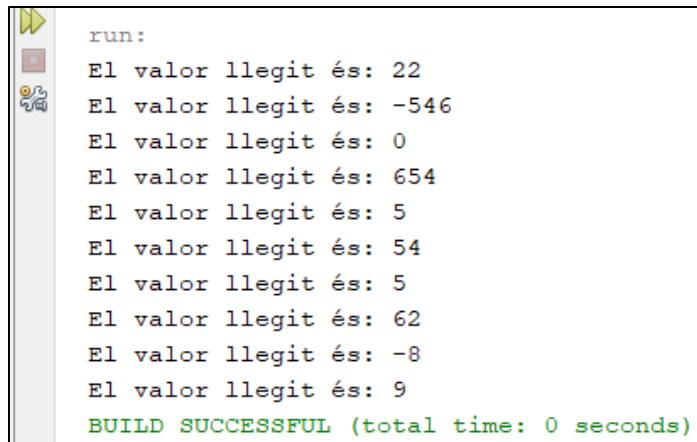
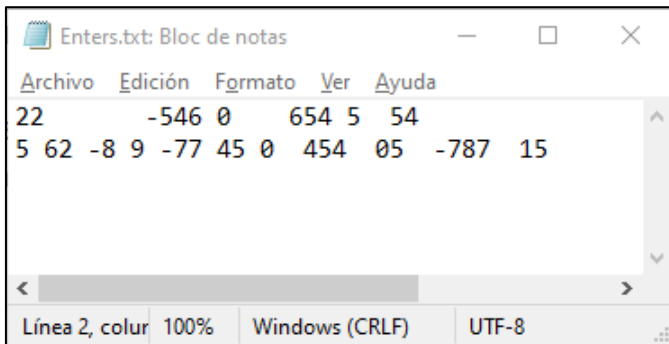
3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



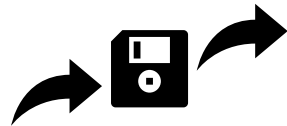
Exemple 7:

- Implementa un programa que llija un arxiu d'enters i mostre els seus 10 primers valors per pantalla
- El programa ha de gestionar les possibles excepcions (de forma genèrica: *Exception*)
- Crea un arxiu amb nombres enters i comprova el teu programa. Les dades poden estar separades per més d'un espai o en diferents línies



3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



Exemple 7:

- A continuació analitza el programa per a saber els diferents tipus d'excepcions que aquest pot llançar
- Modifica el programa per a que gestione les excepcions de forma individualitzada, amb missatges específics per a cada tipus possible d'excepció

Class InputMismatchException

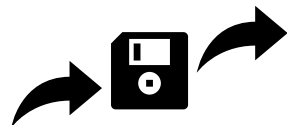
```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.util.NoSuchElementException
          java.util.InputMismatchException
```

All Implemented Interfaces:

Serializable

3. LECTURA I ESCRITURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



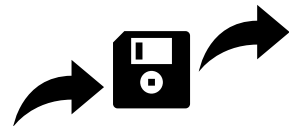
Exemple 7:

```
public static void main(String[] args) {
    try {
        // Intentem obrir el fitxer
        File f = new File("C:/Temporal/Enters.txt");
        Scanner lector = new Scanner(f);

        // Si arriba ací és que ha obert el fitxer
        for (int i = 0; i < NUM_VALORS; i++) {
            int valor = lector.nextInt();
            System.out.println("El valor llegit és: " + valor);
        }
        // Cal tancar el fitxer!
        lector.close();
    } catch (FileNotFoundException e) {
        System.out.println("Error: No es troba l'arxiu");
        e.printStackTrace();
    } catch (InputMismatchException e) {
        System.out.println("Error: El tipus de dada llegida no és un enter");
        e.printStackTrace();
    } catch (NoSuchElementException e) {
        System.out.println("Error: No hi ha més dades a llegir");
        e.printStackTrace();
    }
}
```


3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)

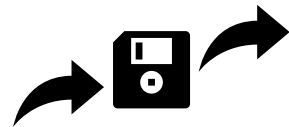


IMPORTANT - 1:

- Quan es duu a terme el procés de lectura d'una seqüència de valors, sempre **cal anar amb compte d'usar el mètode adequat al tipus** de valor que s'espera
- **És la vostra responsabilitat** saber què cal llegir a cada moment, **saber el tipus de dades que aneu a llegir**
- De totes maneres res garanteix que no s'haja comés algun error o que l'arxiu haja sigut manipulat per un altre programa o usuari
- Per tant, **el programa ha de gestionar errors i excepcions** i finalitzar el procés de lectura de forma controlada

3. LECTURA I ESCRIPTURA DE FITXERS

Lectura de fitxers (classe *Scanner*)



IMPORTANT - 2:

- També és **necessari controlar que mai es llisquen més valors dels que hi ha disponibles**
- Intentar llegir un nou valor quan el punter ja ha superat l'últim disponible es considera erroni i llançarà una excepció
- Per a evitar-ho, **serà necessari utilitzar el mètode `hasNext()` abans de llegir**, que ens retornarà *true* si existeix un element a continuació
- Una vegada s'arriba al final de l'arxiu ja no queda més remei que invocar *close()* i finalitzar la lectura



ESTÀ CLAR?

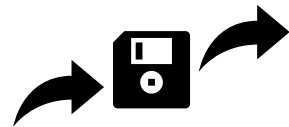
FITXERS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. GESTIÓ DE FITXERS
 1. La classe *File*
 2. Rutes absolutes i relatives
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. BONUS TRACK: Creació d'arxius de projecte .JAR

3. LECTURA I ESCRIPTURA DE FITXERS

Escriptura en fitxers (classe *FileWriter*)



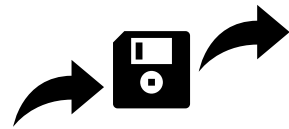
- Per a escriure dades a un arxiu la classe més senzilla d'utilitzar és *FileWriter*
- Aquesta classe té dos constructors que val la pena conèixer:
 - *public FileWriter(File file)*
 - *public FileWriter(File file, boolean append)*
- El primer constructor és molt semblant a *Scanner*. **Només cal passar-li un objecte *File* amb la ruta a l'arxiu**

```
File f = new File("C:\\Programes\\Unitat11\\Document.txt");  
FileWriter writer = new FileWriter(f);
```

- **Si el fitxer no existeix, es crearà un nou**
- Però **si el fitxer ja existeix, el seu contingut s'esborra per complet. DANGER!!** → pèrdua de dades valuoses

3. LECTURA I ESCRIPTURA DE FITXERS

Esriptura en fitxers (classe *FileWriter*)

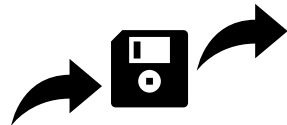


```
File f = new File("C:\\Programes\\Unitat11\\Document.txt");  
FileWriter writer = new FileWriter(f, true);
```

- El segon constructor “*public FileWriter(File file, boolean append)*” té un altre paràmetre de tipus booleà anomenat “*append*” (afegir)
- Aquest paràmetre **permet indicar si volem escriure al final del fitxer o no**
- Si li passem *false* farà el mateix que el constructor anterior (si l'arxiu ja existeix, el sobreescriurà)
- Si li passem *true* obrirà l'arxiu per a escriptura en mode “***append***” i escriurem al final del fitxer sense esborrar les dades ja existents

3. LECTURA I ESCRIPTURA DE FITXERS

Escriptura en fitxers (classe *FileWriter*)



- Per a l'escriptura seqüencial de dades només és necessari utilitzar el mètode

`void write(String str)`

- Escriurà la cadena *str* en el fitxer
- Si es desitja afegir un **salt de línia** es pot agregar `"\n"`
- Perquè el mètode *write()* escriga text correctament **és imprescindible passar-li com a argument un *String***
- Està permès utilitzar dades o variables diferents a *String*, però s'escriurà directament el seu valor en *bytes*, no com a text

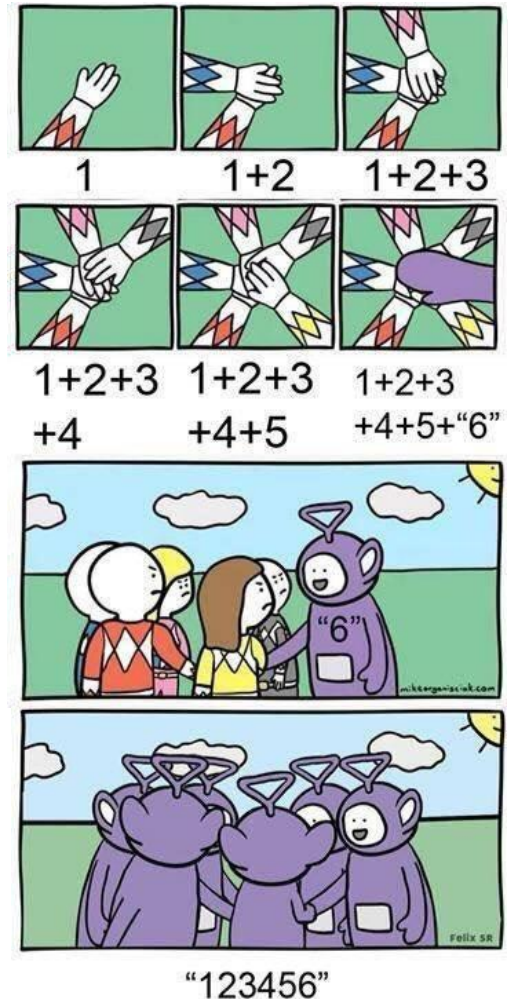
```
writer.write("65"); // Escriu dos caràcters, el 6 i el 5
writer.write(65); // Escriu 65 com a byte, és el caràcter A
```

3. LECTURA I ESCRIPTURA DE FITXERS

Escriptura en fitxers (classe *FileWriter*)

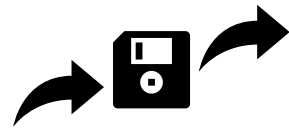
- Quan volem escriure el valor de variables que no siguin *String* serà necessari passar-les a *write()* com *String ... com?*
- Només cal concatenar un *String* buit amb la variable (Java sempre converteix a *String* la concatenació de cadenes de text amb qualsevol altre element): `"" + variable`

```
int edat = 35;  
writer.write("" + edat); // escriu el text "35"
```



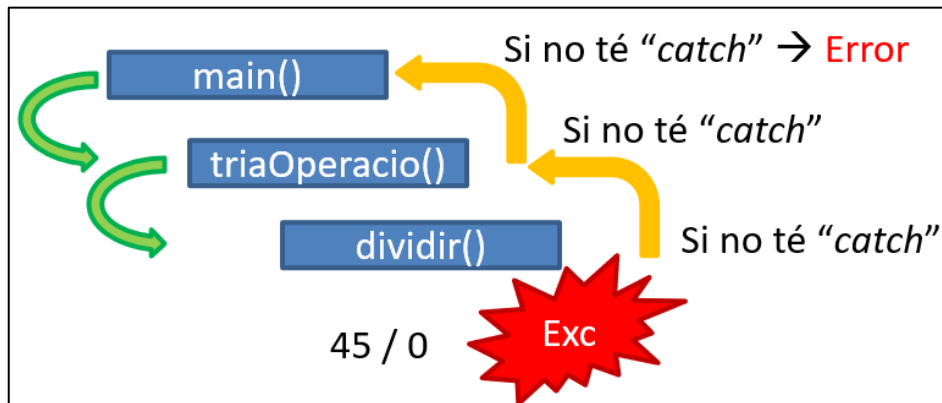
3. LECTURA I ESCRIPTURA DE FITXERS

Espectura en fitxers (classe *FileWriter*)



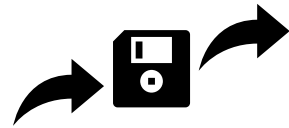
- Una vegada s'ha escrit una dada ja no hi ha marxa arrere
- **No és possible escriure informació abans o enmig de valors** que ja estan escrits

⚡ Tant el constructor de `FileWriter` com el mètode `write()` poden llançar una excepció `IOException` si es produeix algun error inesperat.



3. LECTURA I ESCRIPTURA DE FITXERS

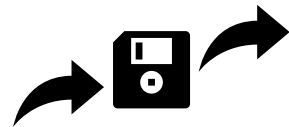
Escriptura en fitxers (classe *FileWriter*)



- Com en el cas de la lectura, la classe *FileWriter* també gestiona un **punter que li permet saber a partir de quina posició del text ha d'anar escrivint**
- Cada vegada que s'invoca un dels seus mètodes d'escriptura, **l'apuntador avança automàticament i no és possible fer-lo retrocedir**
- A efectes pràctics aquest apuntador sempre està al final de l'arxiu, de manera que a mesura que es van escrivint dades l'arxiu va incrementant la seua grandària
- L'escriptura no genera automàticament un delimitador entre valors. **Els espais en blanc o salts de línia que es desitgen incorporar han d'escriure's explícitament**
- En cas contrari els valors quedaran pegats i en una posterior lectura s'interpretaran com un únic valor

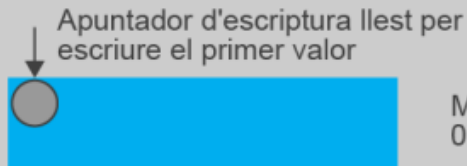
3. LECTURA I ESCRIPTURA DE FITXERS

Escriptura en fitxers (classe *FileWriter*)



Exemple d'escriptura

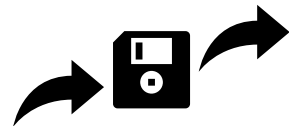
```
File F = new File(...);  
FileWriter escriptor = new FileWriter(f);
```



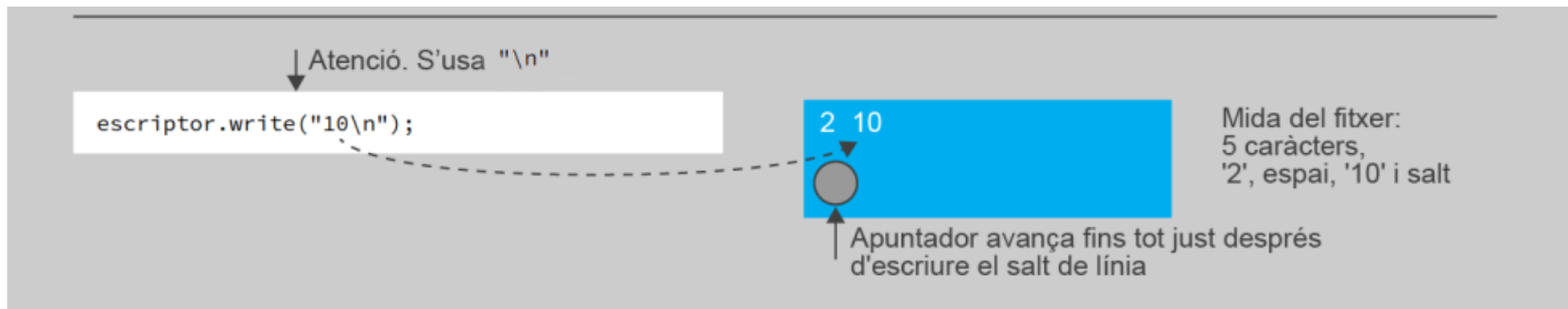
Mida del fitxer:
0 bytes

3. LECTURA I ESCRIPTURA DE FITXERS

Espectura en fitxers (classe *FileWriter*)

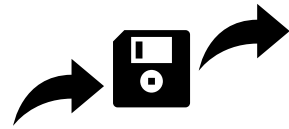


Exemple d'espectura



3. LECTURA I ESCRIPTURA DE FITXERS

Espectura en fitxers (classe *FileWriter*)



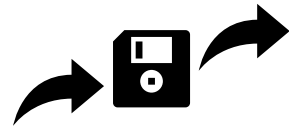
- En escriure en fitxers **el tancament amb `close()` és encara més important** que en la lectura
- Això es deu al fet que els sistemes operatius sovint actualitzen les dades de forma diferida
- És a dir, el fet d'executar una instrucció d'espectura no significa que immediatament s'escriga en l'arxiu. Pot passar un interval de temps variable
- Només **en executar el mètode `close()` es força al sistema operatiu a escriure les dades pendents** (si n'hi haguera)



En acabar l'espectura també **és imprescindible invocar el mètode `close()`** per a tancar-lo i assegurar la correcta espectura de dades.

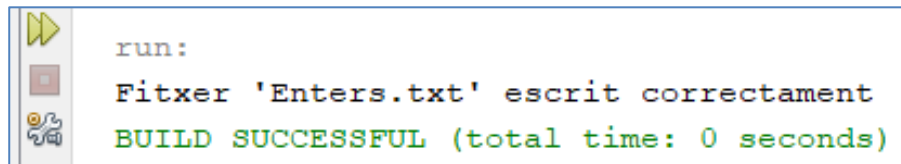
3. LECTURA I ESCRIPTURA DE FITXERS

Espectura en fitxers (classe *FileWriter*)



Exemple 8:

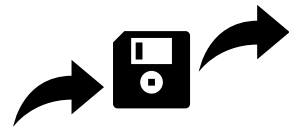
- Implementa un programa que escriu un arxiu anomenat "Enters.txt" dins de la carpeta de treball
- S'escriuen 20 valors enters, començant per l'1 i cada vegada el doble de l'anterior
- Cal que es gestionen les possibles excepcions
- Tingues en compte que si ja existia un arxiu amb el mateix nom, quedarà totalment sobreescrit
- S'ha de confirmar que el procés ha finalitzat satisfactòriament



```
run:  
Fitxer 'Enters.txt' escrit correctament  
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. LECTURA I ESCRIPTURA DE FITXERS

Escriptura en fitxers (classe *FileWriter*)



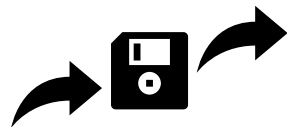
Exemple 8:

- Comprova que ha funcionat correctament llegint l'arxiu amb el programa de l'exemple anterior per a llegir 10 valors enters i mostrar-los per pantalla

```
run:
El valor llegit és: 1
El valor llegit és: 2
El valor llegit és: 4
El valor llegit és: 8
El valor llegit és: 16
El valor llegit és: 32
El valor llegit és: 64
El valor llegit és: 128
El valor llegit és: 256
El valor llegit és: 512
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. LECTURA I ESCRIPTURA DE FITXERS

Espectura en fitxers (classe *FileWriter*)



Exemple 8:

```
public static void main(String[] args) {  
    try {  
        File f = new File("C:/Temporal/Enters.txt");  
        FileWriter fw = new FileWriter(f);  
        int valor = 1;  
        for (int i = 1; i <= 20; i++) {  
            fw.write("" + valor); // escrivim valor  
            fw.write(" "); // escrivim espai en blanc  
            valor = valor * 2; // calculem pròxim valor  
        }  
        fw.write("\n"); // escrivim nova línia  
        fw.close(); // tanquem el FileWriter  
        System.out.println("Fitxer '" + f.getName() + "' escrit correctament");  
    } catch (IOException e) {  
        System.out.println("Error: " + e);  
    }  
}
```

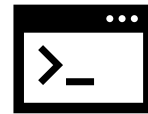
FITXERS

ÍNDEX DE CONTINGUTS

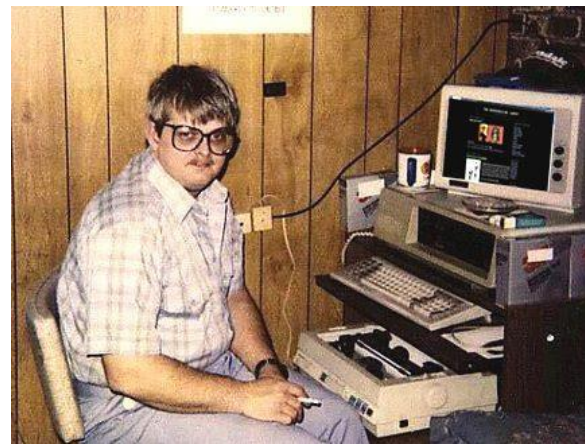
1. INTRODUCCIÓ
2. GESTIÓ DE FITXERS
 1. La classe *File*
 2. Rutes absolutes i relatives
 3. Mètodes de la classe *File*
3. LECTURA I ESCRIPTURA DE FITXERS
 1. Fitxers orientats a caràcter
 2. Lectura de fitxers (classe *Scanner*)
 3. Escriptura en fitxers (classe *FileWriter*)
4. **BONUS TRACK: Creació d'arxius de projecte .JAR**

4. BONUS TRACK

Compilació / execució per consola



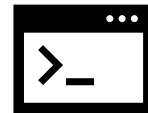
- ☐ Fins ara hem creat tots els programes mitjançant el IDE *Neatbeans* ... fàcil, eh?
- ☐ Però en realitat no és necessari treballar amb un IDE per a poder crear aplicacions en Java (o en qualsevol llenguatge de programació)
- ☐ De fet, "en los orígenes" no existien els IDEs
- ☐ L'única cosa necessària per a poder crear i executar programes en Java és:
 - ☐ un editor de text pla (com *notepad*, *nano*, *vi* ...)
 - ☐ una instal·lació de les eines de desenvolupament de Java ... (?)



LET'S DO IT !!!

4. BONUS TRACK

Compilació / execució per consola



- Escriviu mitjançant un editor de textos un programa (SENSE MIRAR CODI ANTERIOR) que mostre "Hola Pirindola", dins d'una carpeta que res tinga a veure amb *Neatbeans*
- Guardeu-lo amb extensió *.java*
- Obriu una consola en la carpeta que conté l'arxiu i compileu el codi amb *javac*

```
AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/OneDrive - Co
$ javac HolaPirindola.java
HolaPirindola.java:5: error: reached end of file while parsing
}
^
1 error
```

```
AiA@LAPTOP-HF24761L MINGW64 ~/Do
$ javac HolaPirindola.java

AiA@LAPTOP-HF24761L MINGW64 ~/Do
$
```

4. BONUS TRACK

Compilació / execució per consola



- Si la compilació ha funcionat, us haurà creat l'arxiu `.class`

```
AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/OneDrive - Cons
$ ll Hola*
-rw-r--r-- 1 AiA 197609 440 Apr 25 10:21 HolaPirindola.class
-rw-r--r-- 1 AiA 197609 136 Apr 25 10:21 HolaPirindola.java

AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/OneDrive - Cons
$
```

- Ara ja podeu executar el vostre programa mitjançant la comanda `java`

```
AiA@LAPTOP-HF24761L MINGW64 ~/Documents/
$ java HolaPirindola
Hola, Pirindola !!!!


AiA@LAPTOP-HF24761L MINGW64 ~/Documents/
$
```

4. BONUS TRACK

Compilació / execució per consola



Després de 2 trimestres programant ... **sabeu ja què significa lo de *String[] args*?**
Quantes vegades ho heu utilitzat en els vostres programes?



```
public static void main(String[] args) {  
    try {  
        File f = new File("C:/Temporal/Enters.txt");  
        FileWriter fw = new FileWriter(f);  
        int valor = 1;  
        for (int i = 1; i <= 20; i++) {  
            fw.write("" + valor); // escrivim valor  
            fw.write(" "); // escrivim espai en blanc  
            valor = valor * 2; // calculem pròxim valor  
        }  
    }  
}
```

4. BONUS TRACK

Compilació / execució per consola



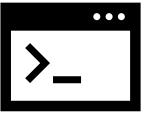
- Es tracta d'un *array* que conté tots els arguments d'entrada que es poden passar des de consola al nostre mètode principal (*main*)
- Els arguments són tractats SEMPRE com *Strings* (incloent els *int*, *double* ...)
- Van separats per espais en la línia del terminal

Modifica ara el teu programa anterior perquè demane un argument en el moment de l'execució i el mostre per pantalla

```
AiA@LAPTOP-HF24761L MINGW64 ~/Docu
$ java Hola Bacora
Hola, Bacora!!!!
AiA@LAPTOP-HF24761L MINGW64 ~/Docu
```

4. BONUS TRACK

Compilació / execució per consola



Modifica el teu programa per a que demane 3 arguments diferents

```
AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/0
$ java HolaEdat Paqui 24 Mur1a
Hola, em dic Paqui tinc 24 anys i visc en Mur1a

AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/0
```

Crea un nou programa que faci la suma de tots els enters que es passen per consola

```
AiA@LAPTOP-HF24761L MINGW64 ~/Docume
$ java Suma 1 2 3 4
La suma de tots els números és:10
```

```
AiA@LAPTOP-HF24761L MINGW64 ~/Docum
$ java Suma 1 2 3 4 -3 -1 25
La suma de tots els números és:31
```

4. BONUS TRACK

Creació d'arxius de projecte .JAR



- ☐ Quan treballem amb molts arxius Java i volem exportar el nostre projecte, és convenient incloure tots els fitxers en un sol arxiu
- ☐ Per a fer-ho, podem crear un arxiu executable *.JAR* que conté comprimit el nostre projecte Java
- ☐ Primer compilarem tot el codi del nostre programa i després crearem l'arxiu *.JAR* a partir dels arxius de classes creats *.CLASS*

<https://www.baeldung.com/java-create-jar>



Utilitza el programa anterior de la Suma per a crear i executar el .JAR

4. BONUS TRACK

Creació d'arxius de projecte .JAR



Exemple 9:

- Recupera el projecte del Videojoc i crea el seu .JAR
- Copia només els arxius .JAVA en una nova carpeta i crea el JAR a partir d'aquests
- Executa el JAR i comprova que ha funcionat correctament

```
AiA@LAPTOP-HF24761L MINGW64 ~/Documents/Professor/OneDrive - Conselleria d'Educació/IES/2022-2023-1
R_Videojocs (main)
$ java -jar Videojoc.jar
*** LimitDePantalla {No es pot col·locar/moure a Darth Vader fins a la posició -211 horitzontal}

### Info personatge ###
Nom: Snoke
Tipus: Enemic
Posició: 23 / -50
Vida disponible: 2

### Info personatge ###
Nom: Darth Vader
Tipus: Enemic
Posició: -200 / 56
Vida disponible: 500
```


Fer Exercicis butlletí B

Autor:

Àngel Olmos Giner

segons el material de Carlos Cacho i Raquel Torres

Llicència:**CC BY-NC-SA 3.0 ES Reconeixement – No Comercial – Compartir Igual (by-nc-sa)**

No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. Aquesta és una obra derivada de l'obra original de Carlos Cacho i Raquel Torres