GENERALITAT VALENCIANA

ceedcv
CENTRE ESPECÍFIC D´EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

UF11.- FITXERS

- Teoria -

PROGRAMACIÓ
CFGS DAW

Joan V. Cassany
jv.cassanycoscolla@edu.gva.es
2022/2023

1. INTRODUCCIÓ

La principal funció d'una aplicació informàtica és la manipulació i transformació de dades.

Fins ara, tots els tipus de dades vists només tenen vigència mentre el programa s'està executant, quan el programa finalitza les dades que contenen desapareixen.

En alguns casos, necessitem que les dades es mantinguen o emmagatzemen de manera persistent entre diferents execucions del programa.

Veurem diferents **classes de Java** que ens permeten crear, llegir, escriure i eliminar fitxers i directoris, entre altres operacions.

També, s'introdueix la **serialització d'objectes** com a mecanisme per a emmagatzemar objectes en fitxers.

2. GESTIÓ DE FITXERS

Entre les funcions d'un **sistema operatiu** està la d'oferir mecanismes genèrics per a gestionar sistemes d'arxius.

Els dispositius físics d'emmagatzematge de dades pot ser molt diferent (magnètic, òptic, etc.), però la manera de gestionar el sistema d'arxius sol ser molt similar en la immensa majoria dels casos: una **estructura jeràrquica** amb carpetes i fitxers.

Molts llenguatges de programació proporcionen **biblioteques** que permeten accedir directament als mecanismes interns que ofereix el sistema.

Java no és cap excepció oferint aquest tipus de biblioteca, en forma del conjunt de classes incloses dins del ***package java.io***.

2. GESTIÓ DE FITXERS

LA CLASSE File

La peça básica per a manipular arxius en un programa Java és la **classe File** que pertany al ***package java.io***, i per tant s'haurà d'importar:

import java.io.File;

Aquesta classe representa una ruta dins del sistema d'arxius i haurà de ser instanciada per a poder invocar els seus mètodes.

File f = new File (String ruta);

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 que realment s'accedeix al sistema de fitxers i es processa la informació.

2. GESTIÓ DE FITXERS

LA CLASSE File

El format de la ruta pot ser diferent segons el sistema operatiu:

- Exemple de ruta Unix: ***/usr/bin***
- Exemple de ruta Windows: ***C:\Windows\System32***

De totes maneres Java ens permet utilitzar la barra d'Unix (***"/"***) per a representar rutes de Windows. També, podem utilitzar la propietat ***File.separator*** que conté el caràcter per defecte del sistema en que ens trobem (ruta="C:" + File.separator + "usr" ...).

Cada objecte representa una única ruta. Per a diferents rutes cal crear diversos objectes.

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

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 :

Windows: ***N:\Documents\Unitat11\apartat1\Activitats.txt (ruta a un arxiu)***
Unix: ***/Documents/Unitat11/apartat1/Activitats.txt (ruta a un arxiu)***

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 que pot variar si el programa canvia d'ubicació. **Quan un programa s'executa** per defecte **se li assigna una carpeta de treball**.

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

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

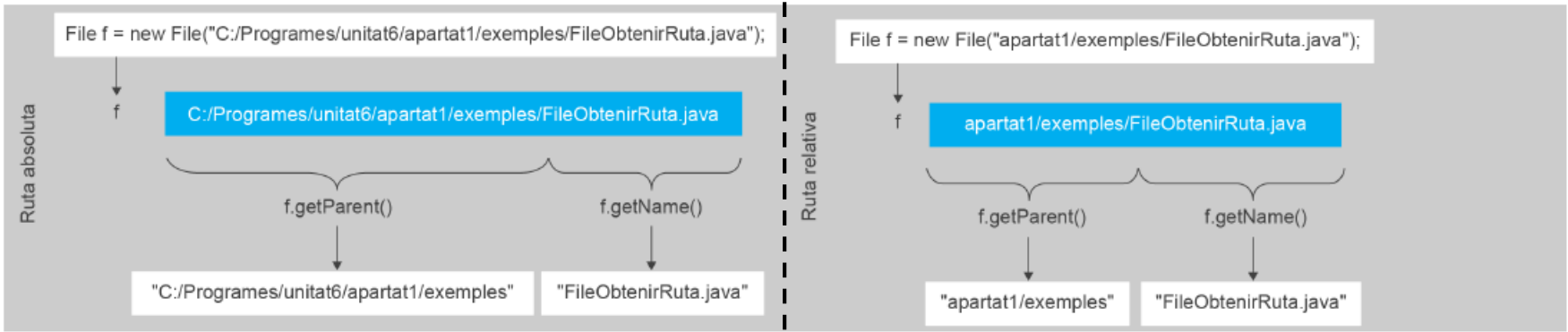
2. GESTIÓ DE FITXERS

MÈTODES: OBTENCIÓ DE LA RUTA

String ***getParent()*** retorna la ruta de la carpeta de l'element referit per aquesta ruta. Si la ruta tractada es refereix a la carpeta arrel ("C:\", "/", etc.), aquest mètode retorna *null*.

String ***getName()*** retorna el nom de l'element referit a la ruta, és a dir l'últim element.

String ***getAbsolutePath()*** retorna la ruta absoluta.



***Nota**: Veure Exemple01*

2. GESTIÓ DE FITXERS

MÈTODES: COMPROVACIÓ D'ESTAT

boolean ***exists()*** comprova si la ruta existeix dins del sistema de fitxers. Retornarà *true* si existeix i *false* en cas contrari.

boolean ***isFile()*** comprova el sistema de fitxers a la recerca de 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()*** funciona com l'anterior però comprova si és una carpeta.

MÈTODES: PROPIETATS D' ARXIUS

long ***length()*** retorna la grandària d'un arxiu en bytes. Utilitzar sols amb arxius.

long ***lastModified()*** retorna l'última data com el nombre de mil·lisegons que han passat des de l'1 de juny de 1970.

Nota*: Veure Exemple02 i Exemple03*

2. GESTIÓ DE FITXERS

MÈTODES: GESTIÓ D' ARXIUS

boolean ***mkdir()*** permet crear la carpeta indicada que no ha d'existir en la ruta en el moment d'invocar el mètode. Per exemple, donada una instància File amb la ruta "*C:/Fotos*", crearà la carpeta "*Fotos*" dins de "*C:*". Retorna *true* si s'ha creat correctament, *false* en cas contrari (si la ruta l'incorrecta, sija existeix o per falta de permisos).

boolean ***delete()*** esborra l'arxiu o carpeta indicada en la ruta. Es podrà esborrar una carpeta sol si està buida. Retorna *true* o *false* segons si s'ha pogut dur a terme.

boolean ***createNewFile()*** crea un arxiu buit. Java ens obligarà a incloure la instrucció dins d'un context de captura d'excepcions per control intern de Java front a errors crítics.

boolean ***renameTo(File destí)*** permet tant canviar el nom com canviar la ubicació que en definitiva és el mateix.

***Nota**: Veure Exemple04 i Exemple05*

2. GESTIÓ DE FITXERS

MÈTODES: LLISTAT D'ARXIUS

File[ ] ***listfiles()*** retorna un vector de tipus File (File[ ]) amb tots els elements continguts en la carpeta (representats per objectes File, un per element). Perquè s'execute correctament la ruta ha d'indicar una carpeta. La grandària del vector serà igual al nombre d'elements que conté la carpeta. Si la grandària és **0**, el valor retornat serà *null* i tota operació posterior sobre el vector serà errònia. L'ordre dels elements és aleatori (al contrari que en l'explorador d'arxius del sistema operatiu, no s'ordena automàticament per tipus ni alfabèticament).

***Nota**: Veure Exemple06*

3. LECTURA I ESCRIPTURA DE FITXERS

La forma més habitual de tractar fitxers és seqüencialment, de manera semblant a com es fa per a llegir-les del teclat.

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, es a dir, es podrà llegir un element quan s'haja llegit els anteriors.

Java diferència entre dos **tipus d'arxius** segons els valors emmagatzemats:

- Els fitxers **orientats a caràcter**, cadenes de text en les quals cada valor es diferencia de l'altre usant un delimitador.
- Els fitxers **orientats a byte**, les dades es representen directament d'acord amb el seu format en binari, sense cap separació.

3. LECTURA I ESCRIPTURA DE FITXERS

FITXERS ORIENTAT A CARÀCTER

Veiem alguns exemples.

Deu valors de tipus real
7 en la primera línia i 3 en la segona

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

Quatre valors de tipus String

Hi havia una vegada

3. LECTURA I ESCRIPTURA DE FITXERS

FITXERS ORIENTAT A CARÀCTER

Per al cas dels fitxers orientats a caràcter, cal usar **dues classes** diferents segons si el que es vol és **llegir** o **escriure** dades en un arxiu.

Hem de tindre en compte que només es duran a terme operacions de lectura o d'escriptura sobre un mateix arxiu, però **no els dos tipus d'operació alhora**.

Per a lectura: classe ***Scanner*** del ***package java.util***

Per a escriptura: classe ***FileWriter*** del ***package java.io***

3. LECTURA I ESCRIPTURA DE FITXERS

LECTURA DE FITXER: CLASSE SCANNER

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);
...
```

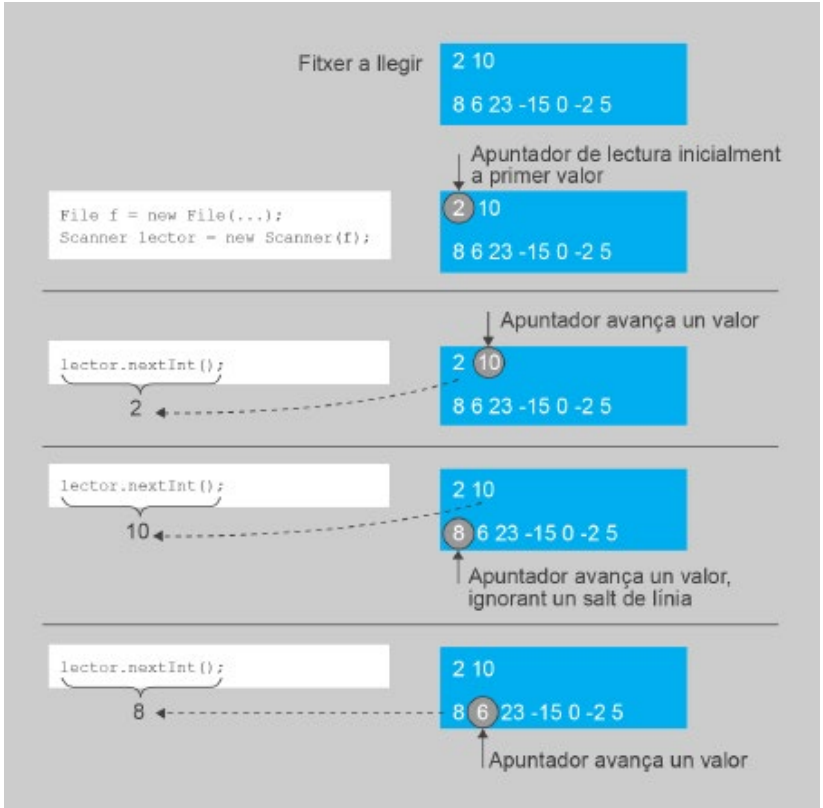
Una vegada instanciat l'objecte Scanner **podem utilitzar els seus mètodes exactament igual que si llegírem de teclat**: hasNext(), next(), nextLine(), nextInt(), nextDouble(), nextBoolean(), etc.

3. LECTURA I ESCRIPTURA DE FITXERS

LECTURA DE FITXER: CLASSE SCANNER

L'objecte **Scanner** gestiona internament un apuntador que indica sobre quin valor actuaran les operacions de lectura.

Cada vegada que es fa una lectura l'apuntador avança automàticament fins al següent valor dins de l'arxiu i no hi ha cap manera de fer-lo retrocedir.



3. LECTURA I ESCRIPTURA DE FITXERS

LECTURA DE FITXER: CLASSE SCANNER

És important recordar la diferència entre el mètode next() i nextLine()

- ***next()*** només llig una paraula individual
- ***nextLine()*** llig tot el text fins al següent salt de línia.

Una vegada s'ha finalitzat la lectura de l'arxiu és imprescindible executar un mètode especial anomenat ***close()***, ja que mentre un arxiu es considera en ús, el seu accés pot veure's limitat.

Al instanciar l'objecte Scanner es poden presentar diverses excepcions:

- El fitxer no existeix (FileNotFoundException)
- Llegir un tipus incorrecte (nextInt() quan hi ha lletres)
- Continuar llegint després d'arribar al final del (sempre podrem utilitzar el mètode hasNext() abans de llegir, que retorn true si existeix un element a continuació).

***Nota**: Veure Exemple07*

3. LECTURA I ESCRIPTURA DE FITXERS

ESCRIPTURA DE FITXER: 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), a aquest constructor només cal passar-li la ruta d'un objecte File.

- Si el fitxer no existeix, es crearà un nou.
- Si el fitxer ja existeix, el seu contingut s'esborra per complet.

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

3. LECTURA I ESCRIPTURA DE FITXERS

ESCRIPTURA DE FITXER: CLASSE FILEWRITER

public FileWriter(File file, boolean append), aquest constructor té un altre paràmetre de tipus booleà anomenat “append” (afegir) que permet indicar si volem escriure al final del fitxer o no.

- Si li passem “false” farà el mateix que el contructor anterior (el sobreescriurà).
- Si li passem “true” obrirà l'arxiu per a escriptura en **mode “append”**, és a dir, escriurem al final del fitxer sense esborrar les dades ja existents.

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

3. LECTURA I ESCRIPTURA DE FITXERS

ESCRIPTURA DE FITXER: CLASSE FILEWRITER

Per a realitzar l'escriptura utilitzarem el mètode ***void write(String str)*** que escriurà la cadena *str* en el fitxer. Si es desitja agregar un **final de línia** es pot agregar ***"\n"***.

Tant el constructor de **FileWriter** com el mètode **write()** poden llançar l'excepció ***IOException*** si es produeix algun error inesperat.

Perquè el mètode **write()** escriga text correctament és imprescindible passar-li com a argument un **String**. Les dades o variables diferents a String s'escriuran com **bytes**.

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

Per a convertir una variable a String només cal concatenar un String buit amb la variable

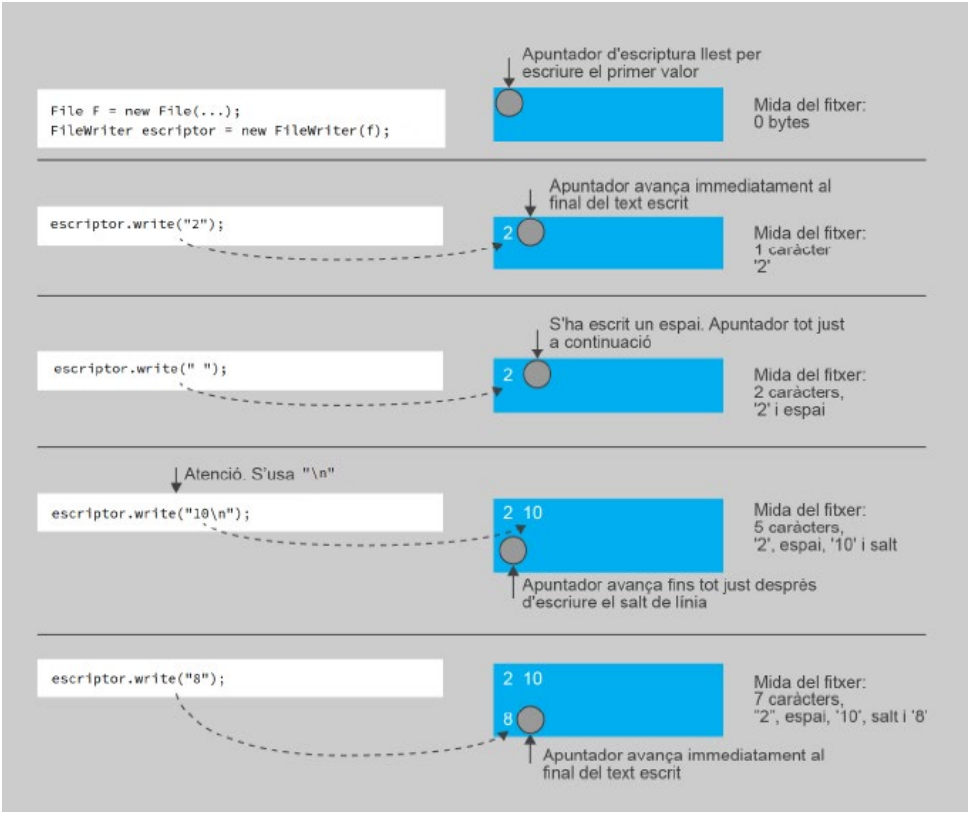
```
int edat = 35;  ➔  writer.write("" + edat); // escriu el text "35"
```

3. LECTURA I ESCRIPTURA DE FITXERS

ESCRIPTURA DE FITXER: CLASSE FILEWRITER

L'escriptura de dades en fitxer té la particularitat que una vegada s'ha escrit una dada ja no hi ha marxa arrere.

Com en el cas de la lectura, la classe ***FileWriter*** també gestiona un apuntador que li permet saber a partir de quina posició del text ha d'anar escrivint.



3. LECTURA I ESCRIPTURA DE FITXERS

ESCRIPTURA DE FITXER: CLASSE FILEWRITER

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 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'escriptura **no significa** que immediatament s'escriga en l'arxiu.

***Nota**: Veure Exemple08*

EXERCICIS PROPOSATS