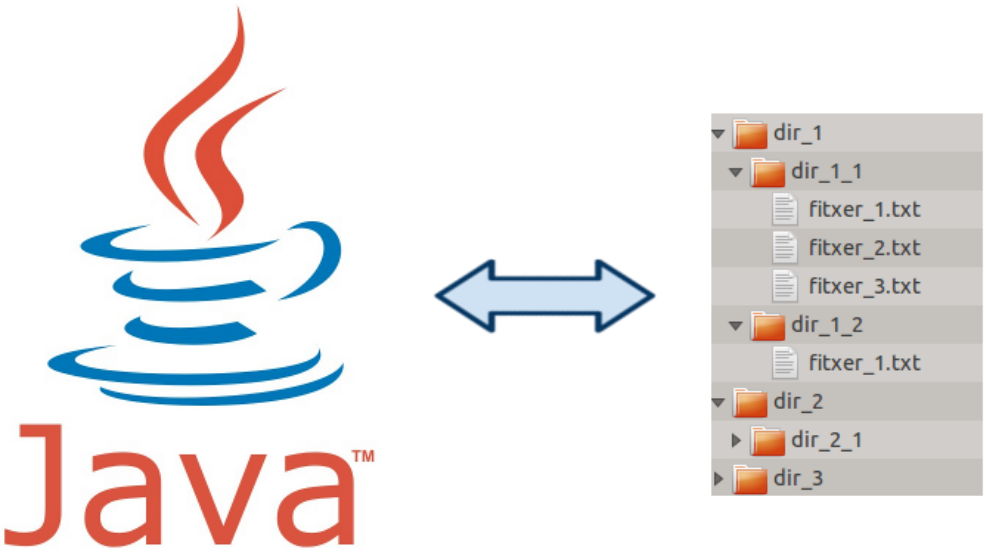
Accés a Dades

Tema 1: Gestió del Sistema de fitxers



Objectius

Objectius

L'objectiu del primer tema és senzillament arribar als fitxers, però no al seu contingut, que es veurà en el proper tema.

Tan sols arribarem a "veure" el fitxer, amb la seua ruta. De fet la manera d'arribar a un directori i a un fitxer serà idèntica. I del fitxer o directori podrem arribar a veure i també modificar les característiques externes: nom, grandària, data d'última modificació, permisos, ...

És un tema molt curtet que es podria haver englobat amb els dos temes següents, també de fitxers. Però hem preferit separar-lo per ser principi de curs, per a poder anar "rodant" sense presses.

1.- Introducció

Un **fitxer** o **arxiu** és un conjunt de bits emmagatzemats en un dispositiu (com podria ser per exemple un disc dur). Els fitxers ens garantitzen la **persistència**, ja que encara que apaguem l'ordinador podrem recuperar la informació, cosa que no passaria amb la informació guardada en memòria RAM. El tema de la persistència és justament el que buscarem durant tot el curs.

Un fitxer tindrà un nom que l'identifica. Aquest nom pot tenir opcionalment una extensió, generalment de 3 caràcters que tradicionalment ha servit per identificar el tipus de fitxer. El nom i l'extensió van separats per un punt.

En els sistemes informàtics actuals, en els quals un únic ordinador pot arribar a tenir guardats milions de fitxers, resulta imprescindible un sistema que permeta una gestió eficaç de localització, de manera que els usuaris puguem moure'ns còmodament entre tants arxius. La major part de **sistemes de fitxers** han incorporat **contenidors jerarquitzats** que actuen a mode de **directoris** facilitant la classificació, la identificació i localització dels arxius. Els directoris s'han acabat popularitzant sota la versió gràfica de **carpetes**. Dins d'un mateix directori, el nom del fitxer (o subdirectori) ha de ser únic.

Hem de tenir en compte, que a més dels fitxers guardats en el dispositiu de l'ordinador (el disc dur, o un altre dispositiu), la necessitat desmesurada d'espai d'emmagatzematge ha dut els Sistemes Operatius a treballar amb una gran quantitat de dispositius i a permetre l'accés remot a uns altres sistemes de fitxers, distribuïts per la xarxa.

- Per a poder gestionar tanta varietat de sistemes de fitxers, alguns sistemes operatius com **Linux o Unix** utilitzen l'estratègia d'unificar tots els sistemes en **un únic sistema de fitxers**, per tal d'aconseguir una forma d'accés unificada i amb una única jerarquia que facilite la referència a qualsevol dels seus components, amb independència del dispositiu en el qual es troben realment ubicats. En Linux, siga quin siga el dispositiu o el sistema remot real on es guardarà l'arxiu, **la ruta tindrà sempre la mateixa forma**. Observeu que per a recórrer la ruta jeràrquica on es troba el fitxer s'utilitza **la barra de dividir**:

```
/dir_1/dir_2/dir_3/.../dir_n/fitxer.txt
```

- Per contra, l'estratègia d'altres Sistemes Operatius com **Windows** passa per mantenir ben diferenciats cadascun dels sistemes i dispositius on es tinga accés. Per distingir el sistema al qual es vol fer referència, Windows utilitza una denominació específica del dispositiu o servidor, que incorpora a la ruta del fitxer o directori a referenciar. Encara que Microsoft ha apostat clarament per la convenció **UNC** (*Uniform Naming Convention*), l'evolució d'aquest sistema operatiu, que té com a origen l'MS-DOS, ha fet que coexistesca amb una altra convenció també molt estesa, la que utilitza una lletra de l'alfabet seguida de dos punts per a identificar el dispositiu on es troba el fitxer. A continuació il·lustrem amb un exemple les dues convencions. Observeu com en els dos casos s'utilitza la **contra-barra**:

```
F:\dir_1\dir_2\dir_3\...\dir_n\fitxer.txt
```

```
\\Servidor_1\dir_1\dir_2\dir_3\...\dir_n\fitxer.txt
```

2.- La classe File. Generalitats.

En Java, per a gestionar el sistema de fitxers s'utilitza bàsicament la classe '**File**'. És una classe que s'ha d'entendre com una referència a la ruta o localització de fitxers del sistema. **NO representa el contingut** de cap fitxer, sinó la ruta del sistema on es localitza el fitxer. Com que es tracta d'una ruta, *la classe pot representar tant* ***fitxers*** *com* ***carpetes o directoris***.

Si fem servir una classe per a representar rutes, s'aconsegueix una total independència respecte de la notació que utilitza cada sistema operatiu per descriure-les. Recordem que Java és un llenguatge multiplataforma i, per tant, perfectament pot donar-se el cas que haguem de fer una aplicació desconeixent el Sistema Operatiu on acabarà executant-se.

L'estratègia utilitzada per cada SO no afecta la funcionalitat de la classe **File**, ja que aquesta, en col·laboració amb la màquina virtual, adaptarà les sol·licituds al SO amfitrió **de forma transparent al programador**, és a dir, sense necessitat que el programador haja d'indicar o configurar res.

Els objectes creats de la classe File es troben estretament vinculats a la ruta amb la qual s'han creat. Això significa que les instàncies de la classe File durant tot el seu cicle de vida **només representaran una única ruta,** la que se'ls va associar en el moment de la creació. La classe File **no disposa** de cap mètode ni mecanisme per modificar la ruta associada. En cas de necessitar noves rutes, **caldrà sempre crear un nou objecte** i no serà possible reutilitzar els ja creats vinculant-los a rutes diferents.

Per a crear un objecte **File** es pot utilitzar qualsevol dels 3 constructors següents:

- **File(String *directori_i_fitxer*)**: indiquem en un únic paràmetre tant el directori com el fitxer, és a dir, el fitxer amb la seua ruta. Recordeu que en sistemes Linux per a la ruta utilitzem la barra de dividir, mentre que en Windows la contra-barra. Com que aquest caràcter és el d'*escape*, s'haurà de posar dues vegades:

```
File fitxer_1 = new File("/home/usuari/AD/T1/exemple1.txt");
```

```
File fitxer_1 = new File("C:\\AD\\T1\\exemple1.txt");
```

> **Nota**
> Cap de les referències anteriors són desitjables, ja que nosaltres intentarem fer programes que funcionen en qualsevol plataforma, i la primera referència no funcionarà en Windows, i la segona no funcionarà en Linux. Al llarg del tema aprendrem com fer les referències de manera que funcionen en qualsevol plataforma.

  Per a fer referència a un directori s'utilitza la mateixa tècnica, com ja havíem vist:

```
File dir = new File("/home/usuari/AD/T1");
```

  En els exemples anteriors hem posat una ruta absoluta, que comença des de l'arrel. Si no la posem absoluta (si no comença per /) serà relativa i començarà en el directori actiu. Si suposem que el directori actiu és **/home/usuari**, d'aquesta manera faríem referència al mateix lloc:

```
File dir = new File("AD/T1");
```

- **File(String *directori*, String *fitxer*)**: en el primer paràmetre (String) indiquem el directori amb ruta, i en el segon el fitxer (sense ruta). Farà referència a un fitxer amb el nom com el segon paràmetre col·locat en el directori referenciat en el primer paràmetre. Observeu com el segon paràmetre podria ser també un directori, i per tant seria una referència a un subdirectori del directori referenciat en el primer paràmetre.

```
File fitxer_2 = new File("/home/usuari/AD/T1" , "exemple2.txt");
```

- **File(File *directori*, String *fitxer*)**: Ara el directori és un File creat anteriorment

```
File fitxer_3 = new File(dir , "exemple3.txt");
```

En els exemples anteriors hem posat directament les rutes. Però els programadors de Java han de fer un esforç per independitzar les aplicacions implementades de les plataformes on s'executaran. Per tant, haurem d'anar amb cura, fent servir tècniques que eviten escriure les rutes directament al codi. Per això encara que ara al principi utilitzarem els 3 constructors, en el futur hauríem d'utilitzar massivament el tercer, ja que com veieu la manera d'especificar la ruta de localització del fitxer, és per mig d'un altre File.

La classe **File** encapsula pràcticament tota la funcionalitat necessària per gestionar un sistema de fitxers organitzat en arbre de directoris. És una gestió completa que inclou:

- Funcions de manipulació i consulta de la pròpia estructura jeràrquica (creació, eliminació, obtenció de la ubicació, etc. de fitxers o directoris)
- Funcions de manipulació i consulta de les característiques particulars dels elements (noms, mida o capacitat, etc.)
- Funcions de manipulació i consulta d'atributs específics de cada Sistema Operatiu, com per exemple els permisos d'escriptura, d'execució, atributs d'ocultació. Només funcionarà si el sistema operatiu amfitrió suporta també la funcionalitat d'aquestos atributs.

No ens permet, però, accedir al contingut dels fitxers. Això es resoldrà en el següent tema.

Nota important

Farem els exemples i exercicis d'aquest tema en un únic projecte d'Eclipse anomenat **Tema1**. En ell us heu de crear 2 paquets, el d'**Exemples** i el d'**Exercicis**. Posteriorment us podeu copiar els exemples en el paquet d'Exemples, cosa que farà que es cree la classe amb el nom apropiat. El següent vídeo il·lustra el procés, per si encara no esteu rodats en Eclipse.

Mirem un exemple. Anem a fer un programa per a traure la llista de fitxers i directoris del directori actual. Per a fer referència al directori actual, utilitzarem ".", que ens serveix per a tots els Sistemes. Per defecte, el directori actiu és el directori del projecte. Per a obtenir la llista d'elements (fitxers i directoris) utilitzarem el mètode **list()** de la classe **File**. Veurem aquest mètode, juntament amb els mètodes més importants en la següent pregunta.

```
import java.io.File;

public class Exemple_1_1 {
   public static void main(String[] args) {
      File f = new File(".");
      System.out.println("Llista de fitxers i directoris del directori
      actual");
      System.out.println("-------------------------------------------------");
      for (String e : f.list())
         System.out.println(e);
   }
}
```

I aquest seria el resultat:

```
Llista de fitxers i directoris del directori actual
---------------------------------------------------
.project
.settings
bin
.classpath
src
```

Si vulguérem traure el contingut d'un directori concret, el posaríem en el moment de definir el File, en compte del punt per a indicar el directori actual. Una altra modificació seria demanar per teclat el directori del qual volem mostrar el contingut:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

public class Exemple_1_2 {
   public static void main(String[] args) throws IOException {
      System.out.println("Introdueix un directori:");
      String ent = new BufferedReader(new
      InputStreamReader(System.in)).readLine();
      File f = new File(ent);
      System.out.println("Llista de fitxers i directoris del directori
      " + ent);
      System.out.println("---------------------------------------------------");
      for (String e : f.list())
         System.out.println(e);
   }
}
```

3.- Funcionalitat de la classe File

La classe **File** conté una sèrie de mètodes que ens permeten traure informació relativa al fitxer o directori al que apunta, així com poder navegar (obtenint el pare o accedint a algun dels directoris fills). També ens permetran manipular ambdues coses, modificant la informació i l'estructura de directoris (crear directoris nous, esborrar, canviar el nom, ...). Veurem alguns d'ells, agrupats per categories.

Per cert, com que ens interessa fer les aplicacions sense haver de dependre de la plataforma, ens serà molt útil poder situar-nos en l'arrel del dispositiu sense haver de posar-lo a mà. Això s'aconsegueix amb el mètode *static* de File anomenat **listRoots()**. En sistemes Linux tornarà un únic element, però en sistemes Windows tornarà l'arrel de cada unitat del sistema, per això és un array. Aquesta és una manera d'obtenir un File que apunta a l'arrel (i en el cas de Windows a l'arrel de C:):

```
File f = File.listRoots()[0];
```

Si en Windows vulguérem anar a l'arrel de **D:**, hauríem de posar **File.listRoots()[1]**, i així successivament.

Una altra cosa que pot dur a engany és que perfectament **pot no existir** el fitxer o directori especificat en la creació del File. Recordeu que no estem accedint encara al contingut dels fitxers. I perfectament podem crear un File d'un fitxer o directori que no existeix, justament per a crear-lo.

Mètodes per a obtenir el nom o la ruta

getName()	Torna el nom del fitxer o directori
getPath()	Torna la ruta (relativa)
getAbsolutePath()	Torna la ruta absoluta
getCanonicalPath()	Torna la ruta absoluta sense possibles redundàncies

getPath() dóna la ruta fins arribar al fitxer, però relativa, tal i com s'especifica en el moment de crear el File.

getAbsolutePath() dóna la ruta absoluta, des de l'arrel. En determinades ocasions poden haver redundàncies en la ruta

getCanonicalPath() dóna la ruta absoluta, des de l'arrel, i sense redundàncies. Té com a inconvenient una utilització més complicada que getAbsolutePath.

En el següent exemple s'intenta mostrar això de les redundàncies, que **getCanonicalPath()** resol completament. Observeu com per a il·lustrar l'exemple fem referència a un fitxer d'una forma complicada. Suposem que el directori actiu és **/home/usuari/workspace/Tema1**, i volem fer referència a un fitxer situat en un subdirectori anomenat **fitxers**

```
import java.io.File;
import java.io.IOException;

public class Exemple_1_3 {

   public static void main(String[] args) throws IOException {
      File f = new File ("fitxers/../fitxers/f1.txt");
      System.out.println("Nom del fitxer: " + f.getName());
      System.out.println("Ruta del fitxer: " + f.getPath());
      System.out.println("Ruta absoluta del fitxer: " +
      f.getAbsolutePath());
      System.out.println("Ruta canònica del fitxer: " +
      f.getCanonicalPath());
   }
}
```

Si suposem que el directori actiu és **/home/usuari/eclipse-workspace/Tema1** (recordeu que per defecte el

directori actiu és el directori on està el projecte), el resultat serà:

```
Nom del fitxer: f1.txt
Ruta del fitxer: fitxers/../fitxers/f1.txt
Ruta absoluta del fitxer: /home/usuari/eclipse-workspace/Tema1/fitxers
/../fitxers/f1.txt
Ruta absoluta del fitxer: /home/usuari/eclipse-workspace/Tema1/fitxers
/f1.txt
```

Recordeu que no cal que existesca el fitxer **f1.txt**, o el subdirectori **fitxers**.

Mètodes per a obtenir els fills o el pare

list()	Torna un **array de Strings** amb els noms de tots els elements continguts en el File
listFiles()	Torna un **array de Files** amb tots els elements continguts en el File
getParent()	Torna el **nom** (string) del pare (si no existeix per ser l'arrel, tornarà nul)
getParentFile()	Torna el **pare com un File** (si no existeix per ser l'arrel, tornarà nul)

Ja hem vist la utilitat de **list()**, que torna un array de strings. En ocasions ens serà de moltíssima utilitat **listFiles()**, ja que torna un array de Files. Si a açò adjuntem els mètodes **getParent()** i **getParentFile()**, veiem que podrem navegar pel sistema de fitxers.

Mètodes per veure l'existència i característiques

exists()	Torna true si el fitxer o directori existeix
isDirectory()	Torna true si és un directori
isFile()	Torna true si és un fitxer
length()	Torna la grandària del fitxer en bytes
lastModified()	Torna la data de modificació del fitxer o directori
setLastModified()	Actualitza la data de modificació del fitxer o directori

Com ja havíem comentat abans, en el moment de crear el File, pot ser existesca o no el fitxer o directori, és a dir, que potser es corresponga o no amb un fitxer real. Si volem comprovar l'existència podem utilitzar el mètode **exists()**.

Anem a modificar l'exemple 2, on tornàvem tots els fitxers i directoris d'un directori introduït per teclat. Primer ens assegurrem que existeix i és un directori. Després el millorarem tornant la grandària de cada fitxer si és un fitxer, i especificant que és un directori, si ho és. Ens convindrà **listFiles()** per a poder mirar si és un fitxer o directori, la grandària, ... També hem aprofitat per crear el mètode estàtic **llistat(File)**, que mostrarà el contingut del directori on apunta el File, i així estructurar-lo un poc millor:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

public class Exemple_1_4 {
    public static void main(String[] args) throws IOException {
        System.out.println("Introdueix un directori:");
        String ent = new BufferedReader(new
InputStreamReader(System.in))
                .readLine();
        File f = new File(ent);
        if (f.exists()){
            if (f.isDirectory()){
                llistaDirectori(f);
```

```
            }
            else
                System.out.println("No és un directori");
        }
        else
            System.out.println("No existeix el directori");
    }

    private static void llistaDirectori(File f) throws IOException {
        System.out.println("Llista de fitxers i directoris del directori
" + f.getCanonicalPath());

System.out.println("-----------------------------------------------------");
        for (File e : f.listFiles()) {
            if (e.isFile())
                System.out.println(e.getName() + " " + e.length());
            if (e.isDirectory())
                System.out.println(e.getName() + " <Directori>");
        }

    }

}
```

Mètodes per als permisos

Els següents mètodes ens permeten consultar i modificar els permisos del File, al més pur estil Linux

canRead()	Torna true si es té permís de lectura sobre el fitxer o directori
canWrite()	Torna true si es té permís d'escriptura sobre el fitxer o directori
canExecute()	Torna true si és executable
setReadable(Boolean,Boolean)	Dóna permís o no de lectura sobre el fitxer, segons el primer paràmetre. En el segon indiquem si afecta només al propietari (true) o a tot el món (false)
setWritable(Boolean,Boolean)	Dóna permís d'escriptura o no, segons el valor del primer paràmetre. El segon actua igual que abans
setExecutable(Boolean,Boolean)	Dóna permís d'execució, segons el valor del primer paràmetre. El segon actua igual que abans

Mètodes de creació i esborrat

Ens permetran crear diectoris, fitxers buits i esborrar-los

createNewFile()	Crea un fitxer nou buit associat al File, sempre que no existesca ja un amb el mateix nom
delete()	Esborra el fitxer o directori
mkdir()	Crea un directori amb el nom indicat en la creació del File. Ha d'existir el directori pare
mkdirs()	Com l'anterior, però si cal crea tots els directoris de la ruta necessaris
renameTo(String nou_nom)	Canvia el nom del fitxer o directori

Mètodes sobre l'espai del dispositiu

També disposem de mètodes que ens diuen l'espai total i lliure del dispositiu on està situat el File

getFreeSpace()	Torna l'espai lliure del dispositiu on està situat el File
getUsableSpace()	Torna l'espai utilitzable per l'aplicació (menor que l'espai lliure)
getTotalSpace()	Torna l'espai total del dispositiu on està situat el File

Exercici

Exercici 1

Realitza un programa que permeta navegar pels directoris de la unitat principal del sistema d'arxius.

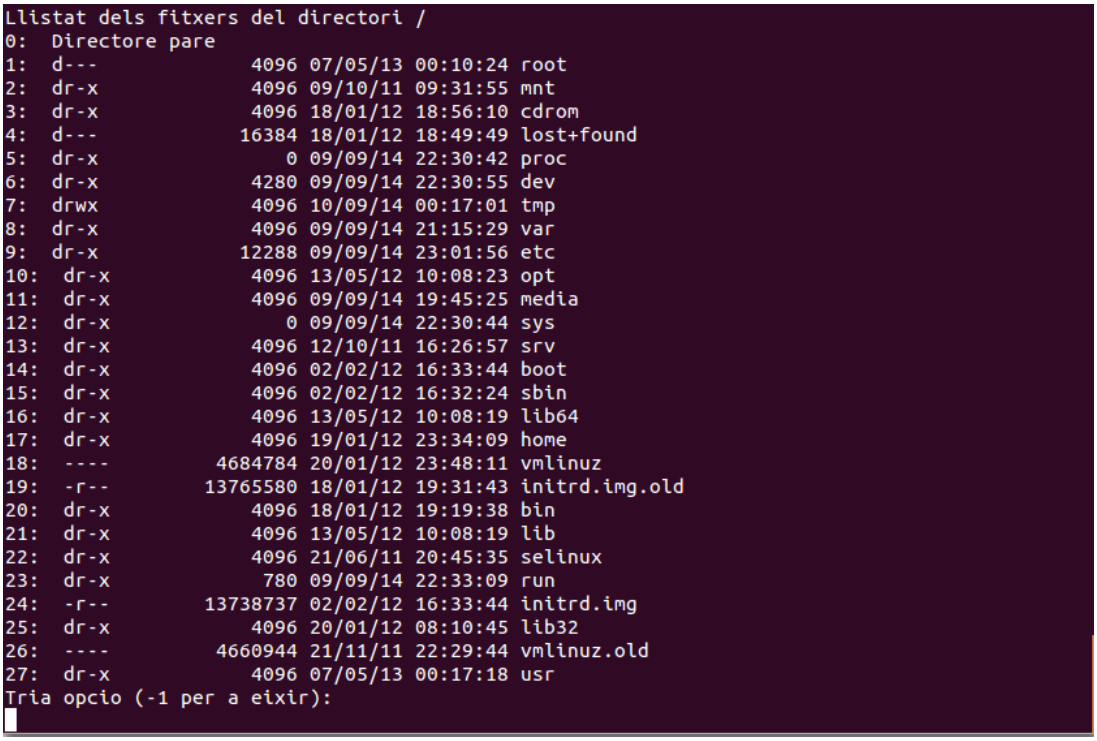
- Ha de començar per l'arrel (/ en Linux; c:\ en Windows). Recordeu que el mètode estàtic **File.listRoots()[0]** ens dóna l'arrel.
- Ha d'indicar el directori que està mostrant.
- Ha de posar com a primera opció anar al directori pare (opció 0).
- Ha de posar un número davant de cada arxiu o subdirectori que s'està mostrant. Observeu que aquest número comença amb 1 (el 0 és per al pare). Si us heu guardat en un array la llista de fitxers i directoris del directori actual, recordeu que el primer element és el 0 (però vosaltres el mostrareu amb un 1 davant).
- En cas de ser un arxiu ha de dir la grandària. En cas de ser un subdirectori, ha d'indicar-lo amb **<directori>**
- Posteriorment ha de deixar introduir un número. Les opcions seran:
  - -1 per acabar
  - 0 anar al directori pare.
  - Qualsevol altre número ha de servir per canviar a aquest directori com a directori actiu. Si era un fitxer, no ha de fer res (en la imatge, no s'ha de poder anar al 19, ja que és un fitxer).
  - S'ha de controlar que existeix el pare (en el cas de l'arrel, no en té). Si no en té, no s'ha de fer res.
  - S'ha de controlar que hi ha permís de lectura sobre un directori, abans de canviar a ell, sinó donarà error (en la imatge, per exemple, segurament no es podrà canviar al directori **root**, ja que no tindrem permís de lectura sobre ell)
  - I s'ha de controlar que el número introduït està en el rang correcte (en la imatge, de -1 fins a 27)

La següent imatge mostra el resultat:

```
Llistat dels fitxers del directori /
0.- Directori pare
1.- root        <directori>
2.- mnt <directori>
3.- cdrom       <directori>
4.- lost+found  <directori>
5.- proc        <directori>
6.- dev <directori>
7.- tmp <directori>
8.- var <directori>
9.- etc <directori>
10.- opt        <directori>
11.- media      <directori>
12.- sys        <directori>
13.- srv        <directori>
14.- boot       <directori>
15.- sbin       <directori>
16.- lib64      <directori>
17.- home       <directori>
18.- vmlinuz    4684784 bytes
19.- initrd.img.old     13765580 bytes
20.- bin        <directori>
21.- lib        <directori>
22.- selinux    <directori>
23.- run        <directori>
24.- initrd.img 13738737 bytes
25.- lib32      <directori>
26.- vmlinuz.old        4660944 bytes
27.- usr        <directori>
Tria opció (-1 per a eixir):
```

Voluntari

Modifica l'anterior per a que també ens proporcione dades sobre si és un directori, els permisos, grandària i data de modificació a l'estil de Linux quan fas **ls -l** . L'aspecte podria ser aquest:



Una altra modificació podria ser que aparegueren en ordre alfabètic.