

0101010
0100101
1101010

UD11 - ACCÉS I MANIPULACIÓ DE FITXERS

Programació – 1r DAW/DAM

0. CONTINGUTS

- Què és un fitxer?
- Fitxers en Java. Paquets `java.io` i `java.nio.file`.
- Fluxos o *streams*
- Fitxer binaris i fitxers de text
- Formes d'accés a un arxiu
- Operacions sobre arxius
- Classes per a la gestió de fluxos de caràcters

1. Introducció

- La **informació continguda en variables, arrays, objectes** o qualsevol altra **estructura de dades** és volàtil (es perd quan es tanca el programa).
- Els **fitxers** permeten **emmagatzemar informació** de manera persistent perquè pugui ser utilitzada en el moment que es necessita
 - *Rànquing d'un joc.*
 - *Viatges donats d'alta en una aplicació per a compartir cotxe*
 - *Carta d'un Restaurant.*



1. Introducció

- Un **ús comú** és l'emmagatzematge de **paràmetres de configuració** d'una **aplicació/joc** (*d'aquesta manera podrem canviar-los sense necessitat de recompilar*).
 - Credencials d'accés a la base de dades.
 - Direcció de xarxa i port de connexió.
 - Idioma de l'aplicació ...

.env

```
DATABASE_URL=mysql://app:12345678@127.0.0.1:3306/automatricula
LOGOUT_URL=http://www.cipfpbatoi.es
SALT_DOC_FINGERPRINT=212c9bf57a83f7b4ecae61232
INTRANET_API=https://intranet.cipfpbatoi.es/api
INTRANET_API_TOKEN=515tr9WoHESlm0RRgzPnVfXewSG3tiQJfj8qE8LM3PKjtMUXLrQNzso
```

1.1 Què és un fitxer?

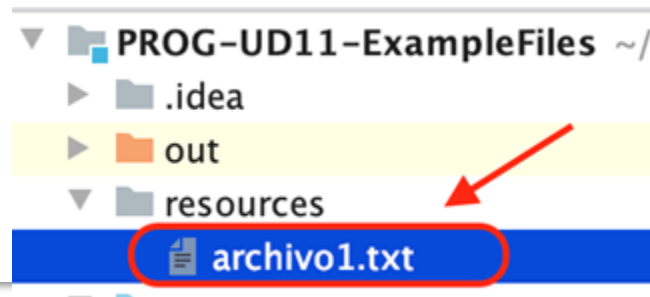
- Un **fitxer** o **arxiu** és un **conjunt de bits** emmagatzemats en un dispositiu de memòria de manera permanent.
- Els arxius tenen una sèrie de **propietats**:
nom, extensió, grandària (bytes), ruta on estan situats, etc.

```
batoi@server$ ls -la
drwxr-xr-x 6 batoi staff 192 18 abr 16.38 .
drwxr-xr-x+ 53 batoi staff 1696 19 abr 19.19 ..
drwxr-xr-x 14 batoi staff 448 28 ena 12.27 ddaw-ud4-a1
drwxr-xr-x 17 batoi staff 544 9 ena 10.44 getting-started
drwxr-xr-x 29 batoi staff 928 18 abr 16.38 omdb-postgresql
drwxr-xr-x 26 batoi staff 832 16 abr 12.56 quickstart
```

2. Fitxers en Java. La classe `File`

- Qualsevol **fitxer** o **directori** que vulguem utilitzar en Java vindrà representat per la classe `File`.

```
public class Example1 {  
  
    public static void main(String[] args) {  
  
        File archivo1 = new File("resources/archivo1.txt");  
  
        System.out.println(archivo1.isFile());  
  
    }  
}
```



Ruta d'un fitxer amb nom **archivo1.txt** situat en una carpeta **resources** situada en el **directori arrel** del projecte

2. Fitxers a Java. La classe `File`

- Les classe `File` disposa de diferents constructors. Pots trobar la informació completa en la [documentació oficial](#) (package `java.io`)

Constructor	Descripció
<code>public File(String rutaCompleta)</code>	Este constructor crea un objecte FILE el que se li ha de passar tota la ruta i el nom de l'arxiu.
<code>public File(String ruta, String nombreArchivo)</code>	En este constructor s'ha de passar la ruta i el nom de l'arxiu.
<code>public File(File rutaDirectorio, String nombreArchivo)</code>	Crea un arxiu amb el nom nombreArchivo dins del directori representat per rutaDirectorio

2.1 La classe `File`. Constructors

Exemple:

```
public class Example2 {  
  
    public static void main(String[] args) {  
  
        // primer constructor  
        File f1 = new File("resources/prog-daw-dam/dades.txt");  
  
        // segon constructor  
        File f2 = new File("resources/prog-daw-dam/", "dades.txt");  
  
        // tercer constructor  
        File directori = new File("resources/prog-daw-dam/");  
        File f3 = new File (directori, "dades.txt");  
  
    }  
}
```


2.2 Referència a fitxers. Rutes

- Les **rules** d'accés als fitxers s'especifiquen de forma diferents en sistemes operatius **Linux / Windows**:

"resources/prog-daw-dam/" vs "resources\\prog-daw-dam\\"

- Si volem que el nostre **programa** siga **multiplataforma**, podem fer ús de la constant *separatorChar* definida en la classe *File*.

```
File f = new File("resources" + File.separatorChar + "dades.txt");
```

També es pot traure amb:

```
System.getProperty("file.separator")
```

2.2 Referència a fitxers. Rutes

Exemple:

```
public class Example3 {  
  
    public static void main(String[] args) {  
  
        File fLinux = new File ("resources/dades.txt");  
        File fWindows = new File ("resources\\dades.txt");  
        File fMultiplataforma =  
            new File ("resources" + File.separatorChar + "dades.txt");  
  
        System.out.println(fLinux.isFile());  
        System.out.println(fWindows.isFile());  
        System.out.println(fMultiplataforma.isFile());  
  
    }  
  
}
```

2.2 Referència a fitxers. Rutes

• 2.2.1 Rutes absolutes vs relatives

- **Ruta absoluta:** Indica la ruta completa del fitxer dins del sistema operatiu de l'usuari. Ha començar per / en linux/mac o \\ en windows).

```
// Ruta absoluta
```

```
File fAbsoluta = new File("/home/batoi/.../dades.txt");
```

- **Ruta relativa:** Indiquem la ruta a partir del directori arrel de la nostra aplicació.

```
// Ruta relativa
```

```
File fRelativa = new File("resources/dades.txt");
```

Les rutes absolutes donaran lloc a problemes d'execució del programa en diferents hosts.

2.2 Referència a fitxers. Rutes

• 2.2.2 Ruta a la carpeta d'usuari

- Si, des de la nostra aplicació Java, necessitem saber quina es la carpeta de treball de l'usuari que està executant:

```
// Ruta carpeta usuari
```

```
String rutaUsuari = System.getProperty("user.home");
```

- Si l'usuari es *batoi*:
 - En linux tornaria: **/home/batoi**
 - En windows: **C:\Users\batoi**
 - En mac: **/Users/batoi**

2.3 Classe File. Mètodes (I)

Mètode	Descripció
<code>boolean isDirectory()</code>	Servix per a saber si estem treballant amb un arxiu o amb un directori .
<code>boolean isFile()</code>	Complementari de l'anterior, servix per a saber si estem treballant amb un arxiu o amb un directori.
<code>boolean exists()</code>	Per a assegurar-nos que l'arxiu existix realment.
<code>boolean delete()</code>	Permet eliminar l'arxiu o directori al qual fa referència de l'objecte <i>File</i> .
<code>boolean renameTo(File dest)</code>	Permet canviar de nom l'arxiu o directori al qual fa referència l'objecte <i>file</i> . Se li passa com a paràmetre un <i>Fila</i> amb nou nom.
<code>boolean createNewFile()</code> throws IOException	Crea un fitxer buit (si no existix)

2.3 Classe File. Mètodes (II)

Mètode	Descripció
<code>boolean canRead()</code>	Ens permet saber si tenim permís de lectura sobre l'arxiu.
<code>boolean canWrite()</code>	Ens permet saber si tenim permís d'escriptura sobre el fitxer.
<code>String getPath()</code>	Retorna la ruta relativa de l'arxiu.
<code>String getAbsolutePath()</code>	Retorna la ruta absoluta de l'arxiu (inclou el nom de l'arxiu).
<code>String getName()</code>	Retorna el nom de l'arxiu .



2.3 Classe File. Mètodes (III)

Mètode	Descripció
<code>String getParent()</code>	Retorna el directori pare (del qual penja l'arxiu o directori).
<code>long length()</code>	Retorna la grandària de l'arxiu en bytes .
<code>boolean mkdir()</code>	Crea un directori
<code>String[] list()</code>	Retorna un array d'objectes <i>string</i> amb els noms els fitxers / directoris que conté el directori.
<code>File[] listfiles()</code>	Retorna un array d'objectes <i>file</i> amb els noms dels arxius / directoris que conté el directori.

2.4 Package `java.nio.file`

- `java.nio.file`: nou paquet per al tractament de fitxers en java.
- Disponible a partir de la versió 1.7 de Java.
- Millora certes limitacions de `java.io`.
- Clases més importants: `Files`, `Paths`, `Path`.
- Classe `Path` ve a substituir a la `File`.

```
File f = new File("/home/batoi/hola.txt");  
Path p = Paths.get("/home/batoi/hola.txt");
```

- Encara que `File` no està obsoleta, es **recomana considerar l'ús de `Path`**.
- No obstant això, encara s'utilitza en molts projectes de Java (codi heretat, aplicacions més antigues o que encara no s'ha migrat a les noves classes)

2.4 Package `java.nio.file`

- **Avantatges:** més funcionalitats, més seguretat i robustesa en la gestió I/O de java.
- Classe **Files**: ofereix molts mètodes estàtics amb operacions útils de I/O (manipulació d'arxius, lectura i escriptura, etc)
- Classe **Paths**: proporciona mètodes estàtics per crear instàncies d'objecte Path.

- **Transformació de File a Path**

```
File f = new File("/home/batoi/hola.txt");  
Path p = f.toPath();
```

- **Transformació de Path a File**

```
Path p = Paths.get("/home/batoi/hola.txt");  
File f = p.toFile();
```

Activitat Prèvia

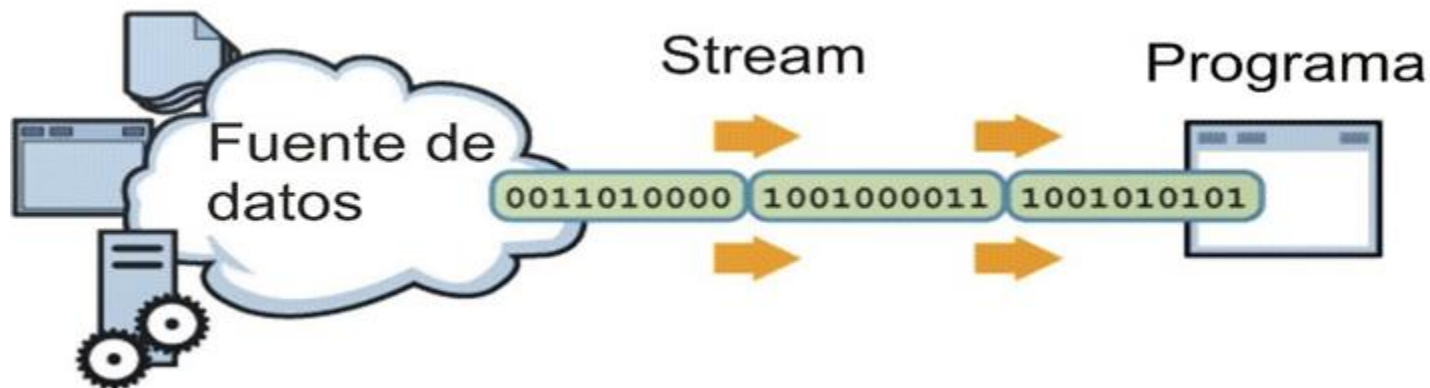
Activitat 1. Implementa una classe FileManager que dispose dels següents **mètodes estàtics** respectant les capçaleres donades:

- `boolean crearFichero (String rutaDirectorio, String nombreArchivo) throws IOException`: crearà un nou arxiu en el directori especificat.
- `void verArchivosDirectorio(String rutaDirectorio)`: Visualitzarà el contingut del directori passat com a paràmetre.
- `void verInformacionArchivo(String rutaDirectorio, String nombreArchivo)`: visualitzarà el nom, la ruta absoluta, si es pot escriure, si es pot llegir, i si és un directori o un arxiu.

Nota: Tots els arxius (els creats i els consultats) han d'ubicar-se en una carpeta `resources/activitat1` situada en l'arrel del projecte.

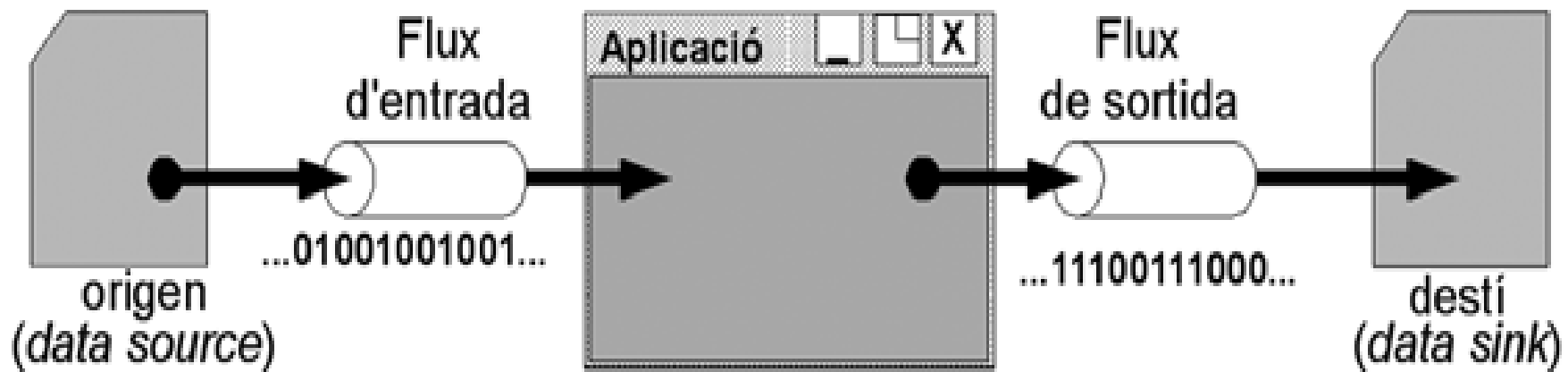
3. Fluxos o *streams*

- En **java** l'accés a la informació d'un fitxer, es fa a través d'un **flux o stream**.
- Es tracta d'una **abstracció** que pot veure's com una connexió entre una **font** i un **destí** per la qual viatja una **seqüència de bits**.



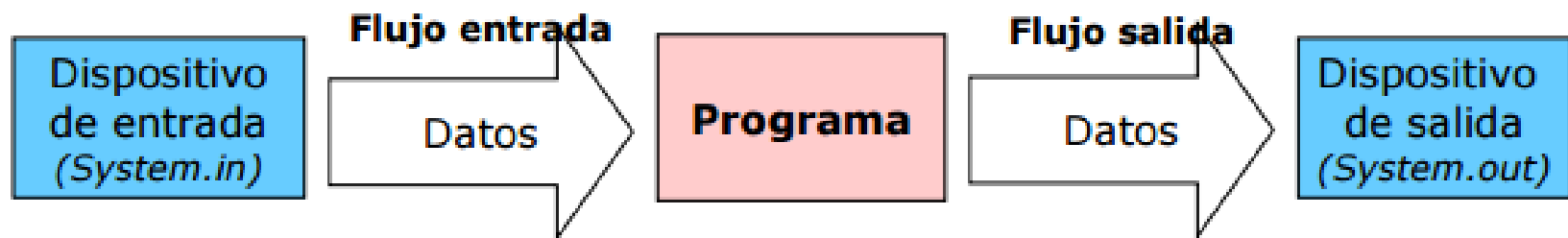
3. Fluxos o *streams*

- Un flux o **stream** té una única **font d'entrada** i un **destí**, per la qual cosa si necessitem llegir i escriure haurem de tindre diferents fluxos.



3.1 Fluxos predeterminats

- Existixen **alguns fluxos estàndard** que ja hem utilitzat i que estan disponibles **sense necessitat de crear-los**:
 - System.out**: eixida estàndard
 - System.in**: entrada estàndard
 - System.err**: eixida d'errors

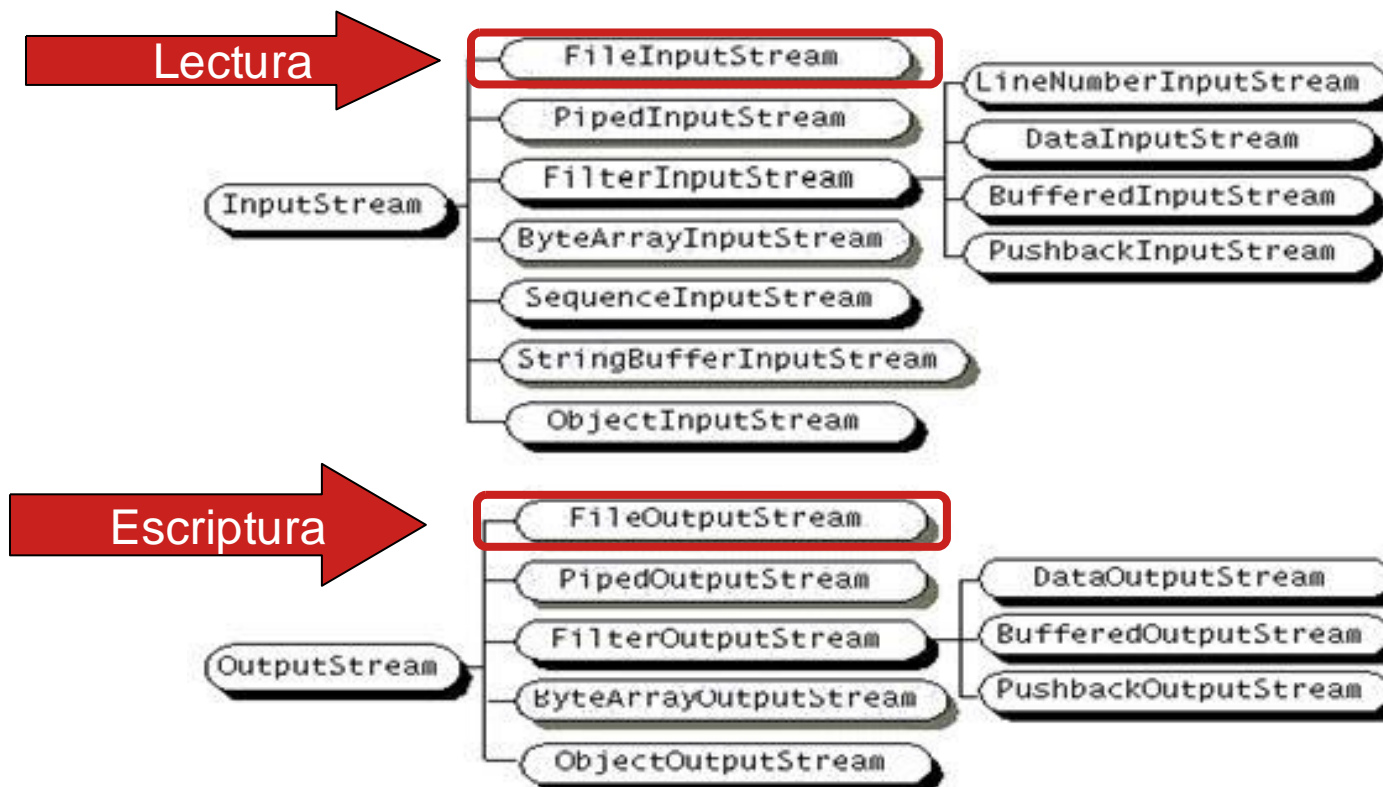


3.2 Tipus

Fluxos de bytes (8 bits)

- Lectura i escriptura de dades binàries. **Arxius binaris.**

Ex. imatges, programes compilats,...



3.2 Tipus

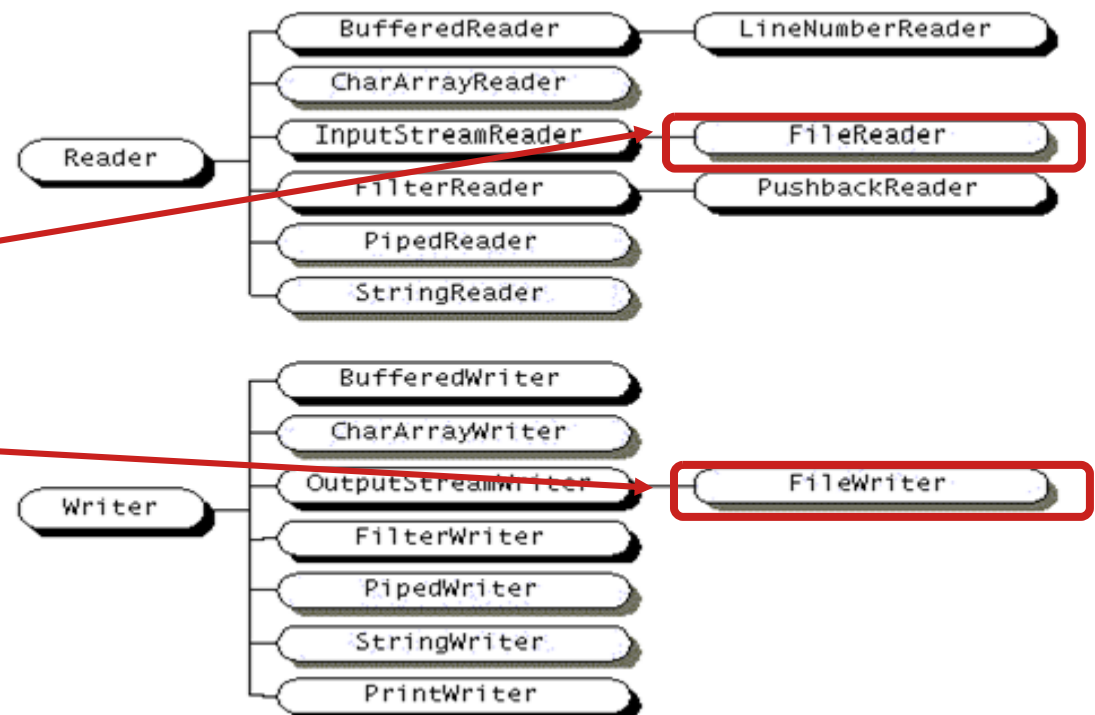
Fluxos de caràcters Unicode (16 bits)

- Lectura i escriptura de caràcters. **Arxius de text**

Ex. fitxer log, fitxer codi font java, fitxer de configuracion, xml, ...

- Opera directament sobre caràcters pel que el **contingut** del fitxer és **llegible** per l'usuari.

Ens centrarem en els
fluxos de tipus
caràcter
per a analitzar el seu
funcionament



4. Fitxers orientats a caràcters

- Un **fitxer o arxiu de text** és un conjunt de caràcters emmagatzemats en un dispositiu de memòria de manera permanent.
- Podríem donar-li una estructura a eixe conjunt de caràcters perquè ens servisca per a emmagatzemar dades útils com **registres** organitzats en **camp**s (malgrat que això no seria una bona idea hui en dia).



Juan Pérez;20;\$500
Ana Alban;28;\$1000
José Mora;30;\$1500

Archivo Clientes

Juan Pérez	20	\$500
Ana Alban	28	\$1000
José Mora	30	\$1500

registro

campo

4.1 Operacions sobre fitxers

- **Creació**
 - El fitxer es crea en disc amb un nom.
 - Es fa l'operació una única vegada.
- **Obertura del Flux**
 - Per a poder treballar amb el contingut de l'arxiu ha d'obrir-se i establir-se **un flux de dades** sobre este.
- **Lectura / Escriptura de dades**
 - Transferència d'informació **des de/cap al** fitxer
- **Tancament del flux de dades**
 - L'arxiu ha de tancar-se quan acabem de treballar amb ell.

Si no duem a terme correctament el tancament no podrem assegurar que la informació es transfereix correctament.

5. Fluxos de caràcters

- Operen directament sobre caràcters **alfanumèrics** utilitzant la codificació estàndard UNICODE.
- Ens basarem en les implementacions `FileReader` i `FileWriter`
 - `FileReader`: lectura de caràcters.
 - `FileWriter`: escriptura de caràcters
- Haurem de prestar especial atenció a la següents excepcions:
 - `FileNotFoundException` (no existix el nom de l'arxiu o no és vàlid).
 - `IOException` (el disc està ple o protegit contra escriptura)

5.1 Lectura d'un arxiu

- Els passos per llegir un fitxer són:
 1. Crear un **objecte** de tipus **File** que faci referència al fitxer (**opcional**).
 2. Crear el **flux d'entrada** de dades amb **FileReader** (*l'arxiu ha d'existir*).
 3. Realitzar les operacions de **lectura**.
 4. **Tancar** el flux de dades.

5.2 *FileReader*. Constructores

Constructors	Descripció
<code>FileReader(File file)</code>	Crea un objecte <i>FileReader</i> a partir de l'arxiu des del qual es vol llegir.
<code>FileReader(String fileName)</code>	Crea un objecte <i>FileReader</i> a partir de l'arxiu des del qual es vol llegir. se li rep com a paràmetre la ruta de l'arxiu.

```
File file = new File("resources" + File.separator, "fitxer1.txt");
FileReader fileReader = null;
try {
    fileReader = new FileReader(file);
} catch (FileNotFoundException e) {
    System.out.println("L'arxiu no existeix");
}
```

L'excepció es llançarà
si el fitxer no existix

5.2 FileReader. Mètodes

Mètodes	Descripció
<pre>int read()</pre>	<p>Llig un caràcter i el retorna la seua codificació. Retorna -1 quan arriba al final de l'arxiu.</p>
<pre>int read(char[] cbuf)</pre>	<p>Llig 'n' caràcters ($n \leq \text{cbuf.length}$). Retorna el nombre de caràcters llegits. Els caràcters llegits estan en les posicions $[0..n-1]$ del array cbuf. Retorna -1 si s'aconsegueix el final de l'arxiu.</p>
<pre>int read(char[] cbuf, int offset, int len)</pre>	<p>Llig 'n' caràcters. Retorna el nombre de caràcters llegits. Els caràcters llegits comencen en la posició offset. Retorna -1 si l'arxiu s'ha acabat.</p>
<pre>void close()</pre>	<p>Tanca l'arxiu.</p>

**** No anem a vore ni utilitzar aquests mètodes.**
Directament anem a utilitzar **BufferedReader** ******

5.3 Exemple (I)

```
private static String leerContenidoFichero() {  
    StringBuilder contenidoFichero = new StringBuilder();  
    try {  
        FileReader fileReader = new FileReader("ex4/fichero1.txt");  
        Int character;  
        do {  
            character = fileReader.read();  
            if (character >= 0) {  
                contenidoFichero.append((char) character);  
            } else {  
                fileReader.close();  
            }  
        } while (character >= 0);  
    } catch (FileNotFoundException e) {  
        System.out.println("El fitxer que vols llegir no existix");  
    } catch (IOException e) {  
        System.out.println("S'ha produït un error en l'accés al fitxer");  
    }  
    return contenidoFichero.toString();  
}
```

1. Creem l'objecte
que representa el flux

2. Llegim caràcter a caràcter
fins que arribe al final
character == -1

3. Tanquem el Stream

Activitat Prèvia

- **Activitat 2 (opcional).**- A partir del codi descrit en l'exemple anterior. Crea un programa per a llegir el contingut d'un arxiu anomenat `fitxer.txt` el contingut del qual serà el teu nom i cognoms.

resources/activitat2/fitxer.txt

NombreAlumno Apellido1 Apellido2

5.4 Escriptura d'un arxiu

Els passos per al treball d'escriptura amb un fitxer són:

- Crear un objecte de tipus `File` que faci referència al fitxer (**opcional**).
- Crear el **flux d'eixida** de dades amb `FileWriter` (*si no existix l'arxiu es crea automàticament*).
- Realitzar les operacions **d'escriptura**.
- **Tancar** el flux de dades.

5.5 *FileWriter*. Constructors

Constructors	Descripció
<code>FileWriter (File file)</code>	Donat un objecte <i>file</i> que serà l'arxiu sobre el qual es vol escriure, construeix un objecte <i>FileWriter</i> sobre ell.
<code>FileWriter (File file, boolean append)</code>	Donat un objecte <i>file</i> que serà l'arxiu sobre el qual es vol escriure, construeix un objecte <i>FileWriter</i> sobre ell al qual es poden afegir dades (No sobreescriu).
<code>FileWriter (String fileName)</code>	Donada la ruta de l'arxiu sobre el qual es vol escriure, construeix un objecte <i>FileWriter</i> sobre ell.
<code>FileWriter (String fileName, boolean append)</code>	Donada la ruta de l'arxiu sobre el qual es vol escriure, construeix un objecte <i>FileWriter</i> sobre ell al qual es poden afegir dades (no sobreescriu).



5.6 FileWriter. Mètodes (I)

Mètodes		Descripció
void write (int)	<p>** No anem a vore ni utilitzar aquests mètodes. Directament anem a utilitzar <code>BufferedWriter</code> **</p>	Escriu un caràcter
Writer append (char c)		Afegir un caràcter a un fitxer
void write (char [] cbuf)		Escriu en el fitxer del array de caràcters.
void write (char [] cbuf, int off, int len)		Escriu en el fitxer els 'n' caràcters del array cbuf a partir de la posició off.

5.6 FileWriter. Mètodes (II)

**** No anem a vore ni utilitzar aquests mètodes. Directament anem a utilitzar **BufferedWriter** ****

Mètodes	Descripció
<code>void write (String str)</code>	Escriu en l'arxiu la cadena 'str'.
<code>void write (String str, int off, int len)</code>	Escriu els caràcters de la cadena 'str' a partir de la posició off.
<code>void flush()</code>	Assegurar que tots els caràcters queden ben escrits en disc, sense tancar el flux.
<code>void close()</code>	Tanca el flux, assegurant que tot queda ben escrit en disc.

5.7 Example (I)

```
public class Exemple5 {  
    public static void main(String[] args) {  
        escribirFichero("Programació 1 DAM/DAW");  
    }  
    private static void escribirFichero(String cadena) {  
        File file = new File("exemple5/fichero1.txt");  
        try {  
            FileWriter fileWriter = new FileWriter(file);  
            fileWriter.write(cadena);  
            fileWriter.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

1. Creem l'objecte
que representa
el flux

2. Escrivim la cadena

3. Tanquem el Stream

Activitat Prèvia

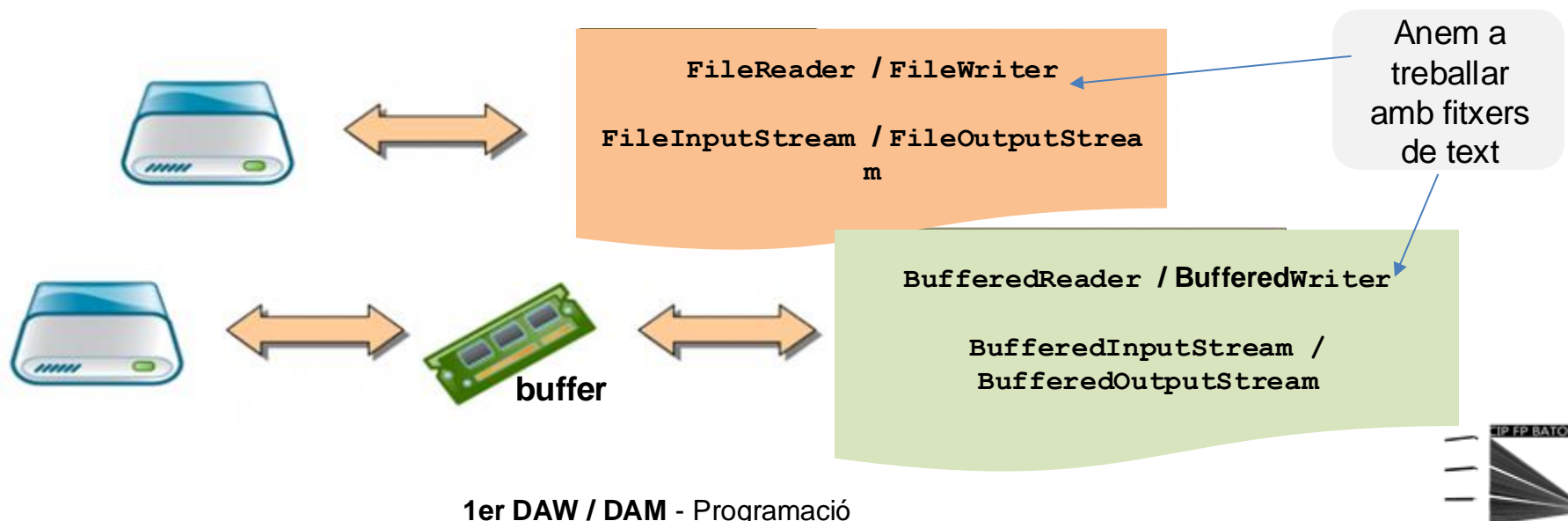
- **Activitat 3 (opcional).**- A partir del codi descrit en l'exemple anterior. Crea un programa que escriga el teu nom i cognoms en un arxiu anomenat `fitxer.txt`

resources/activitat3/fitxer.txt

NombreAlumno Apellido1 Apellido2

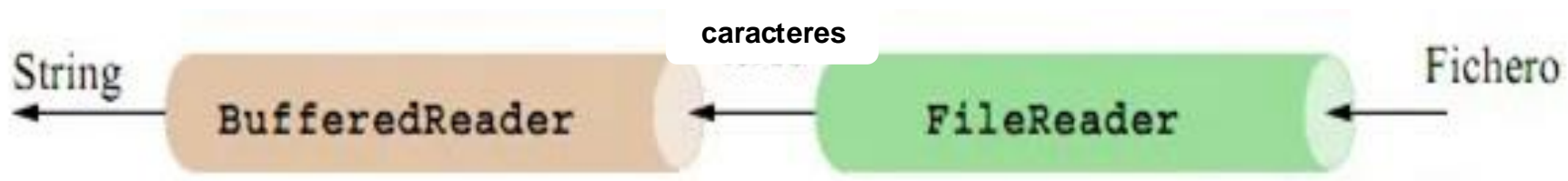
6. La classe `Buffered`

- Les classes `FileReader`, `FileWriter` i els seus equivalents per al maneig d'arxius binaris (`FileInputStream` i `FileOutputStream`) realitzen les **operacions directament** sobre el **dispositiu** d'emmagatzematge.
 - Si l'aplicació fa estes **operacions** de forma molt **repetida**, la seua **execució serà lenta**. A més, el seu ús és **més complex**.
- Per a millorar esta situació, es pot utilitzar un **buffer intermedi**.



6. La classe `Buffered`

- La paraula *buffered* fa referència a la **capacitat** d'emmagatzematge **temporal** en la lectura i escriptura:
 - En les **operacions de lectura** es **lligen més dades** de les que realment es necessiten, de manera que en una possible operació posterior les dades ja es troben en memòria.
 - En les **operacions d'escriptura**, les dades es van guardant en memòria i **no es bolquen a disc** fins que hi haja una **determinada quantitat** de dades.



6.1 La classe `BufferedReader`

- Constructors i mètodes

Constructor	Descripció
<code>BufferedReader(Reader in)</code> Necessitem un <code>FileReader</code>	Crea un buffer per a emmagatzemar els caràcters del flux d'entrada (a través del que es llig la informació de l'arxiu).
Mètodes	Descripció
<code>String readLine()</code>	Llig una línia de text .
<code>int read()</code>	Llig un sol caràcter.
<code>void close()</code>	Tanca el <i>buffer</i> .

6.1 La classe BufferedReader

Exemple

```
public class Example6 {  
    private static void leerFicheroConBuffered() {  
        BufferedReader bufferedReader = null;  
        try {  
            File file = new File("exemple6/fitxer6.txt");  
            FileReader fileReader = new FileReader(file);  
            BufferedReader = new BufferedReader(fileReader);  
            do {  
                String line = bufferedReader.readLine();  
                if (line == null) {  
                    bufferedReader.close();  
                    return;  
                }  
                System.out.println(line);  
            } while (true);  
        } catch (IOException e) {  
            System.out.println("S'ha produït durant la lectura de l'arxiu" + e.getMessage());  
        }  
    }  
}
```

1. Obrim el Stream de dades

2. Inicialitzem el buffer

3. Llegim línia per línia fins que retorna null

4. Tanquem el buffer
(també es tancarà el Stream)

6.1 La classe `BufferedReader`

Cas particular.

- Lectura des de teclat (entrada estàndard).
- El procés es similar a la lectura desde fitxers però canviant l'orige del flux d'entrada.

Buffer d'entrada



Flux d'entrada




Representa
entrada teclat



```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
String lectura = br.readLine();  
System.out.println("L'usuari ha escrit: " + lectura);
```

6.2 La classe `BufferedWriter`

- La classe **`BufferedWriter`** disposa de mètodes per a escriure línies completes.

Constructor	Descripció
<code>BufferedWriter(Writer out)</code>  Necessitem un <code>FileWriter</code>	Crea un buffer per a emmagatzemar els caràcters del flux d'eixida (a través del qual escriurà en l'arxiu).
mètodes	Descripció
<code>void newline()</code>	Escriu una línia en blanc.
<code>void write(int c)</code>	Escriu un caràcter.
<code>void write(String s)</code>	Escriu una cadena.
<code>void close()</code>	Tanca el <i>buffer</i> .

6.2 La classe `BufferedWriter`

Exemple:

```
public class Example7 {  
    private static void escribirFicheroConBuffered() {  
        BufferedWriter bufferedWriter = null;  
        try {  
            File file = new File("exemple7/fitxer7.txt");  
            FileWriter fileWriter = new FileWriter(file, true);  
            bufferedWriter = new BufferedWriter(fileWriter);  
            bufferedWriter.write("Alumne 1");  
            bufferedWriter.newLine();  
            bufferedWriter.write("Alumne 2");  
            bufferedWriter.close();  
        } catch (IOException e) {  
            System.out.println("error escrivint en l'arxiu" + e.getMessage());  
        }  
    }  
}
```

1. Inicialitzem el Stream

Inicialitzem el Buffer

Escrivim en el fitxer

Tanquem el buffer
(també es tancarà el Stream)

Activitat Prèvia

Activitat 4.- Crea un programa que llixi el següent arxiu en el qual tenim el **codi, nom i quantitat** de productes que **hi ha en estoc** en una botiga en línia. A continuació, ha de mostrar el nombre de televisions, monitors i teclats **que es tenen**.

resources/activitat4/productes.txt

```
1,Samarreta,4
4,Pantalons,10
3,Ordinador,20
2,Vídeo-consola,20
9,Telèfon,5
6,Tauleta,8
7,Televisió,14
8,Monitor,34
5,Teclat,12
10,Ratolí,56
11,Joystick,10
12,Joc,2
```

Activitat Prèvia

Activitat 5 (opcional).- Crea un programa que llija el contingut d'un fitxer de text i el bolque en un altre fitxer de text, **convertint cada línia a majúscules**. El fitxer contindrà el nom de tots els mòduls del cicle formatiu en diferents línies.

resources/activitat5/moduls.txt

```
Llenguatge de marques  
Programació  
Bases de dades  
Entorns de desenvolupament  
Sistema Informàtics  
Formació i orientació laboral  
Anglès  
Desenrotllament d'aplicacions en entorn servidor  
Accés a Dades  
Programació multimèdia i dispositius mòbils  
Disseny d'interfícies  
Empresa i iniciativa emprenedora  
Inglés II
```

Activitat Prèvia

Activitat 6.- Imagina que necessitem guardar en un **fitxer de text** les tasques del nostre **Gestor de Tasques** (*Gestor de Tareas*, treballat en la unitat anterior).

De moment, simplifiquem un poc la informació de cada tasca. Només tindrem en compte les dades següents: `codi`, `descripció`, `usuari`, `realitzada`. L'estructura del fitxer seria així:

resources/activitat6/tasques.txt

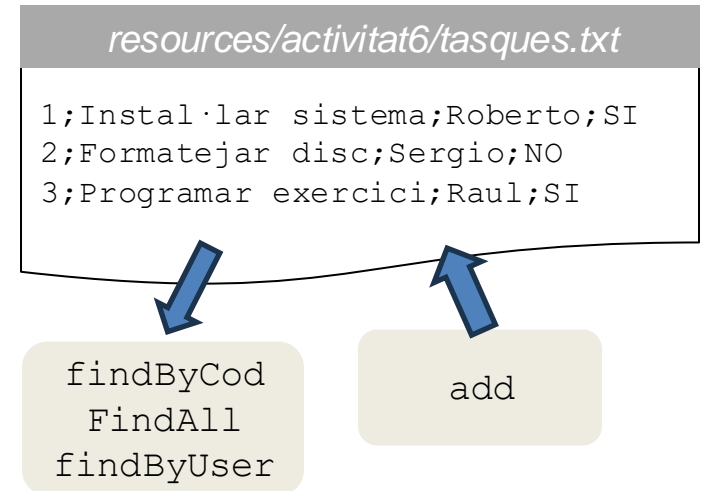
```
1;Instal·lar sistema;Roberto;SI  
2;Formatejar disc;Sergio;NO  
3;Programar exercici;Raul;SI
```

Farem ús de les classes
BufferedReader /
BufferedWriter

Activitat Prèvia

Activitat 6 (continuació) - Es tractaria de programar un conjunt de mètodes per a atendre l'**operativa bàsica** de treball sobre el fitxer: **buscar** (d'acord amb diversos criteris), **guardar** una nova tasca, **modificar**, etc. Concretament, implementarem les següents funcionalitats:

- Obtindre tasca per codi. `public Tarea findByCod(int cod)`
- Obtindre totes les tasques. `public List<Tarea> findAll()`
- Obtindre tasques d'un usuari. `public List<Tarea> findByUser(String usuari)`
- Guardar/afegir tasca. `public void add(Tarea tarea)`
- Modificar tasca. `public void update(Tarea tarea)`
- Eliminar tasca. `public void delete(Tarea tarea)`



No les farem, de moment

Aquests mètodes els encapsularem en una classe que anomenarem **FileTareaDAO** (ja explicarem més avant el perquè d'aquest nom).

6.3 try-with-resources

L'estructura `try-with-resources` permet definir una sèrie de recursos que es **tancaran automàticament** una vegada finalitzat el següent **bloc de sentències**.

Tots els **recursos** han d'implementar la **interfície** `Closeable`

```
public static String leerPrimeraLineaDelArchivo(String path)
{
    try (BufferedReader br = new BufferedReader(new FileReader(path)))
    {
        System.out.println("Llegint primera linea fitxer " + path);
        return br.readLine();
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Recursos

Bloc de sentències
després del qual es
tancaran els recursos
automàticament

Activitat Prèvia

- **Activitat 7.-** Modifica l'activitat 6 perquè faça ús d'un bloc `try-with-resources` per a tancar els buffers de lectura i escriptura.

- Això és tot... de moment :-)