Tema 2

FICHEROS

Tema 2
Ficheros (2)

Tema 2

1. Acceso a los ficheros
2. Operaciones con ficheros Java
3. Apertura y cierre de un fichero
4. Stream de Datos
5. Lectura de un fichero
6. Escritura de un fichero
7. Clasificación de los flujos
8. Los Buffered

1.- Acceso a los ficheros

- Acceso secuencial
  - Accedemos comenzando desde el principio del fichero
  - Para acceder a cualquier parte del fichero tenemos que recorrer el contenido anterior
- Acceso aleatorio
  - Podemos acceder directamente a datos en cualquier posición

- En ambos casos hacemos operaciones de lectura/Escrtura

2. Operaciones con ficheros Java

- **Apertura**: El programa abre el fichero y se prepara para leerlo o escribirlo. Suele "reservar" el fichero para sí
- **Cierre**: Indica que se ha finalizado con las operaciones sobre el fichero. Libera el fichero
- **Lectura**: Lee el fichero o una de sus partes •
- **Escritura**: Permite escribir en el fichero, ya sea añadiendo datos o sobrescribiendo los ya existentes
- **Ejecución**: Similar a la lectura, pero utiliza los datos del fichero para ejecutar un software
- **Creación**: Crea un nuevo fichero con un nombre, extensión y ruta determinados
- **Eliminación**: Elimina un fichero determinado

2. Operaciones con ficheros Java

- Acceso a un fichero se basa en un puntero
  - Siempre apunta a un lugar del fichero
  - O Posición especial " fin de fichero"
- Proceso:
  - Abrir Fichero
  - Operación de lectura **read()**
    - Con esta operación leemos el contenido y lo almacenamos en un buffer
  - **skip()** para situar el puntero a un "numero" de bytes
  - Escritura : método **write()**
    - Volvamos el contenido del buffer en el fichero
  - Cierre : **close()**

3. Apertura y cierre de un fichero

- Cuando se abre un fichero en Java, se "reserva" el fichero para operar con el
- Se establece un flujo de datos desde el fichero a una variable en Java, que representa al fichero
- A partir de esa variable, se pueden realizar todas las operaciones sobre fichero que se quieran
- Cuando se quiere dejar de usar el fichero, se debe cerrar el mismo, cortando el flujo de datos y liberando la variable

3. Apertura y cierre de un fichero

- Para manipular ficheros utilizamos la clase File
- El constructor de la clase File tiene el siguiente esquema:

```
File variableFichero = new File("ruta fichero");
```

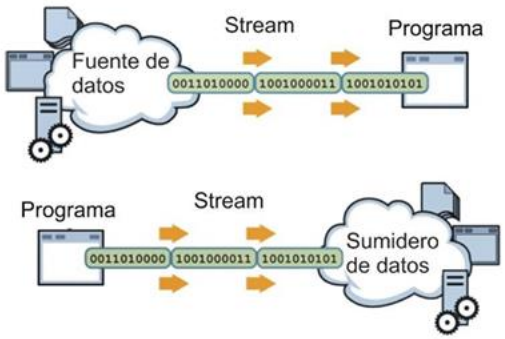
- Al crear el constructor, la variable variableFichero es un objeto con los datos del fichero que se encuentra en la ruta pasada por parámetro, si es que existe
- La ruta puede ser absoluta o relativa •
- Para cerrar un fichero, se usa la siguiente sentencia:

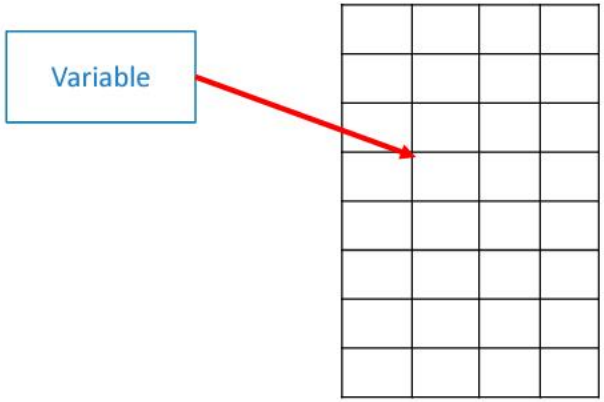
```
variableFichero.close();
```

- Con esto se cierra el fichero y la variable variableFichero pasará a ser nula (null)

4. Stream de Datos

- Para poder realizar las operaciones de lectura y escritura de ficheros, Java establece lo que se conoce como un Stream de datos
- Crea una vía de comunicación entre programa y fichero que permite "moverse" por las distintas partes del fichero
- Existe un puntero que apunta a las partes del fichero





5.- Lectura de un fichero

- Para leer datos de un fichero de texto, utilizaremos las siguientes clases:
- **Clase File**: Para representar el fichero que se quiere leer

```
File fichero = new File("ruta fichero");
```

- **Clase FileReader**: Establece el stream de datos de lectura del fichero. Tiene una serie de métodos para leer carácter a carácter. Al constructor del FileReader recibe el objeto File

```
FileReader reader = new FileReader(fichero);
```

- **Clase BufferedReader**: Crea un buffer a través del FileReader, que permite leer mas de un carácter. El constructor recibe el FileReader como parámetro

```
BufferedReader buffer = new BufferedReader (reader);
```

5.- Lectura de un fichero

- Utiliza la función del BufferedReader llamada readLine(), la cual:
  - Devuelve la siguiente línea de texto si existe
  - Si no existe, devuelve null

```
String linea = buffer.readLine();
```

- Teniendo en cuenta el funcionamiento de readLine(), se puede leer todo el fichero utilizando un bucle while

```
String linea;
while((linea=buffer.readLine()) != null) {
   System.out.println(linea);
}
```

5.- Lectura de un fichero

Ejemplo 1:

Lectura de un fichero de texto

6. Escritura de un fichero

- Para escribir datos en un fichero de texto, utilizaremos las siguientes clases:
- **Clase File**: Para representar el fichero que se quiere leer

```
File fichero = new File("ruta fichero");
```

- **Clase FileWriter**: Establece el stream de datos de escritura del fichero. Tiene una serie de métodos para escribir en ficheros. Al constructor del FileWriter recibe el objeto File

```
writer = new FileWriter(fichero);
```

- Clase PrintWriter: Crea un buffer a través del FileWriter, que permite extender los métodos del FileWriter por otros similares a los que tenemos en la salida de pantalla. El constructor recibe el FileWriter como parámetro FileWriter

```
PrintWriter pw = new PrintWriter(writer);
```

6. Escritura de un fichero

- Entre las funciones que tenemos en el PrintWriter, las mas comunes son:
- **print("texto")**. Imprime el texto pasado por parámetro
- **println("texto")**. Imprime el texto pasado por parámetro y hace un salto de línea
- El constructor del FileWriter puede recibir un segundo parámetro boolean que indica si queremos escribir el fichero desde cero (false) o si queremos añadir texto al existente (true)

```
FileWriter writer = new FileWriter(fichero);
```

```
FileWriter writer = new FileWriter(fichero, false);
```

Desde cero

```
FileWriter writer = new FileWriter(fichero, true);
```

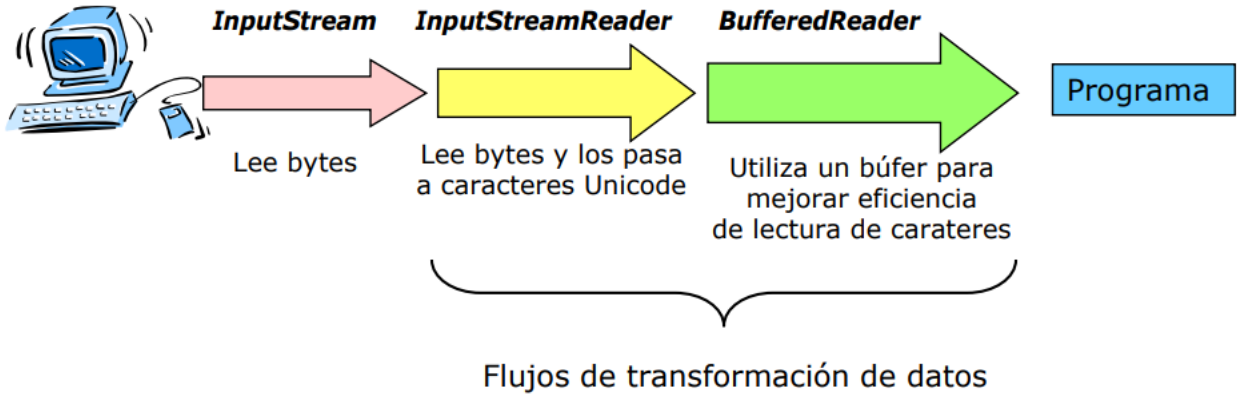
Añadir texto

6. Escritura de un fichero

Ejemplo 2

- Escritura de un fichero

7. Clasificación de los flujos



Clasificación de los flujos

- Representación de la información
  - Flujos de bytes: clases InputStream y OutputStream
  - Flujos de caracteres: clases Reader y Writer
    - Se puede pasar de un flujo de bytes a uno de caracteres con InputStreamReader y OutputStreamWriter

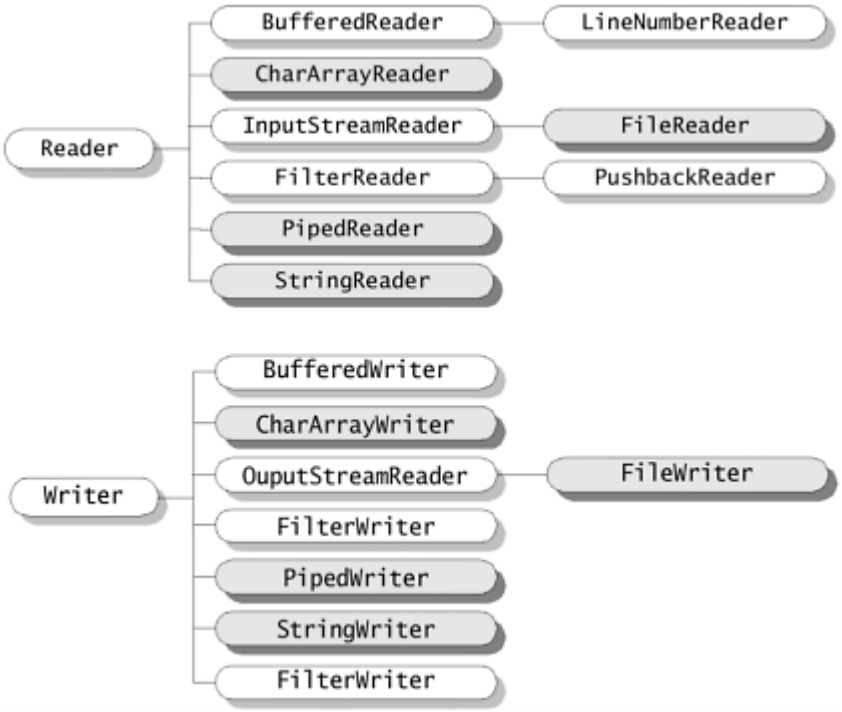
Propósito

- Entrada: InputStream, Reader
- Salida: OutputStream, Writer
- Lectura/Escritura: RandomAccessFile
- Transformación de los datos
  - Realizan algún tipo de procesamiento sobre los datos (p.e. buffering, conversiones, filtrados): BuffuredReader, BufferedWriter

Acceso secuencial a los datos

- Flujo o stream
  - Secuencia de datos
  - Para leer o escribir necesitamos crear un flujo (stream)
  - tipos de flujos :
    - Binario => byte[]
    - Texto => String

Flujo texto

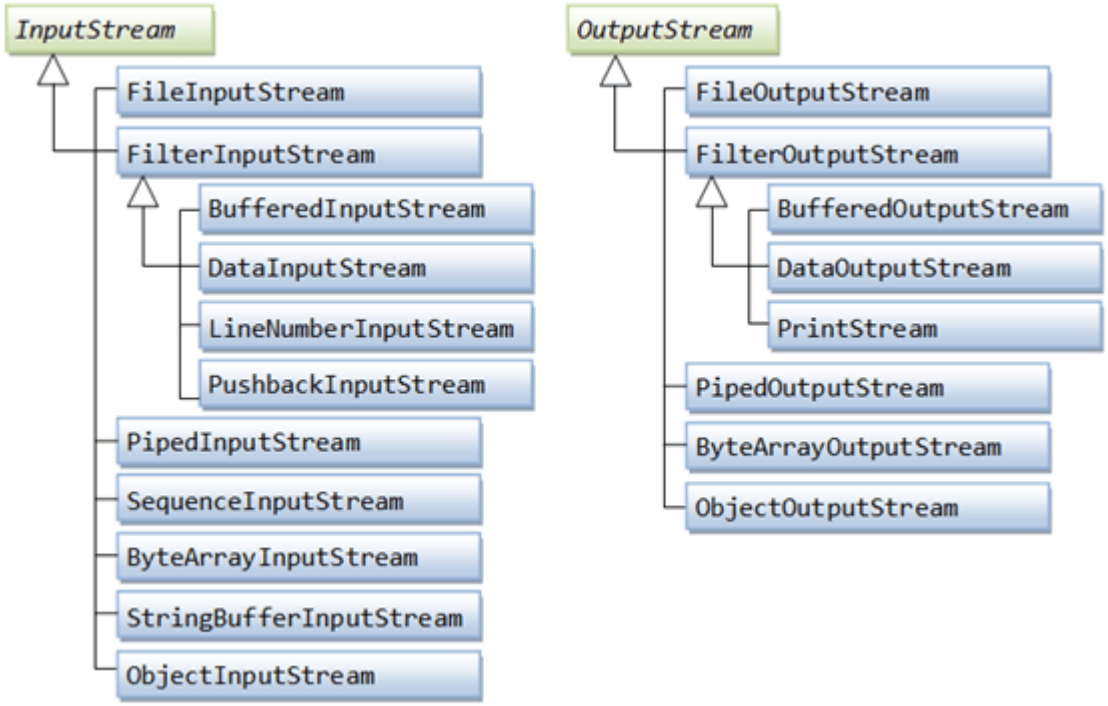

Reader
BufferedReader
LineNumberReader
CharArrayReader
InputStreamReader
FileReader
FilterReader
PushbackReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
OuputStreamReader
FileWriter
FilterWriter
PipedWriter
StringWriter
FilterWriter

Flujo texto

Ejemplo 3:

- Escritura de fichero con BufferWriter

Flujo Binario



8.- Los Buffered

- Si usamos sólo FileInputStream, FileOuputStream, FileReader o FileWriter, cada vez que hagamos una lectura o escritura, se hará fisicamente en el disco duro.
  - Si escribimos o leemos pocos caracteres cada vez, el proceso se hace costoso y lento, con muchos accesos a disco duro.
- Los BufferedReader, BufferedInputStream, BufferedWriter y BufferedOutputStream añaden un buffer intermedio. Cuando leamos o escribamos, esta clase controlará los accesos a disco.
  - Si vamos escribiendo, se guardará los datos hasta que tenga bastantes datos como para hacer la escritura eficiente.
  - Si queremos leer, la clase leerá muchos datos de golpe, aunque sólo nos dé los que hayamos pedido. En las siguientes lecturas nos dará lo que tiene almacenado, hasta que necesite leer otra vez.
- Esta forma de trabajar hace los accesos a disco más eficientes y el programa correrá más rápido. La diferencia se notará más cuanto mayor sea el fichero que queremos leer o escribir.

9.- Problemas en procesado de ficheros

- En las operaciones con ficheros pueden surgir una serie de problemas
- Algunos son:
- Intentar leer/borrar/escribir sobre un fichero que no existe
- Intentar leer/borrar/escribir sobre un fichero que ya está siendo modificado por otro programa
- Intentar escribir o leer un fichero cuando el puntero haya llegado al final del fichero (EOF)