



Los Streams

Índice

- ▶ [Definición](#)
- ▶ [E/S Estándar](#)
- ▶ [Tipos de streams](#)
- ▶ [Jerarquía de flujos de bytes](#)
 - ▶ [Subclases de InputStream](#)
 - ▶ [Subclases de OutputStream](#)
- ▶ [Jerarquía de flujos de caracteres](#)
- ▶ [Lectura/Escritura de ficheros binarios](#)
- ▶ [Lectura/Escritura de ficheros de caracteres](#)
- ▶ [Lectura/Escritura de datos de tipos primitivos](#)
- ▶ [Lectura/Escritura de objetos](#)
- ▶ [Acceso aleatorio a ficheros](#)
- ▶ [Formatos en ficheros de texto](#)



Definición

- ▶ Los flujos de datos o *Streams*, son abstracciones que permiten tratar la comunicación de información entre una fuente y un destino.
- ▶ Los flujos actúan como interfaz con el dispositivo o clase asociada:
 - ▶ Operación independiente del tipo de datos y del dispositivo, por lo que aporta mayor flexibilidad
 - ▶ Diversidad de dispositivos (fichero, pantalla, teclado, red, ...)
 - ▶ Diversidad de formas de comunicación
 - Modo de acceso: secuencial, aleatorio
 - Información intercambiada: binaria, caracteres, líneas
- ▶ Están definidos en el paquete `java.io`



E/S Standar

- ▶ En Java se accede a la E/S estándar mediante los siguientes atributos estáticos de la clase `java.lang.System`:
 - ▶ `System.in` implementa la entrada estándar
 - ▶ `System.out` implementa la salida estándar
 - ▶ `System.err` implementa la salida de errores



Tipos de Streams

- ▶ **Flujos de bytes (8 bits)**

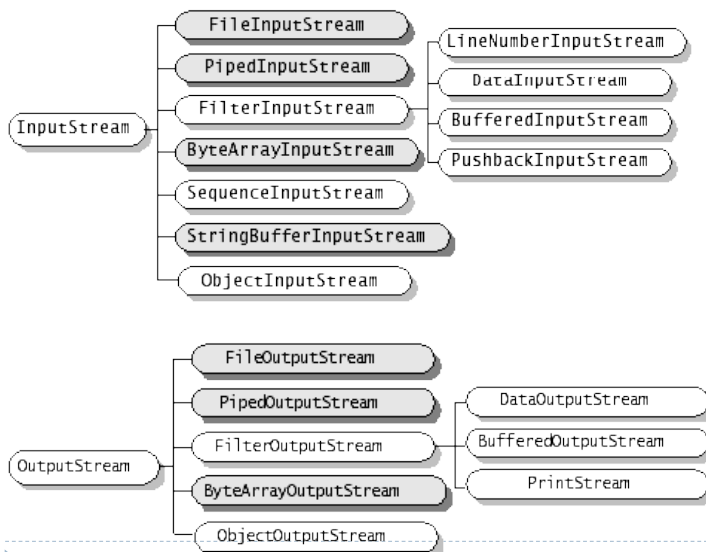
- ▶ Realizan operaciones de entrada y salida de bytes.
- ▶ Están orientados al tratamiento de datos binarios.
- ▶ Utilizan las clases abstractas ***InputStream*** y ***OutputStream*** y sus descendientes.

- ▶ **Flujos de caracteres (16 bits)**

- ▶ Realizan operaciones de entrada y salida de caracteres.
- ▶ Utilizan las clases abstractas ***Reader*** y ***Writer***.
- ▶ Se utilizan sobre todo para la internacionalización ya que permite manejar caracteres Unicode de 16 bits.



Jerarquía de flujos de bytes



Subclases de InputStream

Clase	Función
ByteArrayInputStream	Utiliza un buffer interno para los bytes leídos del stream
StringBufferInputStream	Obsoleta, se recomienda StringReader. Los bytes leídos al stream son proporcionados desde una cadena.
FileInput Stream	Obtiene los bytes de entrada desde un fichero del sistema
PipedInputStream	Permite implementar mecanismos de tubería, para por ejemplo desacoplar la entrada de la salida
FilterInputStream	Toma datos de otro stream transformándolos o añadiéndoles funcionalidad
SequenceInputStream	Permite realizar una concatenación lógica de flujos de entrada

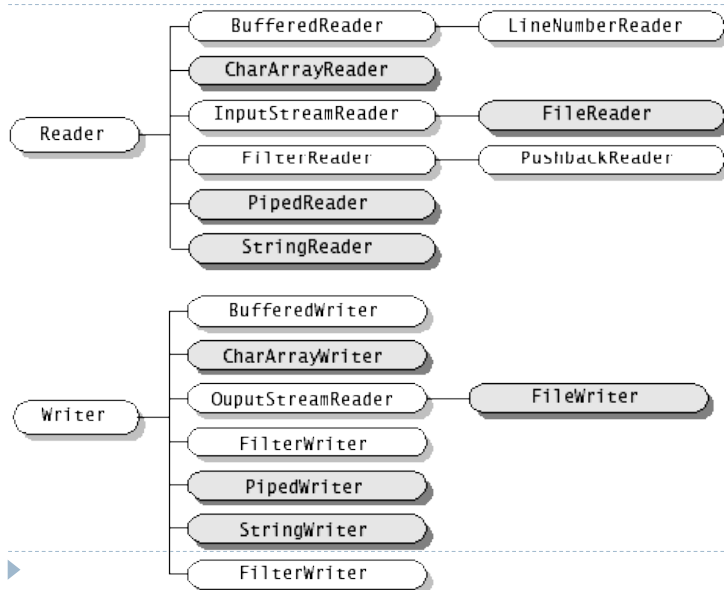


Subclases de OutputStream

Clase	Función
ByteArrayOutputStream	Implementa un flujo de salida en el que los datos se van escribiendo en una matriz de bytes que va creciendo
FileOutputStream	Es un flujo de salida para escribir datos en un fichero o en un descriptor de fichero
PipedOutpuStream	Permite implementar mecanismos de tubería, junto con PipedInputStream
FilterOutputStream	Permite hacer transformaciones en los datos de salida



Jerarquía de flujos de caracteres



Lectura/Escritura de ficheros binarios

- ▶ Se utilizan las clases *FileInputStream* y *FileOutputStream*
- ▶ Lo primero es crear un ejemplar de la clase pasando al constructor una cadena con el nombre del fichero o un File.
- ▶ Para la lectura y la escritura se utilizan los métodos *read* o *write*, que se encargan de leer o escribir un byte.
- ▶ Cuando se terminan las operaciones sobre los datos se cierra el fichero con el método *close*.



Lectura/Escritura en ficheros binarios

► Ejemplo:

```
FileInputStream in = new FileInputStream(fichero);
FileOutputStream out = new
    FileOutputStream("copia_"+fichero);
int c;
while ((c = in.read()) != -1) {
    out.write(c);
in.close();
out.close();
```



Lectura/Escritura en ficheros binarios utilizando memoria intermedia (*buffered*)

Para leer y escribir **bytes** con memoria intermedia:

```
FileInputStream fis;
```

```
fis = new FileInputStream("imagen.jpg")
```

```
BufferedInputStream bis= new BufferedInputStream(fis);
```

```
FileOutputStream fos;
```

```
fos = new FileOutputStream("copia_imagen.jpg");
```

```
BufferedOutputStream bos = new
```

```
BufferedOutputStream(fos);
```



Lectura/Escritura de ficheros de caracteres

- ▶ Se utilizan las clases **FileReader** y **FileWriter**
- ▶ Lo primero es crear un ejemplar de la clase pasando al constructor una cadena con el nombre del fichero o un File.
- ▶ Para la lectura y la escritura se utilizan los métodos **read** o **write**, que se encargan de leer o escribir un carácter.
- ▶ Los caracteres alfanuméricos se pueden tratar en muchos esquemas de codificación, por defecto utilizan UTF-8, pero se puede pasar como argumento a FileReader y FileWriter una Charset determinado (ASCII, UNICODE, ISO-8859-I, etc)
- ▶ Cuando se terminan las operaciones sobre los datos se cierra el fichero con el método **close**.



Lectura/Escritura en ficheros de caracteres

- ▶ **FileReader** además proporciona los siguientes métodos, que devuelven el número de caracteres leídos, o -1, si se ha llegado al final del fichero:
 - ▶ **int read ()**, lee un carácter.
 - ▶ **int read (char[] buf)**, lee caracteres en un array buf.
 - ▶ **int read(char[] buf, int displ, int n)**, lee como máximo n caracteres en el array buf y empezando en buf[displ].



Lectura/Escritura en ficheros de caracteres

- ▶ **FileWriter** además proporciona los siguientes métodos:
 - ▶ **void write (int car)**, escribe un carácter.
 - ▶ **void write (char[] buf)**, escribe un array de caracteres.
 - ▶ **void write (char[] buf, int displ, int n)**, escribe n caracteres en el array buf y empezando en buf[displ].
 - ▶ **void write (String st)**, escribe una cadena de caracteres.
 - ▶ **append (char car)**, añade un carácter al fichero.



Lectura/Escritura en ficheros de caracteres utilizando memoria intermedia (*buffered*)

- ▶ Las clases anteriores traducen directamente cada operación de E/S a una llamada al sistema operativo subyacente. Esto puede ser muy ineficiente cuando se procesan grandes volúmenes de información; resulta mucho más rápido leer y escribir siempre en una zona de memoria intermedia, que solo se llena o vacía (mediante llamadas al sistema operativo) cuando es necesario.
- ▶ Las clases con memoria intermedia para el tratamiento de caracteres son: `BufferedReader` y `BufferedWriter`.
- ▶ El constructor de la clase con memoria intermedia recibe un ejemplar de la clase sin memoria intermedia.
- ▶ Además nos permiten procesar líneas de texto completas, mediante los métodos `readLine()`.



Lectura/Escritura en ficheros de caracteres utilizando memoria intermedia (*buffered*)

► Ejemplos:

```
FileReader fr = new FileReader(fichent)
BufferedReader br = new BufferedReader(fr);

FileWriter fw = new FileWriter(fichsal);
BufferedWriter bw = new BufferedWriter(fw);
```



Lectura/Escritura en ficheros de caracteres haciendo lectura por líneas

- ▶ Para leer textos por líneas se emplean las clases `BufferedReader` y `PrintWriter`. El método de funcionamiento puede ser leer una línea completa, procesarla y escribirla en el archivo de destino.
- ▶ Hay que tener en cuenta que cada sistema operativo hace uso de marcadores de fin de línea diferentes.



Lectura/Escritura en ficheros de caracteres haciendo lectura por líneas

```
BufferedReader flujoDeEntrada = null;
PrintWriter flujoDeSalida = null;
try {
    flujoDeEntrada = new BufferedReader(new FileReader("libro.txt"));
    flujoDeSalida = new PrintWriter(new FileWriter("copia_libro.txt"));
    String linea;
    while ((linea = flujoDeEntrada.readLine()) != null) {
        flujoDeSalida.print(linea);
    }
} finally {
    if (flujoDeEntrada != null) {
        flujoDeEntrada.close();
    }
    if (flujoDeSalida != null) {
        flujoDeSalida.close();
    }
}
```



Lectura/Escritura de datos de tipos primitivos

- ▶ Se utilizan las clases `DataInputStream` y `DataOutputStream`. Estas clases, además de los métodos `read()` y `write()`, proporcionan métodos específicos para cada tipo de dato primitivo.
- ▶ Ejemplos:
 - ▶ `File fich= new File(ruta);`
 - ▶ `FileInputStream fis= new FileInputStream(fich);`
 - ▶ `DataInputStream dis= new DataInputStream(fis);`

 - ▶ `File fich= new File(ruta);`
 - ▶ `FileOutputStream fos= new FileOutputStream(fich);`
 - ▶ `DataOutputStream dos= new DataOutputStream(fos);`



Lectura/Escritura de datos de tipos primitivos (*buffered*)

- ▶ Se pueden utilizar con memoria intermedia. Por ejemplo:

```
archivo_salida = new DataOutputStream(  
    new BufferedOutputStream(  
        new FileOutputStream(ruta)));
```

```
archivo_entrada = new DataInputStream(  
    new BufferedInputStream(  
        new FileInputStream(ruta)));
```

donde ruta denota un ejemplar de *File* o un ejemplar de *String*.



Flujos de datos - DataOutputStream

<code>void flush()</code>	Vuelca en disco la memoria intermedia
<code>int size()</code>	Proporciona el número de bytes escritos hasta el momento.
<code>void write(byte[] b, int ppio, int num)</code>	Escribe num elementos de byte, empezando en ppio.
<code>void write(int b)</code>	Escribe el byte menos significativo del int que se le proporciona como argumento.
<code>void writeBoolean(boolean v)</code>	Escribe un valor boolean en forma de byte
<code>void writeByte(int v)</code>	Escribe el byte menos significativo del int que se le proporciona como argumento.
<code>void writeBytes(String s)</code>	Escribe la cadena en forma de bytes, descartando el byte superior de cada char
<code>void writeChar(int v)</code>	Escribe un char en forma de 2 bytes, con el MSB en primer lugar.
<code>void writeChars(String s)</code>	Escribe la cadena como una sucesión de caracteres (con 2 bytes cada uno).

► 22

Flujos de datos - DataOutputStream

<code>void writeDouble(double v)</code>	Traduce v a long empleando el método de <code>Double.doubleToLongBits</code> , y escribe ese long en forma de 8 bytes con el MSB en primer lugar.
<code>void writeFloat(float v)</code>	Traduce v a int empleando el método de <code>Float.floatToIntBits</code> y escribe ese int en forma de 4 bytes con el MSB en primer lugar.
<code>void writeInt(int v)</code>	Escribe el int v en forma de 4 bytes, con el MSB en primer lugar.
<code>void writeLong(long v)</code>	Escribe el long v en forma de 8 bytes, con el MSB en primer lugar.
<code>void writeShort(int v)</code>	Escribe el short v en forma de 2 bytes, con el MSB en primer lugar.
<code>void writeUTF(String str)</code>	Escribe los caracteres de la cadena en formato UTF-8 modificado

Flujos de datos - DataInputStream

<code>int read(byte[] b)</code>	Lee bytes hasta llenar b o acabar el fichero. Devuelve el número de bytes leídos, o -1 si el archivo está vacío.
<code>int read(byte[] b, int off, int ppio)</code>	Lee hasta len bytes, almacenándolos a partir de ppio.
<code>boolean readBoolean()</code>	Lee un byte y proporciona false si es 0 o true si no.
<code>byte readByte()</code>	Lee un byte.
<code>char readChar()</code>	Lee 2 bytes y los devuelve en forma de carácter.
<code>double readDouble()</code>	Lee 8 bytes y los devuelve en forma de double.
<code>float readFloat()</code>	Lee 4 bytes y los devuelve en forma de float.
<code>void readFully(byte[] b)</code>	Intenta llenar b de bytes. Si acaba el archivo o hay un error de E/S lanza una EOFException o IOException.
<code>void readFully(byte[] b, int ppio, int num)</code>	Intenta leer num bytes, almacenándolos a partir de ppio.

Flujos de datos - DataInputStream

<code>int readInt()</code>	Inverso de <code>writeInt()</code> , lee 4 bytes y devuelve un <code>int</code> . Supone que el MSB está a la izquierda.
<code>long readLong()</code>	Inverso de <code>writeLong()</code> , lee 8 bytes y devuelve un <code>long</code> . Supone que el MSB está a la izquierda.
<code>short readShort()</code>	Inverso de <code>writeShort()</code> , lee 2 bytes y devuelve un <code>short</code> . Supone que el MSB está a la izquierda.
<code>int readUnsignedByte()</code>	Lee un byte y lo extiende por ceros para dar un <code>int</code> , entre 0 y 255. Inverso de <code>writeByte</code> hasta 255.
<code>int readUnsignedShort()</code>	Lee dos bytes para dar un <code>int</code> , entre 0 y 65535. Inverso de <code>writeShort</code> hasta 65535.
<code>String readUTF()</code>	Lee cadenas escritas con <code>writeUTF</code> .
<code>static String readUTF(DataInput in)</code>	Lee cadenas escritas con <code>writeUTF</code> en el ejemplar de <code>DataInput</code> aportado
<code>int skipBytes(int n)</code>	Intenta saltar <code>n</code> bytes, proporciona el número de bytes que ha saltado. No lanza excepciones.

Lectura/Escritura de Objetos

Con algunas limitaciones, los objetos se pueden leer y escribir en flujos de forma similar a los tipos primitivos. Concretamente, las clases `ObjectInputStream` y `ObjectOutputStream` poseen las mismas funcionalidades que `DataInputStream` y `DataOutputStream` (y otras específicas de objetos), lo cual da lugar a que un flujo de objetos pueda contener tanto tipos primitivos como objetos. Los métodos que más interés tienen son

```
Object readObject()  
void writeObject(Object obj)
```

que leen/escriben el próximo objeto presente en el flujo.

Acceso aleatorio

- ▶ Java dispone de la clase `RandomAccessFile` que permite acceder al contenido de un fichero binario de forma aleatoria.
- ▶ Para crear objetos de esta clase, al constructor se le puede pasar o la ruta mediante un `String` o mediante un `File`. En cualquiera de los dos casos hay que pasar un segundo parámetro con el modo de acceso. Ejemplos:
 - ▶ `Fich= new RandomAccessFile(String nombre, String modoAcceso)`
 - ▶ `Fich = new RandomAccessFile(File fic, String modoAcceso)`
 - ▶ El modo de acceso puede ser “r”, “rw”



Acceso aleatorio

- ▶ Los métodos más importantes son:
 - ▶ `long getFilePointer()`
 - ▶ Devuelve la posición del puntero en el fichero.
 - ▶ `void seek(long pos)`
 - ▶ Pone el puntero en la posición dada por pos, tomada desde el inicio.
 - ▶ `long length()`
 - ▶ Devuelve el tamaño del fichero en bytes.
 - ▶ `Int skipBytes (int desplaz)`
 - ▶ Mueve el puntero desde su posición el número de bytes indicado por desplaz.



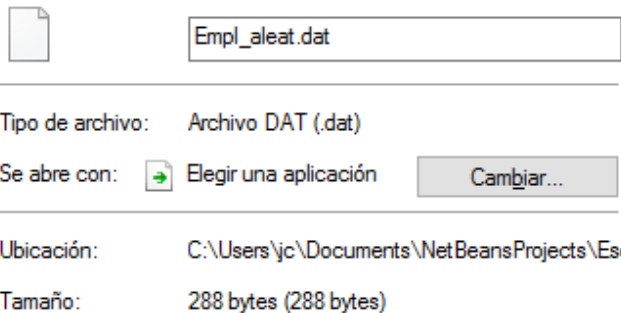
Acceso aleatorio

- ▶ Ejemplo, vamos a guardar los datos de una serie de empleados. Interesa que los datos de cada persona tengan la misma longitud para poder saltar e ir a una persona, para ello limitamos (fijamos) el tamaño de los String antes de escribirlos:
 - ▶ `buffer = new StringBuffer("GARCIA");`
 - ▶ `buffer.setLength(10);`
- ▶ El resto de campos tienen una longitud fija, porque son enteros y double.
 - ▶ `int identif;`
 - ▶ `int departamento;`
 - ▶ `float salario;`



Acceso aleatorio

- ▶ En total: int (4 bytes), string (20 bytes), int (4 bytes) y double (8 bytes) = 36 bytes.
- ▶ Tamaño del fichero = 8 registros x 36 bytes = 288 bytes



The image shows a file dialog box with the following fields and options:

- File name:** Empl_aleat.dat
- Tipo de archivo:** Archivo DAT (.dat)
- Se abre con:** A dropdown menu showing 'Elegir una aplicación' and a 'Cambiar...' button.
- Ubicación:** C:\Users\jc\Documents\NetBeansProjects\Esi
- Tamaño:** 288 bytes (288 bytes)



Acceso aleatorio

```
public class EscribirFichAleatorio {  
    public static void main(String[] args) throws IOException {  
        // TODO code application logic here  
        File fich = new File("Empl_aleat.dat");  
        RandomAccessFile file = new RandomAccessFile(fich, "rw");  
        String[] apellidos = {"MARTÍN", "SORIA", "FERNÁNDEZ", "LUNA", "GARCÍA", "PÉREZ",  
            "RODRÍGUEZ", "MARTÍNEZ"};  
        int[] dept = {10, 20, 30, 20, 10, 40, 30, 40};  
        Double[] salario = {850.65, 12035.36, 2156.36, 1500.32, 989.23, 1566.32, 1866.88, 2356.78};  
        StringBuffer buffer = null;  
        for (int i = 0; i < apellidos.length; i++) {  
            file.writeInt(i + 1); //identificador de empleado  
            buffer = new StringBuffer(apellidos[i]);  
            buffer.setLength(10);  
            file.writeChars(buffer.toString());  
            file.writeInt(dept[i]);  
            file.writeDouble(salario[i]);  
        }  
    }  
}
```

Formatos en ficheros de texto

- ▶ Se trata de dotar de una organización interna a los ficheros de texto.
- ▶ El caso extremo sería, **sin ningún formato**, en este caso sería una secuencia de caracteres de principio a fin. Sería el caso, por ejemplo, de muchos libros disponibles en la red. Su utilidad en muchos casos es el análisis de determinados aspectos lingüísticos.
- ▶ Los principales formatos serían:
 - ▶ Con longitud de campos fija.
 - ▶ Con separador entre campos.
 - ▶ Con un campo por línea.



Formatos en ficheros de texto

- ▶ Con longitud de campos fija o encolumnados.
 - ▶ Cada campo se almacena con una longitud fija para todos los elementos.
 - ▶ Es necesario conocer estos valores de longitudes para poder leer y escribir.
 - ▶ Supongamos que queremos guardar datos de empleados (dni (9 caracteres), edad (2 dígitos), nombre (20 caracteres) y salario mensual (5 dígitos y 2 decimales). Podríamos escribir con un formato `%9s%2d%20s%7.2f\n`
 - ▶ Ejemplo:

▶ I4567432F32	Ana Martín Pérez	1654.43
▶ 4322224M28	Manuel Rodríguez Mar	998.32
▶ 657897H45	Jesús López Castro	10234.23



Formatos en ficheros de texto

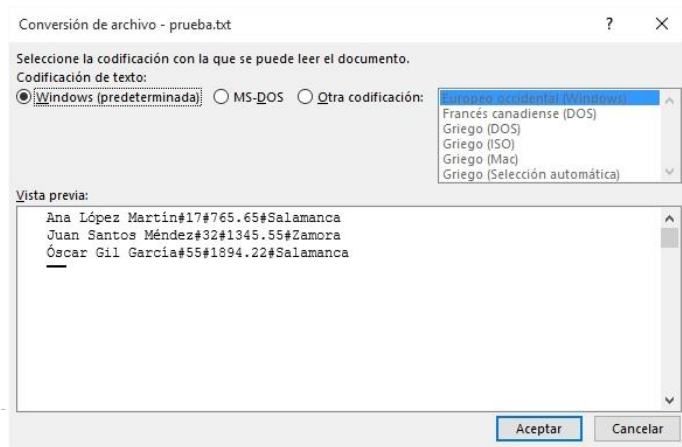
- ▶ Con separador entre campos.

- ▶ Cada campo está separado del siguiente por un separador, que puede ser cualquier carácter, pero que debe ser elegido de forma que no forme parte del contenido de los campos. Caracteres habituales son el tabulador, la coma (,), el punto y coma (;), los dos puntos (:), los dobles dos puntos (::), la almohadilla (#), asterisco (*), etc.
- ▶ El separador (,) es muy habitual dando nombre a un tipo de ficheros llamado .csv (del inglés *comma-separated values*).
- ▶ La mayoría de las hojas de cálculo, los procesadores de textos y los sistemas gestores de bases de datos permiten importar y exportar a este formato.
- ▶ Ejemplo:
 - ▶ I4567432F#32#Ana Martín Pérez#I654.43
 - ▶ 4322224M#28#Manuel Rodríguez Mar#998.32
 - ▶ 657897H#45#Jesús López Castro#I0234.23



Formatos en ficheros de texto

- ▶ Con separador entre campos.
- ▶ Por ejemplo, en Word, nos preguntaría por el tipo de codificación del fichero:



Formatos en ficheros de texto

- ▶ Con separador entre campos.
 - ▶ En Excel, nos permitiría separarlo en columnas, indicándole primero la codificación y posteriormente la separación:

Asistente para importar texto - paso 1 de 3

El asistente estima que sus datos son Delimitados.
Si esto es correcto, elija Siguiente, o bien elija el tipo de datos que mejor los describa.

Tipo de los datos originales

Elija el tipo de archivo que describa los datos con mayor precisión:

☒ Delimitados - Caracteres como comas o tabulaciones separan campos.
☐ De ancho fijo - Los campos están alineados en columnas con espacios entre uno y otro.

Comenzar a importar en la fila: 1 Origen del archivo: Windows (ANSI)

☐ Mis datos tienen encabezados.

Vista previa del archivo C:\Users\jc\Desktop\borro\prueba.txt.

1	Ana López Martín#17#765.65#Salamanca
2	Juan Santos Méndez#32#1345.55#Zamora
3	Oscar Gil García#55#1894.22#Salamanca
4	
5	

Cancelar < Atrás Siguiente > Finalizar

Formatos en ficheros de texto

► Con separador entre campos.

► Excel:

Asistente para importar texto - paso 2 de 3

Esta pantalla le permite establecer los separadores contenidos en los datos. Se puede ver cómo cambia el texto en la vista previa.

Separadores

☐ Tabulación

☐ Punto y coma

☐ Coma

☐ Espacio

☒ Otro: #

☐ Considerar separadores consecutivos como uno solo

Calificador de texto: *

Vista previa de los datos

Ana López Martín	17	765.65	Salamanca
Juan Santos Méndez	32	1345.55	Zamora
Oscar Gil García	55	1894.22	Salamanca

Cancelar

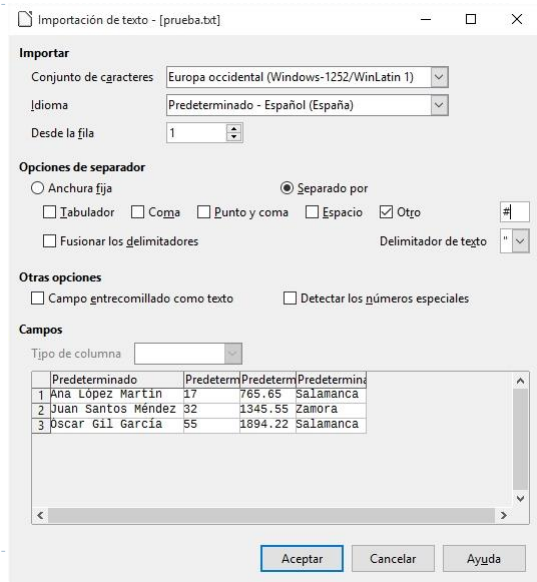
< Atrás

Siguiente >

Finalizar

Formatos en ficheros de texto

- ▶ Con separador entre campos.
- ▶ En otras hojas de cálculo de forma muy similar, como en Calc de Libre Office:



Formatos en ficheros de texto

- ▶ Con un campo por línea.
 - ▶ Aparece cada campo en una línea.
 - ▶ Ejemplo:
 - ▶ I4567432F
 - ▶ 32
 - ▶ Ana Martín Pérez
 - ▶ I654.43
 - ▶ 4322224M
 - ▶ 28
 - ▶ Manuel Rodríguez Mar
 - ▶ 998.32
 - ▶ 657897H
 - ▶ 45
 - ▶ Jesús López Castro
 - ▶ I0234.23



