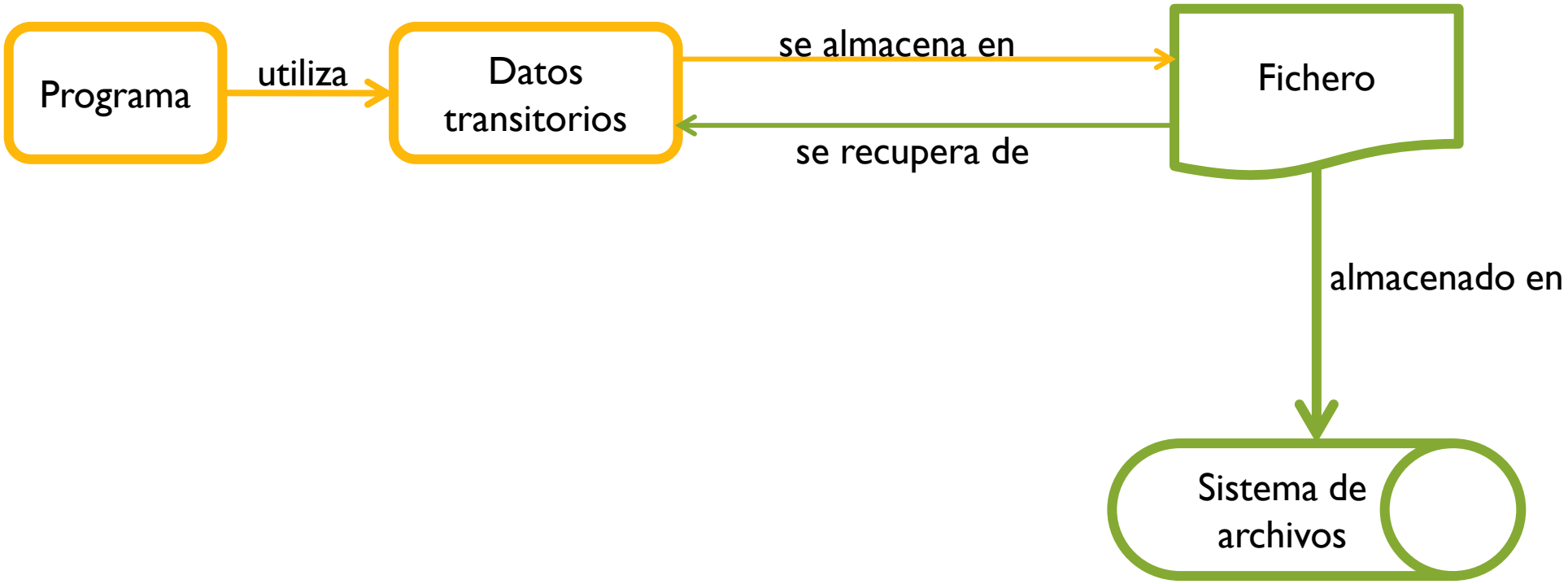
Unidad 1. Manejo de ficheros

Agustín Crespo

Trabajando con Ficheros/Files

Ficheros como elemento de persistencia de datos



Ficheros

- Un fichero es un conjunto de bits que deben ser interpretados por el programa que estamos haciendo
- En Java, todos los ficheros se van a considerar como un flujo (**Stream**) de bits

- Según modo de acceso, pueden ser:
  - Secuenciales
  - Aleatorio

- Según el contenido, pueden ser:
  - Binarios
  - De Texto (hay que tener en cuenta la codificación, se sugiere UTF-8)

- Aunque las bases de datos han reemplazado en gran parte el uso de ficheros, aún se siguen usando mucho en las aplicaciones
- Se utilizan sobre todo cuando:
  - no se prevé un uso multiusuario de los datos,
  - la cantidad de datos no es muy elevada, (y no se prevé que en el futuro vaya a crecer mucho), o
  - se van a manejar estructuras muy sencillas en el programa, con pocas modificaciones de datos

Java y gestión básica de ficheros

- En java, la gestión de ficheros está agrupada en el paquete **java.io**
- Para el sistema operativo, un directorio y un archivo con datos son exactamente iguales, sólo hay una marca que los diferencia.
- En Java utilizaremos la clase **File** para representar tanto un archivo simple como un directorio.
  - https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/File.html

- Para crear un File:
  - En Windows: `new File("C:\\ruta\\hacia\\el\\archivo.txt")`
  - En Linux: `new File("/ruta/hacia/el/archivo.txt")`
  - Si no se pone la unidad (o raíz), se considera como punto de partida el propio directorio de la aplicación

File

Método	Función
`boolean exists()`	Devuelve **true** si el fichero/directorio existe
`boolean isFile()`	Devuelve **true** si el objeto **File** corresponde a un fichero normal
`boolean isDirectory()`	Devuelve **true** si el objeto **File** corresponde a un directorio
`String[] list()`	Devuelve un *array* de **String** con los nombres de ficheros y directorios asociados al objeto **File**
`File[] listFiles()`	Devuelve un *array* de objetos **File** conteniendo los ficheros que estén dentro del directorio representado por el objeto **File**
`boolean canRead()`	Devuelve **true** si el fichero se puede leer
`boolean canWrite()`	Devuelve **true** si el fichero se puede escribir
`long length()`	Devuelve el tamaño del fichero en bytes
`String getName()`	Devuelve el nombre del fichero o directorio
`String getPath()`	Devuelve el camino relativo
`String getAbsolutePath()`	Devuelve el camino absoluto del fichero/directorio

File

Método	Función
`boolean exists()`	Devuelve **true** si el fichero/directorio existe
`boolean isFile()`	Devuelve **true** si el objeto **File** corresponde a un fichero normal
`boolean isDirectory()`	Devuelve **true** si el objeto **File** corresponde a un directorio
`String[] list()`	Devuelve un *array* de **String** con los nombres de ficheros y directorios asociados al objeto **File**
`File[] listFiles()`	Devuelve un *array* de objetos **File** conteniendo los ficheros que estén dentro del directorio representado por el objeto **File**
`boolean canRead()`	Devuelve **true** si el fichero se puede leer
`boolean canWrite()`	Devuelve **true** si el fichero se puede escribir
`long length()`	Devuelve el tamaño del fichero en bytes
`String getName()`	Devuelve el nombre del fichero o directorio
`String getPath()`	Devuelve el camino relativo
`String getAbsolutePath()`	Devuelve el camino absoluto del fichero/directorio

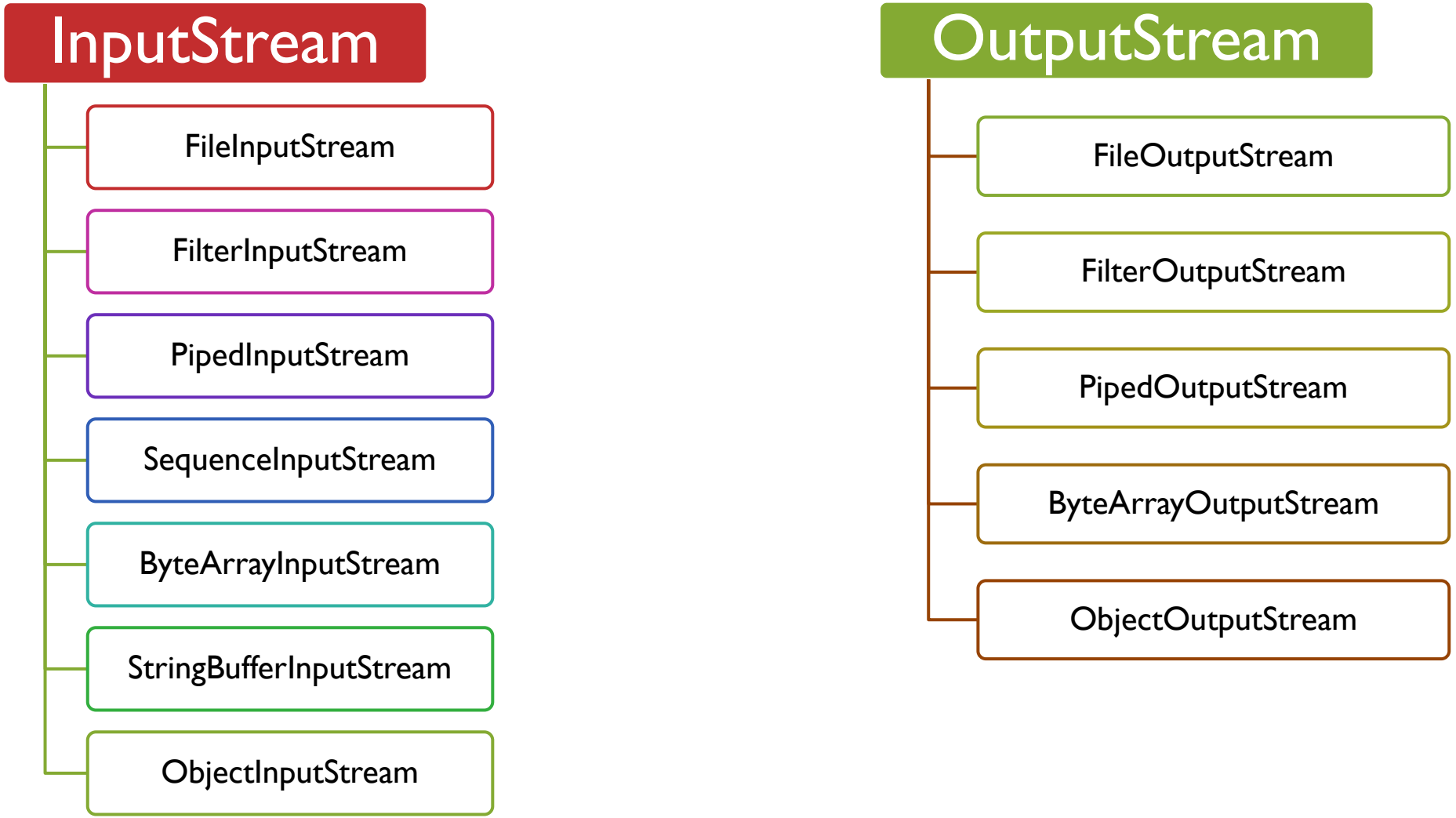
- Ejemplos de File

Trabajando con Flujos/Streams

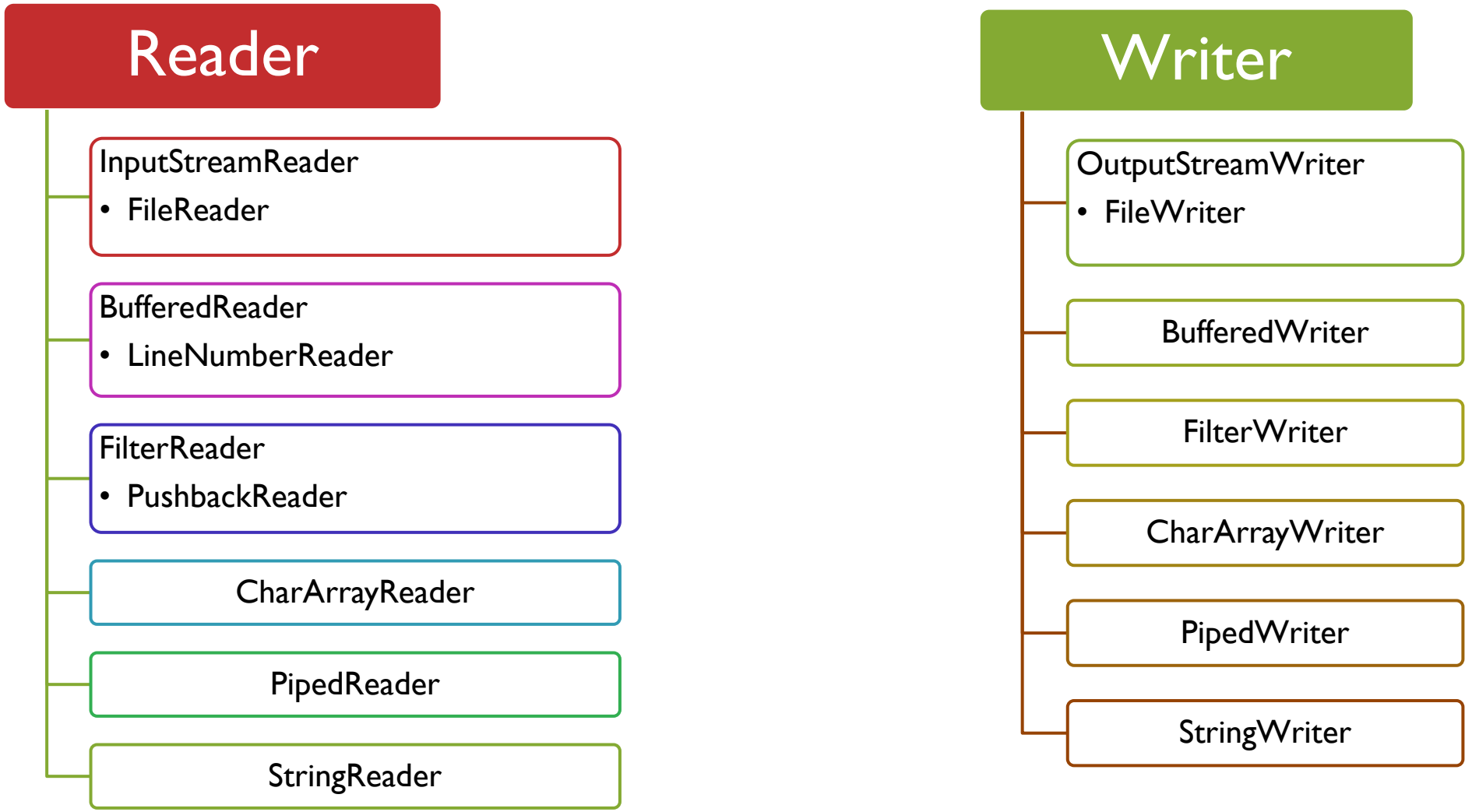
Streams en Java

- En el paquete **java.io** tenemos una característica que simplifica la programación, y es que todos los archivos se consideran un flujo de bytes.
  - Da igual que sea un archivo que tenemos en local o un archivo que nos está llegando por red, todo se considera exactamente igual.
- La forma (y la clase) en la que se va a trabajar dicho flujo de bytes nos va a indicar con qué clases deberemos trabajar.
- Diferenciamos dos clases principales:
  - Flujos de bytes, de 8 bits. Se utilizan para entrada/salida de datos binarios.
    - Clases base: `InputStream` y `OutputStream`
  - Flujo de caracteres, de 16 bits. Se utilizan para operaciones de entrada/salida de caracteres de texto, para favorecer la internacionalización con Unicode
    - Clases base: `Reader` y `Writer`

Flujo de Bytes



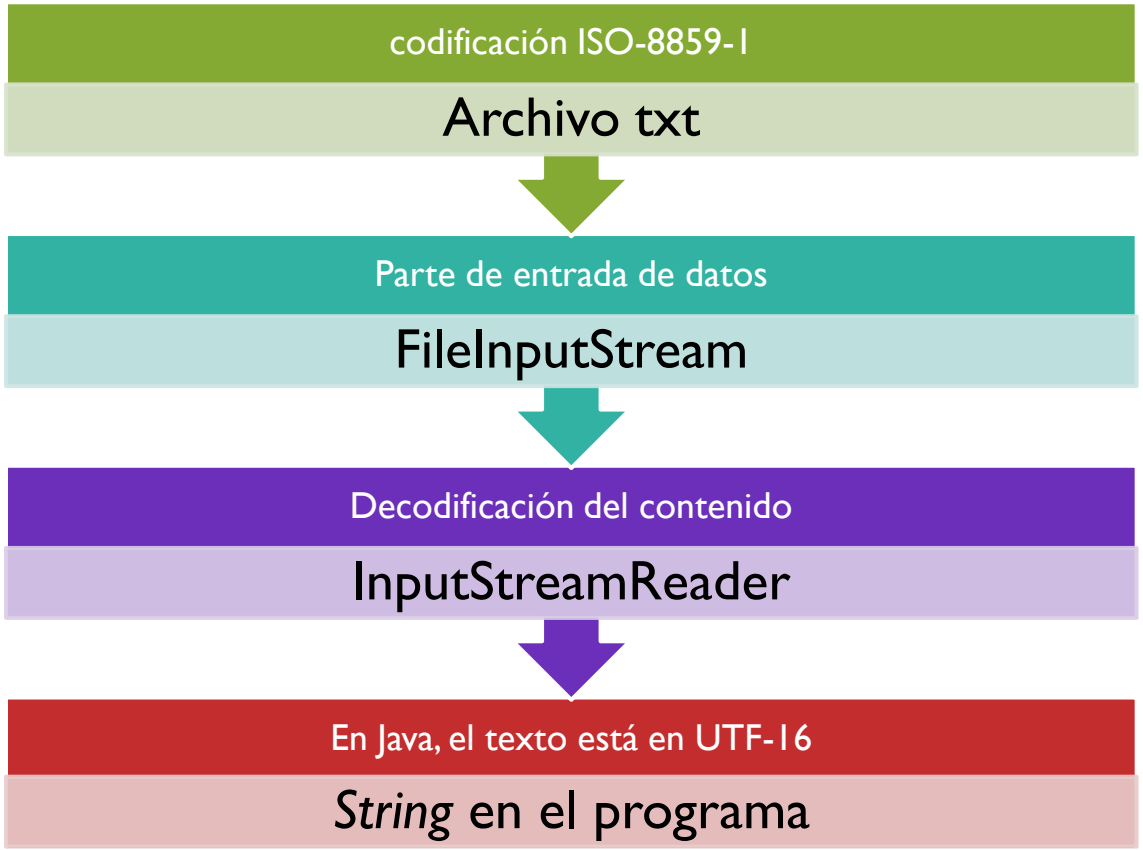
Flujo de texto



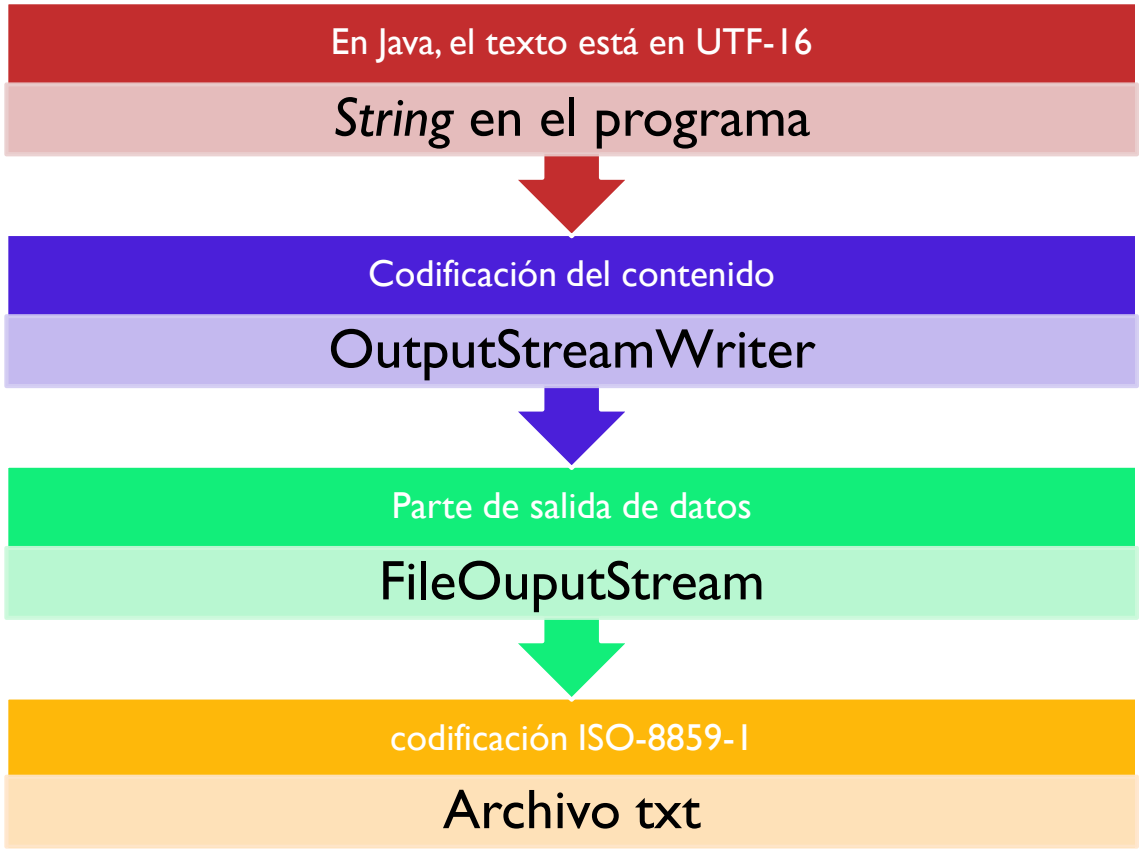
- En ambas jerarquías nos encontramos multitud de clases, cada una especializada y orientada hacia una cosa concreta:
  - Funcionalidad básica de lectura/escritura, dependiendo del tipo de datos
  - Posibilidad de recodificación de texto.
  - Buffering, que permite aumentar la velocidad de lectura/escritura
- Todas estas clases se pueden componer (encapsular) una con otra, para agregar funcionalidades. Dependerá de qué se desea hacer.

Ejemplo

Lectura desde un fichero de texto



Escritura a un fichero de texto



- Ejemplos de ficheros en modo texto
  - Ejemplos normales
  - Ejemplos con buffer de array de caracteres
  - Ejemplos con buffer de línea, usando una clase adecuada

Tarea 1

- Crear un programa que vaya leyendo desde teclado todo lo que escribamos, y lo vaya guardando en un archivo de texto, con extensión TXT.
- Dejará de leer y acabará cuando introduzcamos tres ENTER seguidos. En ese momento, cerrará el archivo de forma adecuada para salvar todo lo que se haya escrito.

Tarea 2

- Crear un programa que recodifique un archivo que se le pasa en el primer parámetro, que estará en la codificación del segundo parámetro, y lo escriba en el archivo (la ruta) que se pasa en el tercer parámetro, con la codificación que hay en el cuarto parámetro.
  - Si el número de parámetros no es 4, dará error directamente.
  - Si el segundo o el cuarto parámetro no son codificaciones de caracteres oportunas, dará error.
  - Si el archivo de origen no existe, dará error.
  - Si el archivo de destino ya existe, dará error.