élogos

ACCESO A FICHEROS

élogos

ÍNDICE

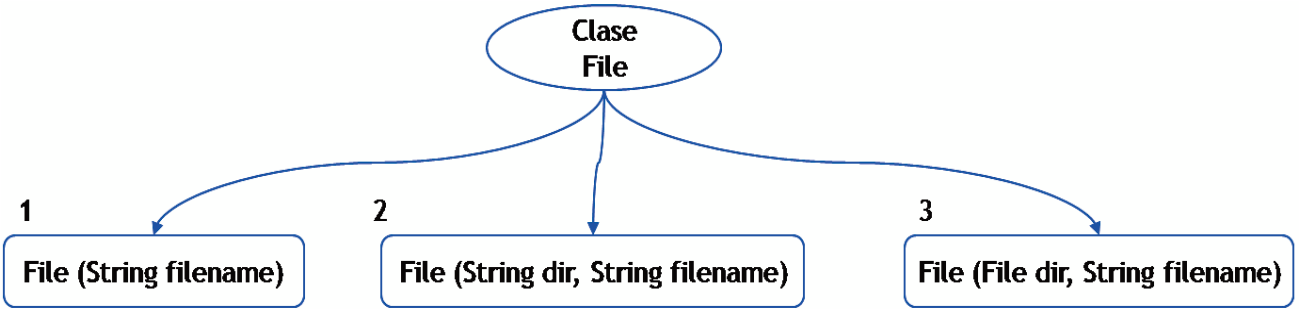
ACCESO A FICHEROS

1. Manejo de ficheros

File

Todas las clases para el manejo de ficheros se encuentran en el paquete java.io

Los constructores de File son los siguientes:



Tiene las siguientes constantes a nivel de clase y contienen respectivamente los símbolos que se utilizan para separar los paths dependiendo de la plataforma en que se está.

```
static final String <File>.pathSeparator    > (;)
static final String <File>.separator        > (\)
        File f2 = new File ("c:\\windows\\win.exe"); // path absoluto
        File f3 = new File (".\\datos\\bd.dat"); // path relativo
```

Métodos File

Con los métodos de la clase File se obtiene información relativa al archivo o ruta con que se ha inicializado el objeto. Así antes de crear un flujo para leer de un archivo es conveniente determinar si el archivo existe, en caso contrario no se puede crear el flujo.

Los métodos más importantes que describen esta clase son los siguientes:

String <File>.getPath()
String <File>.getAbsolutePath()
String <File>.GetCanonicalPath() throws IOException
boolean <File>.isFile()

boolean <File>.isDirectory()
boolean <File>.exists()
boolean <File>.canRead()
boolean <File>.canWrite()
long <File>.length()
boolean <File>.mkDir()
boolean <File>.renameTo (String name)
boolean <File>.delete()

FileInputStream

La clase FileInputStream sirve para acceder al contenido de los ficheros. Se crea a partir del nombre del fichero que se va a tratar. Realiza la lectura del fichero.
Los constructores son los siguientes:

1: FileInputStream (String Filename) throws FileNotFoundException
2: FileInputStream (File Filename) throws FileNotFoundException

Los métodos más importantes que describen esta clase son los siguientes:

- int <InputStream>.read() throws IOException
- Protected <FileInputStream>.finalize() throws IOException

FileOutputStream

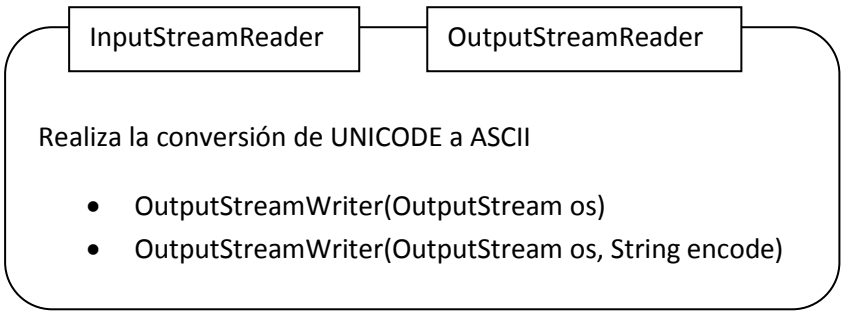
La clase FileOutputStream también sirve para acceder al contenido de los ficheros. Realiza la escritura de los datos del fichero.
Los constructores son los siguientes:

1: FileOutputStream (String Filename) throws IOException.
2: FileOutputStream(String Filename, boolean append) throws IOException.

Array de caracteres y objetos String

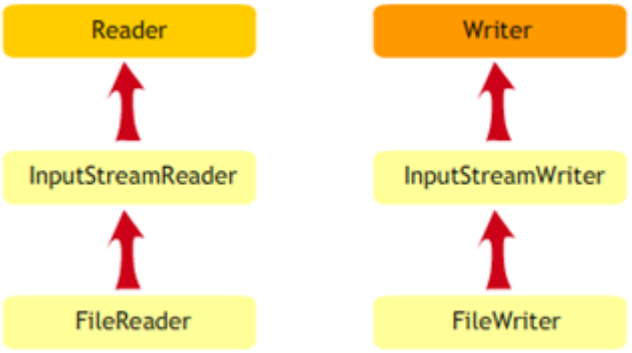
InputStreamReader y OutputStreamWriter

En la conversión de código UNICODE a ASCII y viceversa, las clases InputStreamReader y OutputStreamWriter actúan como filtro para la traducción.



FileReader y FileWriter

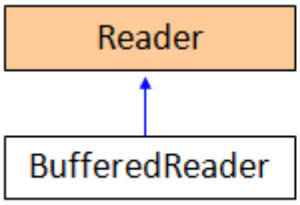
Son derivadas de las anteriores que solo valen para ficheros.



- FileReader (String filename)
- FileReader (File f)
- FileWriter (String filename)
- FileWriter (File f)
- FileWriter (String FileName, boolean append)

BufferedReader y BufferedWriter

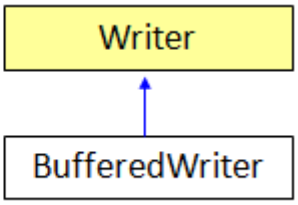
Nos permiten leer y escribir en cualquier objeto Reader y Writer, respectivamente.
En el caso de ficheros, estos objetos Reader y Writer podrían ser los FileReader y FileWriter.



Constructor
BufferedReader (Reader reader)

Métodos

readLine ()
close ()



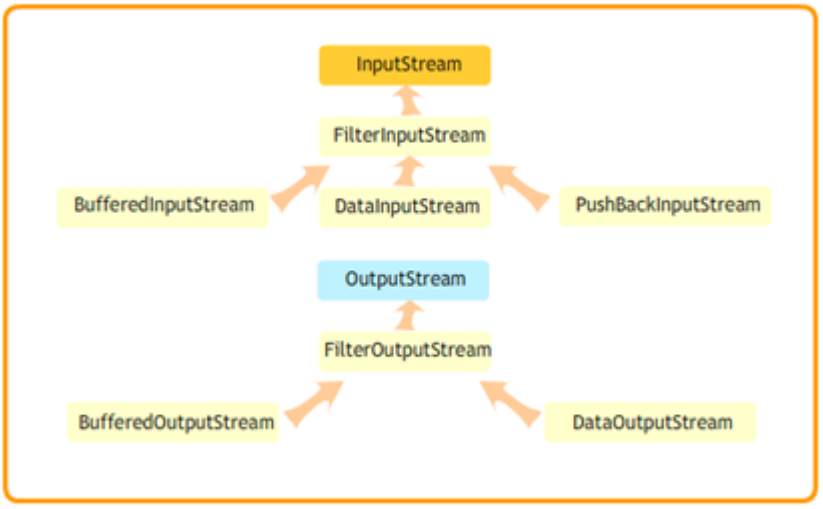
Constructor
BufferedWriter (Writer writer)

Métodos

readLine ()
close ()

2. Filtro de Streams

Nos permiten modificar los datos que están dentro del Stream. Para construir filtros se tienen las siguientes clases:



BufferedInputStream y BufferedOutputStream

Sirven para poner un buffer (memoria RAM) tanto para los Streams de entrada como para los de salida, antes de pasarlo al dispositivo final.

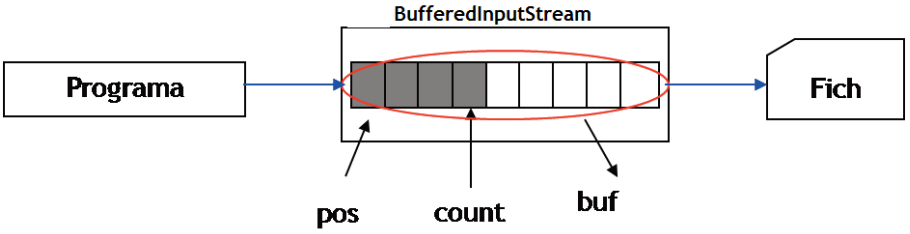
- En el constructor de estas clases se pasa el Stream.
- Al Stream se le añade el buffer. La clase BufferedOutputStream incluye los siguientes atributos protegidos:

  - protected byte[]<BufferedOutputStream> buf;
  - protected int <BufferedOutputStream> count;

Utilización de buffers

Para forzar que se vacie de información el buffer se tiene el método flush() de la clase base.
La clase BufferedInputStream incluye los siguientes atributos protegidos.

- protected[] byte<BufferedInputStream> buf; Array de datos del buffer.
- protected int<BufferedInputStream> count; Indica hasta donde esta cargado el buffer.
- protected int<BufferedInputStream> pos; Indica la siguiente posición a liberar del buffer.



Utilizar un buffer implica una optimización en el tiempo de acceso al fichero.

FileInputStream f1= new FileInputStream("datos.txt").
BufferedInputStream bf1=new BufferedInputStream(f1);
//Primeramente se crea el Stream y luego el buffer asociado.

DataInputStream y DataOutputStream

Son filtros que permiten solucionar el problema de escribir y lee datos que no sean del tipo fundamental byte.

3. Serialización de objetos

Serialización de objetos

Se pueden enviar y recoger datos de un Stream convirtiendo el objeto en una serie de bytes. Posteriormente se podrá introducir en el Stream.
Se tienen dos clases de persistencia:

- Persistencia temporal
- Persistencia espacial

En ambos casos se necesita serializar el objeto.

NotSerialización de objetos

No se permite por defecto serializar los objetos ya que cualquiera puede serializarlos. Para que un objeto se pueda serializar tiene que implementar el **interface Serializable.**

Si el objeto no es serializable y se intenta serializar en ejecución se lanza la siguiente excepción:

NotSerializableException

En este momento ya se pueden crear objetos Punto y serializarlos.

Si una clase contenedora está Serializable y tiene objetos contenidos, estos deben implementar también Serializable (Composición). Si no se haces así se lanza la excepción.

NotSerializableException

Razones para serializar un atributo

Cuando se serializa un objeto, sus atributos se serializan pero no sus métodos. Los atributos estáticos (static) no se serializan ya que pertenecen a la clase y no al objeto. Los atributos con el modificador **transient** no se serializan.

Las razones para serializar el atributo son dos básicamente:

1. Por seguridad
2. Para mantener la consistencia por ejemplo: FileInputStream no es igual en dos máquinas ya que el sistema de ficheros en cada una de ellas es diferente.

Gestión de versiones

En Java por defecto si la clase de serialización y la clase de deserialización no coinciden no se puede recoger el objeto. La excepción que se lanza cuando se produce este error es la siguiente:

InvalidCastException

Ejemplo

```
class Prueba implements Serializable {
int numero = 0;
. . .
. . . se modifica la clase . . .
class Prueba implements Serializable {
int numero = 0;
void NoHaceNada () {};
```

Para comprobar las versiones de las clases Java, Java dispone del número SUID(Serial Unique ID) que es diferente para cada una de las clases que se crean.

C:\>serialver<Clase> Devuelve el número SUID de una clase

Para mantener la compatibilidad entre las distintas versiones de una clase, habrá que declarar la siguiente constante dentro de la clase:

static final long serialVersionUID=6290341731103L; //serialver

El valor de serialVersionUID se corresponderá con la primera versión de la clase. Normalmente el número SUID se pone desde la primera clase generada.