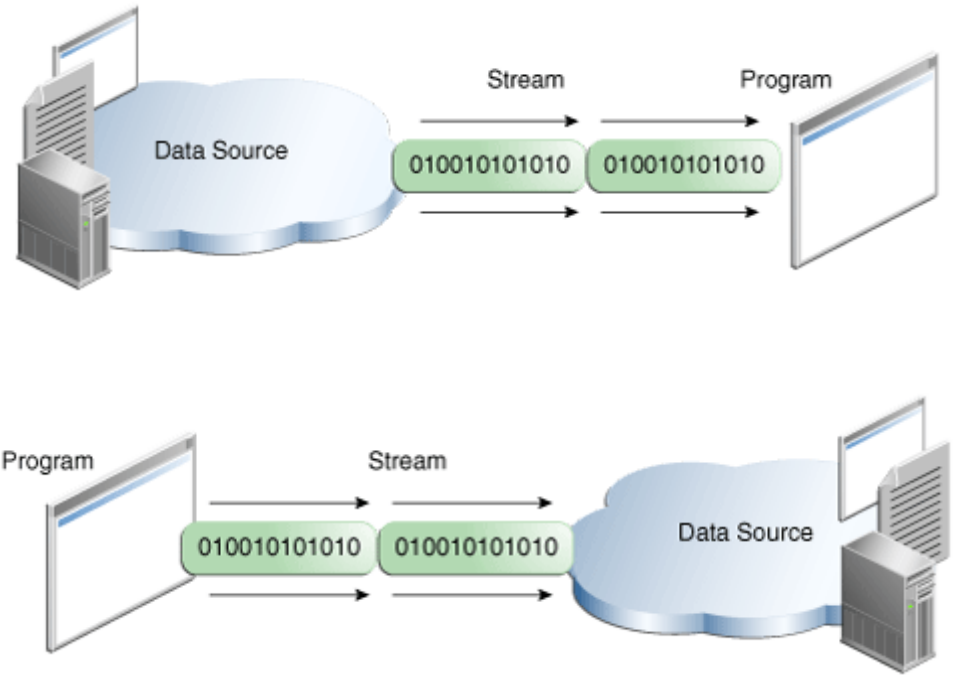
UT9: Lectura y escritura de información. Manejo de ficheros

Clases para gestión de flujos de datos desde/hacia ficheros

¿Qué es un stream? Es una secuencia de bytes, uno tras otro entre un origen y un destino.



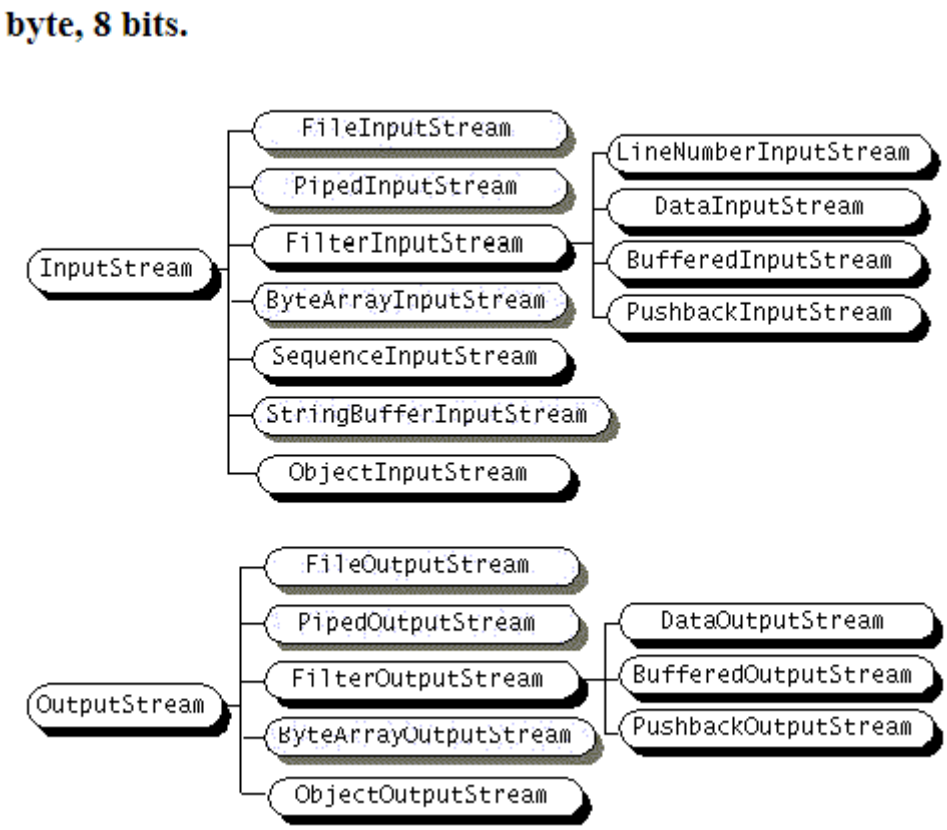
Todos los datos fluyen a través del ordenador desde una entrada hacia una salida. Este flujo de datos se denomina también stream. Hay un flujo de entrada (input stream) que manda los datos desde el exterior (normalmente el teclado) del ordenador, y un flujo de salida (output stream) que dirige los datos hacia los dispositivos de salida (la pantalla o un archivo).

No se almacenan en ninguna parte. Lo que se haga con los bytes es responsabilidad del programador.

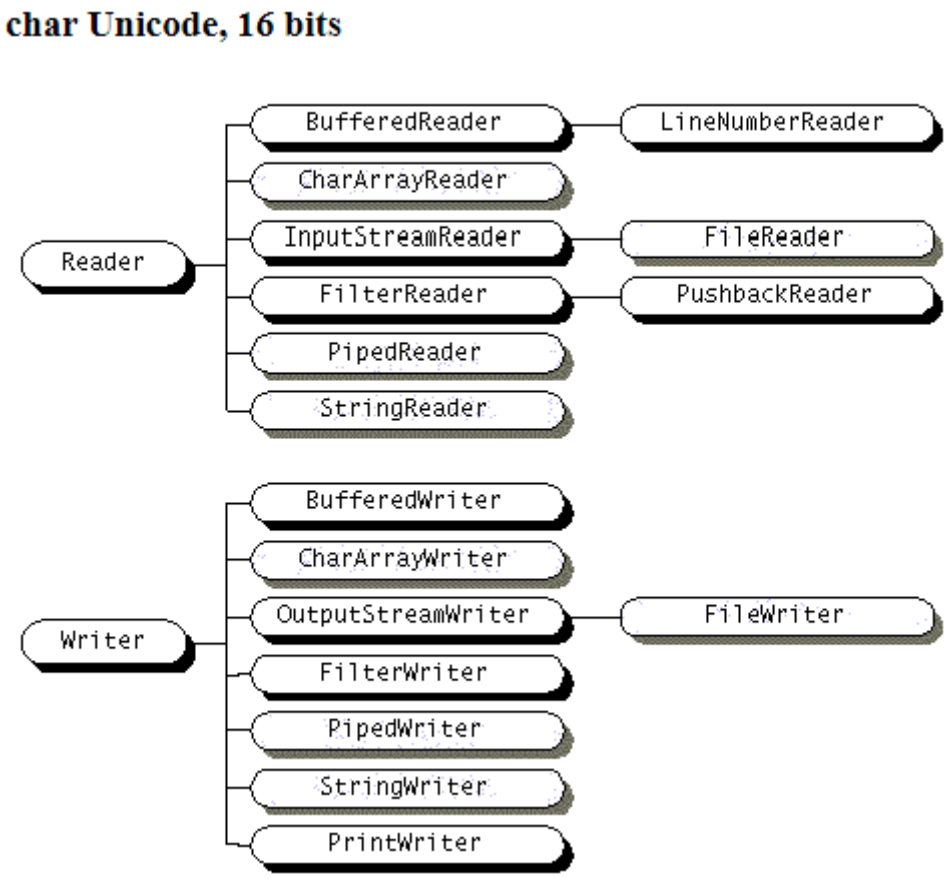
No te puedes mover hacia adelante o hacia atrás en el stream, necesitas almacenarlo en un buffer primero.

Existen dos tipos de flujos según el tipo de dato:

Flujos binarios **byte (8-bit)**: Es el tipo de flujo más primitivo y portable, de hecho, cualquier otro tipo de flujo está construido sobre este porque hablando a bajo nivel todas las operaciones de I/O son flujos de bytes. Nos permitirá trabajar adecuadamente con datos binarios tales como archivos de imagen, sonido, etc. Las clases principales para manejar estos flujos son las clases abstractas InputStream y OutputStream de las cuales heredan otras subclases que implementan en formas más concretas la misma tarea.



Flujos de caracteres **char (16-bit)**: Es un tipo de flujo de caracteres en codificación Unicode, listo para la internacionalización, ideal para trabajar con texto plano. Las clases principales para manejar estos flujos son las clases abstractas Reader y Writer de las cuales heredan otras sub-clases que implementan en formas más concretas la misma tarea. Cualquiera de estas clases realiza la conversión correspondiente de byte a char para leer o de char a byte para escribir.



Presentación sobre flujos de bytes y cadenas: https://es.slideshare.net/tacksuo/flujos-de-bytes-y-cadenas

El proceso para leer o escribir datos consta de tres pasos:

1. Abrir el flujo de datos
2. Mientras exista más información (leer o escribir) los datos
3. Cerrar el flujo de datos

Flujos basados en caracteres

Para trabajar con ficheros de texto usaremos las clases *FileReader* y *FileWriter* para leer o escribir caracteres en el fichero respectivamente. *FileReader* puede generar una *FileNotFoundException* (porque el fichero no exista o no sea válido) y *FileWriter* puede lanzar una *IOException* (porque el disco está lleno o no se pueda escribir). Habrán de ser tratadas dentro de un bloque *try-catch*.

FileReader nos provee de estos métodos para la lectura que devuelven el número de caracteres leídos o un -1 si se ha alcanzado el final de fichero.

Método	Descripción
int read()	Lee un carácter y lo devuelve
int read(char[] buf)	Lee hasta *buf.length* caracteres del fichero y los deja en un array *buf*
int read(char[] buf, int desplazamiento, int n)	Lee hasta *n* caracteres de datos y los deja en un array *buf* comenzando por *buf[desplazamiento]* y devuelve el número leído de caracteres.

Ejemplo de uso:

Este programa le cada uno de los caracteres de un fichero de texto y los muestra por pantalla. El método *read()* puede lanzar una excepción *IOException* y como no se ha capturado se advierte de ello en el *main()*.

```
import java.io.*;


public class LeerFichTexto {
  public static void main(String[] args) throws IOException {
    File fichero = new File("Fichero1.txt");  //declarar fichero
    FileReader fic = new FileReader(fichero); //crear el flujo de
entrada
    int i;
    while ((i = fic.read()) != -1) //se va leyendo un carácter
      System.out.println( (char) i + "==>"+ i);
    fic.close(); //cerrar fichero
   }
}
```

El método *read()* devuelve un entero que es el valor del carácter leído y se hace una conversión (cast) a tipo char. Cuando *read()* devuelva -1 se habrá alcanzado el final del fichero.

Para ir leyendo de 20 en 20 caracteres se hubiera tenido que escribir estas líneas:

```
char b[]= new char[20];
```

```
while ((i = fic.read(b)) != -1) System.out.println(b);
```

Para escribir ficheros de caracteres usamos la clase *FileWriter*. En esta tabla podemos ver los métodos que facilita para la escritura:

Método	Descripción
void write(int c)	Escribe un carácter
void write (char[] buf)	Escribe un array de caracteres
void write (char[] buf, int desplazamiento, int n)	Escribe *n* caracteres de datos en el array *buf* comenzando por *buf[desplazamiento]*
void write (String str)	Escribe una cadena de caracteres
void append (char c)	Añade un carácter a un fichero

Atención: estos métodos pueden lanzar una *IOException*.

El siguiente ejemplo escribe caracteres, uno a uno, en un fichero, los caracteres se obtienen de un String que se convierte en un array de caracteres.

```
 import java.io.*;


public class EscribirFichTexto {

  public static void main(String[] args) throws IOException {

   File fichero = new File("FichTexto.txt");//declarar fichero

   FileWriter fic = new FileWriter(fichero); //crear el flujo de
salida

   String cadena ="Esto es una prueba con FileWriter";

   char[] cad = cadena.toCharArray();//convierte un String en array de
caracteres


   for(int i=0; i<cad.length; i++)

         fic.write(cad[i]);  //se va escribiendo un carácter


   fic.append('*'); //añado al final un *

   fic.close();    //cerrar fichero


  }

}
```

Si se prefiere escribir los caracteres usando todo el array también se puede hacer usando *fic.write(cad)*. Se puede ver en el siguiente ejemplo que escribe cadenas de caracteres obtenidas de un array String; las cadenas se irán insertando una a continuación de la otra sin saltos de línea.

```
import java.io.*;


public class EscribirFichTexto2 {

  public static void main(String[] args) throws IOException {

   File fichero = new File("FichTexto.txt");//declara fichero

   FileWriter fic = new FileWriter(fichero); //crear el flujo de
salida

   String cadena ="Esto es una prueba con FileWriter";

   char[] cad = cadena.toCharArray();//convierte un String en array de
caracteres


   for(int i=0; i<cad.length; i++)

         fic.write(cad[i]);  //se va escribiendo un carácter


   fic.append('*'); //añado al final un *

   fic.write(cad);//escribir un array de caracteres de golpe

   String c="\n*esto es lo último*";

   fic.write(c);//escribir un String


   String prov[] = {"Albacete","Ávila","Badajoz",

               "Cáceres","Huelva","Jaén",

                     "Madrid","Segovia","Soria","Toledo",

                     "Valladolid","Zamora"};


   fic.write("\n");

   for(int i=0; i<prov.length; i++) {

            fic.write(prov[i]); // escribe cada elemento en una línea
nueva

            fic.write("\n");
```

```
   }



   fic.close();    //cerrar fichero

  }
}
```

Advertencia: si el fichero ya existe cuando se vayan a escribir caracteres sobre él, todo lo que hubiera almacenado anteriormente se borrará. Para añadir información al final del fichero sin borrar lo anterior habrá que usar el constructor de *FileWriter* de esta forma:

FileWriter fic = new FileWriter(fichero, true);

FileReader no contiene métodos para leer líneas completas pero *BufferedReader* sí, con *readLine()* lo hace y la devuelve o un null si no hay nada que leer o es el final del fichero.

En este ejemplo se leen líneas de un fichero y se muestran por pantalla.

```
import java.io.*;
public class LeerFichTextoBuf {
  public static void main(String[] args) {
  try{
        File fic = new File("FichTexto.txt");//declara fichero
              BufferedReader fichero = new BufferedReader( new
FileReader(fic));
               String linea;
               while((linea = fichero.readLine())!=null)
               System.out.println(linea);
               fichero.close();
      }
      catch (FileNotFoundException fn ){
               System.out.println("No se encuentra el fichero");}
```

```
      catch (IOException io) {
               System.out.println("Error de E/S ");}


  }
}
```

La clase *BufferedWriter* también deriva de *Writer*, añade un buffer para conseguir una escritura más eficiente de caracteres. En el siguiente ejemplo escribe 10 filas de caracteres en un fichero de texto y después de cada fila salta una línea con el método *newLine()*.

```
2  import java.io.*;
3  public class EscribirFichTextoBuf {
4    public static void main(String[] args) {
5    try{
6        BufferedWriter fichero = new BufferedWriter
7                (new FileWriter("FichTexto1.txt"));
8        for (int i=1; i<11; i++){
9          fichero.write("Fila numero: "+i); //escribe una línea
10         fichero.newLine(); //escribe un salto de línea
11       }
12       fichero.close();
13       }
14     catch (FileNotFoundException fn ){
15               System.out.println("No se encuentra el fichero");}
16     catch (IOException io) {
17               System.out.println("Error de E/S ");}
18   }
19 }
```

Otra clase a nuestra disposición es PrintWriter, deriva de Writer, y aporta los métodos print(String) y println(String), que conocemos de System.out, para escribir en un fichero.