PRÁCTICA 1

1.1 Objectivo

El objetivo de esta práctica es la descripción de los flujos de datos, tanto de entrada como de salida, y utilizar diferentes clases que facilitarán la lectura/escritura de diferentes tipos de datos.

1.2. Flujos de datos (streams)

Los flujos de datos son una herramienta que ofrece Java para trabajar con datos externos, tanto de teclado o pantalla, memoria, ficheros o recursos de red. Podemos imaginarlos como un tubo donde podemos escribir bytes o desde donde podemos leer bytes. No importa lo que pueda haber en el otro extremo del tubo: puede ser un teclado, un monitor, un archivo, un proceso un objeto Java o una conexión TCP/IP.

Todos los flujos que aparecen en Java (englobados generalmente en el paquete java.io) pertenecen a dos clases abstractas comunes: *java.io.InputStream* para los flujos de entrada (aquellos de los que podemos leer) y *java.io.OutputStream* para los flujos de salida (aquellos en los que podemos escribir).

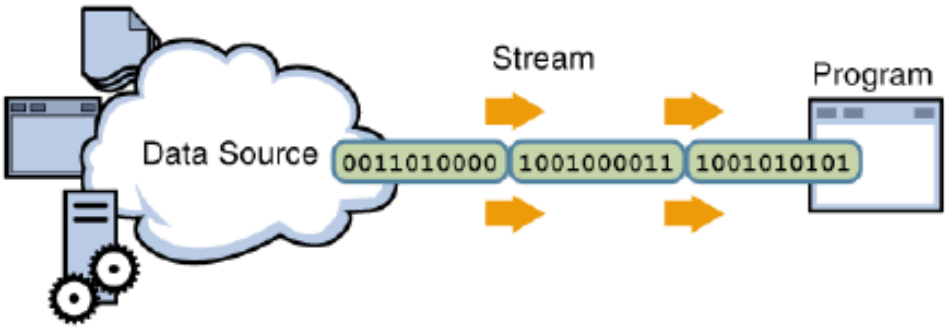

Figura 1. Ilustración de un flujo de lectura. Se abre un flujo desde la fuente.

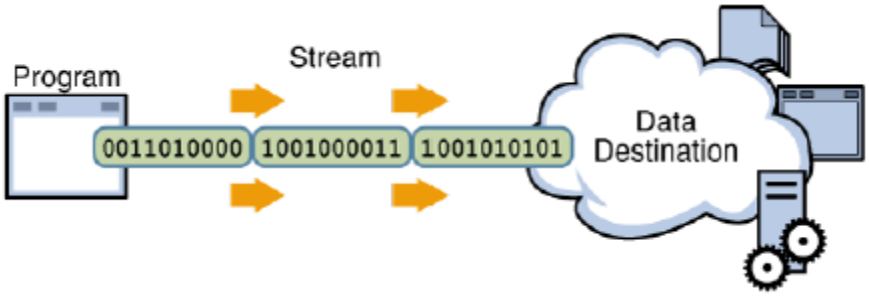

Figura 1. Ilustración de un flujo de escritura. Se abre un flujo hacia el destino.

Ambas clases permiten trabajar sobre flujos de bytes: leen o escriben bytes a través del *stream*.

Tal como se ha comentado, los flujos pueden tener orígenes diversos (un archivo, un socket TCP, etc) pero una vez tenemos una referencia a ellos podemos trabajar siempre de la misma forma: leyendo datos mediante los métodos de la familia read() o escribiendo datos con los métodos de la familia write().

1.3. Flujos de bytes: entrada (lectura).

Se crean objetos de la clase *java.io.InputStream*. Del conjunto de métodos disponibles (ver la clase *InputStream* en http://docs.oracle.com/javase/7/docs/api/) nos fijaremos en los de lectura. Se puede comprobar que es un método sobrecargado (métodos con el mismo nombre pero con diferente funcionalidad; se ejecutará uno u otro en función de los parámetros de la llamada).Se dispone de los siguientes:

- *int read();*
  - Lee un byte
- *int read (byte [] buf);*
  - Lee un conjunto de bytes y los almacena en un array.
  - Retorna el número de bytes leídos.
- *int read (byte [] buf, int offset, int longitud);*
  - Lee un máximo de *longitud* bytes y los almacena en el array a partir de la posición indicada por *offset* (el primer byte se guarda en *buf[offset]*).

Devuelven un -1 en caso de llegar el final del flujo de datos.

Una vez finalizado el uso del flujo de bytes hay que cerrarlo:

- *void close();*
  - Cierra el flujo y libera los recursos asociados al mismo.

Si no se cierra el *InputStream* explícitamente, el flujo asociado se cierra cuando se destruye el objeto.

Ejercicio 1

Iniciaremos con flujos con la entrada estándar. El caso de la entrada estándar es especial dado que no hay que instanciar ningún objeto de la clase *InputStream*. Dentro de la clase *System*, el campo *in* (*System.in*) ya es un objeto de tipo *InputStream* (ver http://docs.oracle.com/javase/7/docs/api/) asociado a la entrada estándar (por defecto el teclado). El flujo ya está abierto y preparado para proporcionar datos de entrada. No será necesario en este caso, por tanto, instanciar un objeto de la clase *InputStream* aunque en general, para otras fuentes (memoria, conexiones de red, etc.), sí que será necesario hacerlo:

```
InputStream flujoEnt = …;
```

A través de ese objeto podemos leer bytes. El siguiente ejemplo permite leer bytes del teclado:

```
// Lectura de un byte
int octeto = System.in.read();
```

Ejercicio 1.1

Cread un proyecto llamado *EntSalEstandar* y utilizad el método *read()* tal como se ha expuesto anteriormente para leer de la entrada estándar (por defecto el teclado). Entrad en modo depuración para poder visualizar el valor de la variable *octeto* (o el nombre que le hayáis puesto).

Ejercicio 1.2

A continuación modificad el código para poder leer de la entrada estándar un conjunto de bytes y almacenarlos en un array de bytes de tamaño 10:

```
byte [] buffer = new byte[10];.
```

Contestad a las siguientes preguntas:
¿Qué pasa si el número de caracteres introducidos por teclado es menor que el rango del array creado (<10)?
¿Qué pasa si el número de caracteres introducidos por teclado es mayor que el rango del array creado (>10)?
Modificad el código para averiguar cuántos bytes se han leído a través del método *read()*;

Ejercicio 1.3

Finalmente modificad el código para que almacene los bytes leídos de la entrada estándar a partir de la 5ª posición del array.

Notad que el byte (8 bits) leído mediante el método *read()* se devuelve como un dato de tipo *int* (32 bits) con un valor entre 0 y 255.

A partir de la versión Java 1.5 está disponible la clase *Scanner* que dispone de diversos métodos para facilitar la lectura desde la entrada estándar.

1.4. Flujos de bytes: salida (escritura).

Se crean objetos de la clase *java.io.OutputStream*. Del conjunto de métodos disponibles (ver la clase *OuputStream* en http://docs.oracle.com/javase/7/docs/api/) nos fijaremos en los de escritura. Al igual que en el caso de la lectura se puede comprobar que es un método sobrecargado (métodos con el mismo nombre pero con diferente funcionalidad; se ejecutará uno u otro en función de los parámetros de la llamada).Se dispone de los siguientes:

- *void write(int c);*
  - Escribe un byte
- *void write(byte [] buf);*
  - Escribe un array de bytes.
- *void write(byte [] buf, int offset, int longitud);*
  - Escribe una porción de un array de bytes: escribe en el flujo de datos *longitud* bytes del array especificado a partir de la posición *offset*.

Una vez finalizado el uso del flujo de bytes hay que cerrarlo:

- *void close();*
    - Cierra el flujo y libera los recursos asociados al mismo.

Si no se cierra el *OutputStream* explícitamente, el flujo asociado se cierra cuando se destruye el objeto.

Ejercicio 2

Trabajaremos ahora con la salida estándar, por defecto la consola. Iniciaremos con flujos de bytes con la salida estándar.

Al igual que con la entrada estándar, el objeto *System.out* no hay que instanciarlo puesto que ya está creado. Por defecto el flujo ya está abierto y preparado para proporcionar datos. No será necesario en este caso, por tanto, instanciar un objeto de la clase *OuputStream*, aunque en general, para otros destinos (ficheros, conexiones TCP/IP, etc.), sí que será necesario hacerlo:

```
OutputStream flujoSal = …;
```

Sólo como nota, *System.out* es de tipo *PrintStream*, clase que hereda de *OutputStream*. Podemos, por tanto, utilizar los métodos de la clase *OutputStream*. La clase *PrintStream* dispone de un conjunto de métodos más potentes para trabajar con flujos de salida pero, de momento, no se trabajará con ellos.

Ejercicio 2.1

Modifica el código creado en el ejercicio 1.1 de forma que aparezca por la salida estándar el valor del byte leído (la variable *octeto*, o el nombre que le hayáis puesto).

¿Aparece por pantalla?

Intenta encontrar mecanismos para que el objeto destino asociado al flujo (pantalla en este caso) reciba el byte enviado.

Ejercicio 2.2

Modificad el código del ejercicio 1.2 de forma que aparezca por la salida estándar el array leído en ese ejercicio.

¿Aparece por pantalla?

Si es el caso, justificad el motivo.

Ejercicio 2.3

Modificad el código del ejercicio 1.3 de forma que aparezcan por la salida estándar los tres elementos del contenido del array leído en el ejercicio 1.2 a partir de la 5ª posición.

Intentad leer 7 elementos del array a partir de la 5ª posición. ¿Qué ocurre?.

Como veremos más adelante, las excepciones se deberán capturar para poder hacer un tratamiento correcto de los errores en tiempo de ejecución.