Ficheros y "streams"

Desde el punto de vista de Java,
cada fichero no es más que una secuencia o flujo de bytes [*stream*].

Los *streams* pueden ser
- de entrada (`InputStream`)
- de salida (`OutputStream`).

Los ficheros pueden almacenar los datos
- como secuencias de caracteres (ficheros de texto)
- como secuencias de bytes (ficheros binarios)

> *Ejemplo*: El número 5 se puede almacenar como el carácter '5' (código ASCII 53, representado en UNICODE mediante la secuencia de bits `0000 0000 0011 0101`)
> o bien como una secuencia de 32 bits en binario (`0000…0101`)

Un programa en Java accede a un fichero creando un objeto asociado a un stream del tipo adecuado (de entrada o de salida, de caracteres para ficheros de texto o de bytes para ficheros binarios).

Los tipos más comunes de streams están definidos en el paquete **`java.io`** de la biblioteca de clases estándar de Java.

Ejemplos: En nuestros programas ya hemos utilizado distintos streams para leer datos desde el teclado y mostrar datos por pantalla:
- `System.out` es un `PrintStream`
- `System.in` es un `InputStream`

La clase File

Acceso a información acerca de ficheros y directorios:

```
import java.io.File;
import java.io.IOException;
import java.util.Date;

public class FileDemo
{
  public void mostrarInfoFichero( String path )
         throws IOException
  {
    File fichero = new File( path );
    if ( fichero.exists() ) {
       System.out.println("Nombre: "+fichero.getName());
       System.out.println("- Ruta completa: "
             + fichero.getAbsolutePath() );
       System.out.println("- Tamaño: "
             + fichero.length() + " bytes");
       System.out.println("- Última modificación: "
             + new Date(fichero.lastModified()) );
       if (fichero.isFile()) {
          System.out.println("- Fichero normal");
       } else if (fichero.isDirectory())  {
          System.out.println("- Directorio");
          mostrarContenidoDirectorio(fichero);
       }
    } else {
       throw new IOException
             ("El fichero '"+path+"' no existe");
    }
  }

  public void mostrarContenidoDirectorio
              (File directorio)
  {
    String ficheros[] = directorio.list();
    for (int i=0; i<ficheros.length; i++)
        System.out.println("\t"+ficheros[i]);
  }
```

```
  public static void main (String args[])
         throws IOException
  {
    FileDemo demo = new FileDemo();

    if (args.length>0) {
       demo.mostrarInfoFichero(args[0]);
    } else {
       System.out.println ("USO:");
       System.out.println ("java FileDemo <fichero>");
    }
  }
}
```

Fichero convencional

```
java FileDemo FileDemo.java
```

```
Nombre: FileDemo.java
- Ruta completa: F:\[fp]\ejemplos\FileDemo.java
- Tamaño: 1496 bytes
- Última modificación: Thu Apr 28 16:31:23 CEST 2005
- Fichero normal
```

Directorio

```
java FileDemo f:\[fp]\ejemplos
```

```
Nombre: ejemplos
- Ruta completa: f:\[fp]\ejemplos
- Tamaño: 0 bytes
- Última modificación: Thu Apr 28 16:31:05 CEST 2005
- Directorio
        FileDemo.class
        FileDemo.java
```

Fichero inexistente

```
java FileDemo xxx.java
```

```
Exception in thread "main" java.io.IOException:
El fichero 'xxx.java' no existe
        at FileDemo.mostrarInfoFichero(FileDemo.java:40)
        at FileDemo.main(FileDemo.java:59)
```

Streams de entrada

Las clases derivadas de **`InputStream`** representan flujos de datos de entrada que pueden provenir de distintas fuentes:

Clase	Fuente de datos	Parámetros del constructor
`ByteArray InputStream`	Array de bytes	Buffer del que leer los bytes
`StringBuffer InputStream`	String	Cadena de caracteres
`File InputStream`	Fichero	Cadena con el nombre del fichero u objeto de tipo `File`
`Piped InputStream`	Pipeline*	`Piped OutputStream`
`Sequence InputStream`	Otros `InputStreams`	Secuencia de varios `InputStreams`

* Un "pipeline" se utiliza para enviar los datos de salida de un programa directamente a la entrada de otro programa (sin pasar por un fichero en disco)

Los tipos anteriores de `InputStreams` se utilizan como fuentes de datos y se conectan luego a un `FilterInputStream` que nos permite leer con mayor "comodidad" los datos de entrada.

`FilterInputStream` es una clase abstracta de la que derivan las siguientes clases:

Clase	Función
`Data InputStream`	Lectura de datos primitivos en binario (int, char…). Véase `DataOutputStream`
`Buffered InputStream`	Usa un buffer para evitar el acceso a disco cada vez que se lee un dato
`LineNumber InputStream`	Mantiene el número de línea actual, al que se accede con `getLineNumber()`
`Pushback InputStream`	Permite "devolver" el último byte leído

Streams de salida

Las clases derivadas de **`OutputStream`** permiten decidir a dónde mandar los datos de salida:

Clase	Destino de los datos	Parámetros del constructor
`ByteArrayOutputStream`	Array de bytes	Tamaño inicial del buffer
`FileOutputStream`	Fichero	Cadena con el nombre del fichero u objeto de tipo File
`PipedInputStream`	Pipeline*	`PipedInputStream`

* Un "pipeline" se utiliza para enviar los datos de salida de un programa directamente a la entrada de otro programa (sin pasar por un fichero en disco)

Los tipos anteriores de `OutputStreams`
se conectan a un `FilterOutputStream`
que facilita generar la salida en el formato adecuado.

`FilterOutputStream` es una clase abstracta
de la que derivan las siguientes clases:

Clase	Función
`DataOutputStream`	Salida de datos primitivos en binario. Véase `DataInputStream`
`PrintStream`	Salida con formato (p.ej. `System.out`)
`BufferedOutputStream`	Salida a través de un buffer. El método `flush()` permite vaciarlo.

Subclases de `Reader`

Clase	Fuente	Uso
`BufferedReader`	Reader	Añade un buffer del que leer los caracteres
`CharArrayReader`	`char[]`	Lectura de caracteres a partir de un array de caracteres
`FileReader`	Fichero	Lectura de caracteres de un fichero de texto
`InputStreamReader`	`InputStream`	Convierte un `InputStream` en un `Reader`
`LineNumberReader`	`Reader`	Mantiene el número de línea
`PipedReader`	Pipeline*	`PipedWriter`
`StringReader`	`String`	Lectura de caracteres de una cadena

Subclases de `Writer`

Clase	Destino	Uso
`BufferedWriter`	`Reader`	Añade un buffer en el que escribir caracteres
`CharArrayWriter`	`char[]`	Escritura de caracteres en un array de caracteres
`FileWriter`	Fichero	Escritura de caracteres en un fichero de texto
`OutputStreamWriter`	`OutputStream`	Conecta un `Writer` con un `OutputStream`
`PipedWriter`	Pipeline*	`PipedReader`
`PrintWriter`	`OutputStream`	Salida con formato en un `OutputStream`
`StringWriter`	`String`	Escritura de caracteres en una cadena

Configuraciones típicas

Leer datos desde el teclado con `System.in`

```
import java.io.*;

public class TestSystemIn
{
  public static void main (String[] args)
         throws IOException
  {
    InputStreamReader reader;
    BufferedReader    input;

    // Secuencia de bytes -> Secuencia de caracteres

    reader = new InputStreamReader(System.in);

    // Secuencia de caracteres -> Secuencia de líneas

    input = new BufferedReader(reader);

    // Lectura de una línea de texto

    System.out.println("Escriba algo...");
    String cadena = input.readLine();
    System.out.println("Ha escrito: "+cadena);
  }
}
```

Configuraciones típicas

Mostrar el contenido de un fichero

Solución 1: Byte a byte

```
import java.io.*;

public class TestReadByte
{
  public static void main(String[] args)
         throws IOException
  {
    FileInputStream     file;
    BufferedInputStream buffered;
    DataInputStream     in;

    // Apertura del fichero (modo de lectura)

    file = new FileInputStream(args[0]);

    // … con buffer

    buffered = new BufferedInputStream(file);

    // … para leer datos en binario

    in = new DataInputStream(buffered);


    // Recorrido secuencial del fichero

    while (in.available() != 0)
          System.out.print((char)in.readByte());


    // Cierre del fichero

    in.close();
  }
}
```

Mostrar el contenido de un fichero

Solución 2: Línea a línea

```
import java.io.*;

public class TestReadLine
{
  public static void main(String[] args)
         throws IOException
  {
    FileReader     file;
    BufferedReader in;
    String         cadena;

    // Apertura del fichero de texto (modo de lectura)

    file = new FileReader(args[0]);

    // … con buffer para leer línea a línea

    in = new BufferedReader(file);


    // Lectura del fichero línea a línea

    cadena = in.readLine();

    while (cadena!=null) {
          System.out.println(cadena);
          cadena = in.readLine();
    }

    // Cierre del fichero

    in.close();
  }
}
```

Caso práctico: Agenda

Vamos a trabajar con los datos de una agenda de contactos:

```
import java.util.Date;

public class Contacto
{
  private String nombre;
  private String teléfono;
  private String email;
  private String dirección;
  private Date   nacimiento;
  private int    grupo;
  private double deuda;

  // Constantes simbólicas

  public static final int TRABAJO = 1;
  public static final int FAMILIA = 2;
  public static final int AMIGOS  = 3;

  // Constructor

  public Contacto (String nombre)
  {
    this.nombre = nombre;
  }

  // Métodos set & get
  ...

  // Salida estándar

  public String toString ()
  {
    return nombre + "(" + email + ")\n"
         + " Teléfono: "+ telefono + "\n"
         + " Cumpleaños: "+ nacimiento + "\n"
         + " Deuda: "+ deuda + "€";
  }
}
```