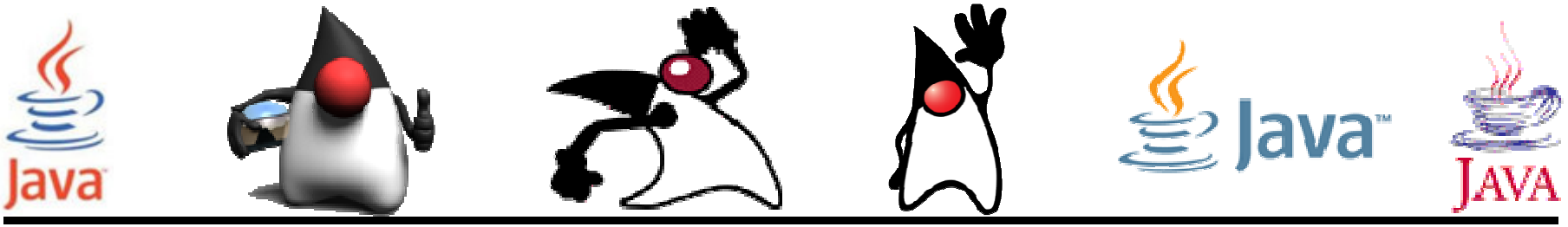


Entrada/Salida basada en Streams

Pedro Corcuera

Dpto. Matemática Aplicada y Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Estudiar las clases de Entrada/Salida basadas en Streams
- Comprender las clases y métodos disponibles para lectura/escritura de ficheros binarios y texto

Índice

- Qué es una entrada/salida stream
- Tipos de Streams
- Jerarquía de clases Stream
- Flujo de control de una operación E/S usando Streams
- Byte – Character – Buffered streams
- Standard streams
- Data streams
- Object streams
- File class

Streams de Entrada/Salida (I/O)

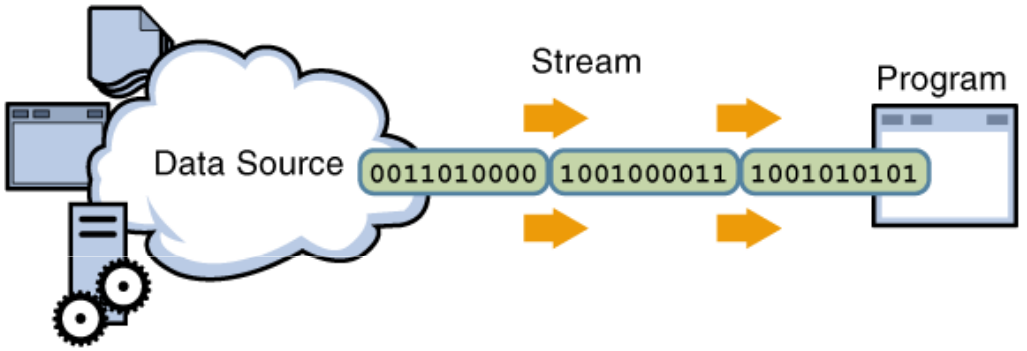
- Un Stream I/O representa una fuente de entrada o un destino de salida
- Un stream puede representar varios tipos diferentes de fuentes y destinos:
  - ficheros en disco, dispositivos, otros programas, un socket de red y arrays de memoria
- Los streams soportan varios tipos de datos
  - bytes simples, tipos de datos primitivos, caracteres localizados, y objetos
- Algunos streams son de paso y otros de conversión

Streams de Entrada/Salida (I/O)

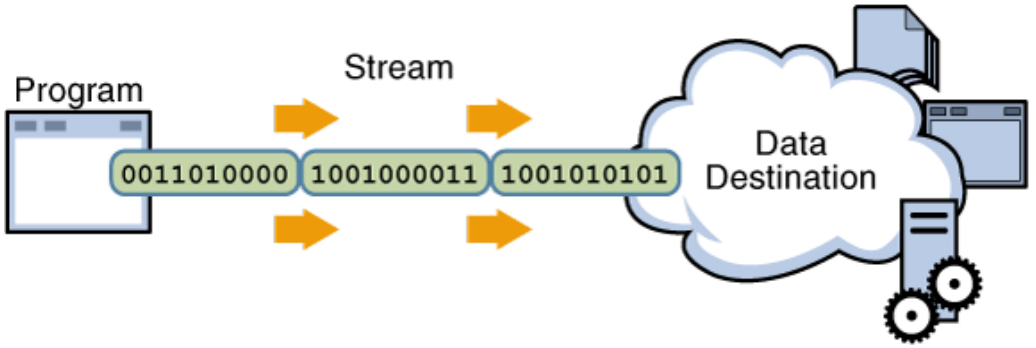
- Sin importar cómo trabajan internamente, todos los streams presentan el mismo modelo simple a los programas que los usan
  - Un stream es una secuencia de bytes
- La entrada/salida basada en streams soprta la lectura o escritura de datos secuencialmente
- Un stream se puede abrir para leer o escribir, pero no la leer y escribir

Stream de Entrada (Input Stream)

- Un programa usa un input stream para leer datos de una fuente, uno a la vez (secuencialmente)



- Un programa usa un output stream para escribir datos a un destino, uno a la vez (secuencialmente)



Tipos de Streams generales

- Byte y Character Streams
  - Character vs. Byte
- Input y Output Streams
  - Basados en fuente o destino
- Node y Filter Streams
  - Si los datos en un stream son o no manipulados o transformados

Streams Byte y Character

- Byte streams
  - Para datos binarios
  - Clases raíz de byte streams (ambas son abstractas):
    - Clase InputStream
    - Clase OutputStream
- Character streams
  - Para caracteres Unicode
  - Clases raíz de character streams (ambas abstractas):
    - Clase Reader
    - Clase Writer

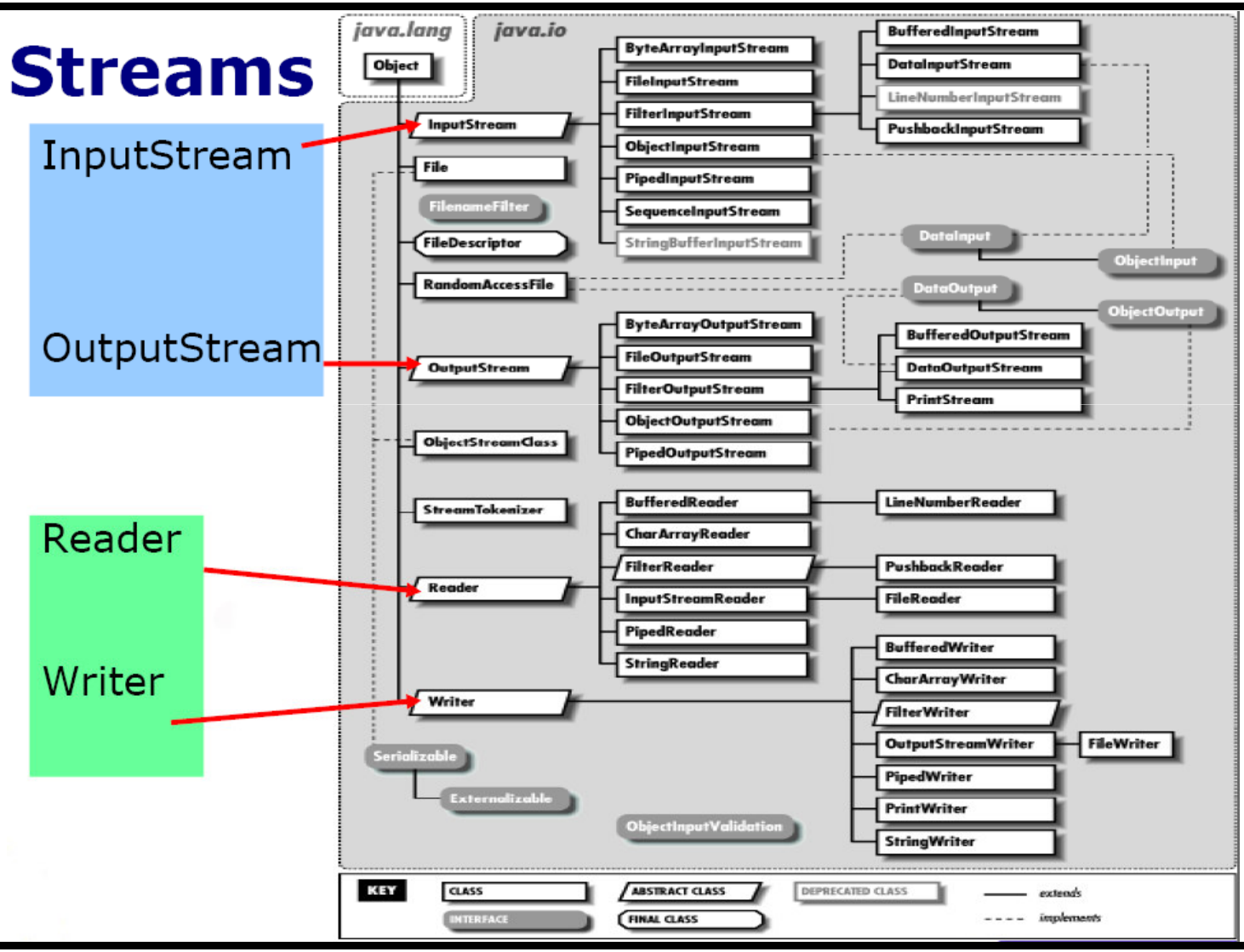
Streams Input y Output

- Input o source streams
  - Pueden leer de estos streams
  - Clases raíz de todos los input streams:
    - Clase InputStream
    - Clase Reader
- Output o sink (destino) streams
  - Pueden escribir en estos streams
  - Clases raíz de todos los output streams:
    - Clase OutputStream
    - Clase Writer

Streams Nodo y Filtro

- Node streams (sumidero de datos)
  - Contienen la funcionalidad básica de lectura o escritura de una ubicación específica
  - Los tipos de nodo streams incluyen ficheros, memoria y pipes
- Filter streams (stream de procesado)
  - Capa sobre los streams nodo entre hilos de ejecución o procesos
  - Para una funcionalidad adicional – alterando o gestionando datos en el stream

Jerarquía de la clase Stream



Clases Abstractas

- InputStream y OutputStream
- Reader y Writer

Clase Abstracta InputStream

Method Summary	
int	available() Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
void	close() Closes this input stream and releases any system resources associated with the stream.
void	mark(int readlimit) Marks the current position in this input stream.
boolean	markSupported() Tests if this input stream supports the mark and reset methods.
abstract int	read() Reads the next byte of data from the input stream.
int	read(byte[] b) Reads some number of bytes from the input stream and stores them into the buffer array b.
int	read(byte[] b, int off, int len) Reads up to len bytes of data from the input stream into an array of bytes.
void	reset() Repositions this stream to the position at the time the mark method was last called on this input stream.
long	skip(long n) Skips over and discards n bytes of data from this input stream.

Clases InputStream Nodo

Clases InputStream Nodo

FileInputStream

A FileInputStream obtains input bytes from a file in a file system.

PipedInputStream

A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.

Clases InputStream Filtro

Clases InputStream Filtro
BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the mark and reset methods.
FilterInputStream contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.
ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.
DataInputStream lets an application read primitive Java data types from an underlying input stream in a machine-independent way
PushbackInputStream adds functionality to another input stream, namely the ability to "push back" or "unread" one byte

Clase Abstracta OutputStream

Method Summary	
	void **close**() Closes this output stream and releases any system resources associated with this stream.
	void **flush**() Flushes this output stream and forces any buffered output bytes to be written out.
	void **write**(byte[] b) Writes b.length bytes from the specified byte array to this output stream.
	void **write**(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to this output stream.
abstract	void **write**(int b) Writes the specified byte to this output stream.

Clases OutputStream Nodo

Clases OutputStream Nodo

FileOutputStream

A file output stream is an output stream for writing data to a File or to a FileDescriptor

PipedOutputStream

A piped output stream can be connected to a piped input stream to create a communications pipe

Clases OutputStream Filtro

Clases OutputStream Filtro
BufferedOutputStream the class implements a buffered output stream
FilterOutputStream is an output stream for writing data to a File or to a FileDescriptor
ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream
DataOutputStream lets an application write primitive Java data types to an output stream in a portable way
PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently

Clase Reader: Métodos

Reader Methods	
abstract void	**close**() Closes the stream and releases any system resources associated with it.
void	**mark**(int readAheadLimit) Marks the present position in the stream.
boolean	**markSupported**() Tells whether this stream supports the mark() operation.
int	**read**() Reads a single character.
int	**read**(char[] cbuf) Reads characters into an array.
abstract int	**read**(char[] cbuf, int off, int len) Reads characters into a portion of an array.
int	**read**(CharBuffer target) Attempts to read characters into the specified character buffer.
boolean	**ready**() Tells whether this stream is ready to be read.
void	**reset**() Resets the stream.
long	**skip**(long n) Skips characters.

Clases Reader Nodo

Clases Reader Nodo
FileReader Convenience class for reading character files
CharArrayReader This class implements a character buffer that can be used as a character-input stream
StringReader A character stream whose source is a string
PipedReader Piped character-input streams

Clases Reader Filter

Clases Reader Filter
BufferedReader Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines
FilterReader Abstract class for reading filtered character streams
InputStreamReader An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset
LineNumberReader A buffered character-input stream that keeps track of line numbers
PushbackReader A character-stream reader that allows characters to be pushed back into the stream

Clase Writer: Métodos

Writer Methods	
Writer	**append**(char c) Appends the specified character to this writer.
Writer	**append**(CharSequence csq) Appends the specified character sequence to this writer.
Writer	**append**(CharSequence csq, int start, int end) Appends a subsequence of the specified character sequence to this writer.
abstract void	**close**() Closes the stream, flushing it first.
abstract void	**flush**() Flushes the stream.
void	**write**(char[] cbuf) Writes an array of characters.
abstract void	**write**(char[] cbuf, int off, int len) Writes a portion of an array of characters.
void	**write**(int c) Writes a single character.
void	**write**(String str) Writes a string.
void	**write**(String str, int off, int len) Writes a portion of a string.

Clases Writer Nodo

Clases Writer Nodo
FileWriter Convenience class for writing character files
CharArrayWriter This class implements a character buffer that can be used as an writer
StringWriter A character stream that collects its output in a string buffer, which can then be used to construct a string
PipedWriter Prints formatted representations of objects to a text-output stream

Clases Writer Filtro

Clases Writer Filtro
BufferedWriter Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings
FilterWriter Abstract class for writing filtered character streams
OutputStreamWriter An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified charset
PrintWriter Prints formatted representations of objects to a text-output stream

Control del flujo de una operación I/O

Crear un objeto stream y asociarlo con la fuente de datos

Dar al objeto stream la funcionalidad deseada a través del encadenamiento del stream

while (hay más información)

    leer (escribir) siguiente dato desde (a) el stream

cerrar el stream

Byte Stream

- Los programas usan byte streams para realizar input y output de bytes (8-bit)
- Todas las clases byte stream descienden de InputStream y OutputStream
- Hay varias clases byte stream
  - FileInputStream y FileOutputStream
- Se usan de forma similar; la diferencia es la forma en que se construyen
- Se deben usar en I/O primitivo o de bajo nivel

Ejemplo: FileInputStream y FileOutputStream

```
public class CopyBytes {
  public static void main(String[] args) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
    try {
      in = new FileInputStream("prueba.txt");
      out = new FileOutputStream("byteprueba.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    } finally {
      if (in != null) { in.close(); }
      if (out != null) {
        out.close();
      }
    }
  }
}
```

Character Stream

- Java utiliza el código Unicode para los caracteres
- La I/O character stream convierte automáticamente este formato interno a y del conjunto de caracteres locales
- Todas las clases character stream descienden de Reader y Writer
- Como en los byte streams, hay clases character stream que se especializan en I/O de ficheros: FileReader y FileWriter

Ejemplo: FileReader y FileWriter

```
public class CopyCharacters {
  public static void main(String[] args) throws IOException {
    FileReader inputStream = null;
    FileWriter outputStream = null;
    try {
      inputStream = new FileReader("prueba.txt");
      outputStream = new FileWriter("characteroutput.txt");
      int c;
      while ((c = inputStream.read()) != -1) {
        outputStream.write(c);
      }
    } finally {
      if ((inputStream != null) {inputStream.close(); }
      if (outputStream != null) {
        outputStream.close();
      }
    }
  }
}
```

Buffered Streams

- Un unbuffered I/O significa que cada solicitud de lectura o escritura es gestionado directamente por el sistema operativo subyacente (ineficiente)
- Para reducir esta sobrecarga, Java implementa los buffered I/O streams
  - Con los buffered input streams se leen datos desde un area de memoria conocida como buffer; la API nativa se invoca sólo cuando el buffer está vacío
  - Para los buffered output streams la API se invoca cuando el buffre está lleno

Creación de Buffered Streams

- Un programa puede convertir un unbuffered stream en un buffered stream usando envolventes. Ejemplo:

```
inputStream =
new BufferedReader(new FileReader("prueba.txt"));
outputStream =
new BufferedWriter(new FileWriter("charoutput.txt"));
```

- Las clases buffered stream son:
  - *BufferedInputStream* y *BufferedOutputStream* crean buffered byte streams
  - *BufferedReader* and *BufferedWriter* crean buffered character streams

Ejemplo: escribe matriz con BufferedOutputStream

```
import java.io.*;
public class EscribeMatrizBufOutSt {
  static double[][] data = {
    { Math.exp(2.0), Math.exp(3.0), Math.exp(4.0) },
    { Math.exp(-2.0), Math.exp(-3.0), Math.exp(-4.0) },
  };
  public static void main(String[] args) {
    int row = data.length;
    int col = data[0].length;
    int i, j;
    for (i = 0; i < row; i++) {
      for (j = 0; j < col; j++) {
     System.out.println("dato[" + i + "][" + j + "] = " +
                        data[i][j]);
      }
    }
```

Ejemplo: escribe matriz con BufferedOutputStream

```
    if (args.length > 0) {
       try {
         DataOutputStream out =
          new DataOutputStream(new BufferedOutputStream(
          new FileOutputStream(args[0])));
         out.writeInt(row);  out.writeInt(col);
         for (i = 0; i < row; i++) {
           for (j = 0; j < col; j++) {
             out.writeDouble(data[i][j]);
           }
         }
          out.close();
       } catch (IOException e) {}
    }
  }
}
```

Ejemplo: lee matriz con BufferedInputStream

```
import java.io.*;
public class LeeMatrizBufInp {
  static double[ ][ ] data;
  public static void main(String[] args) {
    if (args.length > 0) {
      try {
          DataInputStream in =
            new DataInputStream(new BufferedInputStream(
            new FileInputStream(args[0])));
          int row = in.readInt();
          System.out.println("fila = " + row);
          int col = in.readInt();
          System.out.println("columna = " + col);
          data = new double[row][col];
          for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
              data[i][j] = in.readDouble();
              System.out.println("dato[" + i + "][" + j
                                 + "] = " + data[i][j]);
            }
          }
      } catch (IOException e) {}
    }  }
}
```

Uso de Reader y Writer

```
BufferedReader inp =
   new BufferedReader(new FileReader("matriz.dat"));

BufferedReader inp =
 new BufferedReader(new InputStreamReader(System.in));

PrintWriter out =
   new PrintWriter(new BufferedWriter(
          new FileWriter("matriz.dat")));

Writer out =
   new BufferedWriter(new
          OutputStreamWriter(System.out));
```

Estándar Streams en Java

- Tres estándar streams
  - Estándar Input, accedido a través de *System.in*
  - Estándar Output, accedido a través de *System.out*
  - Estándar Error, accedido a través de *System.err*
- Estos objetos son definidos automáticamente y no requieren ser abiertos
- System.out y System.err son definidos como objetos PrintStream

Data Streams

- Data Streams soportan I/O binaria de valores de tipos de datos primitivos (boolean, char, byte, short, int, long, float, y double) así como valores String
- Todos los data streams implementan las interfaces *DataInput* o *DataOutput*
- Las implementaciones más utilizadas de esas interfaces son *DataInputStream* y *DataOutputStream*

DataOutputStream

- *DataOutputStream* sólo puede ser creado como una envolvente para un objeto byte stream existente

```
out = new DataOutputStream(
        new BufferedOutputStream(
        new FileOutputStream(dataFile)));
for (int i = 0; i<prices.length; i++) {
  out.writeDouble(prices[i]);
  out.writeInt(units[i]);
  out.writeUTF(descs[i]);
}
```

Ejemplo: escribe matriz con DataOutput

```
import java.io.*;
public class EscribeMatrizDataOut {
  static double[][] data = {
    { Math.exp(2.0), Math.exp(3.0), Math.exp(4.0) },
    { Math.exp(-2.0), Math.exp(-3.0), Math.exp(-4.0) },
  };
  public static void main(String[] args) {
    int row = data.length;
    int col = data[0].length;
    int i, j;
    for (i = 0; i < row; i++) {
      for (j = 0; j < col; j++) {
     System.out.println("dato[" + i + "][" + j + "] = " +
                        data[i][j]);
      }
    }
```

Ejemplo: escribe matriz con DataOutput

```
    if (args.length > 0) {
      try {
        DataOutputStream out =
           new DataOutputStream(new
                                FileOutputStream(args[0]));
        out.writeInt(row);  out.writeInt(col);
        for (i = 0; i < row; i++) {
          for (j = 0; j < col; j++) {
            out.writeDouble(data[i][j]);
          }
         }
        out.close();
      } catch (IOException e) {}
    }
  }
}
```

Ejemplo: lee matriz con DataInput

```
import java.io.*;
public class LeeMatrizDataInp {
  static double[ ][ ] data;
  public static void main(String[] args) {
    if (args.length > 0) {
      try {
          DataInputStream in =
            new DataInputStream(new FileInputStream(args[0]));
          int row = in.readInt();
          System.out.println("fila = " + row);
          int col = in.readInt();
          System.out.println("columna = " + col);
          data = new double[row][col];
          for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
              data[i][j] = in.readDouble();
              System.out.println("dato[" + i + "][" + j
                                 + "] = " + data[i][j]);
            }
          }
      } catch (IOException e) {}
    }
  }
}
```

DataInputStream

- *DataInputStream* también debe ser creado como una envolvente para un objeto byte stream existente
- La condición End-of-File se detecta capturando EOFException

```
in = new DataInputStream(
       new BufferedInputStream(
       new FileInputStream(dataFile)));
try{
   double price = in.readDouble();
   int unit = in.readInt();
   String desc = in.readUTF();
} catch (EOFException e){ }
```

Object Streams

- Object Streams soportan I/O de objetos
  - Como los Data streams soportan I/O de tipos de datos primitivos
  - El objeto tiene que ser de tipo *Serializable*
- Las clases object stream son *ObjectInputStream* y *ObjectOutputStream*
- Los métodos writeObject y readObject son simples de usar, pero contienen una lógica de gestión de objetos compleja cuando los objetos tiene referencias a otros objetos

Ejemplo: escribe matriz como objeto

```
import java.io.*;
public class EscribeMatrizObj {
  static double[][] data = {
    { Math.exp(2.0), Math.exp(3.0), Math.exp(4.0) },
    { Math.exp(-2.0), Math.exp(-3.0), Math.exp(-4.0) },
  };
  public static void main(String[] args) {
    int row = data.length;
    int col = data[0].length;
    int i, j;
    for (i = 0; i < row; i++) {
      for (j = 0; j < col; j++) {
     System.out.println("dato[" + i + "][" + j + "] = " +
                        data[i][j]);
      }
    }
```

Ejemplo: escribe matriz como objeto

```
    if (args.length > 0) {
      try {
        ObjectOutputStream out = new ObjectOutputStream(
                                 new FileOutputStream(args[0]));
        out.writeObject(data);
        out.close();
      } catch (IOException e) {}
    }
  }
}
```

Ejemplo: lee matriz como objeto

```
import java.io.*;
public class LeeMatrizDataObj {
  static double[ ][ ] data;
  public static void main(String[] args) {
    if (args.length > 0) {
      try {
        ObjectInputStream in =
                        new ObjectInputStream(new FileInputStream(args[0]));
        data = (double[][]) in.readObject();
        int row = data.length;
        int col = data[0].length;
        for (int i = 0; i < row; i++) {
          for (int j = 0; j < col; j++) {
            System.out.println("dato[" + i + "][" + j + "] = "+data[i][j]);
          }
        }
      } catch (IOException e) {}
    }
  }
}
```

Clase File

- La clase File no es un stream
- Es importante porque las clases stream manipulan objetos File
- Son una representación abstracta de los ficheros y pathname de directorios

Clase File: Constructores

Constructor Summary

File(File parent, String child)
Creates a new File instance from a parent abstract pathname and a child pathname string.

File(String pathname)
Creates a new File instance by converting the given pathname string into an abstract pathname.

File(String parent, String child)
Creates a new File instance from a parent pathname string and a child pathname string.

File(URI uri)
Creates a new File instance by converting the given file: URI into an abstract pathname

Clase File: Métodos

Method Summary	
boolean	**canExecute**() Tests whether the application can execute the file denoted by this abstract pathname.
boolean	**canRead**() Tests whether the application can read the file denoted by this abstract pathname.
boolean	**canWrite**() Tests whether the application can modify the file denoted by this abstract pathname.
int	**compareTo**(File pathname) Compares two abstract pathnames lexicographically.
boolean	**createNewFile**() Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
static File	**createTempFile**(String prefix, String suffix) Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static File	**createTempFile**(String prefix, String suffix, File directory) Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
boolean	**delete**() Deletes the file or directory denoted by this abstract pathname.
void	**deleteOnExit**() Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates

Clase File: Métodos

Method Summary	
boolean	**equals**(Object obj) Tests this abstract pathname for equality with the given object.
boolean	**exists**() Tests whether the file or directory denoted by this abstract pathname exists.
File	**getAbsoluteFile**() Returns the absolute form of this abstract pathname.
String	**getAbsolutePath**() Returns the absolute pathname string of this abstract pathname.
File	**getCanonicalFile**() Returns the canonical form of this abstract pathname.
String	**getCanonicalPath**() Returns the canonical pathname string of this abstract pathname.
long	**getFreeSpace**() Returns the number of unallocated bytes in the partition named by this abstract path name.
String	**getName**() Returns the name of the file or directory denoted by this abstract pathname.
String	**getParent**() Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.

Clase File: Métodos

Method Summary	
File	**getParentFile**() Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
String	**getPath**() Converts this abstract pathname into a pathname string.
long	**getTotalSpace**() Returns the size of the partition named by this abstract pathname.
long	**getUsableSpace**() Returns the number of bytes available to this virtual machine on the partition named by this abstract pathname.
int	**hashCode**() Computes a hash code for this abstract pathname.
boolean	**isAbsolute**() Tests whether this abstract pathname is absolute.
boolean	**isDirectory**() Tests whether the file denoted by this abstract pathname is a directory.
boolean	**isFile**() Tests whether the file denoted by this abstract pathname is a normal file.
boolean	**isHidden**() Tests whether the file named by this abstract pathname is a hidden file.

Clase File: Métodos

Method Summary	
long	**lastModified**() Returns the time that the file denoted by this abstract pathname was last modified.
long	**length**() Returns the length of the file denoted by this abstract pathname.
String[]	**list**() Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
String[]	**list**(FilenameFilter filter) Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[]	**listFiles**() Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
File[]	**listFiles**(FileFilter filter) Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[]	**listFiles**(FilenameFilter filter) Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
static File[]	**listRoots**() List the available filesystem roots.
boolean	**mkdir**() Creates the directory named by this abstract pathname.

Clase File: Métodos

Method Summary	
boolean	**mkdirs**() Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories.
boolean	**renameTo**(File dest) Renames the file denoted by this abstract pathname.
boolean	**setExecutable**(boolean executable) A convenience method to set the owner's execute permission for this abstract pathname.
boolean	**setExecutable**(boolean executable, boolean ownerOnly) Sets the owner's or everybody's execute permission for this abstract pathname.
boolean	**setLastModified**(long time) Sets the last-modified time of the file or directory named by this abstract pathname.
boolean	**setReadable**(boolean readable) A convenience method to set the owner's read permission for this abstract pathname.
boolean	**setReadable**(boolean readable, boolean ownerOnly) Sets the owner's or everybody's read permission for this abstract pathname.
boolean	**setReadOnly**() Marks the file or directory named by this abstract pathname so that only read operations are allowed.
boolean	**setWritable**(boolean writable) A convenience method to set the owner's write permission for this abstract pathname.

Clase File: Métodos

Method Summary
boolean **setWritable**(boolean writable, boolean ownerOnly) Sets the owner's or everybody's write permission for this abstract pathname.
String **toString**() Returns the pathname string of this abstract pathname.
URI **toURI**() Constructs a file: URI that represents this abstract pathname.

Ejemplo File Class

```
import java.io.*;

public class FileInfoClass {
   public static void main(String[] args) {
      String fileName = args[0];
      File fn = new File(fileName);
      System.out.println("Nombre: " + fn.getName());
      if (!fn.exists()) {   // Comprueba si el fichero existe
         System.out.println(fileName + " no existe");
         System.out.println("Crea directorio temporal...");
         fileName = "temp";
         fn = new File(fileName); fn.mkdir();
         System.out.println(fileName +
                            (fn.exists()? " existe":" no existe"));
         System.out.println(“Elimina directorio temporal...");
         fn.delete();
       }
       System.out.println(fileName + " es un " +
                          (fn.isFile()? "fichero":" directorio"));
```

Ejemplo File Class

```
        if (fn.isDirectory()) {
          String content[] = fn.list();
          System.out.println("Contenido de este directorio:");
          for (int i = 0; i < content.length; i++) {
            System.out.println(content[i]);
          }
        }
        if (!fn.canRead()) {
          System.out.println(fileName + " no se puede leer");
          return;
        }
        System.out.println(fileName + " is " + fn.length()
                           + " bytes long");
        System.out.println(fileName + " es " + fn.lastModified());

        if (!fn.canWrite()) {
           System.out.println(fileName + " no se puede escribir");
        }
    }
}
```

Internacionalización: codificación de caracteres

- Por defecto, la codificación de caracteres está especificada por una propiedad del sistema

```
file.encoding=ISO8859_1  (ISO-8859-1)  ASCII
```

- Se puede usar otras codificaciones mediante:

```
BufferedReader in =
new BufferedReader(new InputStreamReader(
      new FileInputStream("foo.in"), "GB18030"));
PrintWriter out =
   new PrintWriter(new BufferedWriter(
    new OutputStreamWriter(
        new FileOutputStream("foo.out", "GB18030"))));
```