Entrada y salida en Java

- El paquete **`java.io`**.
- Flujos de datos (*streams*).
  - Flujos de octetos (*bytes)*.
  - Flujos de caracteres.
- La clase **`File`**.
- Serialización de objetos.

El paquete java.io

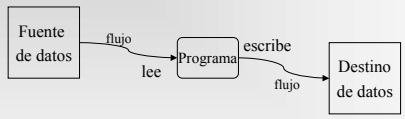
- Este paquete proporciona al sistema las entradas y salidas a través de flujos de datos, las serializaciones y el acceso al sistema de ficheros.
- Está constituido por una serie de interfaces y clases destinadas a definir y controlar los distintos tipos de flujos, el sistema de ficheros y la serializaciones de objetos.

Flujos de datos

- Para controlar el intercambio de información entre un programa y los dispositivos de almacenamiento o de comunicación, Java utiliza la noción de (objeto) flujo o stream.
- Java distingue entre:
  - **flujos de caracteres y flujos de bytes** (16 u 8 bits)
  - **flujos de entrada y flujos de salida**
  - **flujos primarios (iniciales) y flujos secundarios (no iniciales)**

Entrada/Salida

- Basada en Streams (flujos).
  - Esquema de funcionamiento:



  - Fuentes y destinos:
    - ✓ Un array de bytes,
    - ✓ un fichero,
    - ✓ un pipe,
    - ✓ una conexión de red,
    - ✓ ...

La clase `File`

- Representa caminos abstractos: prefijo de unidad y secuencia de nombres, de posibles ficheros o directorios (no son cadenas).
- Constructores:

```
File(File padre, String fichero)
File(String padre, String fichero)
File(String rutaFichero)
```

- Los objetos de esta clase se usarán para la creación de flujos sobre ficheros.

La clase `File`. Independencia del SO

- Constantes definidas en la clase File
  - Separador de nombres en un path

```
char separatorChar      '\' '/' ':'
String separator        "\" "/" ":"
```

  - Separador de un path de otro

```
char pathSeparatorChar ':' ';'
String pathSeparator   ":" ";"
```

- Si queremos trabajar con \libro\capitulo1

```
new File(File.separator + "libro"
         + File.separator + "capitulo1");
```

La clase **File**. Métodos de instancia (I)

- Nombre de ficheros
```
String getName()
String getParent()
String getAbsolutePath()
String getPath()
boolean renameTo(File nuevoNombre)
```
- Predicados sobre ficheros
```
boolean exists()
boolean canWrite()
boolean canRead()
boolean isFile()
boolean isDirectory()
boolean isAbsolute()
```
- Información general
```
long lastModified()
long length()
```
- Borrar un fichero
```
boolean delete()
```


La clase **File**. Métodos de instancia (II)

- Métodos para crear ficheros y directorios
```
boolean createNewFile()
boolean mkdir()
boolean mkdirs()
```
- Métodos para listar directorios
```
String[] list()
String[] list(FilenameFilter)
File[] listFiles()
File[] listFiles(FilenameFilter)
File[] listFiles(FileFilter)
....

interface FilenameFilter {
  boolean accept(File dirActual, String ent);
}
interface FileFilter {
  boolean accept(File path);
}
```


Ej: Listado recursivo de un directorio

```
import java.io.*;

public class DirRec {
  public static void main(String args[]) {
    if (args.length == 0) {
      System.err.println("Uso DirRec <directorio>");
    } else {
      dir(new File(args[0]));
    }
  }

  private static void dir(File entrada) {
     if (!entrada.exists()) {
        System.out.println(entrada.getName() + " no encontrado.");
     } else if (entrada.isFile()) {
        System.out.println(entrada.getAbsolutePath());
     } else if (entrada.isDirectory()) {
        File[] files = entrada.listFiles();
        if (files.length > 0) {
          for (File f : files) {
             dir(f);
          }
        } else {
             System.out.println("vacío");
        }
     }
```

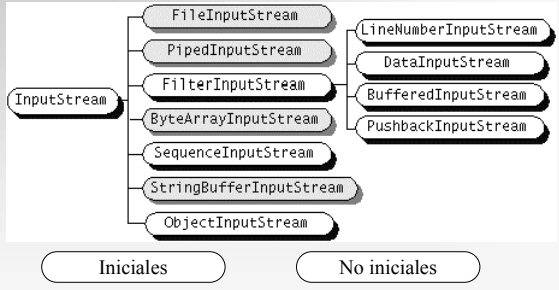

Streams (Flujos)

- Dos grupos de flujos:
  - Flujos binarios (de bytes)
    - ✓Byte Stream. Clases abstractas principales:
      - **InputStream** (`read()`)
      - **OutputStream** (`write(int b)`)
  - Flujos de texto (de caracteres unicode)
    - ✓Character Stream. Clases abstractas principales:
      - **Reader** (`read(char[] b,int o,int l)`)
      - **Writer** (`write(char[] b,int o,int l)`)

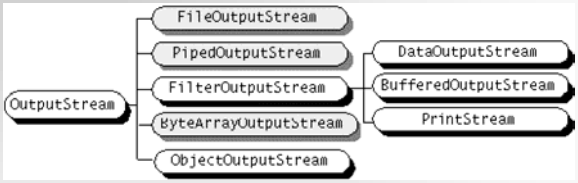
Streams de bytes: **InputStream** y **OutputStream**

- Clases abstractas que definen el comportamiento mínimo de estos flujos.
  IOException si hay error.
- Métodos de instancia de **InputStream:**
```
int read()           // lee un byte y lo devuelve como int
int read(byte[] buf) // lee varios bytes y los guarda en buf
int read(byte[] buf, int offset, int count);
// devuelven -1 si no se lee nada porque se alcanza el final del stream
long skip(long n)    // descarta n bytes de la entrada
```
- Métodos de instancia de **OutputStream**:
```
void write(int buf)
void write(byte[] buf)
void write(byte[] buf, int offset, int count)
void flush();      // descarga el buffer hacia la salida
```
- Métodos de instancias comunes:
```
void close()
```


La familia **InputStream**



La familia **OutputStream**



Iniciales

No iniciales

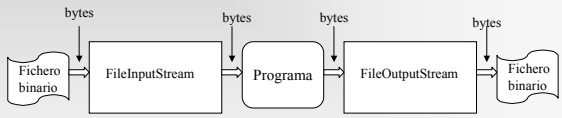
Streams sobre ficheros

- **FileInputStream**
  - `FileInputStream(String name)`
  - `FileInputStream(File name)`
- **FileOutputStream**
  - `FileOutputStream(String name)`
  - `FileOutputStream(String name, boolean append)`
  - `FileOutputStream(File name)`
- Los constructores producen **FileNotFoundException**

Ej: Copia

```
import java.io.*;
public class Copia {
  public static void main(String args[]) {
    FileInputStream  desdeF = null;
    FileOutputStream hastaF = null;
    try {
      desdeF = new FileInputStream(args[0]);
      hastaF = new FileOutputStream(args[1]);
      // Copia de los bytes
      int i = desdeF.read();
      while (i != -1) { // -1 si se alcanza el fin de fichero
        hastaF.write(i);
        i = desdeF.read();
      }
      desdeF.close();
      hastaF.close();
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.err.println("Uso: Copia <origen> <destino>");
    }
    catch (FileNotFoundException e) {
      System.err.println("No existe " + e);
    }
    catch (IOException e) {
      System.err.println("Error de E/S " + e);
    }
  }
}
```


Representación abstracta del programa **Copia**



Filtros

FilterInputStream y **FilterOutputStream**

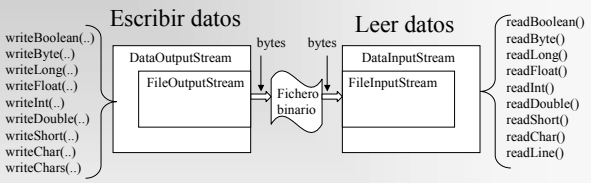
- Envuelven y actúan sobre otros streams proporcionando alguna funcionalidad adicional.
- Únicamente reescriben los métodos de las respectivas clases abstractas pasando las operaciones al stream sobre el que actúan.

- **DataInputStream** y **DataOutputStream**
  - Permiten que las aplicaciones puedan leer y escribir datos de tipos simples de Java desde/sobre el flujo que envuelven.
  - Deben usarse en correspondencia.
- **BufferedInputStream** y **BufferedOutputStream**
  - Proporcionan eficiencia en lectura y escritura.

Filtros: **DataInputStream** y **DataOutputStream**

- Constructores
  - `DataInputStream(InputStream ent)`
  - `DataOutputStream(OutputStream sal)`
- Métodos de instancia
  - Para cada tipo básico existe (también para **String**)
    - `xxxxx readXxxxx()`
    - `void  writeXxxxx(xxxxx)`

Representación abstracta




```
FileOutputStream fos = new FileOutputStream("datos.dat");
DataOutputStream dos = new DataOutputStream(fos);

FileInputStream ldF = new FileInputStream("datos.dat");
DataInputStream disF = new DataInputStream(ldF);
```



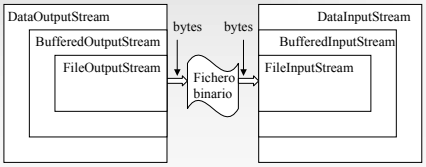
```
import java.io.*;
public class Datos {
  public static void main(String args[]) throws IOException {
    FileOutputStream gdF = new FileOutputStream("datos.dat");
    DataOutputStream dosF = new DataOutputStream(gdF);
    // Escribimos algunos datos
    dosF.writeBoolean(true);
    dosF.writeChar('A');
    dosF.writeByte(Byte.MAX_VALUE);
    dosF.writeInt(Integer.MAX_VALUE);
    dosF.writeDouble(Double.MAX_VALUE);
    dosF.close();  // Cerramos el flujo. Cierra todos
    // Creamos un flujo de entrada de datos
    FileInputStream ldF = new FileInputStream("datos.dat");
    DataInputStream disF = new DataInputStream(ldF);
    // Leemos los datos guardados
    boolean v = disF.readBoolean();
    char    c = disF.readChar();
    byte    b = disF.readByte();
    int     i = disF.readInt();
    double  d = disF.readDouble();
    // Cerramos el flujo
    disF.close();          // cierra todo
    // Mostramos los datos
    System.out.println(v);
    System.out.println(c);
    System.out.println(b);
    System.out.println(i);
    System.out.println(d);
  }
}
```

Ej: **DataInputStream** y **DataOutputStream**

Filtros: **BufferedInputStream** y **BufferedOutputStream**

Proporcionan eficiencia a la hora de leer o escribir mediante el uso de un buffer intermedio

- Constructores:
  BufferedInputStream(InputStream ent)
  BufferedOutputStream(OutputStream sal)



Ej: **BufferedInputStream** y **BufferedOutputStream**

```
import java.io.*;

public class Datos {
  public static void main(String[] args) throws IOException {
    FileOutputStream gdF = new FileOutputStream("datos.dat");
    BufferedOutputStream bosF = new BufferedOutputStream(gdF);
    DataOutputStream dosF = new DataOutputStream(bosF);
    // Escribimos algunos datos
    dosF.writeBoolean(true);
    ...
    // Creamos un flujo de entrada de datos
    FileInputStream ldF = new FileInputStream("datos.dat");
    BufferedInputStream bisF = new BufferedInputStream(ldF);
    DataInputStream disF = new DataInputStream(bisF);
    // Leemos los datos guardados
    boolean v = disF.readBoolean();
    ...
```


Otros **InputStream** y **OutputStream**

Otros flujos de entrada y de salida:

- **ByteArrayInputStream**
- **ByteArrayOutputStream**

(para leer o escribir sobre un buffer o array de bytes)

- **PipedInputStream**
- **PipedOutputStream**

(para establecer un flujo de datos entre distintas hebras)

- **SequenceInputStream**

(para encadenar/secuenciar varios streams de entrada)

Streams orientados a caracteres. **Reader** y **Writer**

- Clases abstractas que definen el comportamiento mínimo de estos flujos
  IOException si hay error
  - Métodos de instancia de **Reader**

```
int read()
int read(char[] b)
int read(char[] b, int off, int len);
  // devuelve -1 si no hay nada que leer
long skip(long n)
```

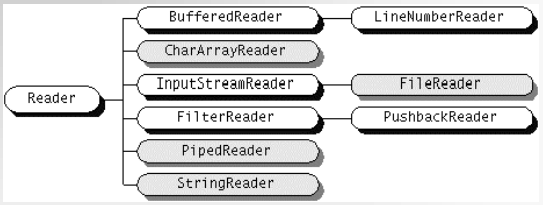
  - Métodos de instancia de **Writer**

```
void write(int b)
void write(char[] b)
void write(String s)
void write(char[] b, int off, int len);
void write(String s, int off, int len);
void flush()
```

  - Métodos de instancias comunes

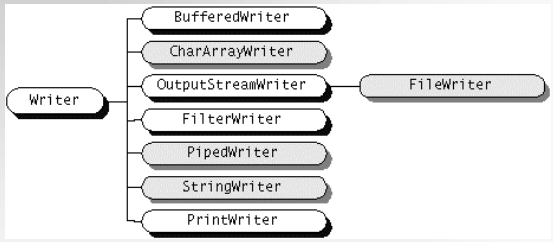
```
void close()
```

La familia **Reader**



Iniciales | No iniciales

La familia **Writer**



Iniciales | No iniciales

Codificación de caracteres

- Java utiliza el juego de caracteres UNICODE
- Cada plataforma tiene una codificación por defecto pero puede alterarse
- Las clases **`Reader`** y **`Writer`** necesitan de un codificador y decodificador
  - **`InputStreamReader`**
    (lee bytes y los decodifica como caracteres)
  - **`OutputStreamWriter`**
    (recibe caracteres y los codifica en bytes)

Normas de codificación

- 8859_1 ISO Latin-1 (contiene ASCII)
- 8859_2 ISO Latin-2
- 8859_3 ISO Latin-3
- 8859_4 ISO Latin/Cyrillic
- UTF8 Standard UTF-8 (cont. ASCII)

`InputStreamReader` y **`OutputStreamWriter`**

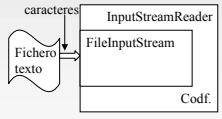
Actúan de puente entre flujos de byte y flujos de caracteres con ayuda de un sistema de codificación. Utilizan un buffer de bytes para agilizar la conversión

- Constructores:
```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, String codigo)
OutputStreamWriter(OutputStream sal)
OutputStreamWriter(OutputStream sal, String codigo)
```
- Métodos de instancia
```
String getEncoding()
```


Lectura de fichero. Opción 1ª

- 1) Crear un **`FileInputStream`**
```
FileInputStream fisF = new FileInputStream("datos.tex");
```
- 2) Crear un **`InputStreamReader`**
```
InputStreamReader isrF = new InputStreamReader(fisF);
```
  - ✓ Aquí podría especificarse un codificador, p.e.:
```
InputStreamReader isrF =
      new InputStreamReader(fisF,"8859_1");
```

Puede leerse con **`read()`** y **`read(cbuf,off,len)`**



Escritura sobre un fichero. Opción 1ª

1. Crear un **`FileOutputStream`**

```
FileOutputStream fosF =
        new FileOutputStream("datos.tex");
```

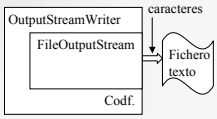
2. Crear un **`OutputStreamWriter`**

```
OutputStreamWriter oswF =
        new OutputStreamWriter(fosF);
```

   - ✓ Aquí podría especificarse un codificador

3. y ahora ...

```
oswF.write("Hola a todos",0,12);
```



`FileReader` y **`FileWriter`**

Son simplificaciones de las clases **`InputStreamReader`** y **`OutputStreamWriter`**.

- Aplican el sistema de codificación por defecto y un tamaño del buffer de bytes por defecto,
- Tienen la misma funcionalidad que sus clases madre.

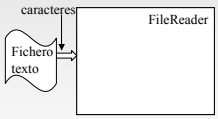
Lectura de fichero. Opción 2ª

Crear un **`FileReader`**

```
FileReader frF = new FileReader("datos.tex");
```

(Automáticamente se crea un **`FileInputStream`** seguido de un **`InputStreamReader`** con codificación por defecto)

Puede leerse con **`read()`** y **`read(cbuf,off,len)`**



Escritura sobre un fichero. Opción 2ª

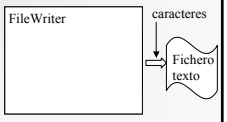
Crear un **`FileWriter`**

```
FileWriter fwF = new FileWriter("datos.tex");
```

(es equivalente a un **`FileOutputStream`** seguido de un **`OutputStreamWriter`** con decodificación por defecto)

y ahora ...

```
fwF.write("Hola a todos",0,12);
```

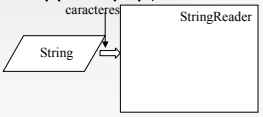


Otros **`Reader`**

Sustituyen a **`FileReader`**:

- **`StringReader`**
- **`PipedReader`**
- **`CharArrayReader`**

```
String st = "Esto es un String";
StringReader sw = new StringReader(st);
int c = sw.read();
while (c != -1) {
    System.out.println((char)c);
    c = sw.read();
}
```

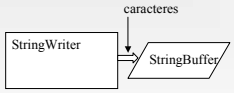


Otros **`Writer`**

Sustituyen a **`FileWriter`**:

- **`StringWriter`**
- **`PipedWriter`**
- **`CharArrayWriter`**

```
StringWriter sw = new StringWriter();
sw.write("Hola a todos");
System.out.println(sw.getBuffer());
```



La clase **PrintWriter**

Permite la impresión de representaciones formateadas de objetos sobre un stream de salida de texto o de bytes

```
PrintWriter(Writer sal)
PrintWriter(Writer sal, boolean flush)
PrintWriter(OutputStream sal)
PrintWriter(OutputStream sal, boolean flush)
```

Métodos de instancia

Para imprimir todos los tipos básicos y objetos:

```
print(xxx d)  println(xxx d)
```


Escritura sobre un fichero. Opción 3ª

1. Crear un **FileOutputStream**

```
FileOutputStream fosF =
        new FileOutputStream("datos.tex");
```

2. Crear un **OutputStreamWriter**

```
OutputStreamWriter oswF =
        new OutputStreamWriter(fosF);
```

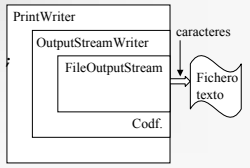
   ✓ Aquí podría especificarse un codificador

3. Crear un **PrintWriter**

```
PrintWriter pwF =
        new PrintWriter(oswF);
```

y ahora ...

```
pwF.println("Hola a todos");
```



Escritura sobre un fichero. Opción 4ª

1. Crear un **FileOutputStream**

```
FileOutputStream fosF =
        new FileOutputStream("datos.tex");
```

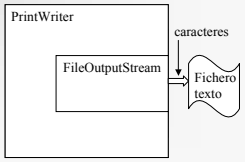
2. Crear un **PrintWriter**

```
PrintWriter pwF = new PrintWriter(fosF);
```

   ➢ Automáticamente se añade un **OutputStreamWriter** con decodificación por defecto

   ➢ y ahora ...

```
pwF.println("Hola a todos");
```



Escritura sobre un fichero. Opción 5ª

1. Crear un **FileWriter**

```
FileWriter fwF = new FileWriter("datos.tex");
```

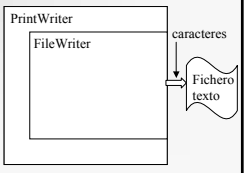
(equivalente a un **FileOutputStream** seguido de un **OutputStreamWriter** con decodificación por defecto)

2. Crear un **PrintWriter**

```
PrintWriter pwF = new PrintWriter(fosF);
```

y ahora ...

```
pwF.println("Hola a todos");
```



BufferedReader y **BufferedWriter**

Proporcionan eficiencia a la hora de leer o escribir. Utilizan un buffer intermedio cuyo tamaño se puede especificar

Constructores

```
BufferedReader(Reader ent)
BufferedReader(Reader ent, int size)
BufferedWriter(Writer sal)
BufferedWriter(Writer sal, int size)
```


BufferedReader y **BufferedWriter**

- **BufferedReader**
  - **Métodos de instancia**

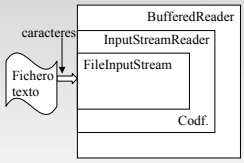
```
String readLine()
int read()
int read(Char[] c,int o,int l)
boolean ready()
boolean markSupported()
void mark(int readAheadLimit)
void reset()
long skip(long n)
void close()
```

- **BufferedWriter**
  - **Métodos de instancia**

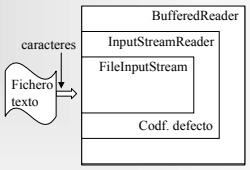
```
String newLine()
void write(int c)
...
void flush()
void close()
```

`BufferedReader` y ficheros

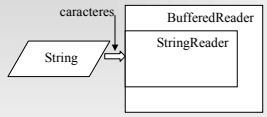
Opción 1ª



Opción 2ª

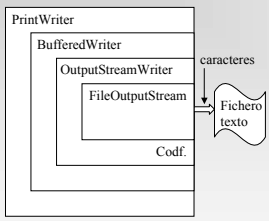


Lectura con buffer de `String`

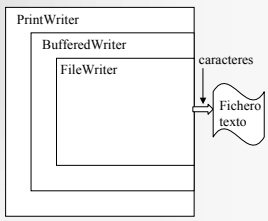


`BufferedWriter` y ficheros

Opción 1ª



Opción 2ª



Terminal E/S

- La clase **`System`** tiene tres variables de clase públicas
  ```
  InputStream in
  PrintStream out, err
  ```
- Para leer con buffer
  ```
  InputStreamReader isr = new InputStreamReader(System.in);
  BufferedReader stdIn = new BufferedReader(isr);
  String s = stdIn.readLine();
  ```
- La clase **`PrintStream`** se comporta igual que **`PrintWriter`** y ambas no provocan **`IOException`**
  ```
  System.out.print(...)
  System.out.println(...)
  ```

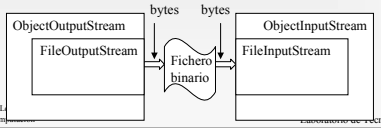

Serialización de objetos

Serialización de objetos

- Podemos incluir objetos dentro de un flujo y después extraerlos del mismo
  - La clase debe implementar la interface **`Serializable`**, que no tiene métodos
  - Cualquier variable definida como **`transient`** no será guardada
  - Se guardarán todos los objetos necesarios para reconstruir el objeto a guardar
  - Las clases de la librería de Java son todas serializables

`ObjectInputStream` y `ObjectOutputStream`

- **`ObjectOutputStream`**
  `ObjectOutputStream(OutputStream sal)`
  - Métodos de instancia
    `void writeObject(Object obj) throws IOException;`
- **`ObjectInputStream`**
  `ObjectInputStream(InputStream in)`
  - Métodos de instancia
    `Object readObject() throws OptionalDataException, ClassNotFoundException, IOException;`



Ejemplo. Guardar una lista

```
import java.io.*;
import java.util.*;

public class Serout {
  public static void main(String[] args)
                        throws IOException {
      List lista = new LinkedList();
      lista.add(new Integer(3));
      lista.add(new Character('a'));
      lista.add(new Double(23.5));
      System.out.println(lista);
      FileOutputStream fos =
             new FileOutputStream("obj.txt");
      ObjectOutputStream oos =
             new ObjectOutputStream(fos);
      oos.writeObject(lista);
      oos.close();
  }
}
```


Ejemplo.Cargar la lista guardada

```
import java.io.*;
import java.util.*;

public class Serin {
  public static void main(String [] args)
                             throws Exception {
      FileInputStream fos =
            new FileInputStream("obj.txt");
      ObjectInputStream ois =
            new ObjectInputStream(fos);
      List lista = (LinkedList) ois.readObject();
      ois.close();
      System.out.println(lista);
  }
}
```