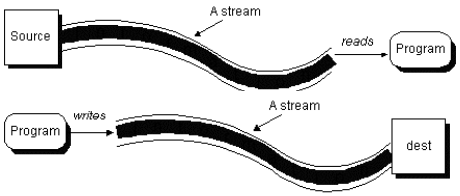
Entrada/Salida

- Basada en Streams (flujos)
  - Hay streams de lectura y de escritura



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

Ficheros. La clase File

- Proporciona herramientas básicas para la manipulación de ficheros
- Constructores
  - File(File path, String fichero)
  - File(String path, String fichero)
  - File(String pathfichero)
- Los objetos de esta clase pueden usarse para la creación de flujos sobre ficheros

Independencia del SO

- Costantes 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 trabajor con \libro\capitulo

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


La clase File. Métodos de instancia I

- Nombre de ficheros

```
String getName()          String getAbsolutePath()
String getPath()          String getParent()
boolean renameTo(File nuevoNombre)
```

- Comprobación de ficheros

```
boolean exist()         boolean isFile()
boolean canWrite()      boolean isDirectory()
boolean canRead()       boolean isAbsolute()
```

- Información general

```
long lasModified()      boolean delete()
long legth()
```

La clase File. Métodos de instancia II

– Herramientas para directorios

```
boolean mkdir()
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);
}
```


Streams (Flujos)

- Dos grupos de flujos
  - Flujos binarios (de bytes)
    - Byte Stream
      - `InputStream`
      - `OutputStream`
  - Flujos de texto (de caracteres)
    - Character Stream
      - `Reader`
      - `Writer`

Stream orientados a byte `InputStream` `OutputStream`

– Clases abstractas que definen el comportamiento mínimo de estos flujos
  - `IOException` si hay error

– Métodos de instancia de `InputStream`

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

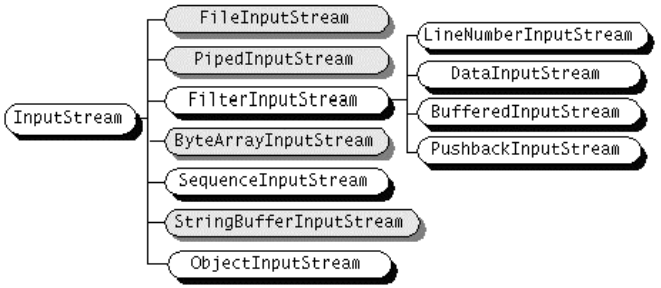
– Métodos de instancia de `OutputStream`

```
void write(int b)       void write(byte[] b)
void write(byte[] b, int off, int len);
```

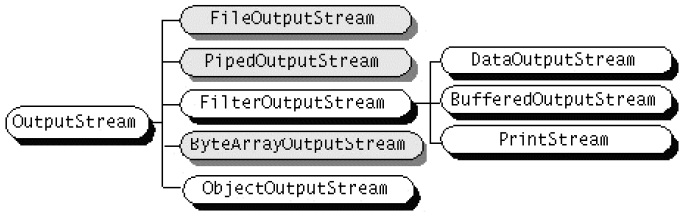
– Métodos de instancias comunes

```
void close()            void flush()
```


La familia InputStream



La familia OutputStream



Iniciales

No iniciales

Stream 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`

- Ejemplos: `Cat` y `Copia`

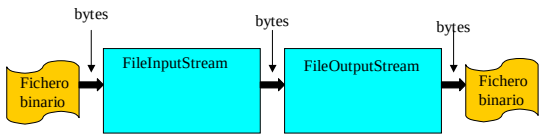
Copia I

```
import java.io.*;
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]);
       } catch (FileNotFoundException e) {
          System.err.println("No existe "+e);
          return;
       } catch (IOException e) {
          System.err.println("No se puede copiar "+e);
          return;
       } catch (ArrayIndexOutOfBoundsException e) {
          System.err.println("Uso: Copia origen destino ");
          return;
       }
```


Copia II

```
       // Copia de los bytes
       try {
          int i = desdeF.read();
          while (i!=-1) { // -1 es fin de fichero
             hastaF.write(i);
             i = desdeF.read();
          }
       } catch (IOException e) {
          System.err.println("Error de e/s: "+e);
       }
       // Cierre de ficheros
       try {
          desdeF.close();
          hastaF.close();
       } catch (IOException e) {
          System.err.println("Error cerrando "+e);
       }
    }
}
```

Representación abstracta del programa `Copia`



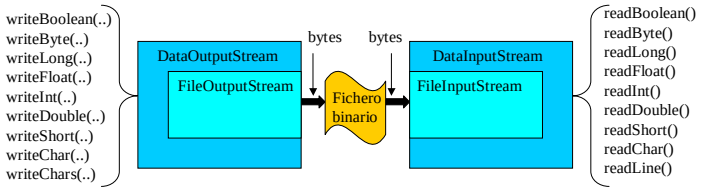
Filtros

- `FilterInputStream` y `FilterOutputStream`
  - Proporcionan funcionalidad adicional `DataInputStream` y `DataOutputStream`
    - Manipulan datos Java sobre el flujo `BufferedInputStream` y `BufferedOutputStream`
    - Proporcionan eficiencia en lectura y escritura

Filtros. `DataInputStream` y `DataOutputStream`

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

Representación abstracta



Ejemplo de DataInputStream y DataOutputStream I

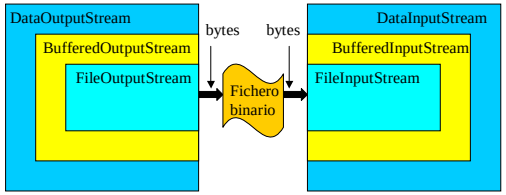
```
import java.io.*;

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);
     // Cerramos el flujo
     dosF.close();  // cierra todos
```


Ejemplo de DataInputStream y DataOutputStream II

```
     // 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);
   }
}
```


Filtros. BufferedInputStream y BufferedOutputStream

- Proporcionan eficiencia a la hora de leer o escribir
  - Utilizan un buffer intermedio
- `BufferedInputStream`
  `BufferedInputStream(InputStream ent)`
- `BufferedOutputStream`
  `BufferedOututStream(OutputStream sal)`



Ejemplo de BufferedInputStream y BufferedOutputStream I

```
import java.io.*;

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);
    ...
```

Ejemplo de `BufferedInputStream` y `BufferedOutputStream` II

```
...
// 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`

- Sustituyen a `FileInputStream`
  - `ByteArrayInputStream`
  - `PipedInputStream`
- `SequenceInputStream`
- Sustituyen a `FileOutputStream`
  - `ByteArrayOutputStream`
  - `PipedOutputStream`

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

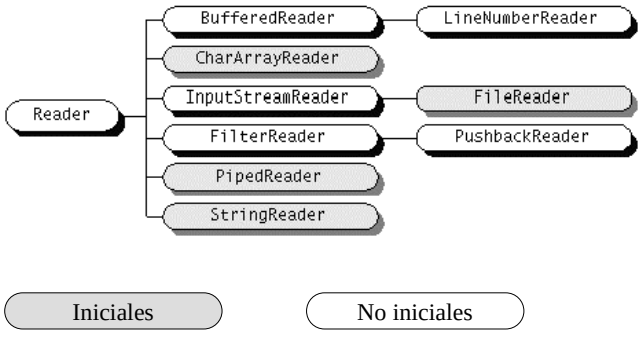
  - Métodos de instancia de `Writer`

```
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);
```

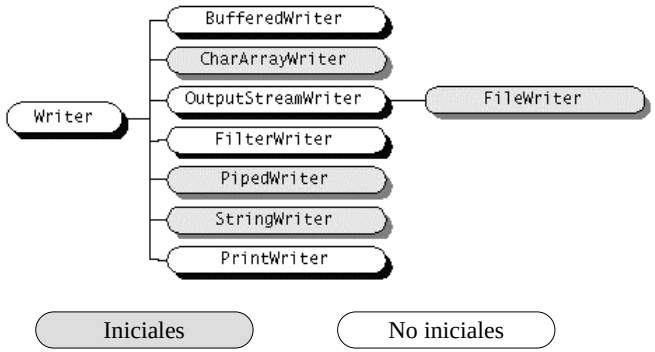
  - Métodos de instancias comunes

```
void close()            void flush()
```


La familia `Reader`



La familia Writer



Codificación de caracteres

- Java utiliza el caracter UNICODE
- Cada plataforma tiene una codificación por defecto pero puede alterarse
- Las clases `Reader` y `Writer` necesitan de un codificador y decodificador
  - `InputStreamReader`
  - `OutputStreamReader`

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`

- `InputStreamReader`

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, String codigo)
```

- `OutputStreamWriter`

```
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");
```

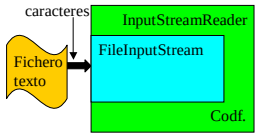
- 2) Crear un `InputStreamReader`

```
InputStreamReader isrF = new InputStreamReader(fisF);
```

  - Aquí podría especificarse un codificador

```
InputStreamReader isrF =
      new InputStreamReader(fisF,"8859_1");
```

  - Solo puede leerse con `read`



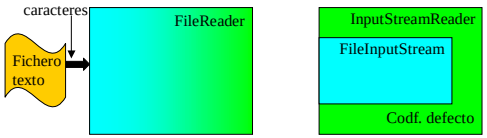
Lectura de fichero. Opción 2ª

- 1) Crear un `FileReader`

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

  - Automáticamente se crea un `FileInputStream` seguido de un `InputStreamReader` con codificación por defecto

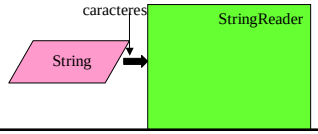
  - Solo puede leerse con `read`



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();
}
```



La clase `PrintWriter`

- Permite representar textualmente los tipos básicos de Java y los objetos

```
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(...)   println(...)
```

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);
```

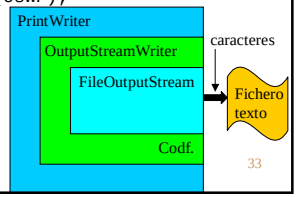
  - Aquí podría especificarse un codificador

- 3) Crear un `PrintWriter`

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

- y ahora ...

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



Escritura sobre un fichero. Opción 2ª

- 1) Crear un `FileOutputStream`

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

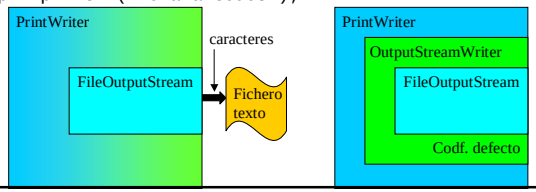
- 2) Crear un `PrintWriter`

```
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 3ª

- 1) Crear un `FileWriter`

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

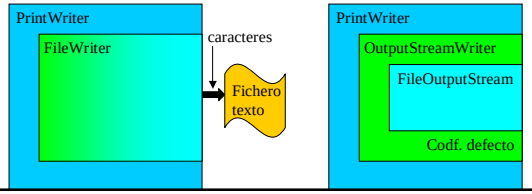
  - Un FileWriter es equivalente a un `FileOutputStream` seguido de un `OutputStreamWriter` con decodificación por defecto

- 2) Crear un `PrintWriter`

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

- y ahora ...

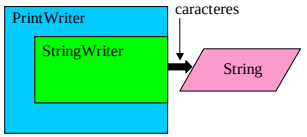
```
pwF.println("Hola a todos");
```



Otros `Writer`

- Sustituyen a `FileWriter`
  - `StringWriter`
  - `PipedWriter`
  - `CharArrayWriter`

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



BufferedReader y BufferedWriter

- Proporcionan eficiencia a la hora de leer o escribir
  - Utilizan un buffer intermedio
- `BufferedReader`

```
BufferedReader(Reader ent)
BufferedReader(Reader ent, int size)
```

- `Bufferedwriter`

```
BufferedWriter(Writer sal)
BufferedWriter(Writer sal, int size)
```


BufferedReader y BufferedWriter

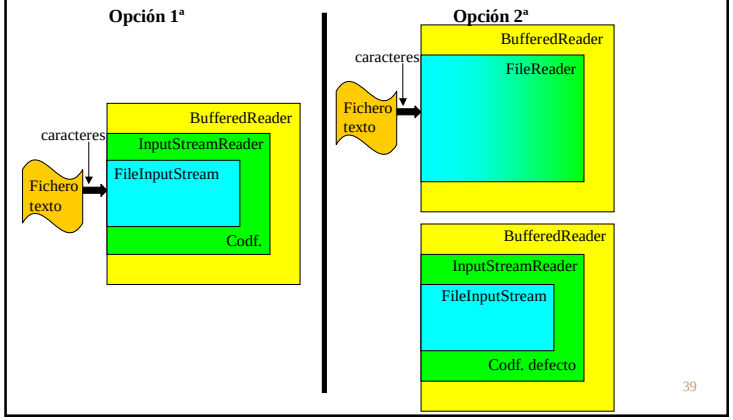
- `BufferedReader`
  - Método de instancia de

```
String readLine()       boolean ready()
void reset()            long skip(long)
void close()
```

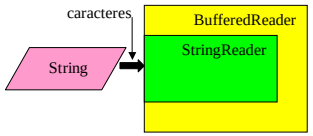
- `BufferedWriter`
  - Método de instancia de

```
String newLine()        void flush()
void close()
```

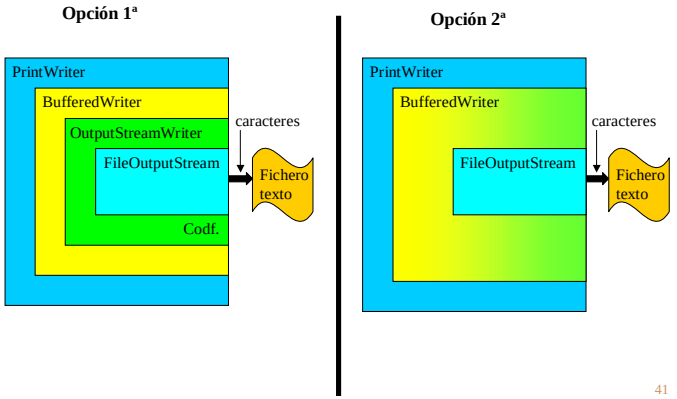

BufferedReader y ficheros



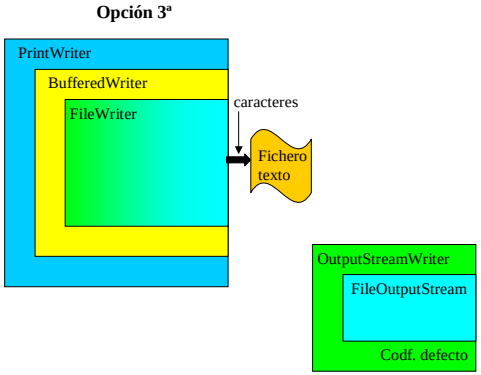
Lectura con buffer de `String`



BufferedWriter y ficheros I



BufferedWriter y ficheros II



Terminal E/S

- La clase `System` tiene tres variables de clase públicas

```
InputStream in
PrintStream out, err
```

  - Para leer con buffer

```
InputStreamReader ier = new InputStreamReader(System.in);
BufferedReader stdInt = new BufferedReader(isr);
String s = stdInt.readLine();
```

  - La clase `PrintStream` se comporta igual que `PrintWriter` pero no provoca `IOException`

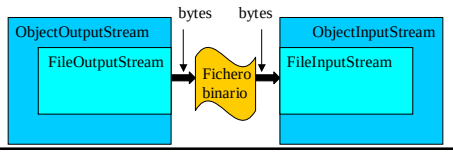
```
System.out.print(...)     System.out.println(...)
```


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 guardaran todos los objetos necesarios para reconstruir el objeto a guardar
  - Las clases de la librería de Java son todas serializables

ObjectInputStream y ObjectOutputStream

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



Ejemplo. Guardar una lista

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