Índice

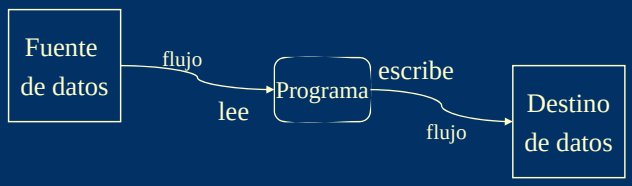
- Introd. a la POO
- El lenguaje Java
- Estruct. Biblioteca
- Excepciones
- Colecciones
- ✓Entrada y salida
- GUIs

Entrada y salida en un lenguaje orientado a objetos

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

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,
    - ✓ ...

La clase `File`

- Proporciona herramientas básicas para la manipulación de ficheros.
- Constructores:

```
File(File ruta, String fichero)
File(String ruta, 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\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)
```

- Predicados sobre ficheros

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

- Información general

```
long lastModified()
long length()
```

- Borrar un fichero

```
boolean delete()
```

VI-5

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

- Métodos para manipular 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);
}
```

VI-6

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
      new DirRec(new File(args[0]));
  }
  public DirRec(File entrada) {
    dir(entrada, 0);
  }
  private void dir(File entrada, int n) {
     if (!entrada.exists()) {
        muestra(n, entrada.getName() + " no encontrado.");
     } else if (entrada.isFile()) {
        muestra(n, entrada.getName());
     } else if (entrada.isDirectory()) {
        File[] files = entrada.listFiles();
        muestra(n, "DIRECTORIO: " + entrada);
        for (File f : files)
           dir(f, n + 1);
     }
  }
  private void muestra(int n, String s) {
    for (int i = 0; i < n ; i++)
      System.out.print("   ");
    System.out.println(s);
  }
}
```

VI-7

Streams (Flujos)

- Dos tipos de flujos:
  - Flujos binarios (de bytes, *byte streams*)
    - **`InputStream`**
    - **`OutputStream`**
  - Flujos de texto (de caracteres, *character streams*)
    - **`Reader`**
    - **`Writer`**

VI-8

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()
// devuelve -1 si se alcanza el final del stream
int read(byte[] buf)
int read(byte[] buf, int offset, int count)
// devuelven -1 si no se lee nada porque se alcanza
// el final del stream
```

- Métodos de instancia de **`OutputStream`**:

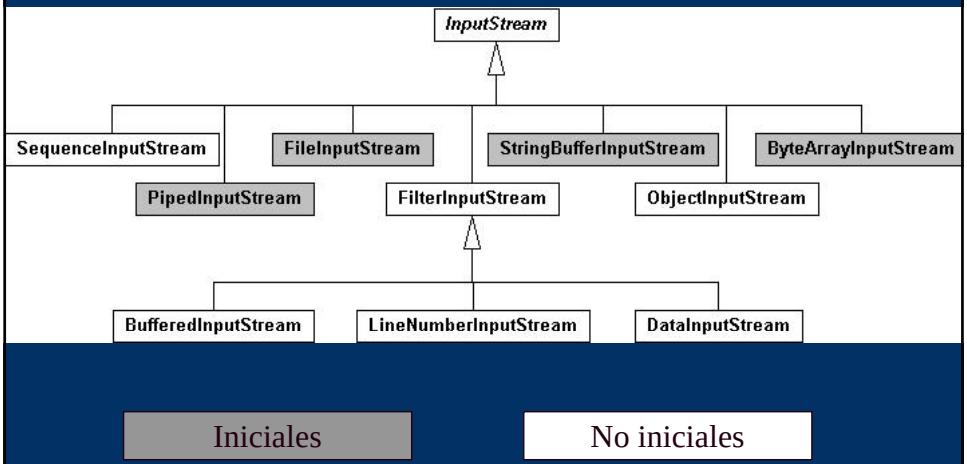
```
void write(int buf)
void write(byte[] buf)
void write(byte[] buf, int offset, int count)
void flush()
```

- Métodos de instancias comunes:

```
void close()
```

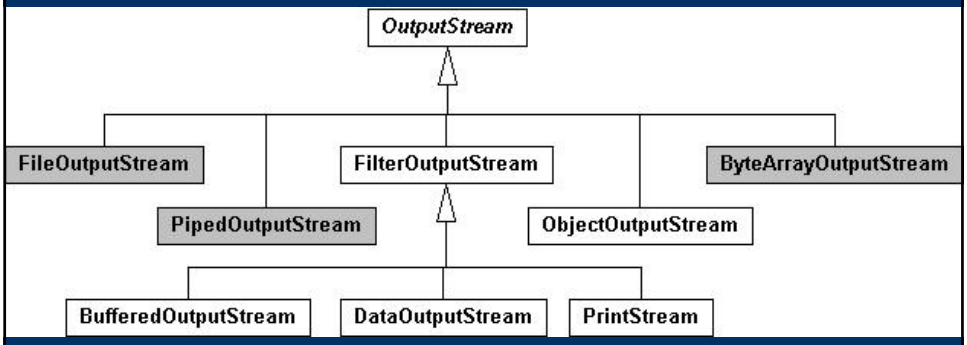
VI-9

La familia **`InputStream`**



VI-10

La familia **`OutputStream`**



Iniciales

No iniciales

VI-11

Streams sobre ficheros

- **`FileInputStream`**

  ```
  FileInputStream(String name)
  FileInputStream(File name)
  ```

- **`FileOutputStream`**

  ```
  FileOutputStream(String name)
  FileOutputStream(String name, boolean append)
  FileOutputStream(File name)
  ```

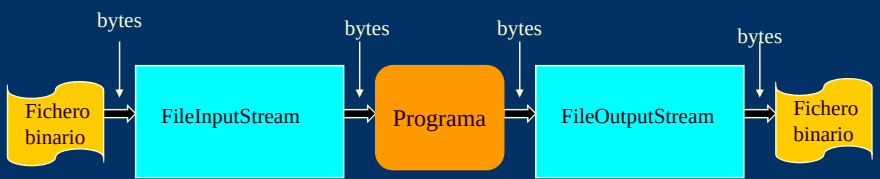
- Los constructores producen **`FileNotFoundException`**

VI-12

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 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

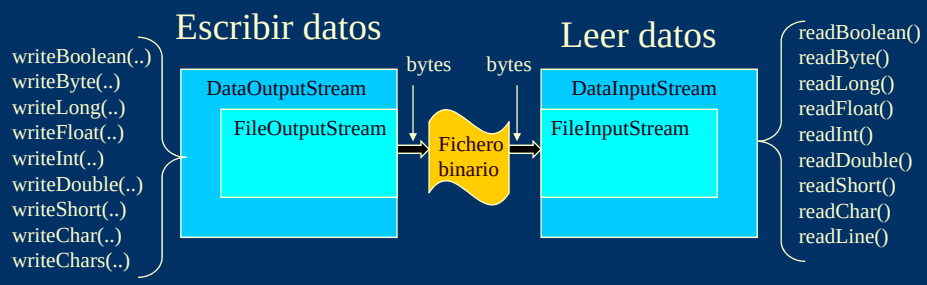
- **`FilterInputStream`** y **`FilterOutputStream`**
  - Proporcionan funcionalidad adicional.
- **`DataInputStream`** y **`DataOutputStream`**
  - Manipulan datos Java sobre el flujo.
- **`BufferedInputStream`** y **`BufferedOutputStream`**
  - Proporcionan eficiencia en lectura y escritura.

Filtros. **`DataInputStream`** y **`DataOutputStream`**

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

✓ Métodos de instancia de **`DataOutputStream`**
  - Para cada tipo básico existe (también para **`String`**)
    `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);
```

VI-17

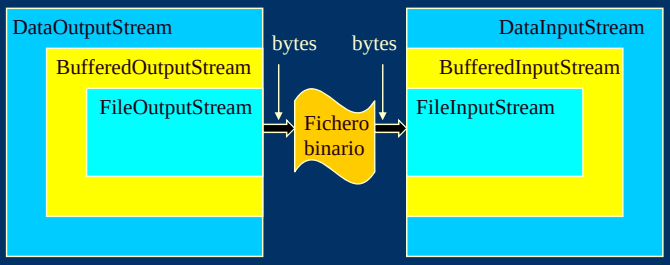
```
import java.io.*;
public class Datos {
  public static void main(String args[]) throws IOException {
    FileOutputStream gdF = new FileOutputStream("datos.dat");
    DataOutputStream dosF = new DataOutputStream(gdF);
    // Escribimos algunos datos
    dosF.writeBoolean(true);
    dosF.writeChar('A');
    dosF.writeByte(Byte.MAX_VALUE);
    dosF.writeInt(Integer.MAX_VALUE);
    dosF.writeDouble(Double.MAX_VALUE);
    // Cerramos el flujo
    dosF.close();
    // 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();
    // Mostramos los datos
    System.out.println(v);
    System.out.println(c);
    System.out.println(b);
    System.out.println(i);
    System.out.println(d);
  }
}
```

Ejemplo de **DataInputStream** y **DataOutputStream**

VI-18

Filtros. **BufferedInputStream** y **BufferedOutputStream**

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



VI-19

Ejemplo de **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);
  ...
  // Cerramos el flujo
  dosF.close();
  // 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();
  ...
  // Cerramos el flujo
  disF.close();
  ...
```

VI-20

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

- Otros flujos de entrada:
  - **`ByteArrayInputStream`**
  - **`PipedInputStream`**
  - **`SequenceInputStream`**
- Otros flujos de salida:
  - **`ByteArrayOutputStream`**
  - **`PipedOutputStream`**

Streams orientados a carácter. **`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
```

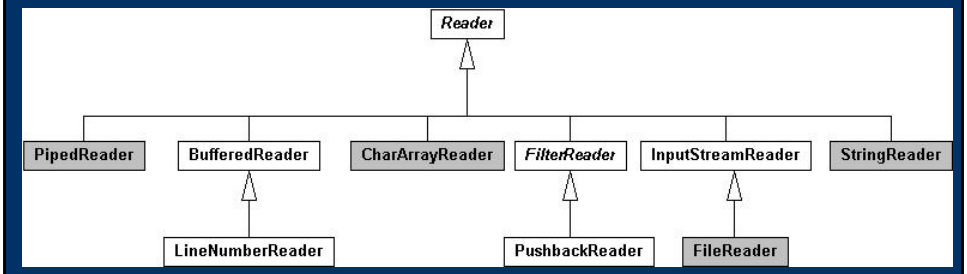
  - Métodos de instancia de **`Writer`**

```
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);
void flush()
```

  - Método de instancia común

```
void close()
```

La familia `Reader`

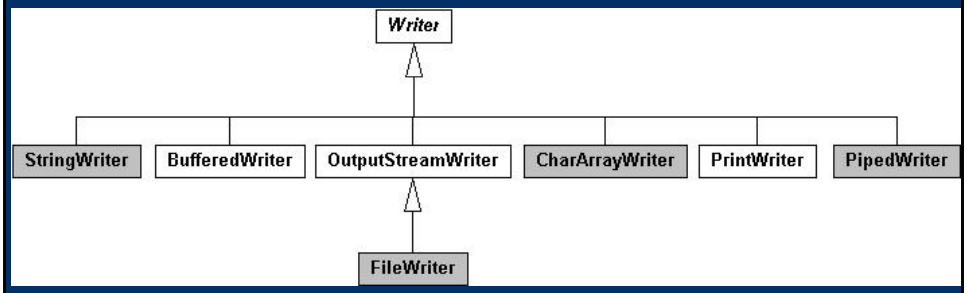


Iniciales

No iniciales

VI-23

La familia `Writer`



Iniciales

No iniciales

VI-24

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`**
  - **`OutputStreamReader`**

VI-25

Normas de codificación

- ISO-8859-1 ISO Latin-1 (contiene ASCII)
- ISO-8859-2 ISO Latin-2
- ISO-8859-3 ISO Latin-3
- ISO-8859-4 ISO Latin-4
- ISO-8859-5 ISO Latin/Cyrillic
- ...
- UTF-8 Standard UTF-8 (contiene ASCII)

VI-26

`InputStreamReader` y `OutputStreamWriter`

- **`InputStreamReader`**
  - `InputStreamReader(InputStream in)`
  - `InputStreamReader(InputStream in, String código)`
- **`OutputStreamWriter`**
  - `OutputStreamWriter(OutputStream sal)`
  - `OutputStreamWriter(OutputStream sal, String código)`
- Métodos de instancia
  - `String getEncoding()`

Lectura de fichero. Opción 1ª

- 1) Crear un **`FileInputStream`**

```
FileInputStream fisF = new FileInputStream("datos.tex");
```

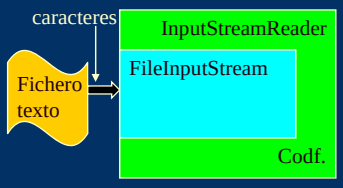
- 2) Crear un **`InputStreamReader`**

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

  - ✓ Aquí podría especificarse un codificador

```
InputStreamReader isrF =
   new InputStreamReader(fisF, "ISO-8859-1");
```

  - ➢ Sólo puede leerse con **`read()`**

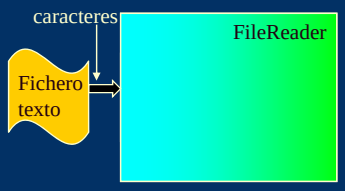


Lectura de fichero. Opción 2ª

- 1) Crear un `FileReader`

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

  - ✓ Automáticamente se crea un **`FileInputStream`** seguido de **un `InputStreamReader`** con codificación por defecto
  - ➢ Sólo puede leerse con **`read()`**

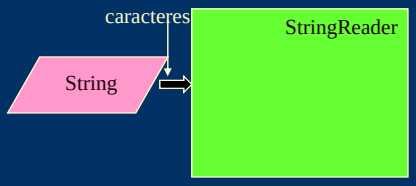


VI-29

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();
}
```



VI-30

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);
```

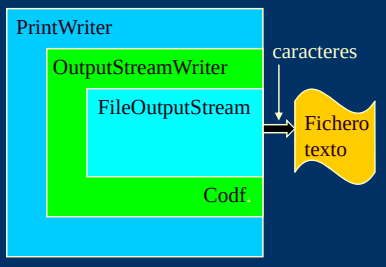
   ✓ Aquí podría especificarse un codificador

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");
```

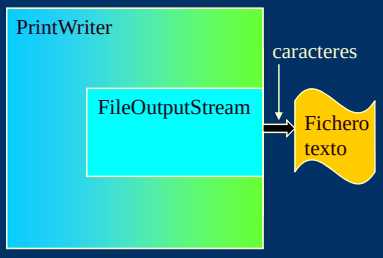
- 2) Crear un **`PrintWriter`**

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

  - Automáticamente se añade un **`OutputStreamWriter`** con decodificación por defecto

- y ahora ...

```
pwF.println(”Hola a todos");
```



VI-33

Escritura sobre un fichero. Opción 3ª

1) Crear un **`FileWriter`**

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

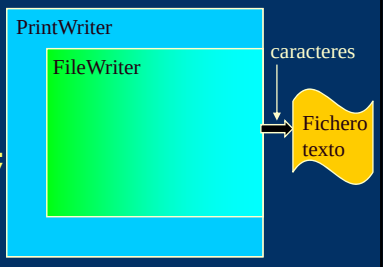
  - Un **`FileWriter`** es equivalente a un **`FileOutputStream`** seguido de un **`OutputStreamWriter`** con decodificación por defecto

2) Crear un **`PrintWriter`**

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

y ahora ...

```
pwF.println(”Hola a todos");
```

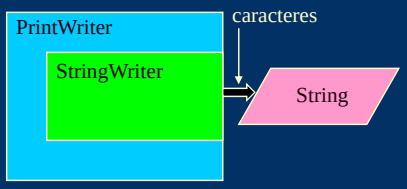


VI-34

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**
  - Métodos de instancia

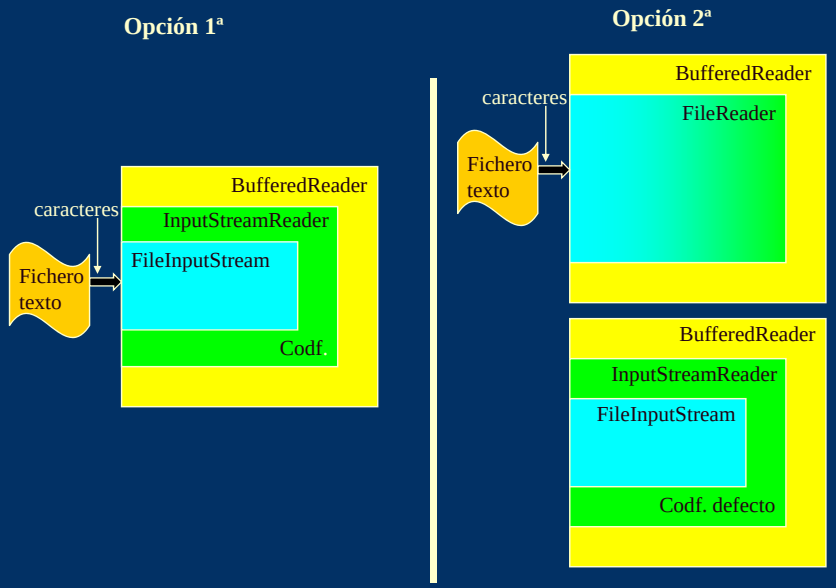
```
String readLine()
boolean ready()
boolean markSupported()
void mark(int readAheadLimit)
void reset()
long skip(long n)
void close()
```

- **BufferedWriter**
  - Métodos de instancia

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

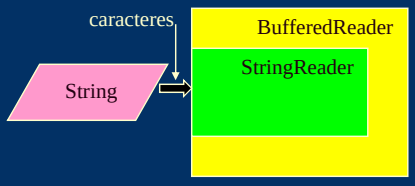
VI-37

BufferedReader y ficheros



VI-38

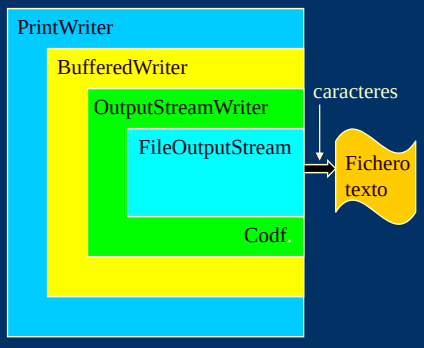
Lectura con buffer de `String`



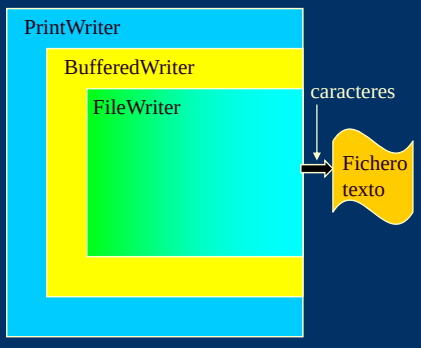
VI-39

`BufferedWriter` y ficheros (I)

Opción 1ª



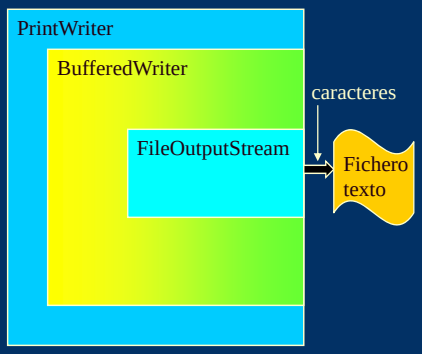
Opción 2ª



VI-40

`BufferedWriter` y ficheros (II)

Opción 3ª



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

- 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`

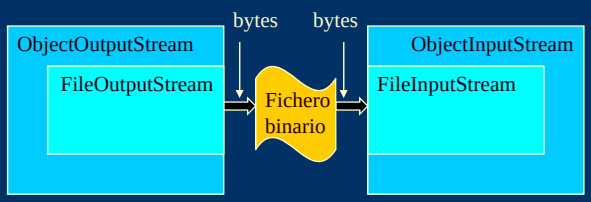
- **`ObjectOutputStream`**

  `ObjectOutputStream(OutputStream sal)`
  - Métodos de instancia

    `void writeObject(Object obj) throws IOException;`
- **`ObjectInputStream`**

  `ObjectInputStream(InputStream in)`
  - Métodos de instancia

    `Object readObject() throws IOException, ClassNotFoundException;`



Ejemplo. Guardar una lista

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

public class Serout {
  public static void main(String[] args)
      throws IOException {
      List<Number> lista = new LinkedList<Number>();
      lista.add(3);
      lista.add(4.5);
      lista.add(56.2);
      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<Number> lista =
           (List<Number>) ois.readObject();
    ois.close();
    System.out.println(lista);
  }
}
```