

# Entrada y salida de datos

# Contenido

- Entrada y salida de datos.
- El paquete java.io
- Ficheros. La clase File
- Flujos binarios
- Flujos de caracteres
- La clase Scanner

# Entrada y salida de datos (I/O)

- La entrada o salida de datos se refiere a la transferencia de datos entre un programa y los dispositivos de almacenamiento o de comunicación.
- La entrada se refiere a los datos que recibe el programa y la salida a los datos que transmite.
- La comunicación puede ser con un usuario humano, mediante un dispositivo físico (pantalla, teclado, impresora, ...), con otro sistema (s. de ficheros o programas) dentro del mismo computador o en un computador remoto utilizando sockets, urls, ...

# El paquete java.io

- Este paquete proporciona lo necesario para la gestión de las entradas y salidas de datos de un programa (a través de flujos).
- Está constituido por una serie de interfaces y clases destinadas a definir y controlar el sistema de ficheros, los distintos tipos de flujos y las serializaciones de objetos.

# Ficheros

- La forma de mantener información permanente en computación es utilizar ficheros (archivos).
- Un fichero contiene una cierta información codificada que se almacena en una memoria interna o externa como una secuencia de bits.
- Cada fichero recibe un nombre (posiblemente con una extensión) y se ubica dentro de un directorio que forma parte de una cierta jerarquía.
- El nombre y la ruta, o secuencia de directorios, que hay que atravesar para llegar a la ubicación de un fichero identifican a dicho fichero de forma unívoca.

# La clase **File**

- Representa **caminos abstractos** (independientes del S.O.) dentro de un sistema de ficheros
- Un objeto de esta clase contiene información sobre el nombre y el camino de un fichero o de un directorio.
- Constructores:
  - File(File padre, String fichero)**
  - File(String padre, String fichero)**
  - File(String rutaFichero)**
- Los objetos de esta clase se pueden crear para directorios y ficheros que ya existan o que no existan

# File. Independencia del S.O.

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

```
char separatorChar '\' '/' ':'
```

```
String separator "\" "/" ":"
```

- Separador de un camino de otro

```
char pathSeparatorChar ':' ';'
```

```
String pathSeparator ":" ";"
```

- Si queremos trabajar con \libro\capitulo

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

# File. Métodos de instancia

- Relativos al nombre y la ruta
  - `String getName()`
  - `String getAbsolutePath()`
  - `String getPath()`
  - `String getParent()`
  - `boolean isAbsolute()`
  - `boolean renameTo(File nuevoNombre)`



```

import java.io.*;
public class TestClassFile {
    public static void main(String[] str) {
        try{//Creación de un fichero en Windows
            File f1 = new File("c:\\JMMB\\fichero1.txt");
            f1.createNewFile();
            //Creación de un directorio. Se admite el separador de UNIX
            File d1 = new File("c:/JMMB/carpetal");
            d1.mkdir();
            //Creación de un directorio y de los directorios intermedios
            File d3 = new File("c:\\JMMB\\carpeta2\\carpeta3");
            d3.mkdirs();
            //Creación de un fichero con un directorio File
            File f2 = new File(d3,"fichero2.txt");
            f2.createNewFile();
            //Creación de un fichero a partir de un camino relativo
            File f3 = new File("../\\fichero3.txt");
            f3.createNewFile();
            //Distintas formas del camino de f3
            System.out.println(f3.getPath());
            System.out.println(f3.getAbsolutePath());
            System.out.println(f3.getCanonicalPath());
            System.out.println(f3.getParent());
            System.out.println(f3.getName());
        } catch(IOException e) {
            System.out.println("ERROR: falló la creación del fichero");
        }
    }
}

```

# Operaciones con objetos File

- **Creación** – Para poder almacenar información en un fichero, lo primero que hay que hacer es crearlo. En el proceso de creación se registra la información necesaria para que el sistema pueda localizar el fichero y manipularlo (y otra información adicional)
- **Eliminación** – Esta operación normalmente elimina la entrada correspondiente al fichero en el directorio en el que esté, imposibilitando así su acceso por parte del S.O. .
- Métodos para crear/consultar ficheros y directorios:
  - `boolean createNewFile()`
  - `boolean mkdir()`
  - `boolean mkdirs()`
  - `boolean exists()`
  - `boolean isFile()`
  - `boolean isDirectory()`
- Método para eliminar un fichero:
  - `boolean delete()`

# File. Métodos para directorios

- 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);
```

```
}
```

```
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);  
    }  
}
```

Ej: Listado  
recursivo de  
un directorio

# Operaciones con ficheros

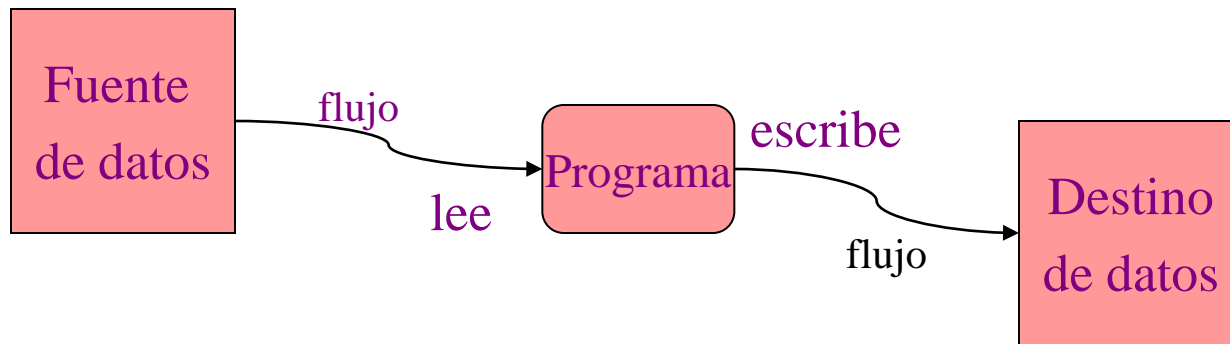
- **Apertura** – En esta operación se localiza e identifica un fichero existente para que se pueda operar con él. La apertura se puede realizar para leer o para escribir.
- **Escritura** – Para poder almacenar información en un fichero, una vez abierto en modo de escritura, hay que transferir la información organizada o segmentada de alguna forma mediante operaciones de escritura.
- **Lectura** – Para poder utilizar la información contenida en un fichero, debe estar abierto en modo de lectura y hay que utilizar las operaciones de lectura adecuadas a la codificación de la información contenida en dicho fichero.
- **Cierre** – Cuando se va a dejar de utilizar un fichero se “cierra” la conexión entre el fichero y el programa. Esta operación se ocupa además de mantener la integridad del fichero, escribiendo previamente la información que se encuentre en algún buffer en espera de pasar al fichero.

# Flujos en Java

- En computación, un flujo de datos o **stream** es una secuencia de datos codificados que se utiliza para transmitir o recibir información.
- Java utiliza la noción de flujo como **objeto** que media entre una fuente o un destino y el programa.
- Java distingue entre:
  - **flujos de entrada y flujos de salida** (s/ sentido de circulación)
  - **flujos de texto y flujos binarios** (s/ codif. 16 u 8 bits)
  - **flujos primarios (iniciales) y flujos secundarios (envoltorios)**

# Entrada/Salida basada en flujos

- Esquema de funcionamiento:



- Fuentes y destinos:
  - array de bytes, (Los flujos siempre conectan con un dispositivo físico )
  - fichero,
  - pipe,
  - conexión de red,
  - consola ...

# Flujos de entrada y de salida

- Flujos binarios (de bytes, *byte streams*)
  - **InputStream** (f. de entrada)
  - **OutputStream** (f. de salida)
- Flujos de texto (de caracteres unicode, *character streams*)
  - **Reader** (f. de entrada)
  - **Writer** (f. de salida)



# Flujos binarios (de bytes)

# InputStream y OutputStream

- Clases abstractas que definen el comportamiento mínimo de estos flujos.

(Sus métodos producen `IOException` cuando hay error)

- Métodos de instancia de **InputStream**:

```
int read()  
int read(byte[] buf)  
int read(byte[] buf, int offset, int count);  
long skip(long n)
```

- 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()
```

# *Streams sobre ficheros*

- **FileInputStream**

`FileInputStream(String name)`

`FileInputStream(File name)`

- **FileOutputStream**

`FileOutputStream(String name)`

`FileOutputStream(String name, boolean  
append)`

`FileOutputStream(File name)`

`FileOutputStream(File name, boolean append)`

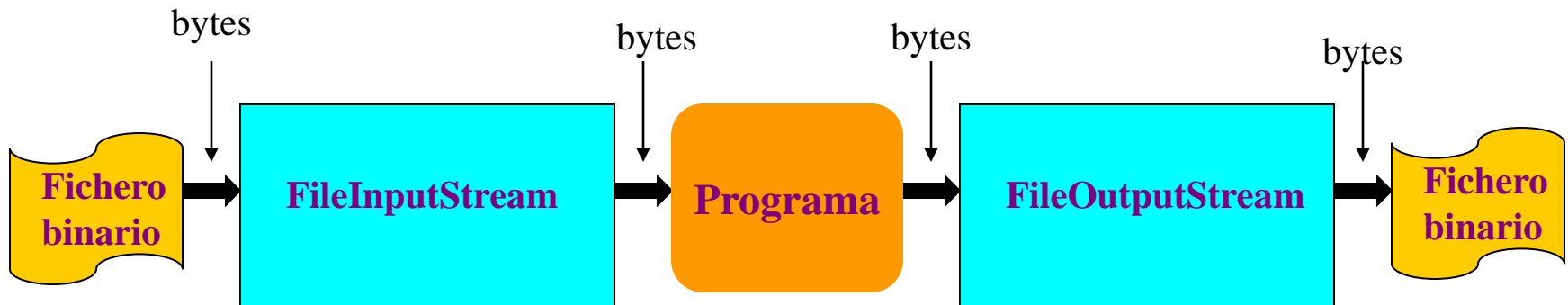
- Los constructores pueden producir  
**FileNotFoundException**

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

# Ej: Copia

```
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: `DataInputStream` `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).

- Constructores

`InputStream ent`  
`DataStream(`

`OutputStream sal)`

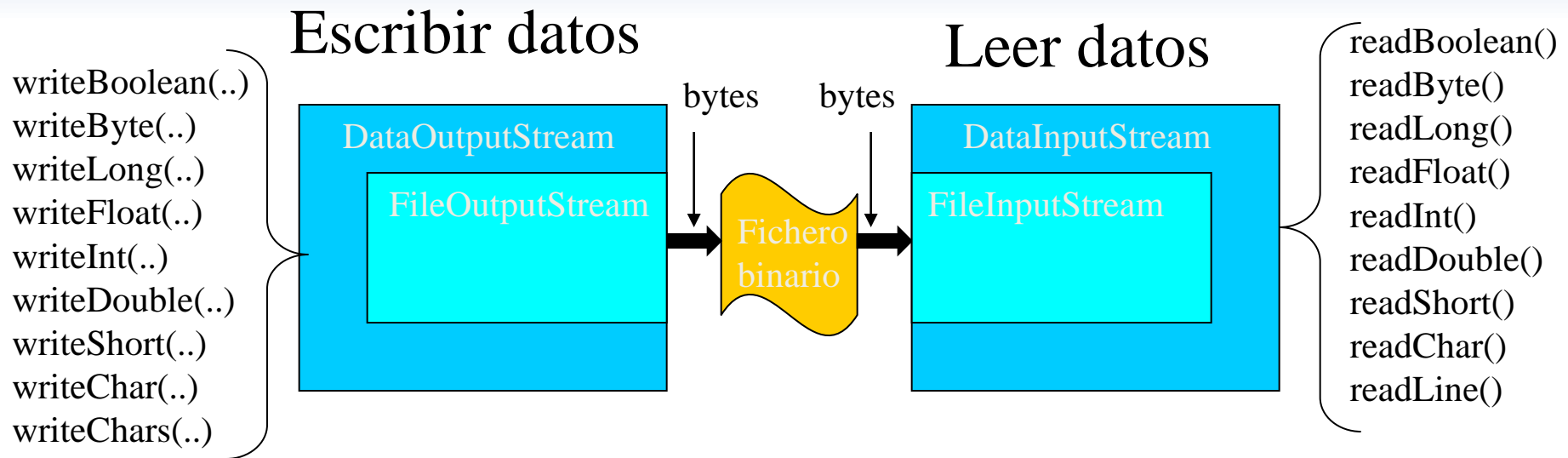
- Métodos de instancia

– 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 Copia2 {
    public static void main(String args[]) throws IOException {
        File d1 = new File("c:\\JMMB");
        d1.mkdirs();
        File f1 = new File("c:\\JMMB\\fichero1.txt");
        f1.createNewFile();
        FileOutputStream fos = new FileOutputStream(f1);
        DataOutputStream dos = new DataOutputStream(fos);
        dos.writeInt(0);
        dos.writeInt(1);
        dos.writeInt(2);
        dos.close();
        //fos = new FileOutputStream(f1);
        fos = new FileOutputStream(f1, true);
        dos = new DataOutputStream(fos);
        dos.writeInt(3);
        dos.writeInt(4);
        dos.writeInt(5);
        dos.close();
        FileInputStream fis = new FileInputStream(f1);
        DataInputStream dis = new DataInputStream(fis);
        int dato;
        while (dis.available() > 0) {
            dato = dis.readInt();
            System.out.print(dato + " ");
        }
    }
}

```



```

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.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);
    }
}

```

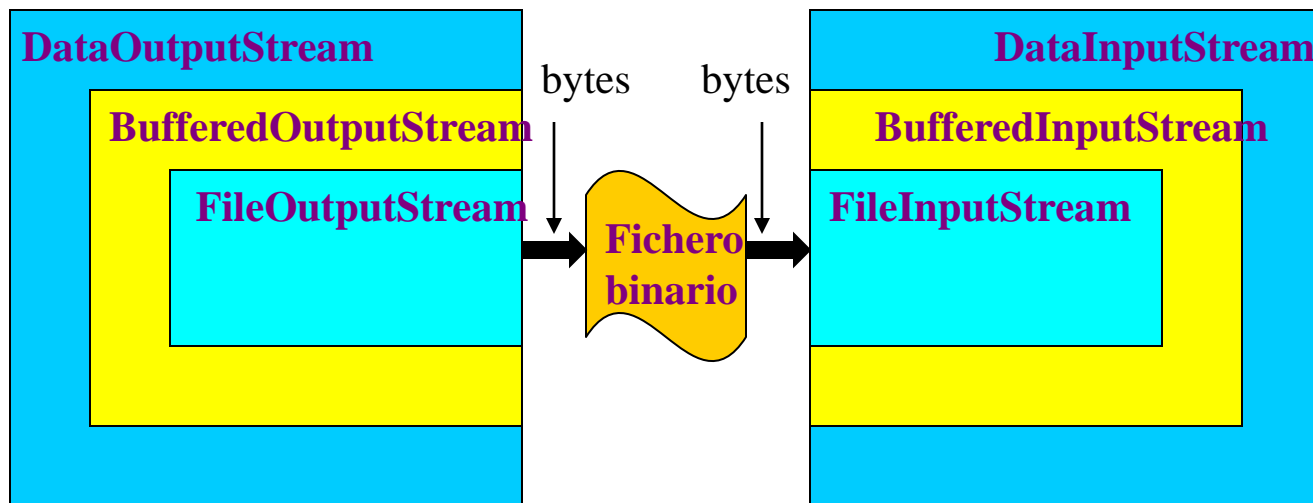
## Ejemplo de DataInputStream y DataOutputStream

# Filtros: `BufferedInputStream` `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 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();
        ...
    }
}
```

# Flujos de texto (caracteres)

# 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);  
long skip(long n)
```

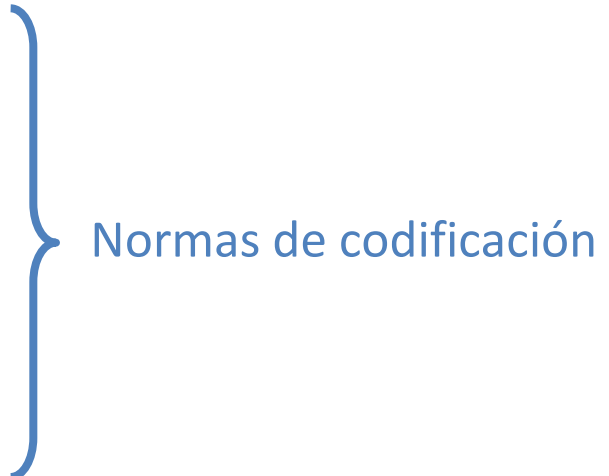
- Métodos de instancia de `Writer`

```
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()
```

- Método de instancia común

```
void close()
```

# 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**
    - **OutputStreamWriter**
  - 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)
- 
- Normas de codificación

# InputStreamReader y OutputStreamWriter

- Establecen el paso entre flujos de bytes y flujos de caracteres.
- Se crean sobre flujos de bytes y decodifican o codifican aplicando alguno de los códigos anteriores.

## 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 de texto: opción 1ª

1) Crear un `FileInputStream`

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

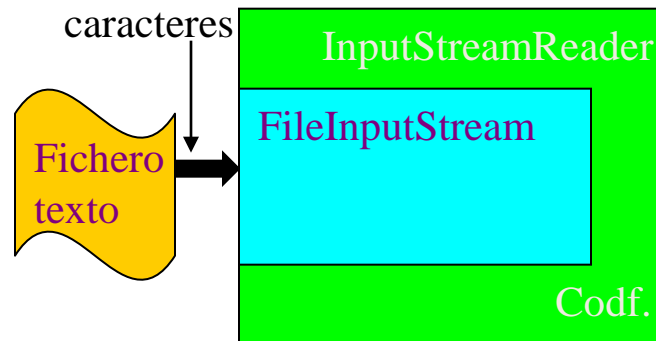
2) Envolverlo en un `InputStreamReader`

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

Aquí podría especificarse un codificador

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

3) Leer con `read()`





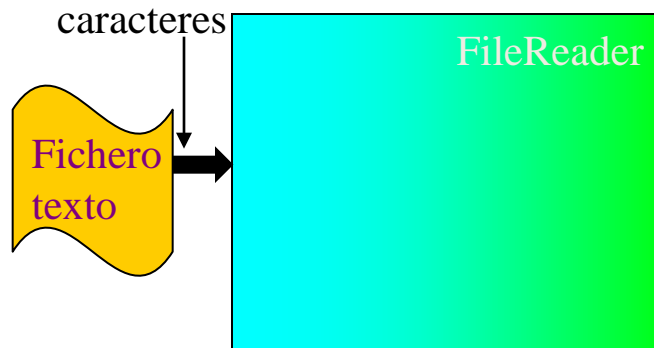
# Lectura de fichero: opción 2ª

- 1) Crear un **FileReader** sobre un **File** o un nombre de fichero  

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

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

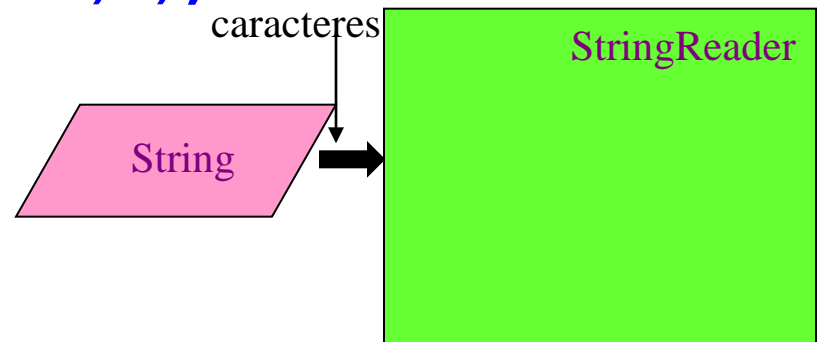
  - Leer con `read( )`



# 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 imprimir representaciones con formato de objetos y tipos básicos de Java sobre flujos de salida de caracteres

```
PrintWriter(File f)
```

```
PrintWriter(String nf)
```

```
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(...)    printf(...)
```

- Sus métodos no lanzan **`IOException`**

# Escritura de fichero de texto: opción 1ª

1. Crear un **FileOutputStream**

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

2. Envolver en un **OutputStreamWriter**

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

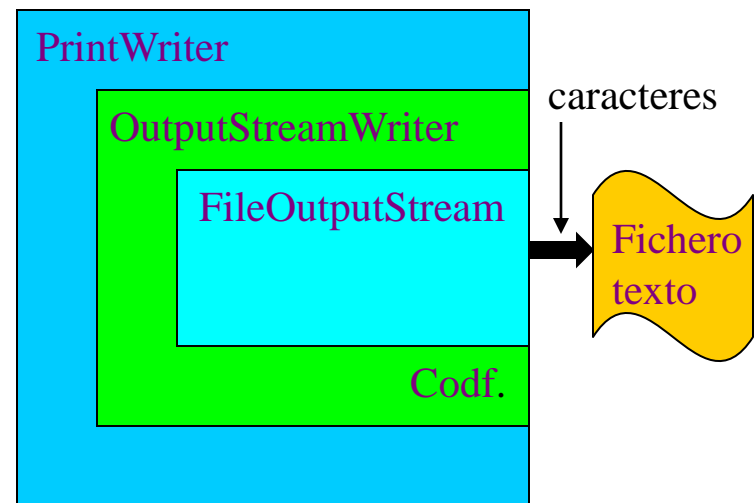
Aquí podría especificarse un codificador

3. Envolver en un **PrintWriter**

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

4. Escribir ...

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



# Escritura de fichero de texto: opción 2ª

## 1) Crear un `FileOutputStream`

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

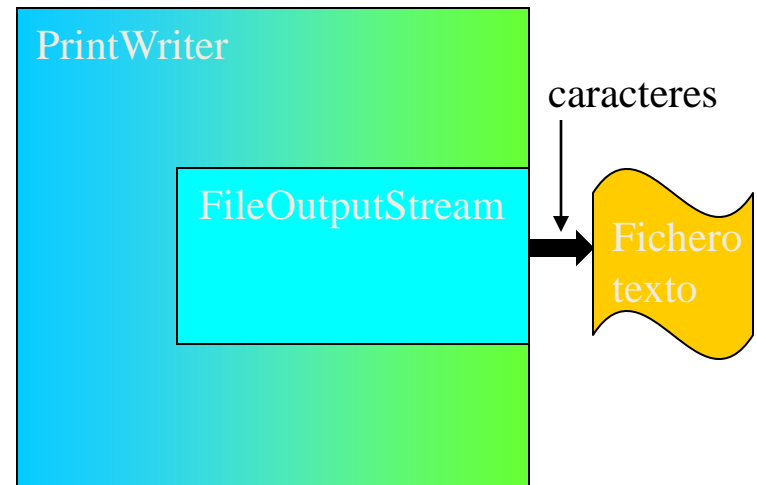
## 2) Crear un `PrintWriter`

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

Automáticamente se añade un `OutputStreamWriter` con decodificación por defecto

## 3) Escribir

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



# Escritura de fichero de texto: opción 3ª

## 1) Crear un `FileWriter`

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

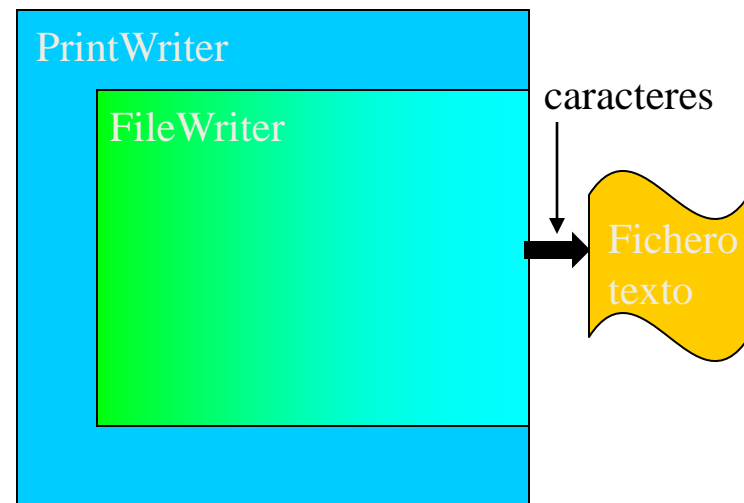
Un `FileWriter` es equivalente a un `FileOutputStream` seguido de un `OutputStreamWriter` con decodificación por defecto

## 2) Crear un `PrintWriter`

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

## 3) Escribir

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



# Escritura de fichero de texto: opción 4ª

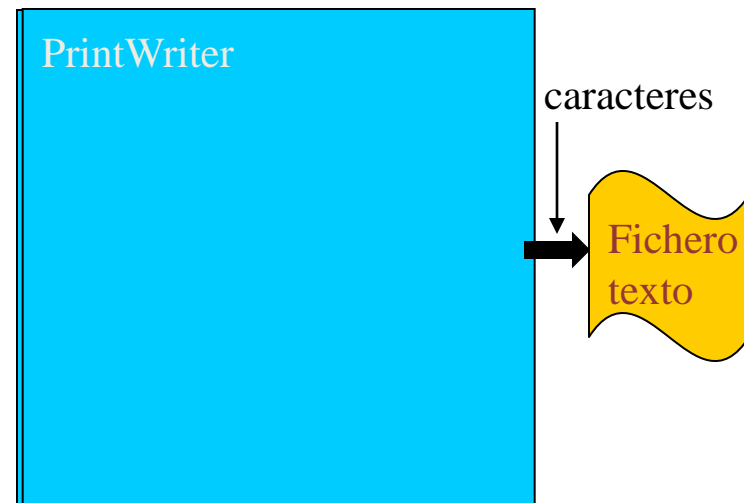
- 1) Crear un `PrintWriter` directamente sobre un fichero

```
PrintWriter pwF = new PrintWriter("datos.tex");
```

Automáticamente se crea un `FileOutputStream` seguido de un `OutputStreamWriter` con decodificación por defecto

- 2) Escribir

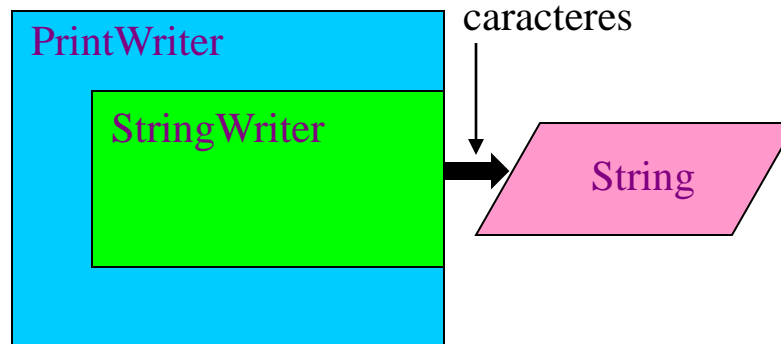
```
pwF.println("Hola a todos");
```



# 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 / 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 / BufferedWriter

- **BufferedReader**

- Métodos de instancia

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

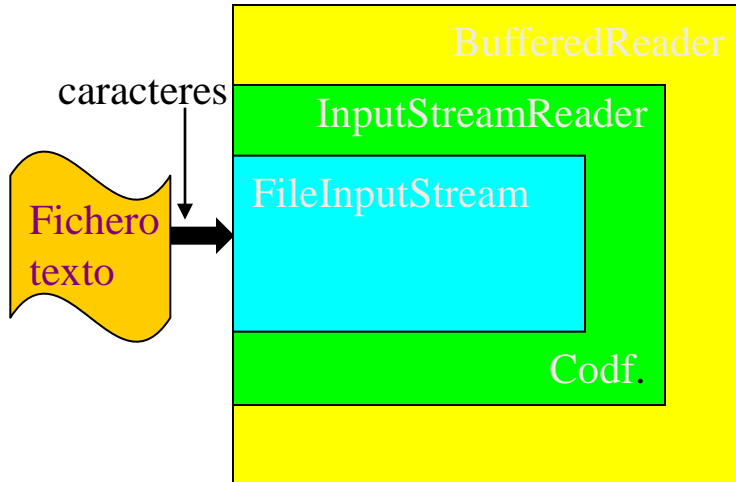
- **BufferedWriter**

- Métodos de instancia

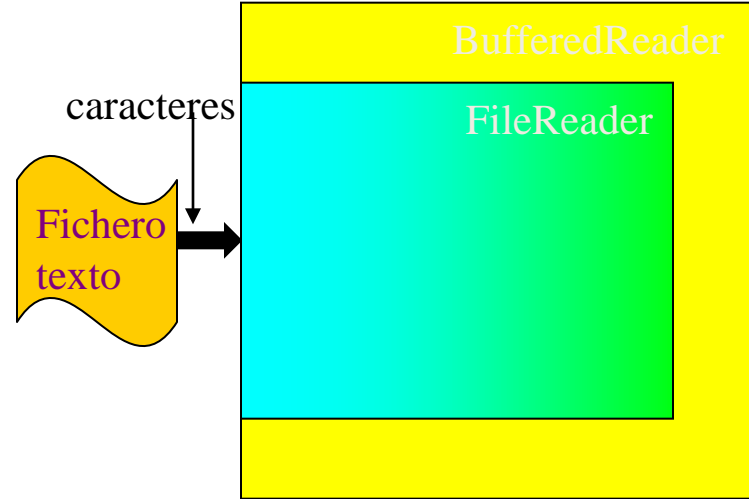
- `String newLine()`
    - `void flush()`
    - `void close()`

# BufferedReader y ficheros

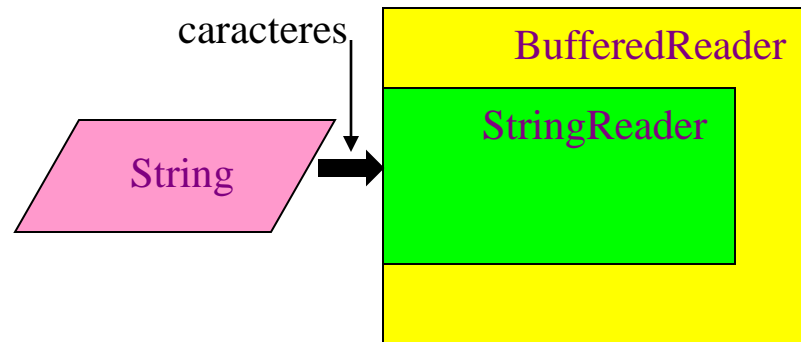
## Opción 1ª



## Opción 2ª

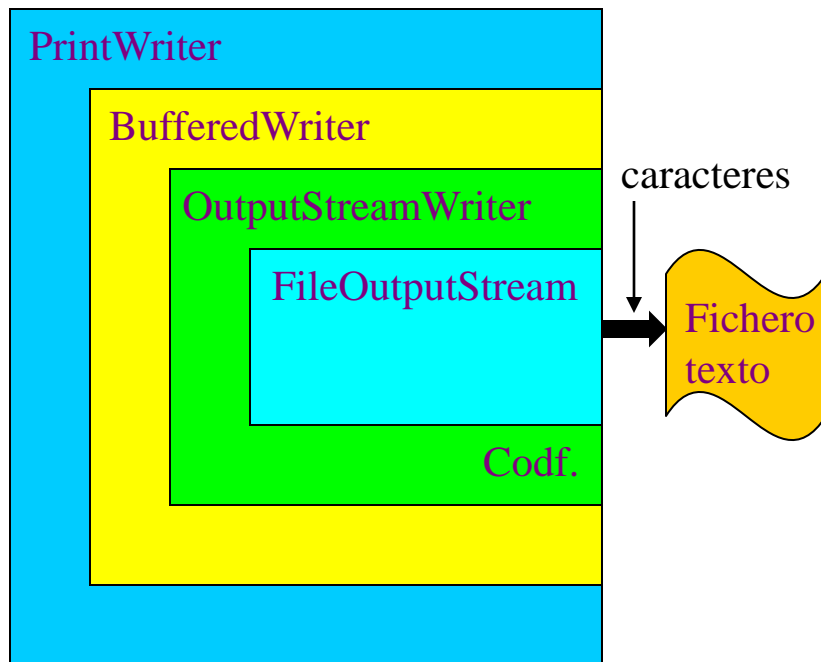


# Lectura con buffer de `String`

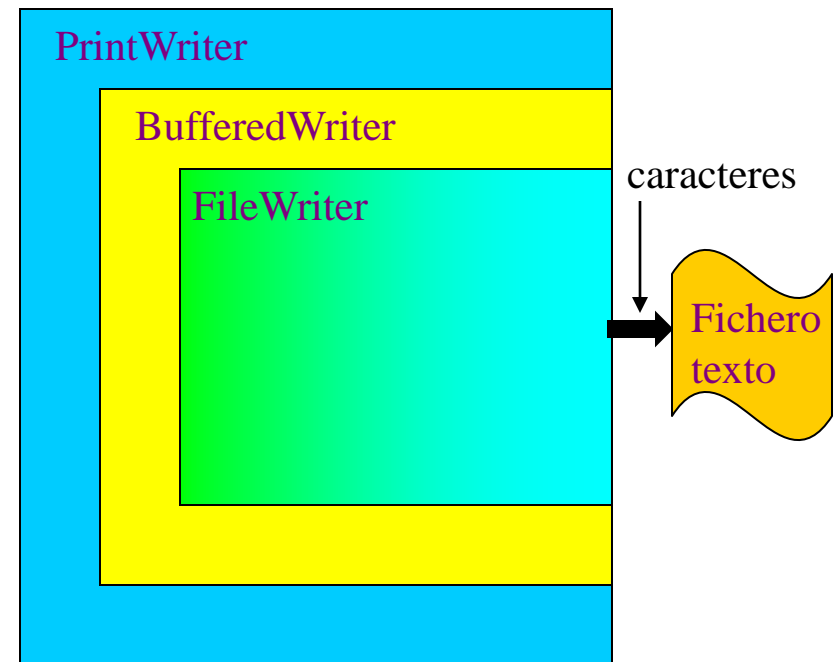


# BufferedWriter y ficheros (I)

Opción 1ª

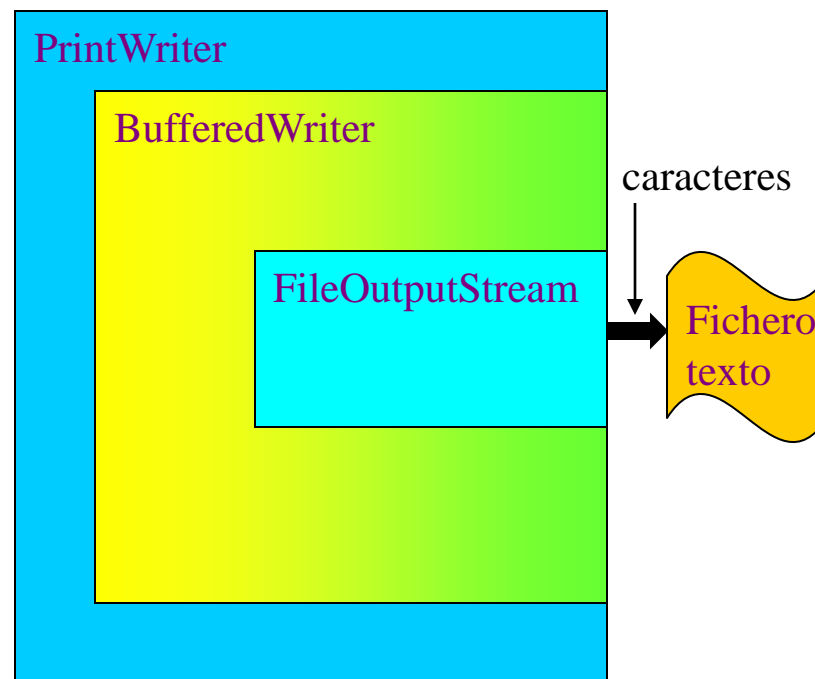


Opción 2ª



# BufferedWriter y ficheros (II)

## Opción 3ª



```

import java.io.*;
public class testBufferedFile {
    public static void main(String[] args) throws IOException {
        if (args.length == 0) {
            System.out.println("ERROR: falta el nombre del fichero");
        } else { // imprimir un fichero de palabras
            PrintWriter pw = new PrintWriter(args[0]);
            pw.println("amor roma mora ramo");
            pw.println("rima mira");
            pw.println("rail liar");
            pw.close();
            // abrir y leer el fichero de palabras
            FileReader fr      = new FileReader(args[0]);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                System.out.println(linea);
                linea = br.readLine();
            }
            br.close();
        }
    }
}

```

```

import java.io.*;
import java.util.StringTokenizer;
public class testBufferedFile {
    public static void main(String [] args) throws IOException {
        if (args.length == 0) {
            System.out.println("ERROR: falta el nombre del fichero");
        } else { // imprimir un fichero de palabras
            PrintWriter pw = new PrintWriter(args[0]);
            pw.println("amor roma mora ramo");
            pw.println("rima mira");
            pw.println("rail liar");
            pw.close();

            // abrir y leer el fichero de palabras
            FileReader fr      = new FileReader(args[0]);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                StringTokenizer st = new StringTokenizer(linea);
                while (st.hasMoreTokens()) {
                    System.out.println(st.nextToken());
                }
                linea = br.readLine();
            }
            br.close();
        }
    }
}

```



# Terminal E/S

- La clase **System** tiene tres variables de clase públicas que se corresponden con la entrada y la salida estándar y la salida de errores

**InputStream in**

**PrintStream out, err**

- Cada programa Java dispone de estos objetos que inicialmente están asociados a la entrada desde la línea de comandos y a la salida hacia la consola
- Estos flujos pueden redirigirse hacia ficheros, sockets, ...o cualquier otro flujo del mismo tipo, mediante los métodos siguientes:

**System.setIn(InputStream in)**

**System.setOut(PrintStream in)**

**System.setErr(PrintStream in)**

# Terminal E/S

- Para leer con buffer desde la entrada estándar:

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

- Para imprimir sobre la salida estándar basta tener en cuenta que la clase **PrintStream** se comporta igual que **PrintWriter** (no provoca **IOException**)

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

```
import java.io.*;

public class TestStandardIO {
    public static void main(String[] args) throws IOException {
        System.out.println("Introduce un número");
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader stdIn = new BufferedReader(isr);
        String entrada = stdIn.readLine();
        int numero = Integer.parseInt(entrada);
        System.out.println("Número leído: " + numero);
        System.out.println("Introduce una palabra: ");
        entrada = stdIn.readLine();
        System.out.println("Palabra leída: " + entrada);
        stdIn.close();
    }
}
```

# La clase Scanner

- Para el análisis sintáctico de cadenas de caracteres.
- Utiliza el espacio en blanco como separador por defecto, y puede utilizar expresiones regulares.
- Puede producir datos primitivos y **String** a partir de las cadenas analizadas.
- Se pueden construir objetos Scanner sobre objetos **String**, **File**, **InputStream** y cualquier **Reader**.
- Métodos de instancia:

```
boolean hasNext(), boolean hasNextXxx(),  
String next(),      xxx nextXxx().
```

```

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

public class TestStandardIO {
    public static void main(String[] args) throws
        IOException {
        ...
        // ----- Uso de la clase Scanner -----
        System.out.println("Introduce tu primer apellido
                           y tu edad: ");
        Scanner sc = new Scanner(System.in);
        String apellido = sc.next();
        int edad = sc.nextInt();
        System.out.println("Datos leídos:");
        System.out.println("Apellido: " + apellido + "\t" +
                           "Edad: " + edad);
    }
}

```