Manejo de Ficheros

Módulo Programación.
Ciclo Desarrollo de Aplicaciones Multiplataforma.

Alberto Carrera Martín
Curso 2018-19

Nota: Adaptación de los materiales del módulo “Acceso a Datos”. Ampliación próximo curso

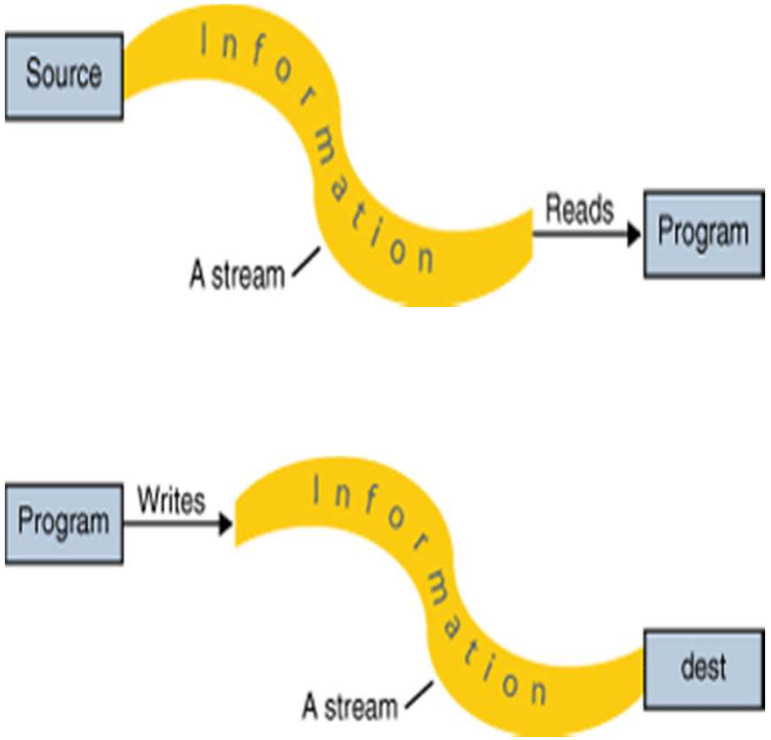
Entrada / Salida estándar

```
import java.io.*;
class CuentaCaracteres {
        public static void main(String args[]) throws IOException {
                int contador=0;
                while(System.in.read()!='\n')
                        contador++;
                System.out.println();
                System.out.println("Tecleados "+contador+"
caracteres.");
        }
}
```

```
import java.io.*;
import java.util.Scanner;
class CuentaCaracteres2 {
    public static void main(String args[]) throws IOException {
        Scanner scanner=new Scanner(System.in);
        System.out.println("Introduzca una cadena: ");
        String teclado = scanner.nextLine();
        System.out.println("Tecleados "+teclado.length()+" caracteres.");
    }
}
```

Streams

- Un stream representa un flujo de información, una secuencia ordenada de datos:
  - procedente de una fuente (teclado, file, memoria, red, etc.) o
  - dirigida a un destino (pantalla, file, etc.)
- Los streams comparten una misma interfaz que hace abstracción de los detalles específicos de cada dispositivo de E/S.
- Todas las clases de streams están en el paquete java.io



Tipos de Streams

Representación de la información

Flujos de bytes

Flujos de caracteres

Propósito

Entrada - (InputStream, Reader)

Salida - (OutputStream, Writer)

Entrada/Salida - (RandomAccessFile) → PRÓXIMO CURSO "Acceso datos"

Acceso

Secuencial

Directo o aleatorio - (RandomAccessFile)→ PRÓXIMO CURSO "Acceso datos"

Por operación

Transferencia de datos

Transformación de los datos (Realizan algún tipo de procesamiento sobre los datos (p.e. buffering, conversiones, filtrados)

Streams

- Define la entrada/salida en términos de ***streams***
- Un *stream* es una secuencia ordenada de datos
- Tienen:
  - una *fuente* = streams de entrada o
  - un *destino* = streams de salida
- El paquete `java.io` tiene dos partes principales:
  - Stream de caracteres (caracteres Unicode de 16 bits)
  - Stream de bytes (8 bits)

Entrada/Salida. Paquete java.io

- E/S puede estar **basada**:
  - **En texto**: *streams* de caracteres legibles

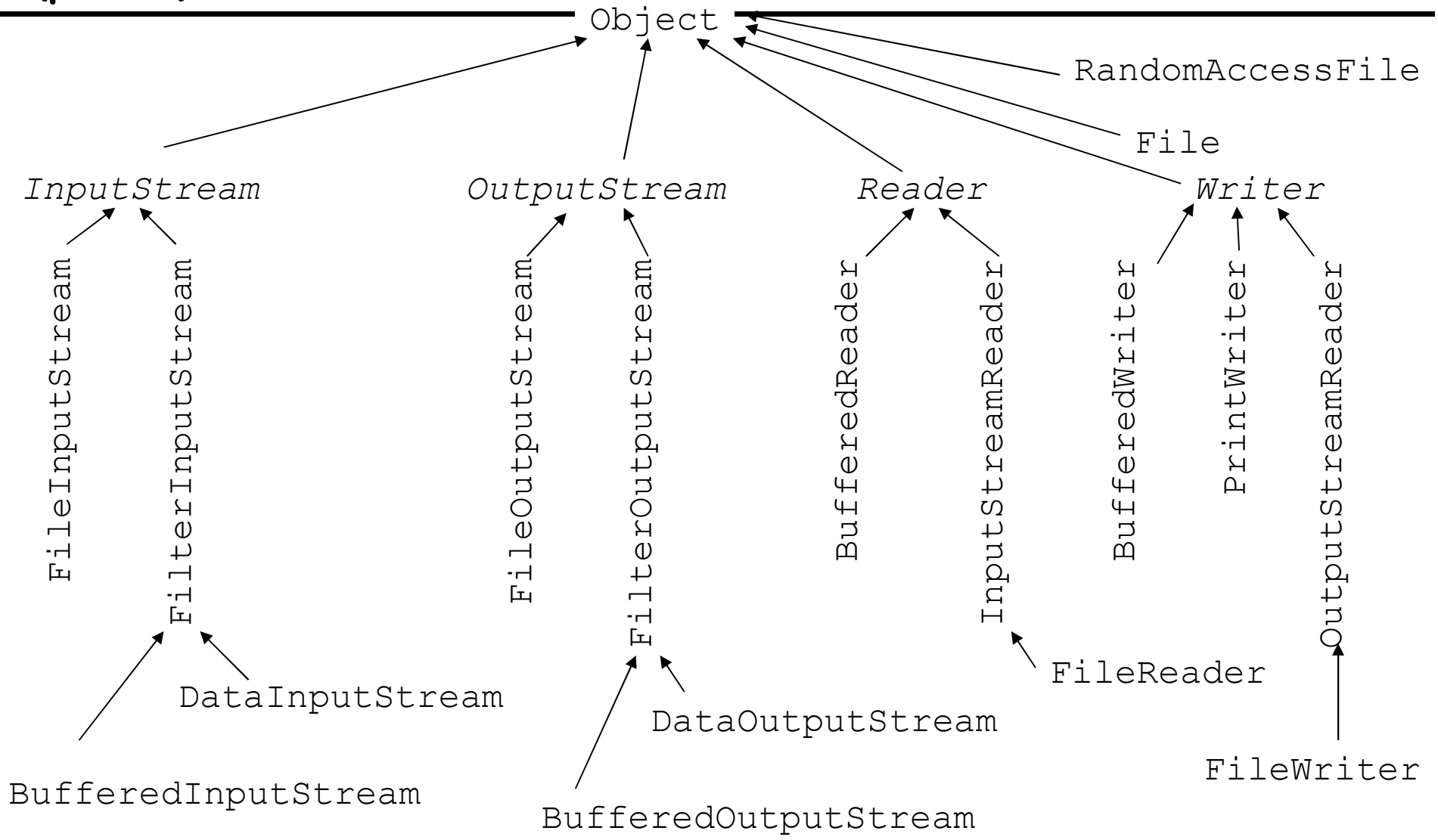
    Ejemplo: el código fuente de un programa
  - **En datos**: *streams* de datos binarios

    Ejemplo: patrón de bits de una imagen
- Los ***streams* de caracteres** se utilizan en la E/S basada en texto.
  - Se denominan **lectores** (*reader*) y **escritores** (*writer*)
- Los ***streams* de bytes** se utilizan en la E/S basada en datos.
  - Se denominan **streams de entrada** y **streams de salida**

Clases principales de java.io

- **Clases de flujo de entrada:**
  - Se utilizan para leer datos de una fuente de entrada (archivo, cadena o memoria)
  - Flujo de bytes: **`InputStream`**, **`BufferedInputStream`**, **`DataInputStream`**, **`FileInputStream`**
  - Flujo de caracteres: **`Reader`**, **`BufferReader`**, **`FileReader`**
- **Clases de flujo de salida:**
  - Son las homólogas a las clases de flujo de entrada y se utilizan para enviar flujos de datos a dispositivos de salida
  - Flujo de bytes: **`OutputStream`**, **`PrintStream`**, **`BufferedOutputStream`**, **`DataOutputStream`** y **`FileOutputStream`**
  - Flujo de caracteres : **`Writer`**, **`PrintWriter`**, **`FileWriter`**
- **Clases de archivo:**
  - **`File`** y **`RandomAccesFile`** (mayor control sobre los archivos)

Jerarquía de clases de `java.io`
(parcial)



InputStream

Método	Descripción
`int read()`	Lee el siguiente byte del flujo de entrada y lo **devuelve** como **un entero**. Cuando alcanza el final del flujo de datos, devuelve -1. EOF
`int read(byte b[])`	Lee hasta b.length bytes y los almacena en la matriz b. Devuelve el número de bytes leídos o -1 cuando se alcanza el final del flujo de datos.
`int read(byte b[], int off, int cantidad)`	Lee hasta cantidad bytes de datos del flujo de entrada, empezando desde la posición indicada off, y los almacena en matriz. b. Devuelve el nº de bytes leído.
`available()`	Devuelve el número de bytes que se pueden leer de un flujo de entrada sin que se produzca un bloqueo por causa de una llamada a otro método que utiliza el mismo flujo de entrada.
`skip(long n)`	Omite la lectura de n bytes de datos de un flujo de entrada y los descarta.
`close()`	Cierra un flujo de entrada y libera los recursos del sistema utilizados por el flujo de datos.

OutputStream

Método	Descripción
`void write(int b)`	Escribe b en un flujo de datos de salida.
`void write(byte b[])`	Escribe b.length bytes de la matriz b en un flujo de datos de salida.
`void write(byte b[], int off, int cantidad)`	Escribe cantidad de bytes de la matriz b en el flujo de datos de salida, empezando en la posición dada por el desplazamiento off
`flush()`	Vacía el flujo de datos y fuerza la salida de cualquier dato almacenado en el búfer.
`close()`	Cierra el flujo de datos de salida y libera cualquier recurso del sistema asociado con él.

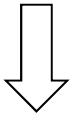
Streams sobre Ficheros

- **`FileInputStream`**: muy similar a la clase `InputStream`, sólo que está diseñada para leer archivos.
  - `FileInputStream(String name)`
  - `FileInputStream(File name)`

- **`FileOutputStream`**: muy similar a la clase `OutputStream`, sólo que está diseñada para escribir en archivos.
  - `FileOutputStream(String name)`
  - `FileOutputStream(String name, boolean append)`
    - Si `append==true` queremos añadir al final del fichero
  - `FileOutputStream(File name)`

Esquema de trabajo con streams

Entrada de datos (leer datos)	Salida de datos (escribir datos)
1. Se crea un objeto flujo de datos de lectura 2. Se leen datos de él con los métodos apropiados 3. Se cierra el flujo de datos	1. Se crea un objeto flujo de datos de escritura 2. Se escriben datos de él con los métodos apropiados 3. Se cierra el flujo de datos



Entrada de datos (leer datos)	Salida de datos (escribir datos)
1. Abrir el flujo del archivo 2. Mientras queden datos leer el siguiente dato 3. Se cierra el flujo de datos	1. Abrir el flujo del archivo 2. Mientras haya datos por escribir escribir en el archivo 3. Se cierra el flujo de datos

Ejemplo lectura byte

```
import java.io.*;
public class EjemploLecturaByte {
    public static void main(String[] args) {
        FileInputStream fis = null;
        int aux = 0;
        try {
            fis = new FileInputStream("C:\\eclipse\\readme\\readme_eclipse.html");
            while((aux = fis.read()) != -1)
                System.out.println(aux + " - " +  (char) aux);
        } catch(FileNotFoundException ex) {
                ex.printStackTrace();
        } catch(IOException ex) {
                ex.printStackTrace();
        } finally {
            try {
                fis.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo escritura byte

```
import java.io.*;
public class EjemploEscrituraChar {
    public static void main(String[] args)
    {
        FileOutputStream fos = null;
        try{
            fos = new FileOutputStream("D:\\Prueba.txt");
            fos.write(67); fos.write(73); fos.write(67); fos.write(76); fos.write(79);
            fos.write(32); fos.write(68);fos.write(65); fos.write(77);
        } catch(FileNotFoundException ex) {
            ex.printStackTrace();
        } catch(IOException ex) {
            ex. printStackTrace();
        } finally {
            try {
                fos.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: streams sobre ficheros

```
import java.io.*;
public class Copia {
  public static void main(String args[]) {
    FileInputStream origen = null;
    FileOutputStream destino = null;
    try {
      origen = new FileInputStream(args[0]);
      destino = new FileOutputStream(args[1],true); //añadir
      int i = origen.read();
      while (i != -1) { // mientras not EOF
        destino.write(i);
        i = origen.read();
      }
     origen.close(); destino.close();
    }catch (IOException e) {
       System.out.println("Error de ficheros”);
     }
   }
}
```

FileReader

Método	Descripción
`int read()`	Lee el siguiente carácter y lo **devuelve** como **un entero**. Cuando alcanza el final del flujo de datos, devuelve -1 **EOF**
`int read(char b[])`	Lee hasta b.length caracteres y los almacena en la matriz b. Devuelve el número de bytes leídos o -1 cuando se alcanza el final del flujo de datos.

`int read(char b[],`
`int off,`
`int cantidad)`
	Lee hasta cantidad de caracteres de datos del flujo de entrada, empezando desde la posición indicada off, y los almacena en matriz. b. Devuelve el número de caracteres leído…

….	Similar a InputStream

FileWriter

Método	Descripción
`void write(int b)`	Escribe un carácter
`void write(char b[])`	Escribe array de caracteres b en un flujo de datos de salida.
`void write(char b[], int off, int cantidad)`	Escribe cantidad de caracteres de la matriz b en el flujo de datos de salida, empezando en la posición dada por el desplazamiento off
`void write(String str)`	Escribe una cadena de caracteres
`append(char c)`	Añade un carácter a un fichero
`close()`	Cierra el flujo de datos de salida y libera cualquier recurso del sistema asociado con él.

Ejemplo lectura char

```
import java.io.*;
public class EjemploLecturaChar {
    public static void main(String[] args){
        FileReader fr = null;
        int aux = 0;
        try{
            fr = new FileReader("C:\\eclipse\\readme\\readme_eclipse.html");
            while((aux = fr.read()) != -1)
                System.out.println((char)aux);
        } catch(FileNotFoundException ex) {
                ex.printStackTrace();
        } catch(IOException ex) {
            ex.printStackTrace();
        } finally {
                try {
                    fr.close();
                } catch(IOException e) {
                    e.printStackTrace();
                }
        }
    }
}
```

Ejemplo escritura char

```
import java.io.*;
public class EjemploEscrituraByte {
    public static void main(String[] args)
    {
        FileWriter fw = null;
        try {
            fw = new FileWriter("D:\\Prueba.txt");
            fw.write('A'); fw.write('l');fw.write('b');fw.write('e');fw.write('r');fw.write('t');
            fw.write('o');fw.write(' ');fw.write('C');fw.write('a');fw.write('r');fw.write('r');
            fw.write('e');fw.write('r');fw.write('a');
        } catch(FileNotFoundException ex) {
            ex.printStackTrace();
        }
        catch(IOException ex) {
            ex. printStackTrace();
        } finally {
            try {
                fw.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

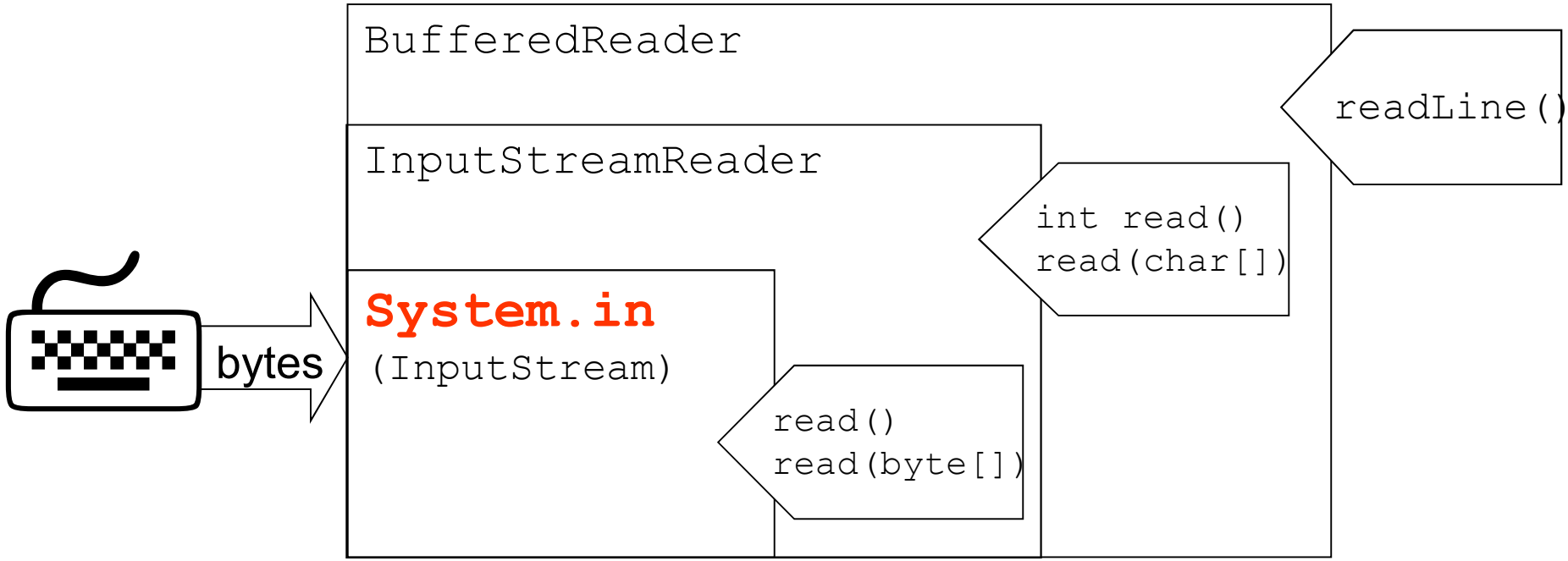
Reader y Writer

- Dar soporte Unicode en todas las operaciones de E/S.
- En ocasiones hay que combinar streams de caracteres y de bytes:
  - **`InputStreamReader`**: convierte un `InputStream` en un `Reader`
  - **`OutputStreamWriter`**: convierte un `OutputStream` en un `Writer`
- Casi todas las clases de la jerarquía de streams de bytes tienen su correspondiente clase `Reader` o `Writer` con interfaces casi idénticas.
- **`BufferedReader`** y **`BufferedWriter`**: almacenamiento temporal en un *buffer*, para no actuar directamente sobre el stream.
- Igual que los streams de bytes se deben cerrar explícitamente para liberar sus recursos asociados (`close`).

Filtros

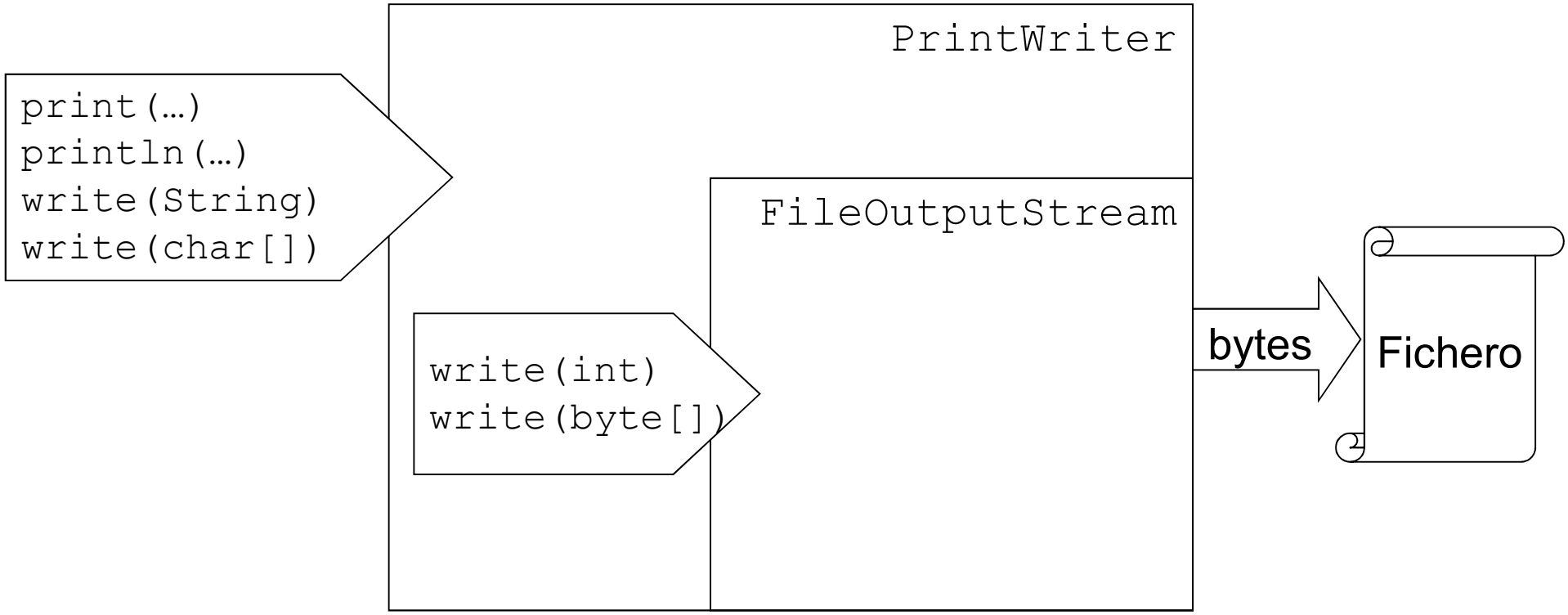
```
import java.io.*;
public class EjemploFiltro {
    public static void main(String[] args) {
        String nombreArchivo = "D:\\prueba.txt";
        FileReader fr;
        BufferedReader filtro;
        String linea;
        try {
            fr = new FileReader(nombreArchivo);
            filtro=new BufferedReader(fr);
            linea=filtro.readLine();
            while (linea !=null) {
                System.out.println(linea);
                linea=filtro.readLine();
            }
            filtro.close();
            fr.close();
        } catch (IOException e) {
            System.out.println("No se puede abrir el archivo para lectura");
    }
  }
}
```

Ejemplo lectura del teclado




```
BufferedReader teclado = new BufferedReader(new
                                 InputStreamReader(System.in));
String entrada = teclado.readLine();
```

Escribir en fichero



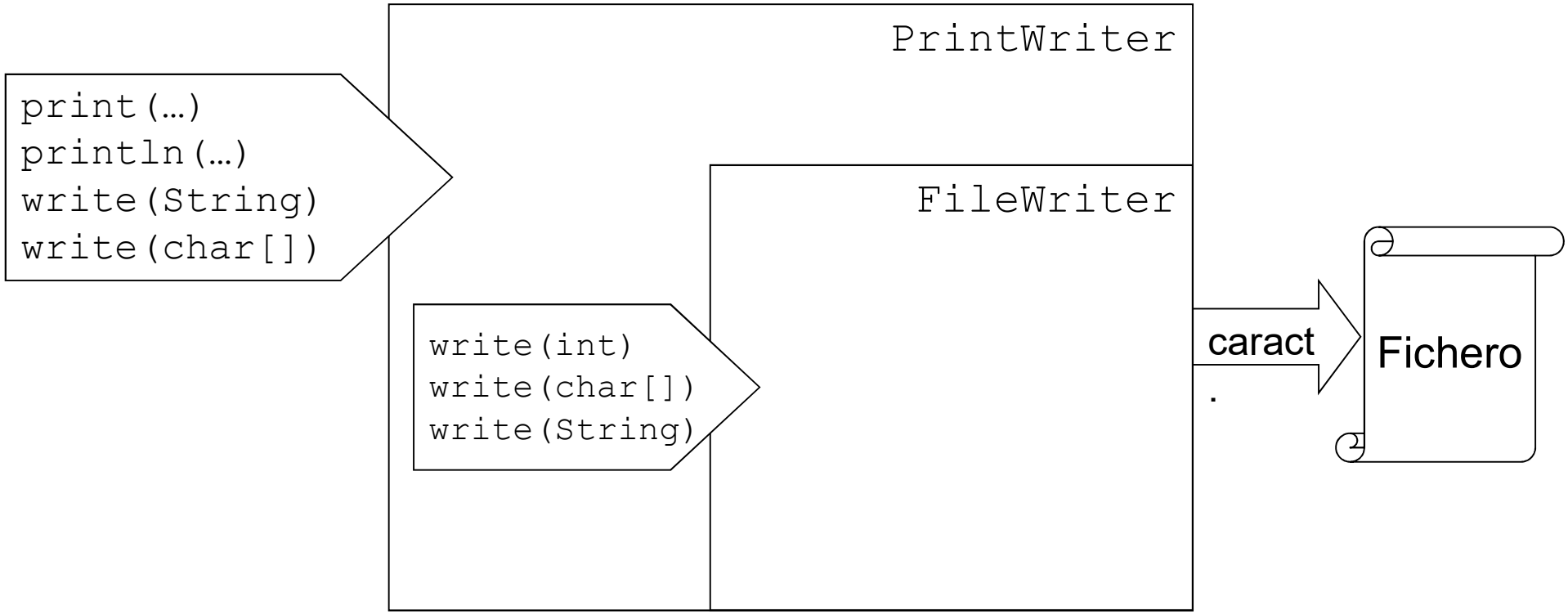

```
FileOutPutStream fos = new FileOutputStream("fichero.txt");
PrintWriter pr = new PrintWriter(fos);
…
pr.println("Escribimos texto");
```

Ejemplo: Escribir en un fichero

```
public class EscribirFichero {
  public static void main(String[] args) {
    try{
      FileOutputStream fos = new FileOutputStream("salida.txt");
      PrintWriter pw = new PrintWriter(fos);
      pw.println("Imprimimos una cadena y un entero " + 5);
      pw.flush();
      pw.close();
      fos.close();
    }catch (FileNotFoundException e){ }
     catch (IOException e2){ }
  }
}
```

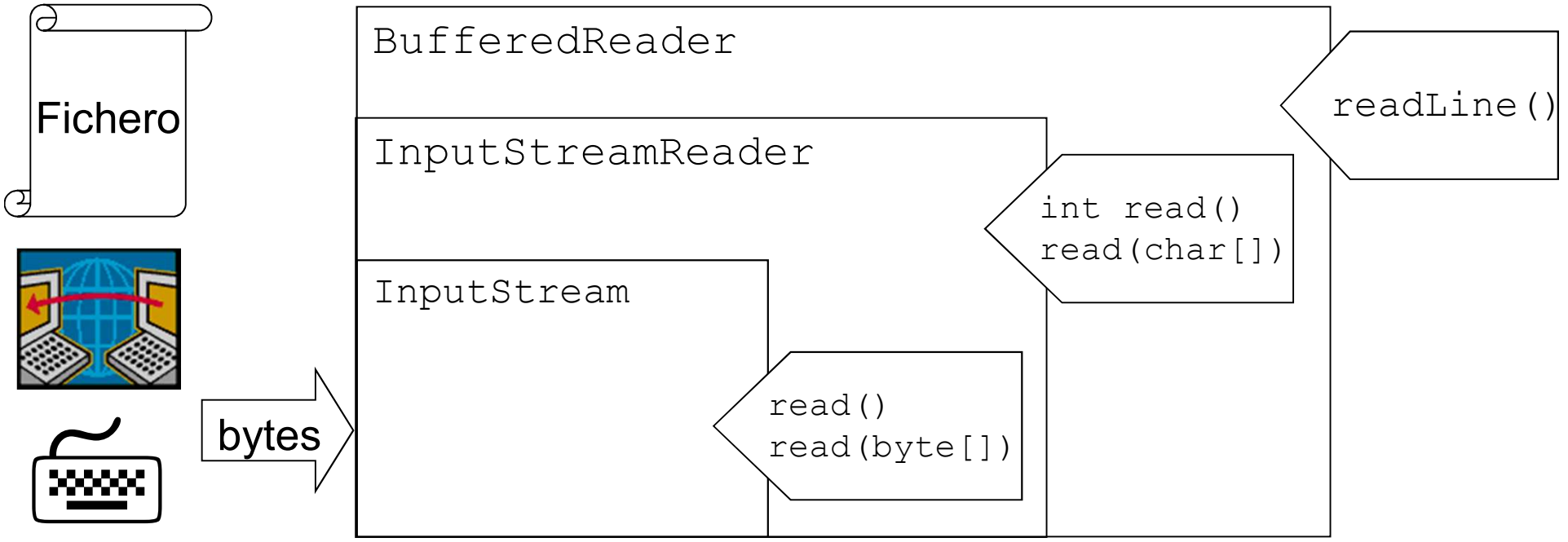
NOTA: La clase PrintWriter, que también deriva de Writer, posee los métodos print(String) y println(String), idénticos a los de System.out. Recibe un String y lo imprimen en un fichero, el segundo con salto de línea

Escribir en fichero (2)



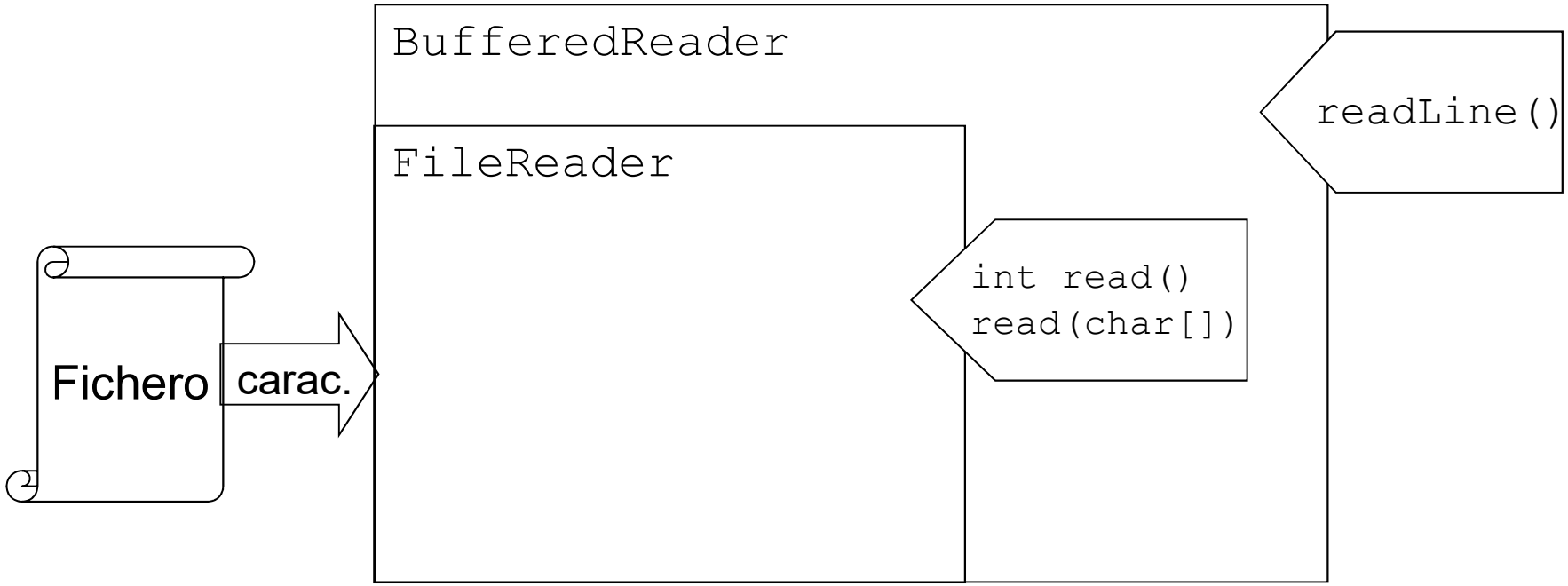

```
FileWriter fw = new FileWriter("fichero.txt");
PrintWriter pr = new PrintWriter(fos);
…
pr.println("Escribimos texto");
```

Lectura de líneas




```
FileInputStream fis = new FileInputStream("fichero.txt");
InputStreamReader isr = new InputStreamReader(fis);
BufferedReader buffer = new BufferedReader(isr);
…
String linea = buffer.readLine();
```

Leer de fichero



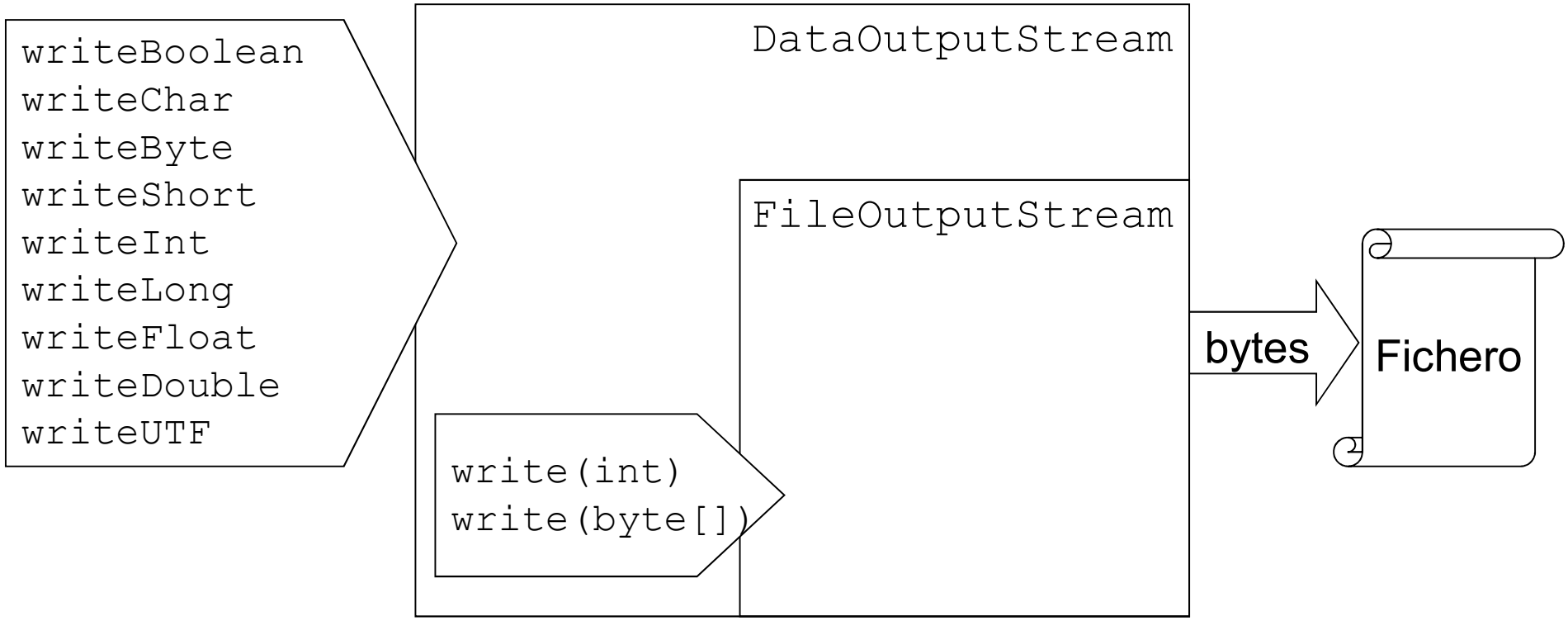

```
FileReader fr = new FileReader("fichero.txt");
BufferedReader buffer = new BufferedReader(fr);
…
String linea = buffer.readLine();
```

Clases `DataInputStream` y `DataOutputStream`

- Permiten transmitir tipos primitivos por un stream.

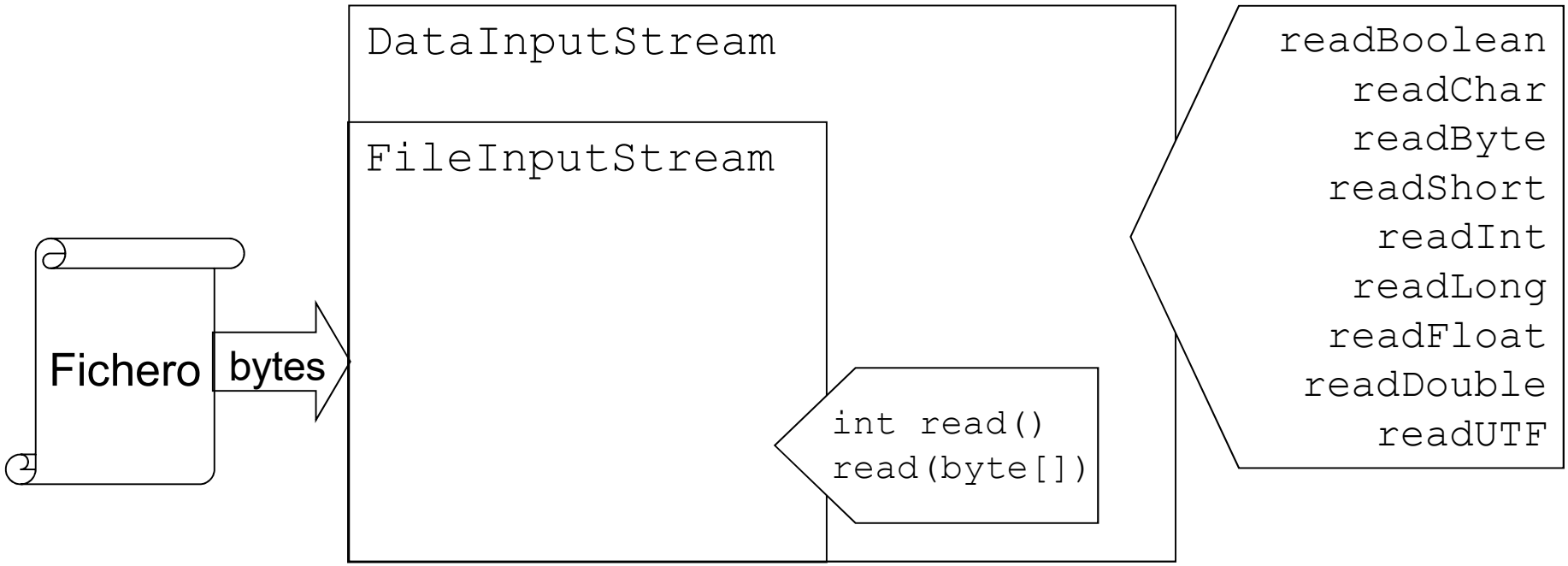
Lectura	Escritura	Tipo
`readBoolean`	`writeBoolean`	`boolean`
`readChar`	`writeChar`	`char`
`readByte`	`writeByte`	`byte`
`readShort`	`writeShort`	`short`
`readInt`	`writeInt`	`int`
`readLong`	`writeLong`	`long`
`readFloat`	`writeFloat`	`float`
`readDouble`	`writeDouble`	`double`
`readUTF`	`writeUTF`	`String`

Escritura en modo datos




```
FileOutputStream fos = new FileOutputStream("salida.dat");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeInt(5);
```

Lectura en modo datos




```
FileInputStream fis = new FileInputStream("salida.dat");
DataInputStream dis = new DataInputStream(fis);
int entero = dis.readInt();
```