ESTRUCTURAS DE CONTROL

Manuel J. Molino Milla Luis Molina Garzón

IES Virgen del Carmen

Departamento de Informática

12 de abril de 2023

Logo

Figura: Logo Java

Contenido

Contenido

Contenido

Contenido

Introducción

JDK 1.0 Introduce el paquete *java.io*, I/O basada en stream

Introducción

JDK 1.0 Introduce el paquete *java.io*, I/O basada en stream

JDK 1.4 Introduce el paquete *java.nio*, I/O basada en buffer

Introducción

JDK 1.0 Introduce el paquete *java.io*, I/O basada en stream

JDK 1.4 Introduce el paquete *java.nio*, I/O basada en buffer

JDK 1.5 Introduce I/O de texto formateado con nuevas clases *Scanner*, *Formatter* o *printf*

Introducción

JDK 1.0 Introduce el paquete *java.io*, I/O basada en stream

JDK 1.4 Introduce el paquete *java.nio*, I/O basada en buffer

JDK 1.5 Introduce I/O de texto formateado con nuevas clases *Scanner*, *Formatter* o *printf*

JDK 1.7 Introduce *NIO.2* con I/O no bloqueante.

Introducción

JDK 1.0	Introduce el paquete *java.io*, I/O basada en stream
JDK 1.4	Introduce el paquete *java.nio*, I/O basada en buffer
JDK 1.5	Introduce I/O de texto formateado con nuevas clases *Scanner*, *Formatter* o *printf*
JDK 1.7	Introduce *NIO.2* con I/O no bloqueante.

Introducción

JDK 1.0	Introduce el paquete *java.io*, I/O basada en stream
JDK 1.4	Introduce el paquete *java.nio*, I/O basada en buffer
JDK 1.5	Introduce I/O de texto formateado con nuevas clases *Scanner*, *Formatter* o *printf*
JDK 1.7	Introduce *NIO.2* con I/O no bloqueante.

Stream o flujos

- Canal de comunicación en las operaciones I/O

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.
  3. Lectura de un fichero.

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.
  3. Lectura de un fichero.
  4. Escritura en un fichero.

Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.
  3. Lectura de un fichero.
  4. Escritura en un fichero.
  5. Evío y recepción de datos por red.

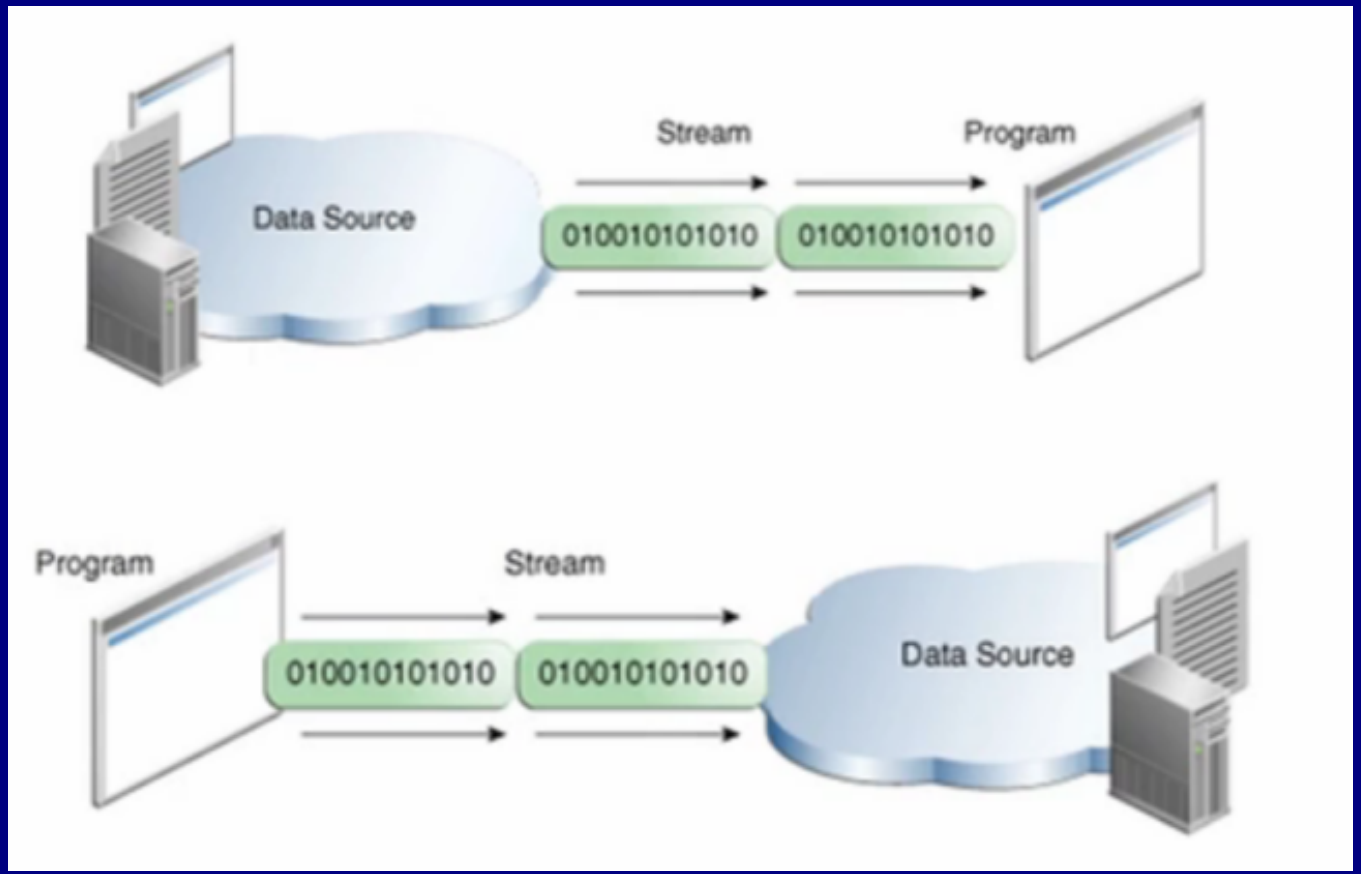
Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.
  3. Lectura de un fichero.
  4. Escritura en un fichero.
  5. Evío y recepción de datos por red.

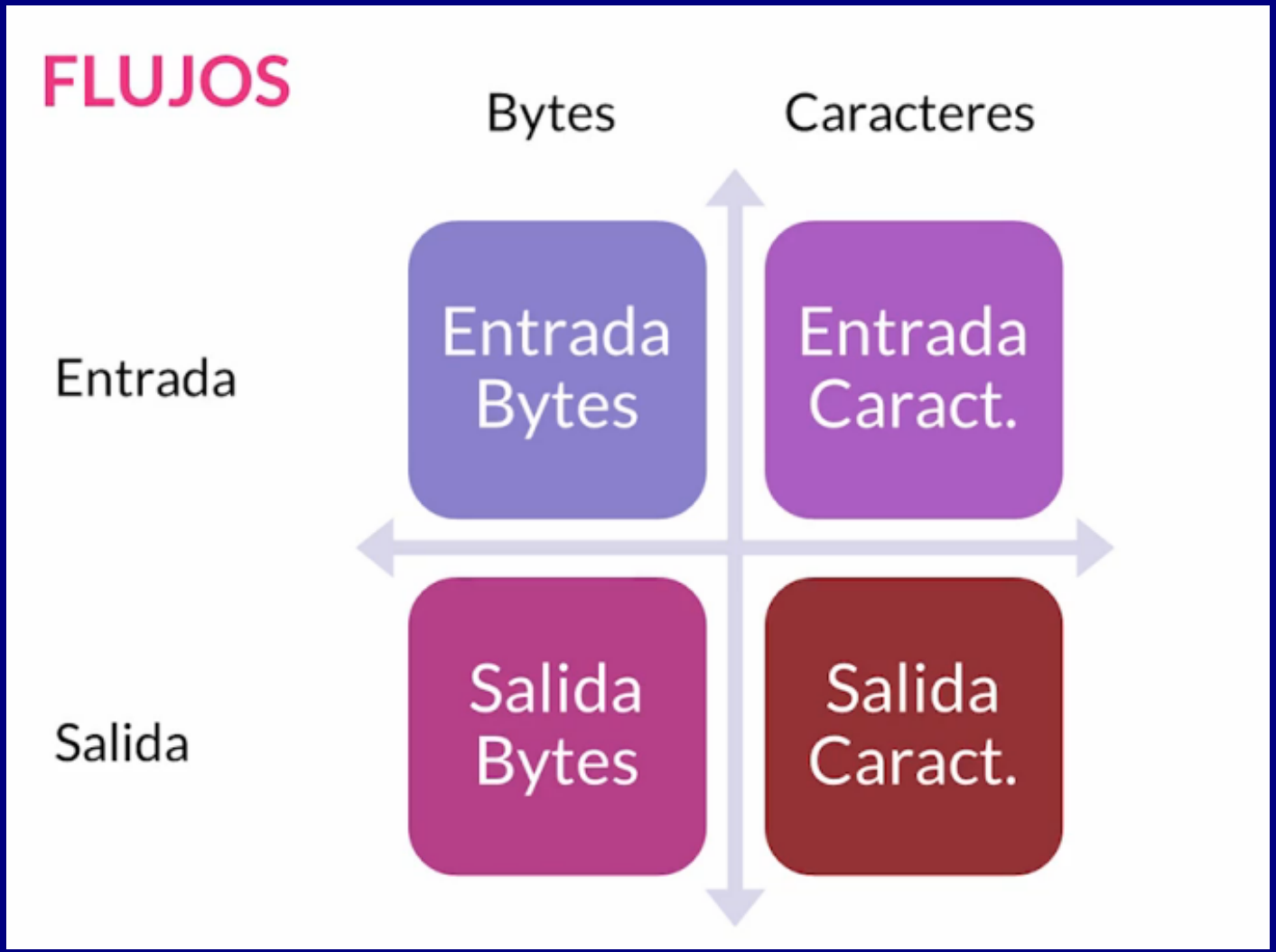
Stream o flujos

- Canal de comunicación en las operaciones I/O
- Tenemos flujos de entrada y flujos de salida.
- Nos da independencia de la procedencia de los datos (bits)
  1. Entrada desde teclado.
  2. Salida a monitor.
  3. Lectura de un fichero.
  4. Escritura en un fichero.
  5. Evío y recepción de datos por red.

Flujos I/O


Stream
Program
Data Source
010010101010
010010101010
Program
Stream
010010101010
010010101010
Data Source

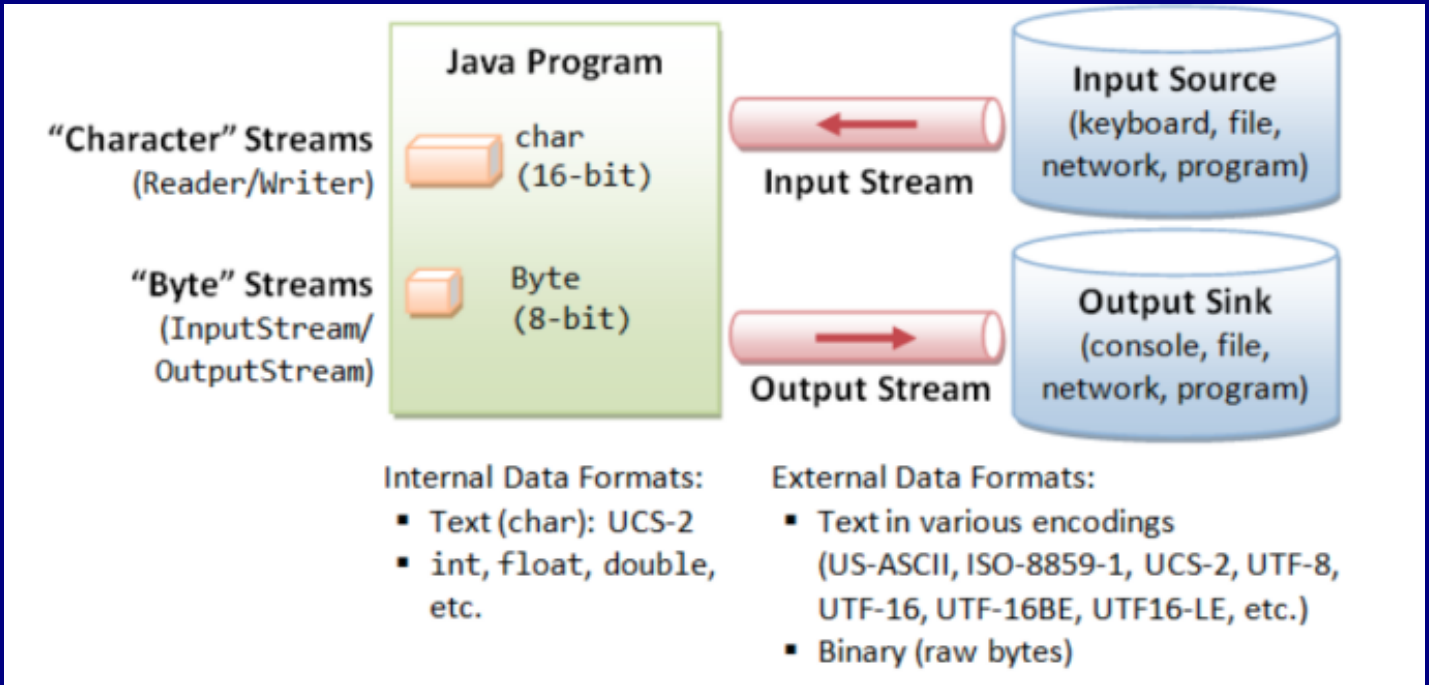
Flujos I/O


FLUJOS
Bytes
Caracteres
Entrada
Salida
Entrada Bytes
Entrada Caract.
Salida Bytes
Salida Caract.

Caracteres stream y byte stream

Java almacena de forma interna los caracteres usando 16-bit UCS-2. Pero la fuente de datos puede almacenar usando codificaciones diferentes como US-ASCII, ISO-8859-x, UTF-8, UTF-16, ...

Java necesita diferenciar entre I/O basada en bytes: procesamiento I/O raw bytes o binary data y en caracteres usando dos bytes.



Flujos de salida

La forma de trabajar es la siguiente:

- Abrir el flujo.

Flujos de salida

La forma de trabajar es la siguiente:

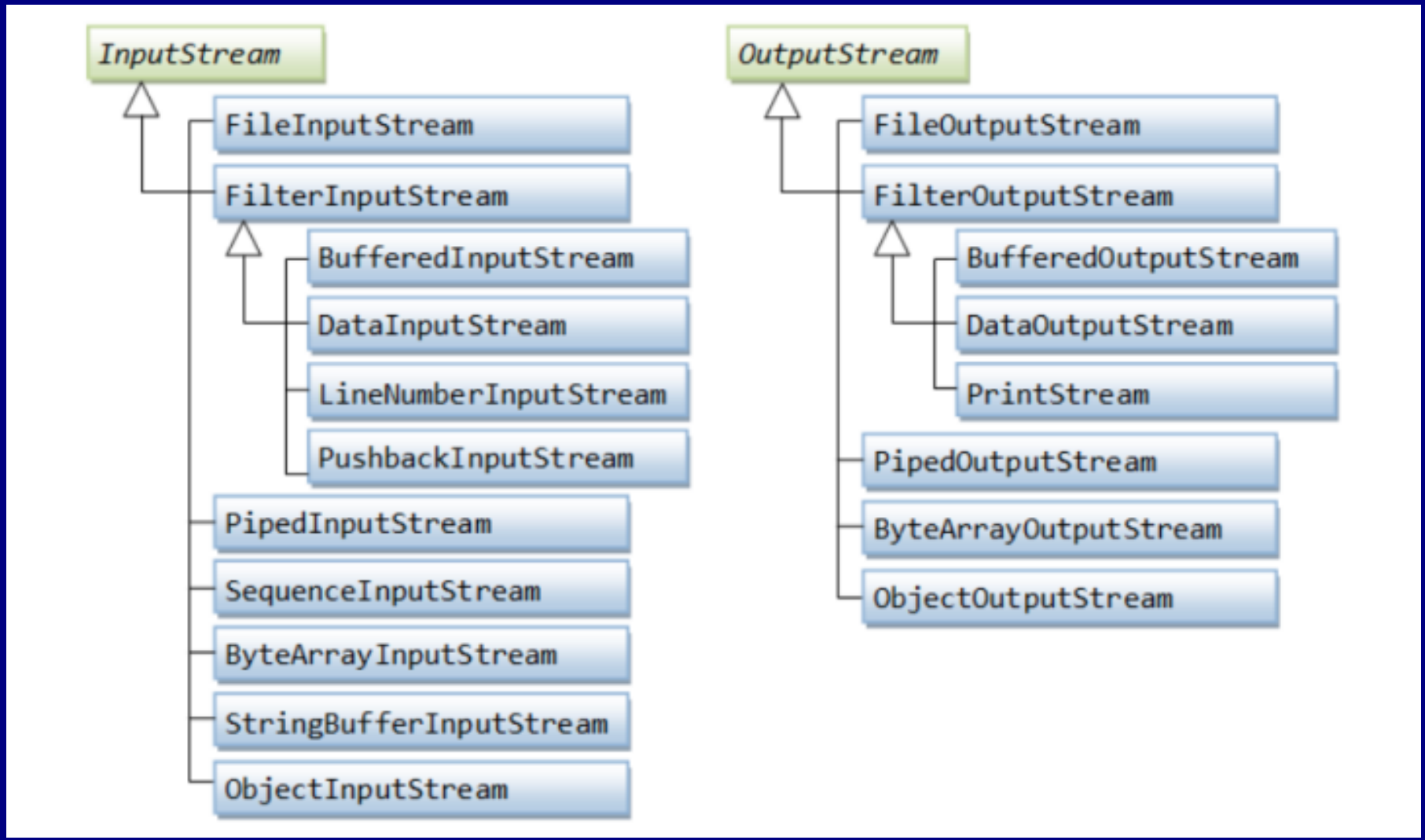
- Abrir el flujo.
- Mientras que haya datos vamos a escribir datos en el flujo.

Flujos de salida

La forma de trabajar es la siguiente:

- Abrir el flujo.
- Mientras que haya datos vamos a escribir datos en el flujo.
- Cerrar el flujo.

InputStream y OutputStream



Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

FileOutputStream permite escribir en un fichero byte a byte.

Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

FileOutputStream permite escribir en un fichero byte a byte.

BufferedOutputStream permite escribir grupo de bytes, en vez de byte a byte

Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

FileOutputStream permite escribir en un fichero byte a byte.

BufferedOutputStream permite escribir grupo de bytes, en vez de byte a byte

ByteArrayOutputStream permite escribir en memoria, obteniendo lo escrito en un *array de bytes*

Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

FileOutputStream permite escribir en un fichero byte a byte.

BufferedOutputStream permite escribir grupo de bytes, en vez de byte a byte

ByteArrayOutputStream permite escribir en memoria, obteniendo lo escrito en un *array de bytes*

Flujos de salida de bytes

OutputStream es la clase padre, abstracta, no se puede instanciar.

FileOutputStream permite escribir en un fichero byte a byte.

BufferedOutputStream permite escribir grupo de bytes, en vez de byte a byte

ByteArrayOutputStream permite escribir en memoria, obteniendo lo escrito en un *array de bytes*

FileOutputStream

Los usamos cuando estamos leyendos datos primitivos.

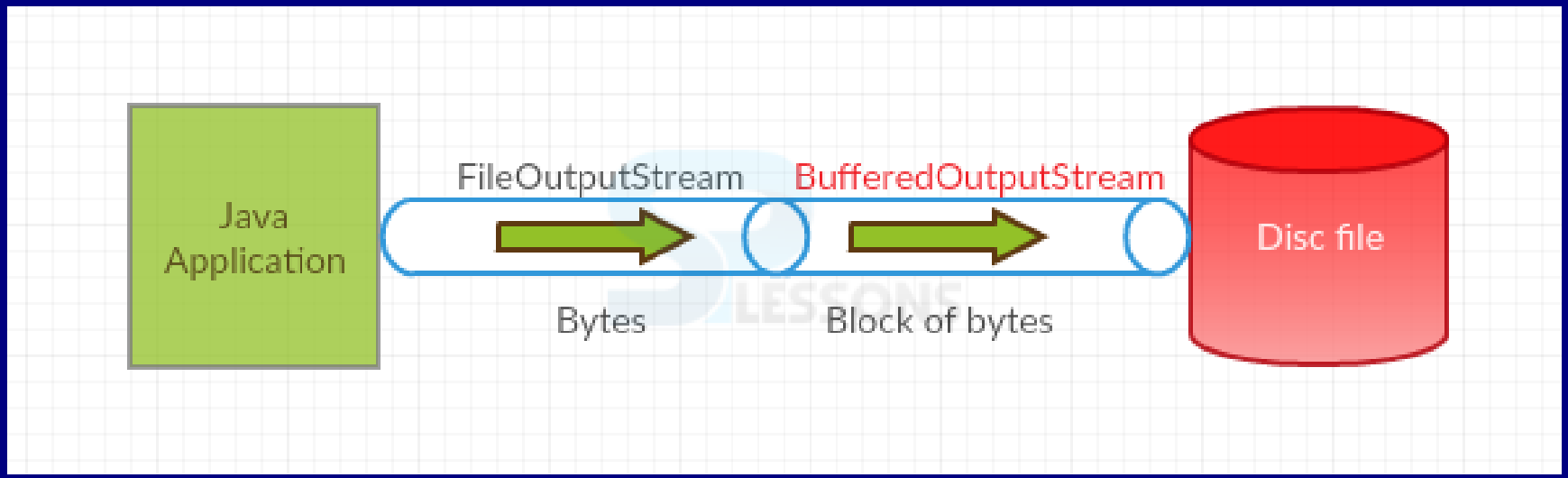
FileOutputStream

Los usamos cuando estamos leyendos datos primitivos.

```
File file = new File("c:/newfile.txt");
String content = "This is the text content";

try (FileOutputStream fop = new FileOutputStream(file)) {

// if file doesn't exists, then create it
  if (!file.exists()) {
    file.createNewFile();
  }

  // get the content in bytes
  byte[] contentInBytes = content.getBytes();

  fop.write(contentInBytes);
  fop.flush();

  System.out.println("Done");

} catch (IOException e) {
  e.printStackTrace();
}
```

BufferedOutputStream

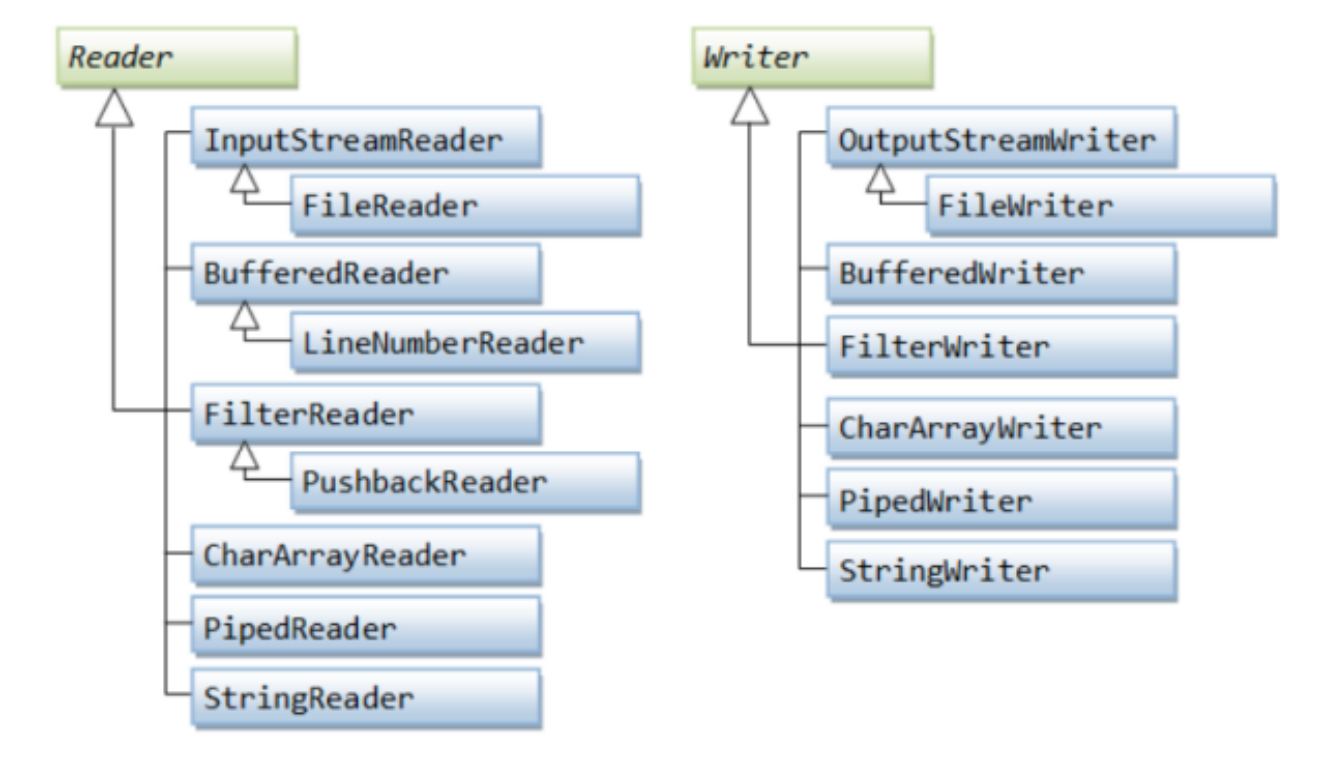
Volcamos el flujo de datos a otro



BufferedOutputStream

```
try (BufferedOutputStream stream =
        new BufferedOutputStream(
          new FileOutputStream("textfile.txt"));){

    stream.write("Hello, World!".getBytes());
    stream.write(System.lineSeparator().getBytes());
    stream.write("I am writting into a file using " +
                 "BufferedOutputStream".getBytes());
    stream.write(System.lineSeparator().getBytes());
    stream.close();
  } catch (IOException ex) {
    ex.printStackTrace();
  }
```

Flujos de salida de caracteres



Flujos de salida de caracteres

Writer es la clase padre, abstracta, no se puede instanciar.

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter
StringWriter	permite escribir en memoria, obteniendo lo escrito en un *String*

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter
StringWriter	permite escribir en memoria, obteniendo lo escrito en un *String*
OutputStreamWriter	permite convertir un *OutputStream* en un *Writer*

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter
StringWriter	permite escribir en memoria, obteniendo lo escrito en un *String*
OutputStreamWriter	permite convertir un *OutputStream* en un *Writer*
PrintWriter	flujo que permite escribir datos básicos de Java.

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter
StringWriter	permite escribir en memoria, obteniendo lo escrito en un *String*
OutputStreamWriter	permite convertir un *OutputStream* en un *Writer*
PrintWriter	flujo que permite escribir datos básicos de Java.

Flujos de salida de caracteres

Writer	es la clase padre, abstracta, no se puede instanciar.
FileWriter	permite escribir en un fichero caracter a caracter.
BufferedWriter	permite escribir grupo de caracteres, en vez de caracter a caracter
StringWriter	permite escribir en memoria, obteniendo lo escrito en un *String*
OutputStreamWriter	permite convertir un *OutputStream* en un *Writer*
PrintWriter	flujo que permite escribir datos básicos de Java.

FileWriter

```
FileWriter fw = null;
String intro = "En un lugar de La Mancha," +
                " de cuyo nombre no quiero acordarme";

try {
  fw = new FileWriter("introquijote.txt");
  for(char c : intro.toCharArray())
    fw.write(c);
} catch (IOException e) {
  e.printStackTrace();
} finally {
    if (fw != null)
       try {
            fw.close();
       } catch (IOException e) {
            e.printStackTrace();
       }
}
```

BufferedWriter

```
BufferedWriter bw = null;
List<String> quijote = List.of("En un lugar de la Mancha,"
        , "de cuyo nombre no quiero acordarme,",
        "no ha mucho tiempo que vivía un hidalgo",
        "de los de lanza en astillero,",
        "adarga antigua, rocín flaco y galgo corredor.");

try {
      bw = new BufferedWriter(new FileWriter("quijote.txt"));
      for (String s : quijote) {
            bw.write(s);
            bw.newLine();
      }
} catch (IOException e) {
      e.printStackTrace();
} finally {
      if (bw != null)
      try {
          bw.close();
      } catch (IOException e) {
          e.printStackTrace();
      }
}
```

PrintStream & PrintWriter

- La clase *PrintStream* y *PrintWriter* se usa para escribir texto formateado bajo *OutputStream*

PrintStream & PrintWriter

- La clase *PrintStream* y *PrintWriter* se usa para escribir texto formateado bajo *OutputStream*
- Tenemos métodos como *print*, *printf* o *format*

PrintStream & PrintWriter

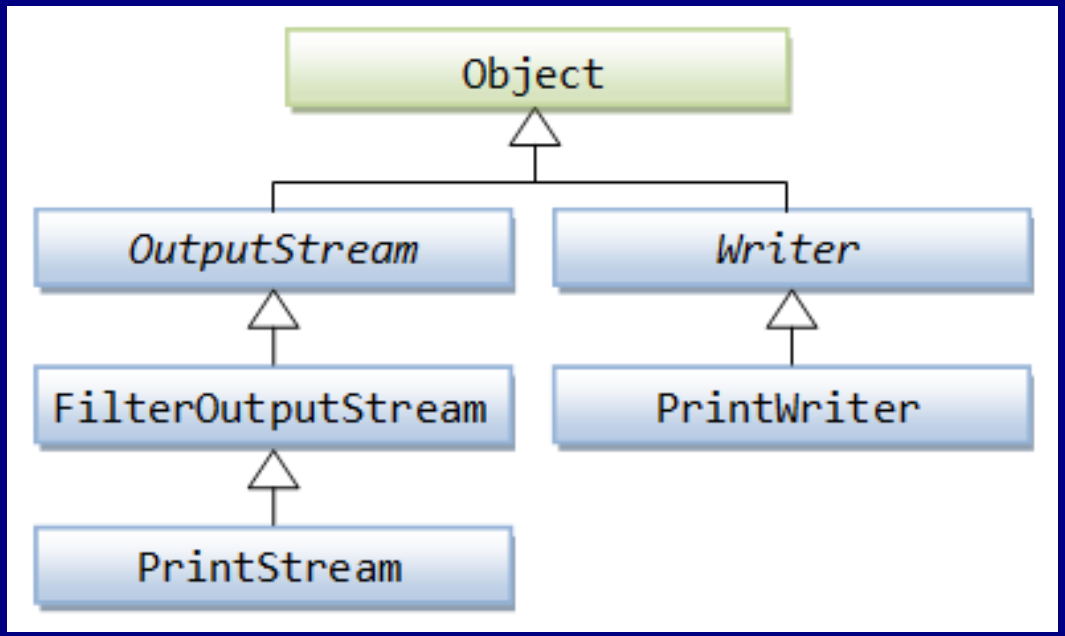
- La clase *PrintStream* y *PrintWriter* se usa para escribir texto formateado bajo *OutputStream*
- Tenemos métodos como *print*, *printf* o *format*

PrintStream & PrintWriter

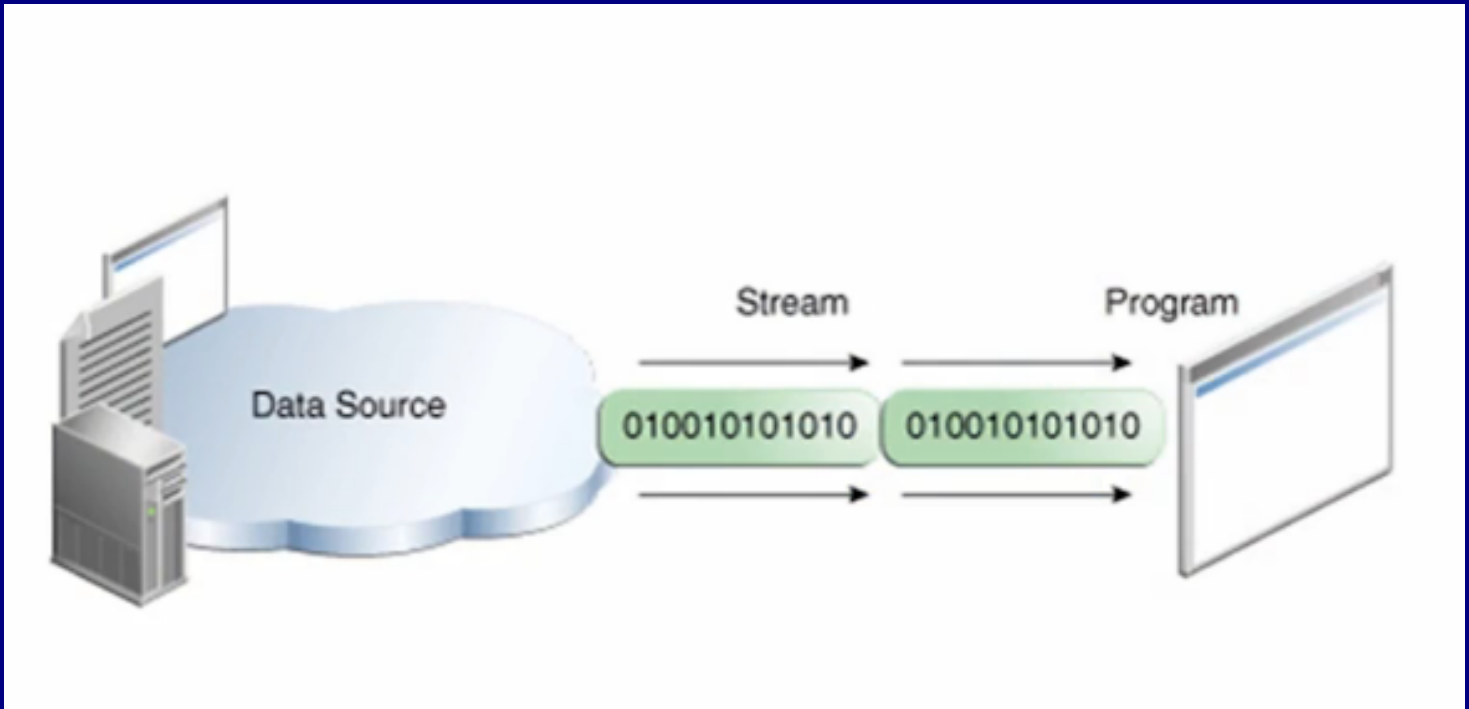
- La clase *PrintStream* y *PrintWriter* se usa para escribir texto formateado bajo *OutputStream*
- Tenemos métodos como *print*, *printf* o *format*

```
import java.io.*;
public class Print{
        public static void main(String[] arg) throws Exception{
                PrintStream output = new PrintStream(
                   new FileOutputStream(new File("hola.txt")));
                output.println(true);
                output.println((int) 123);
                output.println((float) 123.456);
                output.printf("%.2f %n", 12.3698);
                output.close();
        }
}
```

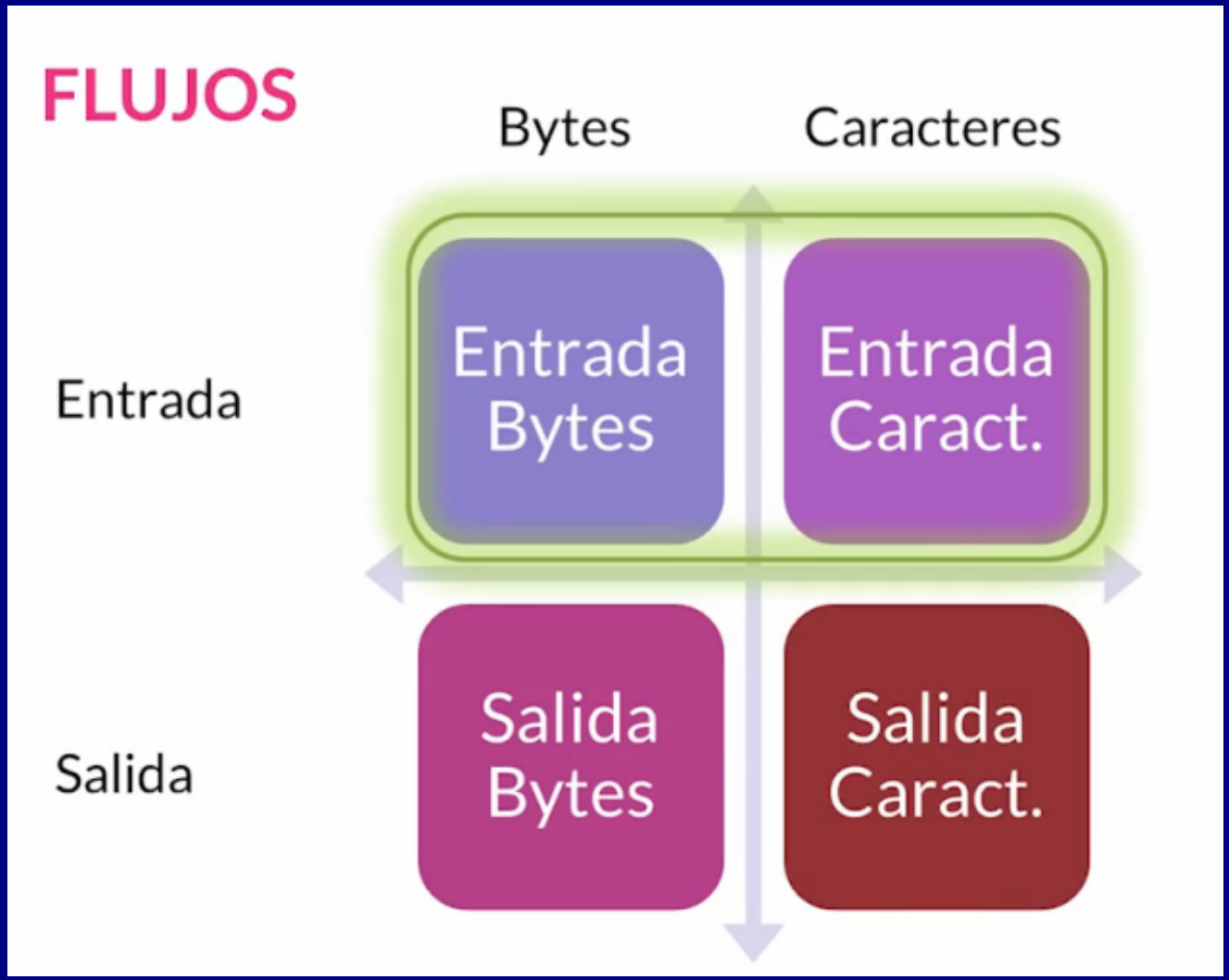
PrintStream & PrintWriter



Flujos de entrada



Flujos de entrada

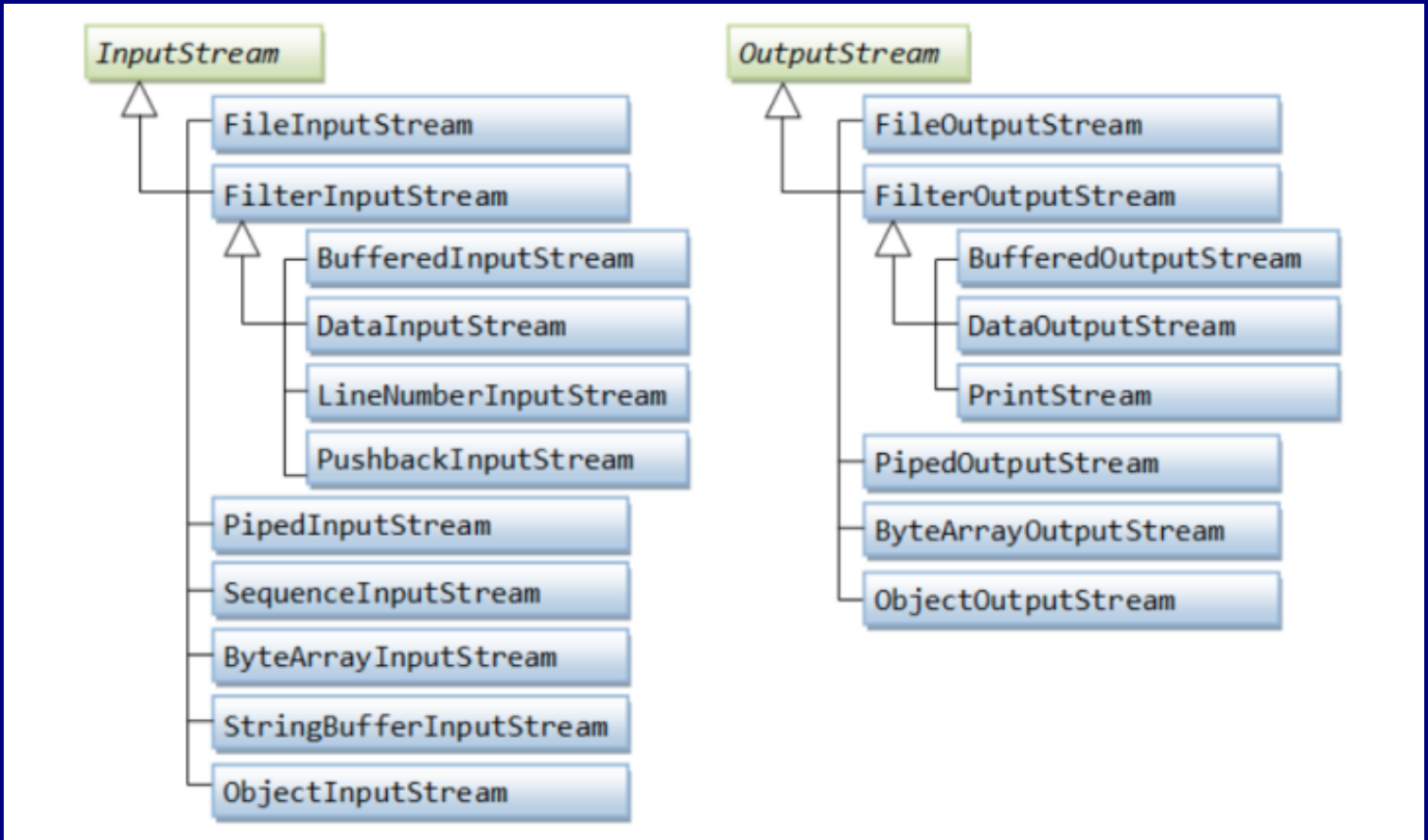


FLUJOS DE ENTRADA

- Patrón básico de uso de flujos de entrada:

```
Abrir el flujo
Mientras hay datos que leer
    Leer datos del flujo
    Procesarlos
Cerrar el flujo
```

Flujos de entrada



Flujos de entrada de bytes

InputStream es la clase padre, abstracta, no se puede instanciar.

Flujos de entrada de bytes

InputStream es la clase padre, abstracta, no se puede instanciar.

FileInputStream permite leer del fichero byte a byte.

Flujos de entrada de bytes

InputStream es la clase padre, abstracta, no se puede instanciar.

FileInputStream permite leer del fichero byte a byte.

BufferedInputStream permite leer grupo de bytes, en vez de byte a byte.

Flujos de entrada de bytes

InputStream es la clase padre, abstracta, no se puede instanciar.

FileInputStream permite leer del fichero byte a byte.

BufferedInputStream permite leer grupo de bytes, en vez de byte a byte.

ByteArrayInputStream flujo que permite leer de memoria un array de bytes.

Flujos de entrada de bytes

InputStream es la clase padre, abstracta, no se puede instanciar.

FileInputStream permite leer del fichero byte a byte.

BufferedInputStream permite leer grupo de bytes, en vez de byte a byte.

ByteArrayInputStream flujo que permite leer de memoria un array de bytes.

Flujos de entrada de bytes

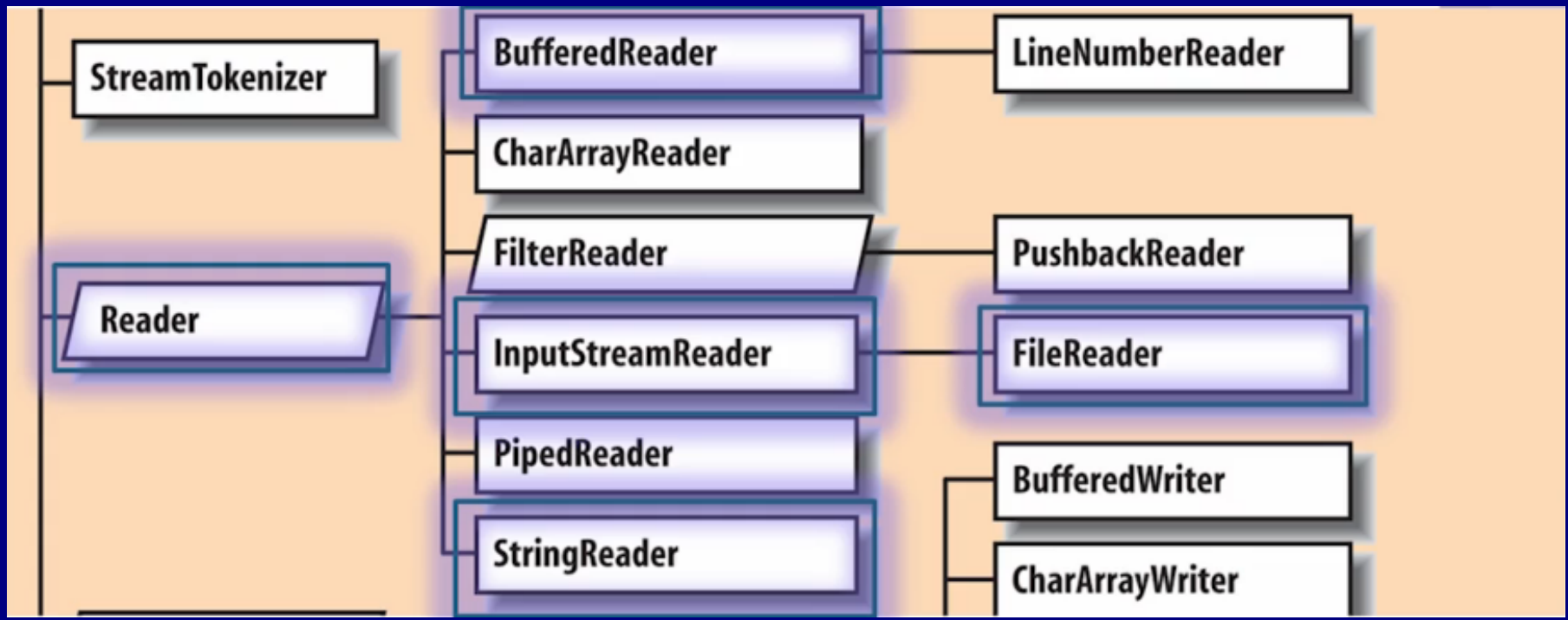
InputStream es la clase padre, abstracta, no se puede instanciar.

FileInputStream permite leer del fichero byte a byte.

BufferedInputStream permite leer grupo de bytes, en vez de byte a byte.

ByteArrayInputStream flujo que permite leer de memoria un array de bytes.

Flujos de entrada de caracteres



Flujos de entrada de caracteres

Reader es la clase padre, abstracta, no se puede instanciar.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.
BufferedReader	permite leer grupo de caracteres, en vez de caracter a caracter.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.
BufferedReader	permite leer grupo de caracteres, en vez de caracter a caracter.
StringReader	flujo que permite leer de memoria un array de caracteres.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.
BufferedReader	permite leer grupo de caracteres, en vez de caracter a caracter.
StringReader	flujo que permite leer de memoria un array de caracteres.
InputStreamReader	Permite transformar un *InputStrema* en un *Reader*.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.
BufferedReader	permite leer grupo de caracteres, en vez de caracter a caracter.
StringReader	flujo que permite leer de memoria un array de caracteres.
InputStreamReader	Permite transformar un *InputStrema* en un *Reader*.

Flujos de entrada de caracteres

Reader	es la clase padre, abstracta, no se puede instanciar.
FileReader	permite leer del fichero caracter a caracter.
BufferedReader	permite leer grupo de caracteres, en vez de caracter a caracter.
StringReader	flujo que permite leer de memoria un array de caracteres.
InputStreamReader	Permite transformar un *InputStrema* en un *Reader*.

Ejemplo sin buffer

```
import java.io.*;
public class FicherosBinariosApp {
  public static void main(String[] args) {

    try(FileInputStream fis=new FileInputStream(
          "D:\\fichero_bin.ddr")){
        int valor=fis.read();
        while(valor!=-1){
           System.out.print((char)valor);
           valor=fis.read();
        }

     }catch(IOException e){
          //.......
     }
   }
}
```

Ejemplo con buffer

```
InputStream inStream = null;
BufferedInputStream bis = null;

try {
   // open input stream test.txt for reading purpose.
   inStream = new FileInputStream("c:/test.txt");

   // input stream is converted to buffered input stream
   bis = new BufferedInputStream(inStream);

   // read until a single byte is available
   while(bis.available()>0) {

      // read the byte and convert the integer to character
      char c = (char)bis.read();

      // print the characters
      System.out.println("Char: "+c);;
   }
} catch(Exception e) {
   // if any I/O error occurs
   e.printStackTrace();
} finally {
   // releases any system resources associated with the stream
   if(inStream!=null)
      inStream.close();
   if(bis!=null)
      bis.close();
}
```

Entrada flujo char

```
import java.io.*;
// Write a text message to an output file, then read it back.
// FileReader/FileWriter uses the default charset for file encoding.
public class BufferedFileReaderWriterJDK7 {
   public static void main(String[] args) {
      String strFilename = "out.txt";
      String message = "Hello, world!\nHello, world again!\n";

      // Print the default charset
      System.out.println(java.nio.charset.Charset.defaultCharset());

      try (BufferedWriter out = new BufferedWriter(new FileWriter(strFilename))) {
         out.write(message);
         out.flush();
      } catch (IOException ex) {
         ex.printStackTrace();
      }

      try (BufferedReader in = new BufferedReader(new FileReader(strFilename))) {
         String inLine;
         while ((inLine = in.readLine()) != null) {  // exclude newline
            System.out.println(inLine);
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
}
```

Problemas de codificación

Lectura de un fichero con codificación iso-8859-1

```
try (BufferedReader in = new BufferedReader(
     new InputStreamReader(
         new FileInputStream("prueba2.txt"), "iso-8859-1"))
            String linea;
            while ((linea = in.readLine()) != null)
                  System.out.println(linea);
} catch (IOException e) {
         e.printStackTrace();
}
```

Para una escritura

```
BufferedWriter out = new BufferedWriter(
  new OutputStreamWriter(
     new FileOutputStream("prueba3.txt"), "iso-8859-1"))
```

Clase Scanner

- JDK 1.5 introduce java.util.Scanner.

Clase Scanner

- JDK 1.5 introduce java.util.Scanner.
- Parsea tokens usando diferentes métodos *nextInt()*, *nextByte()*, *nextShort()*, *nextLong()*, *nextFloat()*, *nextDouble()*, *nextBoolean()*, *next() for String*, y *nextLine()*

Clase Scanner

- JDK 1.5 introduce java.util.Scanner.
- Parsea tokens usando diferentes métodos *nextInt()*, *nextByte()*, *nextShort()*, *nextLong()*, *nextFloat()*, *nextDouble()*, *nextBoolean()*, *next() for String*, y *nextLine()*
- Existen métodos *hasNextXxx()* para chequear la disponibilidad de la entrada.

Clase Scanner

Constructores

```
public Scanner(File source) throws FileNotFoundException
public Scanner(File source, String charsetName) throws FileNotFoundException
// Para System.in
public Scanner(InputStream source)
public Scanner(InputStream source, String charsetName)
// para un String
public Scanner(String source)
```

Clase Scanner

Constructores

```
public Scanner(File source) throws FileNotFoundException
public Scanner(File source, String charsetName) throws FileNotFoundException
// Para System.in
public Scanner(InputStream source)
public Scanner(InputStream source, String charsetName)
// para un String
public Scanner(String source)
```

Ejemplo

```
// Construye un Scanner para parsear un int desde teclado
Scanner in1 = new Scanner(System.in);
int i = in1.nextInt();

// Construye un Scanner para parsear los dobles de un fichero
Scanner in2 = new Scanner(new File("in.txt"));   FileNotFoundException
while (in2.hasNextDouble()) {
   double d = in.nextDouble();
}

// Construye un Scanner para parsear  string
Scanner in3 = new Scanner("This is the input text String");
while (in3.hasNext()) {
   String s = in.next();
}
```

Clase File

- Clase fundamental hasta java 6

Clase File

- Clase fundamental hasta java 6
- A partir de NIO.2 pasa a un segundo plano.

Clase File

- Clase fundamental hasta java 6
- A partir de NIO.2 pasa a un segundo plano.
- Nos permite manejar ficheros y escritorios.

Clase File

- Clase fundamental hasta java 6
- A partir de NIO.2 pasa a un segundo plano.
- Nos permite manejar ficheros y escritorios.

Clase File

- Clase fundamental hasta java 6
- A partir de NIO.2 pasa a un segundo plano.
- Nos permite manejar ficheros y escritorios.

```
File f = new File("file.txt");
  System.out.println("File name :"+f.getName());
  System.out.println("Path: "+f.getPath());
  System.out.println("Absolute path:" +f.getAbsolutePath());
  System.out.println("Parent:"+f.getParent());
  System.out.println("Exists :"+f.exists());
  if(f.exists())
    {
      System.out.println("Is writeable:"+f.canWrite());
      System.out.println("Is readable"+f.canRead());
      System.out.println("Is a directory:"+f.isDirectory());
      System.out.println("File Size in bytes "+f.length());
    }
```

Creación de ficheros

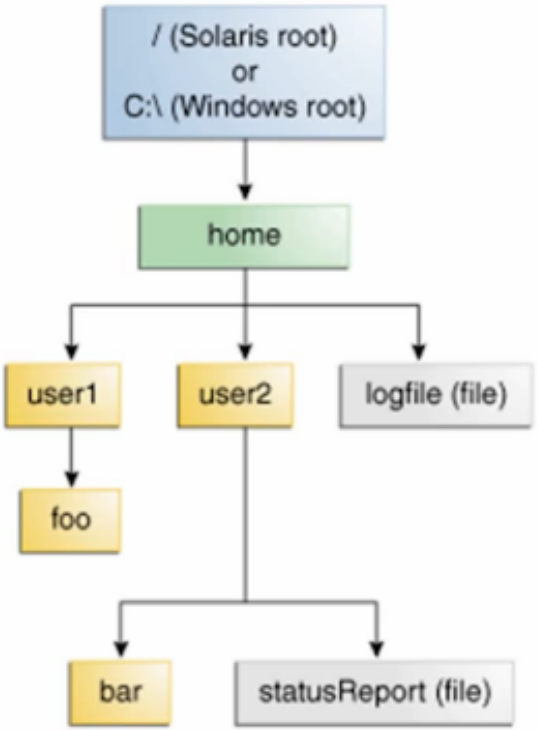
```
try{
  //create a temp file
  File temp = File.createTempFile("temp-file-name", ".tmp");
  System.out.println("Temp file : " + temp.getAbsolutePath());
}catch(IOException e){
  e.printStackTrace();
}
```

Creación de ficheros

```
try{
  //create a temp file
  File temp = File.createTempFile("temp-file-name", ".tmp");
  System.out.println("Temp file : " + temp.getAbsolutePath());
}catch(IOException e){
  e.printStackTrace();
}

File file = new File("c://temp//testFile1.txt");
//Create the file
if (file.createNewFile())
{
   System.out.println("File is created!");
} else {
   System.out.println("File already exists.");
}
//Write Content
FileWriter writer = new FileWriter(file);
writer.write("Test data");
writer.close();
```

Path en Java

¿QUÉ ES UNA RUTA (PATH)?



Es la forma de acceder (o identificar) un fichero dentro del sistema de ficheros

Unix/Solaris
/home/sally/statusReport

Windows
C:\home\sally\statusReport

Tipos de path

ruta absoluta

/home/user/workspace/Proyecto
C:\user\workspace\Proyecto

Tipos de path

ruta absoluta

/home/user/workspace/Proyecto
C:\user\workspace\Proyecto

ruta relativa

workspace/Proyecto

Interfaz Path

- Introducida en Java 7

Interfaz Path

- Introducida en Java 7
- Representa una ruta en el sistema de ficheros.

Interfaz Path

- Introducida en Java 7
- Representa una ruta en el sistema de ficheros.
- Utiliza el nombre del fichero y los directorios padres.

Interfaz Path

- Introducida en Java 7
- Representa una ruta en el sistema de ficheros.
- Utiliza el nombre del fichero y los directorios padres.
- Se suele usar la clase *Paths* y sus métodos estáticos.

Interfaz Path

- Introducida en Java 7
- Representa una ruta en el sistema de ficheros.
- Utiliza el nombre del fichero y los directorios padres.
- Se suele usar la clase *Paths* y sus métodos estáticos.

Interfaz Path

- Introducida en Java 7
- Representa una ruta en el sistema de ficheros.
- Utiliza el nombre del fichero y los directorios padres.
- Se suele usar la clase *Paths* y sus métodos estáticos.

Ejemplo de Path

```
Path path = Paths.get(System.getProperty("user.home"),
                      "documentos", "java", "temario.txt");

System.out.format("toString: %s%n", path.toString());
System.out.format("getFileName: %s%n", path.getFileName());
System.out.format("getName(0): %s%n", path.getName(0));
System.out.format("getNameCount: %d%n", path.getNameCount());
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));
System.out.format("getParent: %s%n", path.getParent());
System.out.format("getRoot: %s%n", path.getRoot());
```

Clase Files

- Introduce muchos métodos estáticos

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.
- Flujos sin *buffered* *newInputStream* y *newOutputStream*

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.
- Flujos sin *buffered* *newInputStream* y *newOutputStream*
- *buffer* con *newBufferedReader* y *newBufferedWriter*

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.
- Flujos sin *buffered* *newInputStream* y *newOutputStream*
- *buffer* con *newBufferedReader* y *newBufferedWriter*
- También los métodos:

```
static byte[] readAllBytes(Path path)
static List<String> readAllLines(Path path,
       Charset cs)
static Path write(Path path, byte[] bytes,
       OpenOption... options)
static Path write(Path path,
       Iterable<? extends CharSequence> lines,
       Charset cs, OpenOption... options)
```

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.
- Flujos sin *buffered* *newInputStream* y *newOutputStream*
- *buffer* con *newBufferedReader* y *newBufferedWriter*
- También los métodos:

```
static byte[] readAllBytes(Path path)
static List<String> readAllLines(Path path,
       Charset cs)
static Path write(Path path, byte[] bytes,
       OpenOption... options)
static Path write(Path path,
       Iterable<? extends CharSequence> lines,
       Charset cs, OpenOption... options)
```

Clase Files

- Introduce muchos métodos estáticos
- Algunos como comprobación de la existencia del fichero, conocer atributos de lectura, escritura, ...
- Métodos para copiar, borrar, mover, ...
- Métodos para trabajar con directorios.
- Creación de ficheros regulares y temporales.
- Flujos sin *buffered* *newInputStream* y *newOutputStream*
- *buffer* con *newBufferedReader* y *newBufferedWriter*
- También los métodos:

```
static byte[] readAllBytes(Path path)
static List<String> readAllLines(Path path,
       Charset cs)
static Path write(Path path, byte[] bytes,
       OpenOption... options)
static Path write(Path path,
       Iterable<? extends CharSequence> lines,
       Charset cs, OpenOption... options)
```

Copiando ficheros

```
Path pathIn = (Path)Paths.get("/usr", "local",
                        "bin", "fileIn.txt");
Path pathOut = (Path)Paths.get(
             "/usr", "local", "bin", "fileOut.txt");
System.out.println("Path of target file: "
                   + pathIn.toString());
System.out.println("Path of source file: "
                        + pathOut.toString());
try {
    System.out.println("Number of bytes copied: "
           + Files.copy(
           pathOut, pathIn,
           StandardCopyOption.REPLACE_EXISTING));
} catch (IOException e) {
            e.printStackTrace();
}
```

Leyendo a nivel de bytes o caracter

bytes

```
Path path = Paths.get("C:/temp/test.txt");
byte[] data = Files.readAllBytes(path);
```

caracteres

```
Path path = Paths.get(URI.create("gs://bucket/lolcat.csv"))
List<String> lines = Files.readAllLines(
   path, StandardCharsets.UTF_8);
```

Escribiendo en fichero

```
private static void writeUsingFiles(String data) {
        try {
            Files.write(Paths.get(
              "/Users/pankaj/files.txt"), data.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

FIN