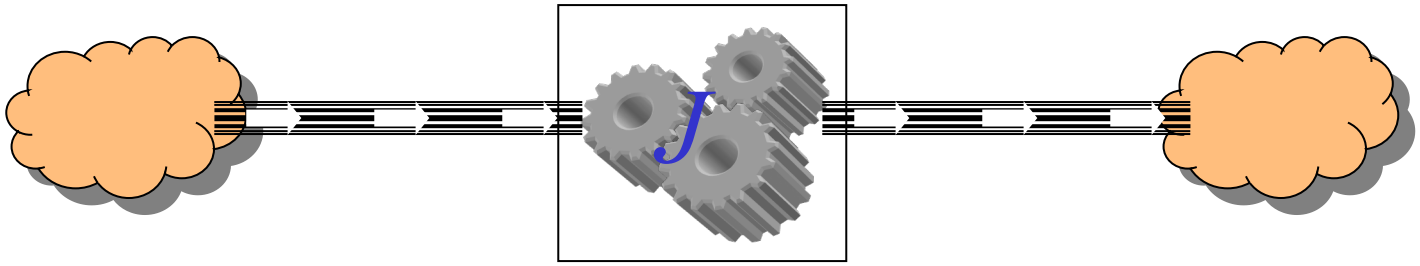
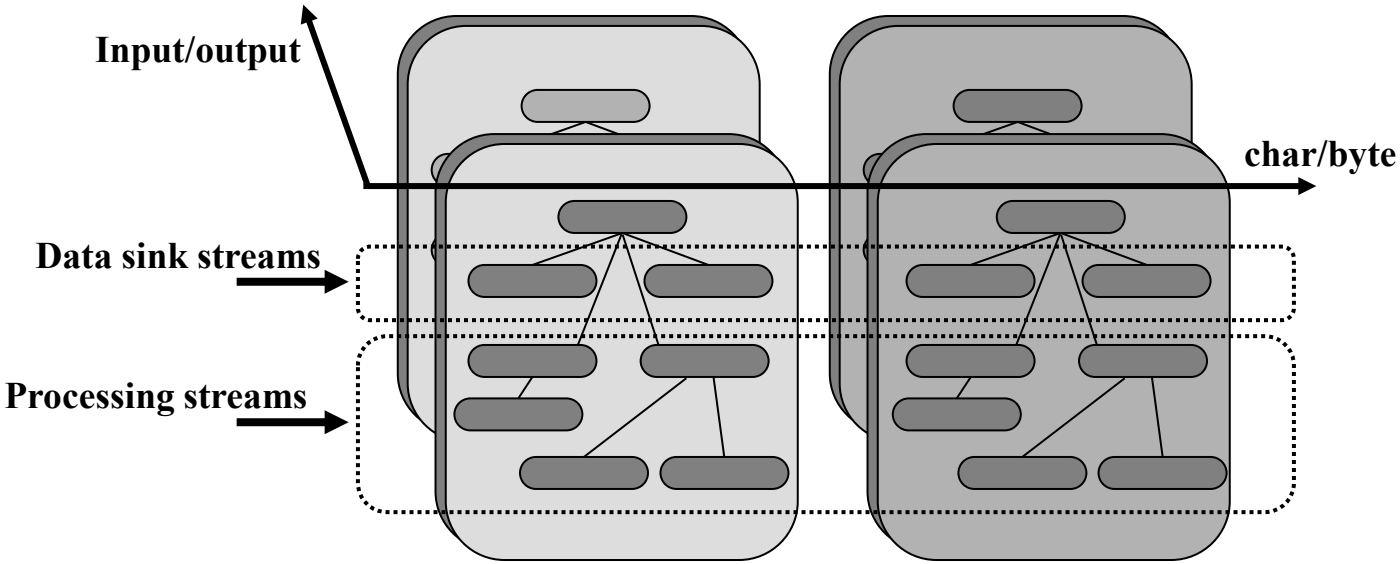
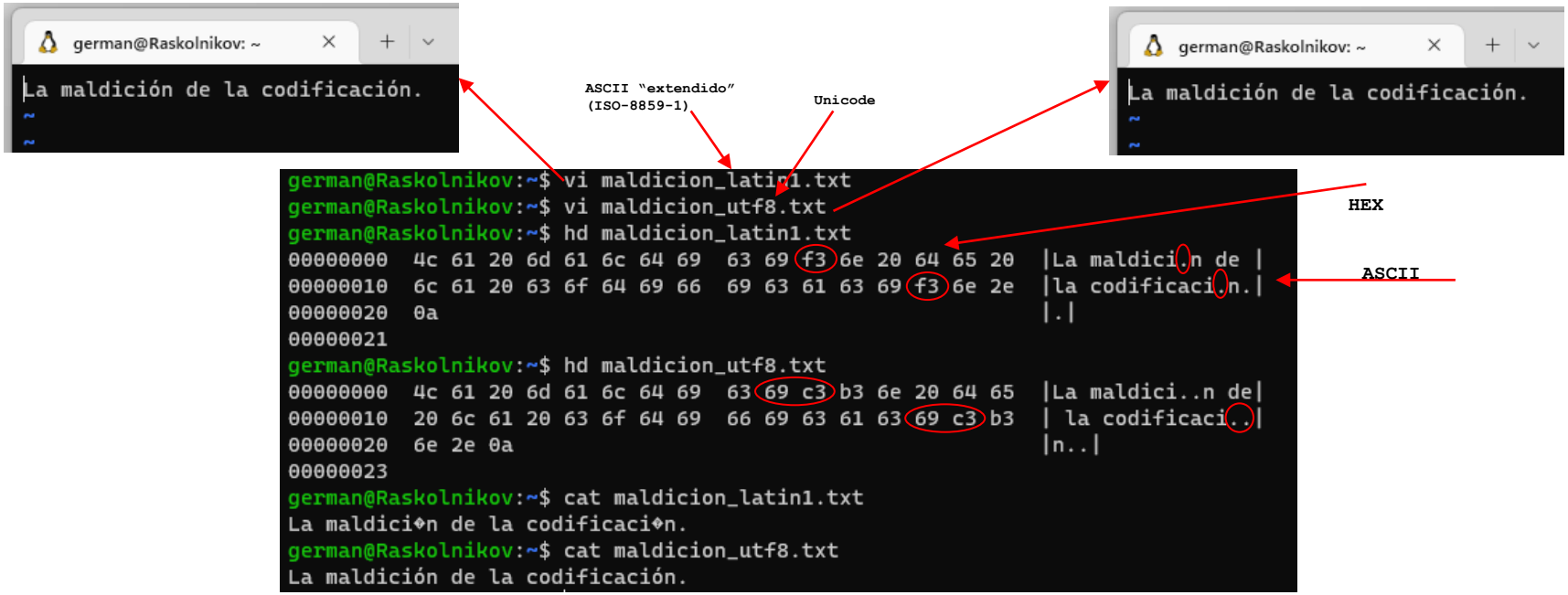




german.bordel@ehu.es

Ojo al problema de los "Charset"



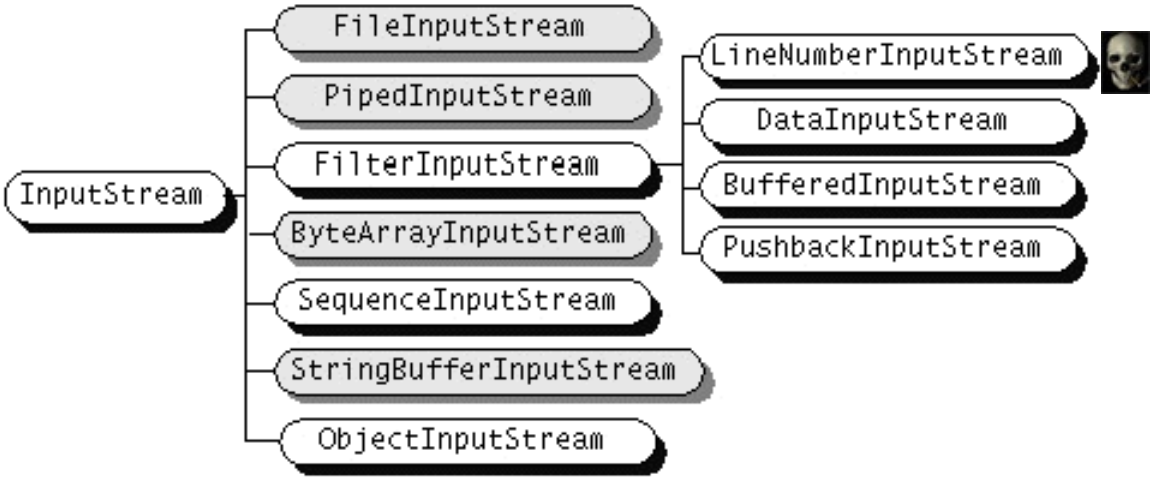
Origen/Destino	Streams de caracteres	Streams de Bytes
Memoria	`CharArrayReader` `CharArrayWriter`	`ByteArrayInputStream` `ByteArrayOutputStream`
	`StringReader` `StringWriter`	`StringBufferInputStream`
“Pipes”	`PipedReader` `PipedWriter`	`PipedInputStream` `PipedOutputStream`
Ficheros	`FileReader` `FileWriter`	`FileInputStream` `FileOutputStream`

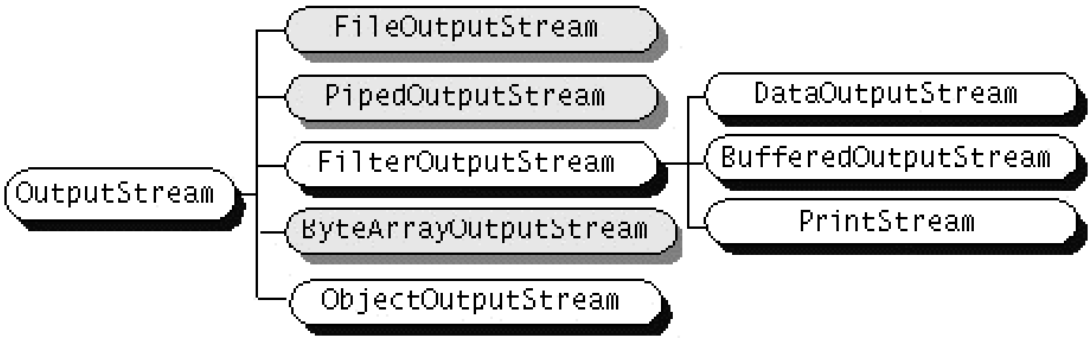
Procesamientos	Streams de caracteres	Streams de Bytes
Conversión de bytes a caracteres	`InputStreamReader`, `OutputStreamWriter`	
Buffering	`BufferedReader`, `BufferedWriter`	`BufferedInputStream`, `BufferedOutputStream`
Filtrado	`FilterReader`, `FilterWriter`	`FilterInputStream`, `FilterOutputStream`
Concatenación		`SequenceInputStream`
Conversión de datos		`DataInputStream`, `DataOutputStream`
Conteo	`LineNumberReader`	`LineNumberInputStream`
Peeking Ahead	`PushbackReader`	`PushbackInputStream`
Impresión	`PrintWriter`	`PrintStream`
Serialización de objetos		`ObjectInputStream`, `ObjectOutputStream`

german.bordel@ehu.es

InputStream

```
int read()
int read(byte cbuf[])
int read(byte cbuf[], int offset, int length)
```





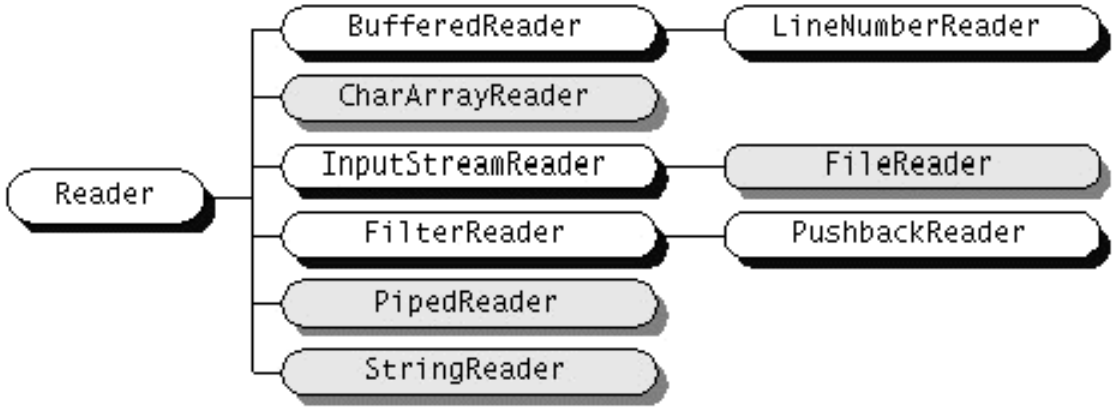
OutputStream

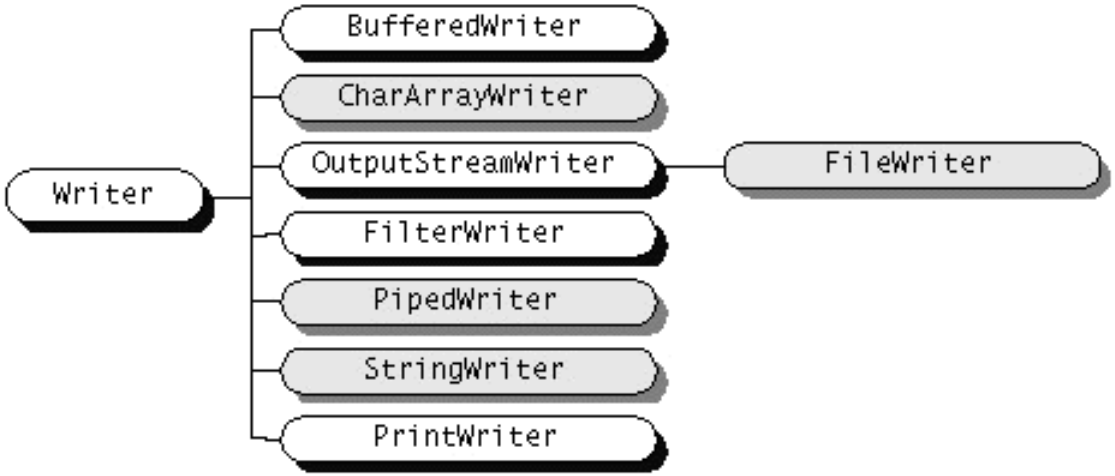
```
int write(int c)
int write(byte cbuf[])
int write(byte cbuf[], int offset, int length)
```

german.bordel@ehu.es

Reader

```
int read()
int read(char cbuf[])
int read(char cbuf[], int offset, int length)
```





Writer

```
int write(int c)
int write(char cbuf[])
int write(char cbuf[], int offset, int length)
```

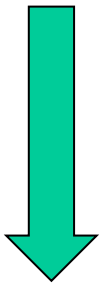
german.bordel@ehu.es

Una simple copia de ficheros

```
import java.io.*;

public class FileCopy {
 public static void main(String[] args) throws IOException {
 try (InputStream entrada = new FileInputStream("origen");
 OutputStream salida = new FileOutputStream("destino");) {
 byte[] buffer = new byte[1024];
 int bytesLeidos;
 while ((bytesLeidos = entrada.read(buffer)) != -1)
 salida.write(buffer, 0, bytesLeidos);
 }
 }
}
```

E/S de objetos

Tema:serialización



```
FileOutputStream out = new FileOutputStream("Tiempos");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
-------------------------------------------------------
FileInputStream in = new FileInputStream("Tiempos");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

german.bordel@ehu.es

Serialización

```
package java.io;
public interface Serializable {
 // there's nothing in here!
};
```

```
public class MySerializableClass implements Serializable
 ...
}
```

```
private void writeObject(ObjectOutputStream s)
 throws IOException {
 s.defaultWriteObject();
 // customized serialization code
}
```

```
private void readObject(ObjectInputStream s)
 throws IOException {
 s.defaultReadObject();
 // customized deserialization code
 ...
 // followed by code to update the object, if necessary
}
```

```
package java.io;
public interface Externalizable extends Serializable
{
 public void writeExternal(ObjectOutput out)
 throws IOException;
 public void readExternal(ObjectInput in)
 throws IOException,
 java.lang.ClassNotFoundException;
}
```

Palabras reservadas en Java

abstract	assert***	boolean	break	byte
case	catch	char	class	const*
continue	default	do	double	else
enum***	extends	final	finally	float
for	goto*	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp**	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances. The writeExternal and readExternal methods of the Externalizable interface are implemented by a class to give the class complete control over the format and contents of the stream for an object and its supertypes. These methods must explicitly coordinate with the supertype to save its state. These methods supercede customized implementations of writeObject and readObject methods.

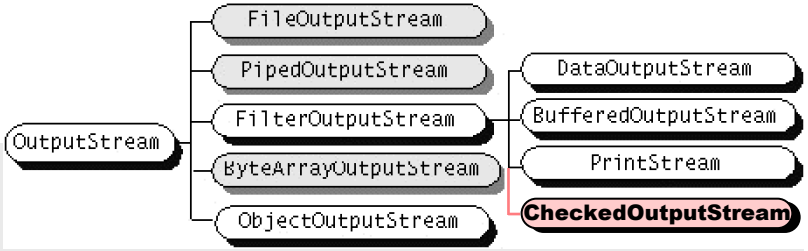
Object Serialization uses the Serializable and Externalizable interfaces. Object persistence mechanisms can use them as well. Each object to be stored is tested for the Externalizable interface. If the object supports Externalizable, the writeExternal method is called. If the object does not support Externalizable and does implement Serializable, the object is saved using ObjectOutputStream.

When an Externalizable object is reconstructed, an instance is created using the public no-arg constructor, then the readExternal method called. Serializable objects are restored by reading them from an ObjectInputStream.

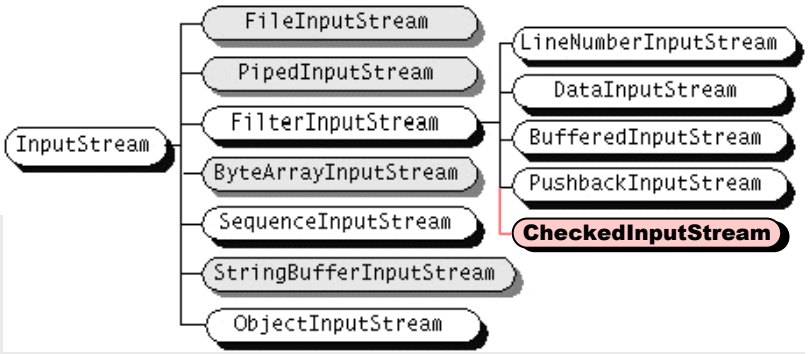
An Externalizable instance can designate a substitution object via the writeReplace and readResolve methods documented in the Serializable interface.

german.bordel@ehu.es

Añadiendo nuestros propios streams. Un ejemplo.




```java
import java.io.*;
public class CheckedOutputStream extends FilterOutputStream {
      private final Checksum cksum;

      public CheckedOutputStream(OutputStream out, Checksum cksum) {super(out); this.cksum = cksum;}

      @override  public void write(int b) throws IOException {out.write(b); cksum.update(b);}

      @override  public void write(byte[] b) throws IOException {write(b, 0, b.length);}

      @override
      public void write(byte[] b, int off, int len) throws IOException {out.write(b, off, len); cksum.update(b, off, len); }

      public long getChecksum() {return cksum.getValue();}
}
```




```java
import java.io.*;

public class CheckedInputStream extends FilterInputStream {
      private final Checksum cksum;

      public CheckedInputStream(InputStream in, Checksum cksum) {super(in); this.cksum = cksum; }

      @override  public int read() throws IOException {int b = in.read(); if(b != -1)cksum.update(b); return b; }

      @override  public int read(byte[] b) throws IOException {return read(b, 0, b.length); }

      @override
      public int read(byte[] b, int off, int len) throws IOException {
            len = in.read(b, off, len); if (len != -1) cksum.update(b, off, len); return len;
      }

      public long getChecksum() {return cksum .getValue();}
}
```

german.bordel@ehu.es

Usando la memoria RAM y nuestros propios streams .

```
public static void main(String[] args) throws IOException {
      byte[] buffer;                                                          //Array de bytes a manejar como Stream

      try (
            ByteArrayOutputStream os=new ByteArrayOutputStream();             //El destino es un array de bytes
            CheckedOutputStream cos=new CheckedOutputStream(os, new CRC32()); //Le añadimos la capacidad de controlar el checksum)
            BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(cos)); //Vamos a leer como texto y con buffer
      ) {
            bw.write("Hola, mundo”);                                          //Escribimos un array de bytes
            bw.flush();                                                       //Nos aseguramos de que no se retiene en el BufferedWriter
            System.out.println(cos.getChecksum());                            //Consultamos el checksum
            buffer=os.toByteArray();                                          //Este es el array de bytes sobre el que hemos escrito
      }

      try (
            ByteArrayInputStream is=new ByteArrayInputStream(buffer);         //El origen es el array de bytes
            CheckedInputStream cis=new CheckedInputStream(is, new CRC32());   //Le añadimos la capacidad de controlar el checksum
            BufferedReader br=new BufferedReader(new InputStreamReader(is));  //Vamos a leer como texto y con buffer
      ) {
            System.out.println(br.readLine());                                //Hacemos la lectura y la mostramos en consola
            System.out.println(cis.getChecksum());                            //Consultamos el checksum
      }
}
```

```
german@Raskolnikov:Java -jar StreamsDemo.jar
768689634
Hola, mundo
768689634
```

Hay mucho más IO

p.ej **java.io.RandomAccessFile**

Constructores

```
RandomAccessFile(File file, String mode)
RandomAccessFile(String name, String mode)
```

Métodos

```
void close()
FileDescriptor getFD()
long getFilePointer()
long length()
int read()
int read(byte[] b)
int read(byte[] b, int off, int len)
boolean readBoolean()
byte readByte()
char readChar()
double readDouble()
float readFloat()
void readFully(byte[] b)
void readFully(byte[] b, int off, int len)
int readInt()
String readLine()
long readLong()
short readShort()
int readUnsignedByte()
int readUnsignedShort()
String readUTF()
void seek(long pos)
void setLength(long newLength)
int skipBytes(int n)
void write(byte[] b)
void write(byte[] b, int off, int len)
void write(int b)
void writeBoolean(boolean v)
void writeByte(int v)
void writeBytes(String s)
void writeChar(int v)
void writeChars(String s)
void writeDouble(double v)
void writeFloat(float v)
void writeInt(int v)
void writeLong(long v)
void writeShort(int v)
void writeUTF(String str)
```

De hecho, además de java.io existe java.nio

german.bordel@ehu.es

En Java 1.4 aparece java.nio para mejorar el rendimiento y la escalabilidad de las operaciones de entrada/salida, con un enfoque diferente (en Java 7 y Java 13 se añaden más mejoras).

Característica	java.io	java.nio (desde Java 1.4)
Modelo de acceso	Orientado a flujo (stream)	Orientado a bloque (buffer)
Bloqueo	Siempre bloqueante	Soporta I/O no bloqueante
Lectura/escritura	InputStream, OutputStream, Reader, Writer	Channel, Buffer
Multiplexación	No disponible	Selector permite manejar múltiples canales
Acceso a archivos	FileInputStream, FileReader	FileChannel, Path (desde Java 7)
Rendimiento	Adecuado para tareas simples	Mejor para servidores y alto rendimiento
Uso típico	Lectura/escritura secuencial	Procesamiento masivo, servidores, sockets

Aunque obedezca a una arquitectura diferente, algunos de sus recursos pueden ser válidos también en tareas que no requieran procesamiento masivo y por tanto usemos java.io normalmente:

p.ej **java.nio.Files** (clase maleta para trabajar con ficheros)

- ejemplo de un método en esta clase: `static byte[] readAllBytes(Path path)`
- la copia de ficheros usando java.nio: `Files.copy(new File("org").toPath(),new File("dst").toPath());`