AAD

Acceso a Datos
UD02T02 – Flujos JAVA
(Acceso secuencial)

IES Ciudad Escolar
Desarrollo de Aplicaciones Multiplataforma

Índice

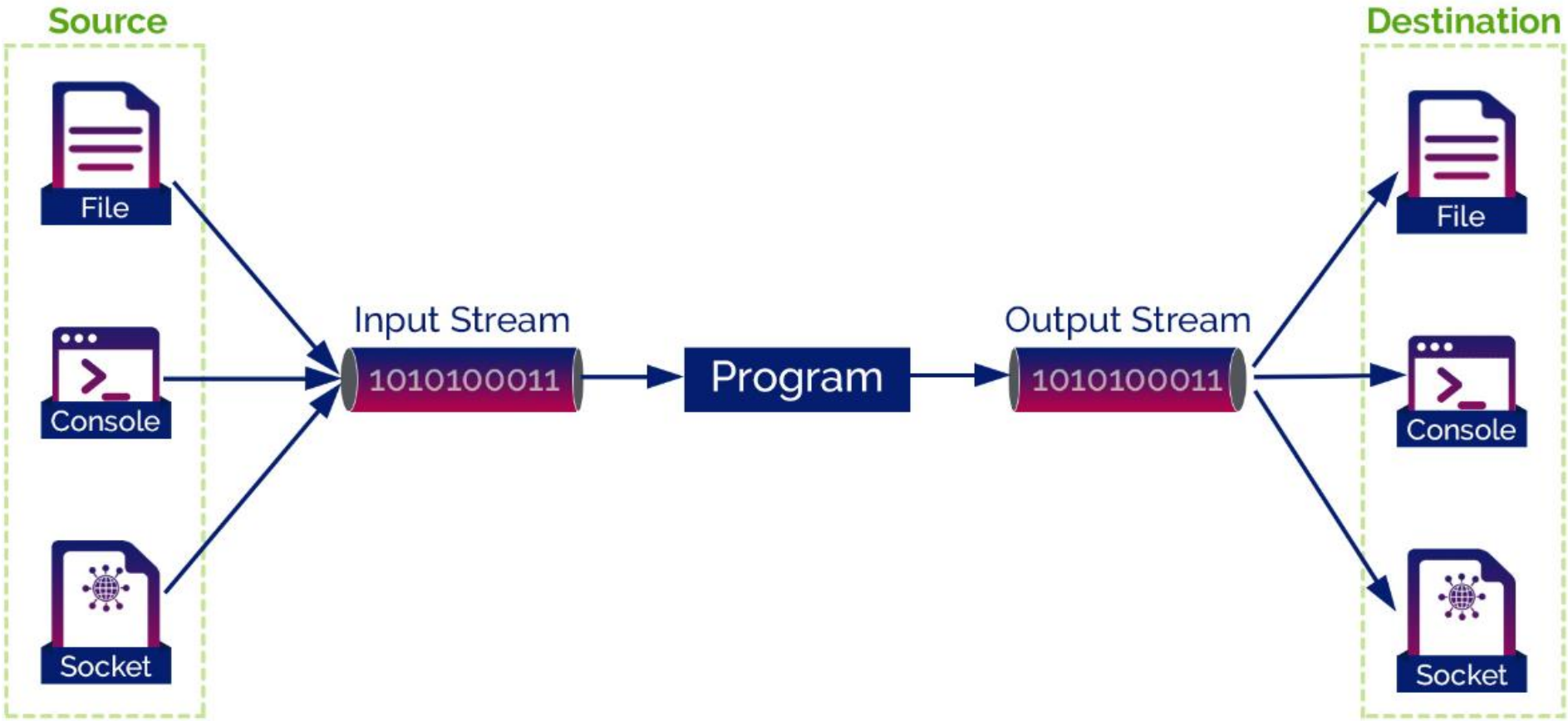
- Flujos (Streams)
- Flujos binarios (Serialización)
- Flujos de texto

Flujos o Streams

- Un **stream** es una abstracción de Java que representa a un **flujo** o secuencia de bytes.
- Permiten procesar ficheros, buffer de memoria o conexiones de red de la misma forma.
- Stream está definido en el paquete **java.io**
- En Java, todo programa crea automáticamente 3 flujos asociados a la consola:
  - **System.out**: Flujo de salida estándar.
  - **System.in**: Flujo de entrada estándar.
  - **System.err**: Flujo de error estándar.

Flujos o Streams

Las operaciones de E/S de un programa tienen dos flujos diferenciados: uno de entrada (lectura) y otro de salida (escritura).



Flujos o Streams

- Existen dos tipos de flujos principales:
    - los **flujos binarios** (8 bits) y los **flujos de texto** (16 bits)
- Los flujos binarios leen o escriben bytes mientras que los flujos de texto leen o escriben caracteres (Unicode 16 bits).
- Para evitar que cada lectura o escritura se acceda directamente a la fuente, se puede utilizar un buffer intermedio entre la fuente (disco físico, socket …) y el flujo.
- Sea el flujo que sea, la **operativa** siempre supone la misma secuencia de acciones:

1º) **Abrir el flujo**
2º) **Usar el flujo (para leer o escribir)**
3º) **Cerrar el flujo**

Flujos binarios

- El flujo binario se define a partir de dos clases abstractas: **InputStream** y **OutputStream**.
- La primera para operaciones de entrada (lectura) y la segunda para operaciones de salida (escritura).
- De ellas derivan una serie de clases concretas para llevar a cabo las diferentes operaciones de E/S usando flujos binarios.
- Algunas de las clase más comunes son:

De entrada	Operación E/S	De salida
BufferedInputStream	Sobre buffer	**BufferedOutputStream**
ByteArrayInputStream	Sobre array de bytes	**ByteArrayOutputStream**
DataInputStream	Sobre tipos básicos de datos	**DataOutputStream**
ObjectInputStream	Sobre objetos	**ObjectOutputStream**
FileInputStream	Sobre ficheros	**FileOutputStream**
		PrintStream - print() println()

Se escribe texto de igual forma que a través del flujo System.out sólo que el destino puede ser un fichero, un socket, etc.

Flujos binarios (Ejemplo I)

```
import java.io.*;

public class LecturaTecladoBytes {

        public static void main(String[] args) throws IOException {

                BufferedInputStream bis = new BufferedInputStream(System.in);

                try {
                        System.out.print("Introduzca un caracter: ");
                        char c = (char)bis.read();
                        System.out.println("Has introducido '" + c + "'");
                }
                catch(Exception e) {
                        System.out.println(e);
                }
                finally {
                        bis.close();
                }
        }
}
```

Flujos binarios (Ejemplo II)

```
import java.io.*;

public class LecturaFicheroBytes {

        public static void main(String[] args) throws IOException {

                FileInputStream fis = new FileInputStream(new File("C:\\dir\\dataFile.dat"));
                BufferedInputStream bis = new BufferedInputStream(fis);

                try {
                        char c = (char)bis.read();
                        System.out.println(“Datos leídos del fichero - '" + c + "'");
                }
                catch(Exception e) {
                        System.out.println(e);
                }
                finally {
                        input.close();
                }
        }
}
```

Flujos binarios: Serialización

- Para poder escribir un objeto en un flujo (ObjectOutputStream) es preciso que sea "**serializable**".

- "Serializar un objeto" consiste en **convertirlo a una secuencia de bytes**, para por ejemplo poder enviarlo por una red y reconstruirlo luego a partir de esos bytes en el destino o para guardarlo en un fichero.

- Su clase (o alguna de sus superclases) debe implementar el **interface Serializable** que no define ningún método, por lo tanto no hay que añadir ningún método extra a la clase y además debe disponer de un constructor por defecto y que todos sus atributos sean serializables

  Tanto los tipos fundamentales de datos, como sus wrapper, así como la clase java.lang.String son serializables). Si hubiera atributos que fueran otras clases, éstos a su vez también deben ser Serializable.

- Si a la hora de escribir un objeto, no queremos que almacene un atributo concreto, tendremos que anteponer a la declaración de dicho atributo la palabra clave **transient**.

  OJO!!! Una vez que hemos serializado un objeto no podremos modificar la clase (ni tan siquiera añadir o eliminar métodos) ya que de los contrario no coincidirán las versiones de las clases produciéndose un error.

Flujos de texto

- El flujo de texto se define a partir de dos clases abstractas: **Reader** y **Writer**.
- La primera para operaciones de entrada (lectura de caracteres) y la segunda para operaciones de salida (escritura de caracteres).
- De ellas derivan una serie de clases concretas para llevar a cabo las diferentes operaciones de E/S usando flujos de texto.
- Algunas de las clase más comunes son:

De entrada	Operación E/S	De salida
BufferedReader	Sobre buffer	**BufferedWriter**
CharArrayReader	Sobre array de caracteres	**CharArrayWriter**
StringReader	Sobre cadena de caracteres	**StringWriter**
FileReader	Sobre ficheros	**FileWriter**
InputStreamReader	Convierte datos de bytes a caracteres	**PrintStream** - print() println()

Se escribe texto de igual forma que a través del flujo System.out sólo que el destino puede ser un fichero, un socket, etc.

Flujos de texto (Ejemplo I)

```
import java.io.*;

public class LecturaFicheroChar {

        public static void main(String[] args) throws IOException {

                Reader r = new FileReader();

                try {
                        char c = (char) r.read();
                        System.out.println("Datos leidos del fichero- '" + c + "'");
                }
                catch(Exception e) {
                        System.out.println(e);
                }
                finally {
                        input.close();
                }
        }
}
```

Flujos de texto (Ejemplo II)

```
import java.io.*;

public class EscrituraFichero {

        public static void main(String[] args) throws IOException {

                Writer out = new FileWriter("("C:\\dir\\dataFile.txt");

                String msg = “Buenos días";

                try {
                        out.write(msg);
                        System.out.println("Writing done!!!");
                }
                catch(Exception e) {
                        System.out.println(e);
                }
                finally {
                        out.close();
                }
        }
}
```