La Sénia
Institut d'Educació Secundària
Paiporta

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Flujos de Entrada y Salida

Curso 2023 - 2024

1. Concepto de Flujo

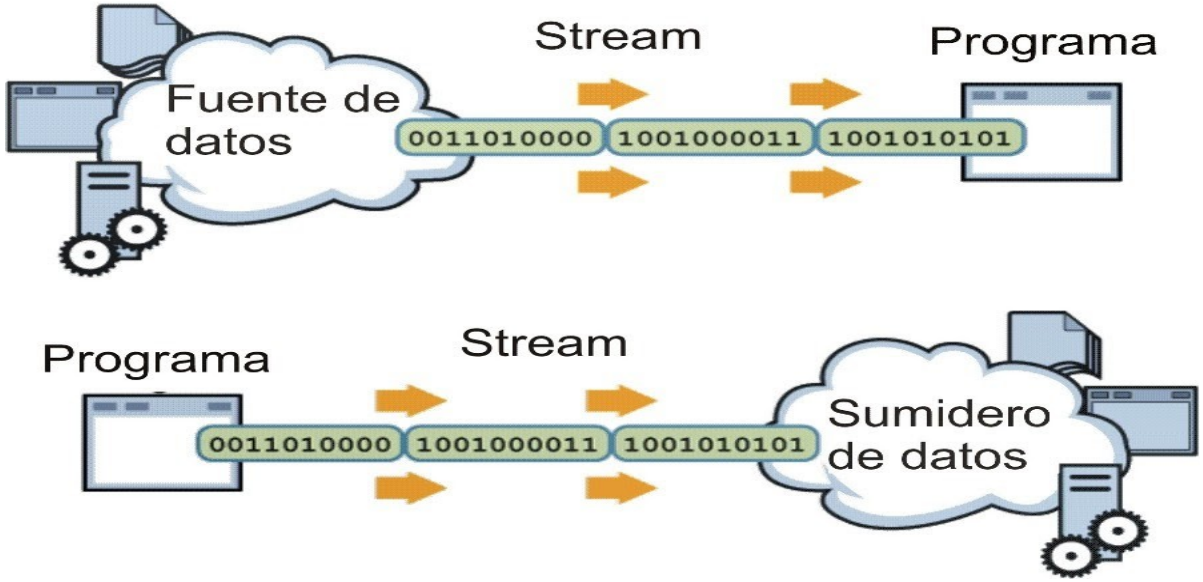
- Un programa consta de unos **datos de entrada** que se **procesan** aplicando unas instrucciones de modo que se obtienen unos resultados o **datos de salida**

- Se define el flujo de los datos (**stream**) como el conjunto de datos que se obtienen de un dispositivo externo (fichero, teclado, red[socket],…) y tras procesarse, y/o se almacenan en un dispositivo externo (fichero, pantalla, red[socket], impresora,…)

2. Tipos de flujos

Los Flujos de Datos o stream pueden ser de varios tipos:

- Flujos de bytes o binarios: Transmiten enteros entre 0 y 255
- Flujo de caracteres: Permiten el trasiego de caracteres (normalmente ASCII).



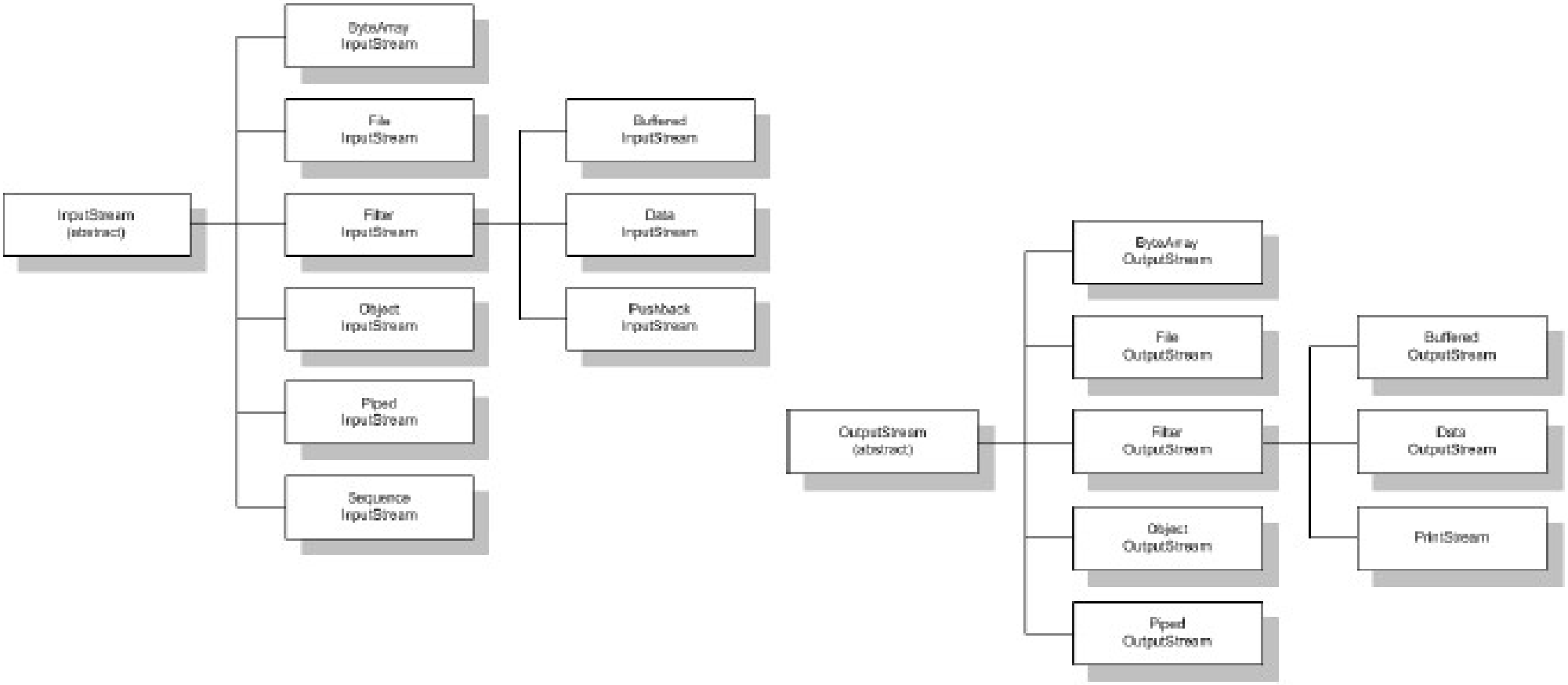
3. Flujos E/S en Java

Gestionar flujos en java: **java.io**

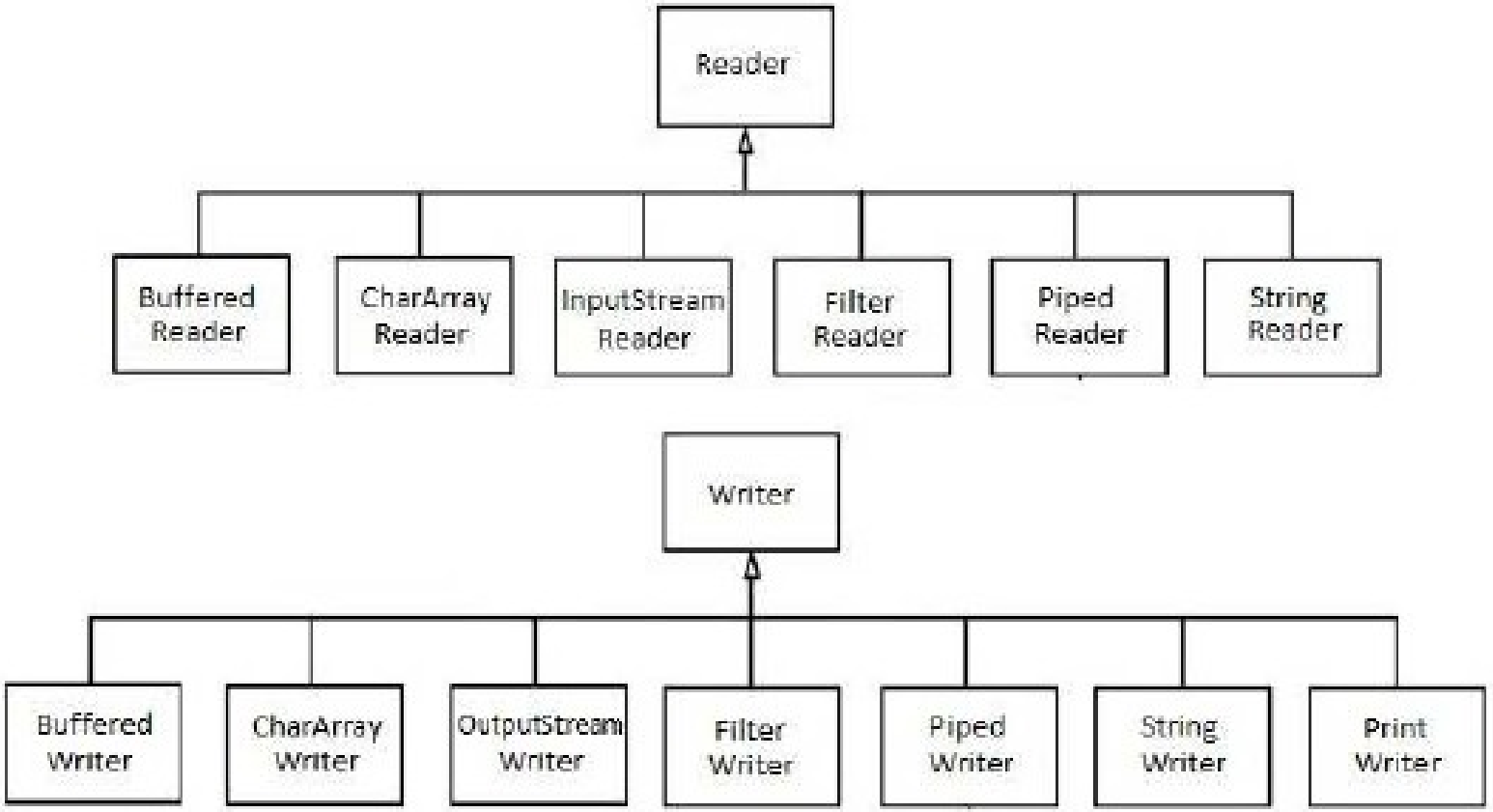
- Flujos de bytes: clases **InputStream** y **OutputStream**. Para ficheros **FileInputStream** y **FileOuputStream**.
- Flujo de caracteres: clases **Reader** y **Writer.** Para ficheros **FileReader** y **FileWriter**.
- Flujos estándar:
  - Flujo de entrada (teclado): **System.in**
  - Flujo de Salida (pantalla): **System.out**
  - Flujo de Errores: **System.err** (se envían los errores a un fichero o a la consola)

3. Flujos E/S en Java

Flujos de bytes



Flujos de caracteres



3. Flujos E/S en Java

Procedimiento de Lectura

1. Abrir un flujo a una fuente de datos creando el objeto stream (Teclado, Fichero, Socket remoto)
2. Mientras existan datos disponibles → **Leer datos**
3. Cerrar el flujo (método close)

Procedimiento de Escritura

1. Abrir un flujo a una fuente de datos creando el objeto stream (Pantalla, Fichero, Socket local)
2. Mientras existan datos disponibles → **Escribir datos**
3. Cerrar el flujo (método close)

Nota: para los flujos estándar (System.in y System.out) ya se encarga el sistema de abrirlos y cerrarlos

3. Flujos E/S en Java

Gestionar flujos en java: java.io

- Flujos de bytes: clases **InputStream** y **OutputStream**. Para ficheros **FileInputStream** y **FileOuputStream**.
- Flujo de caracteres: clases **Reader** y **Writer.** Para ficheros **FileReader** y **FileWriter**.
- Flujos estándar:
  - Flujo de entrada (teclado): **System.in**
  - Flujo de Salida (pantalla): **System.out**
  - Flujo de Errores: System.err (se envían los errores a un fichero o a la consola)

4. Flujos de entrada desde teclado

La clase **Scanner** de Java ofrece métodos para leer valores de entrada de varios tipos y está incluida en el paquete **java.util**

Los valores de entrada pueden venir de varias fuentes como el teclado (System.in) o datos almacenados en un archivo

Debemos crear un objeto de la clase Scanner y asociarlo al dispositivo de entrada (en nuestro caso el teclado):

Scanner sc = new Scanner(**System.in**)

Otros métodos de Scanner:

n=sc.next(); s=sc.NextLine(); sc.hasNext()

4. Flujos de entrada desde teclado

Principales constructores y métodos de la clase Scanner	
public **Scanner** (InputStream source)	Crea un nuevo Scanner a partir de un flujo de entrada de datos como es el caso de System.in (para poder leer desde teclado).
public String **next** () public String **next** (String pattern)	Obtiene el siguiente elemento leído del teclado como un String (si coincide con el patrón especificado). Lanza NoSuchElementException si no quedan más elementos por leer.
public String **nextLine** ()	Se lee el resto de línea completa, descartando el salto de línea. Devuelve el resultado como un String. Lanza NoSuchElementException si no quedan más elementos por leer.
public int **nextInt** () public long **nextLong** () public short **nextShort** () public byte **nextByte** () public float **nextFloat** () public double **nextDouble** () public boolean **nextBoolean** ()	Devuelve el siguiente elemento como un int siempre que se trate de un int. Ídem para long, short, byte, float, double y boolean. Lanza InputMismatchException en caso de no poder obtener un valor del tipo apropiado. Lanza NoSuchElementException si no quedan más elementos por leer.
public boolean **hasNext** ()	Devuelve true si queda algún elemento por leer.
public boolean **hasNextLine** ()	Devuelve true si queda alguna línea por leer.
public boolean **hasNextInt** () public boolean **hasNextLong** () public boolean **hasNextShort** () public boolean **hasNextByte** () public boolean **hasNextFloat** () public boolean **hasNextDouble** () public boolean **hasNextBoolean** ()	Devuelve true si el siguiente elemento a obtener se puede interpretar como un int. Ídem para long, short, byte, float, double y boolean.
public Scanner **useLocale** (Locale l)	Establece la configuración local del Scanner a la configuración especificada por el Locale l.

4. Flujos de entrada desde teclado

Ejemplo:

```
Scanner sc = new Scanner(System.in);

int numClase;
String nombre;
double nota;

System.out.println("Introduce el número de clase:");
numClase = sc.nextInt();

sc.nextLine(); //NOTA: captura el retorno de carro

System.out.println("Introduce el nombre del alumno:");
nombre = sc.nextLine();

System.out.println("Introduce la nota del exámen:");
nota = sc.nextDouble();
```

5. Flujos de salida por pantalla

Para la salida por pantalla tenemos **System.out**

System.**out**.print(“texto”) #Muestra el texto por pantalla

System.**out**.println(“texto”) #Muestra el texto + salto línea

Salida con formato:

System.**out**.printf(“formato”,valores):

%d entero

%s cadena de caracteres (terminada en ‘\0’)

%c carácter ASCII del número pasado como parámetro

%f coma flotante con signo

%e coma flotante con signo en notación científica

5. Flujos de salida por pantalla

Ejemplo:

```
int a = 8, b = 3, resultado = 0;
resultado = (a + b);
System.out.printf("La suma es: %d %n", resultado); //Imprime 11
resultado = (a - b);
System.out.printf("La resta es: %d %n", resultado); //Imprime 5
double res = (double) a / b;  //2.6666666666666667
System.out.printf("La división es: %f\n", res);
//Se pueden imprimir diferentes variables en una misma instrucción
System.out.printf("La división entre %d y %d es igual a %f \n", a, b,
res);
System.out.printf("%.2f %n", res); // Imprime con dos decimales
System.out.printf("%5.2f %n", res); // 5 digitos con dos decimales
//_2,67 (un espacio en blanco delante del 2)
System.out.printf("%7.3f %n", res); // __2,667
System.out.printf("%07.3f %n", res); //002,667 - rellena con ceros
System.out.printf("%10.4f %n", res); // 2,6667  blanco
System.out.printf("%010.0f %n", res); //0000000003
```

5. Flujos de salida por pantalla

Ejemplo:

```
//%c se sustituye por un carácter
//%s se sustituye por la variable de texto
//%S se sustituye por la variable de texto, en mayúsculas

//El salto de línea se puede indicar con \n o %n

System.out.printf("La suma se expresa con el signo %c %n", '+');

String texto = "Mayor";

//Imprime: El resultado es Mayor
System.out.printf("El resultado es: %s \n", texto);

//Imprime: El resultado es MAYOR
System.out.printf("El resultado es: %S %n", texto);
```

5. Flujos de salida por pantalla

DecimalFormat (permite definir un formato de número decimal):

```
import java.text.*;
    DecimalFormat formateador = new DecimalFormat("####.####");
    System.out.println(formateador.format(.12342383));
   //Imprime hasta 4 cifras enteras y 4 decimales
   formateador = new DecimalFormat("0000.0000");
   System.out.println(formateador.format (1.82)); // Imprime: 0001,8200
   //Automáticamente redondea
   double aa = 1.2345, bb = 1.2356;
   formateador = new DecimalFormat("#.##");
   System.out.println(formateador.format( aa )); // La salida es 1,23
   System.out.println(formateador.format( bb )); // La salida es 1,24
   //Porcentajes
   formateador = new DecimalFormat("###.##%");
   System.out.println (formateador.format(0.6844)); // Imprime: 68,44
   //Cambia la notación decimal
   DecimalFormatSymbols simbolos = new DecimalFormatSymbols();
   simbolos.setDecimalSeparator('.');
   formateador = new DecimalFormat("####.####", simbolos);
   System.out.println (formateador.format (3.43242383)); // Imprime: 3.4324
```

Uso de la clase DecimalFormat:

```
//Imprime el número pasado como parámetro con cuatro decimales, es decir: 7,1234
DecimalFormat formateador = new DecimalFormat("####.####");
System.out.println(formateador.format(7.12342383));

//Imprime con 4 cifras enteras y 4 decimales
formateador = new DecimalFormat("0000.0000");
System.out.println(formateador.format (1.82)); // Imprime: 0001,8200

//Redondeo
double aa = 1.2345, double bb = 1.2356;
formateador = new DecimalFormat(“#.##”);
System.out.println(formateador.format( aa )); // La salida es 1,23
System.out.println(formateador.format( bb )); // La salida es 1,24

//Porcentajes
formateador = new DecimalFormat(“###.##%”);
System.out.println (formateador.format(0.6844)); // Imprime: 68,44%

//Simbolos
DecimalFormatSymbols simbolos = new DecimalFormatSymbols();
simbolos.setDecimalSeparator(‘.’);
formateador = new DecimalFormat(“####.####”, simbolos);
System.out.println (formateador.format (3.43242383)); // Imprime: 3.4324
```

#: es sustituido por un número o ninguno, no obligatorio
0: debe haber un número de forma obligatoria, en otro caso se rellena con 0