GESTIÓN DE FLUJOS EN JAVA (STREAM)

Programación de servicios y procesos

Curso 2024/2025

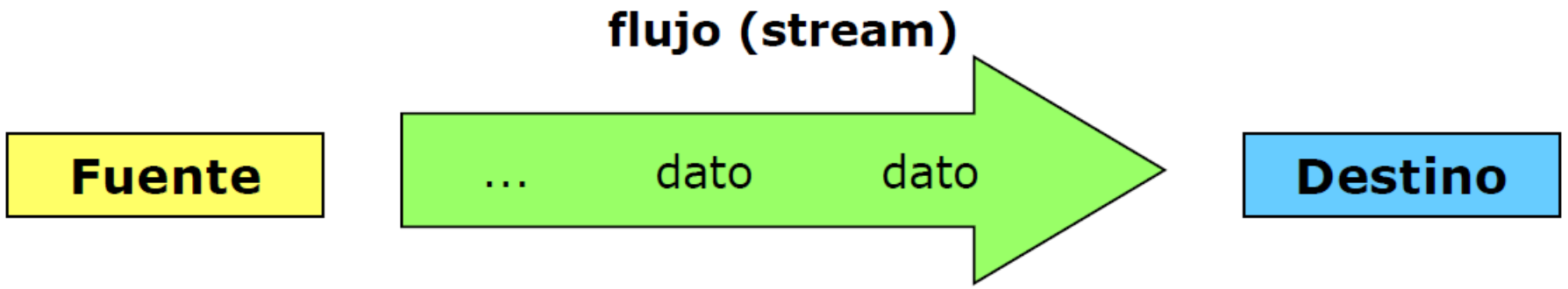
E/S con flujos (stream)

En Java se define la abstracción de stream (flujo) para tratar la comunicación de información entre el programa y el exterior.

- Entre una fuente y un destino fluye una secuencia de datos

Los flujos actúan como interfaz con el dispositivo o clase asociada

- Operación independiente del tipo de datos y del dispositivo
- Mayor flexibilidad (p.e. redirección, combinación)
- Diversidad de dispositivos (fichero, pantalla, teclado, red, …)
- Diversidad de formas de comunicación
  - Modo de acceso: secuencial, aleatorio
  - Información intercambiada: binaria, caracteres, líneas



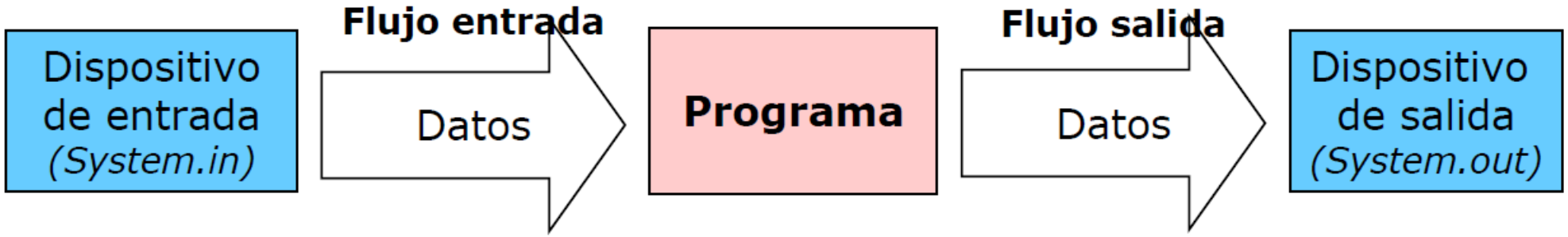
Flujos estándar

Como en Unix:

- Entrada estándar - habitualmente el teclado
- Salida estándar - habitualmente la consol
- Salida de error - habitualmente la consola

En Java se accede a la E/S estándar a través de campos estáticos de la clase java.lang.System

- System.in implementa la entrada estándar
- System.out implementa la salida estándar
- System.err implementa la salida de error



Flujos estándar

System.in

- Instancia de la clase InputStream: flujo de bytes de entrada
- Métodos:
  - read() permite leer un byte de la entrada como entero
  - skip(n) ignora n bytes de la entrada
  - available() número de bytes disponibles para leer en la entrada

System.out

- Instancia de la clase PrintStream: flujo de bytes de salida
- Metodos para impresión de datos:
  - print(), println()
  - flush() vacía el buffer de salida escribiendo su contenido

System.err

- Funcionamiento similar a System.out
- Se utiliza para enviar mensajes de error (por ejemplo a un fichero de log o a la consola)

Flujos estándar

Ejemplo:

```
import java.io.*;

class LecturaDeLinea {
        public static void main( String args[] ) throws IOException {
        int c;
        int contador = 0;
        // se lee hasta encontrar el fin de línea
        while( (c = System.in.read() ) != '\n' )
        {
                contador++;
                System.out.print( (char) c );
        }
        System.out.println(); // Se escribe el fin de línea
        System.err.println( "Contados "+ contador +" bytes en total." );
        }

}
```

Utilización de los flujos

Los flujos se implementan en las clases del paquete java.io

Esencialmente todos funcionan igual, independientemente de la fuente de datos

Clases java.io.Reader y java.io.Writer

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

int write(int aCharacter)
int write(char buffer[])
int write(char buffer[], int offset, int length)
```

Utilización de los flujos

Lectura

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

Escritura

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

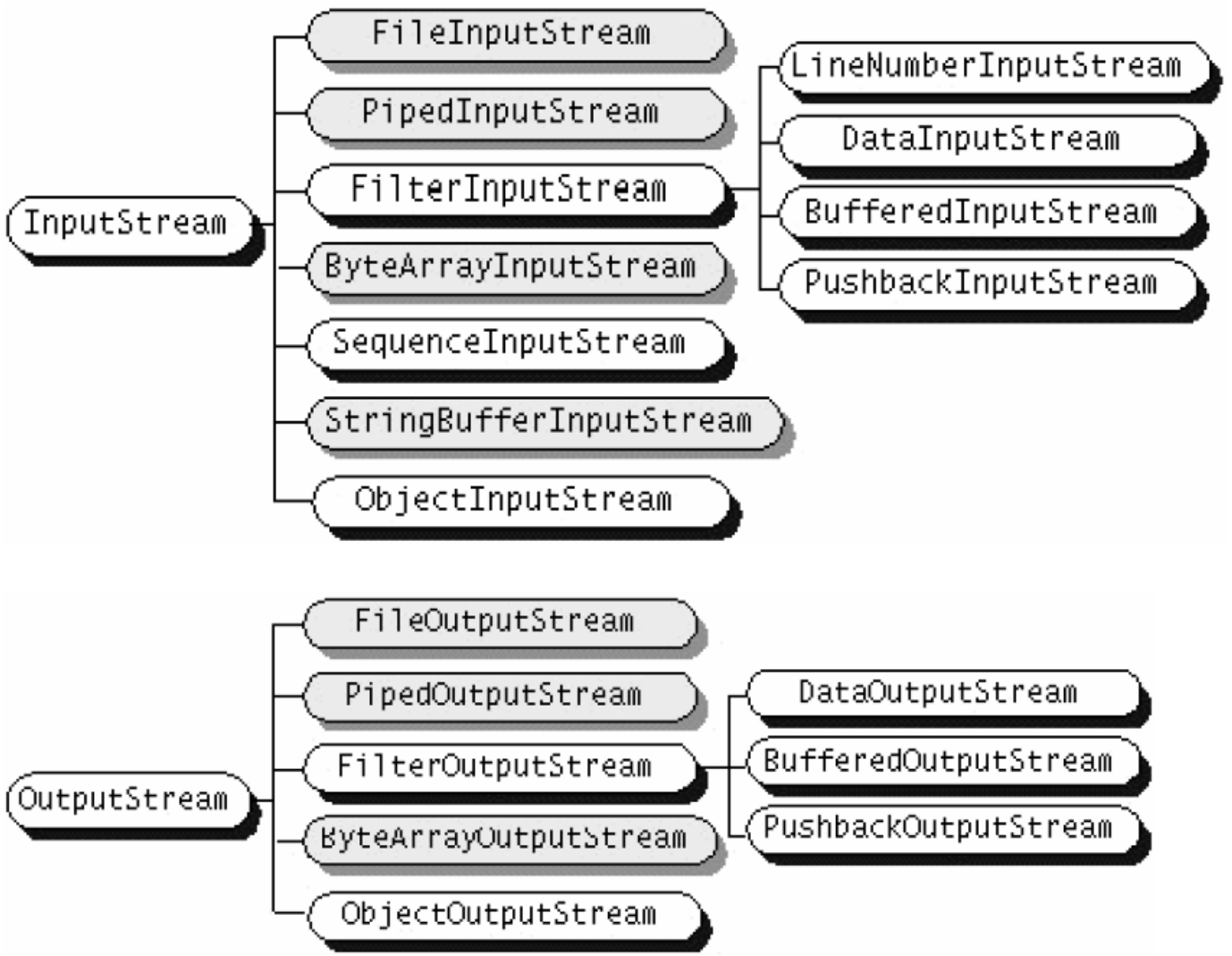
Nota: para los flujos estándar ya se encarga el sistema de abrirlos y cerrarlos

Un fallo en cualquier punto produce la excepción IOException

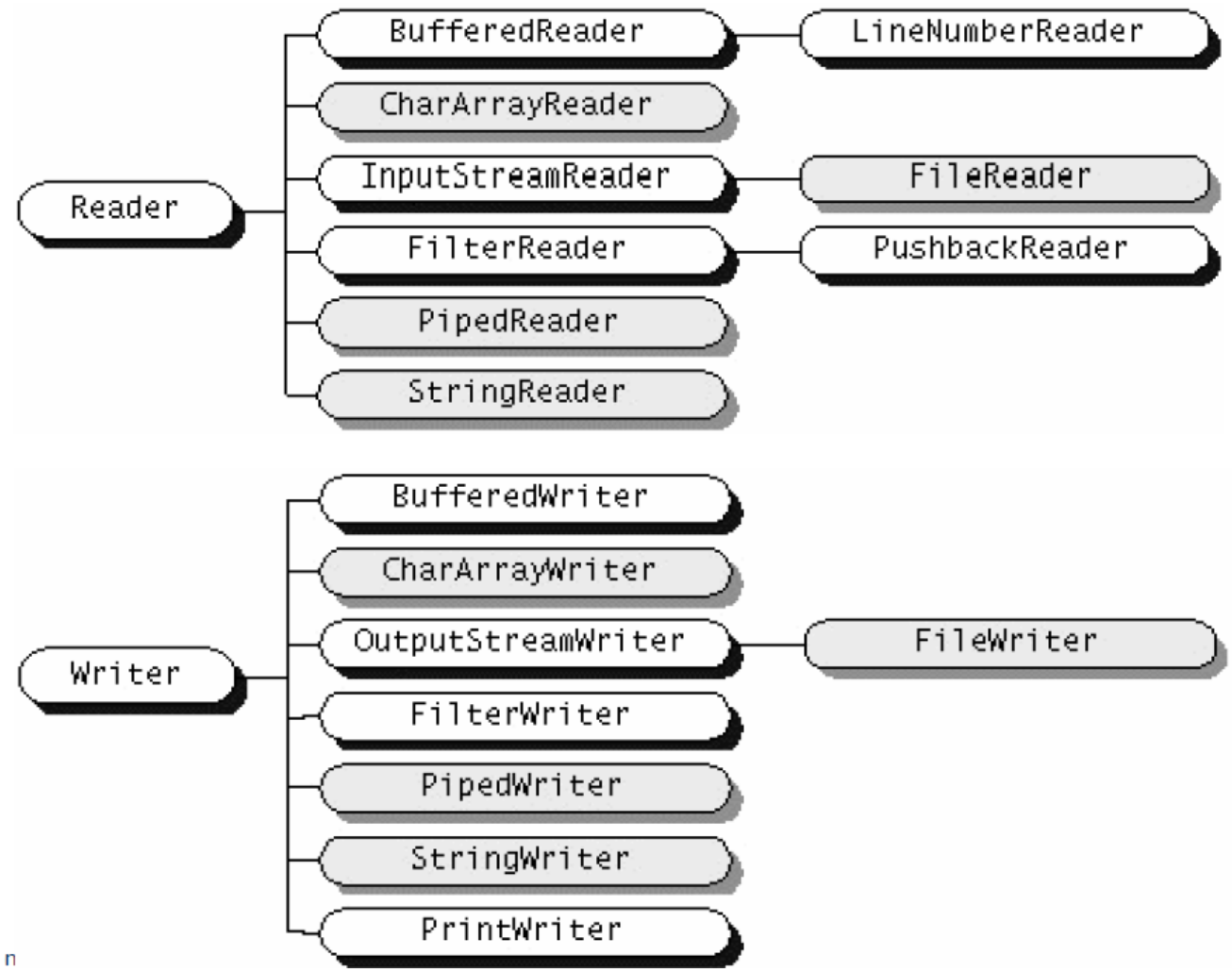
Clasificación de flujos

- Representación de la información
  - Flujos de bytes: clases InputStream y OutputStream
  - Flujos de caracteres: clases Reader y Writer
    - Se puede pasar de un flujo de bytes a uno de caracteres con InputStreamReader y OutputStreamWriter
- Propósito
  - Entrada: InputStream, Reader
  - Salida: OutputStream, Writer
  - Lectura/Escritura: RandomAccessFile
  - Transformación de los datos
    - Realizan algún tipo de procesamiento sobre los datos (p.e. buffering, conversiones, filtrados): BuffuredReader, BufferedWriter
- Acceso
  - Secuencial
  - Aleatorio - (RandomAccessFile)

Jerarquía de flujos de bytes



Jerarquía de flujos de caracteres



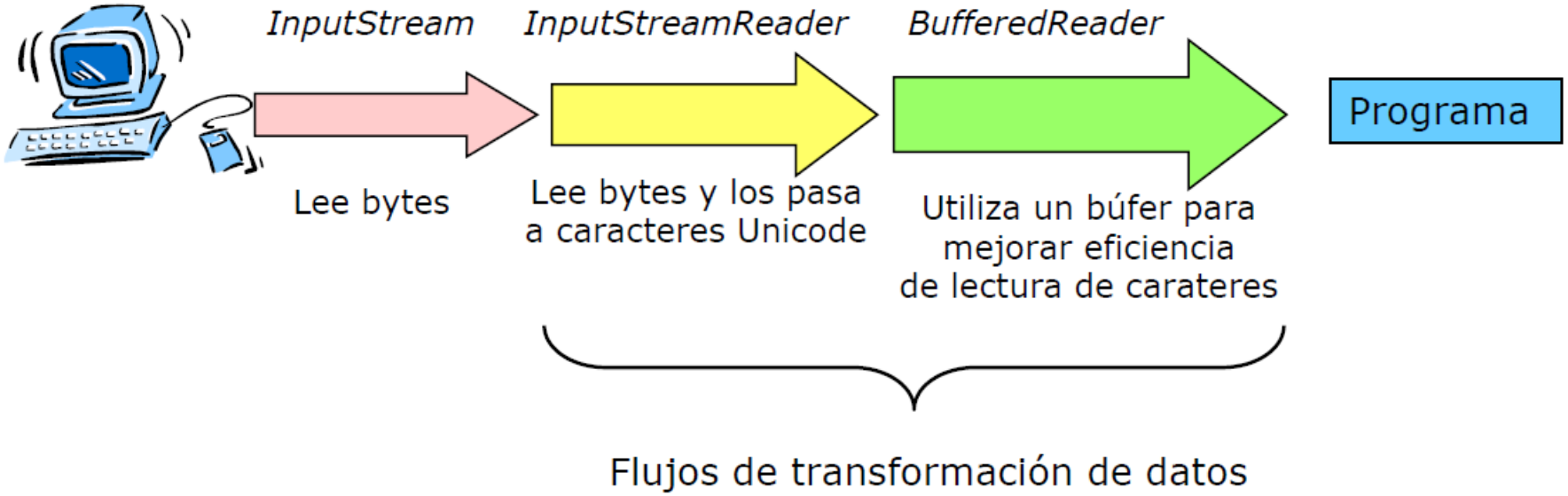
Entrada de caracteres

- InputStreamReader
  - Lee bytes de un flujo InputStream y los convierte en caracteres Unicode
  - Métodos de utilidad
    - read() lee un único carácter
    - ready() indica cuando está listo el flujo para lectura
- BufferedReader
  - Entrada mediante búfer, mejora el rendimiento
  - Método de utilidad
    - readLine() lectura de una línea como cadena

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

Combinación de flujos

Los flujos se pueden combinar para obtener la funcionalidad deseada



Combinación de flujos

Ejemplo:

```
import java.io.*;

public class Eco {
	public static void main (String[] args) {
		BufferedReader entradaEstandar = new BufferedReader
			(new InputStreamReader(System.in));

		String mensaje;

		System.out.println ("Introducir una línea de texto:");
		mensaje = entradaEstandar.readLine();
		System.out.println ("Introducido: \"" + mensaje + "\"");
	}
}
```

La clase Teclado

Ejemplo:

```
import java.io.*;

public class Teclado {
        /** variable de clase asignada a la entrada estándar del sistema */
        public static BufferedReader entrada =
                new BufferedReader(new InputStreamReader(System.in));
        /** lee una cadena desde la entrada estándar
        * @return cadena de tipo String
        */
        public static String leerString() {
                String cadena="";
                try {
                        cadena = new String(entrada.readLine());
                }catch (IOException e) {
                        System.out.println("Error de E/S"); }
                return cadena; } // la clase Teclado continua
```

La clase Teclado

```
// ...continuación de la clase teclado
/** lee un numero entero desde la entrada estandar
* @return numero entero de tipo int
*/
public static int leerInt() {
        int entero = 0;
        boolean error = false;
        do {
                try {
                        error = false;
                        entero = Integer.valueOf(entrada.readLine()).intValue();
                }catch (NumberFormatException e1) {
                        error = true;
                        System.out.println("Error en el formato del numero, intentelo de nuevo.");
                }catch (IOException e) {
                System.out.println("Error de E/S");}
        } while (error);
        return entero;
        }
} // final de la clase Teclado
```

Flujos de bytes especiales

File streams

- Para escribir y leer datos en ficheros

Object streams

- Para escribir y leer objetos
- Implementa lo que se denomina serialización de objetos (object serialization)
    - Es posible guardar un objeto con una representación de bytes

Filter streams

- Permiten filtrar datos mientras se escriben o leen
    - Se construyen sobre otro flujo
- Permiten manipular tipos de datos primitivos
- Implementan las interfaces DataInput y DataOutput y pueden heredar de las clases FilterInputStream y FilterOutputStream
    - El mejor ejemplo son las clases DataInputStream y DataOutputStream para leer y escribir datos de tipos básicos