

UT 1 - Programación multiproceso

2º DAM – Curso 2024/2025

Parte 3 – Procesos en Java

Andrés Marina Díaz
andresmd@educastur.org

Creación de procesos en Java

Creación de procesos

Clase principal abstracta: **Process** (en Java, representa un proceso iniciado mediante la ejecución de un comando de sistema).

Para llegar a ejecutarlo, interactuar con ellos, etc, necesitamos el apoyo de unas de estas clases:

- **ProcessBuilder** (más flexible, moderno, y complejo)
- **Runtime** (más simple y con métodos deprecados, más en desuso)

Creación de procesos

➤ Process:

La clase *Process* representa un proceso que se ha iniciado mediante la ejecución de un comando del sistema.

Proporciona métodos para interactuar con el proceso, como obtener su salida, manejar su entrada y esperar a que finalice.

➤ ProcessBuilder

ProcessBuilder es una clase más avanzada que permite crear y configurar un proceso antes de ejecutarlo. Proporciona una interfaz más flexible y orientada a objetos para iniciar procesos externos.

Creación de procesos

```
public class ProcessBuilder00 {
```

```
    public static void main(String[] args) throws IOException, InterruptedException {  
        // Crear el ProcessBuilder para abrir la calculadora (calc.exe)  
        ProcessBuilder pb = new ProcessBuilder("calc.exe");  
        pb.start();  
    }
```

```
}
```

```
5  
6 public class ProcessBuilder00 {
```

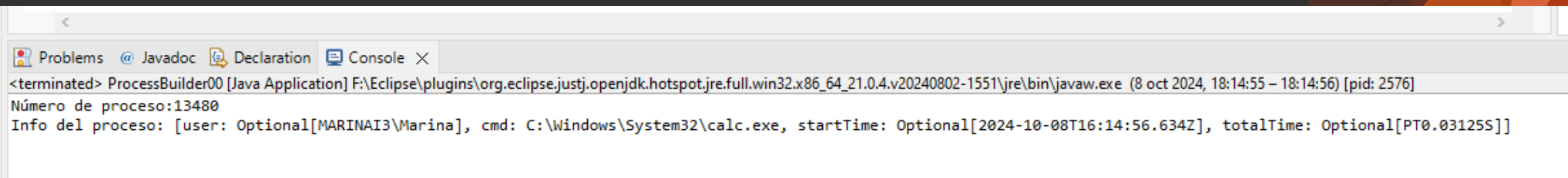
```
7  
8 public static void main(String[] args) throws IOException, InterruptedException {  
9     // TODO Auto-generated method stub  
10    // Crear el ProcessBuilder para abrir la calculadora (calc.exe)  
11    ProcessBuilder pb = new ProcessBuilder("calc.exe");  
12  
13    // Iniciar el proceso  
14    Process proceso = pb.start();  
15  
16  
17  
18 }
```

```
19  
20 }  
21
```

Métodos interesantes

➡ pid() e info():

```
5
6 public class ProcessBuilder00 {
7
8     public static void main(String[] args) throws IOException, InterruptedException {
9         // TODO Auto-generated method stub
10        // Crear el ProcessBuilder para abrir la calculadora (calc.exe)
11        ProcessBuilder pb = new ProcessBuilder("calc.exe");
12
13        // Iniciar el proceso
14        Process proceso = pb.start();
15
16        System.out.println("Número de proceso:" + proceso.pid());
17        System.out.println("Info del proceso: " + proceso.info());
18    }
19
20 }
21
22 }
```



The screenshot shows the Eclipse IDE's console window. The title bar includes tabs for Problems, Javadoc, Declaration, and Console. The console output shows the execution of the ProcessBuilder00 Java application. The first line indicates the application has terminated. The second line shows the process ID (pid) as 13480. The third line shows the process information, including the user (MARINAI3\Marina), the command (C:\Windows\System32\calc.exe), and the start and total times.

```
<terminated> ProcessBuilder00 [Java Application] F:\Eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.4.v20240802-1551\jre\bin\javaw.exe (8 oct 2024, 18:14:55 – 18:14:56) [pid: 2576]
Número de proceso:13480
Info del proceso: [user: Optional[MARINAI3\Marina], cmd: C:\Windows\System32\calc.exe, startTime: Optional[2024-10-08T16:14:56.634Z], totalTime: Optional[PT0.03125S]]
```

Métodos interesantes

- **isalive()** : Comprobar si el proceso está vivo (bool)
- **waitfor()**: Esperar a que el proceso termine (ojo con aplicaciones con GUI, puede ser impreciso)
- **destroy()**: Matar proceso

Comunicación entre procesos

Comunicación entre procesos

Un proceso recibe información, la transforma y produce resultados mediante su:

- **Entrada estándar (*stdin*):** lugar de donde el proceso lee los datos de entrada que requiere para su ejecución.
 - ❑ Normalmente es el teclado.
 - ❑ No se refiere a los parámetros de ejecución del programa.
- **Salida estándar (*stdout*):** sitio donde el proceso escribe los resultados que obtiene.
 - ❑ Normalmente es la pantalla, también puede ser un fichero
- **Salida de error (*stderr*):** sitio donde el proceso envía los mensajes de error.
 - ❑ Habitualmente es el mismo que la salida estándar, pero es más común dirigir esta salida a un fichero

Comunicación entre procesos

- En Java, el proceso hijo no tiene su propia interfaz de comunicación, por lo que el usuario **no puede comunicarse** con él directamente.
- *Stdin, stdout y stderr* están redirigidas al **proceso padre** a través de los flujos de datos siguientes.

Comunicación entre procesos

OutputStream: flujo de datos desde el proceso padre al hijo.

- El *stream* está conectado por un *pipe* a la entrada estándar (*stdin*) del proceso hijo.



Comunicación entre procesos

InputStream: flujo de datos desde el proceso hijo al padre.

- El stream está conectado por un pipe a la salida estándar (stdout) del proceso hijo.



Comunicación entre procesos

ErrorStream: flujo de error del proceso hijo.

- Similar a InputStream
- El stream está conectado por un pipe a la salida estándar (stderr) del proceso hijo. Por defecto, está conectado al mismo sitio que stdout.



Comunicación entre procesos

Métodos de java en proceso padre:

- **InputStream:** flujo entrada.
 - ❑ `getInputStream()`
- **OutputStream:** flujo salida.
 - ❑ `getOutputStream()`
- **ErrorStream:** flujo error.
 - ❑ `getErrorStream()`



Comunicación entre procesos

- Utilizando los anteriores *streams* el proceso padre puede enviarle datos al proceso hijo y recibir los resultados de salida que este genere (también comprobar errores).
- Hay que tener en cuenta que en algunos sistemas operativos el tamaño de los buffers de entrada y salida que corresponde a *stdin* y *stdout* está limitado. En este sentido, un fallo al leer o escribir en los flujos de entrada o salida del proceso hijo puede provocar que el proceso hijo se bloquee. Por eso, en Java se suele realizar la comunicación padre-hijo a través de un buffer utilizando los *streams* vistos.

Guía *InputStream*

Paso 1: Obtener el flujo de entrada (InputStream)

- ▶ Cuando se crea un proceso hijo en Java, puedes acceder a la salida estándar (stdout) de ese proceso a través del método `getInputStream()` del objeto `Process`. Esto te devuelve un objeto de tipo `InputStream`.
- ▶ **InputStream**: Este es el **flujo de bytes** que proviene del proceso hijo. Lee los datos byte por byte, lo que es eficiente para manejar archivos binarios o flujos de datos en bruto, pero no es tan cómodo cuando quieres leer texto

```
InputStream myInputStream = proceso.getInputStream();
```


Guía *InputStream*

Paso 2: Convertir el *InputStream* a caracteres (*InputStreamReader*)

- ▶ Si estás manejando texto (por ejemplo, la salida del proceso hijo), es mejor convertir **estos bytes en caracteres**. Para eso, usamos *InputStreamReader*.
- ▶ *InputStreamReader*: Es un puente entre un **flujo de bytes** (*InputStream*) y un **flujo de caracteres** (*Reader*). Convierte los bytes en caracteres basándose en la codificación de caracteres (por defecto, usa la codificación del sistema, pero puede especificarse otra).

```
InputStreamReader myinputstreamreader = new InputStreamReader(myinputStream);
```

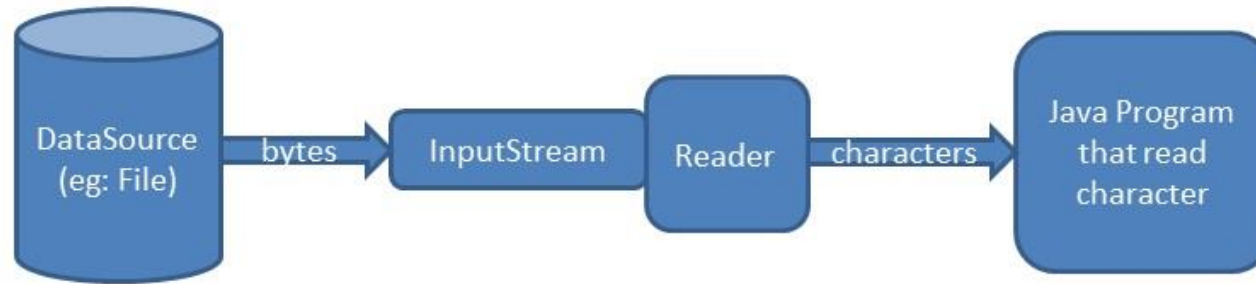
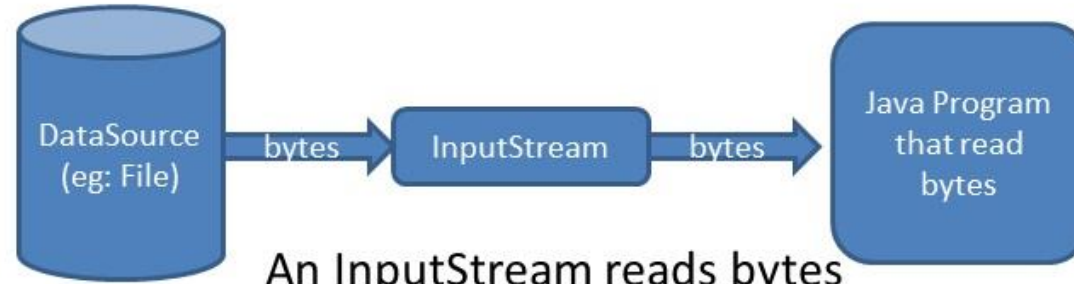
Guía *InputStream*

Paso 3: Leer de manera eficiente usando **BufferedReader**

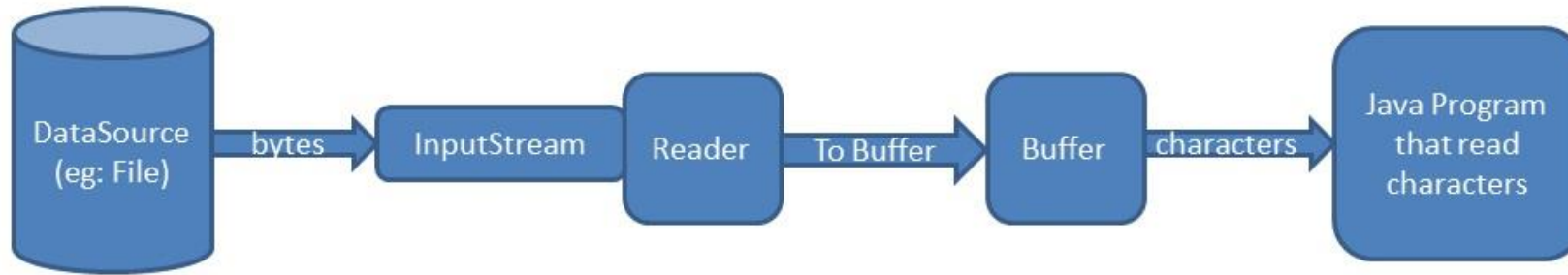
- ▶ Leer directamente desde un *InputStreamReader* implica que los datos se leen carácter por carácter, lo que puede ser ineficiente si estamos leyendo grandes cantidades de texto. Para optimizarlo, usamos ***BufferedReader***
- ▶ ***BufferedReader***: Este lector envuelve un Reader (como el *InputStreamReader*) y añade una memoria intermedia (buffer) que almacena temporalmente grandes bloques de caracteres, permitiendo leer líneas completas o grandes porciones de texto de una sola vez, lo que mejora la eficiencia.

```
BufferedReader mybufferedReader = new BufferedReader(myinputStreamReader);
```

Esquema *InputStream*



A Reader works with an InputStream and reads characters



A BufferedReader pre-fetches characters in to a buffer for performance

Ejemplo *InputStream*

Vamos a crear un programa en Java que lea la salida de un programa que hayamos creado previamente en C.

Programa en C inicial:

```
int main()  
{  
    printf("Esta es una linea de prueba\n");  
    return 0;  
}
```

Consejo: Para evitar posibles errores se puede usar `fflush` para vaciar el `stdout` del programa en C.

Ejemplo *InputStream*

```
8 public class EjemploInputReader01 {
9
10     public static void main(String[] args) throws IOException {
11
12         //Creacion de un objeto process con ProcessBuilder y nuestro ejecutable
13         Process procesohijo = new ProcessBuilder("resources/EjemploInputReader.exe").start();
14
15         //Paso 1 conseguir un objeto inputStream con el metodo getInputStream de la clase process
16         InputStream myIS = procesohijo.getInputStream();
17
18         //Paso 2: conseguir un objeto inputStreamReader utilizando el inputStream obtenido en el paso 1
19         InputStreamReader myISR = new InputStreamReader(myIS);
20
21         //Paso 3: conseguir el bufferedreader utilizando el inputStreamreader obtenido en el paso 2
22         BufferedReader myBR = new BufferedReader(myISR);
23
24         String line;
25         System.out.println("Salida del proceso hijo: ");
26
27         //Utilizando el metodo de la clase bufferedreader .readline() leer todo el contenido hasta el final
28         while((line= myBR.readLine()) != null) {
29             System.out.println(line);
30         }
31     }
```

Console x

<terminated> EjemploInputReader01 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (11 oct 2024, 13:20:47 – 13:20:48) [pid: 14644]

Salida del proceso hijo:

Esta es una línea de prueba

Ejemplo *InputStream*

```
8 public class EjemploInputReader01 {
9
10     public static void main(String[] args) throws IOException {
11
12         //Creacion de un objeto process con ProcessBuilder y nuestro ejecutable
13         Process procesohijo = new ProcessBuilder("resources/EjemploInputReader.exe").start();
14
15         //Todos los pasos a la vez
16         BufferedReader myBR = new BufferedReader(new InputStreamReader(procesohijo.getInputStream()));
17
18         String line;
19         System.out.println("Salida del proceso hijo: ");
20
21         //Utilizando el metodo de la clase bufferedreader .readline() leer todo el contenido hasta el final
22         while((line= myBR.readLine()) != null) {
23             System.out.println(line);
24         }
25     }
```

Console ×

<terminated> EjemploInputReader01 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (11 oct 2024, 13:28:40 – 13:28:41) [pid: 12464]

Salida del proceso hijo:

Esta es una linea de prueba

Guía *OutputStream*

Paso 1: Obtener el flujo de salida (*OutputStream*)

- ▶ Cuando se crea un proceso hijo en Java, **puedes enviarle datos al proceso** a través de su entrada estándar (stdin). Para hacerlo, se utiliza el método ***getOutputStream()*** del objeto *Process*. Este método devuelve un objeto de tipo *OutputStream*.
- ▶ ***OutputStream***: Es un flujo de bytes que permite al proceso padre enviar datos al proceso hijo. Estos datos se escriben como bytes, lo que es útil para escribir archivos binarios o flujos en bruto, pero no es tan conveniente para escribir texto.

```
OutputStream myOutputStream = proceso.getOutputStream();
```

Guía *OutPutStream*

Paso 2: Convertir el *OutputStream* para escribir texto (*PrintStream*)

- Si estás enviando texto al proceso hijo (por ejemplo, comandos o datos), es más fácil usar un objeto que **simplifique la escritura de texto**. Para esto, puedes usar *PrintStream*.
- ***PrintStream***: Es un tipo de *OutputStream* **especializado para escribir datos en formato de texto** (cadenas, números, etc.) de una manera **más cómoda**. Ofrece métodos como `print()` y `println()` para escribir texto de forma directa, en lugar de preocuparte por convertir los datos a bytes

```
PrintStream myprintStream = new PrintStream(myoutputStream);
```


Guía *OutputStream*

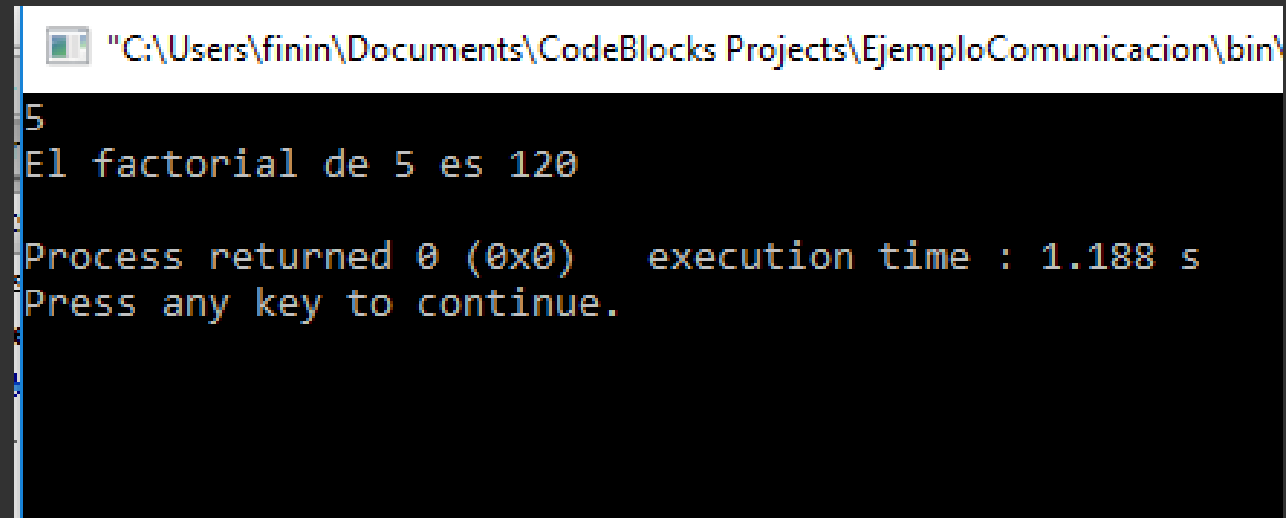
Paso 3: Asegurarte de que los datos se envíen (*flush*)

- Los datos que escribes en un flujo (*OutputStream* o *PrintStream*) **no se envían inmediatamente**; en lugar de eso, se almacenan temporalmente en un buffer de salida. Para asegurarte de que los datos se envíen de inmediato al proceso hijo, **puedes usar el método *flush()***.
- *flush()*: Este método **vacía el buffer y envía todos los datos pendientes** que todavía no han sido escritos al flujo. Es necesario cuando quieres que los datos lleguen al proceso hijo sin retrasos.

myprintStream.flush();

Ejemplo *OutputStream*

Vamos a crear un programa en Java que mande un entero a un programa C, el cual calculará su factorial y devolverá un texto como respuesta.



```
"C:\Users\finin\Documents\CodeBlocks Projects\EjemploComunicacion\bin\
5
El factorial de 5 es 120
Process returned 0 (0x0)   execution time : 1.188 s
Press any key to continue.
```

Ejemplo *OutputStream*

```
12 public static void main(String[] args) throws IOException {  
13  
14     Process procesohijo = new ProcessBuilder("resources\\factorial.exe").start();  
15  
16     //Establecemos el bufferedreader para la lectura del resultado del proceso hijo  
17     BufferedReader myBR = new BufferedReader(new InputStreamReader(procesohijo.getInputStream()));  
18  
19     //Establecemos el printStream para enviar datos al proceso hijo  
20     PrintStream myPS = new PrintStream(procesohijo.getOutputStream());  
21  
22     Scanner reader = new Scanner(System.in);  
23     int numero = 0;  
24     numero = reader.nextInt();  
25  
26     //Introducir el numero en el stream y utilizar el flush para enviarlo inmediatamente  
27     myPS.println(numero);  
28     myPS.flush();  
29  
30     //Código para la lectura de la respuesta del proceso hijo  
31     String line;  
32     System.out.println("Salida del proceso hijo: ");  
33     while((line= myBR.readLine()) != null) {  
34         System.out.println(line);  
35     }
```

Console ×

<terminated> ComunicacionEntreProcesos2 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (11 oct 2024, 14:08:15 – 14:08:17) [pid: 9172]

5

Salida del proceso hijo:
El factorial de 5 es 120

DUDAS, PREGUNTAS, INQUIETUDES

