3. Creación de procesos con Java

- Java dispone en el paquete java.lang de varias clases para la gestión de procesos.
- Una de ellas es la clase **ProcessBuilder**. Cada instancia de **ProcessBuilder** gestiona una colección de atributos del proceso.
- El método **start()** crea una nueva instancia de **Process** con esos atributos y puede ser invocado varias veces desde la misma instancia para crear nuevos subprocesos con atributos inéditos o relacionados.

3. Creación de procesos con Java

- La clase **Process** proporciona métodos para realizar la entrada desde el proceso, obtener la salida del proceso, esperar a que el proceso se complete, comprobar el estado de salida del proceso y destruir el proceso.

3. Creación de procesos con Java

Algunos métodos importantes de **Process** son:

MÉTODOS	MISIÓN
InputStream getInputStream ()	Devuelve el flujo de entrada conectado a la salida normal del subproceso. Nos permite leer el stream de salida del subproceso, es decir, podemos leer lo que el comando que ejecutamos escribió en la consola.
int waitFor ()	Provoca que el proceso actual espere hasta que el subproceso representado por el objeto **Process** finalice. Devuelve 0 si ha finalizado correctamente.
InputStream getErrorStream()	Devuelve el flujo de entrada conectado a la salida de error del subproceso. Nos va a permitir poder leer los posibles errores que se produzcan al lanzar el subproceso.
OutputStream getOutputStream()	Devuelve el flujo de salida conectado a la entrada normal del subproceso. Nos va a permitir escribir en el stream de entrada del subproceso, así podemos enviar datos al subproceso que se ejecute.
void destroy ()	Elimina el subproceso.
int exitValue ()	Devuelve el valor de salida del subproceso.
boolean isAlive ()	Comprueba si el subproceso representado por **Process** está vivo

3. Creación de procesos con Java

- **Por defecto el proceso (o subproceso) que se crea no tiene su propia terminal o consola**. Todas las operaciones de E/S serán redirigidas al proceso padre, donde se puede acceder a ellas usando los métodos **getOutputStream(), getInputStream() y getErrorStream().**
- El proceso padre utiliza estos flujos para alimentar la entrada y obtener la salida del subproceso. En algunas plataformas se pueden producir bloqueos en el subproceso debido al tamaño del buffer limitado para los flujos de E/S estándar.

3. Creación de procesos con Java

Cada constructor de **ProcessBuilder** gestiona los siguientes atributos de un proceso:

- **Un comando**. Es unas lista de cadenas que representa el programa que se invoca y sus argumentos si los hay.
- **Un entorno** (environment) con sus variables.
- **Un directorio de trabajo**. El valor por defecto es el directorio de trabajo del proceso en curso.

3. Creación de procesos con Java

- **Una fuente de entrada estándar**. Por defecto, el subproceso lee la entrada de una tubería. El código Java puede acceder a esta tubería a través de la secuencia de salida devuelta por **Process.getOutputStream().** Sin embargo, la entrada estándar puede ser redirigida a otra fuente con **redirectInput().**
- **Un destino para la salida estándar y la salida de error**. Por defecto, el subproceso escribe en las tuberías de la salida y el error estándar. El código Java puede acceder a estas tuberías a través de los flujos de entrada devueltos por **Process.getInputStream()** y **Process.getErrorStream().** Igual que antes, la salida estándar y el error estándar pueden ser redirigido a otros destinos utilizando **redirectOutput**() y **redirectError**().

3. Creación de procesos con Java

- Una propiedad **redirectErrorStream**. Inicialmente, esta propiedad es false, significa que la salida estándar y salida de error de un subproceso se envían a dos corrientes separadas, que se pueden acceder a través de los métodos **Process.getInputStream**() y **Process.getErrorStream**().

3. Creación de procesos con Java

Algunos de los métodos de la clase **ProcessBuilder** son:

MÉTODOS	MISIÓN
ProcessBuilder command (String argumentos ...)	Define el programa que se quiere ejecutar indicando sus argumentos como una lista de cadenas separadas por comas.
List < String > command ()	Devuelve todos los argumentos del objeto **ProcessBuilder**.
Map < String , String > environment ()	Devuelve en una estructura Map las variables de entorno del objeto **ProcessBuilder**.
ProcessBuilder redirectError (File file)	Redirige la salida de error estándar a un fichero.
ProcessBuilder redirectInput (File file)	Establece la fuente de entrada estándar en un fichero.
ProcessBuilder redirectOutput (File file)	Redirige la salida estándar a un fichero.
File directory()	Devuelve el directorio de trabajo del objeto **ProcessBuilder**.
ProcessBuilder directory(File directorio)	Establece el directorio de trabajo del objeto **ProcessBuilder**.
Process start ()	Inicia un nuevo proceso utilizando los atributos del objeto **ProcessBuilder**.

3. Creación de procesos con Java

Veamos el Ejemplo01 donde un proceso creado en Java ejecuta y abre una aplicación instalada en el sistema operativo.

Para iniciar un nuevo proceso que utiliza el directorio de trabajo y el entorno del proceso en curso escribimos la siguiente orden:

```
Process p = new
ProcessBuilder("Comando","Argum1").start();
```

Más información consulta la documentación del API de la clase ProcessBuilder y Process

3. Creación de procesos con Java

Por ejemplo, para ejecutar el comando DIR de DOS podemos escribir lo siguiente:

```
Process pb = new
ProcessBuilder("CMD","/C","DIR").start();
```

Para los comandos de Windows que no tienen ejecutable (ejemplo DIR) es necesario utilizar el comando CMD.EXE. Entonces para hacer un DIR desde un programa Java tendríamos que construir un objeto ProcessBuilder con los siguientes argumentos: "CMD", "C/" y "DIR". CMD inicia una nueva instancia del intérprete de comandos Windows. Para ejecutar comandos de Windows escribimos:

- CMD /C comando: ejecuta el comando especificado y luego finaliza.
- CMD /K comando: ejecuta el comando especificado, pero sigue activo.

3.1. Procesos con Java.
Lectura salida proceso

- Utilizaremos el método **getInputStream**() de la clase **Process** para leer el stream de salida del proceso, es decir, para leer lo que el comando envía a la consola. Definiremos así el stream:

```
InputStream is = p.getInputStream();
```

Para leer la salida usamos el método **read**() de **InputStream** donde nos devolverá carácter a carácter la salida generada por el comando.

3.2. Procesos con Java.
Lectura salida proceso

El método **waitFor**() hace que el proceso actual espere hasta que el subproceso representado por el objeto **Process** finalice. Este método recoge lo que **System.exit**() devuelve, **por defecto** en un programa Java si no se incluye esta orden el valor devuelto **es 0**, que normalmente responde a una finalización correcta del proceso. Podemos nosotros poner el valor entero que consideremos para indicar que ha finalizado ok o ko. **Lo habitual es 0 para ok y otro número para ko.**

Veamos un Ejemplo02 de Java donde se ejecutan comandos y se muestra por pantalla la salida utilizando estos métodos.

3.3. Procesos con Java.
Proceso lanza otro proceso Java

Veamos un Ejemplo03 donde muestra un programa Java que ejecuta el programa Java anterior, el programa se ejecutará desde el entorno de Eclipse.

- *El proceso a ejecutar se encuentra en la carpeta bin del proyecto y en un paquete en concreto, por ello será necesario crear un objeto* ***File*** *que referencie a dicho directorio.*
- *Después para establecer el directorio de trabajo para el proceso que se va a ejecutar se debe usar el método* ***directory****(), a continuación se ejecutará el proceso.*
- *Por último será necesario recoger el resultado de salida usando el método* ***getInputStrem()*** *del proceso.*

3.4. Procesos con Java.
Captura de errores con Java

La clase **Process** tiene el método **getErrorStream**() que nos va a permitir obtener un stream para poder leer los posibles errores que se produzcan al lanzar el proceso.

Veamos un Ejemplo02Error de Java donde se ejecuta el comando ps (o DIR en Windows) de manera errónea y se muestra por pantalla el error.

3.4. Procesos con Java.

Enviar datos al stream de entrada del proceso con Java

La clase **Process** tiene el método **getOutputStream**() que nos permite escribir en el stream de entrada del proceso, así podemos enviarle datos.

El método **write**() envía los bytes al stream, el método **getBytes**() codifica la cadena en una secuencia de bytes que utilizan juego de caracteres por defecto de la plataforma.

3.4. Procesos con Java.

Enviar datos al stream de entrada del proceso con Java

Veamos un Ejemplo05 de Java donde se envía una cadena de texto como datos de entrada a otro programa. Cada línea de texto que enviamos debe finalizar en \n.

3.5. Procesos con Java.
Variables de entorno - procesos con Java

La clase **ProcessBuilder** tiene el método **environment**() que devuelve las variables de entorno del proceso; el método **command**() sin parámetros, devuelve los argumentos del proceso definido en el objeto **ProcessBuilder**; y con parámetros donde se define un nuevo proceso y sus argumentos como hemos usado en otros ejemplos previos.

Veamos el Ejemplo06 de Java donde se utilizan estos métodos

3.5. Procesos con Java.

Redireccionamiento de entrada y salida - procesos con Java

La clase **ProcessBuilder** tiene los métodos **redirectOutput**() y **redirectError**() que nos permiten redirigir la salida estándar y de error a un fichero.

Veamos el Ejemplo07 de Java donde se utilizan estos métodos

3.5. Procesos con Java.

Redireccionamiento de entrada y salida - procesos con Java

La clase **ProcessBuilder** tiene el método **redirectInput()** para indicar que la entrada al proceso se encuentra en un fichero.

Podemos ejecutar varios comandos del sistema operativo dentro de un fichero BAT e indicar al proceso que la entrada de datos está en dicho fichero.

Veamos el Ejemplo08 de Java donde se utilizan estos métodos

3.5. Procesos con Java.

Redireccionamiento de entrada y salida - procesos con Java

Para llevar a cabo el redireccionamiento, tanto de entrada como de salida del proceso que se ejecuta, podemos usar la clase **ProcessBuilder.Redirect.** El redireccionamiento puede ser uno de los siguientes:

- El valor especial **Redirect.INHERIT**, indica que la fuente de entrada y salida del proceso será la misma que la del proceso actual.
- **Redirect.from(File)**, indica redirección para leer de un fichero, la entrada al proceso se encuentra en el objeto **File**.

3.5. Procesos con Java.

Redireccionamiento de entrada y salida - procesos con Java

- **Redirect.to(File),** indica redirección para escribir en un fichero, el proceso escribirá en el objeto **File** especificado.
- **Redirect.appendTo(File)**, indica redirección para añadir a un fichero, la salida del proceso se añadirá al objeto **File** especificado.

- *Veamos el Ejemplo08_2 de Java donde se utilizan los métodos Redirect.from(File), Redirect.To(File) y Redirect.appendTo(File).*
- *Veamos el Ejemplo 09 de Java para ver Redirect.INHERIT*