

BOLETÍN 1 DE PROBLEMAS DE PROCESOS

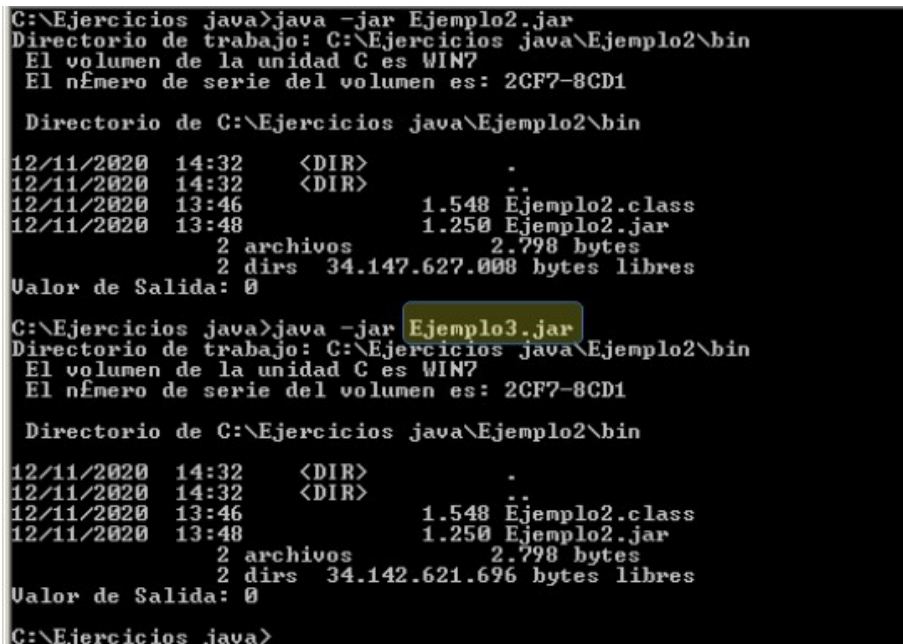
Problema 1

Crea una clase que ejecute el comando DIR y muestre el resultado por la salida estandar.
Usar el método `getInputStream()` de la clase `Process` para leer el stream de salida del proceso (leer lo que el comando envía a la consola)

Definiremos el stream como: `InputStream is = p.getInputStream();`

Para leer la salida usar el método `read()` de `InputStream`.

Crea el ejecutable (.exe) para ser ejecutado en windows. Modifica el comando `dir` por `ls` para crear el ejecutable (.jar) y ejecutarlo desde linux.



```
C:\Ejercicios java>java -jar Ejemplo2.jar
Directorio de trabajo: C:\Ejercicios java\Ejemplo2\bin
El volumen de la unidad C es WIN7
El número de serie del volumen es: 2CF7-8CD1

Directorio de C:\Ejercicios java\Ejemplo2\bin
12/11/2020  14:32    <DIR>          .
12/11/2020  14:32    <DIR>          ..
12/11/2020  13:46                1.548 Ejemplo2.class
12/11/2020  13:48                1.250 Ejemplo2.jar
                2 archivos                2.798 bytes
                2 dirs 34.147.627.000 bytes libres
Valor de Salida: 0

C:\Ejercicios java>java -jar Ejemplo3.jar
Directorio de trabajo: C:\Ejercicios java\Ejemplo2\bin
El volumen de la unidad C es WIN7
El número de serie del volumen es: 2CF7-8CD1

Directorio de C:\Ejercicios java\Ejemplo2\bin
12/11/2020  14:32    <DIR>          .
12/11/2020  14:32    <DIR>          ..
12/11/2020  13:46                1.548 Ejemplo2.class
12/11/2020  13:48                1.250 Ejemplo2.jar
                2 archivos                2.798 bytes
                2 dirs 34.142.621.696 bytes libres
Valor de Salida: 0

C:\Ejercicios java>
```

Problema 2

Crea un programa java que ejecuta el programa del ejercicio anterior. Como el proceso a ejecutar se encuentra en la carpeta bin del proyecto 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 y por último será necesario recoger el resultado de salida usando el método `getInputStream()` del proceso:

```
import java.io.*;
public class Ejercicio2 {

    public static void main (String[] args) throws IOException{
        //Creamos objeto file al directorio donde está Ejercicio2

        //El proceso a ejecutar es Ejercicio1
    }
}
```

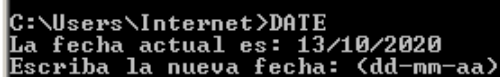
```
        //se establece el directorio donde se encuentra el ejecutable  
        // se ejecuta el proceso  
        // Obtenemos la salida devuelta por el proceso  
    }  
}
```

La salida mostrará los ficheros y carpetas del directorio definido en la variable directorio. Si ambos ficheros están en la misma carpeta o directorio , no será necesario establecer el directorio de trabajo para el objeto ProcessBuilder. Si el Ejemplo2 a ejecutar se encontrase en la carpeta D:\\PSP, tendríamos que definir el objeto directorio de la siguiente manera: *File directorio =new File("D:\\PSP")*

ENVIAR DATOS AL STREAM DE ENTRADA DEL PROCESO

Problema 3

Deseamos ejecutar un proceso que necesita información de entrada. Por ejemplo, si ejecutamos DATE desde la línea de comandos y pulsamos la tecla [Intro] nos pide escribir una nueva fecha



```
C:\Users\Internet>DATE  
La fecha actual es: 13/10/2020  
Escriba la nueva fecha: <dd-mm-aa>
```

La clase Process posee el método **getOutputStream()** que nos permite escribir en el stream de entrada del proceso , así podemos enviarle datos. El siguiente ejemplo ejecuta el comando DATE y le da los valores 15-06-18 . Con el método **write()** se 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.

Con **getErrorStream()** nos permite obtener un stream para poder leer los posibles errores que se produzcan al lanzar el proceso.

```
import java.io.*;  
  
public class Ejemplo3{  
  
    public static void main(String[] args) throws IOException {
```

```
// escritura -- envia entrada a DATE

// vacía el buffer de salida

// lectura -- obtiene la salida de DATE

// Comprobación de error -0 bien -1 mal

}
```

Problema 4

Ejemplo de aplicación que lee una cadena desde la entrada estándar y la visualiza.

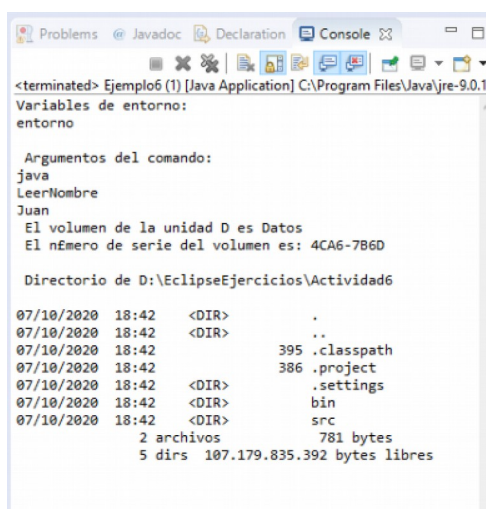
Con el método `getOutputStream()` podemos enviar datos a la entrada estándar del programa. Supongamos que queremos enviar la cadena “Hola Juan”

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

Problema 5

Crea un programa que use varios métodos de la clase `ProcessBuilder`:

- **`environment()`** que devuelve las variables de entorno del proceso; el método
- **`command()`** sin parámetros, que devuelve los argumentos del proceso definido en el objeto `ProcessBuilder`; y con parámetros donde se define un nuevo proceso y sus argumentos.



```
<terminated> Ejemplo6 (1) [Java Application] C:\Program Files\Java\jre-9.0.1\
Variables de entorno:
entorno

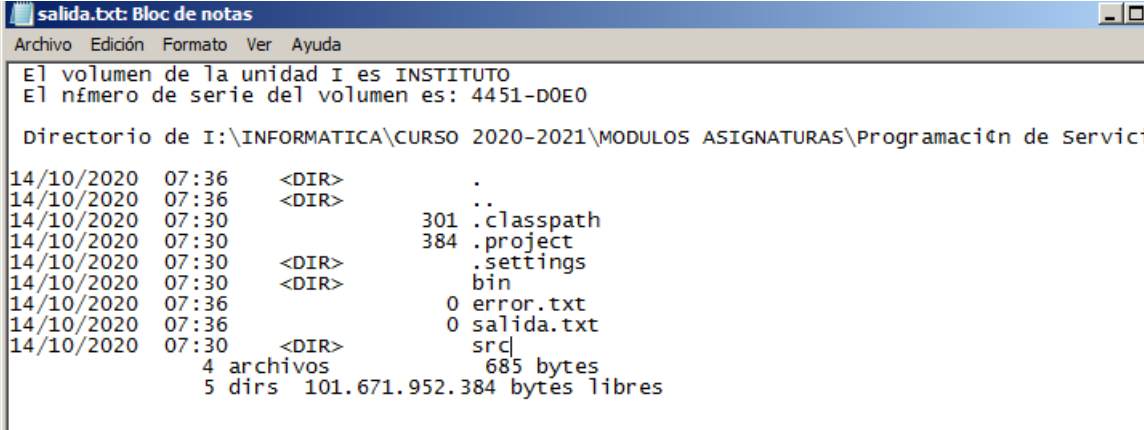
Argumentos del comando:
java
LeerNombre
Juan
El volumen de la unidad D es Datos
El nEmero de serie del volumen es: 4CA6-7B6D

Directorio de D:\EclipseEjercicios\Actividad6
07/10/2020 18:42 <DIR>      .
07/10/2020 18:42 <DIR>      ..
07/10/2020 18:42          395 .classpath
07/10/2020 18:42          386 .project
07/10/2020 18:42 <DIR>      .settings
07/10/2020 18:42 <DIR>      bin
07/10/2020 18:42 <DIR>      src
2 archivos          781 bytes
5 dirs 107.179.835.392 bytes libres
```

Problema 6

REDIRECCIONANDO LA ENTRADA Y LA SALIDA

Los métodos `redirectOutput()` y `redirectError()` nos permiten redirigir la salida estándar y de error a un fichero . En el siguiente ejercicio se ejecuta el comando `DIR` y envía la salida al fichero `salida.txt` , si ocurre algún error se envía a `error.txt`



```
salida.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
El volumen de la unidad I es INSTITUTO
El número de serie del volumen es: 4451-D0E0

Directorio de I:\INFORMATICA\CURSO 2020-2021\MODULOS ASIGNATURAS\Programación de Servi
14/10/2020 07:36 <DIR> .
14/10/2020 07:36 <DIR> ..
14/10/2020 07:30 301 .classpath
14/10/2020 07:30 384 .project
14/10/2020 07:30 <DIR> .settings
14/10/2020 07:30 <DIR> bin
14/10/2020 07:36 0 error.txt
14/10/2020 07:36 0 salida.txt
14/10/2020 07:30 <DIR> src
4 archivos 685 bytes
5 dirs 101.671.952.384 bytes libres
```

Problema 7

También podemos ejecutar varios comandos del sistema operativo dentro de un fichero `.BAT` Este ejemplo ejecuta los comandos MS-DOS que se encuentran en el fichero “fichero.bat” . Se utiliza el método `redirectInput()` para indicar que la entrada al proceso se encuentra en un fichero, es decir la entrada para el comando `CMD` será el fichero `bat` . La salida del proceso se envía al fichero `salida.txt` y la salida de error al fichero `error.txt`

Para llevar a cabo el redireccionamiento, tanto de entrada como de salida del proceso que se ejecuta , también 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`.
- **`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.

Realiza el ejercicio usando esta clase (`ProcessBuilder`) y sin usarla.

Problema 8:

En este ejercicio se debe desarrollar una solución multiproceso al problema de sincronizar y comunicar dos procesos hijos creados a partir de un proceso padre.

Se debe de escribir una clase Java que ejecute dos comandos (cada hijo creado ejecutará uno de ellos) con sus respectivos argumentos y redireccione la salida estándar del primero a la entrada estándar del segundo. Por sencillez, los comandos y sus argumentos irán directamente escrito en el código del programa.

Problema 9

Escribe una clase llamada Ejecuta que reciba como argumentos el comando y las opciones del comando que se quiere ejecutar. El programa debe de crear un proceso hijo que ejecute el comando con las opciones correspondiente mostrando un mensaje de error en el caso de que no se realizase correctamente la ejecución. El padre debe de esperar a que el hijo termine de informar si se produjo alguna anomalía en la ejecución del hijo.

Problema 10

Escribe un programa “Aleatorio” que realice lo siguiente:

- Cree un proceso hijo que está encargado de generar números aleatorios. Este proceso hijo escribirá en su salida estándar un número aleatorio del 0 al 10 cada vez que reciba una petición de ejecución por parte del padre.

Nota: no es necesario utilizar JNI , solamente crear un ejecutable y llamar correctamente al mismo desde Java.

- El proceso padre lee líneas de la entrada estándar y por cada línea que lea solicitará al hijo que le envíe un número aleatorio , lo leerá y lo imprimirá en pantalla.
- Cuando el proceso padre reciba la palabra “fin” , finalizará la ejecución del hijo y procederá a finalizar su ejecución.

Ejemplo de ejecución:

ab (enter)

7

abcdef (enter)

1

Pepe (enter)

6

fin (enter)

Problema 11

Escribe un clase llamada Mayusculas que haga lo siguiente:

- Cree un proceso hijo
- El proceso padre y el proceso hijo se comunicarán de forma bidireccional utilizando streams.

- El proceso padre leerá líneas de su entrada estándar y las enviará a la entrada estándar del hijo (utilizando el OutputStream del hijo)
- El proceso hijo leerá el texto por su entrada estándar, lo transformará todo a letras mayúsculas y lo imprimirá por su salida estándar .
- El padre imprimirá en pantalla lo que recibe del hijo a través del InputStream del mismo.

Ejemplo de ejecución

hola (enter)

HOLA

mundo (enter)

MUNDO

Problema 12

Crea un programa Java llamado LeerNombre.java que reciba desde los argumentos de main() un nombre y lo visualice en pantalla. Utiliza System.exit(1) para una finalización correcta del programa y System.exit(-1) para el caso que no se haya introducido los argumentos correctos en main().

Problema 13

Realiza un programa parecido al **Problema 3.java** para ejecutar LeerNombre.java . Utiliza el método waitfor() para comprobar el valor de salida del proceso que se ejecuta. Prueba la ejecución del programa dando valor a los argumentos de main() y sin darle valor. ¿Qué valor devuelve waitfor() en un caso y en otro?

Problema 14

Partiendo del **Problema 3.java** , muestra los errores que se producen al ejecutar un programa java que no exista.

Problema 15

Escribe un programa Java que lea dos números desde la entrada estándar y visualice su suma. Controlar que lo introducido por teclado sean dos números. Haz otro programa Java para ejecutar el anterior.

Clase ProcessBuilder

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 .

Nota: La Interface **Map** (java.io.Map) en Java, nos permite representar una estructura de datos para almacenar pares "**clave/valor**"; de tal manera que para una clave solamente tenemos un valor. Esta estructura de datos también es conocida en otros lenguajes de programación como "**Diccionarios**"

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

- **Un comando** . Es una 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.
- **Una fuente de entrada estándar** . Por defecto, el subprocesso 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()** . En este caso, **Process.getOutputStream()** devolverá una secuencia de salida nulo.
- **Un destino para la salida estándar y la salida de error**. Por defecto , el subprocesso 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()** . En este caso,

Process.getInputStream() y/o **Process.getErrorStream()** devuelven una secuencia de entrada nula.

- **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()**

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” , “Argumento1”).start();

EJERCICIOS COMPLEMENTARIOS PARA SUBIR NOTA

Problema 1:

Realizar un programa que cree cuatro procesos A, B, C y D de forma que A sea padre de B, B sea padre de C y C sea padre de D

Problema 2

Realiza un programa "copiaConc" al que se le pase una lista de nombres de archivos y para cada archivo "f" cree un nuevo proceso que se encargue de copiar dicho archivo a "f.bak"

(Nota: Para realizar este ejercicio debe de dominar el concepto de archivos)

Problema 3

Realiza un programa "ejecuta" que lea de la entrada estándar el nombre de un programa y cree un proceso hijo para ejecutar dicho programa.

Problema 4

Realice un programa "aviso" que reciba como argumentos un entero "n" y el nombre de un archivo "f" de forma que cada "n" segundos compruebe si f ha sido modificado.

Nota: Este programa puede ser lanzado en segundo plano para saber cuando un usuario linux recibe correo (fichero *var/mail/usuario*)

Problema 5

Modificar el programa anterior (llamalo ejeTemp) para que ejecute cada "n" segundos el programa apuntado por "f".

Problema 6

Realiza un programa que cree dos procesos, de forma que en el proceso padre se le pida al usuario un mensaje por la entrada estándar. Cuando el usuario escriba dicho mensaje, se enviará mediante una tubería al proceso hijo, el cual se encargará de imprimirlo por la salida estándar.

Problema 7

Escriba un programa que cree tres nuevos procesos y los comunique por tuberías para ejecutar la siguiente línea de comandos

```
$> who -u | grep old | sort -r
```

Notas:

- No se puede usar llamada al sistema para resolver el ejercicio. Una llamada al sistema es un método o función que puede invocar un proceso para solicitar un cierto servicio al sistema operativo

- No se puede dejar procesos huérfanos
- Ejemplos de llamadas al sistema de linux y windows

Tipo de system call	Función	Linux	Windows
Control del proceso	Crear proceso	fork()	CreateProcess()
Control del proceso	Terminar proceso	exit()	ExitProcess()
Gestión de archivos	Crear/abrir archivo	open()	CreateFile()
Gestión de archivos	Leer archivo	read()	ReadFile()
Gestión de archivos	Editar archivo	write()	WriteFile()
Gestión de archivos	Cerrar archivo	close()	CloseHandle()
Gestión de dispositivos	Abrir dispositivo	read()	ReadConsole()
Gestión de dispositivos	Cerrar dispositivo	close()	CloseConsole()
Gestión de la información	Definición de un intervalo de tiempo específico	alarm()	SetTimer()
Gestión de la información	Pausa (por ejemplo, de un proceso)	sleep()	Sleep()
Comunicación	Crear Pipe (memoria intermedia para el flujo de datos entre dos procesos)	pipe()	CreatePipe()
Comunicación	Creación de una memoria compartida (Shared memory)	shmget()	CreateFileMapping()