Procesos en Java

Java permite codificar programas capaces de trabajar con **procesos**. En concreto, permite hacer esto de tres formas diferentes:

- Llamar a **procesos java** creados por nosotros.
- Llamar a **procesos de Windows/Linux** (como el Notepad, pro ejemplo).
- Llamar a **comandos de la Windows** (como ipconfig).

Sincronización de procesos en Java

Java permite trabajar con procesos mediante una serie de clases y métodos. A modo de resumen, son los siguientes:

Mecanismo	Clase	Método
Ejecuta proceso	Runtime	exec ()
Ejecuta proceso	ProcessBuilder	start ()
Termina proceso	System	exit (valor)
Espera finalización proceso	Process	waitFor()

Un **ejemplo sencillo** de una clase que lanza un proceso secundario sería el siguiente. En primer lugar, creamos una clase normal **SecondaryProcess** de java con su correspondiente método **main**. Esta clase de prueba no va a tener código, pero supondremos que en alguna parte generará un error. Esto impedirá que el proceso finalice con éxito. Para comunicárselo de forma adecuada al **PrimaryProcess**, le enviará un código de error. En nuestro ejemplo, es el **103**. Si hubiese finalizado con éxito, le habríamos enviado un **0**. Nosotros somos quienes decidimos el significado de estos códigos.

```
public class SecondaryProcess {

    public static void main(String[] args) {
        System.out.println("Secondary process running...");

        // The process do some stuff, but something goes wrong...

        // System.exit() ends teh JVM instances in execution. If exit (0) then it ended
        // OK. If not, there was an error. The number is the error code, which we decide.

        System.exit(103);
    }
}
```

Vamos ahora con el **PrimaryProcess**. Existen dos formas de lanzar a ejecución el **SecondaryProcess**. Vemos a la primera. Notarás que el proceso principal ejecuta la función **waitFor ()**, que afectos prácticos, ordena detener la ejecución de este proceso en espera de que finalice el secundario. El programa efectivamente se detendrá en espera de que concluya el otro. ¿Cómo se tratan las respuestas? Pues preguntando exactamente cuál es el valor de retorno del **SecondaryProcess** (103)**.**

```
public static void main(String[] args) {
    System.out.println("Primary process running...");
    System.out.println("Launching secondary process...");

    ProcessBuilder processBuilder = null;
    Process process = null;
    try {
        // We prepare de builders...
        processBuilder = new ProcessBuilder("java", "es.ProcesoSecundario");
        processBuilder.directory(new File("bin"));
        process = processBuilder.start();

        // The primary process waits until secondary process ends
        int valorRetorno = process.waitFor();
        if (valorRetorno == 0) {
            System.out.println("Secondary process finsished!!");
        } else {
            System.out.println("Secondary process finsished with errors: " + valorRetorno);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

La otra forma de lanzar el proceso es mediante Runtime:

```
public static void main(String[] args) {
    System.out.println("Primary process running...");
    System.out.println("Launching secondary process...");

    try {
        // We prepare the process...
        String[] processInfo = {"java", "es.ProcesoSecundario"};
        Runtime runtime = Runtime.getRuntime();
        Process process = runtime.exec(processInfo);

        // The primary process waits until secondary process ends
        int valorRetorno = process.waitFor();
        if (valorRetorno == 0) {
            System.out.println("Secondary process finsished!!");
        } else {
            System.out.println("Secondary process finsished with errors: " + valorRetorno);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

A efectos prácticos, en este ejemplo hemos realizado una **Sincronización** de Procesos. En lugar de descargar el trabajo únicamente en un proceso, hemos delegado una tarea en un proceso secundario, y hemos hecho que el principal espere a su finalización. Date cuenta que de si NO estuviesen sincronizados, uno podría haber terminado antes que el otro.

El Runtime

Toda aplicación de Java dispone de una única instancia de la clase **Runtime**, la cuál le permite interactuar con su entorno de ejecución. Esto le permite entre otras cosas interactuar con el sistema operativo mediante la función **exec ()**. Adicionalmente, la clase Runtime ofrece otros métodos de interés:

Método	Descripción
destroy ()	Destruye el proceso
exitValue ()	Devuelve el valor de retorno
getErrorStream ()	Proporciona un InputStream conectado a la salida de error del proceso
getInputStream ()	Proporciona un InputStream conectado a la entrada del proceso
getOutputStream ()	Proporciona un InputStream conectado a la salida del proceso
isAlive ()	¿Está el proceso en ejecución?
waitFor ()	Espera a que el proceso termine

Siguiendo la lógica anterior, podemos lanzar un Notepad a ejecución mediante el ProcessBuilder:

```
try {
    ProcessBuilder processBuilder = new ProcessBuilder("Notepad.exe");

    // Info about the process
    Map <String, String> environment = processBuilder.environment();
    String number = environment.get ("NUMBER_OF_PROCESSORS");
    System.out.println("Numero de procesadores: " + number);

    // Start the notepad
    Process process = processBuilder.start();

    long pid = process.pid();
    System.out.println("Process PID: " + pid);

    // The primary process waits until secondary process ends
    int valorRetorno = process.waitFor();
    System.out.println("Secondary process finsished! - " + valorRetorno);

} catch (Exception e) {
    e.printStackTrace();
}
```

E igualmente con Runtime:

```
public static void main(String[] args) {
    System.out.println("Launching notepad...");

    String[] processInfo = { "notepad" };
    try {
        Process process = Runtime.getRuntime().exec(processInfo);

        long pid = process.pid();
        System.out.println("Process PID: " + pid);

        // The primary process waits until secondary process ends
        int valorRetorno = process.waitFor();
        System.out.println("Secondary process finsished! - " + valorRetorno);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Nótese que en este caso el proceso principal finaliza, aunque la ventana del Notepad sigue abierta.

Habrás notado que en el fondo tanto **ProcessBuilder** como **Runtime** hacen básicamente lo mismo: retornar un objeto **Process**. En el fondo, la diferencia entre ambos se limita a cómo hacen dicha acción y los métodos que permiten utilizar. En resumen:

ProcessBuilder

- Instancia un nuevo proceso, y le podemos pasar como argumento un programa que ya existe en nuestro equipo.
- Una vez inicializado, disponemos de todas las características del proceso en el objeto Process, como el PID.

Runtime

- Encapsula el entorno de trabajo de la JVM.
- Permite recuperar información del sistema, como número de procesadores, memoria disponible, llamar al Garbage Collector…

Finalmente, **Process**:

- Encapsula la información de un programa en ejecución.
- **ProcessBuilder** y **Runtime** trabajan con ella
- Permite usar Streams para redireccionar la entrada/salida al proceso padre empleando los métodos getInputStream (), getOutputStream () y getErrorStream ().

Comunicación entre procesos

En cualquier sistema operativo multitarea moderno (Windows, Linux, etc.), todo proceso que se ejecuta tiene asociados tres **flujos estándar**:

- **stdin (entrada estándar)**: normalmente el teclado.
- **stdout (salida estándar)**: normalmente la consola/pantalla.
- **stderr (salida de error estándar)**: normalmente también la consola, pero separado de stdout.

Cuando un proceso en java ejecuta la instrucción System.out.println("Hola") lo que está haciendo es escribir en el stdout del proceso creado por el SO.

Por tanto, podemos cambiar estas entradas y salidas estándar de procesos para que se comuniquen entre ellos. Por ejemplo:

- Redireccionamiento a archivos: Podemos redirigir la entrada y salida estándar del proceso secundario mediante los métodos redirectInput (), redirectOutput () y redirectError () de ProcessBuilder.

- Conexión E/S entre procesos: Podemos usar getInputStream (), getOutputStream () y getErrorStream (). Un proceso escribe en el OutputStream mientras otro lee del InputStream.

Por ejemplo, una comunicación simple podría ser la siguiente. Tenemos un proceso secundario que lanza a ejecución el proceso cmd.exe (línea de comandos de Windows). Nos aprovechamos del OutputStream para ejecutar una serie de comandos, y luego, recogemos el resultado mediante un InputStream. El resultado de la ejecución es:

```
Process running...
Microsoft Windows [Versi□n 10.0.26100.6584]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Trastero\eclipse-workspace\Multiproceso>echo Hello from Java
Hello from Java

C:\Trastero\eclipse-workspace\Multiproceso>dir
 El volumen de la unidad C es Blade 15
 El n□mero de serie del volumen es: 7615-7439

 Directorio de C:\Trastero\eclipse-workspace\Multiproceso

16/09/2025  15:22    <DIR>          .
16/09/2025  15:22    <DIR>          ..
16/09/2025  15:22               396 .classpath
16/09/2025  15:22               388 .project
16/09/2025  15:22    <DIR>          .settings
16/09/2025  17:35    <DIR>          bin
16/09/2025  17:35    <DIR>          src
               2 archivos            784 bytes
               5 dirs  734.439.333.888 bytes libres
```

El código es:

```
System.out.println("Process running...");

ProcessBuilder processBuilder = null;
Process process = null;
OutputStream outputStream = null;
InputStream inputStream = null;
InputStreamReader inputStreamReader = null;
BufferedReader bufferedReader = null;
try {
    processBuilder = new ProcessBuilder("cmd.exe");
    process = processBuilder.start();
    outputStream = process.getOutputStream();

    // We write some info..
    PrintWriter writer = new PrintWriter(outputStream, true);
    writer.println("echo Hello from Java");
    writer.println("dir");
    writer.flush();

    // We read the output
    inputStream = process.getInputStream();
    inputStreamReader = new InputStreamReader(inputStream);
    bufferedReader = new BufferedReader(inputStreamReader);
    String line;
    while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
    }

} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Ejercicio Propuesto - 1

Crea un programa en Java que obtenga todas las variables de entrono de tu equipo y las muestra por pantalla.

Ejercicio Propuesto - 2

Crea un programa en Java que obtenga la dirección MAC de tu equipo y la muestre por pantalla.

Ejercicio Propuesto - 3

Crea un programa en Java que detecte si el bloc de notas se está ejecutando y en caso afirmativo cree un proceso que lo elimine de la ejecución (matar el proceso).

Ejercicio Propuesto - 4

Crea un programa en Java que ejecuta un ".bat" previamente preparado y recoge la salida en un archivo y los errores en otro.

Ejercicio Propuesto - 5

Crea un programa en Java con dos procesos. El subproceso únicamente recoge por teclado un texto. El proceso principal lanza al subproceso, le 'coge' sus InputStream y OutputStream. Una vez hecho eso, le 'manda' al subproceso "Introduce un texto" y lee la información introducida por teclado en aquel proceso.