

**MP\_0490.**  
**Programación de servicios y procesos**  
**UF1. Programación multiproceso. Hilos**

# **1.1. Programas, ejecutables, procesos y servicios**

# Índice

---

☰	Objetivos	3
☰	Programas	4
☰	Ejecutables	6
☰	Procesos	8
☰	Multiproceso	12
☰	Lanzar procesos desde Java	14
☰	Lanzar procesos con argumentos	16
☰	Operaciones de entrada y salida en los procesos	18
☰	Lanzar varios procesos	24
☰	Servicios	27
☰	Resumen	28

# Objetivos

---

En esta lección perseguimos los siguientes objetivos:

1

Comprender los conceptos básicos relacionados con la asignatura de "Programación de servicios y procesos".

2

Distinguir entre *programa, ejecutable, proceso y servicio*.

3

Distinguir entre los conceptos de *multiproceso y multitarea*.

4

Crear programas Java capaces de lanzar procesos.

---

¡Ánimo y adelante!

# Programas

**Un programa es una secuencia de tareas cuya misión consiste en resolver un determinado problema de software.**

**Un programa está compuesto por...**

- **Instrucciones.**  
Conjunto de acciones que la computadora debe ejecutar para resolver el problema de software propuesto. Denominamos **algoritmo** al conjunto de instrucciones y la forma de ordenarlas para su ejecución.
  
- **Datos.**  
Constituyen la información que las instrucciones requieren para llevar a cabo las acciones.

**En función del objetivo que persiguen, los programas pueden dividirse en...**

## Programas del sistema

Forman parte del **sistema operativo** y permiten al usuario interactuar con la computadora. Algunos de estos programas sirven para gestionar los dispositivos, y otros para suministrar al usuario una interfaz para que pueda realizar las tareas básicas.

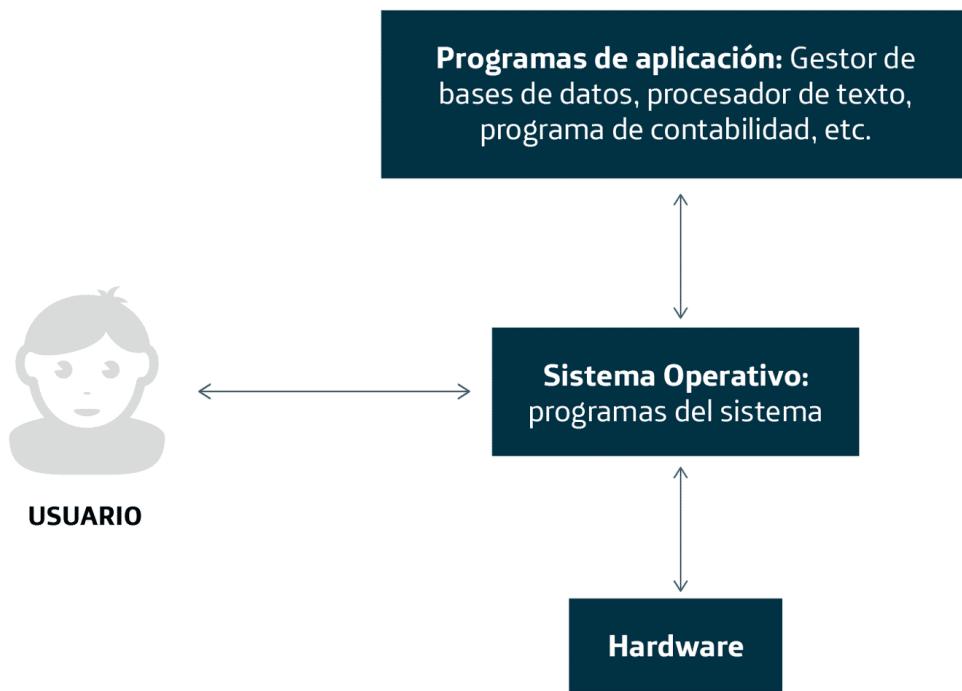
## software de programación

Programas que forman parte del **software de programación**: utilizado por los programadores para escribir el código fuente de los programas, verificar la sintaxis, compilar, depurar, etc.

## Programas de aplicación

Forman parte del software de aplicación y facilitan al usuario la ejecución de determinados trabajos. Ejemplos: procesadores de texto, hojas de cálculo, programas de contabilidad, etc.

**El siguiente esquema refleja cómo el Sistema Operativo actúa como intermediario:**



El Sistema Operativo se sitúa en el centro entre los programas de aplicación, el hardware y el usuario, actuando como puente o intermediario entre todos ellos.

# Ejecutables

**Llamamos ejecutable al archivo capaz de lanzar un programa, es decir, ponerlo en funcionamiento.**

Un archivo ejecutable encierra un programa y, además, contiene la estructura e información necesarias para que el sistema operativo sea capaz de ejecutarlo. Su extensión en Windows es **.exe**.

Un programa ejecutable está **traducido a código máquina**, es decir, todas las instrucciones y los datos han sido traducidos al sistema binario, una codificación a base de ceros y unos.

No hay que olvidar que este es el único lenguaje que entiende la computadora. *A priori*, no podemos saber las instrucciones que están almacenadas en un archivo ejecutable, aunque existen programas desensambladores capaces de realizar la traducción desde código máquina a lenguaje ensamblador.

Un ejemplo está en la aplicación en línea <https://onlinedisassembler.com>.

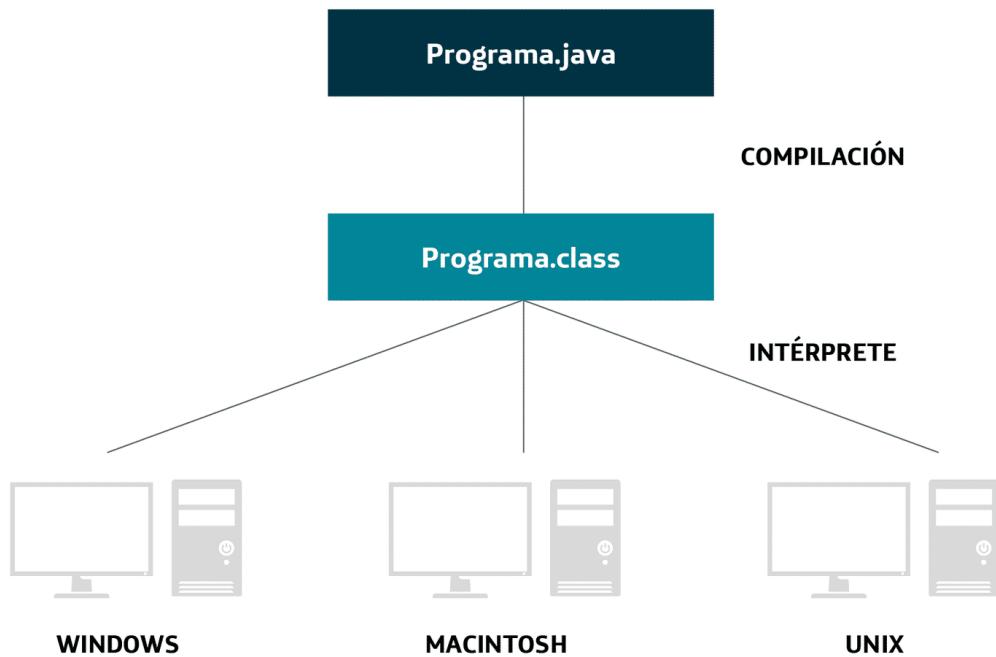


Los programas **.exe** son ejecutados por el sistema operativo, por lo que, generalmente, tienen la problemática de que son dependientes de la plataforma hardware y software. Un ejecutable creado para correr en una máquina Intel con un sistema operativo Windows, no podrá funcionar en un Mac o, simplemente, en un sistema operativo Linux. Java soluciona este problema con su **bytecode** interpretado por la máquina virtual de Java.

## Los ejecutables Java

En Java, los programas fuentes (ficheros `.java`) deben ser compilados, pero no para crear código máquina directamente, sino para crear un código que queda a medio camino entre el código fuente y el código máquina (fichero `.class`), al que denominamos *bytecode* (o *bycode*).

El *bytecode* es un tipo de codificación, parecida al ensamblador, que deberá ser interpretada por una máquina virtual, línea a línea, según se vaya ejecutando. El encargado de esta interpretación línea a línea es el compilador JIT (*just-in-time*) que va traduciendo el *bytecode* instrucción a instrucción para el microprocesador y sistema operativo en el que nos encontramos. Este proceso es lo que hace que Java sea independiente de la plataforma.



Los archivos `.class` son interpretados por la máquina virtual de java de cada tipo de plataforma.

## Procesos

---

**Un proceso es un programa en ejecución. Puesto que un proceso está en ejecución, está consumiendo recursos del sistema.**



**Definición de la RAE de proceso informático:** ejecución de diversas instrucciones por parte del microprocesador, de acuerdo a lo que indica un programa.

---

El sistema operativo es el encargado de gestionar los procesos, creándolos y borrándolos cada vez que sea necesario.

**La ejecución de un proceso implica:**

1

La reserva de una determinada memoria de trabajo.

2

Carga de trabajo para el procesador, que tendrá que ir ejecutando las instrucciones incluidas en el proceso.

3

Cambios de estado del proceso, que se reflejan en los valores de los registros de la CPU.

## Estados que puede atravesar un proceso:

### Nuevo

El proceso acaba de ser creado.

### En ejecución

El proceso está haciendo uso del procesador en este momento.

### Bloqueado

El proceso no seguirá ejecutándose hasta que el usuario responda a un evento externo.

Ejemplo: el proceso es creado por un programa con una interfaz de usuario que está en espera de que el usuario haga clic en algún botón para realizar alguna acción.

### Listo

El proceso no está en este momento en ejecución, aunque puede pasar a estarlo en cualquier momento.

### Terminado

El proceso ha finalizado su ejecución y se liberan los recursos del sistema que estaba consumiendo.

Cabe señalar que el estado de finalización de un proceso puede dar lugar a distintas circunstancias o sucesos, que se reflejan en el estado del proceso en el momento que va a ser eliminado:

- **Salida normal:** el proceso ha cumplido su misión y ha finalizado, a petición del usuario o de otro proceso que lo ha puesto en marcha.
- **Salida de error:** el proceso finaliza por un error en tiempo de ejecución.
- **Salida de error fatal:** similar al anterior, pero por causa más grave, por ejemplo: intento de escribir en un área de memoria ocupada.
- **Eliminado por otro proceso:** un caso típico podría ser cuando abortamos un proceso desde el administrador de tareas de Windows.

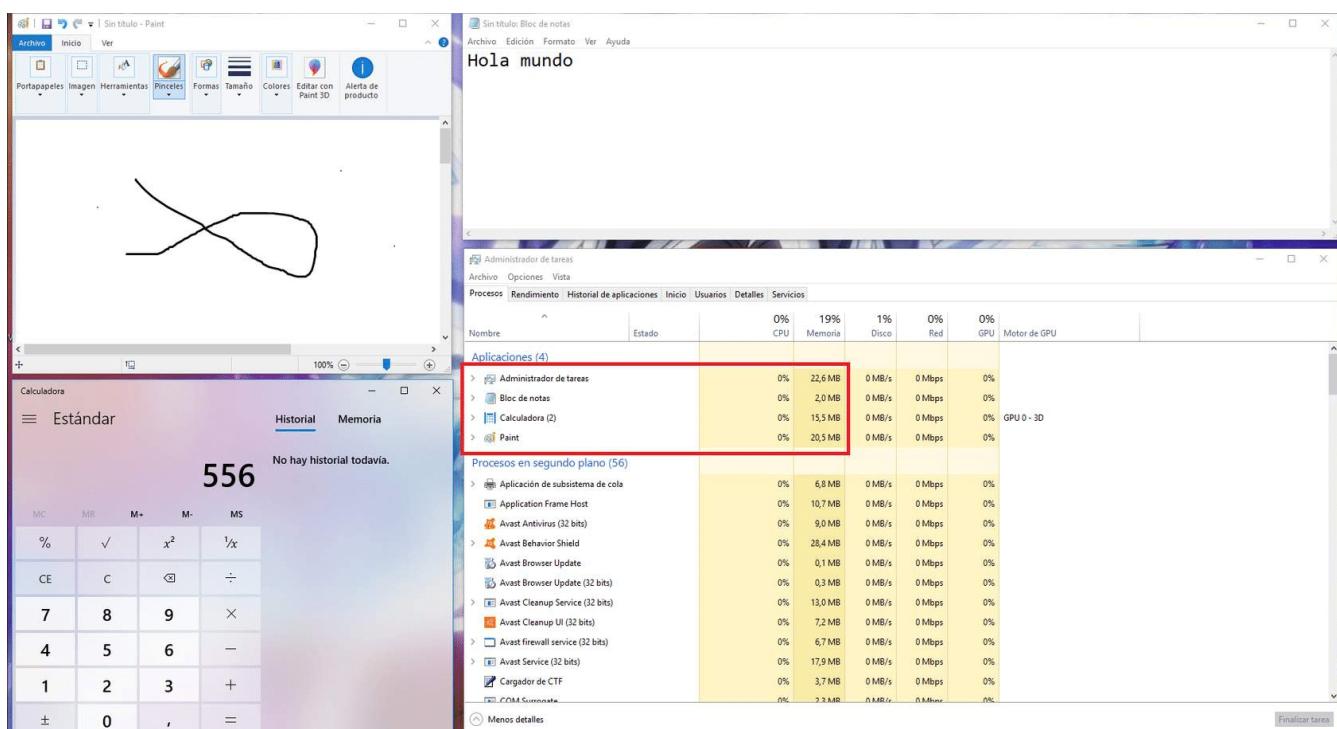


Estados que atraviesa un proceso.

## Comprueba los procesos activos en Windows 10

El administrador de tareas de Windows te permite ver los procesos que el sistema operativo está gestionando.

Como práctica, inicia varios procesos, por ejemplo, puedes ejecutar la calculadora de Windows, el *bloc de notas* y el programa *paint*. Luego, pulsa *ctrl+alt+supr* para activar el administrador de tareas y ver los procesos.



Cómo comprobarás en la imagen, puedes ver los procesos que está gestionando el sistema operativo, el porcentaje de uso de la CPU y la memoria que están consumiendo.

Te recomendamos que intentes situar todas las aplicaciones que has abierto en el escritorio, dejándolas visibles. Después, prueba a interactuar con las distintas aplicaciones y comprobar cómo va cambiando el porcentaje de uso de la CPU en el administrador de tareas.

# Multiproceso

---

**Se denomina multiproceso a la ejecución simultánea de varios procesos.**

Para que exista **multiprocesamiento** en el sentido estricto, es necesario un ordenador que contenga dos o más procesadores, lo que permitirá la ejecución simultánea de varios programas que compartirán el uso de la memoria central y de los dispositivos periféricos. Además de varios procesadores, será necesario un sistema operativo capaz de gestionar el multiprocesamiento.

El multiprocesamiento puede simularse con una única CPU, haciendo una distribución de tiempos para cada uno de los procesos en ejecución.

Aunque tu ordenador cuente con una única CPU, puedes estar ejecutando varios programas a la vez. Seguro que habitualmente tienes abierto un programa de procesamiento de textos, un programa de gráficos y, a la vez, un navegador web. ¿Cómo es posible?

El sistema operativo se encarga de asignar el uso de la CPU por intervalos de tiempo a los distintos procesos abiertos. Si, por el contrario, tu ordenador cuenta con varios procesadores, entonces varios procesos podrán compartir el mismo procesador o no; eso será cuestión de la gestión que haga el sistema operativo de los procesos.

## ¿Qué es la programación concurrente?

Un **programa concurrente** es el que lanza varios procesos al mismo tiempo, que colaboran entre sí para realizar una tarea compartida.

Pongamos un ejemplo: queremos crear un programa que realice un dibujo despacio, de modo que el usuario vaya viendo los trazos. Imagina que el dibujo contendrá un árbol y una casa. Si queremos que la casa y el árbol se vayan trazando simultáneamente, será necesaria la ejecución de dos procesos concurrentes, aunque ambos colaboren para lograr el mismo fin, la confección del dibujo.

## Y, ¿qué son la programación paralela y la programación distribuida?

- La **programación paralela** se ejecuta en un único ordenador, ya se disponga de varios procesadores o uno solo.
- La **programación distribuida** se ejecuta en varios ordenadores, que se comunican en red y colaboran para realizar una tarea común.

# Lanzar procesos desde Java

**Java tiene sus mecanismos para comunicarse con el sistema operativo y darle la orden de iniciar un proceso.**

Veamos un ejemplo, utilizando la clase *ProcessBuilder* situada en el paquete *java.lang*.

**(i) Recuerda que el paquete *java.lang* es el único que es importado de manera predeterminada en todos los proyectos.**

Este sencillo programa lanza la *calculadora* de Windows.

```
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/calc.exe");
        try {
            proceso.start();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

**(i) Puede que la ruta "C:/Windows/System32/calc.exe" sea diferente en tu equipo. Comprueba la ruta donde se encuentra el archivo *calc.exe* y corrígela, si es necesario.**

Hasta que no se ejecuta la sentencia `proceso.start()`, realmente no se lanza el proceso. Una vez que la *calculadora* de Windows se ha iniciado, nuestro programa ha finalizado. El proceso lleva consigo una operación de entrada/salida, ya que implica el acceso al fichero `calc.exe`, lo que podría ocasionar una excepción de tipo `IOException`.

Si queremos que nuestro programa se mantenga en ejecución a la espera de que termine el proceso iniciado, podemos utilizar el método `waitFor()`, lo que podría provocar una excepción de tipo `InterruptedException` en caso de que el proceso sea abortado por una situación de error.

Prueba a cambiar el código, dejándolo así:

```
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/calc.exe");
        try {
            Process p = proceso.start();
            p.waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Ahora, el programa no termina hasta que no se cierra la *calculadora* de Windows; lo demuestra el hecho de que no aparece el mensaje *Proceso lanzado con éxito* hasta ese momento. El método `start()` devuelve un objeto de tipo `Process` que dispone del método `waitFor()`. Nuestro programa queda detenido en la siguiente línea:

`p.waitFor();`

También podemos ahorrar alguna línea de código y la declaración de la variable `p`, dejándolo así:

```
proceso.start().waitFor();
```

# Lanzar procesos con argumentos

Algunos procesos requieren argumentos. El constructor de la clase *ProcessBuilder* admite la entrada de dichos argumentos.

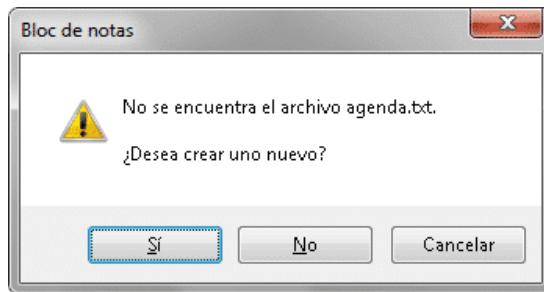
El formato del constructor de *ProcessBuilder* tiene el siguiente formato:

```
new ProcessBuilder("ruta programa", argumento1, argumento2, ....,  
argumentoN);
```

El siguiente ejemplo, inicia el *bloc de notas* y, además, especifica en el segundo argumento el nombre del archivo que deberá abrir.

```
import java.io.IOException;  
  
public class Principal {  
    public static void main(String[] args) {  
        ProcessBuilder proceso;  
        proceso = new ProcessBuilder("C:/Windows/System32/notepad.exe",  
"agenda.txt");  
        try {  
            proceso.start().waitFor();  
            System.out.println("Proceso lanzado con éxito");  
        } catch (IOException | InterruptedException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

Cuando ejecutes el programa, puesto que no tendrás un fichero con nombre "agenda.txt", te aparecerá el siguiente cuadro de diálogo:



Haz clic en el botón *sí* y será creado en la carpeta del proyecto de Eclipse. En las siguientes ejecuciones ya no te saldrá.

## Cambio del directorio de trabajo del proceso lanzado

En el programa anterior, hemos lanzado *notepad.exe* y dicho programa tiene un directorio de trabajo, es decir, un espacio donde guardará, de manera predeterminada, los archivos que se creen dentro de él. Como no hemos especificado ningún directorio de trabajo, ha utilizado como ubicación predeterminada la del proyecto Eclipse. Es posible cambiar dicho comportamiento, utilizando el método *directory()* antes de lanzar el proyecto.

Cambia de nuevo el programa para dejarlo así:

```
import java.io.File;
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/notepad.exe", "agenda.txt");
        try {
            proceso.directory(new File("F:\\"));
            proceso.start().waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Como argumento para el método *directory()*, pasamos un objeto *File* que apunte a un directorio con la ruta deseada. En nuestro caso, apuntamos a un *pendrive* que ocupa la unidad *F*.

# Operaciones de entrada y salida en los procesos

Cualquier programa típico lleva consigo una entrada de datos, un proceso o algoritmo, y una salida de resultados. Además, podrían producirse salidas de error.

El lanzamiento de un proceso desde un programa Java podría requerir de la gestión de estas entradas y salidas, lo que se efectúa a través de los siguientes métodos:

1. `ProcessBuilder.redirectInput(File fich);`

Recibe un objeto `File`, que apunta al fichero que contiene los datos de entrada para que el proceso pueda realizar su tarea. De aquí recoge las entradas estándar (`System.in`).

2. `ProcessBuilder.redirectOutput(File fich);`

Recibe un objeto `File`, que apunta al fichero donde se escribirán los datos de salida del proceso, es decir, los resultados. En este fichero escribe las salidas estándar (`System.out`).

3. `ProcessBuilder.redirectError(File fich);`

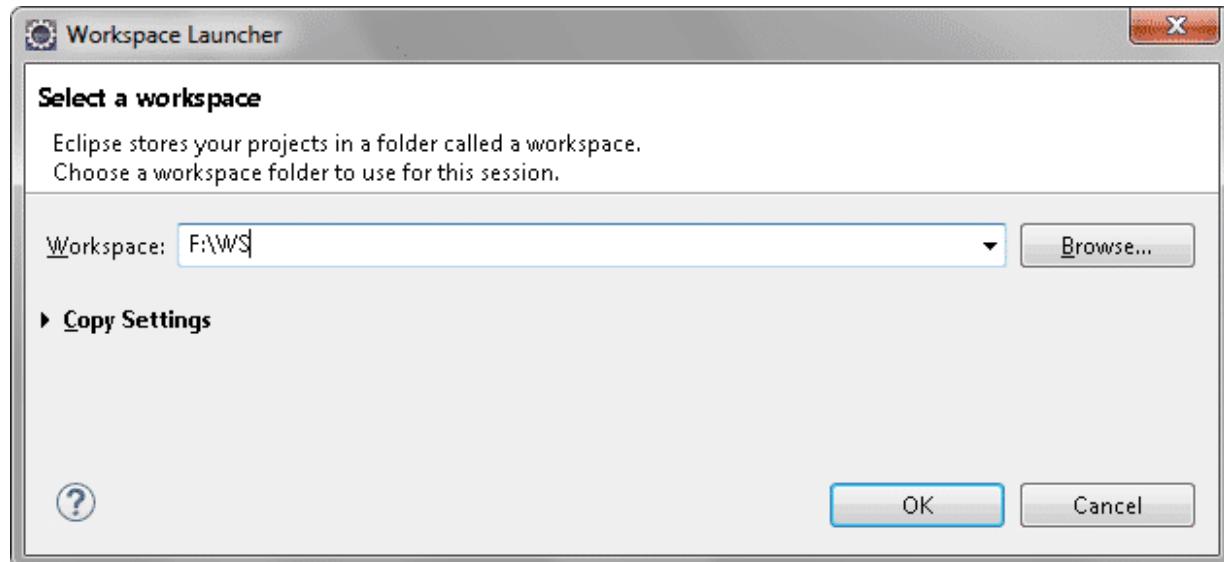
Recibe un objeto `File` que apunta al fichero donde se escribirán los datos de salida de error del proceso. En este fichero escribe las salidas de error (`System.err`).

## Sigue con cuidado los siguientes pasos para realizar una práctica

1

Crea un nuevo *workspace* en Eclipse.

Abre Eclipse en un nuevo espacio de trabajo (*workspace*). Es interesante que la ruta sea sencilla, ya que crearemos un programa que finalmente ejecutaremos desde el símbolo del sistema, por lo que resultará incómodo tener que escribir una ruta muy larga. Hemos elegido una carpeta llamada WS dentro de un pendrive que ocupa la unidad F.

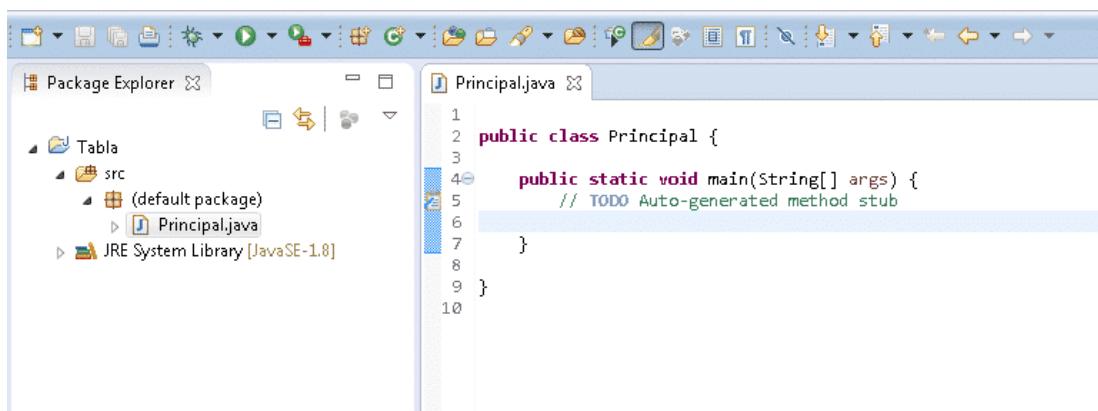


2

Crea el programa que servirá como proceso.

Vamos a comenzar por crear el programa que posteriormente lanzaremos como proceso desde otro programa. Será un programa sencillo que escriba la tabla de multiplicar de un número.

Crea un nuevo proyecto Java, llamado *Tabla*, y dentro de él una clase llamada *Principal* con método *main*. Este debe ser el aspecto de tu proyecto hasta ahora:



Completa el código de la clase *Principal*.

```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner lector = new Scanner(System.in);
        System.out.println("Tabla del número: ");
        int n = lector.nextInt();
        System.out.println(n);

        for (int i=1; i<=10; i++) {
            System.out.println(n + " X " + i + " = " + (n*i));
        }
        lector.close();
    }
}
```

Como ves, se trata de un programa muy simple, pero cuenta con su entrada de datos, proceso y salida de resultados, que es lo que nos interesa en este momento. Puedes ejecutarlo desde Eclipse para comprobar su funcionamiento; verás que solicita un número por teclado y, después, escribe en pantalla el número introducido y su tabla de multiplicar.

A screenshot of the Eclipse IDE terminal window. The title bar says '<terminated> Principal [Java Application] C:\Pro'. The terminal displays the following output:  
Tabla del número:  
6  
6  
6 X 1 = 6  
6 X 2 = 12  
6 X 3 = 18  
6 X 4 = 24  
6 X 5 = 30  
6 X 6 = 36  
6 X 7 = 42  
6 X 8 = 48  
6 X 9 = 54  
6 X 10 = 60

3

Crea el programa que lanzará el proceso.

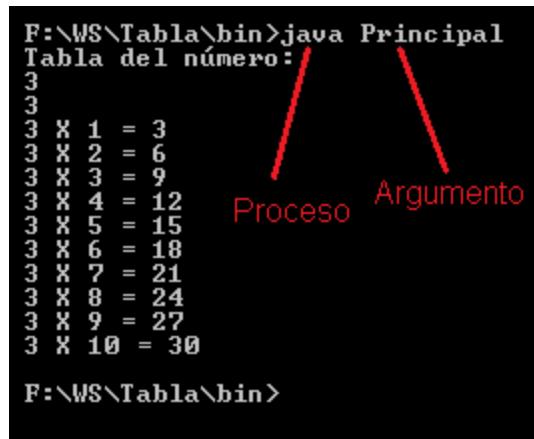
Crearemos ahora una clase llamada *Lanzador*. Para simplificar, lo haremos en el mismo proyecto, aunque perfectamente podría estar en otro proyecto independiente.

```
import java.io.File;
import java.io.IOException;

public class Lanzador {

    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("java", "Principal");
        proceso.redirectError(new File("errores.txt"));
        proceso.redirectOutput(new File("salida.txt"));
        proceso.redirectInput(new File("entrada.txt"));
        try {
            proceso.start();
            System.out.println("El proceso ha sido lanzado con éxito");
            System.out.println("Examina los archivos errores.txt y salida.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Recuerda que cuando ejecutas un programa Java desde el símbolo del sistema, utilizas el programa *Java* seguido del nombre de la clase a ejecutar, que será el argumento. Por esa razón pasamos la cadena *Principal* como argumento del proceso *Java* en la construcción del objeto *ProcessBuilder*.



```
F:\WS\Tabla\bin>java Principal
Tabla del número:
3
3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
F:\WS\Tabla\bin>
```

Si ejecutas ahora desde Eclipse, lo primero que te encontrarás es con un error, porque el archivo *entrada.txt* no existe.

4

Crea el archivo de entrada para el proceso.

Crea el archivo *entrada.txt* con ayuda del *bloc de notas* y guárdalo en la misma ruta donde se encuentran las clases ya compiladas *Principal.class* y *Lanzador.class*. Para nosotros, la ruta es "F:\WS\Tabla\bin". Escribe en el archivo un número entero, por ejemplo, 5. Al ejecutar el programa *Principal*, dicho número será recogido por la siguiente sentencia:

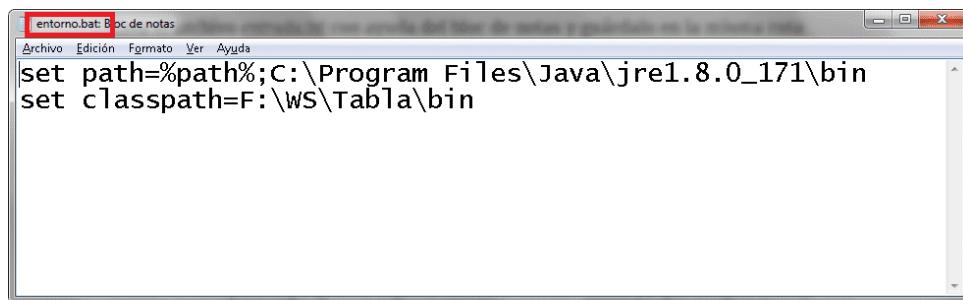
```
int n = lector.nextInt();
```

Dentro de Eclipse, el programa seguirá dando el mismo error, ya que Eclipse trabaja con su propia estructura de carpetas y tiene dificultades para encontrar el archivo *Principal.class*. Es el momento de comenzar a trabajar desde el símbolo del sistema.

5

Configura las variables de entorno para ejecutar desde el símbolo del sistema.

A partir de ahora es mejor que pases a trabajar con el símbolo del sistema, pero antes debes configurar algunas variables de entorno. Lo mejor es que crees un archivo de procesamiento por lotes, llamado *entorno.bat*, con el siguiente contenido:



En la variable *path* estás indicando el lugar donde se encuentra el archivo *Java.exe*. En la variable *classpath* estás especificando dónde se encuentran las clases Java que van a ser ejecutadas. **Ambas rutas serán diferentes en tu equipo, así que deberás adaptarlas para tu ejercicio.** Deja el archivo *entorno.bat* guardado también en la misma ubicación donde están los archivos *Principal.class* y *Lanzador.class*. Todo debe estar ubicado en el mismo sitio.

6

Ejecuta la clase *Lanzador.class*.

Abre una ventana del símbolo del sistema, ejecutando el comando *cmd* desde el cuadro *Buscar programas y archivos*.



La siguiente imagen ilustra los pasos que debes seguir para ejecutar el programa desde el símbolo del sistema.

```
ca Administrador: Símbolo del sistema
Microsoft Windows [Versión 6.1.7601]
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\USUARIO>F:
F:>>CD WS\Tabla\bin
F:\WS\Tabla\bin>entorno
F:\WS\Tabla\bin>set path=C:\Program Files <x86>\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Microsoft SQL Server\12.0\Tools\Binn\;C:\Program Files\Java\jdk1.8.0_40\bin;C:\Program Files\Visual Studio Server\bin;C:\apache-maven-3.3.3\bin;C:\PROGRAM~2\Groovy\GROOVY~1.3\bin;C:\Program Files <x86>\Skype\Phone\;C:\Program Files <x86>\BaseX\bin;;C:\Users\USUARIO\Desktop\Java Metro\jdk1.8.0_65\bin\;C:\Program Files\Java\jrei.8.0_171\bin
F:\WS\Tabla\bin>set classpath=F:\WS\Tabla\bin
F:\WS\Tabla\bin>java Lanzador
El proceso ha sido lanzado con éxito
Examina los archivos errores.txt y salida.txt
F:\WS\Tabla\bin>_
```

Como puedes comprobar, nos hemos situado en la ruta *F:\WS\Tabla\bin*, ubicación donde están los programas. Luego, hemos ejecutado el archivo de procesamiento por lotes, lo que ha provocado la ejecución de las dos órdenes *set*. Por último, hemos ejecutado el programa *Lanzador*.

---

La salida del programa estará en el archivo *salida.txt*, donde se encontrará la tabla de multiplicar.

---

## Lanzar varios procesos

En este apartado, modificaremos el proyecto anterior con el fin de lanzar procesos simultáneos.

Vamos a comenzar por realizar una pequeña modificación en la clase *Principal*.

```
import java.time.LocalDateTime;
import java.util.Scanner;

public class Principal {

    public static void main(String[] args) throws InterruptedException {
        Scanner lector = new Scanner(System.in);
        System.out.println("Tabla del número: ");
        int n = lector.nextInt();
        System.out.println(n);

        for (int i=1; i<=10; i++) {
            System.out.println(LocalDateTime.now() + " ----- " + n + " X
" + i + " = " + (n*i));
            Thread.sleep(250);
        }

        lector.close();
    }
}
```

Hemos introducido dos cambios:

1

En la salida de la tabla de multiplicar, hemos añadido la fecha y hora. Esto nos servirá cuando inspeccionemos el archivo de salida para saber exactamente en qué momento se ha ejecutado cada *System.out.println*.

2

Por cada línea de salida, hacemos dormir el proceso un cuarto de segundo (250 milisegundos). Para ello, estamos utilizando la clase *Thread*, cuyo estudio abordaremos en la siguiente lección.

---

Y ahora, vamos a modificar la clase *Lanzador*.

```
import java.io.IOException;

public class Lanzador {
    public static void main(String[] args) {
        ProcessBuilder proceso1, proceso2;
        proceso1 = new ProcessBuilder("java", "Principal");
        proceso2 = new ProcessBuilder("java", "Principal");

        proceso1.redirectError(new File("errores1.txt"));
        proceso1.redirectOutput(new File("salida1.txt"));
        proceso1.redirectInput(new File("entrada1.txt"));

        proceso2.redirectError(new File("errores2.txt"));
        proceso2.redirectOutput(new File("salida2.txt"));
        proceso2.redirectInput(new File("entrada2.txt"));

        try {
            proceso1.start();
            proceso2.start();
            System.out.println("Los procesos se han lanzado");
            System.out.println("Examina los archivos de salida");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Como ves, estamos haciendo lo mismo que antes, solo que ahora lanzamos dos procesos, cada uno de ellos con diferentes archivos para la entrada y la salida. Vuelve a ejecutar el programa *Lanzador* desde el símbolo de sistema, pero no te olvides de crear primero los archivos *entrada1.txt* y *entrada2.txt*, cada uno de ellos con un número diferente.



Si ya has ejecutado el programa, te recomendamos que abras los archivos *salida1.txt* y *salida2.txt*. Junto a cada línea verás que aparece la fecha y hora exacta de la ejecución, de modo que puedes comprobar que ambos procesos se han estado ejecutando simultáneamente.

Ahora, prueba a añadir el método *waitFor()* en el lanzamiento de cada proceso.

```
proceso1.start().waitFor();
proceso2.start().waitFor();
```

En este caso, si vuelves a ejecutar, comprobarás que los procesos no se simultanean; hasta que no termina un proceso no comienza el siguiente, algo que puedes comprobar de nuevo a través de las fechas. **Esto podría resultar interesante a la hora de establecer comunicación entre procesos. El archivo de salida de un proceso podría ser el de entrada para otro proceso.**

## Servicios

---

**Un servicio es un programa que se ejecuta en segundo plano y que tiene como finalidad prestar servicio al sistema operativo o a otros programas.**

A diferencia de otros programas, los servicios no requieren la intervención del usuario; normalmente se ejecutan sin que este sea consciente.

Los servicios pueden **iniciarse y detenerse**. La mayoría de ellos pueden configurarse para que se inicien automáticamente al arrancar el sistema operativo. El usuario no interactúa con el servicio, pero este es necesario para el funcionamiento del sistema o de otras aplicaciones que lo requieran.

Algunos ejemplos de servicios con los que cuenta el sistema operativo son:

- Administrador de sesiones de usuario.
- Servicio de cola de impresión.
- Servicio de escritorio remoto.
- Servicio de actualización de zona horaria.
- Servicio de informe de errores de Windows.

Todos estos programas actúan constantemente en segundo plano sin que seamos conscientes y son necesarios para el buen funcionamiento del sistema.

## Resumen

---

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Un **programa** es una secuencia de tareas cuya misión consiste en resolver un determinado problema de software.
- Un **ejecutable** es un archivo capaz de lanzar un programa, es decir, ponerlo en funcionamiento.
- Un **proceso** es un programa en ejecución.
- Java tiene sus mecanismos para comunicarse con el sistema operativo y darle la orden para que inicie un proceso. Uno de estos mecanismos consiste en el uso de la clase *ProcessBuilder*, ubicada en el paquete *java.lang*.
- Se denomina **multiproceso** a la ejecución simultánea de varios procesos, lo que requiere de varios procesadores.
- La **multitarea** es la ejecución de varios procesos concurrentes, que pueden compartir o no la misma CPU.



**PROEDUCA**