Tema 1
Programación multiproceso

2ª DAM
Curso 2025/26

Contenidos

- Procesos, Estados de un proceso.
- Gestión de procesos.
- Creación de procesos en Java

Introducción

- Actualmente las máquinas que conocemos ejecutan varios programas a la vez.
- Un programa que se carga en memoria se convierte en “proceso”

Procesos y Sistema Operativo

- Un **proceso** es un programa en ejecución y consiste en:
  - Código del programa
  - Datos
  - Pila del programa
  - Contador del programa
  - Puntero a pila y otros registros
  - Toda la información para ejecutar el programa

Procesos y Sistema Operativo

El sistema operativo es quien decide qué proceso se ejecutará en la CPU.

La información del proceso se almacena en una estructura de datos llamada *Bloque de Control de Proceso* o **BCP**

BCP

- Un **registro** especial donde el sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular.
- Al terminar el proceso el BCP se borra.
- Es una estructura de datos con campos

Estructura BCP

- Identificación de Proceso (PID, PPID)
- Registros Visibles para el Usuario
- Registros de Control y de Estado:
  - Contador de programa
  - Códigos de condición
  - Información de estado
- Punteros de pila

Estructura BCP

- Información de Planificación y de Estado
  - Estado del proceso
  - Prioridad
  - ….
- Estructuración de datos
- Comunicación entre Procesos
- Privilegios de los procesos
- Gestión de Memoria
- Propiedad de los Recursos y Utilización

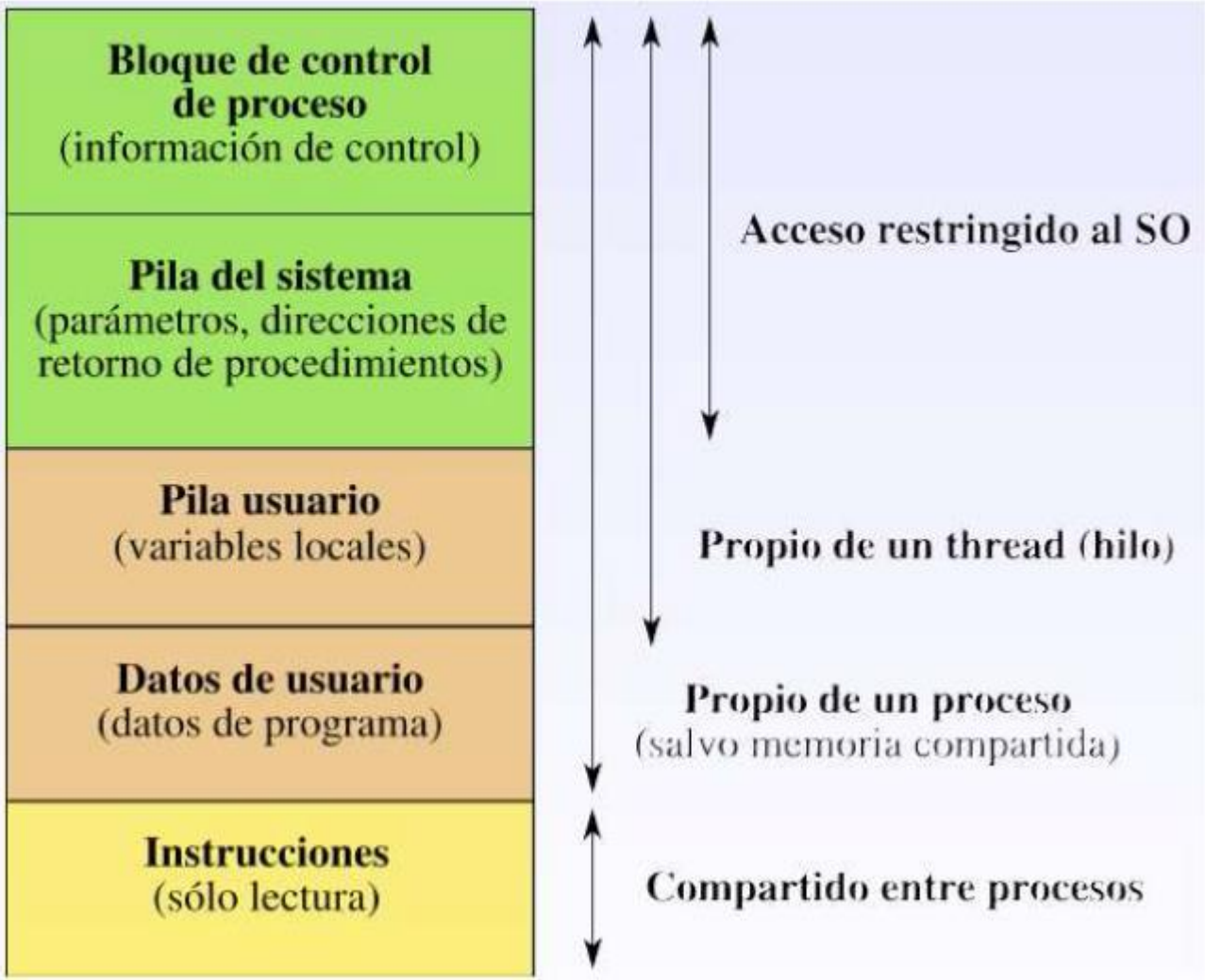
Procesos

- Un proceso o tarea es manejado por el sistema operativo y está **formado** por:
  - Las **instrucciones** del programa que van a ser ejecutadas.
  - El **estado** de ejecución en un momento concreto..
  - Memoria que ha reservado y el contenido de la misma.
  - Más **información** para que el sistema operativo pueda planificarlo.

Procesos

- Un proceso o tarea es manejado por el sistema operativo y está **cargado en memoria** por regiones:
  - **Región de código** --> tamaño fijo y de sólo lectura (podrá ser compartida por más de un proceso)
  - **Región de datos** --> variables globales y estructuras dinámicas.
  - **Región de pila** --> datos temporales (puede desbordarse)

Procesos


Bloque de control de proceso
(información de control)
Pila del sistema
(parámetros, direcciones de retorno de procedimientos)
Pila usuario
(variables locales)
Datos de usuario
(datos de programa)
Instrucciones
(sólo lectura)
Acceso restringido al SO
Propio de un thread (hilo)
Propio de un proceso
(salvo memoria compartida)
Compartido entre procesos

Procesos. Características

- Para que un proceso pueda empezar su ejecución, debe de **residir** completamente en memoria y tener asignados todos los recursos que necesite.
- Dos procesos pueden asociarse al mismo programa.
- Cada proceso está protegido del resto de procesos, ningún otro proceso podrá escribir en las zonas de memoria pertenecientes a ese proceso.

Procesos. Características

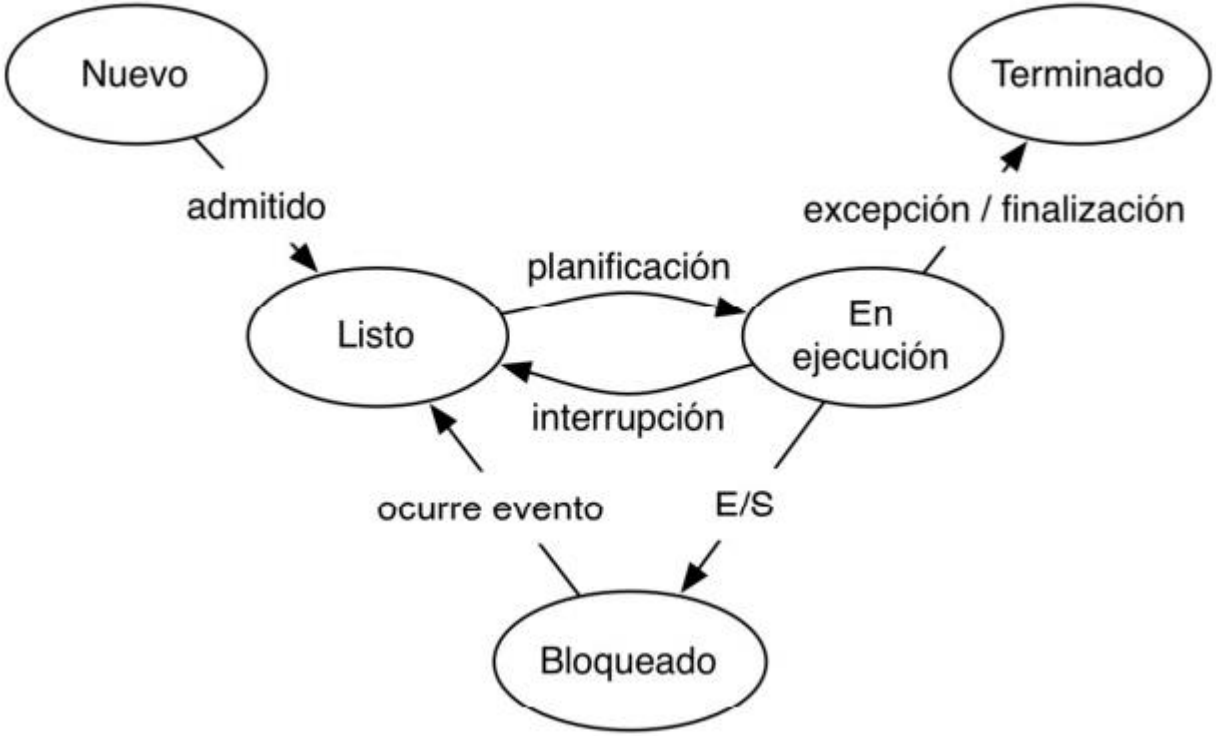
- Los procesos pertenecientes a los usuarios se ejecutan en el modo usuario del procesador (con restricciones de acceso a los recursos), los que pertenecen al sistema se ejecutarán en el modo kernel del procesador (podrán acceder a cualquier recurso).
- Para que un proceso de usuario acceda a los recursos tendrá que hacerlo por medio de **llamadas al sistema**.
- Los procesos se podrán **comunicar**, **sincronizar** y **colaborar** entre ellos.

Procesos. Estados

- Distintos procesos coexisten en el sistema operativo.
- La salida de un proceso puede ser la entrada para un segundo proceso --> se produce un **bloqueo** en el segundo.
- El sistema operativo también puede decidir que ha de terminar un proceso.

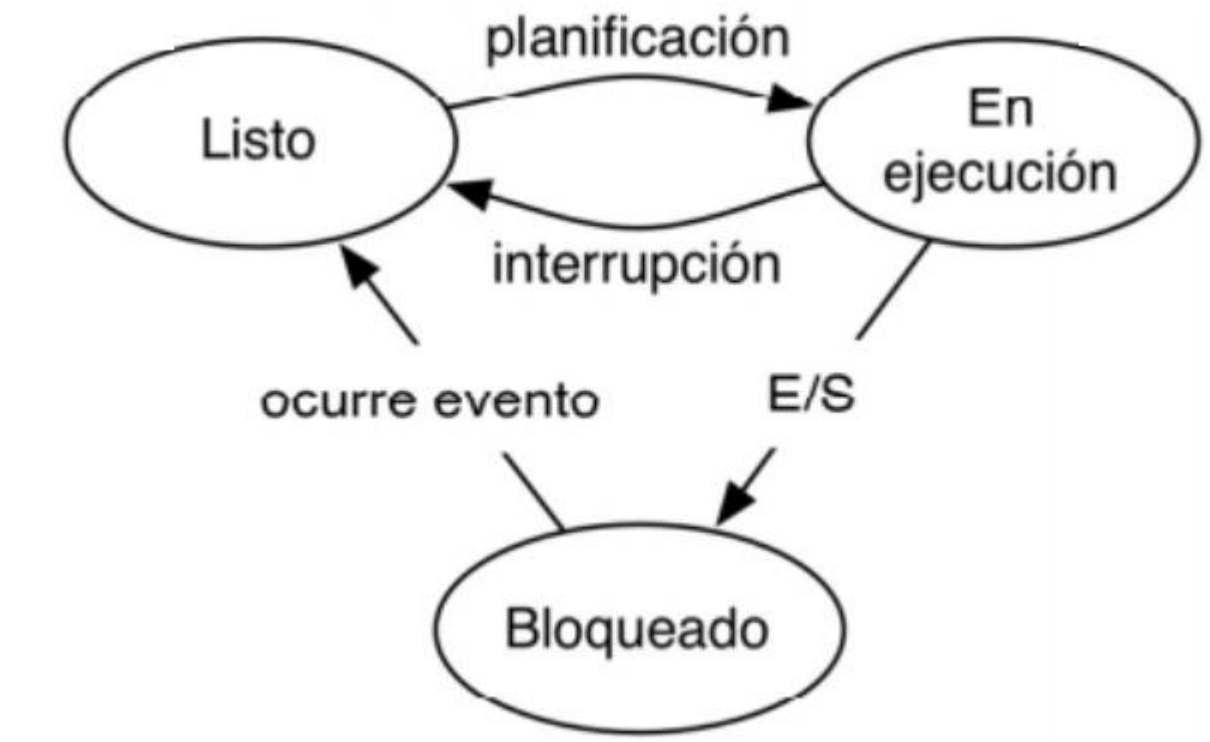
Procesos. Estados

Estados por los que pasaría …



Procesos. Estados

Simplificaremos los estados en 3: ejecución, bloqueado y listo.



Procesos. Estados

- **En ejecución**: Proceso está utilizando la CPU, las instrucciones se están ejecutando.
- **Listo**: Proceso está en condiciones de ejecutarse, pero está esperando que se le asigne tiempo de CPU.
- **Bloqueado** : Proceso está esperando que ocurra un evento externo (como la terminación de una E/S).

Procesos. Transiciones

- **De Ejecución – a Bloqueado**: Un proceso pasa de ejecución a bloqueado (bloqueo) cuando ejecuta una instrucción que implica la espera de un evento, por ejemplo debe esperar que ocurra efectivamente la E/S, espera por la activación de un semáforo, etc.
- **De Ejecución – a Listo**: Un proceso pasa de ejecución a listo (fin de tiempo), cuando se le acaba el tiempo asignado por el planificador de procesos del sistema, en este momento el sistema debe asignar el procesador a otro proceso.

Procesos. Transiciones

- **De Listo – a Ejecución**: Un proceso pasa de listo a ejecución (despachar) cuando el sistema le otorga un tiempo de CPU.
- **De Bloqueado – a Listo**: Un proceso pasa de bloqueado a listo (ocurre un evento) cuando el evento externo que esperaba sucede.

Procesos con Java

- Java dispone de mecanismos para lanzar procesos y relacionarse con ellos.
- Para la gestión de procesos usaremos el paquete *java.lang*
- La clase **Runtime**
  - Cada aplicación java dispone de una **única instancia** de la clase Runtime, el entorno de ejecución de la aplicación y la obtendremos con el método *getRuntime()*
  - El método *exec(String comando)* similar al visto en C, devuelve un objeto Process

Procesos en Java

exec

```
public Process exec(String command)
            throws IOException
```

Executes the specified string command in a separate process.

This is a convenience method. An invocation of the form `exec(command)` behaves in exactly the same way as the invocation `exec(command, null, null)`.

Parameters:

`command` - a specified system command.

Returns:

A new `Process` object for managing the subprocess

Throws:

`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess

`IOException` - If an I/O error occurs

`NullPointerException` - If `command` is `null`

`IllegalArgumentException` - If `command` is empty

See Also:

`exec(String[], String[], File)`, `ProcessBuilder`

Procesos con Java

- El método exec devuelve un proceso, encapsulado en la clase Process

```
Process p = r.exec("CMD /C DIR");
```

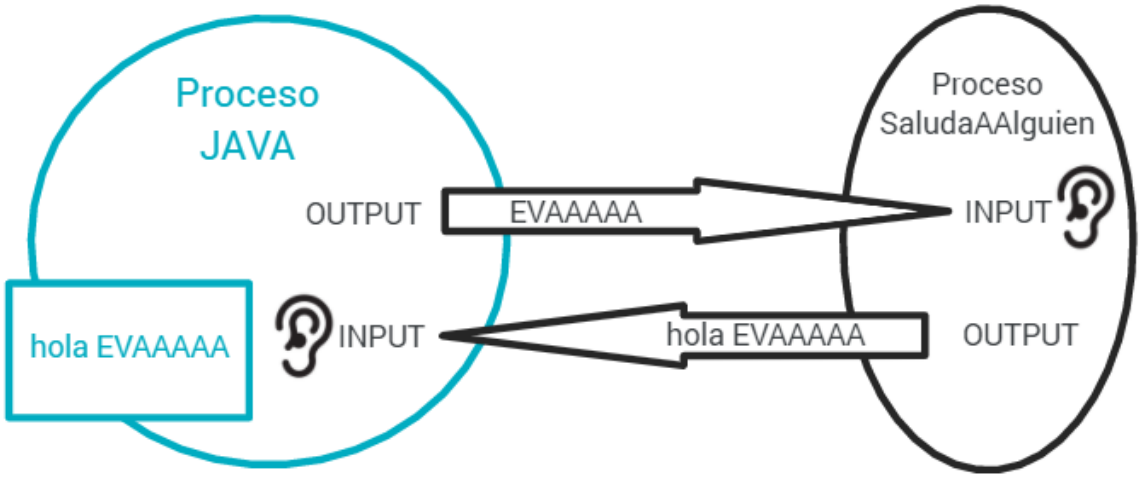
- Dispondremos del control de ese proceso utilizando los métodos de la clase Process
  - Podemos matarlo
  - Esperar a que termine (waitfor)
  - Gestionar lo que lea como entrada
  - Gestionar lo que obtenga como salida
  - …
- Ejemplo un programa que muestra la salida de otro proceso

Procesos con Java

- ¡CUIDADO!
  - getInputStream para leer la **salida** del proceso.
  - getErrorStream para leer la salida de **error** del proceso.
  - getOutputStream para escribir lo que **entrará** en el proceso.
  - **exec(),** aunque ejecuta una línea de comando, **no** actúa exactamente igual que un **interprete de comandos** (no es capaz de interpretar la redirección, símbolo “>”)

Procesos con Java

- Cuando queremos que un proceso reciba una cadena de entrada escribiremos cogiendo el input del proceso (getInputStream)
- Utilizaremos getOutputStream para escribir en la entrada del proceso creado lo que queramos que este proceso lea:



Procesos con Java

2.1.1 Ejecutar el proceso

El siguiente código genera un proceso en Windows, indicando al entorno de ejecución que ejecute el programa. Esta llamada se realiza sin parámetros y sin gestionar de ninguna manera el proceso generado.

```
Runtime.getRuntime().exec("Notepad.exe");
```

El método **exec** puede recibir como parámetro una cadena de caracteres. En dicha cadena, **separados por espacios**, se indicarán, además del programa, los diferentes parámetros.

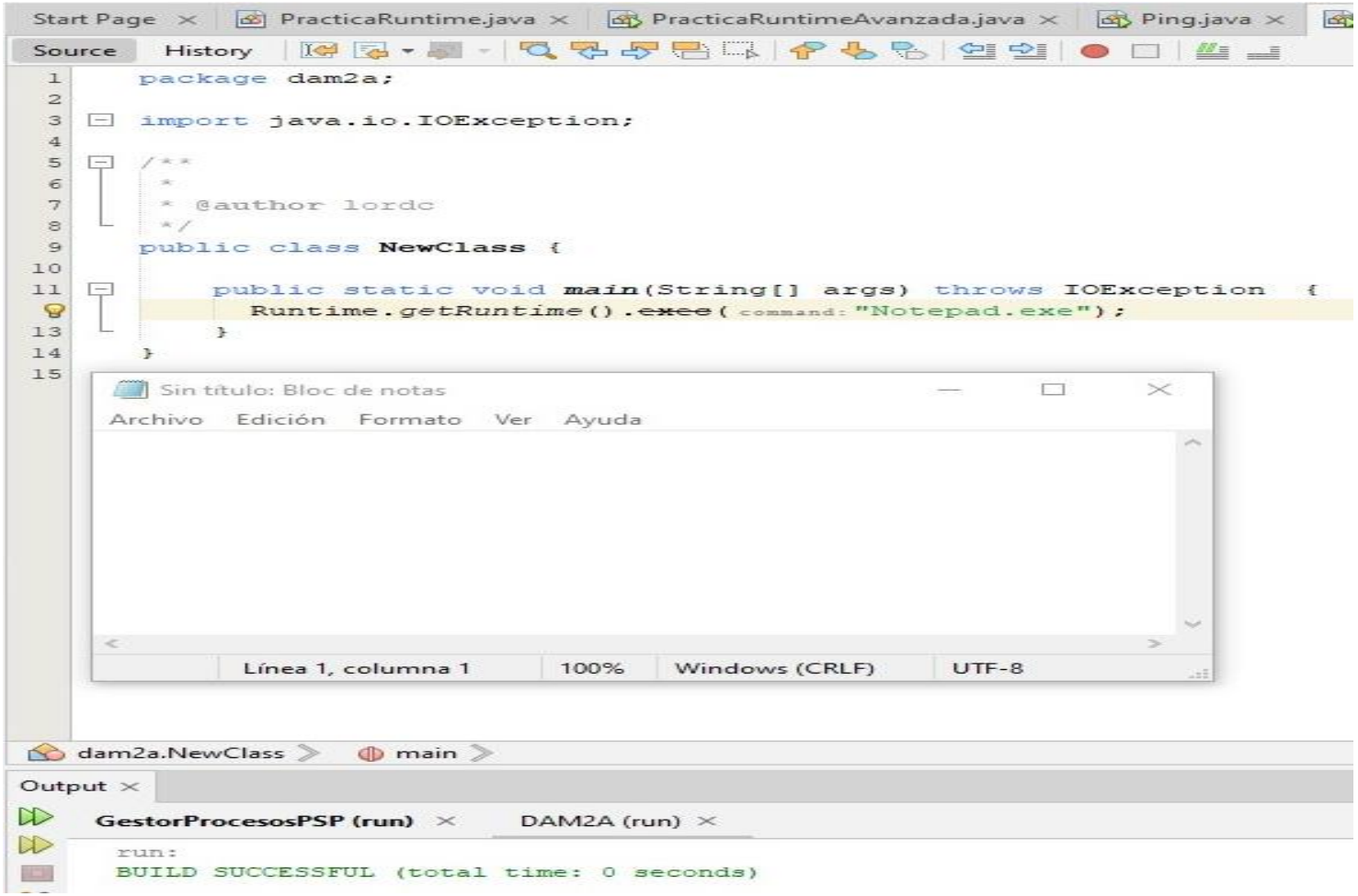
El siguiente código ejecuta el bloc de notas indicando que notas.txt es el fichero que debe abrir o crear si no existe.

```
Runtime.getRuntime().exec("Notepad.exe notas.txt");
```

Alternativamente se puede proporcionar un array de Strings con el programa y los parámetros:

```
String[] infoProceso = {"Notepad.exe","notas.txt"};
Runtime.getRuntime().exec(infoProceso);
```

Procesos con Java



Procesos con Java

- **Ejecución Básica - Sin Parámetros**
- Runtime.getRuntime().exec("Notepad.exe");

1. Runtime.getRuntime(): Obtiene la "máquina virtual" de Java que está corriendo
2. .exec(): Le dice "oye, ejecuta este programa por mí"
3. "Notepad.exe": El programa que queremos lanzar

- **Comportamiento**

1. Abre el Bloc de Notes
2. Java "lanza y olvida" - no controla lo que pasa después
3. El proceso hijo (Notepad) vive independientemente

Procesos con Java

- **Ejecución con Parámetros - String Única**
- Runtime.getRuntime().exec("Notepad.exe notas.txt");

1. "Notepad.exe notas.txt": Programa + parámetro separados por espacio
2. notas.txt: Parámetro que le pasamos a Notepad
    - Si existe: lo abre
    - Si no existe: lo crea

    - "Es como escribir en la línea de comandos: Notepad.exe notas.txt"

Procesos con Java

- **Ejecución con Array - Forma Recomendada**

```
String[] infoProceso = {"Notepad.exe", "notas.txt"};
Runtime.getRuntime().exec(infoProceso);
```

- **Explicación Detallada**

1. **Array de Strings**: Cada elemento es una parte del comando
2. [0]: Siempre el programa ejecutable
3. [1], [2], **...**: Parámetros adicionales

Procesos con Java

¿Por qué esta forma es mejor?

```
// ❌ PROBLEMA con espacios en rutas
Runtime.getRuntime().exec("Notepad.exe \"mi carpeta\\notas.txt\"");

// ✅ SOLUCIÓN con array
String[] comando = {"Notepad.exe", "mi carpeta\\notas.txt"};
Runtime.getRuntime().exec(comando);
```

Procesos con Java

- **Ejemplos Prácticos**

❑ Crea un programa Java que ejecute la calculadora del sistema operativo. Debes implementar DOS versiones diferentes:

➢ Usando un String único con el comando completo

➢ Usando un array de Strings donde cada elemento sea una parte del comando

❑ Desarrolla un programa Java que abra el explorador de archivos del sistema en una carpeta específica.

Procesos con Java

- SOLUCIONES:
  - EJ1

```
// Forma simple
Runtime.getRuntime().exec("calc.exe");

// Forma con array
String[] comando = {"calc.exe"};
Runtime.getRuntime().exec(comando);
```

  - EJ2

```
// Abrir carpeta específica
String[] comando = {"explorer.exe", "C:\\Users\\Alumno\\Documents"};
Runtime.getRuntime().exec(comando);
```

Procesos con Java

2.1.2 Gestión del proceso

Para gestionar el proceso debemos obtener una referencia de la instancia. Si se necesita esperar a que el proceso ejecutado termine se puede utilizar **waitFor()**. Este método suspende la ejecución del programa que ha arrancado el proceso y queda a la espera de que este termine, proporcionando un código de finalización.

```
String[] infoProceso = {"Notepad.exe","notas.txt"};
Process proceso = Runtime.getRuntime().exec(infoProceso);
int codigoRetorno = proceso.waitFor();
System.out.println("Fin de la ejecución: " + codigoRetorno);
```

Procesos con Java

- [INICIO Java]
- ↓
- 1. Prepara: {"Notepad.exe", "notas.txt"}
- ↓
- 2. Ejecuta → Se abre Bloc de notas 📄 con el archivo
- ↓
- 3. Java ESPERA ⏳ **(waitFor())**
- ↓
- [TU TRABAJAS en el Bloc de notas...]
- ↓
- 4. Cierras el Bloc de notas 🚪
- ↓
- 5. Java CONTINÚA ▶️
- ↓
- 6. Muestra: "Fin de la ejecución: 0" ✅
- ↓
- [FIN]

Procesos con Java

¿Qué hace?

- waitFor() → **Detiene** el programa Java hasta que el Bloc de notas se cierre
- codigoRetorno → Guarda un número que indica cómo terminó el programa

Comportamiento

- **Sin** waitFor() → Java sigue ejecutándose inmediatamente
- **Con** waitFor() → Java **se congela** hasta que cierres el Bloc de notas

Procesos con Java

- Otros métodos proporcionados son:
- **destroy()** Destruye el proceso sobre el que se ejecuta
- **exitValue()** Devuelve el valor de retorno del proceso cuando este finaliza.
- **getErrorStream()** Proporciona un InputStream conectado a la salida de error del proceso
- **getInputStream()** Proporciona un InputStream conectado a la salida normal del proceso
- **getOutputStream()** Proporciona un OutputStream conectado a la entrada normal del proceso.
- **isAlive()** Determina si el proceso está en ejecución o no.
- **waitFor()** Detiene la ejecución del programa que lanza el subproceso

Procesos con Java

Ejemplos

Creamos un programa que haga ping a un sitio web y se imprima el resultado por pantalla:

```
Process process = Runtime.getRuntime().exec("ping www.google.com");
BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
String line = "";
while ((line = reader.readLine()) != null) {
   System.out.println(line);
}
```

Determinar si estamos ejecutando el programa sobre Windows o sobre Linux

```
String so = System.getProperty("os.name");
String comando;
// Determinar comando Windows o Linux
if (so.equals("Linux")) {
    // Some code here
} else {
   // Some other code here
}
```

Procesos con Java

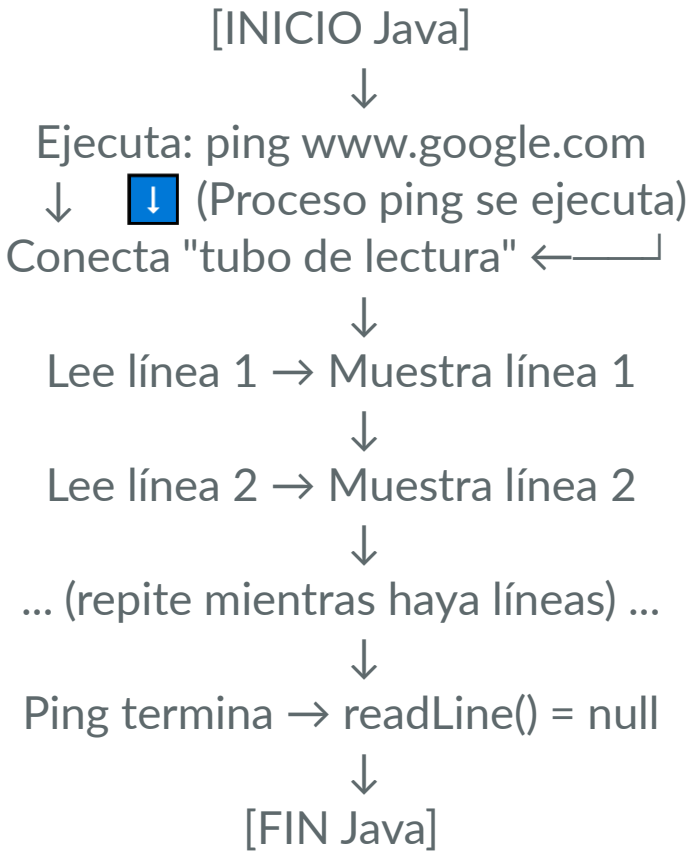
- ping www.google.com

Haciendo ping a www.google.com [142.250.200.4] con 32 bytes de datos:
Respuesta desde 142.250.200.4: bytes=32 tiempo=25ms TTL=116
Respuesta desde 142.250.200.4: bytes=32 tiempo=26ms TTL=116
Respuesta desde 142.250.200.4: bytes=32 tiempo=24ms TTL=116
Respuesta desde 142.250.200.4: bytes=32 tiempo=25ms TTL=116

Estadísticas de ping para 142.250.200.4:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0 (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 24ms, Máximo = 26ms, Media = 25ms

Procesos con Java

- **FLUJO DE EJECUCIÓN**



- **¿Qué hace?**
  - **process.getInputStream()** → Obtiene el "tubo de salida" del proceso ping
  - **InputStreamReader** → Convierte bytes a caracteres (texto)
  - **BufferedReader** → Lee el texto de forma eficiente línea por línea

- HACER LOS EJERCICIOS PROPUESTOS CON LA CLASE RUNTIME

Procesos con Java

- Otra forma de crear procesos , con la clase **ProcessBuilder**.
- a partir del JDK 1.5
- **ProcessBuilder es la forma MODERNA y FLEXIBLE de crear procesos**

Lo que antes escribíamos así:

```
Runtime runtime = Runtime.getRuntime();
Process process = runtime.exec(command);
```

Ahora lo escribimos así:

```
String command[];
Process process = new ProcessBuilder(command).start();
```

Procesos con Java

2.2.1 Creación y ejecución de un proceso

ProcessBuilder, al igual que Runtime, permite crear procesos. La creación más sencilla es indicando el programa a ejecutar.

```
new ProcessBuilder("Notepad.exe");
```

Es importante saber que esta construcción no implica la ejecución. Esta se realiza con strat()

```
new ProcessBuilder("Notepad.exe").start();
```

También admite parámetros que serán enviados al proceso a crear:

```
new ProcessBuilder("Notepad.exe","datos.txt").start();
```

l comando también se puede establecer en tiempo de construcción

```
ProcessBuilder pb = new ProcessBuilder("cmd.exe","/C","dir");
```

Another way to start a process

```
        // Create ProcessBuilder.
        ProcessBuilder p = new ProcessBuilder();
        // Use command "notepad.exe" and open the file.
        p.command("notepad.exe");
        // p.command("notepad.exe", "C:\\file.txt");
        p.start();
```

Procesos con Java

ProcessBuilder (del jdk 1.5) gestiona:

- Comando, como una lista de cadenas.
- Entorno de ejecución y sus variables (en un objeto Map)
- Directorio de trabajo.
- Podemos activar/desactivar la opción de que la salida de error del proceso creado pueda ser redirigida.
- Modificar el objeto processBuilder antes de hacer start(). (no es sencillo)

Procesos con Java

2.2.2 Gestión de un proceso

El método start() proporciona un objeto de tipo Process como resultado de la ejecución.

```
Process proceso = new ProcessBuilder("Notepad.exe").start();
int valorRetorno = proceso.waitFor();
```

Otros métodos proporcionados son:

- **start** inicia un nuevo proceso usando atributos especificados
- **command** permite obtener o asignar el programa y los argumentos de la instancia de ProcessBuilder
- **directory** Permite asignar el directorio de trabajo del proceso
- **environment** Proporciona información sobre el entorno de ejecución del proceso
- **redirectError** Permite determinar el destino de la salida de errores
- **redirectInput** Permite determinar el origen de la entrada estándar
- **redirectOuput** Permite determinar el destino de la salida estándar.
- **inheritIO()** Redirecciona para mostrar el resultado por pantalla.

Procesos con Java

- **Ejemplo 1: Básico - Calculadora**

```
public class EjemploBasico {
    public static void main(String[] args) throws Exception {
        ProcessBuilder pb = new ProcessBuilder("calc.exe");
        Process proceso = pb.start();
        System.out.println("Calculadora abierta. PID: " + proceso.pid());
    }
}
```

Procesos con Java

¿Qué hace?

- ProcessBuilder → Clase moderna para crear procesos (mejor que Runtime.exec)
- new ProcessBuilder("calc.exe") → Crea un "constructor de procesos" para la calculadora
- pb → Variable que guarda la configuración del proceso
- .start() → **Ejecuta realmente** el proceso (abre la calculadora)
- Process proceso → Guarda una referencia al proceso en ejecución

- La calculadora es independiente de Java

Procesos con Java

Secuencia completa

[INICIO Java]
↓
Prepara ProcessBuilder para "calc.exe"
↓
Ejecuta .start() → Se abre Calculadora 🧮
↓
Muestra: "Calculadora abierta. PID: XXXX"
↓
[PROGRAMA JAVA TERMINA]
↓
[Calculadora sigue abierta independientemente] ✅