UT I - PROGRAMACIÓN MULTIPROCESO

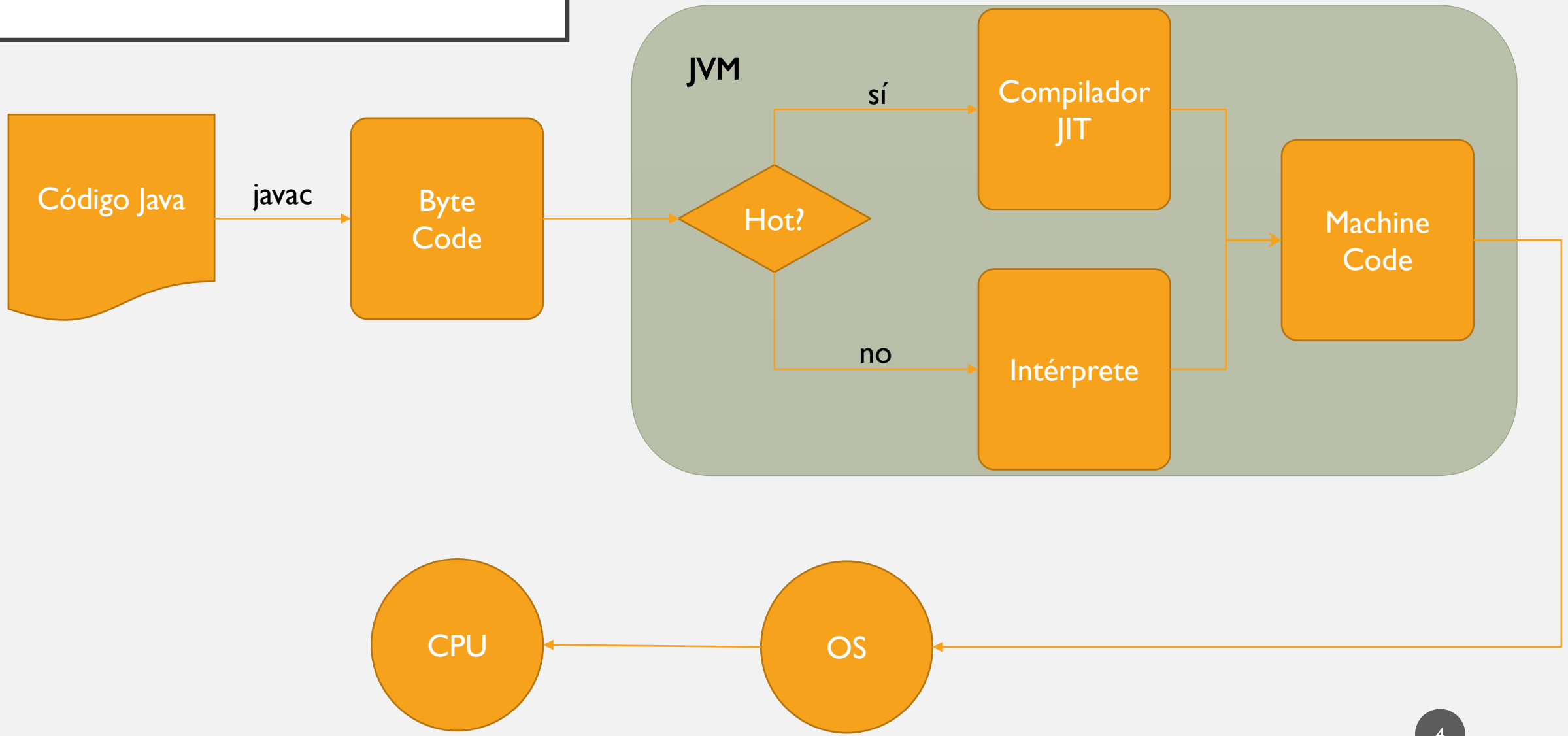
Programación de procesos y servicios

EJECUTABLES. PROCESOS. SERVICIOS.

EJECUTABLES

- Archivo con la estructura necesaria para que el Sistema Operativo ejecute el programa que contiene
- Extensión .exe en Windows
- Un programa en Java: ¿Es un ejecutable?

JVM MODERNA


JVM
Código Java
javac
Byte Code
Hot?
sí
no
Compilador JIT
Intérprete
Machine Code
CPU
OS

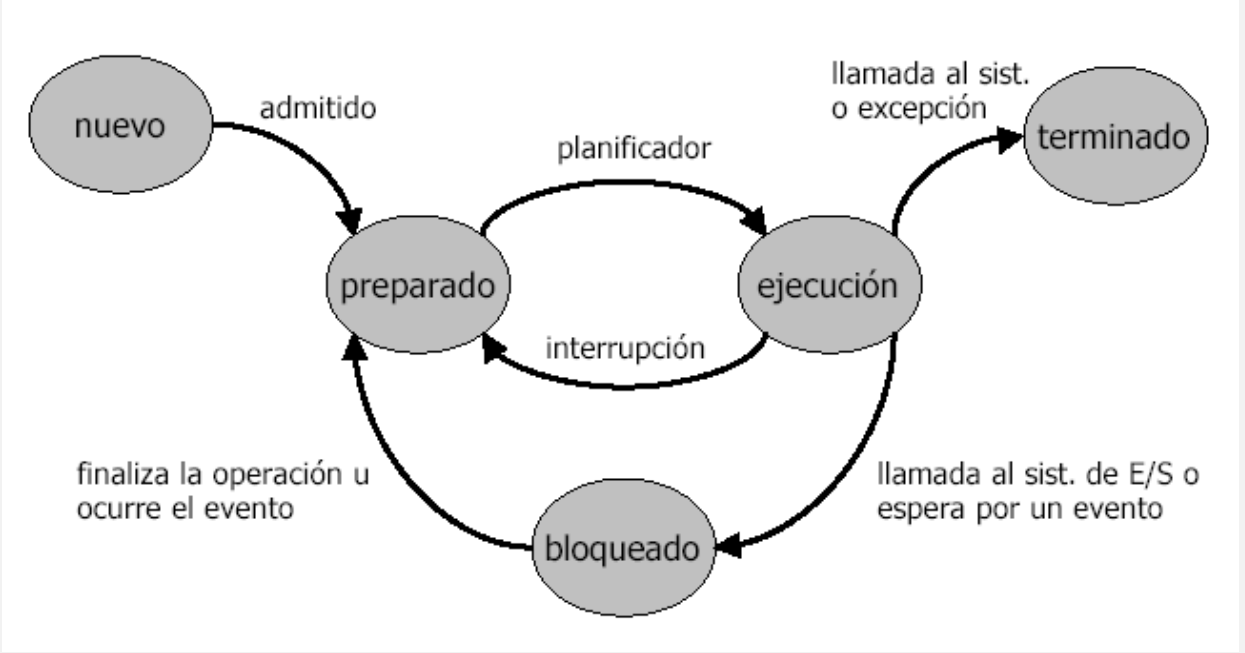
PROCESOS

- Cuando el Sistema Operativo ejecuta un programa lo hace dentro de un PROCESO que necesita de:
  - Tiempo de CPU
  - Memoria
  - Archivos
  - Dispositivos de E/S

PROCESOS

- El proceso consta de
  - El código del programa
  - La actividad actual
    - Contador de programa
    - Registros de CPU
  - Pila
    - Parámetros
    - Variables locales
    - Direcciones de retorno
  - Sección de datos
    - Variables globales
    - Memoria dinámica

ESTADOS


nuevo
admitido
planificador
llamada al sist.
o excepción
terminado
preparado
ejecución
interrupción
finaliza la operación u
ocurre el evento
bloqueado
llamada al sist. de E/S o
espera por un evento

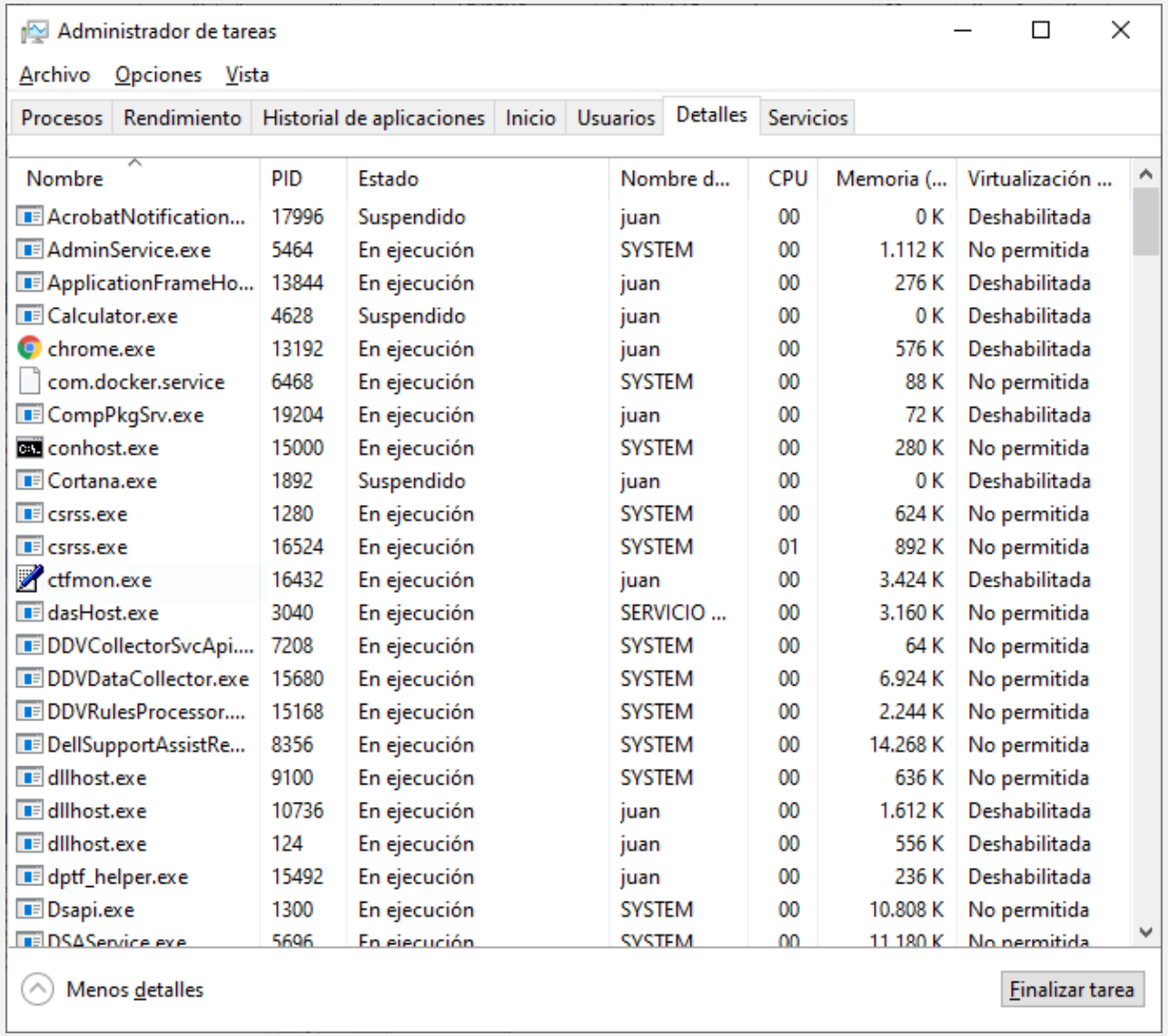
ESTADOS

- El sistema operativo controla los estados de un proceso:
  - Nuevo: el proceso se está creando.
  - Preparado: esperando que se le asigne a un procesador.
  - En ejecución: el proceso está en la CPU ejecutando instrucciones.
  - Bloqueado: proceso esperando a que ocurra un suceso (ej. terminación de E/S o recepción de una señal).
  - Terminado: finalizó su ejecución o falló, por tanto no ejecuta más instrucciones y el SO le retirará los recursos que consume.
- Solo un proceso puede estar ejecutándose en cualquier procesador en un instante dado, pero muchos procesos pueden estar listos y esperando.

SERVICIO

- Un servicio es un proceso que no muestra ninguna ventana ni gráfico en pantalla.
- No está pensado para que el usuario lo maneje directamente.
- Habitualmente, un servicio es un programa que atiende a otro programa.

ADMINISTRADOR DE TAREAS



¿Y JAVA?

¿Qué ocurre cuando ejecutamos un programa escrito en JAVA?

CREACIÓN DE PROCESOS EN JAVA

PROCESS BUILDER

- ProcessBuilder permite crear procesos en el sistema operativo
- Constructores
  - ProcessBuilder(String... command)
  - ProcessBuilder(List<String> command)
  - command es lo mismo que escribiríamos en la consola del sistema: el nombre del programa seguido por los argumentos del programa
- Métodos
  - start(), crea y devuelve un Process con los parámetros pasados al constructor del ProcessBuilder

VARARGS

- Argumentos de longitud variable
- Indicados como …
- Solo en la última posición de los parámetros de una función
- Acepta array, lista, o secuencia de argumentos separados por comas

PROCESS

- Process representa a un proceso del sistema operativo
- No se construye directamente, siempre a través de ProcessBuilder
- Métodos
  - isAlive: nos dice si el proceso está vivo
  - pid: el identificador de proceso
  - waitFor: bloquea el hilo actual esperando a que termine el proceso (admite timeout)
  - exitCode: el código de salida del proceso

PRACTICAMOS

1. CrearProceso. Crear un proceso en Java que lance un programa externo.
2. SumaConcurrente. Crear varios procesos concurrentes para repartir trabajo entre los procesadores.
3. SumaConcurrenteSalida. Crear varios procesos concurrentes para repartir trabajo entre los procesadores y recuperar su salida.

CREAR PROCESO

CREAR PROCESO

- Utilizando la clase ProcessBuilder lanza un programa que esté instalado en tu máquina

1. Localiza la ruta al programa a ejecutar (por ejemplo Acrobat Reader)
2. Crea el ProcessBuilder pasándole la ruta
3. Crea un Process llamando a start en el ProcessBuilder
4. Imprime el PID del proceso por consola
5. Espera a que termine el proceso
6. Imprime el código de salida del proceso por pantalla

CREAR PROCESO CON ARGUMENTOS

- Modifica el ejemplo anterior para abrir un fichero con Acrobat Reader
- Debes añadir la ruta al fichero como un segundo argumento del constructor de ProcessBuilder que, recuerda, recibe varargs

SYSTEM.GETPROPERTY

- Utilidad para obtener información del entorno de ejecución
- “user.dir” es el directorio de trabajo
  - El directorio desde donde se llama al comando java en una consola
  - El directorio raíz del proyecto cuando ejecutamos desde IntelliJ

SUMA CONCURRENTE

SUMA CONCURRENTE: OBJETIVO

Lanzar varios procesos en paralelo para utilizar toda la potencia de los núcleos del procesador

¿QUÉ PROCESOS VAMOS A LANZAR?

Vamos a lanzar varios procesos cuyo código hemos escrito en JAVA y que realiza una suma de una serie de números

¿CÓMO SE LANZA UN PROCESO EN JAVA?

Es necesario ejecutar el comando

`java`

¿DÓNDE ESTÁ EL CÓDIGO DE LOS PROCESOS QUE QUEREMOS LANZAR?

En la clase `Adder` **del fichero** `Adder.java` **dentro del mismo proyecto que nuestro** `Main.java`

¿ENTONCES CÓMO LANZO EL PROCESO?

Usamos el comando `java` al que le pasamos el fichero ya compilado `Adder.class`

¿Y CÓMO SABE EL COMANDO `java` DÓNDE ESTÁ EL FICHERO `Adder.class`?

Usamos el parámetro `-classpath` o `-cp` pasando la **ruta absoluta** al fichero `Adder.class` **hasta** el directorio que contiene el **primer elemento** del **nombre de paquete** de la clase

¿Y CÓMO SABE EL COMANDO `java` DÓNDE ESTÁ EL FICHERO `Adder.class`?

1. Si mis `.class` están en `c:\java\MyClasses\utility\myapp`
2. Y mis clases están en el paquete `utility.myapp`
3. Entonces el classpath tiene que ser `c:\java\MyClasses\`
4. Y para lanzar la clase `Cool` que está en el paquete `utility.myapp`
5. La orden es

```
java -cp c:\java\MyClasses\ utility.myapp.Cool
```

¿CÓMO PONGO TODO ESTO JUNTO PARA LANZAR COOL CON PROCESSBUILDER?

Recuerda que a `ProcessBuilder` le pasamos cada uno de los 4 elementos del comando separados por comas

```
ProcessBuilder processBuilder = new
ProcessBuilder("java","-cp",
"c:\java\MyClasses\","utility.myapp
.Cool");
```

Y SI ESTOY USANDO UN IDE ¿DÓNDE ESTÁN LOS `.class`?

Para encontrar dónde está la raíz de tu proyecto usa
`System.getProperty("user.dir")`

Después concatena la ruta hasta donde tu IDE pone los ficheros compilados `.class`

CONOCE TU IDE
es tu herramienta de trabajo

¿QUÉ TIENE QUE TENER LA CLASE QUE QUIERO EJECUTAR EN EL PROCESO?

`main`

¿Y SI QUIERO PASARLE PARÁMETROS A LA CLASE?

```
public static void
main(String[] args)
```

SUMA CONCURRENTE. SALIDA

STREAMS

- En java, para **leer** datos desde teclado, desde un fichero o desde un socket, y para **escribir** datos por pantalla, a un fichero o a un socket utilizamos Streams (flujos)
- Por defecto un proceso tiene tres Streams disponibles
- System.out: la salida estándar (generalmente la consola)
  - Instancia de PrintStream →print(),println()
- System.in: la entrada estándar (generalmente la consola)
  - Instancia de InputStream →read(), devuelve byte
- System.err: la salida de error (generalmente la consola)

READERS

- Los métodos read de un InputStream (como System.in) devuelven bytes
- Si queremos leer caracteres utilizamos InputStreamReader

```
InputStreamReader inputStreamReader = new  InputStreamReader(System.in);
char[] array = new char[100];
int read = inputStreamReader.read(array, 0, 100);
```

BUFFERED READERS

- Leer utilizando buffers de caracteres es incómodo
- Podemos utilizar BufferedStreamReader creado a partir de un InputStreamReader para leer líneas completas

```
InputStreamReader inputStreamReader = new InputStreamReader(System.in);
BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
bufferedReader.readLine();
```

WRITERS

- System.out es fácil de usar, pero no me sirve si quiero escribir por un Stream que no sea la salida estándar, por ejemplo a un fichero o a un socket, o en el caso de comunicación entre procesos.
- Si tenemos un OutputStream (trabaja con bytes), podemos utilizar un OutputStreamWriter para escribir caracteres.
- Análogamente a las operaciones de lectura, podemos utilizar un PrintWriter que trabaja con cadenas. ¡Recuerda usar autoflush!

```
OutputStream outputStream;

OutputStreamWriter outputStreamWriter = new OutputStreamWriter(outputStream);

PrintWriter printWriter= new PrintWriter(outputStreamWriter,true);

printWriter.println("hey");
```

LIMPIEZA

- Debemos liberar correctamente los recursos del sistema, especialmente ficheros y sockets, cuando ya no los necesitemos.
- Los Streams, Readers, Writers y derivados tienen un método **close** para liberar los recursos utilizados.
- El compilador se encarga de añadir la operación de cerrado en un **finally** si utilizamos try-with-resources

LEER DATOS DE UN SUBPROCESO

- El proceso padre puede obtener un InputStream para **leer** datos desde un proceso hijo llamando a Process.getInputStream.
- A partir de ese InputStream creamos InputStreamReader y BufferedReader según nuestras necesidades.
- El proceso hijo utilizará System.out para escribir datos, pero estos no llegan a la salida estándar a la consola, sino al InputStream del padre

ENVIAR DATOS A UN SUBPROCESO

- El proceso padre puede obtener un OutputStream para **escribir** datos hacia un proceso hijo llamando a Process.getOutputStream.
- A partir de ese OutputStream creamos OutputStreamReader y PrintWriter según nuestras necesidades.
- El proceso hijo utilizará System.in para leer los datos, pero no estará leyendo desde teclado sino los datos que le envíe el proceso padre