1 PROGRAMACIÓN DE PROCESOS

OBJETIVOS DE LA UNIDAD:

- Programar y controlar los procesos desde Java.

1. Métodos clase Process

1. Métodos clase Process

Método	Funcionalidad
`void destroy()` `public Process destroyForcibly()`	Termina el proceso. El primer método permite una terminación limpia y ordenada del proceso. El segundo lo termina inmediatamente.
`int exitValue()`	Devuelve el valor de salida, o código de retorno, del proceso. Por convención, un valor 0 indica terminación normal, y otro valor se interpretará como un código de error. Se puede terminar un programa en Java con un código de retorno distinto de 0 con `System.exit(código)`.
`ProcessHandle.Info info()`	Devuelve la información actual del proceso.
`boolean isAlive()`	Comprueba si el proceso está vivo.
`long pid()`	Devuelve el PID o identificador de proceso.

1. Métodos clase Process

1. Métodos clase Process

`int waitFor()`	Hace que el hilo en ejecución espere hasta que el proceso haya terminado. Devuelve el valor de salida del proceso. Un valor cero se entiende que corresponde a una ejecución sin errores, mientras que un valor distinto de cero corresponde a un código de error. Si el proceso es de un programa en Java, es el valor devuelto por `System.exit()`, o cero si no se terminó la ejecución con `System.exit()`.
`boolean waitFor(long timeout, TimeUnit unit)`	Hace que el hilo en ejecución espere hasta que el proceso haya terminado, durante un tiempo máximo indicado por `timeout`. Devuelve `true` si el proceso ha terminado por sí mismo antes del tiempo máximo indicado, y `false`en caso contrario.

2. Métodos Redirección ProcessBuilder

2.1 FICHEROS: Métodos de redirección para clase ProcessBuilder de Java. Relación con línea de comandos Linux

Métodos	Línea de comandos de Linux
`redirectInput(new File (fichero))`	*comando < fichero*
`redirectOutput(new File(fichero))`	*comando > fichero*
`redirectOutput(Redirect.appendTo(new File(fichero)))`	*comando >> fichero*
`redirectOutput(Redirect.DISCARD)`	*comando > /dev/null*
`redirectError(new File(fichero))`	*comando 2> fichero*
`RedirectError(Redirect.appendTo(new File(fichero)))`	*comando 2>> fichero*
`redirectError(Redirect.DISCARD))`	*comando 2> /dev/null*

2. Métodos Redirección ProcessBuilder

2.1 FICHEROS: Métodos de clase ProcessBuilder para redirección de entrada y salida

Método	Funcionalidad
`ProcessBuilder inheritIO()`	Redirige la salida estándar y de error de los subprocesos creados hacia las del proceso padre, y su entrada estándar desde la del proceso padre.

`ProcessBuilder redirectInput(File f)`
`ProcessBuilder redirectOutput(File f)`
`ProcessBuilder redirectError(File f)`
	Redirige las entrada y salida estándares y de error, respectivamente, desde o hacia:
	– Un fichero (con `File f`).
	– La correspondiente del proceso padre (con `Redirect.INHERIT`).
	– La salida estándar y de error se pueden descartar (con `Redirect.DISCARD`).

2. Métodos Redirección ProcessBuilder

2.1 EJEMPLO Redireccionamiento con Java.

```java
package procredirfichafich;

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

public class ProcRedirFichAFich {

  public static void main(String[] args) {

    if (args.length < 3) {
      System.out.println("ERROR: indicar: fichero_entrada fichero_salida patron");
      return;
    }

    String nomFichEntrada = args[0];
    String nomFichSalida = args[1];
    String patron = args[2];

    System.out.printf("Buscando patrón \"%s\" en fichero %s, salida a fichero %s\n",
            patron, nomFichEntrada, nomFichSalida);

    ProcessBuilder pb = new ProcessBuilder("grep", patron);
    pb.redirectInput(new File(nomFichEntrada));
    pb.redirectOutput(new File(nomFichSalida));

    try {
      Process p = pb.start();
    } catch (IOException e) {
      System.out.println("ERROR: de E/S");
      e.printStackTrace();
    }

  }

}
```

2. Métodos Redirección ProcessBuilder

2.1 Ejecución de procesos (Programas de java)

Una opción interesante en la programación de los procesos en Java, es la posibilidad de crear el proceso hijo a ejecutar en **una clase de Java**.

De esta forma tendremos la clase principal **(Lanzador)** y luego el programa en **java (Proceso hijo),** que hemos desarrollado en java.

En realidad lo único que tenemos que hacer es lanzar desde el programa Lanzador el comando

```
Pb= new ProcessBuilder("java", "-cp" ,"build\\classes\\",clase,arg1,arg2,…..)
```

En la siguiente diapositiva tienes un **ejemplo**, donde el proceso hijo hace la suma de los números incluidos en un rango. Dicho rango se indica en el proceso padre y el hijo lo ejecuta.

2. Métodos Redirección ProcessBuilder

2.1 Ejemplo Proceso hijo en java

Proceso Padre

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

public class Lanzador {
    public static void main(String[] args)
    {
        ProcessBuilder pb;
        try {
            pb = new ProcessBuilder(
                "java", "-cp", "build\\classes\\","Sumador",args[0],args[1]);
            pb.redirectError(new File("errores.txt"));
            pb.redirectOutput(new File("resultado.txt"));
            pb.start();
        }
        catch (IOException e)
        {
            System.out.println("Error acceso fichero");
        }
    }
}
```

Proceso hijo

```
public class Sumador
{
    public static void main(String[] args)
    {
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int resultado=0;
        for (int i=n1;i<=n2;i++)
            resultado=resultado+i;
        System.out.println(resultado);
    }
}
```

2. Métodos Redirección ProcessBuilder

2.2 Métodos de redirección a streams de la clase Process.

En el contexto de la **programación de procesos en Java**, los streams desempeñan un papel fundamental. Estos **conductos virtuales** permiten la transferencia eficiente de datos entre el programa principal y los procesos secundarios o fuentes y destinos externos.

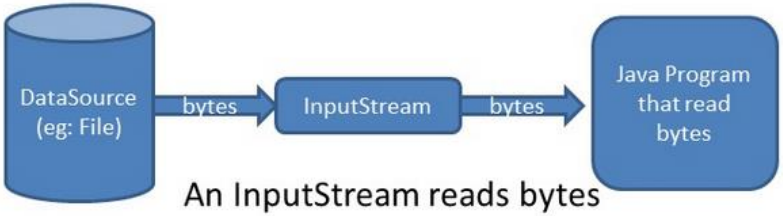
Métodos de `Process` para obtener *streams* asociados a entrada estándar y a salidas estándar y de error

Método	Funcionalidad
`InputStream getInputStream()`	Devuelve un *stream* de entrada conectado con la salida estándar del proceso.
`OutputStream getOutputStream()`	Devuelve un *stream* de salida conectado con la entrada estándar del proceso.
`InputStream getErrorStream()`	Devuelve un *stream* de entrada conectado con la salida de error del proceso.

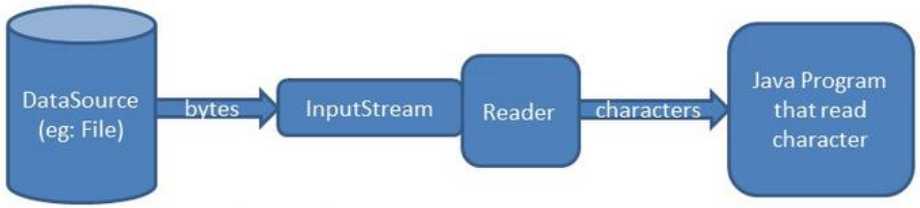
2. Métodos Redirección ProcessBuilder

2.2.1 getInputStream()

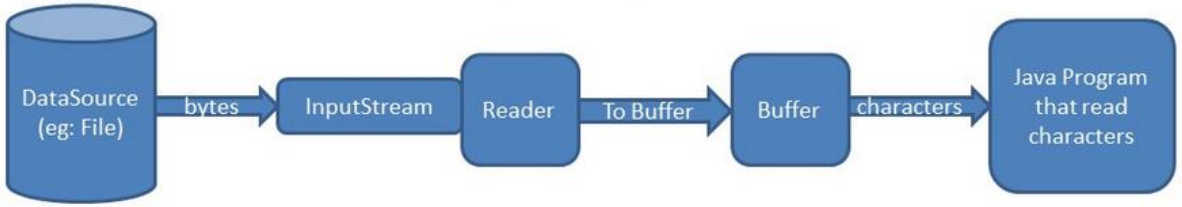
Devuelve un **stream de entrada al proceso padre** conectado con la **salida estándar** del proceso hijo. Para poder tratar el stream en java se usa en combinación de **InputStreamReader() y BufferStreamReader()**



An InputStream reads bytes



A Reader works with an InputStream and reads characters



2. Métodos Redirección ProcessBuilder

2.2.1 Ejemplo de uso de getInputStream()

```
1  Process p = pbuilder.start();
2  BufferedReader processOutput =
3      new BufferedReader(new InputStreamReader(p.getInputStream()));
4
5  String linea;
6  while ((linea = processOutput.readLine()) != null) {
7      System.out.println("> " + linea);
8  }
9  processOutput.close();
```

2. Métodos Redirección ProcessBuilder

2.2.2 getErrorStream()

Podemos usar un **schema similar al usado anteriormente**, con la salvedad de que ahora en vez de llamar a getInputStream() lo hacemos con **getErrorStream().**

```java
Process p = pbuilder.start();
BufferedReader processError =
    new BufferedReader(new InputStreamReader(p.getErrorStream()));
// En este ejemplo, por ver una forma diferente de recoger la información,
// en vez de leer todas las líneas que llegan, recogemos la primera línea
// y suponemos que nos han enviado un entero.
int value = Integer.parseInt(processError.readLine());
processError.close();
```

2. Métodos Redirección ProcessBuilder

2.2.3 getOutputStream()

No sólo podemos recoger la información que envía el proceso hijo sino que, además, también podemos **enviar información desde el proceso padre al proceso hijo.**

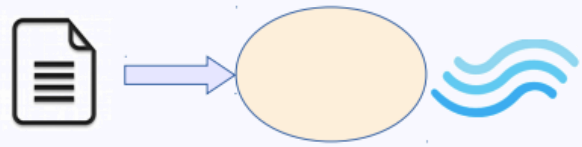
```java
PrintWriter toProcess = new PrintWriter(
    new BufferedWriter(
        new OutputStreamWriter(
            p.getOutputStream(), "UTF-8")), true);
toProcess.println("sent to child");
```

Un ejemplo de stream de entrada en java sería **System.in,** que representa la entrada estándar por teclado.

```
// Creamos un objeto BufferedReader para leer desde System.in
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

2. Métodos Redirección ProcessBuilder

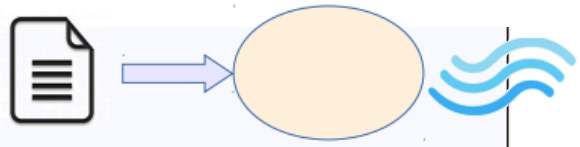
2.2.4 Ejemplo completo (Entrada Fichero y salida Stream).



```
package procentradafichsalidastream;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class ProcEntradaFichSalidaStream {

  public static void main(String[] args) {

    if (args.length < 2) {
      System.out.println("ERROR: indicar: fichero_entrada patron");
      return;
    }
    String nomFichEntrada = args[0];
    String patron = args[1];

    System.out.printf("Buscando patrón \"%s\" en fichero %s, se lee salida de stream\n",
            patron, nomFichEntrada);

    ProcessBuilder pb = new ProcessBuilder("grep", patron);

    // Entrada desde fichero
    pb.redirectInput(new File(nomFichEntrada));
```

2. Métodos Redirección ProcessBuilder

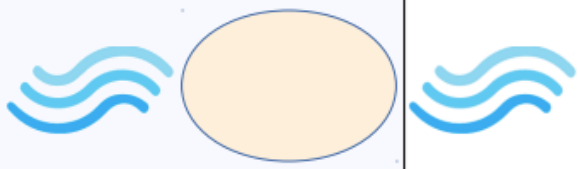
2.2.4 Ejemplo completo (Entrada Fichero y salida Stream).



```
    try {

      Process p = pb.start();
      p.waitFor();  // Hay que esperar a que termine antes de leer salida

      try (InputStream is = p.getInputStream();
              InputStreamReader isr = new InputStreamReader(is, "UTF-8");
              BufferedReader br = new BufferedReader(isr)) {
        String linea;
        int i = 1;
        while ((linea = br.readLine()) != null && linea.length() > 0) {
          System.out.printf("%d: [%s]\n", i++, linea);
        }
      }

    } catch (InterruptedException e) {
      System.out.println("INFO: proceso interrumpido");
      e.printStackTrace();
    } catch (IOException e) {
      System.out.println("ERROR: de E/S");
      e.printStackTrace();
    }
  }
}
```

2. Métodos Redirección ProcessBuilder

2.2.5 Ejemplo completo (Entrada Stream y salida Stream).



```
package procentradadestreamsalidaastream;

import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.IOException;

// Lee línea a línea de entrada estándar. Para cada línea leida, calcula el hash md5.
// La salida se obtiene con un stream, para poder eliminar el guión que el comando md5sum escribe
al final
public class ProcEntradaDeStreamSalidaAStream {

  public static void main(String[] args) {

    ProcessBuilder pb = new ProcessBuilder("md5sum");

    try (InputStreamReader isstdin = new InputStreamReader(System.in, "UTF-8"); // Lee
línea a línea de entrada estándar
            BufferedReader brstdin = new BufferedReader(isstdin)) {

      String linea;
```

2. Métodos Redirección ProcessBuilder

2.2.5 Ejemplo completo (Entrada Stream y salida Stream).

```
        while ((linea = brstdin.readLine()) != null && linea.length() != 0) {

          Process p = pb.start();                          // Lanza el proceso

          try (OutputStream osp = p.getOutputStream();
                  OutputStreamWriter oswp = new OutputStreamWriter(osp, "UTF-8")) {
            oswp.write(linea);  // Envía línea leída al proceso
          }
          try {

            p.waitFor();          // Espera a que termine antes de leer salida

            try (InputStream isp = p.getInputStream();
                    InputStreamReader isrp = new InputStreamReader(isp);
                    BufferedReader brp = new BufferedReader(isrp)) {

              String salidaProc = brp.readLine();  // Salida proceso es una sola línea

              String[] cadenas = salidaProc.split(" ");
              System.out.printf("%s\n", cadenas[0]);
            }
          } catch (InterruptedException e) {
          }
          System.out.print("Introduce línea:");
        }
      } catch (IOException e) {
        System.out.println("ERROR: de E/S");
        e.printStackTrace();
      }
    }
}
```