PROCESOS EN JAVA (TEMA 1)

Santiago Rodenas Herráiz (srodher115@g.educaand.es)

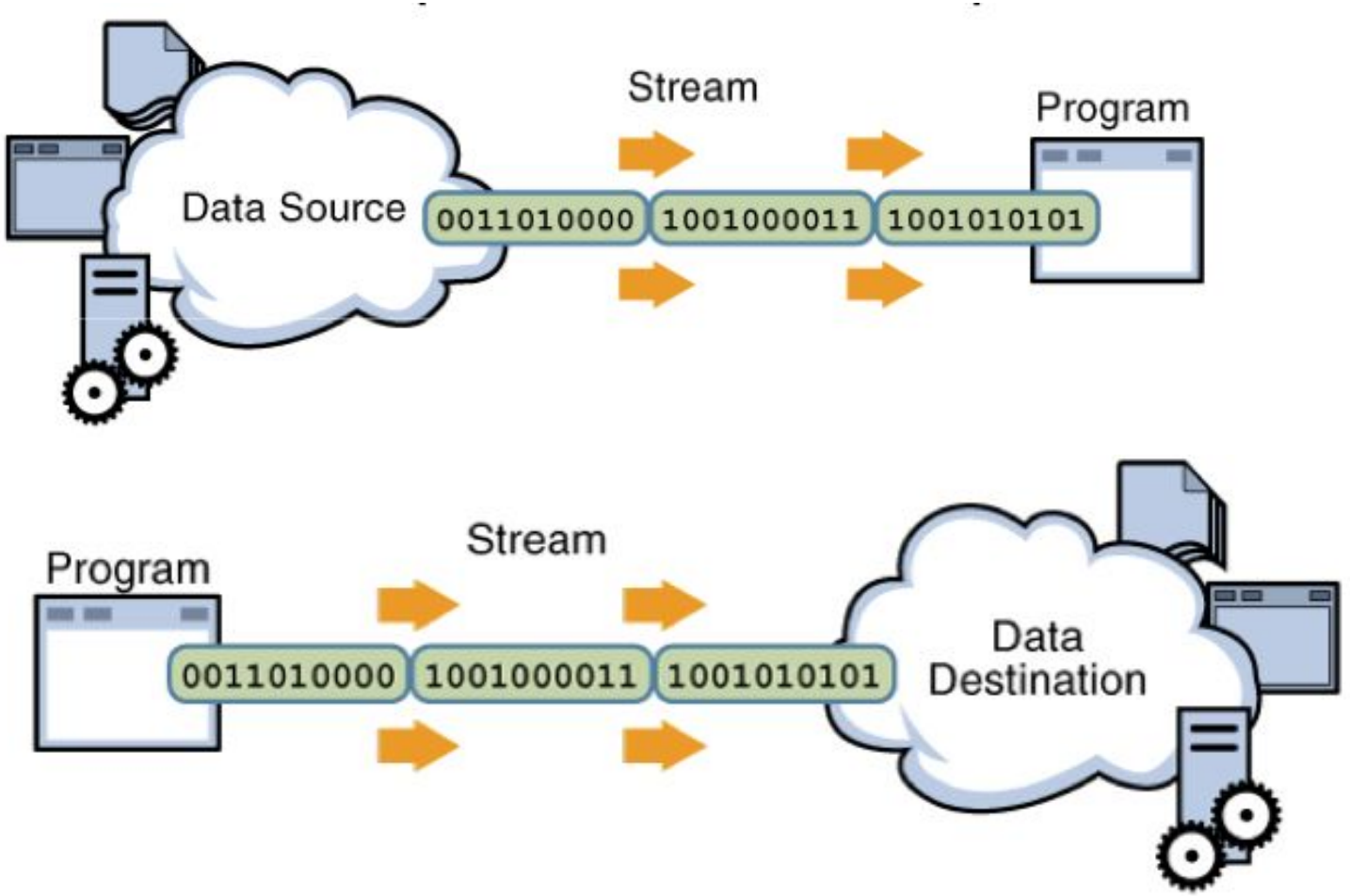
IES VIRGEN DEL CARMEN (Departamento de Informática)

Curso 21/22

STREAM EN PROCESOS.

- Un Stream I/O representa una fuente de entrada/salida.
- Un Strem es una secuencia de bytes a/desde un dispositivo,
- Pueden ser ficheros en disco, dispositivos, otros programas, etc.
- Soportan varios tipos de datos:
  - Bytes, tipos de datos primitivos, caracteres.

STREAM EN PROCESOS.



STREAM EN PROCESOS.

- Cada proceso en Java, incorpora tres variables de una instancia de la clase java.lang.System. Son variables estáticas y son flujos de bytes.
  - System.in implementa la entrada estándar (teclado)
  - System.out implementa la salida estándar (pantalla)
  - System.err implementa la salida de error estándar (pantalla)

STREAM EN PROCESOS.

```
import java.io.*;

class LecturaDeLinea {
      public static void main( String args[] ) throws IOException {
         int c;
         int contador = 0;
         // se lee hasta encontrar el fin de línea
         while( (c = System.in.read() ) != '\n' )
            {
                     contador++;
                     System.out.print( (char) c );
            }
         System.out.println();    // Se escribe el fin de línea
         System.err.println( "Contados "+ contador +" bytes en total." );
      }
}
```

STREAM EN PROCESOS

- Disponemos de Byte Streams y de Character Streams.
- Los Streams de bytes s utilizan para entrada/salida de bytes, por ejemplo ficheros.
  - Disponemos de dos clases abstractas:
    - InputStream, y OutputStream.
      - Subclase BufferedInputStream.- Contiene métodos para leer bytes del buffer (área de memoria)
      - Subclase BufferedOutputStream.- Contiene métodos para escribir bytes en el buffer (área de mem.).
      - FileInputStream.- Contiene métodos para leer bytes de un fichero.
      - FileOutputStream.- Contiene métodos para escribir en un fichero.
      - ……

STREAM EN PROCESOS

- Los Streams de carácter s utilizan para entrada/salida de caracteres, por ejemplo un buffer.
  - Disponemos de dos clases abstractas:
    - Reader y Writer.
      - Subclase BufferedReader .- Contiene métodos para leer caracteres en un buffer.
      - Subclase BufferedWriter .- Contiene métodos para escribir caracteres en un buffer.
      - Subclase FileReader .- Contiene métodos para leer caracteres de un fichero
      - Subclase FileWriter .- Contiene métodos para escribir caracteres de un fichero.
      - Subclase InputStremReader .- Contiene métodos para convertir bytes y caracteres.
      - Subclase OutputStreamReader .- Contiene métodos para convertir caracteres a bytes.

STREAM EN PROCESOS

- Los **Streams Actualizados:**
  - Clase **PrintWriter** del paquete **java.io.PrintWriter**
    - A diferencia con los stream de salidas anteriores, esta clase puede convertir directamente los tipos de datos básicos de Java y otros datos a los caracteres correspondientes bajo la codificación predeterminada del sistema, y luego generarlos.
    - Acepta un OutputStream como parámetro.
    - Es ideal para trabajar con ficheros.
  - Clase **Scanner** del paquete **java.util**
    - Es una manera muy sencilla de poder leer datos de manera primitiva por teclado (enteros, reales, etc).
    - Acepta como parámetro System.in
    - También podemos leer directamente desde un fichero.
    - Métodos:
      - nextInt, nextShort, nextFloat, nextDouble
      - nextLine
      - hasNextBoolean, hasNextInt, hasNextLine

STREAM EN PROCESOS

- Ejemplo de lectura de texto desde un fichero Flujos clásicos.

```
try {
    BufferedReader reader =
           new BufferedReader(new
  FileReader("nombrefichero"));
    String linea = reader.readLine();
    while(linea != null) {
       // procesar el texto de la línea
  linea = reader.readLine();
    }
    reader.close();
}
catch(FileNotFoundException e) {
    // no se encontró el fichero
}
catch(IOException e) {
    // algo fue mal al leer o cerrar el fichero
}
```

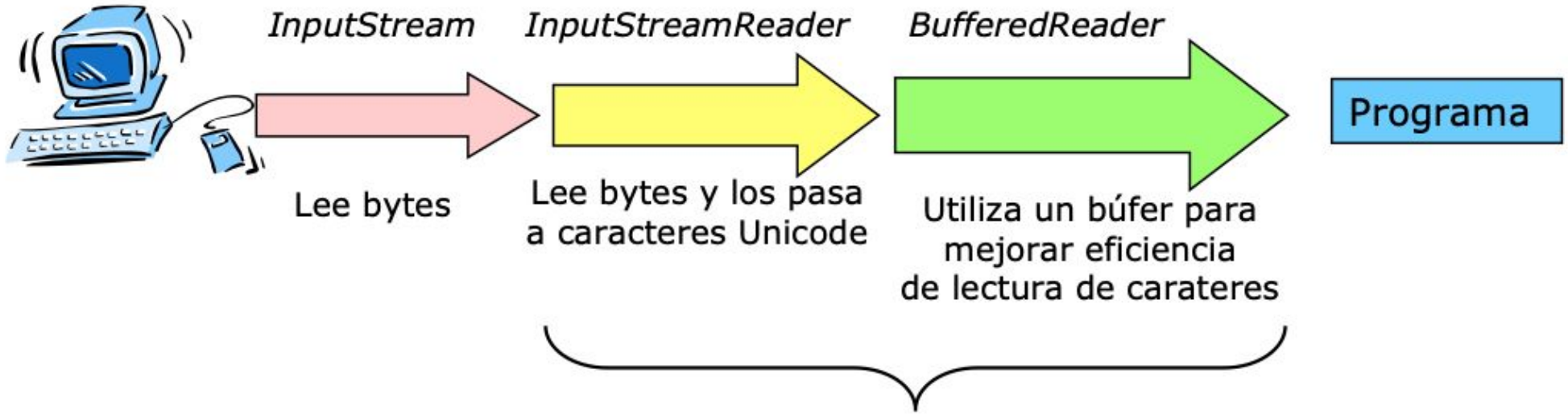
STREAM EN PROCESOS

- Ejemplo de lectura de texto desde un fichero Flujos actualizados.

```java
/**
 * Ejemplo escritura en fichero
 * de la lectura utilizando Scanner
 * y PrintWriter
 */
//package procesos java;

import java.io.FileNotFoundException;
import java.util.Scanner;
import java.io.PrintWriter;

public class lecturaFlujoEntrada {


    public static void main(String[] args) {
        Scanner in =new Scanner(System.in);
        PrintWriter salida=null;
        String linea=null;

        try{
            salida = new PrintWriter("/tmp/salida.out");
            while ((linea = in.nextLine()) != null  && linea.length() != 0) {
                salida.println(linea);
            }
            salida.flush();  //limpiamos buffer del /n
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        } finally {
            salida.close();
        }
    }

}
```

STREAM EN PROCESOS

- Ejemplo de lectura de texto a partir de un flujo de entrada de bytes..



STREAM EN PROCESOS

- Ejemplo de lectura de texto a partir de un flujo de entrada de bytes..

```
import java.io.*;

public class Eco {
  public static void main (String[] args)  {
     BufferedReader entradaEstandar = new BufferedReader
                    (new InputStreamReader(System.in));

     String mensaje;

     System.out.println ("Introducir una línea de texto:");
     mensaje = entradaEstandar.readLine();
     System.out.println ("Introducido: \"" + mensaje + "\"");
  }
}
```

RUNTIME

- Java incorpora un conjunto de clases para la gestión de procesos tanto de usuario.
- Cualquier proceso de JAVA, **posee o se asocia con un objeto RUNTIME** que le da información. Este se puede recuperar con el método estático getRuntime()
- Con la clase **RunTime**, podemos abrir otros programas, por ejemplo notepad, internet explorer o ejecutar comandos del S.O.
- Contiene varios métodos:
  - **getRutime()**.- Devuelve un objeto del tipo Runtime asociado a la aplicación actual en ejecución.
  - **Exec()**.- Método que permite abrir programas externos al que estamos ejecutando. Permite crear nuevos procesos, por tanto devuelve un objeto de tipo **Process**.
  - Puede producir un tipo de excepción **Exception**

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Runtime.html

RUNTIME

- Ejemplo de Método Exec()

```
try
{
  /* directorio/ejecutable es el path del ejecutable y un nombre */
  Process p = Runtime.getRuntime().exec ("directorio/ejecutable");
}
catch (Exception e)
{
  /* Se lanza una excepción si no se encuentra en ejecutable o el fichero no es
ejecutable. */
}
```

RUNTIME

- Problemas con el comando exec()
  - Cuando desde Java llamamos a otro proceso, su salida por pantalla **ya no existe**.
  - Para leer lo que está pasando en el ejecutable externo, debemos usar el objeto del tipo **Process** que nos devuelve la Runtime.exec().
  - Process representa de alguna forma, lo que está pasando en el proceso externo y de ahí podemos visualizarlo.

- ¿Podemos recuperar la información que manda el nuevo proceso a la instancia inicial de Java?
  - Si, pero necesitamos asociar **stream** entre ambos procesos.
  - La idea es asociar ambos procesos como si se tratara de una pipe o tubería. Si recordamos un comando del tipo ps aux | grep python

PROCESS

- La clase Process, permite crear un proceso.
- Métodos de Process para obtener stream asociados a entrada estándar y a salidas estándar y de error:
  - **InputStrem getInputStream()**.- Devuelve un **stream** de entrada al proceso que creó el Process, conectado con la salida estándar del nuevo proceso. Podemos utilizar la clase Scanner y lo veremos más adelante.
  - **OutputStream getOutputStream()**.- Devuelve un **stream** de salida conectado con la entrada estándar del nuevo proceso. Podemos utilizar la clase PrintWriter y lo veremos maś adelante.
  - **InputStream getErrorStream()**.- Devuelve un **stream** de entrada conectado con la salida de error del proceso.

  - La idea es que si desde un programa java queremos recuperar la salida de un nuevo proceso del tipo Process, debemos crear un objeto del tipo

    Process Instancia_p = **Runtime.getRuntime.exec**(“<comando”>)

    InputStream entrada = Instancia_p.**getInputStream() //datos binarios**

    (Ya tenemos conexión de ambos procesos: Pactual Instania_p)

PROCESS

- ¿Por qué hemos hablado de flujos de I/O para procesos?
  - Porque una vez ejecutado el método exec(), hay que recuperar el flujo del proceso externo, por ejemplo para ver resultados del mismo en pantalla.
  - Es como si ese proceso externo al ejecutarse, los datos que genera tuviéramos que recuperarlos para poder visualizarlos.
  - Para ello, utilizamos el método getInputStream() de una instancia de Process, para recuperar su flujo, pasarlo a un buffer y mandarlo a pantalla. Sería algo como:
    - 1.- Recuperamos un objeto **Runtime** de la instancia de nuestro proceso Java que estamos ejecutando. Llamando al método **getRuntime()**
    - 2.- Con el **Runtime**, llamamos a su método **exec()** y devuelve un objeto de la clase **Process (Nuevo proceso)**.
    - 3.- Con el objeto **Process**, recuperamos su stream llamando a **getInputStream()**
    - 4.- Con el flujo de entrada del proceso externo, ya podemos convertir sus datos de tipo byte a caracteres por medio de **InputStreamReader**.
    - 5.- Pasamos esos caracteres a memoria (buffer) para tratarlos de una manera más eficiente.
    - 6.- Leemos desde el buffer de memoria e imprimimos en pantalla.

PROCESS

- Ejemplo:

```
// Se lanza el ejecutable.
Process p=Runtime.getRuntime().exec ("cmd /c dir");

// Se obtiene el stream de salida del programa
InputStream is = p.getInputStream();

/* Se prepara un bufferedReader para poder leer la salida más comodamente. */
BufferedReader br = new BufferedReader (new InputStreamReader (is));

// Se lee la primera linea
String aux = br.readLine();

// Mientras se haya leido alguna linea
while (aux!=null)
{
    // Se escribe la linea en pantalla
    System.out.println (aux);

    // y se lee la siguiente.
    aux = br.readLine();
}
```

```
public class PruebaRuntime {

    /** Creates a new instance of PruebaRuntime */
    public PruebaRuntime()
    {
        try
        {
            // Se lanza el ejecutable.
            Process p=Runtime.getRuntime().exec ("cmd /c dir");

            // Se obtiene el stream de salida del programa
            InputStream is = p.getInputStream();

            /* Se prepara un bufferedReader para poder leer la salida más comodamente. */
            BufferedReader br = new BufferedReader (new InputStreamReader (is));

            // Se lee la primera linea
            String aux = br.readLine();

            // Mientras se haya leido alguna linea
            while (aux!=null)
            {
                // Se escribe la linea en pantalla
                System.out.println (aux);

                // y se lee la siguiente.
                aux = br.readLine();
            }
        }
        catch (Exception e)
        {
            // Excepciones si hay algún problema al arrancar el ejecutable o al leer su salida.*/
            e.printStackTrace();
        }
    }

    /**
     * Crea la clase principal que ejecuta el comando dir y escribe en pantalla
     * lo que devuelve dicho comando.
     *
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        new PruebaRuntime();
    }
}
```

PROCESS

- Otro Ejemplo:
- También se puede llamar a exec, pasandole un array de String.

  String cad [] = {“ping”,”8.8.8.8”};

  Procesess = runTime.exec(cad);

- **waitFor**().- Espera a que el subproceso finalize.
- **destroy**().- Finaliza el proceso en cuestión
- **exitValue**().- Devuelve un valor de salida cuando el subproceso finaliza.

```java
import java.io.*;

class JavaPing{

    public static void main(String[ ] args) {

        Runtime runTime= Runtime.getRuntime();
        Process process=null;

        try {
            process = runTime.exec("ping " + args[0]);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(process.getInputStream()));

            for (int i=0; i< 10; i++)
                System.out.println("Saludo desde programacion 2021" + in.readLine());

        } catch (IOException e) {
            System.out.println("No pudimos correr el ping desde nuestra clase");
            System.exit(-1);
        }

        if (process!=null)
            process.destroy();

        try {
            process.waitFor();
        } catch( InterruptedException e ) {
                System.out.println("No pudimos esperar porque termino");
                System.exit(-1);
        }

        System.out.println("Estado de termino: "+process.exitValue());
        System.exit(0);
    }
}
```

PROCESS

- Qué hace:
- Lanza un subproceso ping.

Proceso padre:

- Crea el subproceso ping y captura de su Stream 10 ping.
- Debe destruir el ping con destroy()
- Espera a que muera con waitFor() porque puede ser que haya habido problemas.
- Lo normal es que al destruir al hijo, éste acabe antes de que llegue a hacer el waitFor().
- Recoge el valor de retorno del process e informa.

proceso2.java (~/Documents/trabajo_instituto/instituto_21_22/1DAM/programaciónSanti/ejer

```java
import java.io.*;

class proceso2{

    public static void main(String[ ] args) {

        Runtime runTime= Runtime.getRuntime();
        Process process=null;

        try {
            process = runTime.exec("ping " + args[0]);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(process.getInputStream()));

            for (int i=0; i< 10; i++)
                System.out.println("Saludo desde PSP 21/22" + in.readLine());

        } catch (IOException e) {
            System.out.println("No pudimos correr el ping desde nuestra clase");
            System.exit(-1);
        }

        if (process!=null){
            process.destroy();
            System.out.println("Me he cargado el ping....");
        }

        try {
            System.out.println("Ahora esperaré a que acabe mi proceso ping");
            process.waitFor();
            System.out.println("Ya no existe mi proceso ping");
        } catch( InterruptedException e ) {
                System.out.println("No pudimos esperar porque termino");
                System.exit(-1);
        }

        System.out.println("Estado de termino: "+process.exitValue());
        System.exit(0);
    }
}
```

PROCESS

```
~/Documents/trabajo_instituto/instituto_21_22/1DAM/programaciónSanti/ejemplos> java proceso2 8.8.8.8
Saludo desde PSP 21/22PING 8.8.8.8 (8.8.8.8): 56 data bytes
Saludo desde PSP 21/22Request timeout for icmp_seq 0
Saludo desde PSP 21/22Request timeout for icmp_seq 1
Saludo desde PSP 21/22Request timeout for icmp_seq 2
Saludo desde PSP 21/22Request timeout for icmp_seq 3
Saludo desde PSP 21/22Request timeout for icmp_seq 4
Saludo desde PSP 21/2264 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=1046.842 ms
Saludo desde PSP 21/2264 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=2048.454 ms
Saludo desde PSP 21/2264 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=46.224 ms
Saludo desde PSP 21/2264 bytes from 8.8.8.8: icmp_seq=6 ttl=118 time=20.472 ms
Me he cargado el ping....
Ahora esperar[redacted] a que acabe mi proceso ping
Ya no existe mi proceso ping
Estado de termino: 143
```

PROCESS

- Otro ejemplo más sencillo:
- ¿Qué hace?
- ¿Qué sucede si muriera antes el proceso padre?

```java
public class ProcessDemo {

   public static void main(String[] args) {
      try {
         // create a new process
         System.out.println("Creating Process...");
         Process p = Runtime.getRuntime().exec("notepad.exe");

         // cause this process to stop until process p is terminated
         p.waitFor();

         // when you manually close notepad.exe program will continue here
         System.out.println("Waiting over.");
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

PROCESS

- Otro ejemplo más sencillo:
- ¿Qué hace?
- ¿Qué es un Thread.sleep(10000)?

```java
public class ProcessDemo {

   public static void main(String[] args) {
      try {
         // create a new process
         System.out.println("Creating Process...");
         Process p = Runtime.getRuntime().exec("notepad.exe");

         // wait 10 seconds
         System.out.println("Waiting...");
         Thread.sleep(10000);

         // kill the process
         p.destroy();
         System.out.println("Process destroyed.");

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

PROCESSBUILDER

- Con **ProcessBuilder**, se puede configurar previamente su ejecución.
- Se utiliza para crear procesos de sistemas operativos. Á partir de la versión 1.5 del JDK.

  ProcessBuilder(List<String> parámetros)

  ProcessBuilder(String1, String2, ….,Stringn);
- El método **start**() crea una instancia de Process con los atributos que le pasamos en el **start**().
- A los atributos pasados al **ProcessBuilder**, al menos hay que pasarle el nombre de la aplicación o comando.
- Con una misma instancia de **ProcessBuilder**, se pueden crear varios subprocesos invocando al método **start**(), por tanto se pueden replicar procesos con los mismos parámetros.
- Se dice que es asíncrono, porque cada subproceso puede

```
ProcessBuilder proceso= new ProcessBuilder("ls");
Process proceso1 = proceso.start();
Process proceso2 = proceso.start();
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/ProcessBuilder.html

PROCESSBUILDER

- A diferencia entre ProcessBuilder y el método exec de la Runtime, es que los parámetros deben ser String o pasando un array de String.
- Ejemplo:
  - ProcessBuilder pb = new ProcessBuilder(new String[] {“find”, “/”, “-name”, “\”*\””});
    o
  - ProcessBuilder pb = new ProcessBuilder(“find”, “/”, “-name”, “\”*\””);
- Otra de las grandes ventajas, es que podemos heredar las entradas/salidas entre el proceso padre y el nuevo. De esta manera, no es necesario crear flujos de entrada/salida. Esto se consigue con el método **inheritIO()**.

CLASE PROCESSBUILDER

- Ejemplo:

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;


public class proceso1{


        public void ejecutarProceso(String file){
            ProcessBuilder pb;
            try{
                    pb = new ProcessBuilder().command("cat", file);
                    Process proceso = pb.start();
                    System.out.println("Desde PSP 21/22, Soy el proceso padre y crearé el cat");
                    BufferedReader br = new BufferedReader(new InputStreamReader(proceso.getInputStream()));
                    String linea;
                    while ( (linea=br.readLine()) != null){
                        System.out.println(linea);
                    }
            } catch(Exception e){
                    e.printStackTrace();
            }
        }



        public static void main(String[] args){
            //System.out.println("Ejemplo de proceso1 con cat");
            String fichero="/tmp/msg";
            proceso1 p = new proceso1();
            p.ejecutarProceso(fichero);
            System.out.println("Finalizado");

        }
}
```

```
~/Documents/trabajo_instituto/instituto_21_22/1DAM/programaciónSanti/ejemplos> java proceso1
Desde PSP 21/22, Soy el proceso padre y crear el cat
Desde un fichero
Finalizado
```

PROCESSBUILDER

- Métodos importantes de **ProcessBuilder**:
  - long **pid()**.- Delvuelve el identificador del proceso actual.
  - boolean **isAlive()**.- Comprueba si el proceso está vivo.
  - **waitFor()** .- Hace que el proceso actual, espere hasta que el nuevo no termine. Devuelve cero si ha finalizado sin errores, mientras que distinto de 0, depende del timpo de error. Por ejemplo:
    - 1 //entendemos un error
    - 2 //entendemos una interrupción
  - **waitFor(long, TimeUnit)**.- Hace que el proceso actual, espere a que el nuevo proceso finalice por una unidad máxima de tiempo.
    - long especifica el tiempo
    - TimeUnit es una clase que inica la unidad de tiempo. MILLISECONDS, etc.
  - int **exitValue()**. Devuelve el valor de salida de ejecución de un proceso. Podemos hacer que antes de acabar, devuelva un valor. Por ejemplo:
    - System.exit(codigo)

```java
import java.util.Arrays;
import java.util.concurrent.TimeUnit;
import java.io.IOException;

public class EjecutaComandoTimeout{

        public static int MAX_TIEMPO_EJECUCION = 500;

        public static void main(String argv[]){
                String comando = new String({"find", "/", "-name", "\"*\""})
                ProcessBuilder pb = new ProcessBuilder(comando);

                System.out.printn("Como proceso actual, voy a ejecutar un comando find por un tiempo mediante otro proceso\n");
                pb.inheritIO();  //el proceso actual y el nuevo, utilizan las mismas E/S
                pb.redirectErrorStream(true);  //para que no salgan las salidas y errores mezclados
                try{
                        Process p = pb.start();  //lanzamos el nuevo proceso
                        if (!p.waitFor(MAX_TIEMPO_EJECUCION, TimeInit.MILLISECONDS)){
                                p.destroy();  //nos cargamos al proceso hijo.
                                /*
                                        waitFor devuelve true si el nuevo proceso
                                        ha terminado por si mismo antes del
                                        tiempo indicado
                                */
                                System.out.println ("El proceso lanzado no ha finalizado a tiempo su ejecución\n");
                        }

                }
                catch(IOException e){
                        System.out.println ("Error al intentar lanzar un nuevo proceso. Ponemos información detallada\n");
                        e.printStackTrace();
                        System.exit(1); //error en el proceso
                }
                catch(InterruptedException ex){
                        System.out.println("El proceso ha sido interrumpido mediante interrupción\n");
                        System.exit(2);  //proceso interrumpido
                }
        }
}
```

Redireccionamiento

- **redirectInput(new File f):** Redirigimos la entrada de un fichero al comando. Ej: comando < f
- **redirectOutput(new File f):** Redirigimos la salida del comando a un fichero. Ej: comando > f
- **redirectOutput(Redirect.appendTo(new File f)):** Redirigimos la salida del comando añadiendolo a un fichero. Ej: comando >> f
- **redirectError(new File f):** Redirigimos la salida de error del comando a un fichero. Ej: comando 2> f
- **redirectError(Redirect.appendTo(new File f)**: Redirigimos la salida de error del comando añadiendolo a un fichero. Ej: comando 2>>f
- **redirectError(Redirect.DISCARD)**: Deshechamos el error que produce el comando. Ej: comando 2> /dev/null

Redireccionamiento

```java
public static int lanzaProcesoSalidaFichero(String [] comando) throws IOException, InterruptedException{
    String nombreFichero = "salida.out";
    ProcessBuilder pb = new ProcessBuilder();
    pb.command(comando);
    File fich = new File("/tmp/" + nombreFichero);
    pb.redirectOutput(fich);
    Process p = pb.start();
    int exitCode = p.waitFor();
    return exitCode;
}
```

Redireccionamiento

Es muy conveniente, poder redireccionar la salida del nuevo proceso, a la del padre, para ello tenemos:

- **ProcessBuilder pb new ProcessBuilder(comando);**
- **pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);**
  - Es la forma que tenemos de redirigir la salida del nuevo proceso, al mismo que tiene el padre actual. La idea es compartir el mismo flujo de salida:

    flujo entrada → padre → **flujo salida** ← nuevo proceso

Redireccionamiento

- **Métodos para obtener stream asociados a la entrada/salida estándar.**
  - InputStream **getInputStream**.- Devuelve un stream de entrada conectado a la salida del nuevo proceso. Lo utilizaremos para obtener el resultado de ejecución del nuevo proceso hacia el padre.
    - Podemos utilizar la **Clase Scanner** para leer la salida de un proceso.

```
Scanner in;
in = new Scanner (p.getInputStream());
```

  - OutputStream **getOutputStream**.- Devuelve un stream de salida conectado a la entrada del nuevo proceso. Nosotros lo utilizaremos para que el nuevo pueda aceptar parámetros del padre.
    - Podemos utilizar la Clase **PrintWriter**.

```
Process p = instanciaProcessBuilder.start();
PrintWriter out = new PrintWriter(p.getOutputStream())
System.out.println(linea);
```

  - InputStream **getErrorStream**.- Devuelve un stream de entrada conectado a la salida de error del nuevo proceso.

PROCESSBUILDER

- El siguiente ejemplo, lo que hace es lanzar un comando "**nslookup**", donde la salida se redirigirá a la del padre, para que podamos ver qué hace en todo momento. Por tanto:
  - **Creamo**s el nuevo proceso con el nslookup
  - **Redirigimos la salida** del nuevo proceso que ejecuta el nslookup, a la salida del padre.
  - Desde el padre, **redirigimos la entrada** estándar hacia un BufferedReader para leer línea a línea en formato carácter según UTF-8. Lo que leeremos es un dominio desde teclado dentro de un bucle while.
  - Para cada dominio leído desde el padre, hay que mandárselo al hijo para que complete el nslookup. Para ello, debemos obtener el **stream de salida** conectado con la entrada estándar del nuevo proceso.
  - Escribimos en dicho **stream** el dominio en UTF-8 y el padre espera con un waitFor a que el hijo complete su ejecución.

```java
import ...

public class outputStream
{
    public static void main(String[] args) {
        ProcessBuilder pb = new ProcessBuilder( ...command: "nslookup");
        pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);
        try {
            InputStreamReader isstdin = new InputStreamReader(System.in,  charsetName: "UTF-8");
            BufferedReader brstdin = new BufferedReader(isstdin);

            String linea;
            System.out.println("Introducir nombre del dominio");
            while ((linea = brstdin.readLine())!= null && linea.length() != 0){
                Process p = pb.start();
                try {
                    OutputStream osp = p.getOutputStream();
                    OutputStreamWriter oswp = new OutputStreamWriter(osp,  charsetName: "UTF-8");
                    oswp.write(linea);
                } catch (IOException e){
                    e.printStackTrace();
                }
                try{
                    p.waitFor();
                } catch ( InterruptedException ex)
                {
                    ex.printStackTrace();
                }
            }
            System.out.println("Introduce nombre de dominio: ");
        } catch (IOException e) {
            System.out.println("Error de E/S");
            e.printStackTrace();
        }

    }
}
```

PROCESSBUILDER

- El siguiente ejemplo es el mismo pero adaptado a la **clase Scanner y PrinterWriter**

```
import java.io.*;
import java.util.Scanner;

public class nslookupScanner {

    public static void main(String[] args) throws IOException {
        Scanner in;
        String linea;
        ProcessBuilder pb = new ProcessBuilder("nslookup");
        pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);

        in = new Scanner(System.in);

        System.out.println("Introducir nombre de dominio");

        while (((linea = in.nextLine()) != null) && (linea.length() !=0)) {
            Process p = pb.start();
            try (PrintWriter out = new PrintWriter(p.getOutputStream())){
                System.out.println(linea);
                out.println(linea);
                out.flush();
            }
            try {
                p.waitFor();
            } catch (InterruptedException e) {}
            System.out.println("Introducir nombre de dominio");
        }
    }

}
```

PREGUNTAS???