1.8 Método getOutputStream

getOutputStream()

Recordamos que es parte de la clase *Process* y se utiliza para obtener un flujo de salida (*OutputStream*) del proceso que has iniciado.

Este flujo de salida permite enviar datos a la entrada estándar (stdin) del proceso, lo que es útil cuando deseas interactuar con el proceso enviándole comandos o datos.

```
OutputStream outputStream = process.getOutputStream();
```

1.8 Método getOutputStream

getOutputStream()

El método *write()* envía los bytes al *stream*, el método *getBytes()* codifica la cadena en una secuencia de bytes que utilizan juego de caracteres por defecto de la plataforma

```
OutputStream out = prc.getOutputStream();
out.write("some text".getBytes());
out.close();
```

1.8 Método getOutputStream

EJEMPLO

Ejecuta el comando *DATE* en la consola de Windows y le pasas una fecha como datos de entrada.

1.8 Método getOutputStream

```java
public static void main(String[] args) throws IOException{
    // TODO Auto-generated method stub

    //Creamos el proceso
    ProcessBuilder pb = new ProcessBuilder ("CMD");
    //Lanzamos el proceso
    Process p = pb.start();

    //Enviamos una entrada al comando DATE
    try (OutputStream os = p.getOutputStream()){
        os.write("date 01-12-22/n".getBytes());
        os.flush(); //Enviar datos inmediatamente
    }

    // Leemos la salida del proceso
    try (InputStream is = p.getInputStream();
         BufferedReader br = new BufferedReader(new InputStreamReader(is))) {
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println("Salida --> " + line);
        }
    }

    // Leemos cualquier error
    try (InputStream is = p.getErrorStream();
         BufferedReader br = new BufferedReader(new InputStreamReader(is))) {
        String liner;
        while ((liner = br.readLine()) != null) {
            System.out.println("Error --> " + liner);
        }
    }

    // Esperamos a que el proceso termine
    try {
        int exitVal = p.waitFor();
        System.out.println("Valor de Salida " + exitVal);
    } catch (InterruptedException ie) {
        ie.printStackTrace();
    }


    }
}
```

1.9 Métodos environment y command

environment()

Es un método de la clase ***ProcessBuilder*** que se utiliza para acceder y modificar las variables de entorno del proceso que se está creando.
Este método devuelve un mapa (***Map<String, String>***) que representa las variables de entorno que se pueden leer y modificar

```
ProcessBuilder test = new ProcessBuilder();
Map entorno = test.environment();
System.out.println("Variables de entorno:");
System.out.println(entorno);
```

1.9 Métodos environment y command

command ()

Es un método de la clase ***ProcessBuilder*** que se utiliza para acceder o especificar el comando y sus argumentos que se desean ejecutar en un nuevo proceso. Este método es fundamental para definir qué acción realizará el proceso.

Este método devuelve los argumentos (***tipo List***) del proceso definido en el objeto ***ProcessBuilder*** (nombre del proceso y sus argumentos)

```
// devuelve el nombre del proceso y sus argumentos
List l = test.command();
Iterator iter = l.iterator();
System.out.println("\nArgumentos del comando:");
while (iter.hasNext())
    System.out.println(iter.next());
```

1.9 Redireccionamiento de entrada y salida

Hay ocasiones en que queremos redirigir la entrada, salida o error a un fichero, para ello, se utilizan los métodos de la clase ***ProcessBuilder***: ***redirectInput()***, ***redirectOutput()*** y ***redirectError()***

Recordémoslos.

1.9 Redireccionamiento de entrada y salida

redirectOutput ()

Es un método de la clase *ProcessBuilder* que se utiliza para redirigir la salida estándar (stdout) de un proceso a un archivo o a otro flujo de salida.

Esto es útil cuando deseas guardar la salida de un comando o proceso en un archivo en lugar de que se imprima en la consola.

```
// Redirigir la salida estándar a un archivo
processBuilder.redirectOutput(new File("output.txt"));
```

1.9 Redireccionamiento de entrada y salida

redirectInput ()

Se utiliza en el contexto de la clase *ProcessBuilder* para redirigir la entrada estándar (stdin) de un proceso a un archivo o a otro flujo de entrada.

Esto es útil cuando deseas que un proceso lea datos de un archivo en lugar de esperar la entrada del usuario desde la consola.

```
// Redirigir la entrada estándar desde un archivo
processBuilder.redirectInput(new File("input.txt"));
```

1.9 Redireccionamiento de entrada y salida

redirectError ()

Es parte de la clase *ProcessBuilder* y se utiliza para redirigir la salida de error estándar (stderr) de un proceso a un archivo o a otro flujo de salida.

Esto es útil para capturar mensajes de error de un proceso sin que se impriman en la consola, permitiendo su almacenamiento o análisis posterior

```
// Redirigir la salida de error a un archivo
processBuilder.redirectError(new File("error.log"));
```

1.9 Redireccionamiento de entrada y salida

Importante: Estas sentencias deben ir antes de lanzar el proceso.

Por ejemplo, para el método ***redirectOutput()***, la sentencia sería:

```
File fileName = new File("fichero.txt");
pb.redirectOutput(fileName);
```

1.9 Redireccionamiento de entrada y salida

EJEMPLO

Ejecuta el comando DIR y redirige la salida estándar y la de error a los ficheros ***salida.txt*** y ***error.txt*** respectivamente

1.9 Redireccionamiento de entrada y salida

```java
public class Ejemplo07 {
  public static void main(String args[]) throws IOException {
    ProcessBuilder pb = new ProcessBuilder("CMD","/C" ,"DIR");

    File fOut = new File("salida.txt");
    File fErr = new File("error.txt");

    pb.redirectOutput(fOut);
    pb.redirectError(fErr);
    pb.start();
  }
}// Ejemplo7
```

La sentencia new ***File("salida.txt")*** crea una instancia al fichero, pero no crea el fichero.

Si queremos crear el fichero deberíamos llamar al método ***createNewFile()*** en la instancia de ***File***:

File Fout = new File("salida.txt");
Fout.createNewFile();

1.9 Redireccionamiento de entrada y salida

EJEMPLO

Vamos a ejecutar un fichero BAT con varios comandos del sistema operativo e indicar al proceso que la entrada de datos está en dicho fichero.

El proceso que lanzamos será el ***CMD*** (sin parámetros)

El ***fichero.bat*** tendrá (por ejemplo) los siguientes comandos:

```
1 MKDIR NUEVO
2 CD NUEVO
3 ECHO CREO FICHERO > Mifichero.txt
4 DIR
5 DIRR
6 ECHO FIN COMANDOS
```

1.9 Redireccionamiento de entrada y salida

```java
public class Ejemplo08 {
  public static void main(String args[]) throws IOException {
    ProcessBuilder pb = new ProcessBuilder("CMD");

    File fBat = new File("fichero.bat");
    File fOut = new File("salida.txt");
    File fErr = new File("error.txt");

    pb.redirectInput(fBat);
    pb.redirectOutput(fOut);
    pb.redirectError(fErr);
    pb.start();

  }
}// Ejemplo8
```

1.9 Redireccionamiento de entrada y salida

Para llevar a cabo el redireccionamiento, tanto de entrada como de salida del proceso que se ejecuta, podemos usar la clase ***ProcessBuilder.Redirect***. El redireccionamiento puede ser uno de los siguientes:

- ***Redirect.INHERIT***, indica que la fuente de entrada y salida del proceso será la misma que la del proceso actual.

```
 ProcessBuilder pb = new ProcessBuilder("yourcommand");
pb.redirectOutput(Redirect.INHERIT);
pb.redirectError(Redirect.INHERIT);
Process p = pb.start();
```

1.9 Redireccionamiento de entrada y salida

- ***Redirect.from(File)***, indica redirección para leer de un fichero, la entrada al proceso se encuentra en el objeto File.
- ***Redirect.to(File)***, indica redirección para escribir en un fichero, el proceso escribirá en el objeto File especificado.

```
builder.redirectError(ProcessBuilder.Redirect.to(tempFile));
builder.redirectOutput(ProcessBuilder.Redirect.to(tempFile));
final Process process = builder.start();
process.waitFor();
process.exitValue();
```

1.9 Redireccionamiento de entrada y salida

- ***Redirect.appendTo(File)***, indica redirección para añadir a un fichero, la salida del proceso se añadirá al objeto File especificado

```
pb.redirectOutput(Redirect.appendTo(TESTS_LOG));
pb.redirectError(Redirect.appendTo(TESTS_LOG));
Process p = pb.start();
```

1.9 Redireccionamiento de entrada y salida

EJEMPLO

Ejecuta el proceso ***DIR*** y redirige la salida a la salida estándar del proceso que lo ejecuta (es decir, a consola)

1.9 Redireccionamiento de entrada y salida

```java
package com;
/*Redirect.INHERIT, indica que la fuente de entrada y salida del proceso
 * será la misma que la del proceso actual.*/
import java.io.IOException;

public class Ejemplo09 {
    public static void main(String args[]) throws IOException {
        ProcessBuilder pb = new ProcessBuilder("CMD", "/C", "DIR");
        //la salida a consola
        pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);
        Process p = pb.start();

    }
}// Ejemplo9
```