

1. .- Lanzar programa

- a. Crea un programa que lance un proceso con ProcessBuilder que se encargue de ejecutar el programa donde se muestran los procesos activos de Ubuntu.

El programa lanza el administrador de tareas de linux.

```
//A) Crear un proceso con ProcessBuilder
ProcessBuilder proceso = new ProcessBuilder(command: "gnome-system-monitor");
```

- b. Muestra información del proceso.

```
//B) Muestra informacion del proceso
System.out.println("Informacion del proceso " + p.info());
```

```
Informacion del proceso [user: Optional[sergio], cmd: /usr/bin/gnome-system-monitor, s
```

- c. Muestra si el proceso está activo.

```
//C) Muestra si el proceso esta Activo
System.out.println("¿Esta vivo el proceso? " + p.isAlive());
```

```
¿Esta vivo el proceso? true
```

- d. Muestra el PID del proceso.

```
//D) Muestra el PID del proceso.
System.out.println("PID del proceos: " + p.pid());
```

```
PID del proceos: 7533
```

- e. Muestra el código de retorno del proceso.

En este caso me saltaba un error ya que seguía ejecutando así que decidí usar un try catch para que el programa siguiera ejecutando y abajo está el mensaje de error.

```
try {
    //E) Muestra el codigo de retorno del proceso
    System.out.println("Codigo del retorno: " + p.exitValue());
} catch (IllegalThreadStateException ex){
    System.out.println(x: ex.getMessage());
}
```

```
process hasn't exited
```

- f. Espera 5 segundos y mata el proceso.

Podía usar el waitfor(Pero preferí usar el Thread sleep).

```
//F) Espera 5 segundos y mata al proceso
Thread.sleep(1:5000);
p.destroy();
```

- g. Muestra de nuevo si el proceso está activo.

```
¿Esta vivo el proceso? false
```

- h. Muestra el código de retorno del proceso.

```
Codigo del retorno: 0
```

Abajo adjunto la ejecución completa del ejercicio y su código.

```
//A) Crear un proceso con ProcessBuilder
ProcessBuilder proceso = new ProcessBuilder(command: "gnome-system-monitor");

try {
    Process p = proceso.start();

    //B) Muestra informacion del proceso
    System.out.println("Informacion del proceso " + p.info());

    //C) Muestra si el proceso esta Activo
    System.out.println("¿Esta vivo el proceso? " + p.isAlive());

    //D) Muestra el PID del proceso.
    System.out.println("PID del proceos: " + p.pid());

    try {
        //E) Muestra el codigo de retorno del proceso
        System.out.println("Codigo del retorno: " + p.exitValue());
    } catch (IllegalThreadStateException ex) {}
    System.out.println(x: ex.getMessage());

    //F) Espera 5 segundos y mata al proceso
    Thread.sleep(5000);
    p.destroy();

    //G) Muestra de nuevo si el proceso esta activo.
    System.out.println("¿Esta vivo el proceso? " + p.isAlive());

    //H) Muestra el codigo de retorno del proceso
    System.out.println("Codigo del retorno: " + p.exitValue());

} catch (IOException | InterruptedException ex) {
    System.out.println(x: ex.getMessage());
}
}
```

ut - Ejercicio1 (run) x

```
run:
Informacion del proceso [user: Optional[sergio], cmd: /usr/bin/gnome-system-monitor, ...]
¿Esta vivo el proceso? true
PID del proceos: 7533
process hasn't exited
¿Esta vivo el proceso? false
Codigo del retorno: 0
BUILD SUCCESSFUL (total time: 5 seconds)
```

2. .-Lanzar comando en Ubuntu 22.04.

- a. Crear un programa que lance un proceso con ProcessBuilder que se encargue de ejecutar un comando sin argumentos.

Aquí se puede ver como dejo los 3 argumentos preparados en 3 String distintos y creo con el ProcessBuilder un proceso sin argumentos y le mando por argumento el comando ("ls").

```
//A) Crea un programa que lance un proceso con ProcessBuilder que se encargue de
// ejecutar un comando sin argumentos.

String comando = "ls";

//B) Crea un programa que lance un proceso con ProcessBuilder que se
// encargue de ejecutar un comando con un argumento.
String argumento = "-R";

//C) Crea un programa que lance un proceso con ProcessBuilder que se encargue
// de ejecutar un comando con dos argumentos

String argumento2 = "-t";

ProcessBuilder procesoSinArgumento = new ProcessBuilder(command:comando);
procesoSinArgumento.inheritIO();
```

Creó el proceso y luego lo inició.

```
Process p;
try {
    //A)
    System.out.println(x: "COMANDO SIN ARGUMENTOS");
    p = procesoSinArgumento.start();
    System.out.println();
}
```

```
COMANDO SIN ARGUMENTOS
build
build.xml
manifest.mf
nbproject
src
```

- b. b) Crear un programa que lance un proceso con ProcessBuilder que se encargue de ejecutar un comando con un argumento.

Aquí creó otro ProcessBuilder y le asignó por argumento el comando y su argumento y hago un waitFor() para que espere (no se porque, si no lo pongo no me muestra la salida del comando con argumento, pero luego si me muestra la salida del ejercicio C).

```
ProcessBuilder procesoConArgumentos = new ProcessBuilder(command:comando, command:argumento);
procesoConArgumentos.inheritIO();
```

```
//B)
System.out.println(x: "COMANDO CON UN ARGUMENTO");
p = procesoConArgumentos.start();
p.waitFor();
System.out.println();
```

```

COMANDO CON UN ARGUMENTO
.:
build
build.xml
manifest.mf
nbproject
src
test

./build:
classes

./build/classes:
ejercicio2

./build/classes/ejercicio2:
Ejercicio2.class

./nbproject:
build-impl.xml
genfiles.properties
private
project.properties
project.xml

```

- c. c) Crear un programa que lance un proceso con ProcessBuilder que se encargue de ejecutar un comando con dos argumentos.

```

ProcessBuilder procesoConVariosArgumentos = new ProcessBuilder(command:comando, command:argumento, command:argumento2);
procesoConVariosArgumentos.inheritIO();

```

```

//C)
System.out.println(x: "COMANDO CON VARIOS ARGUMENTOS");
p = procesoConVariosArgumentos.start();

```

COMANDO CON VARIOS ARGUMENTOS

```
.:  
test  
build  
build.xml  
manifest.mf  
nbproject  
src  
  
./test:  
  
./build:  
classes  
  
./build/classes:  
ejercicio2  
  
./build/classes/ejercicio2:  
Ejercicio2.class  
  
./nbproject:  
build-impl.xml  
genfiles.properties  
private  
project.properties  
project.xml
```

3. .- Prueba la ejecución de ProcessBuilder en las siguientes situaciones y comenta los resultados obtenidos.

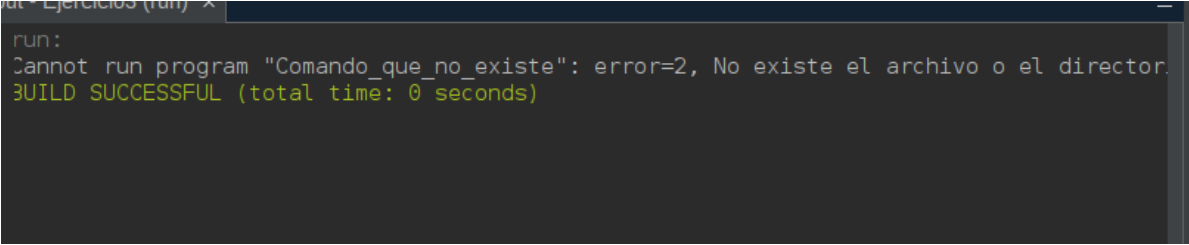
a. Ejecución de programa que no existe.

```
//A) Ejecucion de programa que no existe.  
ProcessBuilder programaNoExiste = new ProcessBuilder(command: "Comando_que_no_  
programaNoExiste.inheritIO());
```

```
Process p;  
  
try {  
    p = programaNoExiste.start();  
    p.waitFor();  
  
    p = comandoFallo.start();  
    p.waitFor();  
} catch (IOException | InterruptedException ex) {  
    System.out.println(x: ex.getMessage());  
}
```

- b. Ejecución de un proceso que termine con un código de error, por ejemplo, la ejecución de "ls /et"

```
ProcessBuilder comandoFallo = new ProcessBuilder(command: "ls /et");  
comandoFallo.inheritIO();
```



A screenshot of a terminal window titled "Ejercicios (run)". The terminal shows the output of a program execution. The first line is "run:". The second line is an error message: "Cannot run program \"Comando_que_no_existe\": error=2, No existe el archivo o el directorio." The third line is "BUILD SUCCESSFUL (total time: 0 seconds)".

```
run:  
Cannot run program "Comando_que_no_existe": error=2, No existe el archivo o el directorio.  
BUILD SUCCESSFUL (total time: 0 seconds)
```