Análisis y Corrección de Errores en el Código Java (‘ProcessAuto.java‘)

12 de octubre de 2025

1. Introducción

Este documento detalla los errores encontrados en el código original de la clase `ProcessAuto.java` y explica por qué los métodos `redireccionSalida` y `parametrosDinamicos` no funcionaban como se esperaba.

2. Error 1: Redirección de Salida Incorrecta

El problema principal en el método `redireccionSalida` estaba en el orden de las operaciones.

2.1. Código Original

```
public void redireccionSalida(String comando){
    ProcessBuilder pb= new ProcessBuilder(comando);

    try {
        Process p= pb.start(); // 2. El proceso se inicia y termina.
        p.waitFor();
    } catch (IOException | InterruptedException e) {
        throw new RuntimeException(e);
    }
    // 1. La redireccion se configura DEMASIADO TARDE.
    pb.redirectOutput(new File("standar_output.txt"));
    pb.redirectError(new File("error.txt"));
}
```

2.2. Análisis del Error

La configuración de un `ProcessBuilder` (`pb`) debe completarse **antes** de que se inicie el proceso con `pb.start()`. En el código original:

1. El proceso se iniciaba y se esperaba a que terminara (`p.waitFor()`).
2. Solo después de que el proceso había finalizado, se intentaba configurar la redirección de su salida y error.

Para cuando se ejecutaban las líneas `pb.redirectOutput(...)`, el proceso ya había terminado y su salida ya se había gestionado (en este caso, descartado). Por lo tanto, los ficheros `standard_output.txt` y `error.txt` se creaban vacíos o no se actualizaban.

2.3. Solución Aplicada

Se invirtió el orden: primero se configura el `ProcessBuilder` y luego se inicia el proceso.

```
public void redireccionSalida(String comando){
    ProcessBuilder pb = new ProcessBuilder(comando.split("\\s+"));

    // 1. La redireccion se configura ANTES de iniciar.
    pb.redirectOutput(new File("standard_output.txt"));
    pb.redirectError(new File("error.txt"));
```

```
7
8       try {
9           // 2. Ahora el proceso se inicia con la configuracion ya aplicada.
10          Process p = pb.start();
11          p.waitFor();
12      } catch (IOException | InterruptedException e) {
13          e.printStackTrace();
14      }
15  }
```

3. Error 2: Falta de Visibilidad y Lógica Rígida

En el método `parametrosDinamicos`, el problema era doble: el resultado del proceso no era visible y la construcción del comando era demasiado rígida.

3.1. Código Original

```
1  public void parametrosDinamicos(String comando, String paquetes, String host){
2      // 1. El comando siempre incluye "-c" y "paquetes".
3      ProcessBuilder pb= new ProcessBuilder(comando, "-c", paquetes, host);
4      Process p= null;
5      try {
6          p = pb.start();
7          p.waitFor();
8          // 2. La salida del proceso no se muestra en ningun lado.
9      } catch (IOException | InterruptedException e) {
10         throw new RuntimeException(e);
11     }
12 }
```

3.2. Análisis del Error

1. **Falta de visibilidad**: Por defecto, la salida estándar y de error de un proceso hijo no se conecta a la consola del proceso padre. Aunque el comando (ej. `ping`) se ejecutaba correctamente, su salida no se mostraba en la terminal, dando la impresión de que "no hacía nada".

2. **Lógica rígida**: El constructor de `ProcessBuilder` siempre recibía los argumentos `c"` y `paquetes`. Esto causaba problemas si el usuario introducía `"N"`, ya que se construía un comando inválido como `ping -c N google.com`.

3.3. Solución Aplicada

1. Se usó `pb.inheritIO()` para conectar la entrada, salida y error del proceso hijo con los del proceso padre. Esto hace que el resultado del comando aparezca en la consola.

2. Se añadió lógica para construir la lista de comandos dinámicamente, añadiendo los parámetros `c"` y el número de paquetes solo si el usuario no introduce `"N"`.

```
1  public void parametrosDinamicos(String comando, String paquetes, String host){
2      List<String> commandList = new ArrayList<>();
3      commandList.add(comando);
4
5      // 2. Se anaden los parametros solo si es necesario.
6      if (!"N".equalsIgnoreCase(paquetes)) {
7          commandList.add("-c");
8          commandList.add(paquetes);
9      }
10     commandList.add(host);
11
12     ProcessBuilder pb = new ProcessBuilder(commandList);
13
14     // 1. Se hereda la E/S para ver la salida en la consola.
15     pb.inheritIO();
16
17     try {
```

```
        Process p = pb.start();
        p.waitFor();
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }
}
```