Java ProcessBuilder

ProcessBuilder is used to create operating system processes. Its `start()` method creates a new `Process` instance with the following attributes:

- command
- environment
- working directory
- source of input
- destination for standard output and standard error output
- redirectErrorStream

Java ProcessBuilder running program

A program is executed with `command()`. With `waitFor()` we can wait for the process to finish.

com/zetcode/ExecuteProgram.java

```
package com.zetcode;

import java.io.IOException;

public class ExecuteProgram {

    public static void main(String[] args) throws IOException,
InterruptedException {

        var processBuilder = new ProcessBuilder();

        processBuilder.command("notepad.exe");

        var process = processBuilder.start();

        var ret = process.waitFor();

        System.out.printf("Program exited with code: %d", ret);
    }
}
```

The program executes the Windows Notepad application. It returns its exit code.

Java ProcessBuilder command output

The following example executes a command and shows its output.

com/zetcode/ProcessBuilderEx.java

```
package com.zetcode;

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

```
public class ProcessBuilderEx {

    public static void main(String[] args) throws IOException {

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cal", "2019", "-m 2");

        var process = processBuilder.start();

        try (var reader = new BufferedReader(
            new InputStreamReader(process.getInputStream()))) {

            String line;

            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }

        }
    }
}
```

The example runs Linux `cal` command.

```
processBuilder.command("cal", "2019", "-m 2");
```

The `command()` executes the `cal` program. The other parameters are the options of the program. In order to run a command on Windows machine, we could use the following: `processBuilder.command("cmd.exe", "/c", "ping -n 3 google.com")`.

```
var process = processBuilder.start();
```

The process is lauched with `start()`.

```
try (var reader = new BufferedReader(
    new InputStreamReader(process.getInputStream()))) {
```

With the `getInputStream()` method we get the input stream from the standard output of the process.

```
February 2019
Su Mo Tu We Th Fr Sa
                1  2
    3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28
```

This is the ouptput.

Java ProcessBuilder redirect output

With `redirectOutput()`, we can redirect the process builder's standard output destination.

com/zetcode/RedirectOutputEx.java

```
package com.zetcode;
```

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

public class RedirectOutputEx {

    public static void main(String[] args) throws IOException {

        var homeDir = System.getProperty("user.home");

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cmd.exe", "/c", "date /t");

        var fileName = new File(String.format("%s/Documents/tmp/output.txt",
homeDir));

        processBuilder.redirectOutput(fileName);

        var process = processBuilder.start();

        try (var reader = new BufferedReader(
                new InputStreamReader(process.getInputStream()))) {

            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        }
    }
}
```

The program redirects the builder's output to a file. It runs the Windows `date` command.

```
processBuilder.redirectOutput(fileName);
```

We redirect the process builders standard output to a file.

```
try (var reader = new BufferedReader(
    new InputStreamReader(process.getInputStream()))) {

    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
}
```

Now the output goes to the file.

```
$ echo %cd%
C:\Users\Jano\Documents\tmp
$ more output.txt
Thu 02/14/2019
```

The current date was written to the `output.txt` file.

Java ProcessBuilder redirect input and output

The next example redirects both input and output.

src/resources/input.txt

```
sky
blue
steel
morning
coffee
earth
forest
```

This are the contents of the `input.txt` file.

com/zetcode/ProcessBuilderRedirectIOEx.java

```
package com.zetcode;

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

public class ProcessBuilderRedirectIOEx {

    public static void main(String[] args) throws IOException {

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cat")
                .redirectInput(new File("src/resources", "input.txt"))
                .redirectOutput(new File("src/resources/", "output.txt"))
                .start();
    }
}
```

In the program, we redirect input from an `input.txt` file to the `cat` command and redirect the command's output to the `output.txt` file.

Java ProcessBuilder inherit IO

The `inheritIO()` sets the source and destination for subprocess standard I/O to be the same as those of the current Java process.

com/zetcode/ProcessBuilderInheritIOEx.java

```
package com.zetcode;

import java.io.IOException;

public class ProcessBuilderInheritIOEx {

    public static void main(String[] args) throws IOException,
InterruptedException {

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cmd.exe", "/c", "dir");

        var process = processBuilder.inheritIO().start();

        int exitCode = process.waitFor();
        System.out.printf("Program ended with exitCode %d", exitCode);
    }
}
```

By inheriting the IO of the executed command, we can skip the reading step. The program outputs the contents of the project directory and the message showing the exit code.

```
02/14/2019  04:55 PM    <DIR>          .
02/14/2019  04:55 PM    <DIR>          ..
02/19/2019  01:11 PM    <DIR>          .idea
02/14/2019  04:55 PM    <DIR>          out
02/14/2019  04:52 PM                   433 ProcessBuilderInheritIOEx.iml
02/14/2019  04:53 PM    <DIR>          src
               1 File(s)            433 bytes
               5 Dir(s)  157,350,264,832 bytes free
Program ended with exitCode 0
```

We get both the output of the executed command and of our own Java program.

Java ProcessBuilder environment

The `environment()` method returns a string map view of the process builder's environment.

com/zetcode/ProcessBuilderEnvEx.java

```
package com.zetcode;

public class ProcessBuilderEnvEx {

    public static void main(String[] args) {

        var pb = new ProcessBuilder();
        var env = pb.environment();

        env.forEach((s, s2) -> {
            System.out.printf("%s %s %n", s, s2);
        });

        System.out.printf("%s %n", env.get("PATH"));
    }
}
```

The program shows all environment variables.

```
configsetroot C:\WINDOWS\ConfigSetRoot
USERDOMAIN_ROAMINGPROFILE LAPTOP-OBKOFV9J
LOCALAPPDATA C:\Users\Jano\AppData\Local
PROCESSOR_LEVEL 6
USERDOMAIN LAPTOP-OBKOFV9J
LOGONSERVER \\LAPTOP-OBKOFV9J
JAVA_HOME C:\Users\Jano\AppData\Local\Programs\Java\openjdk-11\
SESSIONNAME Console
...
```

This is a sample output on Windows.

In the next program, we define a custom environment variable.

com/zetcode/ProcessBuilderEnvEx2.java

```
package com.zetcode;

import java.io.IOException;

public class ProcessBuilderEnvEx2 {
```

```
    public static void main(String[] args) throws IOException {

        var pb = new ProcessBuilder();
        var env = pb.environment();

        env.put("mode", "development");

        pb.command("cmd.exe", "/c", "echo", "%mode%");

        pb.inheritIO().start();
    }
}
```

The program defines a `mode` variable and outputs it on Windows.

```
pb.command("cmd.exe", "/c", "echo", "%mode%");
```

The `%mode%` is a Windows syntax for environment variables; on Linux we use `$mode`.

Java ProcessBuilder directory

The `directory()` method sets the process builder's working directory.

com/zetcode/ProcessBuilderDirectoryEx.java

```
package com.zetcode;

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

public class ProcessBuilderDirectoryEx {

    public static void main(String[] args) throws IOException {

        var homeDir = System.getProperty("user.home");

        var pb = new ProcessBuilder();

        pb.command("cmd.exe", "/c", "dir");
        pb.directory(new File(homeDir));

        var process = pb.start();

        try (var reader = new BufferedReader(
                new InputStreamReader(process.getInputStream()))) {

            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        }
    }
}
```

The example sets the home directory to be the process builder's current directory. We show the contents of the home directory.

```
var homeDir = System.getProperty("user.home");
```

We get the user's home directory.

```
pb.command("cmd.exe", "/c", "dir");
```

We define a command which executes the `dir` program on Windows.

```
pb.directory(new File(homeDir));
```

We set the process builder's directory.

```
Volume in drive C is Windows
Volume Serial Number is 4415-13BB

Directory of C:\Users\Jano

02/14/2019  11:48 AM    <DIR>          .
02/14/2019  11:48 AM    <DIR>          ..
10/13/2018  08:38 AM    <DIR>          .android
01/31/2019  10:58 PM               281 .bash_history
12/17/2018  03:02 PM    <DIR>          .config
...
```

This is a sample output.

Java ProcessBuilder non-blocking operation

In the following example, we create a process which is asynchronous.

com/zetcode/ProcessBuilderNonBlockingEx.java

```
package com.zetcode;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
import java.util.stream.Collectors;

public class ProcessBuilderNonBlockingEx {

    public static void main(String[] args) throws InterruptedException,
            ExecutionException, TimeoutException, IOException {

        var executor = Executors.newSingleThreadExecutor();

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cmd.exe", "/c", "ping -n 3 google.com");

        try {

            var process = processBuilder.start();

            System.out.println("processing ping command ...");
            var task = new ProcessTask(process.getInputStream());
```

```
            Future<List<String>> future = executor.submit(task);

            // non-blocking, doing other tasks
            System.out.println("doing task1 ...");
            System.out.println("doing task2 ...");

            var results = future.get(5, TimeUnit.SECONDS);

            for (String res : results) {
                System.out.println(res);
            }

        } finally {
            executor.shutdown();
        }
    }

    private static class ProcessTask implements Callable<List<String>> {

        private InputStream inputStream;

        public ProcessTask(InputStream inputStream) {
            this.inputStream = inputStream;
        }

        @Override
        public List<String> call() {
            return new BufferedReader(new InputStreamReader(inputStream))
                    .lines()
                    .collect(Collectors.toList());
        }
    }
}
```

The program creates a process that runs the ping command on the console. It is executed in a separate thread with the help of the `Executors.newSingleThreadExecutor()` method.

```
processing ping command ...
doing task1 ...
doing task2 ...

Pinging google.com [2a00:1450:4001:825::200e] with 32 bytes of data:
Reply from 2a00:1450:4001:825::200e: time=108ms
Reply from 2a00:1450:4001:825::200e: time=111ms
Reply from 2a00:1450:4001:825::200e: time=112ms

Ping statistics for 2a00:1450:4001:825::200e:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 108ms, Maximum = 112ms, Average = 110ms
```

This is the output.

Java ProcessBuilder pipe operation

A pipe is a technique for passing information from one program process to another.

com/zetcode/ProcessBuilderPipeEx.java

```
package com.zetcode;
```

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

public class ProcessBuilderPipeEx {

    public static void main(String[] args) throws IOException {

        var homeDir = System.getProperty("user.home");

        var processBuilder = new ProcessBuilder();

        processBuilder.command("cmd.exe", "/c", "dir | grep [dD]o");

        processBuilder.directory(new File(homeDir));
        processBuilder.inheritIO().start();
    }
}
```

The example sends information from a `dir` commmand to the `grep` command through the pipe (|).

```
Volume in drive C is Windows
11/14/2018  06:57 PM    <DIR>          .dotnet
02/18/2019  10:54 PM    <DIR>          Documents
02/17/2019  01:11 AM    <DIR>          Downloads
```

This is the output.