Working with Processes in Java

Processes

A process is a program running under the control of the Operating System.

- The OS starts the process.
- Usually, the process runs as a separate thread of computation.
- Input to, and output from the process is controlled by the OS.

Example of a Process

For example, when you type

```
javac TableTool.java
```

at the command-line, the operating system starts a process running the java compiler, and passes the command-line argument 'TableTool.java' to the process.

The javac program requires no further input than the relevant '.java' and '.class' files.
No keyboard input ('stdin') is used by the program. Output from the javac program is relayed (by the OS) to standard output ('stdout').
If an error occurs in the running of the javac program, or if it is interrupted (Control-C), error messages are sent (by the OS) to standard error output ('stderr').

Creating Processes in Java

Processes can be created from a Java program using the java.lang.Runtime class.
(Note that, for security reasons, applets can not create processes.)

The Runtime Class

From the API:
Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime method.

Runtime Methods

```
public static Runtime getRuntime()
```

Returns the runtime object associated with the current Java application.
Most of the methods of class Runtime are instance methods and must be invoked with respect to the current runtime object.

`public Process exec(String command) throws IOException`
Executes the specified string command in a separate process.

Creating Processes in Java

So we can run the Unix ls command by:

```
Runtime rt = Runtime.getRuntime();
Process lsProc = rt.exec("ls");
```

But how do we get the results?

The (abstract) Process Class

From the API:
The Runtime.exec methods create a native process and return an instance of a subclass of Process that can be used to control the process and obtain information about it.

Process Methods

- `int exitValue()`
- `InputStream getErrorStream()`
- `InputStream getInputStream()`
- `OutputStream getOutputStream()`

Getting Output from a Process

```
Runtime rt = Runtime.getRuntime();
Process lsProc = rt.exec("ls");
InputStream in = lsProc.getInputStream();
BufferedReader br =
     new BufferedReader(new InputStreamReader(in));
String line;
while ((line = br.readLine()) != null)
  System.out.println(line);
br.close();
```

Process Methods

```
public abstract int waitFor()
                throws InterruptedException
```

causes the current thread to wait, if necessary, until the process represented by this Process object has terminated. This method returns immediately if the subprocess has already terminated. If the subprocess has not yet terminated, the calling thread will be blocked until the subprocess exits.

Waiting for a Process to End

```
lsProc.waitFor();
int ev = lsProc.exitValue();
System.out.println("ls exited with: "+ev);
```

The main thread suspends until ls ends, then gets and prints the exit status.

Not Waiting for a Process to End

```
// lsProc.waitFor();
int ev = lsProc.exitValue();
System.out.println("ls exited with: "+ev);
```

This might throw an exception. . .

The Price of Impatience

```
RunEg.class
RunEg.java
RunEg.java~
Exception in thread "main"
java.lang.IllegalThreadStateException:
 process hasn’t exited
 java.lang.UNIXProcess.exitValue (UNIXProcess.java:220)
 at RunEg.main(RunEg.java:20)
Process RunEg exited abnormally with code 1
```

Sending Data to a Process

The procedure for sending data to a process (its input) is similar.
We use Process.getOutPutStream(), and write to that stream.

```
Process grepProc = rt.exec("grep java");
OutputStream out = grepProc.getOutputStream();
PrintWriter pw =
     new PrintWriter(new OutputStreamWriter(out));
pw.println("I love coffee");
pw.println("I love tea");
pw.println("I love the java");
pw.println("and the java love me");
pw.close();
```

Sending Data to a Process

```
in = grepProc.getInputStream();
br = new BufferedReader
(new InputStreamReader(in));
while ((line = br.readLine()) != null)
  System.out.println(line);
br.close();
```

Sending Data from One Process to Another

As a final example, we'll look at how to take the output from one process, and use that as input to another process. We'll send the output from ls to grep. Our program will effectively be a pipe:

```
ls | grep java
```

Sending Data from One Process to Another

```
lsProc = rt.exec("ls");
in = lsProc.getInputStream();
grepProc = rt.exec("grep java");
out = grepProc.getOutputStream();
int b;

while((b = in.read()) != -1)
  out.write(b);
lsProc.waitFor();
in.close();
out.close();

in = grepProc.getInputStream();
while((b = in.read()) != -1)
  System.out.write(b);
grepProc.waitFor();
in.close();
```