Using Pipelines to Other Java Processes

Version:

This file was updated slightly "Wed Feb 15 17:49:52 2012". The JavaPipe class has also been updated.[1]

Purpose:

To gain familiarity with the instructor supplied Java pipeline class. This class is a necessary part of the CPSC 101 team term project for 2012.

Due Date:

This assignment is due Friday, 2012-03-02 *at the beginning of class*.

Background Reading Assignment:

⇒ Find an online description of UNIX pipes, such as

- http://www.december.com/unix/tutor/pipesfilters.html, *or*
- http://en.wikipedia.org/wiki/Pipeline_%28Unix%29
- What does "`ls | wc`" at the command-line prompt accomplish?

The `JavaPipe` class

JAVA has the ability to execute separate processes and communicate with them via pipes. In order to simplify this process for the 2012 team term project, Dr. Casperson has created a `JavaPipe` class for student use. A Jar file containing the class can be found at `http://web.unbc.ca/~casper/Semesters/2012W/101-project.php`, as can a link to documentation for the class.

This class is custom designed for communicating with another subservient JAVA program. For instance, suppose that you have created a program `Reverse.class` that reads strings from `System.in` and writes the reversed strings to `System.out`:

[1] In particular, the `java.io.PrintWriter` returned by the `.getOutputWriter()` method now automatically does a flush whenever `.println(...)` is used. Thanks to Mr. Kranz for noticing this.

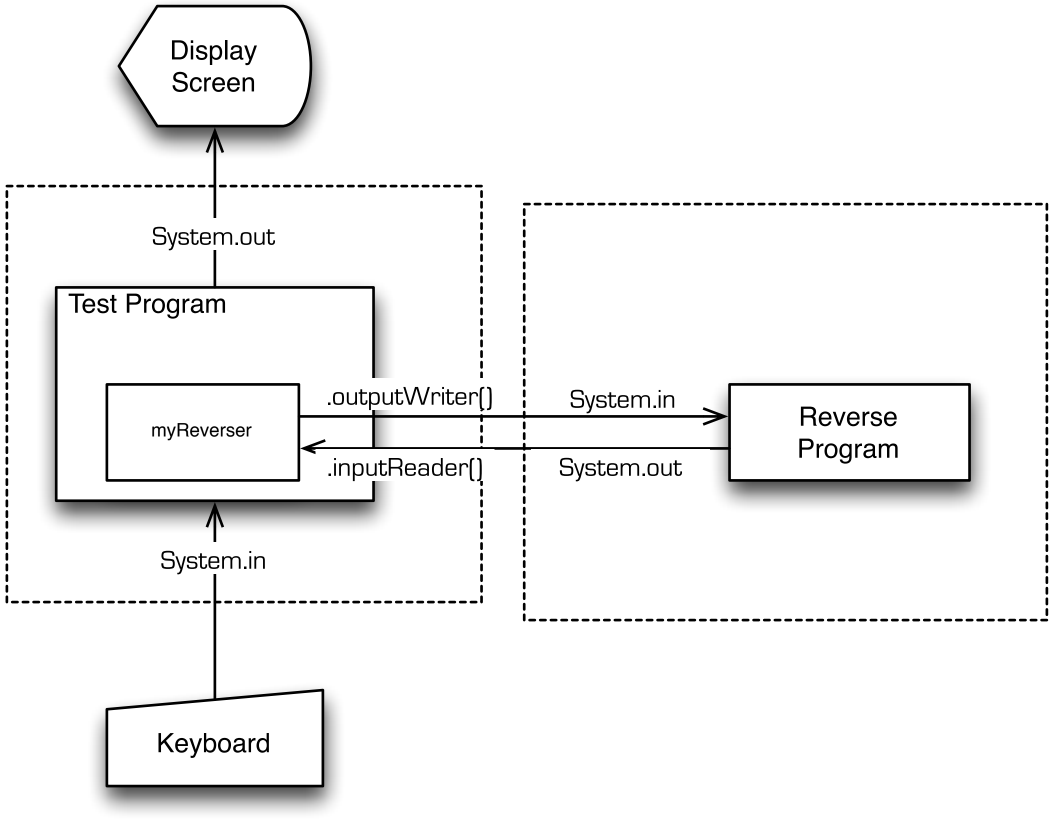

Figure 1: Illustration of a program using `JavaPipe`

Assuming that all of the defaults are set correctly, you can set up a pipeline to this program as shown next. (Explanation follows.)

```
1  import java.util.Scanner ;
2  import ca.unbc.cpsc101.JavaPipe ;
3  //...
4      JavaPipe myReverser = new JavaPipe("Reverse");
5      myReverser.start() ; // causes the program to start and
6                           // and the readers and writers to be set up.
7      Scanner pipeScanner = new Scanner(myReverser.getInputReader()) ;
8      myReverser.getOutputWriter().println("derF") ;
9      System.out.println(pipeScanner.nextLine()) ;
10     myReverser.destroy() ;
```

(see Figure 1)

The import on line 2 shows that the `JavaPipe` class is found in package `ca.unbc.cpsc101`. The constructor on line 4 creates a `JavaPipe` object, but it does *not* start the process.

The `myReverser.start()` command on line 5 starts up a second JAVA process that runs the main method in the `Reverse.class`. However, this second JAVA process does not have `System.in` and `System.out` connected to the keyboard and display. Instead they are connected by pipes to `myReverser.getOutputWriter()` and `myReverser.getInputReader()`.

Thus when, on line 8, the main JAVA program writes “”derF”” to `myReverser.getOutputWriter()`, this appears on `System.in` of the secondary JAVA process.

Assuming that the `Reverse` program then writes `"Fred"` to its `System.out` this becomes available as input through `myReverser.getInputReader()`, and can be read through a `java.util.Scanner` in the standard way (lines 7 and 9).

Tests

⇒ Write a simple `Reverse` class and test it directly from the command line. Then test it indirectly through a `JavaPipe` somewhat as shown above.

Caution The timing of the steps is critical. For instance, if one attempts to read from `.getInputReader()` while the process at the other end of the pipe is also waiting for input, the program will hang with both processes waiting for the other to provide input.

⇒ Write a `Responder` class that reads commands as per the Score 4 specification and sends back syntactically correct responses. Test it from the command line. Then test it indirectly through a `JavaPipe`.