Batch Language Processing

Overview

This project will have teams building a batch language processor. The scope of this effort is to build a tool that parses and executes a batch files containing a number of commands. A batch file, or batch, contains one or more commands that are executed sequentially. Each command executed by our batch processor will be executed as a process and communicate (pass information) using files or pipes.

Single Command Batch with Streams

Keeping with the tradition of most operating systems, each command (process) reads text from a stream and writes text into a stream. These streams are named stdin (input) and stdout (output). There is a third stream stderr on which a process write error messages. It is common for commands to read its input from a file directed though stdin and to write its output into a second file directed though stdout.

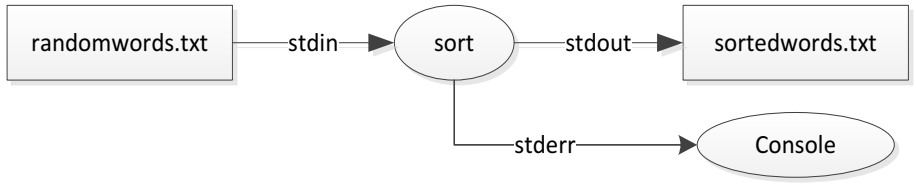


Figure 1: Streams

Figure 1 illustrates the SORT command (running in a process) that is reading unsorted words from a file from stdin, writing the sorted words to a file from stdout, and any error messages are directed to the console though stderr.

Two Command Batch with Streams

A two command batch can be created by using a file as an intermediate stage to transfer the results from the first command to the second.

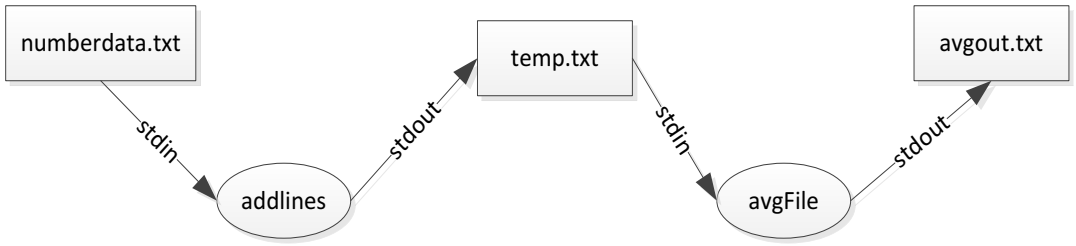


Figure 2: Interconnecting Two Commands With Streams

Two Command Batch with Pipes

One method of interconnecting processes is to connect the stdout of a process to the stdin of a second process. This method is called a “pipe” because bytes written into one of its ends are delivered to the other end.

One method of implementing a pipe is to read bytes from the stdout stream of the source process and write to the stdin stream of the destination process.

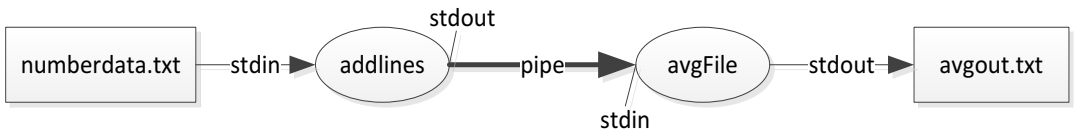


Figure 3: Interconnecting Two Processes across a Pipe.

Batch Language Commands

The following sections describe the four commands to be implemented by this batch processor.

wd Command

wd	
Description	Sets the batch's working directory i.e. the directory the batch will execute within.
Arguments	
id	A name that uniquely identifies the command in the batch file.
path	The path to the working directory

file Command

File	
Description	Identifies a file that is contained within the batch's working directory.
Arguments	
id	A name that uniquely identifies the command in the batch file.
path	The path to the file including its name and extension. The path will always be evaluated relative to the working directory specified by the 'wd' command.

cmd Command

cmd	
Description	A command that will be executed in a process.
Arguments	
id	A name that uniquely identifies the command in the batch file.
path	The path to the executable. If the path is relative, it will use the system's executable PATH to locate the executable file.
args	This is a string that contains the arguments that will be passed to the executable specified by the 'path' option.
in	This is the ID of the file (specified in a file command) that will be directed to the executable process's stdin stream.
out	This is the ID of the file (specified in a file command) that will be directed to the executable process's stdout stream.

pipe Command

Pipe implements connection between two processes (P1 | P2) where the output from P1 is directed (piped) to the input of P2. So this command requires two subcommands that will be executed concurrently.

pipe	
Description	Pipe is an interconnection between two processes (cmd). The two cmd's identified
Arguments	
id	A name that uniquely identifies the pipe in the batch file.
Cmd Element	Two CMD elements that define the P1 and P2 described above. Both commands will be executed concurrently with P1 stdout copied to P2 stdin.

Batch Files to Be Executed

Each of the following sections describes a batch file that your processor will execute. You have been provided the input data files numberdata.txt and randomwords.txt.

Batch1: batch1.xml

A batch that executes the **DOS DIR** command and directs output into a file named dirout.txt.

```
<batch>
	<wd id='swd1' path="work" />
	<file id="file1" path="dirout.txt" />
	<cmd id="cmd1" path="cmd" args='/c dir' out='file1' />
</batch>
```

Batch2: batch2.xml

A batch that executes two commands. Each command sorts the contents of randomwords.txt . The first command sorts and writes its output to sortedwords.txt. The second command reverse sorts and writes its output to reversesort.txt.

```
<batch>
	<wd id='swd1' path="work" />
	<file id="file1" path="randomwords.txt" />
	<file id="file2" path="sortedwords.txt" />
	<cmd id="cmd1" path="sort" in='file1' out='file2' />
	<file id="file3" path="reversesort.txt" />
	<cmd id="cmd2" path="sort" args='/R' in='file2' out='file3' />

</batch>
```

Batch3: batch3.xml

A batch that implements an operation in two steps interconnected through files.

The first command reads the file numberdata.txt which containing several lines and each line contains several numbers. This command will output a several lines where each output line contains the sum of the numbers read from the input file.

The second command will average the numbers read from each line of the file produced by the first command. This command will output the average as a single number.

This batch ties the two operations together using files i.e. the output of cmd1 is the input of cmd2.

Notice that these two commands are implemented as Java applications that have been packaged into jar files. The java runtime executes a jar file using the '-jar' option i.e. java –jar addLines.jar.

```
<batch>
	<wd id='swd1' path="work" />
	<file id="file1" path="numberdata.txt" />
	<file id="file2" path="sumout.txt" />
	<cmd id="cmd1" path="java.exe" args="-jar addLines.jar" in='file1' out='file2' />
	<file id="file3" path="avgout.txt" />
	<cmd id="cmd2" path="java.exe" args="-jar avgFile.jar" in='file2' out='file3' />
```

```
</batch>
```

Batch4: batch4.xml

A batch that implements an operation in two steps interconnected through files.

This batch is similar in function to batch3 except that the two commands will be interconnected though a pipe.

The <pipe> command joins two commands. The output (stdout) from the first command is directed into the input (stdin) of the second command. **NOTE: For the same of simplicity, you can assume that your pipe element contains two and only two <cmd> elements.**

```
<batch>
  <wd id='swd1' path="work" />
  <file id="file1" path="numberdata.txt" />
  <file id="file2" path="avgout1.txt" />
  <pipe id=”pipe1”>
    <cmd id="addLines" path="java.exe" args="-jar addLines.jar" in='file1' />
    <cmd id="avgFile" path="java.exe" args="-jar avgFile.jar" out='file2' />
  </pipe>
</batch>
```

Batch5: batch5.broken.xml

This is an example of a batch containing an error. The file ID in cmd1 is incorrect (i.e. filee2).

```
<batch>
  <wd id='swd1' path="work" />
  <file id="file1" path="numberdata.txt" />
  <file id="file2" path="sumout.txt" />
  <cmd id="cmd1" path="java.exe" args="-jar addLines.jar" in='file1' out='filee2' />
  <file id="file3" path="avgout.txt" />
  <cmd id="cmd2" path="java.exe" args="-jar avgFile.jar" in='file2' out='file3' />
</batch>
```

When executed this file should produce an error message. The format of the error message is up to the team, but for example:

```
Error Processing Batch Unable to locate OUT FileCommand with id: filee2
utdallas.cs4348.batchProcessor.ProcessException: Unable to locate OUT FileCommand with id:
filee2
  at utdallas.cs4348.batchProcessor.commands.CmdCommand.execute(CmdCommand.java:49)
  at utdallas.cs4348.batchProcessor.BatchProcessor.executeBatch(BatchProcessor.java:16)
  at utdallas.cs4348.batchProcessor.BatchProcessor.main(BatchProcessor.java:39)
```

Notice that the error message is produced by an exception handler. This should be a design feature of your processor i.e. convert any library (IOException) or error conditions detected by application logic into an application-specific exception class. Catch and display those exceptions in the main().

Executing Batches

The required method of executing the batch processor will be to package and execute from an executable jar file. In Eclipse, you can export any Run Configuration as a "Runnable Jar" using the export options under the file menu.

It will be possible to execute your processor as an executable jar file. For example :

java –jar myBatchProcessor.jar work\batch1.xml

Trace Messages

As the batch file is executed, trace messages must to be printed to stdout. Each batch command should result in its own trace message. Any errors or exceptions generated are also printed to the console.

For example: batch4.xml produces:

```
Parsing wd
Parsing file
Parsing cmd
Parsing file
Parsing cmd
Parsing pipe
The working directory will be set to work
File Command on file: numberdata.txt
Command: addLines
addLines Deferring Execution
File Command on file: avgout1.txt
Command: avgFile
avgFile Deferring Execution
Pipe Command
Waiting for cmd1 to exit
cmd1 has exited
Waiting for cmd2 to exit
cmd2 has exited
Finished Batch
```

The messages presented above are only an example, but your processor should give the operator some idea of the progress of the batch's execution and certainly notify the operator of any errors that occurred.

Recommended Project Design

This section discusses a basic design that should be applied to your implementation. The design makes use of polymorphism in the parsing and executing of the four command types present in the batch language.

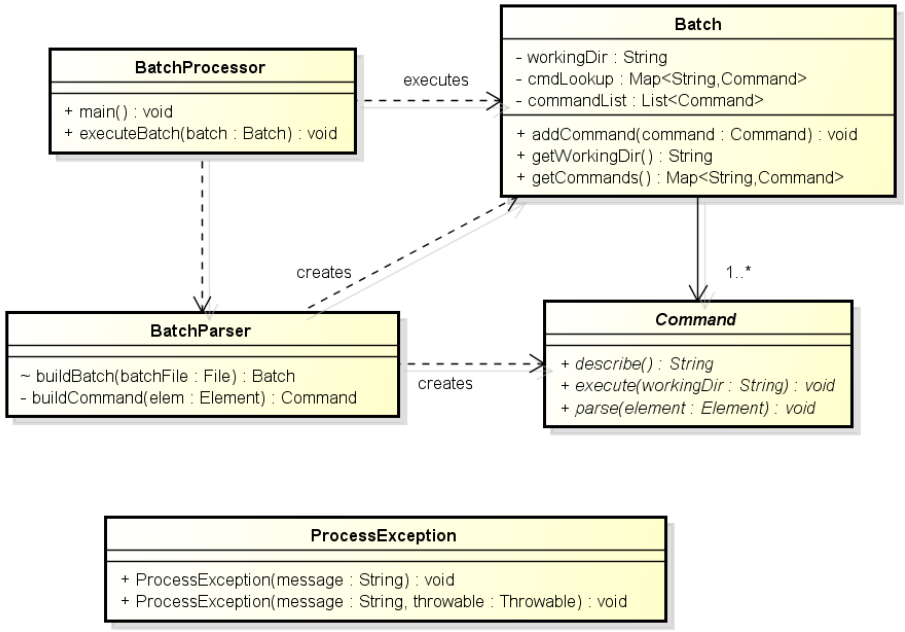
Application Structure

The following UML class diagram presents an overview of the suggested project's design. The application is divided into three processing classes:

BatchProcessor: This is the main class which drives both the parsing of the batch file into commands and the execution of those commands.

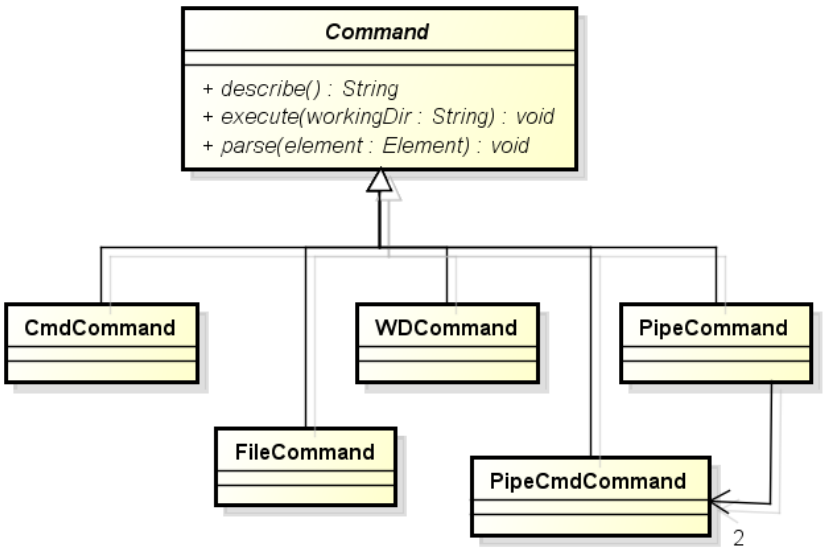
Batch: This class maintains the N Commands that were parsed from the given batch file.

BatchParser: This class builds an instance of Batch containing the N Commands parsed from the XML document provided in the batch file. The parser is responsible for visiting each of the XML elements in the given XML document and generating the correct Command subclass from the element. Note that the actual parsing of the element should be delegated to the correct Command subclass.



Commands Classes

The following UML class diagram describes the recommended structure of the classes that implement each of the four types of batch commands that our batch processor can execute. Note the use of polymorphism in the three basic operations that can be performed on a Command. Note that Command is an abstract class that defines three abstract methods: **describe**() used to print a message to the console when the Command is executed. **parse**() should parse and extract the information contained in the given XML Element. **execute**() should execute the command.



Example Execution

The following UML sequence diagram provides an overview of the two stages of the batch processor's execution. 1) The processor parses the Commands from the given XML batch file. 2) The processor executes each of the Commands.