Key Word In Context – Pipe and Filter + Bridge

Our old “Pipe”, the one that implements queue, is now going to be “PipeQueue”.
You have to create PipeList as well, which uses a List instead of a Queue to accomplish the same goal. You can use any List, but using “Collections.synchronizedList(new ArrayList<>())” is a good solution for our multithreaded use case.

Design requirement:

1. Input, CircularShifter, Alphabetizer, and Output should all be able to use any Pipe.
2. MasterControl will now be customizable.
3. You will be graded on the quality of your bridge pattern implementation. Heed this warning again – fully functional is the bare minimum to get a grade, not to get a perfect grade.

Guidelines:

Follow the tests!

MasterControl:

The start() method will now take in 3 Pipes. The first is for Input to CS. The second is for CS to Alpha. The third is for Alpha to Output. Using these, create the 4 filters with the correct pipes (use null where there should be no pipe).
Then call each of the filters to filter in the correct order.

In the main method, you can launch the start method using any combination of pipes, for example:

```
new MasterControl().start(new PipeList(), new PipeQueue(), new PipeList());
```

Submission

Same as A1. I am grading on design + clean code + following directions.

Educator notes:

Please look at the tests. Due to the nature of the Bridge pattern, there was a lot of duplicated code in the tests, testing queue vs list for each of the classes. I did my best to refactor this duplicate code away, leveraging helper methods.

The bad: I added heavier Thread.sleeps in the tests to decrease the inconsistency in the tests. I will fix this next time I teach this course by having the file name customizable. Please bear with and run tests individually if you have to. You can run any test alone by right clicking on just that test name.