Project 2 - Components

Due Oct 18 by 11:59pm **Points** 40

Due: Sunday, October 18
Points: 40 points (another related assignment will be worth 10 points)
Deliverables:

- Code submitted to Web-CAT

Overview

Create a **bounded Pipe component** (also called a Deque) consistent with the following class diagram.

Pipe Component Diagram (https://drive.google.com/file/d/1JQ435TeUXXRSXueZ23mF_xAnVFcVwYi0/view?usp=sharing)

Please ensure:

- All methods in CircArrayPipe should run in linear time (no loops)
- The append method in AbstractPipe should be implemented as a recursive method (no loops and it should call itself)
- The package name for all classes should be "boundedpipe"
- For other requirements, see grading section (what grader will check) below

The recordings on a bounded stack component should help a great deal, especially with the list-based pipe implementation. The array-based pipe implementation, however, is significantly more complicated than the array-based stack implementation. To get an idea of how a circular-array pipe should work see this **set of slides (https://drive.google.com/file/d/16t417PxAb-1atboZ1R4atW_5acl1QRDn/view?usp=sharing)**. A pipe is similar to a queue. In fact another name for a pipe data structure is a deque. A pointer-stack implementation will be covered in a later Q&A and should help with the linked-pipe.

Bounded Stack Component Recordings

The bounded pipe interface should be named `Pipe` and it should have the following methods:

```
void prepend(E element);
void append(E element);
E removeFirst();
E removeLast();
int length();
int capacity();
Pipe<E> newInstance();
void clear();
boolean isEmpty();
boolean isFull();
```

```
    void append(Pipe<E> that);
    Pipe<E> copy();
```

The Pipe interface should extend `Iterable`. And the pipe should not allow null values. The interface should be fully documented with JavaDocs. The methods should list and document all exceptions. If any argument is a null value, an illegal argument exception should be thrown. If any argument is unacceptable for any reason, an illegal argument exception should be thrown. If any method is called when the pipe is not in a correct state, an illegal state exception should be thrown.

The method newInstance creates a new, empty bounded pipe with the same capacity as the calling pipe. The method append that takes another pipe empties its pipe argument (that).

The complete pipe component will also have an abstract class `AbstractPipe` and three concrete implementations: `ListPipe`, `CircArrayPipe`, and `LinkedPipe`. The `toString` method should look similar to the one in the stack component. For example, `[A, B, C]:20` represents a pipe with a bound of 20, where A is the first element in the pipe and C is the last element in the pipe.

Write unit tests for your classes that cover all of your code. Treat the instructor as your customer: if any requirements seem unclear or ambiguous, ask about them on Piazza! To assist in questions about the interface methods, try to frame your question in terms of pre-state, statement, post-state. For example:

```
How does the removeFirst method work?
For example, what is the post-state in the following code?

[pre-state] p = [A, B, C]:20
[statement] x = p.removeFirst();
[post-state] x = ??? and p = ???
```

Submit your code to Web-CAT.

The Web-CAT portion of Project 2 is worth 40 points:

- 25 points from correctness and code coverage
- 5 points for style
- 10 points for code/implementation issues that grader will check. For example:
    - Pipe interface should have complete Javadocs specifying what each method does. Include all relevant annotations, such as @param, @return, *and* @throws.
    - The "append" method in AbstractPipe must be recursive (no loops)
    - AbstractPipe must only depend on public pipe methods. It should not use any subclasses (ListPipe, CircArrayPipe or LinkedPipe)
    - AbstractPipe should only import Iterator. It should not use any collection or helper classes like Arrays or Collections
    - The equals method should not depend on toString methods, and it should not be implemented by comparing strings. Similarly, the equals method should not depend on the hashCode method

- Concrete classes must not use Java classes beyond those listed in the diagram. For example, Array implementation should not use Arrays, List, or Collections. List implementation should not use Array, Arrays, or Collections. Linked implementation should not use List.
- CircArrayPipe methods must all be constant-time (no loops)

A separate related assignment (questions that require short answers) will be posted in the second or third week and will be worth an additional 10 points.