Implementing a Deque

Summary: In this assignment, you will implement a deque ADT from an interface, while meeting performance constraints by using a list data structure.

1 Background

In this assignment, we will be implementing a Deque ADT (see Requirements). Your job is to implement the interface that is provided for this ADT, given a performance requirement. **All operations (except toString) must be O(1)**. You may not import any packages other than java.util.NoSuchElementException.

In previous courses, the primary technique we used to store collections of information was arrays. Recently, we have been introduced to linked lists. Lists are a new way to structure data that lets us quickly add and remove elements. Unlike arrays - where we have to create an entirely new array and copy the contents of the existing array into it. Of course, we need a way to determine which of these data structures we should use for a particular problem. It is not always the case that we should choose to use a list, they are slower than arrays when it comes to index based access. There are always trade-offs. Considering our problem, we might suspect that a list will be appropriate. The reason for that, is that we need to support enqueue and dequeue in constant time. Arrays won't do that for us - they have a fixed size and require all the elements to be copied to a new array when it needs to be resized. On the flip side, we also know that lists are slower for index based access. Well, Deques won't require index based access. The user can only add/remove elements from the head or the tail. They cannot access arbitrary nodes. Thus, the drawback of lists won't impact us. For this assignment, you should plan to use a doubly linked list.

This document is separated into four sections: Background, Requirements, Testing, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Testing, we give a quick overview of the test code that is already provided. Lastly, Submission discusses how your source code should be submitted on Canvas.

2 Requirements [32 points]

For this assignment, you will be writing a Deque ADT (i.e., LastnameDeque). A deque is closely related to a queue - the name deque stands for "double-ended queue." The difference between the two is that with a deque, you can insert, remove, or view, from either end of the structure. Attached to this assignment are the two source files to get you started:

- Deque.java - the Deque interface; the specification you must implement.
- BaseDeque.java - The driver for Deque testing; includes some simple testing code.

Your first step should be to review these files and satisfy yourself that you understand the specification for a Deque. Your goal is implement this interface. Implementing this ADT will involve creating 9 methods:

- public LastNameDeque() - a constructor [3 points]
- public void enqueueFront(Item element) - see interface [5 points]
- public void enqueueBack(Item element) - see interface [6 points]
- public Item dequeueFront() - see interface [5 points]
- public Item dequeueBack() - see interface [6 points]

- public Item first() - see interface [1 points]
- public Item last() - see interface [1 points]
- public boolean isEmpty() - see interface [1 points]
- public int size() - see interface [1 points]
- public String toString() - see interface [3 points]

From Section 1, be sure that you:

- Make all operations (except toString) work in O(1).
- Do not import any packages other than java.util.NoSuchElementException.
- Use a doubly linked list data structure. Hint: you may need to create a node class.

3 Testing

A few simple test operations have already been provided. These are very simple tests - passing them is *necessary but not sufficient* for knowing that your program works properly. The sample output from these tests is:

```
size: 3
contents:
8 9 4
4
8
9
1
11
size: 2
contents:
1 11
```

When you set about writing your own tests, try to focus on testing the methods in terms of the integrity of the overall linked list. If you compare the tests that you given, with the parts of your program that they actually use (the “code coverage”), you’ll see that these tests only use a fraction of the conditionals that occur in your program. Consider writing tests that use the specific code paths that seem likely to hide a bug. You should also consider testing things that are not readily apparent from the interface specification. For example, write a test that prints your list starting at the head, and another that starts at the end. See if they give the same result.

4 Submission

The submission for this assignment has only one part: a source code submission. It should be uploaded to the submission link on Canvas.

Writeup: For this assignment, no write up is required.

Source Code: Please zip all of your source code (.java) files together as "LastNameDeque.zip" (e.g. "AcunaDeque.zip"). Be sure that you use the correct compression format - if you do not use the right format, we may be unable to your submission. Do not include your project files.