CSC 360 Assignment # 5
Total Points: 50

Implementing Circular Linked Lists

Download from Canvas the following files:

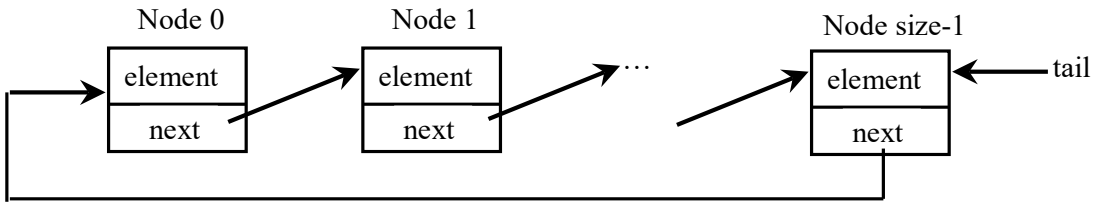
- **MyList.java**
- **TestMyCircularLinkedList.java**

Your task is to create a class **MyCircularLinkedList<E>** (*details described below*) that implements the **MyList<E>** interface and that will pass all of the forty-four tests given in **TestMyCircularLinkedList.java** test program.

The methods that you need to implement are:

```
public void add(int index, E e);
public E get(int index);
public int indexOf(Object e);
public int lastIndexOf(E e);
public E remove(int index);
public E set(int index, E e);
public boolean contains(Object e);
public int size();
public String toString();
public void clear();
public void addFirst(E e);
public void addLast(E e);
public E getFirst();
public E getLast();
public E removeFirst();
public E removeLast();
public java.util.Iterator iterator()
```

The **MyList** interface is the similar to the one used in Chapter 24. The methods in your class are supposed to produce the same results as the corresponding methods in the textbook's **MyLinkedList** class. The big difference is you are ***required*** to implement **MyCircularLinkedList** class as a singly-linked ***circular*** list with a ***tail pointer***, as indicated by the following diagram:



You should have only two data fields in **MyCircularLinkedList**: ***tail*** and ***size***. Note that a data field for the head of the list is unnecessary because, if the list is non-empty, then the head is always *tail.next*. If the list is empty then ***tail*** should be *null* and ***size*** should be *zero*.

Of course, as with Liang's **MyLinkedList** class, you will need to have an *inner class* for your list *nodes* and an *inner class* for your *iterators*. These inner classes will need a few data fields as well, similar to those that Liang used.

Even though the data structure that you use for this assignment is circular, the client code that uses your class will not be able to detect this: Your list will seem like a normal *MyList* object whose *first element* is at *index 0* and whose *last element* is at index *size() - 1*. There is no element before the first element and no element after the last element.

Bonus Point
Implementing the iterator's **remove()** method is only necessary if you want to earn bonus credit *(see the bonus exercise section below)*. If you are not attempting the bonus exercise then please make your iterator's remove() method throw an ***UnsupportedOperationException***.

```
@Override
public void remove() {
    throw new UnsupportedOperationException();
}
```

Simplifying Assumptions
You may assume that **null** values are never put into the list and that the **contains(E e), indexOf(E e),** and **lastIndexOf(E e)** methods are never passed null as a parameter.

Testing
Once you have completed **MyCircularLinkedList**, you should be able to run the **main** method in **TestMyCircularLinkedList.java** and receive a report that forty-four tests therein are successful:

```
(1) [America]
Test 1 successful
(2) [Canada, America]
Test 2 successful
…
Test 43 successful
Test 44 successful
Required tests completed
```

Of course, you are welcome to run additional tests on your solution.

Submission
Submit you **MyCircularLinkedList.java** file on Canvas. Include your name at the top of the file, as well as a brief description of what the code does.

Bonus Implementation (5 points)

For the bonus exercise, implement the *iterator* **remove()** method so that client code can use iterators to remove elements. Recall from Section 20.3 that the job of the **remove()** method is to remove from the underlying collection the last element that was returned by the iterator.

The following documentation from gives more details:
https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html#remove--

> **`default void remove()`**
>
> Removes from the underlying collection the last element returned by this iterator (optional operation). This method can be called only once per call to `next()`. The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.
>
> **Throws:** IllegalStateException - if the next method has not yet been called, or the remove method has already been called after the last call to the next method

If you successfully complete the bonus exercise then you should see the test program report eleven additional successful tests:

```
…
Test 43 successful
Test 44 successful
Required tests completed
Test 45 successful
Test 46 successful
…
Test 55 successful
Bonus exercise tests completed
```

For this bonus exercise, it is permissible for your iterator's **remove()** method to traverse the list. In other words, it is okay if your **remove()** method requires ***O(n)*** time. If you like, you can think about how to make your **remove()** method run in ***O(1)*** time, but it is not required for this exercise.