CS 0445 Spring 2026
Recitation Exercise 6

Introduction:

In Recitation Exercise 4 you completed the implementation of the DLList<T> class, which is a circular, doubly-linked list implementation of the author's ListInterface<T>. **Before completing this exercise, look over Recitation Exercise 4 again and obtain either your solution or my solution to Exercise 4.** That solution will be your starting point for this exercise.

Recently in lecture we have discussed recursion and recursive algorithms. In this exercise you will convert the DLList<T> implementation from Recitation Exercise 4 into a recursive solution, which will contain no loops anywhere in the code.

Exercise:

Make a copy of the DLList.java solution from Recitation Exercise 4 and name it DLListRec.java.

Open that file and read it carefully. Note that there are only three methods in the file that contain iteration (i.e. loops):

```
public boolean contains(T anEntry)

public T [] toArray()

private Node getNodeAt(int givenPosition)
```

Several of the other methods utilize the getNodeAt() method, but they do not directly have any iteration within them. Thus, if you convert the three methods above into recursive versions then the entire class will be free of iteration and have an exclusively recursive solution.

Recursive methods often require additional parameters to allow information to be passed from call to call and to allow for testing for base cases. In order to hide these additional parameters from the user (to utilize functional abstraction) we often implement recursive methods in two parts: 1) a non-recursive stub that satisfies the public specification of the method and 2) a recursive method that does the primary work for the solution. Note that for all of the methods above, the algorithm must proceed down nodes within a linked-list. However, none of the methods has a Node for a parameter. Thus, adding one or more additional parameters to recursive versions of these methods is a good approach. In the stub methods you can test for some special cases and initialize some variables (including creating the array object that must be returned in the toArray() method). You can then pass the necessary parameters to the recursive versions of the methods. The user does not even need to know that recursion is being utilized in this code (other than by the name of the class).

For help with these methods see handouts SeqSLinked.java and SeqSDemo.java but note (very important) that the list in SeqSLinked.java is null-terminated, whereas the list in DLListRec.java is circular. Thus, testing for the end of the list in contains() or toArray() will have to be done in a different way. There is more than one way to handle this – think about how you can do it.

Once you have completed your modified DLListRec<T> class, test it with the main program CS445Rec6.java. Your output should match that shown in file Rec6Out.txt [note that this main program is identical to CS445Rec4.java except in the name of the objects]. To run this program you will need the following files:

    ListInterface.java (provided)
    DLListRec.java (written as specified by you)
    CS445Rec6.java (provided)

ListInterface.java, CS445Rec6.java and DLList.java (the starting point for your new class) are on Canvas at Files/Recitation/Recitation6/. However, if you completed Recitation Exercise 4 correctly, please feel free to start with your own DLList.java file rather than mine.

As usual, if you are able, please volunteer to demonstrate and explain your solution to the rest of your class at the end of recitation. If you are unable to complete the exercise, you may view my solution after it is posted onto Canvas.