This homework will give you practice with writing doubly-linked lists and using subclasses. This is an individual assignment; you may not share code with other students.

Copy the Homework 4 directory by doing the following, starting from your home directory.

```
   cp -r cs61b/hw/hw4 .
   cd hw4
```

When you did Project 1, you probably noticed that the DList ADT doesn't allow you to walk through an DList and process each node as you go. Either you must violate the ADT by manipulating the DListNode pointers directly from your RunLengthEncoding class, or you must use the slow `n() method to access each successive element, thereby obtaining a toOcean() method that runs in time proportional to N^2, where N is the size of the list. Because we didn't know about Java packages, we were unable to develop a really satisfying list ADT.

In this homework, you will implement a doubly-linked list ADT that allows an application to hold list nodes and hop from node to node quickly. How do we make the list an ADT if applications can get access to list nodes? It's easy: we put all the list code in a package called "list", and we declare the fields of DListNode protected—except the "item" field, which is public. Applications can’t access the "prev" or "next" fields of a DListNode, so they can’t violate any List invariants.

I've chosen to make the "item" field public because it doesn't take part in any invariants, so it does no harm to make it public. Applications may read and change "item" as they please. In fact, no method is provided for reading the "item" field indirectly.

Part I (6 points)
------------------------
`list/DList.java` contains a skeleton of a doubly-linked list class. Fill in the method implementations.

Your DList should be circularly-linked, and its head should be a sentinel node (which holds no item) as described in Lecture 8. An empty DList is signified by a sentinel node that points to itself. Some DList methods return DListNode; they should NEVER return the sentinel under any circumstances. Your DList should satisfy the following invariants.

1) For any DList d, d.head != null.
2) For any DListNode x in a DList, x.next != null.
3) For any DListNode x in a DList, x.prev != null.
4) For any DListNode x in a DList, if x.next == y, then y.prev == x.
5) For any DListNode x in a DList, if x.prev == y, then y.next == x.
6) For any DList d, the field d.size is the number of DListNode.

NOT COUNTING the sentinel, that can be accessed from the sentinel (d.head) by a sequence of "next" references.

The DList class includes a `newNode()` method whose sole purpose is to call the DListNode constructor. All of your methods that insert a new node should call this method; they should not call the DListNode constructor directly. This will help minimize the number of methods you need to override in Part III.

Do not change any of the method prototypes; as usual, our test code expects you to adhere to the interface we provide. Do not change the fields of DList or DListNode. You may add private/package helper methods as you please.

You are welcome to create a `main()` method with test code. It will not be graded. We'll be testing your DList class, so you should too.
Submitting your solution

Change (cd) to your hw4 directory, which should contain a file called GRADER and the list directory. The list directory should contain DList.java, DListNode.java, LockDList.java, LockDListNode.java, and any other .java files required by those classes. Make sure your code compiles and your tests run correctly on the _lab_ machines just before you submit.

Your GRADER file should include your name, login, and answer to Part II. Note that it’s just called GRADER, with no extension. "submit" won’t take it if you name it GRADER.txt or something like that. It won’t take "grader" either; it’s gotta be all-caps.

From your hw4 directory, type "submit hw4". (Note that "submit" will not work if you are inside the list directory!) After submitting, if you realize your solution is flawed, you may fix it and submit again. You may submit as often as you like. Only the last version you submit before the deadline will be graded.