CS1332 Summer 2015
Homework 2 Lazy Delete List

Goal: To code a doubly linked list and understand its basic operations, and to create an iterator. We are going to code a lazy delete list.

This is a list that operates in every way like a regular linked list, except for the remove method. In remove, we just mark a node as deleted, but we don't actually remove it from the list. When adding, if there are no deleted nodes, we just add normally at the tail of the list. If there are deleted nodes, then we put the data into the deleted node, and set it to undeleted.

Note that this kind of list would be unordered. Since we are reusing some node at an arbitrary position in the list, the insertion order would not be maintained. You could not use this type of list with data that had to be maintained in insertion order.

The list also has a compress function that will remove all the deleted nodes, yielding a traditional list with all the nodes having undeleted values.

Our list will be doubly linked to make compression easier. To make our operations a little faster, you should maintain a stack of the deleted nodes. On insertion, if the stack is not empty, then pop off a node and update its state to have the inserted value and mark it as undeleted. You don't remove the node from the list and put it on the stack, you just put a reference to it on the stack, and then use this reference to go directly to the node. This also makes keeping the deletedNode count easier since it is just the size of the stack. For this homework, you may use java.util.Stack to store these nodes. Note that Stack is built on class Vector which is synchronized for threading. This makes it a little slower than using an ArrayList, but for the purposes of our homework this work fine and lets you apply using a stack in a useful way.

Our list will also support the Iterable interface, thus we will need to provide an Iterator for our list. Your iterator should support the hasNext and next functions, but it should throw the UnsupportedOperation exception for the remove method.

When an iterator is called for, you should compress the list so there are no deleted nodes. You should make your supporting classes for the iterator as inner classes of the LazyDeleteLinkedList class.

PROVIDED FILES:
LazyDeleteList.java the interface file. You are to implement the methods in this file.
LazyDeleteListTest.java the JUnit test file. Remember from class that this is the last time we will supply you with tests that are exactly the same as we will use for grading. In the future you will get some tests to get you started, but not all the tests we will use.

CODE YOU SHOULD WRITE:

LazyDeleteLinkedList.java this is your class that implements the LazyDeleteList interface using a doubly linked list. The only built-in class you may use is Stack. The actual list must be implemented by you.

Node an inner class that represents a single node in the doubly linked list. It contains data (of type T) and links to the next and previous nodes in the list. Additionally it should have a boolean marker that

indicates whether the node is deleted or not. Make this an inner class of the LazyDeleteLinkedList class.

LazyListIterator an inner class the implements the Iterator interface with the hasNext and next functions. Make this an inner class of LazyDeleteLinkedList.

TURN IN:

The file you wrote: LazyDeleteLinkedList.java

As always, you may add any private helper methods you want in the implementation file, but you may NOT change any of the public interface. Also as usual, the bug bounty extra credit is there if you find bugs in your implementation that are not caught by the provided Junits.

GRADING:

1. Non-compiling code will receive a zero.
2. Use of any Collection class other than Stack will result in a zero.
3. Failure to properly implement the list and obtain the correct running times. Running times are shown on the interface javadoc. Penalties will vary depending on how bad your implementation violates the running times of the operations.
4. If you obey the constraints above and pass the JUnits with a green bar, you will receive a 100 on this homework. This is the last homework we will be giving you complete tests.