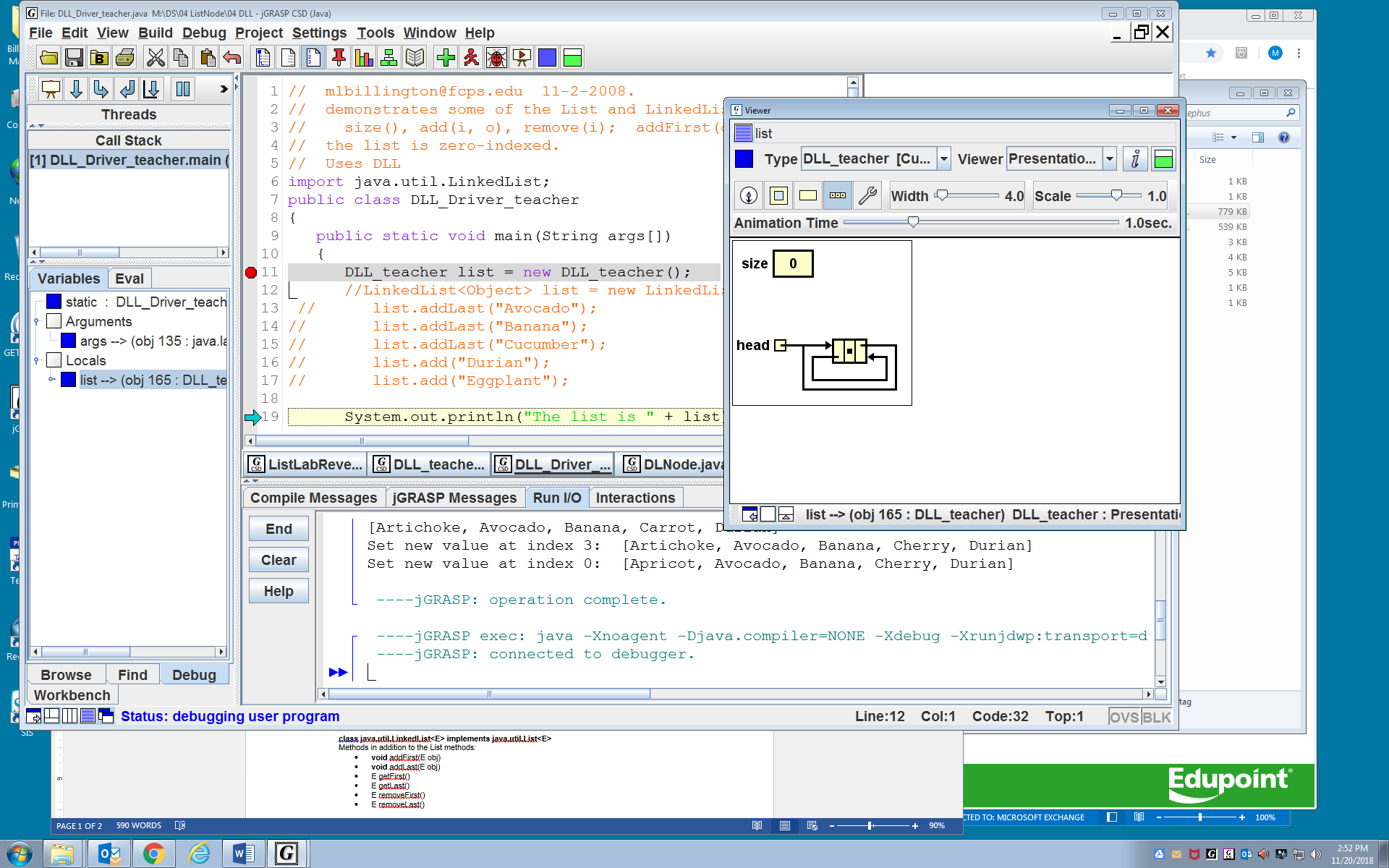
**The DLL Class**

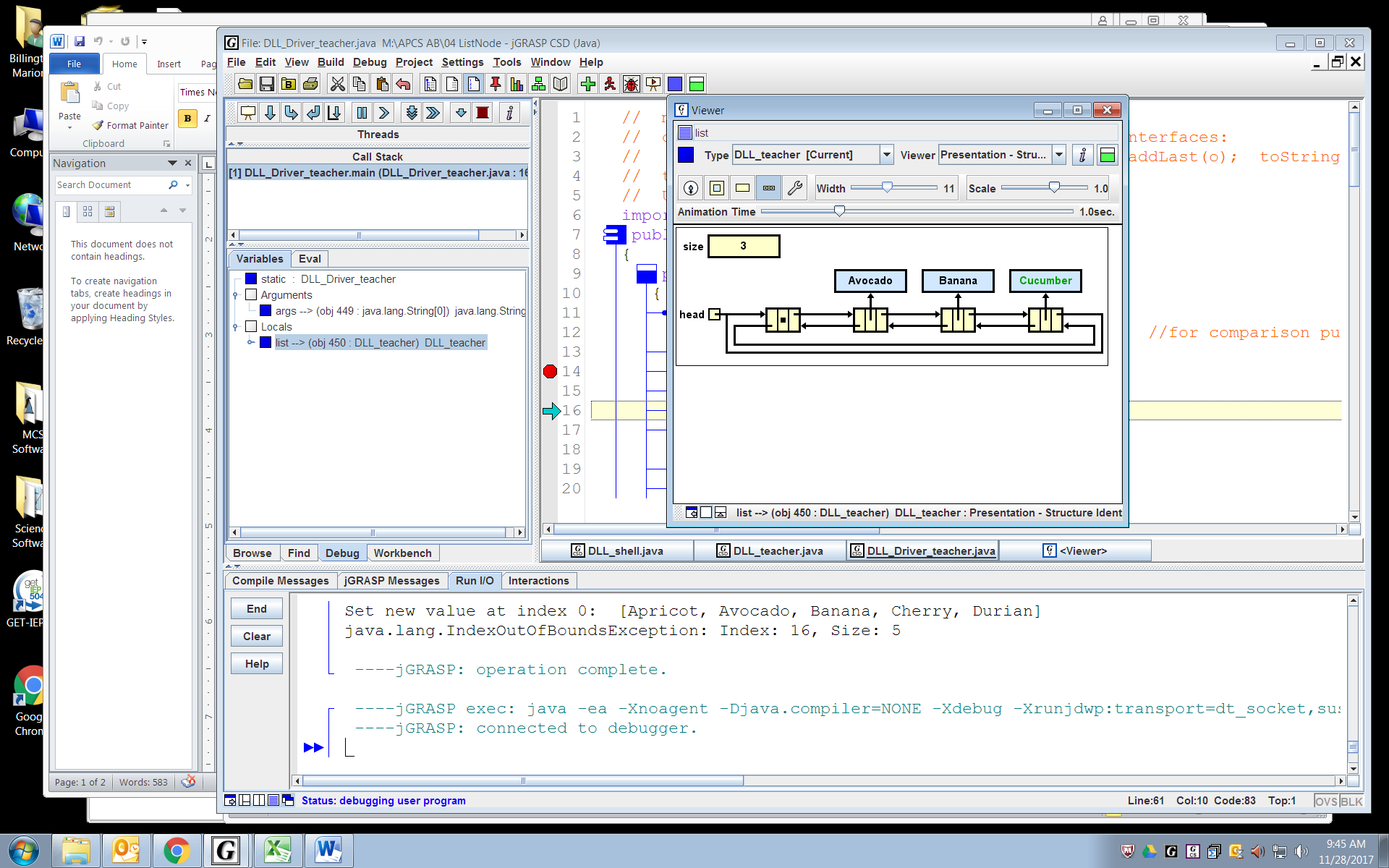
**(Circular, Doubly Linked Lists)**

Of all the species of linked lists, the circular, doubly linked list with a dummy head node is the most powerful. In this lab we make such a class and implement selected methods in Java's List interface and its LinkedList class. We will use Object, not <E>, as the values in each DLNode.



A DLNode object has a private field holding an object, a prev pointer, and a next pointer. In a default DLNode object, the prev and the next both point to this.

The DLL class has two private fields, one of which points to a DLNode object. This DLNode is a dummy node, meaning that it is always present, does not store data, and whose only purpose is to simplify the insertion and deletion operations; specifically, one item of data is no longer a special case. Notice the two private fields in the DLL object named myList below.



DLL myList

Implement the methods for this data structure (using Object instead of <E>) as specified in the two College Board interfaces below. Also write a toString method, so that the list can print itself with commas and square brackets. For example, System.out.println(myList) outputs [Avocado, Banana, Cucumber]

As you write each method, test it in a driver class to see that it actually works. If you are using JGrasp, setting a breakpoint and stepping through the action can be quite instructive.

**interface java.util.List<E>**

* **int** size()
* **boolean** add(E obj) *// appends* obj *to end of list; returns* true
* **void** add(**int** index, E obj) *// inserts* obj *at position* index *(0 <= index <= size)*

*// moving elements at position* index *and higher*

*// to the right (adds 1 to their indices); adjusts size*

* E get(**int** index)
* E set(**int** index, E obj) *// replaces the element at* index *with* obj

*// returns the element formerly at* index

* E remove(**int** index) *// removes element from position* index*, moving*

*// elements at position* index + 1 *and higher to the*

*// left (subtracts 1 from their indices) and adjusts*

*// size; returns the element formerly at* index

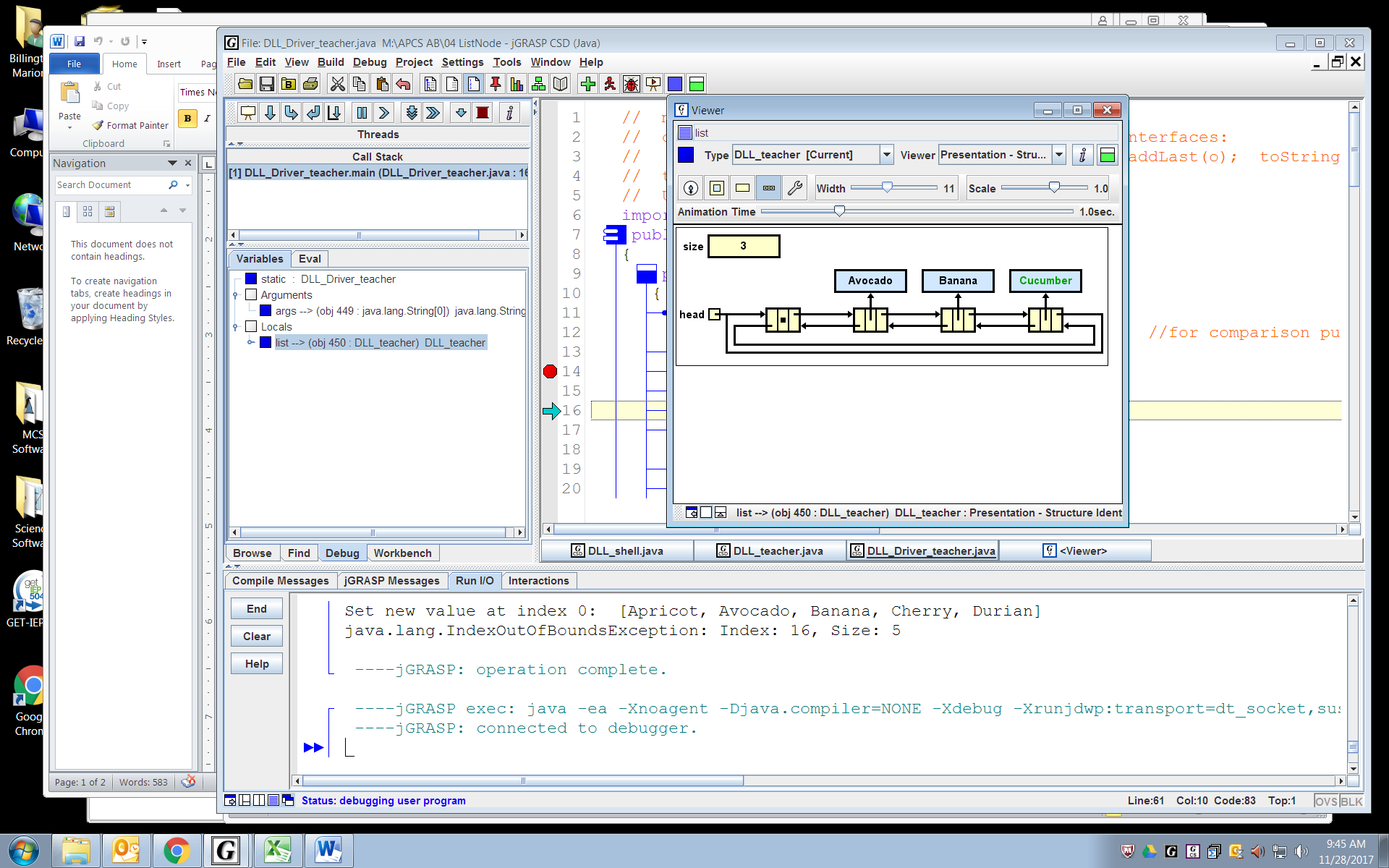
StringtoString()

**class java.util.LinkedList<E>** **implements java.util.List<E>**

Methods in addition to the List methods:

* **void** addFirst(E obj)
* **void** addLast(E obj)
* E getFirst()
* E getLast()
* E removeFirst()
* E removeLast()

**Assignment**



The driver DLL\_Driver is given to you. You are to complete the DLL class, which includes the code for DLNode. You will submit DLL.

class DLL //DoubleLinkedList  
 { private int size = 0;  
 private DLNode head = new DLNode(); //dummy node--very useful--simplifies the code  
 public int size()  
 {  
   
 }

/\* appends obj to end of list; increases size; @return true \*/  
 public boolean add(Object obj)  
 {  
 addLast(obj);  
 return true;  
 }  
 /\* inserts obj at position index (the list is zero-indexed). increments size. \*/  
 public void add(int index, Object obj) throws IndexOutOfBoundsException   
 {   
 if( index > size || index < 0 )  
 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);

}

/\* return obj at position index (zero-indexed). \*/  
 public Object get(int index) throws IndexOutOfBoundsException  
 {  
 if(index >= size || index < 0)  
 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);

}  
 /\* removes the node from position index (zero-indexed). decrements size.  
 @return the object at position index (zero-indexed). \*/  
 public Object remove(int index) throws IndexOutOfBoundsException  
 {  
 if(index >= size || index < 0)  
 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);

}

/\* appends obj to end of list; increases size \*/  
 public void addLast(Object obj)  
 {

}

//lots more methods in this class