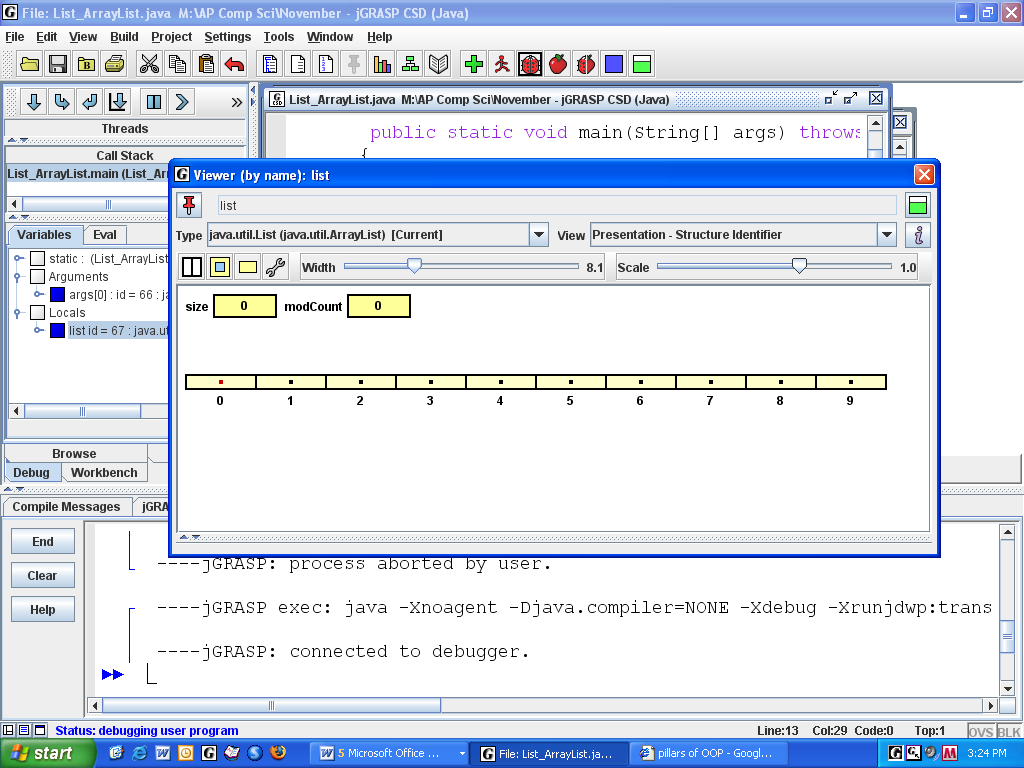
**TJArrayList<E>**

Recall that interfaces have no code. All they do is specify the headers of the methods which the concrete class must implement. You have just written a DLL class that implemented the List<E> interface as a circular, doubly linked list of list nodes. This time, you will write a TJArrayList<E> class that implements the methods from List<E> interface using a *backing array* and using *generic types*. From the perspective of a user, who sees only the interface, the DLL object and the TJArrayList object produce exactly the same behavior, which is what the user cares about. The implementation details are hidden from the user, which is a good thing. Such Encapsulation is one of the four pillars of Object-Oriented Programming, along with Abstraction, Inheritance, and Polymorphism.

Here is a picture of a default TJArrayList<E> object referenced by myList:



myList

**myArrray**

The default TJArrayList has a private myArray of 10 cells (all null) and a private size of 0. The syntax to instantiate an array of generic objects is myArray = (E[]) new Object[10];

Given this data structure, implement the methods specified on the cheat sheet’s List interface.

As you may remember, arrays cannot be re-sized. Consequently, if you add (myArray.length + 1) items to the array, your add methods will have to incorporate the following algorithm: \*\*\* create a new array that is twice as large, copy the old array into it, assign the new element to the next cell, update the size, and make myArray point to the new array.

One of the behaviors of ArrayList<E> is that null cannot appear in the middle of the array. Consequently, your remove method must shift the remaining Objects one cell to the left.

Also write a toString method, inserting commas and square brackets, so that each TJArrayList object can print itself with square brackets and commas, e.g. [Apple, Banana, Fig]. The toString method is called automatically at the command System.out.println(myList);.

As you write each method, test it in a driver class (TJArrayList\_Driver) to see that it actually works--especially that the add method actually doubles the size of the array. Setting a breakpoint in JGrasp and using the debugger to step through each line of code can be quite helpful.

**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

**boolean** contains(E obj)

StringtoString()

48 class TJArrayList<E> // generic type  
 49 {  
 50 private int size; //the number of elements stored in TJArrayList  
 51 private E[] myArray;   
 52 public TJArrayList()   
 53 {  
 54 myArray = (E[]) new Object[10]; //default constructor makes 10 cells  
 55 size = 0;  
 56 }  
 57 public int size()  
 58 {  
 59   
 60 }  
 61 /\* appends obj to end of list; increases size;  
 62 must be an O(1) operation when size < array.length,   
 63 and O(n) when it doubles the length of the array.  
 64 @return true \*/  
 65 public boolean add(E obj)  
 66 {  
 67   
 68   
 69   
 70   
 71   
 72   
 73   
 74   
 75   
 76   
 77   
 78 }  
 79 /\* inserts obj at position index. increments size.   
 80 \*/  
 81 public void add(int index, E obj) throws IndexOutOfBoundsException

82 {  
 83 if(index > size || index < 0)  
 84 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);  
 85   
 86   
 87   
 88

89   
 90   
 91   
 92   
 93   
 94   
 95   
 96   
 97   
 98 }  
 99   
100 /\* return obj at position index.   
101 \*/  
102 public E get(int index) throws IndexOutOfBoundsException   
103 {  
104 if(index >= size || index < 0)  
105 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);  
106   
107   
108 }  
109 /\* replaces obj at position index.   
110 @return the original object.   
111 \*/  
112 public E set(int index, E obj) throws IndexOutOfBoundsException   
113 {   
114 if(index >= size || index < 0)  
115 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);  
116

// lots more methods . . . . . . . . . . . . . . . . .