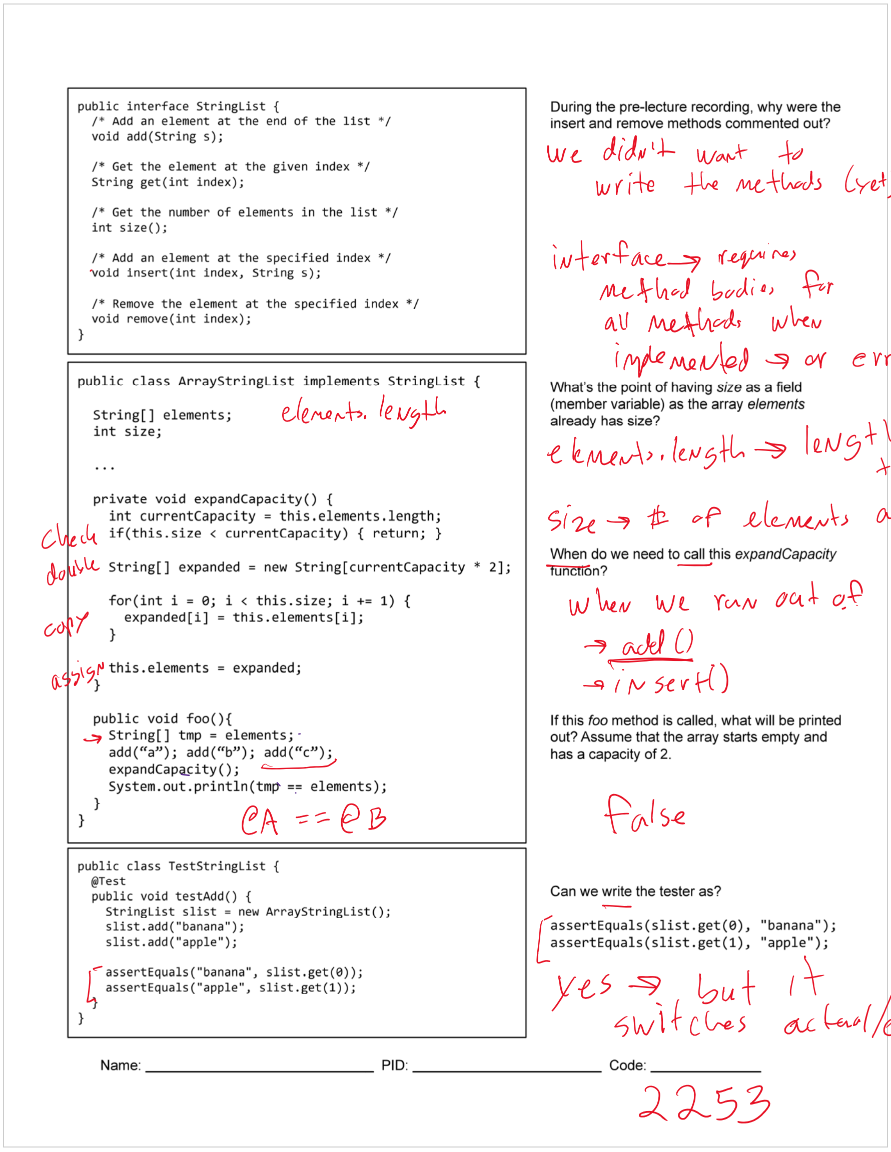
CSE12 - Lecture 4

Monday, April 10, 2023 8:00 AM

PA1 due tomorrow @10pm



```
public interface StringList {
  /* Add an element at the end of the list */
  void add(String s);

  /* Get the element at the given index */
  String get(int index);

  /* Get the number of elements in the list */
  int size();

  /* Add an element at the specified index */
  void insert(int index, String s);

  /* Remove the element at the specified index */
  void remove(int index);
}
```

```
public class ArrayStringList implements StringList {

  String[] elements;        elements.length
  int size;

  ...

  private void expandCapacity() {
    int currentCapacity = this.elements.length;
check if(this.size < currentCapacity) { return; }

double String[] expanded = new String[currentCapacity * 2];

    for(int i = 0; i < this.size; i += 1) {
copy  expanded[i] = this.elements[i];
    }

assign this.elements = expanded;
  }

  public void foo(){
->  String[] tmp = elements;
    add("a"); add("b"); add("c");
    expandCapacity();
    System.out.println(tmp == elements);
  }                          @A == @B
}
```

```
public class TestStringList {
  @Test
  public void testAdd() {
    StringList slist = new ArrayStringList();
    slist.add("banana");
    slist.add("apple");

    assertEquals("banana", slist.get(0));
    assertEquals("apple", slist.get(1));
  }
}
```

During the pre-lecture recording, why were the insert and remove methods commented out?

we didn't want to write the methods (yet)

interface → requires method bodies for all methods when implemented → or error

What's the point of having *size* as a field (member variable) as the array *elements* already has size?

elements.length → length of the array → capacity

size → # of elements added to the data structure

When do we need to call this *expandCapacity* function?

when we run out of space

→ add()

→ insert()

If this *foo* method is called, what will be printed out? Assume that the array starts empty and has a capacity of 2.

false

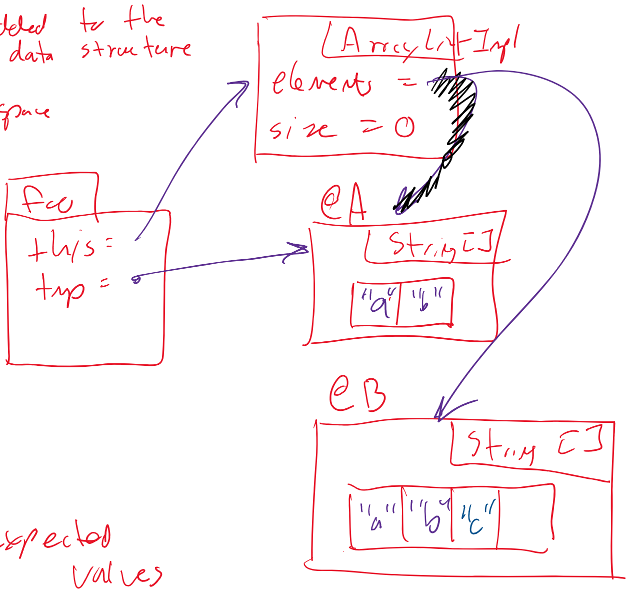
Can we write the tester as?

```
assertEquals(slist.get(0), "banana");
assertEquals(slist.get(1), "apple");
```

yes → but it switches actual/expected values in output jUnit

Name: ______________ PID: ______________ Code: ______________

2253



```
public class ArrayStringList implements StringList {
   ...

  /* Add an element at the end of the list */
  public void add(String s); {
    expandCapacity();
    this.elements[this.size] = s;
    this.size += 1;
  }

  /* Add an element at the specified index */
  public void insert(int index, String s) {

     expandCapacity();
     for ( int i= size-1; i>= index; i--) {
        this.elements[i+1] = this.elements[i];
     }
     this.elements [index] = s;
      this. size  += 1;
  }

  /* Remove the element at the specified index */
  public void remove(int index) {

     for (int i= index; i< size-1; i+=1) {
         this.elements [ i ] = this.elements [i+1];
     }
     this.elements [ size - 1] = null;
      this.size -= 1;

  }
}
```

Write a test case for the ArrayList insert method:

```
ArrayStringList s = new .....
s.insert (0, "a");
assertEquals ( "a", s.elements [0]);
s.insert (0, "b");
assertEquals ("b", s.elements [0]);    ] → assertArrayEquals
             "a", s.elements [1]
```

Implement the ArrayList insert method.

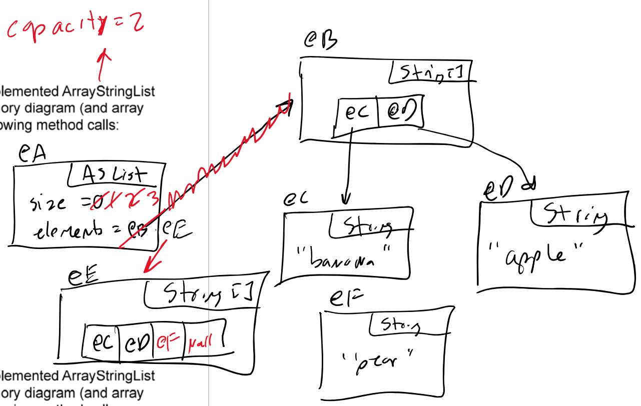
Write a test case for the ArrayList remove method:

```
     s= new
s.add("a");
s.add ("b");
s.remove (0);
assertEquals ("b", s.elements [0]);
```

Implement the ArrayList remove method.

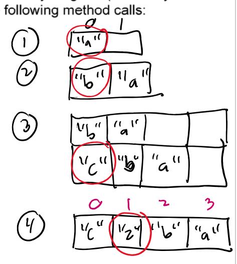
Assuming a fully implemented ArrayStringList class, draw the memory diagram (and array contents) for the following method calls:

add("banana");
add("apple");
add("pear");



Assuming a fully implemented ArrayStringList class, draw the memory diagram (and array contents) for the following method calls:

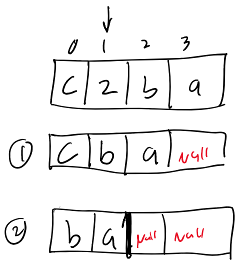
(1) insert(0, "a");
(2) insert(0, "b");
(3) insert(0, "c");
(4) insert(1, "z");



Assuming a fully implemented ArrayStringList class, draw the memory diagram (and array contents) for the following method calls:

add("c");
add("z");
add("b");
add("a");

(1) remove(1);
(2) remove(0);



0
1
2
3
c
z
b
a
c
b
a
null
b
a
null
null