```
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);
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class TestStringList {
  public void testAddThenGet() {
    StringList slist = new ArrayStringList();
    slist.add("banana");
    slist.add("apple");
    assertEquals( barara , slist.get(0));
assertEquals( apple , slist.get(1));
  @Test
  public void testAddThenSize() {
    StringList slist = new ArrayStringList();
    slist.add("banana");
    slist.add("apple");
    assertEquals(
                                      , slist.size());
}
```

```
public class ArrayStringList implements StringList {
  String[] elements;
  // How will we construct it?
  // How will we implement the methods?
}
```

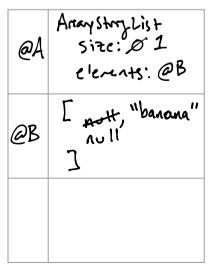
<b>eA</b> .add(	"barana")
this	@A
S	"banana"
returns	: void

.add(	)
this	
s	
returns	:

.get(	)	
this		t
index		i
returns	:	r

.get(	)
this	
index	
returns	:

testAddThenGet()		
slist	@A	
returns: nothing	(void)	



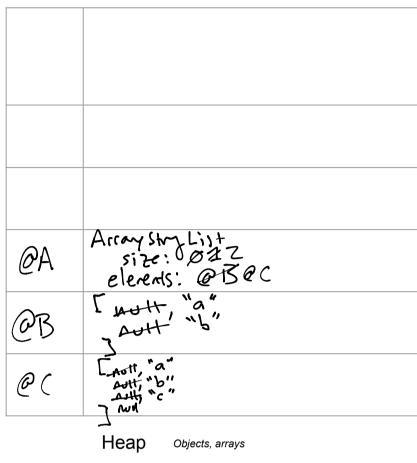
Stack Method calls and variables

Heap

Objects, arrays

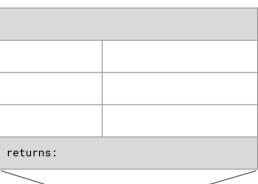
```
public class TestStringList {
  ... all code from other side ...
  @Test
  public void testAddMany() {
     StringList slist = new ArrayStringList();
     slist.add("a");
    slist.add("b");
    slist.add("c");
     slist.add("d");
     slist.add("e");
    assertEquals("e", slist.get(4));
assertEquals("d", slist.get(3));
assertEquals("c", slist.get(2));
assertEquals("b", slist.get(1));
     assertEquals("a", slist.get(0));
public class ArrayStringList {
  ... all code from other side ...
  private void expandCapacity() {
     int currentSize = this.elements.length;
     if(this.size < currentSize) { return; }</pre>
    String[] expanded = new String[currentSize*?]
    for(int i = 0; i < Current Size ; i += 1) {

expanded [ ] ] = this.elements[i];
     this.elements = expanded;
  }
```



## Key ArrayList idea:

when storage runs out in the array stored in elements, make a new array with more capacity and copy elements over.



.add	(	)
this		
s		
returns:		

		_
.add	(	)
this		
s		
return	s:	

.add	(	)
this		
s		
returns:		

.add	(	)
this		
S		
return	s:	

.add	(	)
this		
s		
returns:		

.get	(	)
this		
S		
return	s:	

testAddMany()

slist

returns: nothing (void)