

# CSE12 - Lecture 4

Monday, April 8, 2024 10:00 AM

PA1 due Wednesday @ 8am

## Lecture 4

```
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);
}
```

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

```
public class ArrayStringList implements StringList {
    String[] elements;
    int size;

    ...

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

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

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

        this.elements = expanded;
    }

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

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() → before adding  
→ insert()

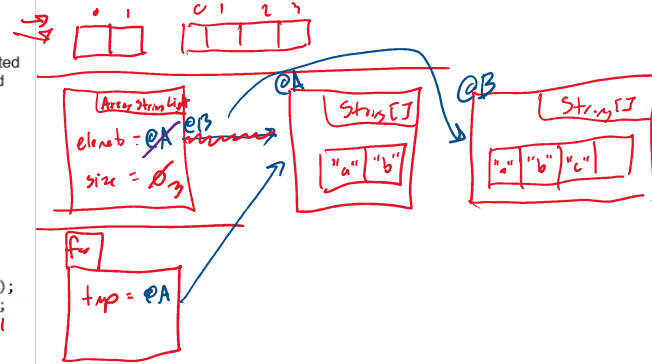
If this foo method is called, what will be printed out? Assume that the array starts empty and has a capacity of 2.

false

Can we write the tester as?

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

Yes → but it switches expanded & actual values reported by JUnit



Name: \_\_\_\_\_ PID: \_\_\_\_\_ Code: 4000

```

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++) {
            this.elements[i] = this.elements[i+1];
        }
        this.elements[size-1] = null;
        this.size -= 1;
    }
}

```

Write a test case for the ArrayList insert method:

```

ArrayList s = new ...
s.insert(0, "a");
assert Equals("a", s.elements[0]);
s.insert(0, "a");
assert Equals("b", s.elements[0]);
    "a", s.elements[1]

```

Implement the ArrayList insert method.

Write a test case for the ArrayList remove method:

```

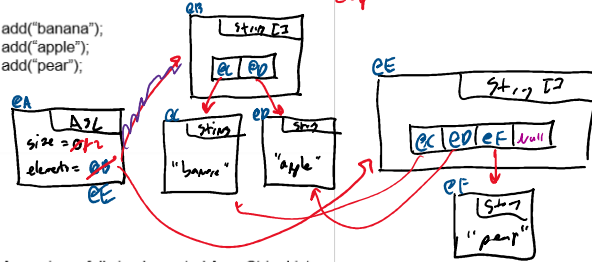
s = new ...
s.add("a");
s.add("b");
s.remove(0);
assert Equals("b", s.elements[0]);

```

Implement the ArrayList remove method.

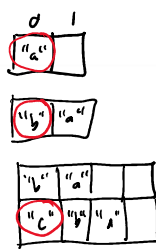
Assuming a fully implemented ArrayStringList class, draw the memory diagram (and array contents) for the following method calls:

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



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

insert(0, "a");  
insert(0, "b");  
insert(0, "c");  
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");

remove(1);  
remove(0);

