

Lecture 3: More ArrayList

* Assignment 1

- Due next Thursday
- Like ArrayList, uses arrays to store data
- Stores a collection of letters
- Seems kind of boring, but good review, and we will use it later in the quarter
- Resources
 - Style/commenting guides §
 - IPL
 - Message board

* A summary of our ArrayList so far

- Added: a second add() method
 - First add calls the second add - less redundancy
- Added: a set() method
 - Why isn't it enough to have add() and remove() (which could do it)?
- More extensive exception checks
- More commenting
 - This stuff is HARD! We are picky
 - An extra private exception check method
 - Must follow the public interface of the spec EXACTLY
 - What if you work for a company
 - Write a "quick and dirty version", then "nice version"
 - What if the first version had extra methods? People might use them
 - INCLUDES CONSTRUCTORS

* A common operation - adding all element of another structure

- addAll(ArrayList other)

```
public void addAll(ArrayList other) {
    for (int i = 0; i < other.size(); i++)
        add(other.get(i));
}
```
- Start by calling add() using accessor methods
- But it can be more efficient to access fields directly

```
public void addAll(ArrayList other) {
    for (int i = 0; i < other.size(); i++)
        add(other.elementData[i]);
}
```
- Sometimes you can't solve the problem at all without field access
- Very useful for your homework - you'll also have to do some kind of "bulk" method with another object as a parameter

* Another issue with our code: if we run out of room!

- Sometimes you will add enough to exceed the capacity
- What should you do?
 - § ▪ We can't grow the array, because of how it is stored on the

computer

- § ▪ Must be CONTIGUOUS - that's how access is fast

- § ▪ Would overwrite some other objects
 - Create a new, bigger array, and copy things over § ▪ How much should we increase the size?
 - * By 1 --> very inefficient
 - * Double --> if we grow from 100 to 200, only have to copy once
 - * “Amortized” - spread out over the 200 adds, the cost of growth is small
 - * Actual Java ArrayList - 50%
 - Can use Arrays.copyOf()

```
public void ensureCapacity(int capacity) {
    if (capacity > elementData.length) {
        int newCapacity = elementData.length * 2 + 1;
        if (capacity > newCapacity) {
            newCapacity = capacity;
        }
        elementData = Arrays.copyOf(elementData,
newCapacity);
    }
}
```

*Summary

- private fields
- class constants for “magic numbers”
- initialize fields in the constructor
- use “this()” to reduce redundancy in constructor calls
- throw exceptions to prevent misuse of your code
- document all preconditions (including exceptions), postconditions
- boolean zen when dealing with boolean expressions
- when overloading methods, have more general call more specific method
- add private helper methods if needed
- Can access private fields of object in methods of the same class