Adding a value at the end

- How do we add a value to the end?

/* Add a value at the end */
boolean add (String value) {
    elements[numElements] = value;
    numElements++;
    return true;
}

Yikes! What if the array is full???

Changing the size of the ArrayList

- When the array inside the ArrayList becomes full, we need to:
  - Allocate a new, larger array
  - Copy the array elements to the new array

Questions:
- How do we know when the array is full?
- How much bigger should we make the array?
- How do we do this without the programmer using the ArrayList class needing to do anything special?

Knowing when the array is full

- When we constructed the array, we made it large enough to hold capacity ints.
- It’s full when numElements == capacity.
- After we make the array bigger, how will we know when it becomes full again??
- Need to update the value of capacity
Increasing Capacity

```java
private void grow() {
    capacity = 2 * capacity;
    /* Copy the data into a new array,
       filling extra entries with null */
    elements = Arrays.copyOf(elements, capacity);
}
```

O() cost ?
O(n)

Adding a value at the end

- How do we add a value to the end?

```java
/* Add a value at the end */
boolean add(int value) {
    elements[numElements] = value;
    numElements++;
    return true;
}
```

Adding a value at the end

/* Add a value at the end */
boolean add(int value) {
    if (numElements == capacity) {
        grow();
    }
    elements[numElements] = value;
    numElements++;
    return true;
}

What is the O() cost of add when grow is NOT called?

Does it matter that grow is called infrequently?
Adding a value at the end

Assuming the elements array starts with size 1, how many times is `grow` called for the first n calls to add?

```java
boolean add (int value) {
    elements[numElements] = value;
    numElements++;
    return true;
}
```

If the first n calls are \(O(n \log_2 n)\), how much does a single call to add cost?

```java
private void grow() {
    capacity = 2 * capacity;
    elements = Arrays.copyOf(elements, capacity);
}
```
Inserting in the Middle

/* Add a value at a position */
void add (int position, int value) {

    elements[position] = value;
    numElements++;
}

ArrayList size Method

int size() {
    return numElements;
}

ArrayList Limitation

- Only Object types can be placed in ArrayLists.
- Cannot have ArrayLists of primitive types:
  - int, boolean, double
- Wrapper types:
  - Integer, Boolean, Double are ok
  - ArrayList<Integer>
Array List Efficiency

How much is the speed of this methods dependent on the number of elements in the vector?

- int size()
- String get (int position);
- String set (int position, String value);
- boolean add (int value);
- void add (int position, String value);
- String remove (int position);
- void grow();

Summary

- Arrays provide an abstraction of arrays that can change size dynamically
- Arrays give us most of the functionality of arrays plus:
  - Dynamic size
  - Insertion and removal
  - Bounds checking on the position
- Limitation:
  - Can only use Object types for the elements