ArrayLists in memory and code

Tuesday, October 4, 2022 11:21 AM

(this.theArray.length is 3)

eltcount = 0

loc 1012		**ArrList** end: 0
loc 1013	0	null
loc 1014	1	null
loc1015	2	null

eltcount = 1

loc 1012		**ArrList** end: 0
loc 1013	0	“hello”
loc 1014	1	null
loc1015	2	null

eltcount = 2

loc 1012		**ArrList** end: 1
loc 1013	0	“hello”
loc 1014	1	“there”
loc1015	2	null

```
public void addLastNoResize(String newItem) {
    if (!(isEmpty())) {
        this.end = this.end + 1;
    }
    this.eltcount = this.eltcount + 1;
    this.theArray[this.end] = newItem;
}
```

this lets us differentiate between the case where end = 0 because the list is empty vs. end = 0 because that's the last-used slot of the first element in the list (we'll need this distinction when we talk about addFirst next time)

Adding to a full ArrList

Tuesday, October 4, 2022 1:35 PM

loc 1012	**ArrList** theArray: loc1013 end: 2 eltcount: 3
loc 1013	“hello”
loc 1014	“there”
loc1015	“brown”

Assume this ArrList is named AL and was initialized with
`AL = new ArrList(3)`
Now run `AL.addLast(“bear”)`

What happens?

```java
public void addLastNoResize(String newItem) {
    if (!(isEmpty())) {
        this.end = this.end + 1;
    }
    this.eltcount = this.eltcount + 1;
    this.theArray[this.end] = newItem;
}
```

Option 1?

loc 1012	**ArrList** theArray: loc1013 end: 2 eltcount: 3
loc 1013	“hello”
loc 1014	“there”
loc1015	~~“brown”~~ → “bear”

Option 2?

loc 1012	**ArrList** theArray: loc1013 end: 2 eltcount: 3
loc 1013	“hello”
loc 1014	“there”
loc1015	“brown”
loc1016	“bear”

Option 3?

loc 1012	**ArrList** theArray: loc1013 end: 2 eltcount: 3
loc 1013	“hello”
loc 1014	“there”
loc1015	“brown”

Exception!

(Why? Because we increment end to 3, and then try to store "bear" in theArray[3] (loc1016), but Java only gave us up to loc1015 for theArray. Something else may have been put in that memory location after we initialized theArray, and because Java can't guarantee that it hasn't, it throws an Exception.)

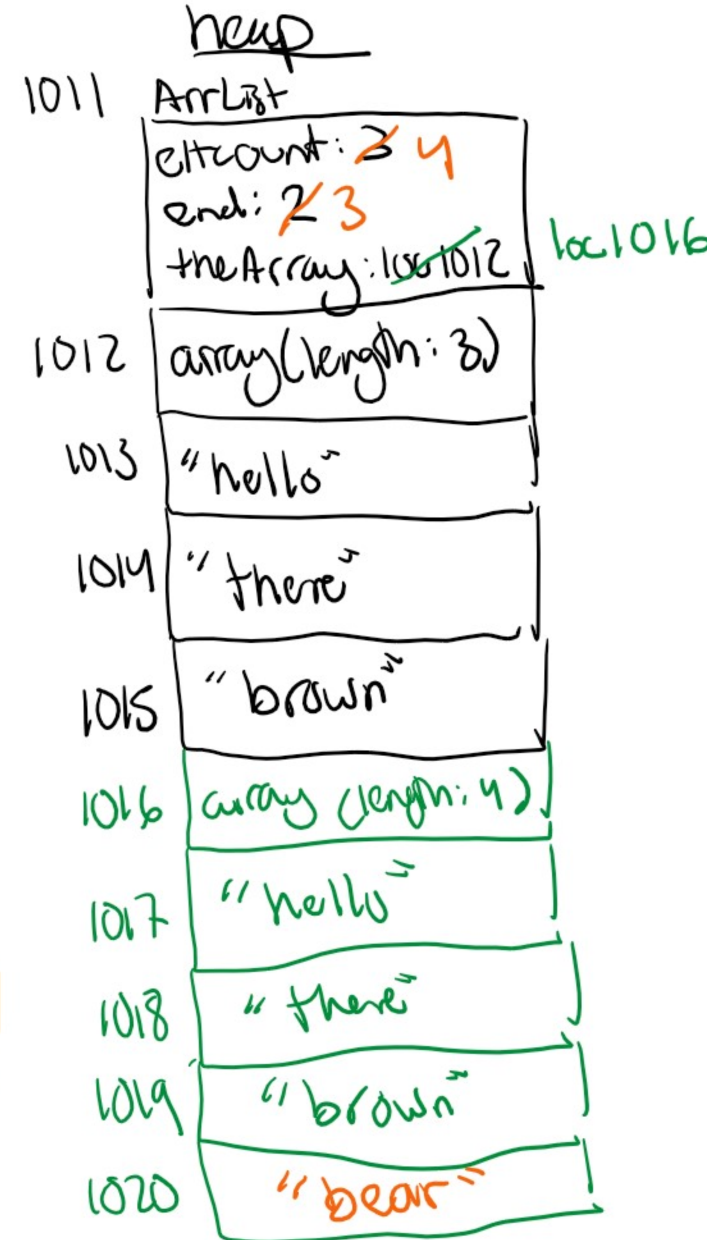
Resizing theArray for addLast

Tuesday, October 4, 2022 1:53 PM

```java
private void resize(int newSize) {
    // make the new array
    String[] newArray = new String[newSize];
    // copy items from the current theArray to newArray
    for (int index = 0; index < theArray.length; index++) {
        newArray[index] = this.theArray[index];
    }
    // change this.theArray to refer to the new, larger array
    this.theArray = newArray;
}
public void addLast(String newItem) {
    if (this.isFull()) {
        // add capacity to the array
        this.resize(this.theArray.length + 1);
        // now that the array has room, add the item
        this.addLast(newItem);
    } else {
        if (!(this.isEmpty())) {
            this.end = this.end + 1;
        }
        this.eltcount = this.eltcount + 1;
        this.theArray[end] = newItem;
    }
}
```

as long as we correctly resized theArray, this recursive call will work because, when we enter the recursive call, this.isFull will be false

Memory when we run AL.addLast("bear"):



Runtime

Tuesday, October 4, 2022 2:40 PM

Summarize Worst-Case Runtimes (in terms of number of elements in the list)

	LinkList	MutableList	ArrList
size	O(N) or O(1)	O(1)	O(1)
addFirst	O(1)	O(1)	...
addLast	O(N)	O(1)	O(N)
get(index)	O(N)	O(N)	O(1)

addLast is O(N) for ArrList because, in the worst case, we have to resize the array, and copying over the elements from the old array to the new array is linear in the number of elements. But we don't always have to do this resizing... can we improve the runtime of addLast?

Cost for addLast:

loc 1012	**ArrList** theArray: loc1013 end: 2 eltcount: 3
loc 1013	"hello"
loc 1014	"there"
loc1015	"brown"

constant
constant
constant
linear
linear
linear
linear
⋮

What if resize added two spaces each time?

constant
constant
constant
linear
constant
linear
constant
⋮

Linear operation becomes less frequent!

In practice, double the size every time (so the cost of the linear operation buys us that many constant operations for the next N adds, which is still linear in the worst case but constant in the typical case)
(this picture is drawn horizontally for screen space reasons)