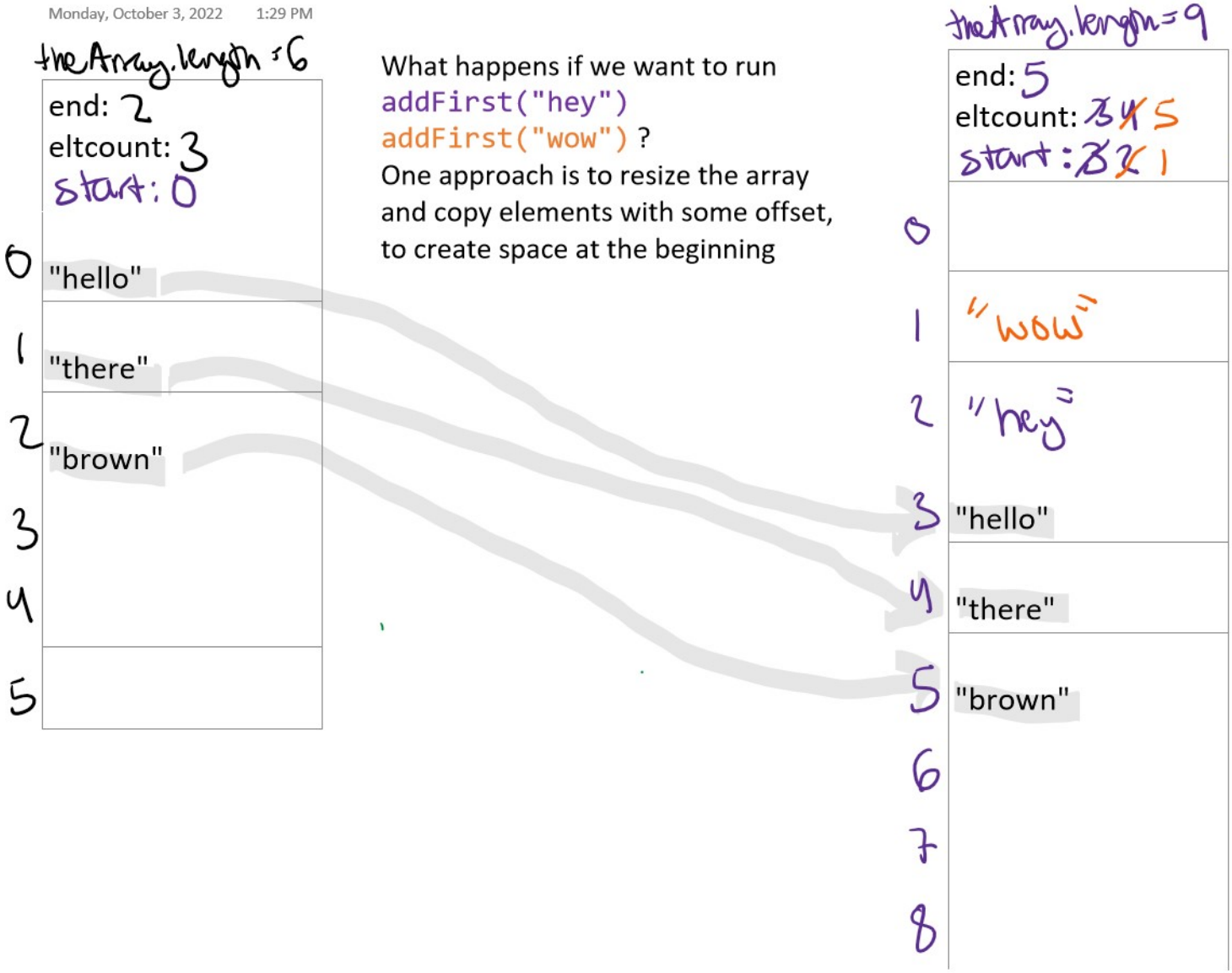
addFirst -- one possible approach

Monday, October 3, 2022 1:29 PM

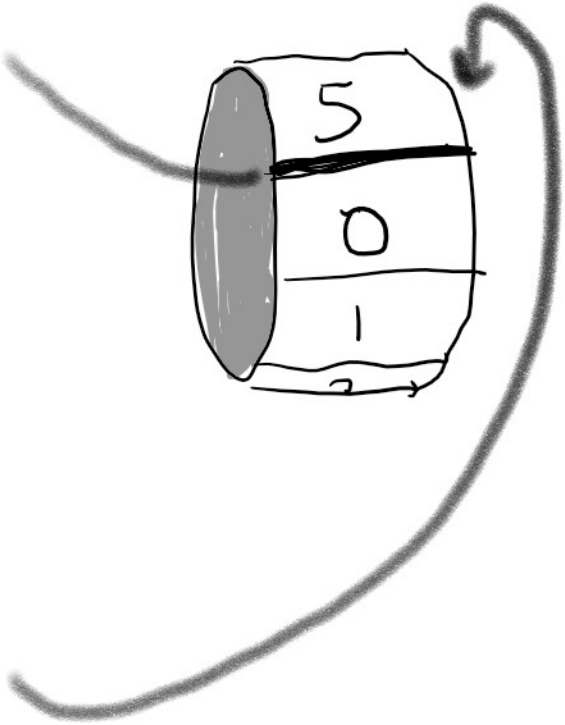


addFirst -- a "circular" approach

Monday, October 3, 2022 1:29 PM

another approach is to use the space at indices 3-5 by imagining the list "wraps around" back to 0 -- like we cut the list out and rolled it into a cylinder:

	end: 2 eltcount: 3 start: 0
0	"hello"
1	"there"
2	"brown"
3	
4	
5	



Then, when we call `addFirst("hey")`, we would want the "hey" to come before the first element (i.e. at index 5):

	end: 2 → 5 eltcount: 3 → 4 start: 0 → 5
0	"hello"
1	"there"
2	"brown"
3	
4	
5	"hey"

coding "circular" addFirst

Tuesday, October 4, 2022 3:26 PM

One way to code this circular case is, for addFirst, to write

```
public void addFirst(String newItem){
    if (this.isFull()){
        // assume resize works as intended for now
        this.resize(this.theArray.length * 2);
        this.addFirst(newItem);
    } else{
        if (!this.isEmpty()) {
            if (this.start == 0) {
                this.start = this.theArray.length-1;
            } else {
                this.start = this.start-1;
            }
        }
        this.eltcount = this.eltcount+1;
        this.theArray[this.start] = newItem;
    }
}
```

wrap this.start around to the last available index of the array (for example, going from 0 to 5 in the previous page)

replaces these lines of code

An alternative to this code is to use %, the remainder. Some examples of modulo:
0 % 6 = 0
1 % 6 = 1
6 % 6 = 0
12 % 6 = 0
15 % 6 = 3

Note that (x + n) % n == x % n (for example, 1 % 6 == 7 % 6 == 1)

In math, -1 % 6 = 5 (because *-1 / 6 = (-1 * 6) + 5*, so 5 is the remainder).
In our code, we want (0 - 1) % 6 to also equal 5, but Java handles modulo of negative numbers strangely, so we add this.theArray.length to make sure this.start - 1 % this.theArray.length is always non-negative:

```
this.start = (this.start - 1 + this.theArray.length) % this.theArray.length
// mathematically equivalent to (this.start - 1) % this.theArray.length
```

debugging addFirst

Monday, October 3, 2022 1:29 PM

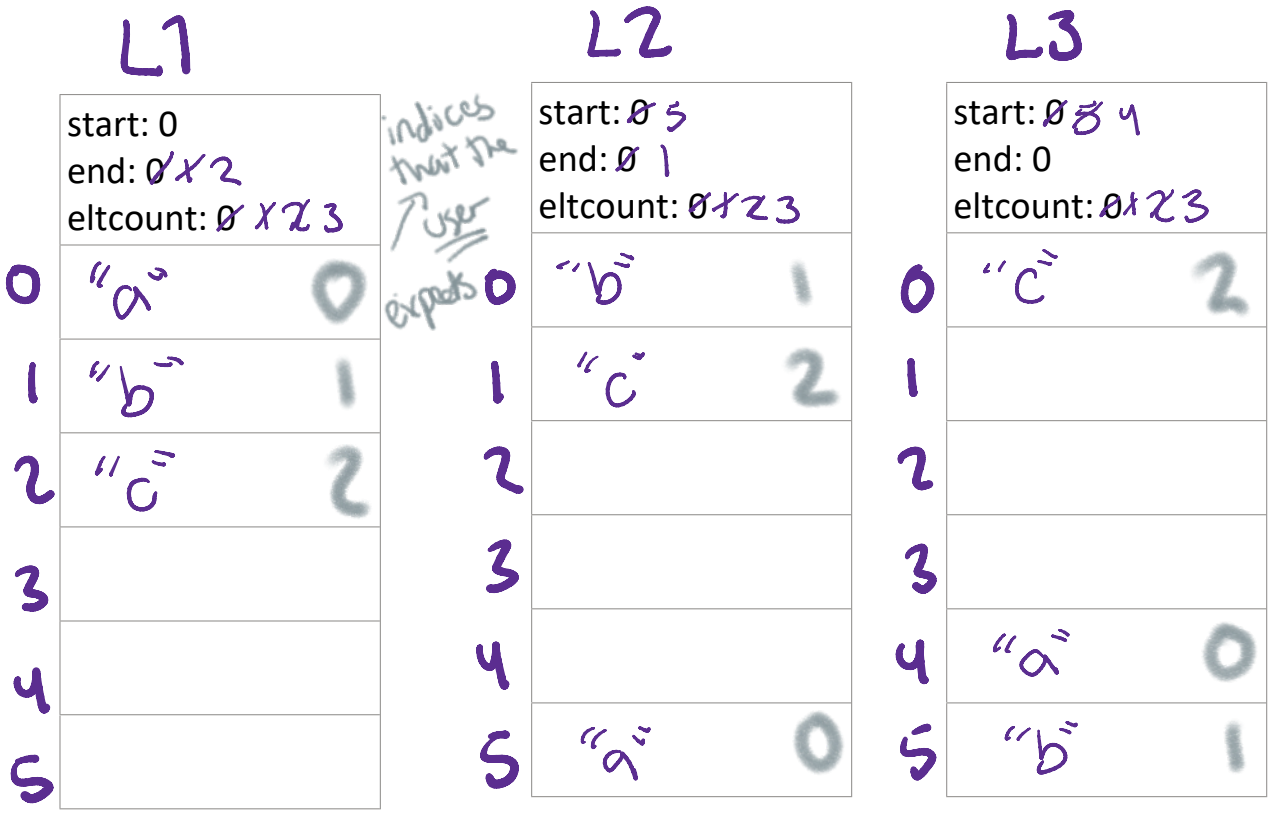
Consider the following test cases, which result in the ArrLists to the right:

```
ArrList L1 = new ArrList(6);
L1.addLast("a");
L1.addLast("b");
L1.addLast("c");

// L1: a b c
assertEquals("b", L1.get(1));

ArrListL2 = new ArrList(6);
L2.addFirst("b");
L2.addFirst("a");
L2.addLast("c");
// L2: a b c
L2.get(1);
assertEquals("b", L2.get(1));

ArrList L3 = new ArrList(6);
L3.addLast("c");
L3.addFirst("b");
L3.addFirst("a");
// L3: a b c
assertEquals("b", L3.get(1));
```



Note that, even though our underlying array has used different slots, and start and end are different for each of L1, L2, and L3, the **user** expects each of the lists to look and behave the same from their perspective. In class, we saw that the second and third assertions failed, and we used these drawings to debug our code. We saw that the debugger showed the same theArray/start/end/eltcount that we expected, which led us to diagnose that our bug was in the get method, which we revised to give back the (wrapped around) offset from the start of the array:

```
public String get(int index) {
    if ((index >= 0) && (index < this.eltcount)) {
        return theArray[(index + this.start) % this.theArray.length];
    }
    throw new IllegalArgumentException("arrayindex" + index + "outofbounds");
}
```