Lesson 72: `ArrayList` Dangers (W23D4)

Balboa High School

Michael Ferraro

February 4, 2016

Do Now

- In a new project called `L72`, create a class called `MyArrayResizer`.
- Create a `main()` that does the following:
  1. declares `char[] word` to be size 4
  2. initializes contents of `word` as

g	r	a	p

  3. prints out the contents of the array[1]
  4. resizes the array to size 6
  5. sets the elements following 'p' to be 'e' and 's'
  6. prints out the new contents of the array

[1] Remind teacher to show you shortcut to print `char[]`.

Aim

Students will examine possible problems when using certain, sometimes dangerous, `ArrayList` methods.

Questions Re: `ArrayList` Methods?

What questions do you have about the methods in PS #12, §7.2 (p322)?

```
boolean isEmpty()
int size()
void add(E obj)
E get(int idx)
E set(i, E obj)
```

```
boolean contains(E obj)
int indexOf(E obj)
String toString()
void add(int idx, E obj)
E remove(int idx)
```

What's the problem with `remove()`?

- In the past, you were shown that `ArrayList`'s `remove()` method is *destructive*, removing elements from an array

What's the problem with `remove()`?

- In the past, you were shown that `ArrayList`'s `remove()` method is *destructive*, removing elements from an array
- That was part of your intro to copy constructors — send a copy of an `ArrayList` to a method so that calls to `remove()` wouldn't permanently change your original `ArrayList`

What's the problem with `remove()`?

- In the past, you were shown that `ArrayList`'s `remove()` method is *destructive*, removing elements from an array
- That was part of your intro to copy constructors — send a copy of an `ArrayList` to a method so that calls to `remove()` wouldn't permanently change your original `ArrayList`
- Sometimes, we *want* to change the `ArrayList`, and `remove()` can have a few unintended consequences...

Consider This Example

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add( new Integer(5) );
list.add( new Integer(4) );
list.add( new Integer(6) );
list.add( new Integer(6) );
list.add( new Integer(-3) );

//remove all instances of '6'
for ( int i = 0 ; i < list.size() ; i++ ) {
    if ( list.get(i) == 6 ) {
        list.remove(i);  // <---- WHAT WILL HAPPEN?
    }                              // PUT CODE IN NEW CLASS
}                                  // AND FIND OUT.
                                   // SUGGEST A 1-LINE FIX!

//recursively calls toString() on each Integer elt:
System.out.println( list );
```

What happened?

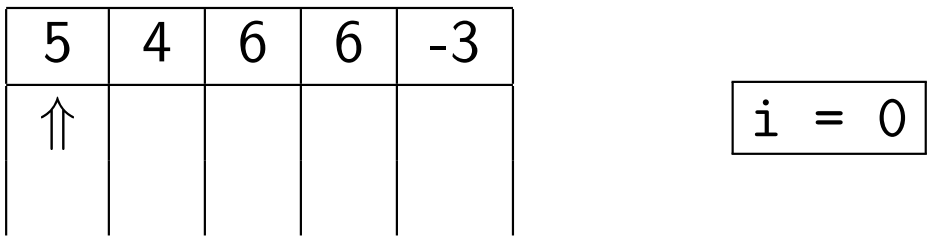
```
for ( int i = 0 ; i < list.size() ; i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	6	-3

i = ?

What happened?

```
for ( int i = 0 ; i < list.size() ; i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```



What happened?

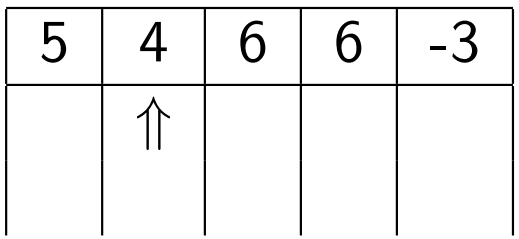
```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	6	-3
⇑ 6?				

i = 0

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```



`i = 1`

What happened?

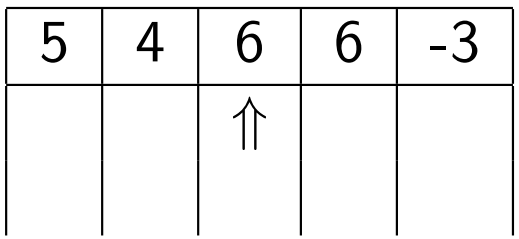
```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	6	-3
	⇑ 6?			

i = 1

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```



`i = 2`

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	6	-3
		⇑ 6?		

i = 2

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	X	6	-3
		⇑ 6?		

i = 2

What happened?

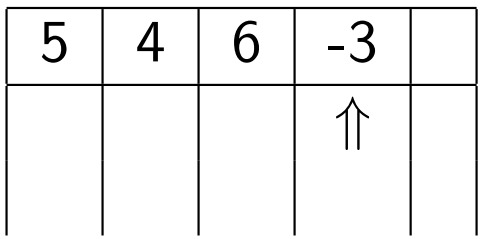
```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	-3	
		⇑ 6?		

i = 2

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```



`i = 3`

What happened?

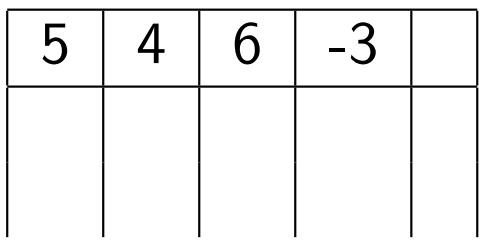
```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```

5	4	6	-3	
			⇑	
			6?	

i = 3

What happened?

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) list.remove(i);
}
```



`i = 3`

Wasn't the goal to remove all 6s?

One-Line Fix

```
for ( int i = 0 ; i < list.size(); i++ ) {

    if ( list.get(i) == 6 ) {
        list.remove(i);
        i--;                              //FIX!
    }

}
```

Fixed Version

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) {
        list.remove(i);
        i--;
    }
}
```

5	4	6	6	-3
		⇑ 6?		

i = 2

Fixed Version

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) {
        list.remove(i);
        i--;
    }
}
```

5	4	X	6	-3
		⇑ 6?		

i = 2

Fixed Version

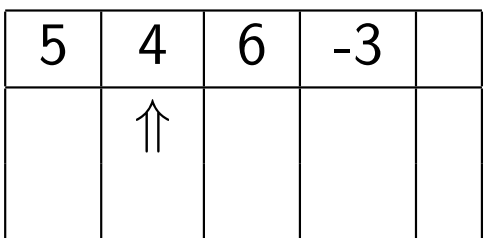
```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) {
        list.remove(i);
        i--;
    }
}
```

5	4	6	-3	
		⇑ 6?		

i = 2

Fixed Version

```
for ( int i = 0 ; i < list.size(); i++ ) {
    if ( list.get(i) == 6 ) {
        list.remove(i);
        i--;
    }
}
```




```
i = 1
```

Another Way Altogether

Litvin doesn't mention a common way to work around the issue illustrated in the last slide!

This alternative approach is presented in PS #12, §7.3.2.

for-each() Loops

Life without for-each():

```
ArrayList<String> words = new ArrayList<String>();

//populate words...

for ( int i = 0 ; i < words.size() ; i++ ) {
    System.out.println( words.get(i) );
}
```

for-each() Loops

Life with for-each():

```
ArrayList<String> words = new ArrayList<String>();

//populate words...

for ( String s : words ) {
    System.out.println( s );
}
```

for-each() Loops

Life with for-each():

```
ArrayList<String> words = new ArrayList<String>();

//populate words...

for ( String s : words ) {
    System.out.println( s );
}
```

- the programmer is freed from dealing with a counter variable and using subscript notation[2]

[2] i.e., the index of an element is used inside square brackets, as in arr[3]

for-each() Loops

Life with for-each():

```
ArrayList<String> words = new ArrayList<String>();

//populate words...

for ( String s : words ) {
    System.out.println( s );
}
```

- the programmer is freed from dealing with a counter variable and using subscript notation
- for-each() is syntactically more elegant, easier

Trouble in Paradise?

- While `for-each()` may appear to have it made, there are occasions when it's simply unusable!

Trouble in Paradise?

- While `for-each()` may appear to have it made, there are occasions when it's simply unusable!
- Litvin §9.6 describes such situations

Trouble in Paradise?

- While `for-each()` may appear to have it made, there are occasions when it's simply unusable!
- Litvin §9.6 describes such situations
- PS #12, §8.3 makes sure you understand these situations!

Next. . .

- Read Litvin §11.5 & §9.6 carefully, making sure to understand the examples.
  - Ch. 9 reading here
  - Ch. 11 reading here
- Work on PS #12, §7.3-§8.

HW

- Finish §§7-8 of PS #12, inclusive.
- Be ready to start §9 next class.