LEC 04

ArrayList

ArrayList

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

Best picnic/BBQ Food?

Instructor Ido Avnon

TAs Abby Williams
Chloë Mi Cartier
Connor Sun
Cynthia Pan
Katharine Zhang
Marcus Sanches
Rohini Arangam

Lecture Outline

- Announcements
- ArrayList Recap
- ArrayList Examples

Announcements

- Programming Assignment 0 due Friday, June 5th at 11:59 PM
- Plan to release C0 grades and feedback tomorrow!
  - One ESNU grade for Culminating Project Checkpoints
  - General grading turnaround is ~1 week
- Quiz 0 is next Tuesday, June 9th!
  - Check the Ed post for instructions and logistics (coming soon)

Quiz 0 (Func. Decomp., File I/O, ArrayList)

- 50 minute in-person paper quiz
- Please Bring
  - Writing Implement
  - Identification (especially if this is your first time in section)
  - Notes Sheets (if you want)
- Resources
  - Lecture/Section Material
  - Friday ArrayList Example
  - Quiz Resources (coming Friday)
  - Cheat Sheets (provided on quiz)

Lecture Outline

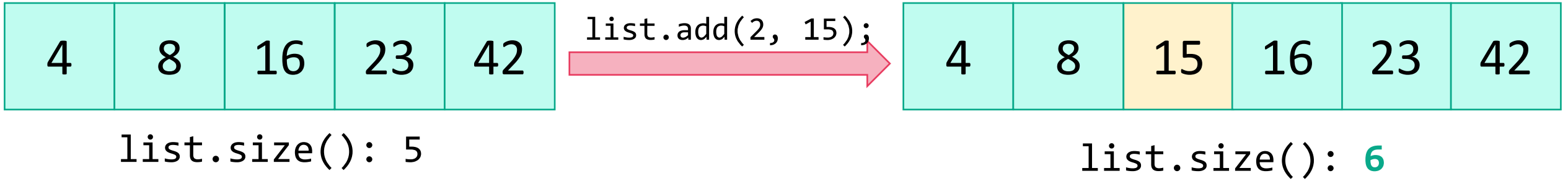
- Announcements
- ArrayList Recap
- ArrayList Examples

ArrayList

ArrayLists are very similar to arrays

- Can hold multiple pieces of data (elements)
- Zero-based indexing
- Elements must all have the same type
  - ArrayLists can only hold Objects, so might need to use "wrapper" types: `Integer, Double, Boolean, Character, etc.`

But ArrayLists have dynamic length (so they can resize!)



ArrayList Methods

Method	Description
`add(type element)`	Adds *element* to the ***end*** of the `ArrayList`
`add(int index, type element)`	Adds *element* to the specified ***index*** in the `ArrayList`
`size()`	Returns the number of elements in the `ArrayList`
`contains(type element)`	Returns `true` if *element* is contained in the `ArrayList`, `false` otherwise
`get(int index)`	Returns the element at *index* in the `ArrayList`
`remove(int index)`	Removes the element at *index* from the `ArrayList` and returns the removed element.
`indexOf(type element)`	Returns the index of *element* in the `ArrayList`; returns -1 if the *element* doesn't exist in the `ArrayList`
`set(int index, type element)`	Sets the element at *index* to the given *element* and returns the old value

ArrayList Methods

- Whenever referring to “the ArrayList”, we are referring to the ArrayList we're calling the method *on*!

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(0, "world");
list.indexOf("world"); // what is the output?
```

```
String[] list = new String[2];
list[0] = "hello";
list[0] = "world";
list[1] = "hello";
//... indexOf?
```

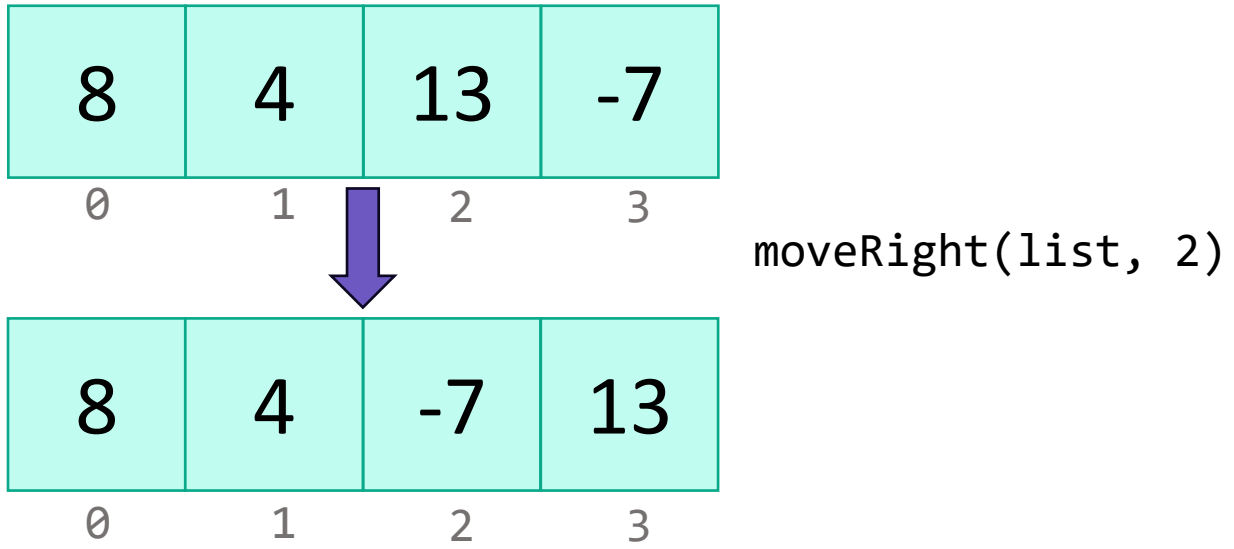
Lecture Outline

- Announcements
- ArrayList Recap
- **ArrayList Examples** 

moveRight

Write a method called `moveRight` that accepts an `ArrayList` of integers `list` and an `int` n and moves the element at index n one space to the right in `list`.

For example, if `list` contains `[8, 4, 13, -7]` and our method is called with `moveRight(list, 2)`, after the method call `list` would contain `[8, 4, -7, 13]`.



Practice : Think

sli.do #cse122

What ArrayList methods (and in what order) could we use to implement the `moveRight` method?

A)
```
list.remove(n);
list.add(n);
```

B)
```
int element = list.remove(n);
list.add(n, element);
```

C)
```
list.add(n);
list.remove(n-1);
```

D)
```
int element = list.remove(n);
list.add(n+1, element);
```

Practice : Pair

sli.do #cse122

What ArrayList methods (and in what order) could we use to implement the `moveRight` method?

A)
```
list.remove(n);
list.add(n);
```

B)
```
int element = list.remove(n);
list.add(n, element);
```

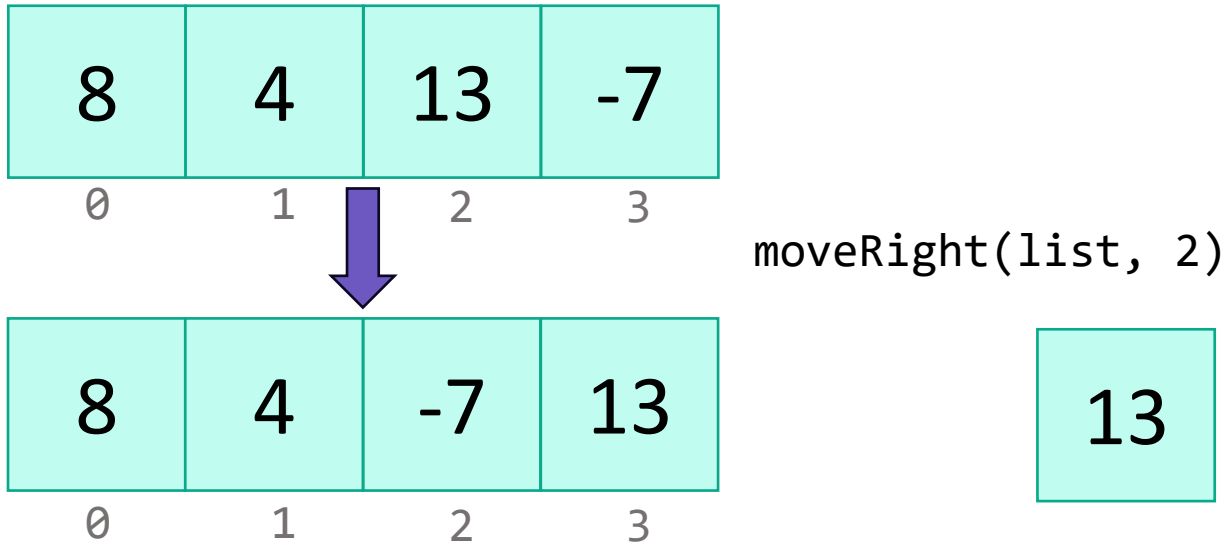
C)
```
list.add(n);
list.remove(n-1);
```

D)
```
int element = list.remove(n);
list.add(n+1, element);
```

moveRight

Write a method called `moveRight` that accepts an `ArrayList` of integers `list` and an `int` `n` and moves the element at index `n` one space to the right in `list`.

For example, if `list` contains `[8, 4, 13, -7]` and our method is called with `moveRight(list, 2)`, after the method call `list` would contain `[8, 4, -7, 13]`.



Edge Cases! (And Testing)

When writing a method, especially one that takes input of some kind (e.g., parameters, user input, a Scanner with input) it's good to think carefully about what assumptions you can make (or cannot make) about this input.

Edge case: A scenario that is uncommon but possible, especially at the “edge” of a parameter's valid range.

❓ What happens if the user passes a negative number to `moveRight`?

❓ What happens if the user passes a number larger than the length of the list to `moveRight`?

More testing tips on the course website's Resources page!

addAll

Write a method called `addAll` that accepts two `ArrayLists` of Characters, `list1` and `list2` , and an integer `location` as parameters and copies **all** of the elements from `list2` into `list1` at the specified `location`.

Before:

```
list1 : ['i', 'a', 'm']
list2 : ['m', 's']

addAll(list1, list2, 1)
```

After:

```
list1 : ['i','m', 's', 'a', 'm']
list2 : ['m', 's']
```

compareToList

Write a method called `compareToList` that accepts two `ArrayLists` of integers `list1` and `list2` as parameters and compares the elements of the two lists, printing out the locations of common elements in each of the `ArrayLists`.

For example, if `list1` contained `[5, 6, 7, 8]` and `list2` contained `[7, 5, 9, 0, 2]`, a call to `compareToList(list1, list2)` would produce output such as:

```
- 5 (list1 at 0, list2 at 1)
- 7 (list1 at 2, list2 at 0)
```

Practice : Think

sli.do #cse122

Spend 2 min discussing about how you would implement this method with a neighbor! Submit a short pseudocode of your answer

Write a method called `compareToList` that accepts two `ArrayLists` of integers `list1` and `list2` as parameters and compares the elements of the two lists, printing out the locations of common elements in each of the `ArrayLists`.

For example, if `list1` contained `[5, 6, 7, 8]` and `list2` contained `[7, 5, 9, 0, 2]`, a call to `compareToList(list1, list2)` would produce output such as:

```
- 5 (list1 at 0, list2 at 1)
- 7 (list1 at 2, list2 at 0)
```

Practice : Pair

sli.do #cse122

Spend 2 min discussing about how you would implement this method with a neighbor! Submit a short pseudocode of your answer

Write a method called `compareToList` that accepts two `ArrayLists` of integers `list1` and `list2` as parameters and compares the elements of the two lists, printing out the locations of common elements in each of the `ArrayLists`.

For example, if `list1` contained `[5, 6, 7, 8]` and `list2` contained `[7, 5, 9, 0, 2]`, a call to `compareToList(list1, list2)` would produce output such as:

```
- 5 (list1 at 0, list2 at 1)
- 7 (list1 at 2, list2 at 0)
```