Lecture 1: Welcome!

CSE 373: Data Structures and Algorithms

Agenda

- Mic check
- Introductions
- Syllabus
- Dust off data structure cobwebs
- Meet the ADT
- List Case Study

Waitlist/ Overloads

- There are no overloads
- Sorry we have no control over these things :/
- Email cse373@cs.washington.edu for all registration questions
- Many students move around, likely a spot will open
- Keep coming to lecture!

Hello!

I am Kasey Champion

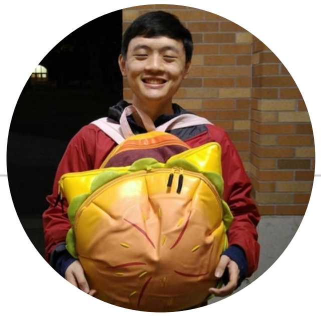
Software Engineer @ Karat

High School Teacher @ Franklin High

champk@cs.washington.edu

@techie4good

Hello!



Zach Chun

Software Engineer @ Amazon

Previously CSE 373 TA

chunz2@uw.edu

Two lecturers??

Please email cse373-20sp-lecturers@u.washington.edu for lecturer specific question / concerns

- Questions about course administration
- Concerns about grading
- Extenuating circumstances

We will both have office hours

- Answer questions about lecture content
- Help with homework
- Discuss any of the above lecturer specific questions

Happy to make specific appointments

- Shared Calendly coming soon

Class Style

Kasey & Zach have to go to their “real jobs” after this

- Your TAs
- Each other

Please come to lecture (yes, there will be recordings)

- Poll-everywhere
- Collaboration (helping other students in the class!)
- Ask questions! Point out mistakes!

Sections

- TAs = heroes
- Exam Practice problems
- Sections start this week

A note about remote life

We are all figuring this out as we go!

Lecture
- Please be prepared to interact throughout the hour
- Poll Everywhere
- Zoom interactions
- Breakouts

Section
- Similar to lecture
- Please be prepared to work with other students
- Video
- Mic

A note about time zones
- We understand many of you are no longer in “PST”
- We will do our best to provide supplemental times

Piazza
- Please feel free to use this to meet and engage with one another

Office Hours
- Please be prepared to share your screen
- Turn on mic and video

Let us know what works!
- Share what you’ve seen elsewhere
- Use the anonymous feedback form
- Always happy to take suggestions / feedback ☺

https://pollev.com/uwcse373

Course Administration

Course Page
- All course content/announcements posted here
- Pay attention for updates!

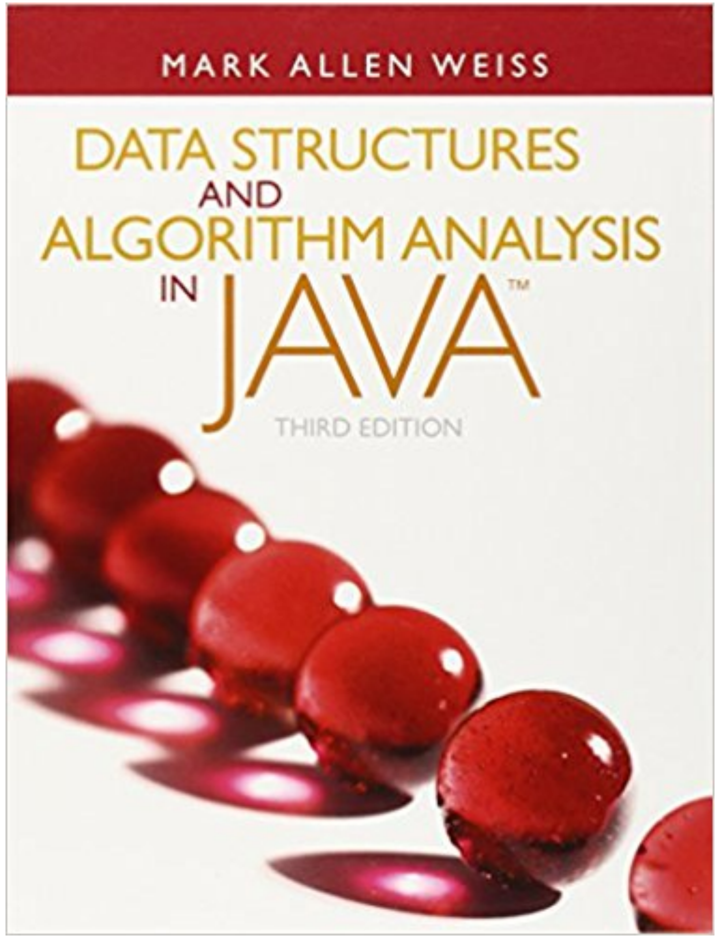
Canvas
- Grades will be posted here

Office Hours
- Will be posted on Course Page
- Will start next week

Piazza
- Great place to discuss questions with other students
- Will be monitored by course staff
- No posting of project code!

Gradescope
- HW Turn in

Textbook
- Optional
- Data Structures and Algorithm Analysis in Java by Mark Allen Weiss



Syllabus + Website + Meet the TAs

To the website!

https://courses.cs.washington.edu/courses/cse373/20sp/syllabus/

Grade Break Down

Homework (55%)

- Programming Projects (35%)
  - Partners **GREATLY** encouraged, but possible to do solo
  - Graded automatically
- Written exercises (20%)
- Turn in your own work, but can collaborate with others (see academic collaboration policy)
  - Graded by TAs

Exams (45%)

- Midterm Exam #1 – Friday April 24th at 8:30-9:20(15%)
- Midterm Exam #2 – Friday May 29th at 8:30-9:20 (15%)
- Final Exam – "Take-home over finals week exam, collaboration encouraged" (15%)

Syllabus

Homework Policies

- 7 late days

Exams

- 2 midterms
  - 50 minutes, during class time
  - Randomized
  - NO COLLABORATION
- 1 final
  - 48 hours to complete
  - Free response style
  - Will be collaborative
- Rules
  - No make ups! Let us know ASAP if you cannot attend an exam
  - Open book

Academic Integrity

- No posting code on discussion board or ANYWHERE online
- We do run MOSS
- No directly sharing code with one another (except for partners)

Extra Credit

- Post lecture-questions
- Worth up to 0.05 GPA bump

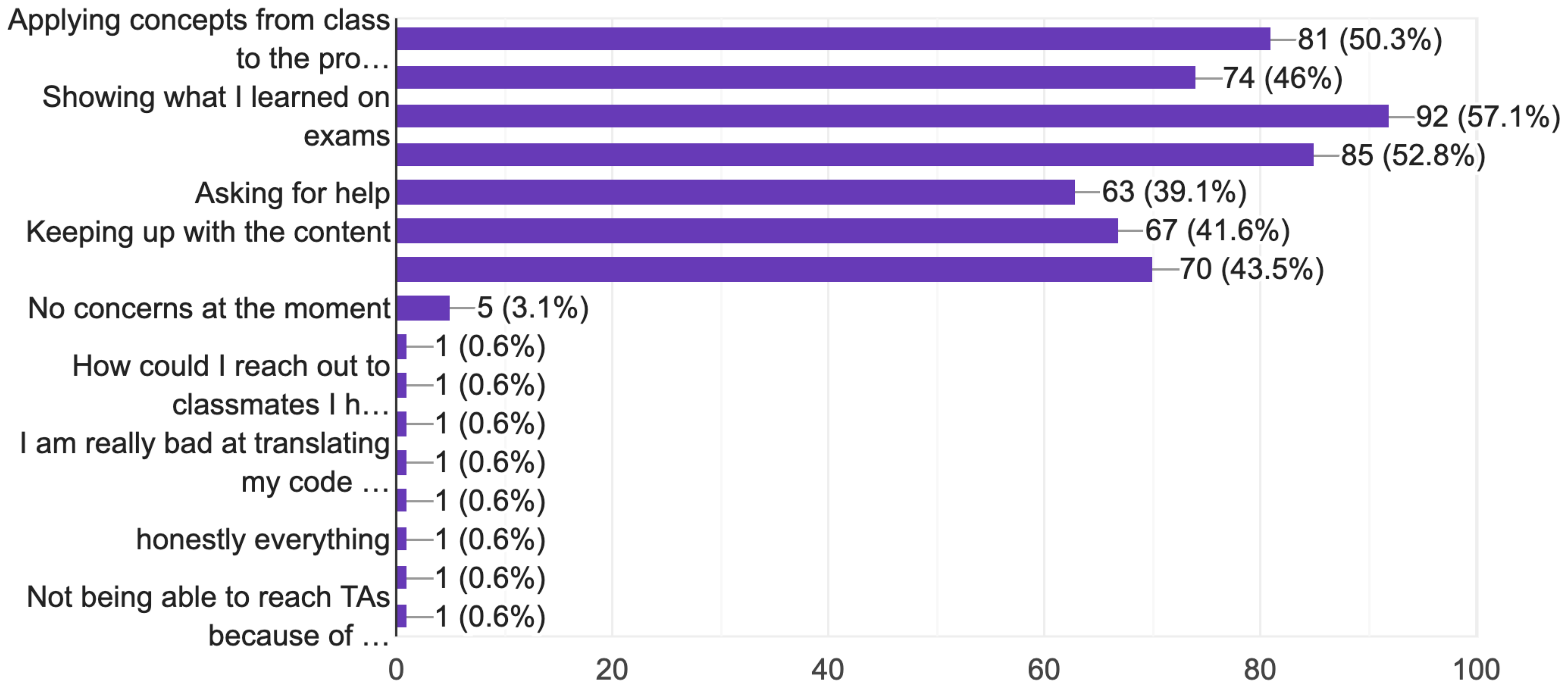
Questions?

Clarification on syllabus, General complaining/moaning

https://pollev.com/uwcse373

What are some of your concerns about the course?

161 responses



What is this class about?

CSE 143 – OBJECT ORIENTED PROGRAMMING

- Classes and Interfaces
- Methods, variables and conditionals
- Loops and recursion
- Linked lists and binary trees
- Sorting and Searching
- O(n) analysis
- Generics

CSE 373 – DATA STRUCTURES AND ALGORITHMS

- Design decisions
- Design analysis
- Implementations of data structures
- Debugging and testing
- Abstract Data Types
- Code Modeling
- Complexity Analysis
- Software Engineering Practices

Data Structures and Algorithms

What are they anyway?

Basic Definitions

Data Structure

- A way of organizing and storing data
- Examples from CSE 14X: arrays, linked lists, stacks, queues, trees

Algorithm

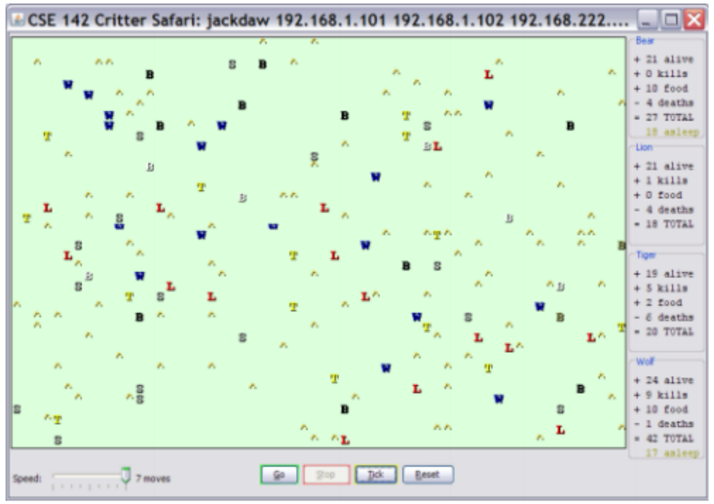
- A series of precise instructions to produce to a specific outcome
- Examples from CSE 14X: binary search, merge sort, recursive backtracking

Review: Clients vs Objects

CLIENT CLASSES

A class that is executable, in Java this means it contains a Main method

```
public static void main(String[] args)
```


CSE 142 Critter Safari: jackdaw 192.168.1.101 192.168.1.102 192.168.222....

```
Bear
+ 21 alive
+ 0 kills
+ 10 food
- 4 deaths
= 27 TOTAL
  10 asleep
Lion
+ 21 alive
+ 1 kills
+ 0 food
- 4 deaths
= 18 TOTAL
Tiger
+ 19 alive
+ 5 kills
+ 2 food
- 6 deaths
= 20 TOTAL
Wolf
+ 24 alive
+ 9 kills
+ 10 food
- 1 deaths
= 42 TOTAL
  17 asleep
```

OBJECT CLASSES

A coded structure that contains data and behavior

Start with the data you want to hold, organize the things you want to enable users to do with that data

1. Ant

constructor	`public Ant(boolean walkSouth)`
color	red
eating behavior	always returns `true`
fighting behavior	always scratch
movement	if the Ant was constructed with a `walkSouth` value of `true`, then alternates between south and east in a zigzag (S, E, S, E, ...); otherwise, if the Ant was constructed with a `walkSouth` value of `false`, then alternates between north and east in a zigzag (N, E, N, E, ...)
toString	`"%"` (percent)

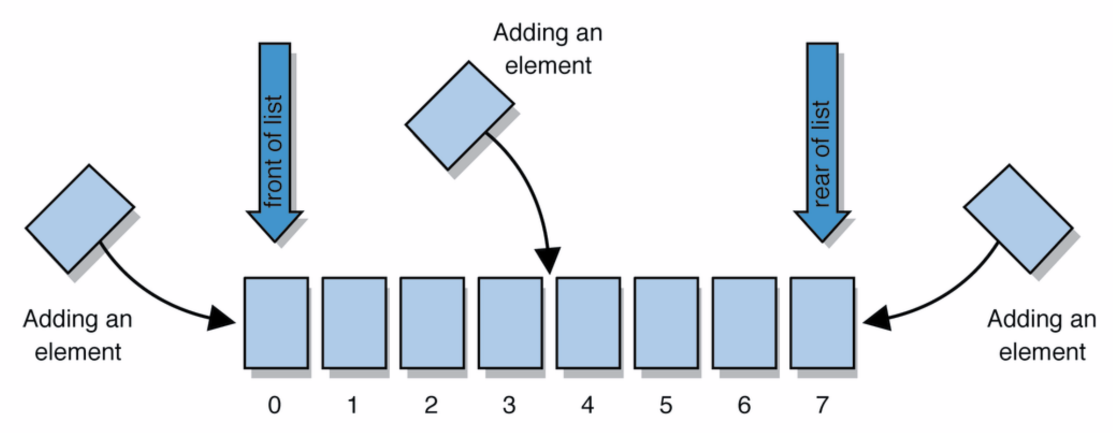
Abstract Data Types (ADT)

Abstract Data Types

- An abstract definition for expected operations and behavior
- Defines the input and outputs, not the implementations

Review: List - a collection storing an ordered sequence of elements

- each element is accessible by a 0-based index
- a list has a size (number of elements that have been added)
- elements can be added to the front, back, or elsewhere
- in Java, a list can be represented as an ArrayList object



Review: Interfaces

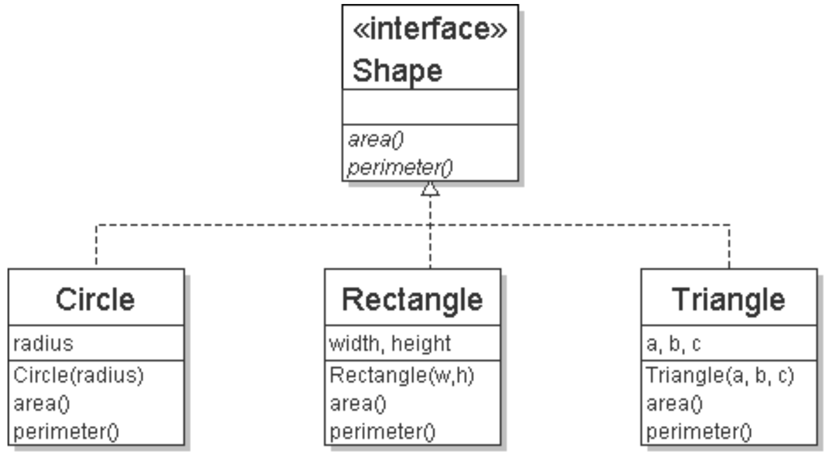
interface: A construct in Java that defines a set of methods that a class promises to implement

- Interfaces give you an is-a relationship *without* code sharing.
  - A `Rectangle` object can be treated as a `Shape` but inherits no code.
- Analogous to non-programming idea of roles or certifications:
  - "I'm certified as a CPA accountant.
    This assures you I know how to do taxes, audits, and consulting."
  - "I'm 'certified' as a Shape, because I implement the Shape interface.
    This assures you I know how to compute my area and perimeter."

```
public interface name {
    public type name(type name, ..., type name);
    public type name(type name, ..., type name);
    ...
    public type name(type name, ..., type name);
}
```

Example

```
// Describes features common to all
// shapes.
public interface Shape {
        public double area();
        public double perimeter();
}
```



Review: Java Collections

Java provides some implementations of ADTs for you!

ADTs	Data Structures
Lists	`List<Integer> a = new ArrayList<Integer>();`
Stacks	`Stack<Character> c = new Stack<Character>();`
Queues	`Queue<String> b = new LinkedList<String>();`
Maps	`Map<String, String> d = new TreeMap<String, String>();`

But some data structures you made from scratch… why?

Linked Lists - LinkedIntList was a collection of ListNode

Binary Search Trees – SearchTree was a collection of SearchTreeNodes

Full Definitions

Abstract Data Type (ADT)

- *A definition for expected operations and behavior*
- A mathematical description of a collection with a set of supported operations and how they should behave when called upon
- Describes what a collection does, not how it does it
- Can be expressed as an interface
- Examples: List, Map, Set

Data Structure

- *A way of organizing and storing related data points*
- An object that implements the functionality of a specified ADT
- Describes exactly how the collection will perform the required operations
- Examples: LinkedIntList, ArrayIntList

ADTs we'll discuss this quarter

- List
- Set
- Map
- Stack
- Queue
- Priority Queue
- Graph
- Disjoint Set

Questions?

Clarification on anything we’ve talked about?

https://pollev.com/uwcse373

This is where we ended for lecture 1

Case Study: The List ADT

list: a collection storing an ordered sequence of elements.

- Each item is accessible by an index.
- A list has a variable size defined as the number of elements in the list
- Elements can be added to or removed from any position in the list

Relation to code and our mental image of a list:

```
List<String> names = new ArrayList<>();        // []
names.size();                                  // evaluates to 0
names.add(”Amanda");                           // [“Amanda”]
names.add(”Anish");                            // [“Amanda, Anish”]
names.insert("Brian”, 0);                      // [“Brian”, “Amanda”, “Anish”]
names.size();                                  // evaluates to 3
```

Case Study: List Implementations

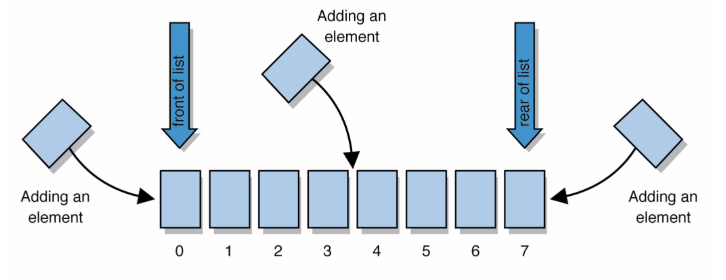
List ADT

state

Set of ordered items
Count of items

behavior

get(index) return item at index
set(item, index) replace item at index
add(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items



ArrayList

uses an Array as underlying storage

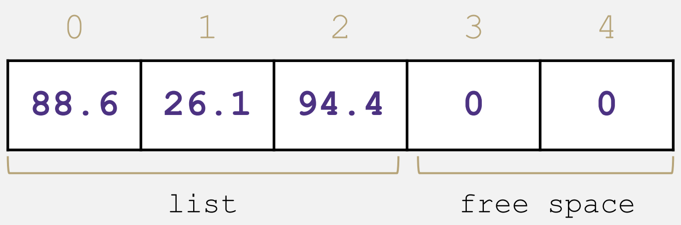
ArrayList<E>

state

```
data[]
size
```

behavior

```
get return data[index]
set data[index] = value
add data[size] = value,
if out of space grow data
insert shift values to
make hole at index,
data[index] = value, if
out of space grow data
delete shift following
values forward
size return size
```



LinkedList

uses nodes as underlying storage

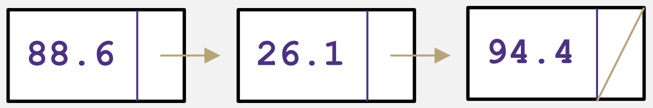
LinkedList<E>

state

```
Node front
size
```

behavior

```
get loop until index,
return node's value
set loop until index,
update node's value
add create new node,
update next of last node
insert create new node,
loop until index, update
next fields
delete loop until index,
skip node
size return size
```



Case Study: The List ADT: ArrayList

How do Java / other programming languages implement ArrayList to achieve all the List behavior?

On the inside:

- stores the elements inside an array (which has a fixed capacity) that typically has more space than currently used (For example when there is only 1 element in the actual list, the array might have 10 spaces for data),

- stores all of these elements at the front of the array and keeps track of how many there are (the size) so that the implementation doesn't get confused enough to look at the empty space. This means that sometimes we will have to do a lot of work to shift the elements around.

List view

["Brian", "Amanda", "Anish"]

ArrayList view

["Brian", "Amanda", "Anish", null, null, null, null]
(this is the internal array with extra space)

Implementing ArrayList

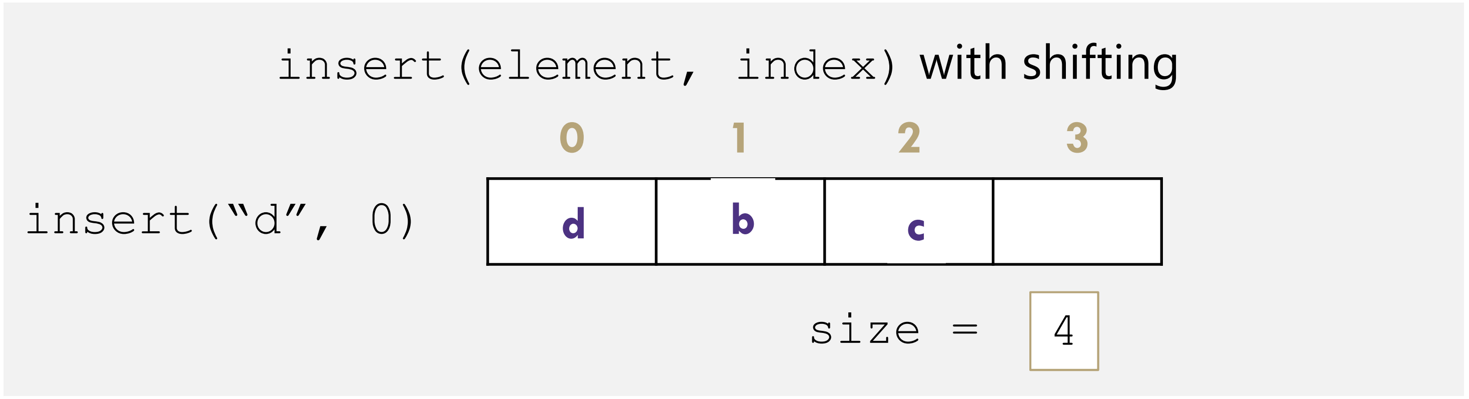
ArrayList<E>

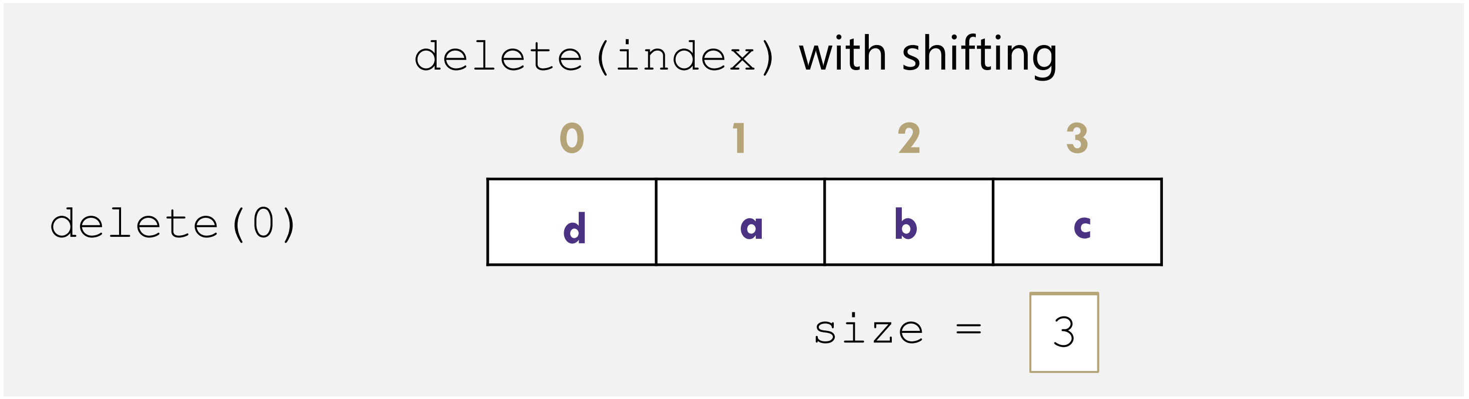
state

```
data[]
size
```

behavior

```
get return data[index]
set data[index] = value
add data[size] = value,
if out of space make data
into a bigger array and
copy everything over
insert shift values to
make hole at index,
data[index] = value, if
out of space grow data
delete shift following
values forward
size return numberOfItems
```





Take 1 Minute

Yes/No should we overwrite index 3 (the old c) with null? Everyone please vote with the yes/ no buttons and message in the chat your explanations why.

Implementing ArrayList

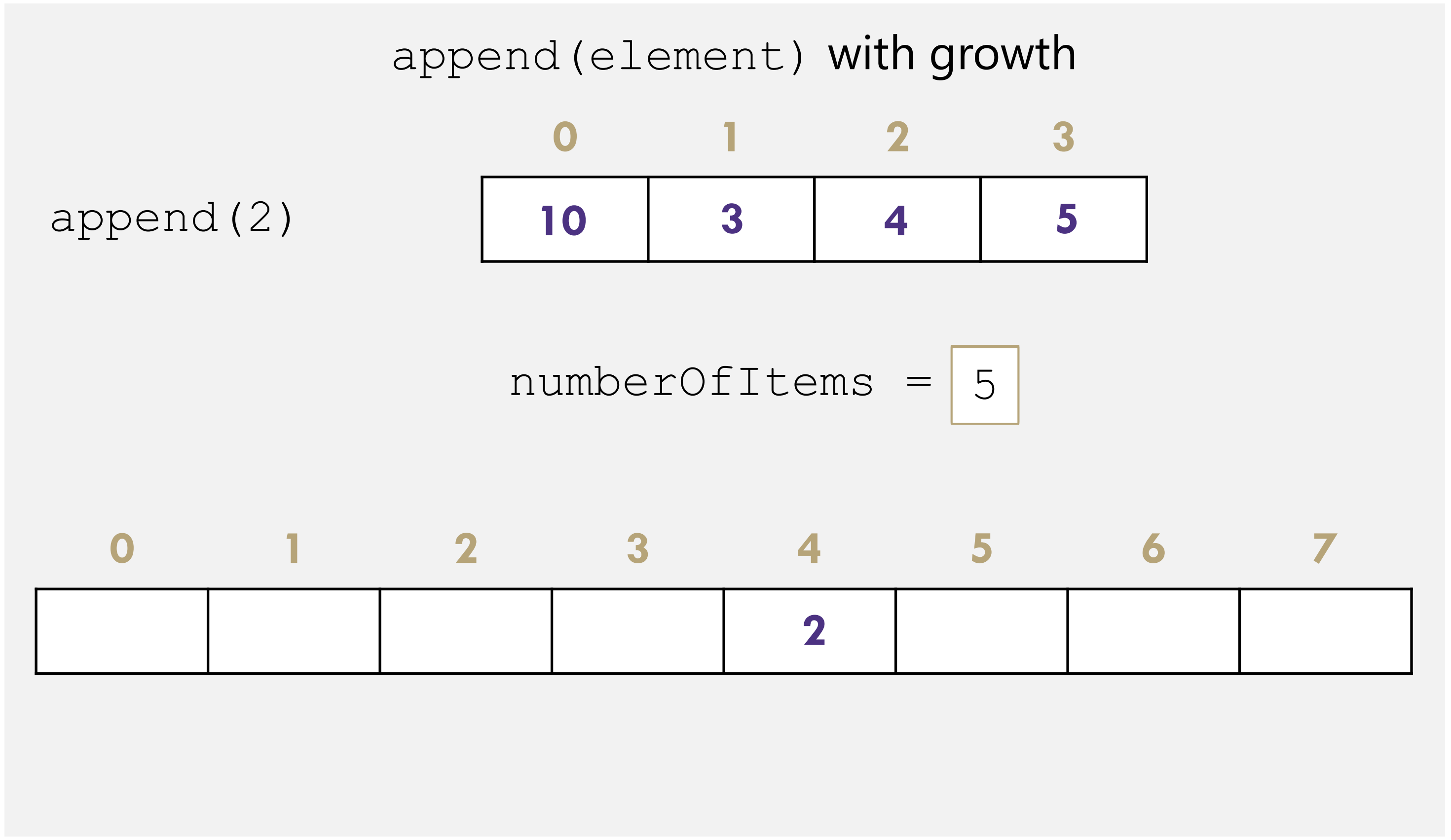
ArrayList<E>

state

```
data[]
size
```

behavior

```
get return data[index]
set data[index] = value
add data[size] = value,
if out of space grow data
insert shift values to
make hole at index,
data[index] = value, if
out of space grow data
delete shift following
values forward
size return size
```



Case Study: List Implementations

List ADT

state
Set of ordered items
Count of items

behavior
get(index) return item at index
set(item, index) replace item at index
add(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items

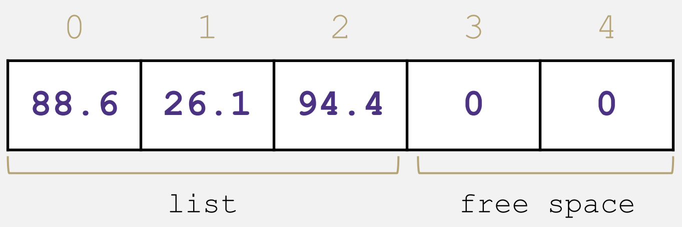
Take 1 Minute spam the chat

What method/situations will be much faster for LinkedList than for ArrayList?

ArrayList
uses an Array as underlying storage

ArrayList<E>

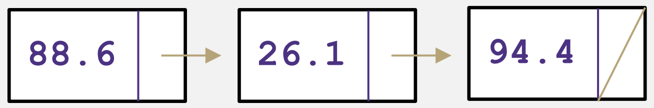
```
state
 data[]
 size
behavior
 get return data[index]
 set data[index] = value
 add data[size] = value,
 if out of space grow data
 insert shift values to
 make hole at index,
 data[index] = value, if
 out of space grow data
 delete shift following
 values forward
 size return size
```



LinkedList
uses nodes as underlying storage

LinkedList<E>

```
state
 Node front
 size
behavior
 get loop until index,
 return node's value
 set loop until index,
 update node's value
 add create new node,
 update next of last node
 insert create new node,
 loop until index, update
 next fields
 delete loop until index,
 skip node
 size return size
```



Design Decisions

For every ADT there are lots of different ways to implement them

Based on your situation you should consider:

- Memory vs Speed
- Generic/Reusability vs Specific/Specialized
- One Function vs Another
- Robustness vs Performance

This class is all about implementing ADTs based on making the right design tradeoffs!

> A common topic in interview questions

Design Decisions

Take 1 Minute spam the chat

Dub Street Burgers is implementing a new system for ticket (i.e. food order) management.

When a new ticket comes in, it is placed at the end of the set of tickets.

Food is prepared in approximately the same order it was requested, but sometimes tickets are fulfilled out of order.

Let's represent tickets as a list. Which of our ADT implementations should we use?

Why?

Design Decisions

Let’s represent tickets as a list. Which of our ADT implementations should we use?

Why?

ArrayList

Creating a new ticket is very fast (as long as we don’t resize), and I want the cooks to be able to see all the orders right away.

LinkedList

We’ll mostly be removing from the front of the list, which is much faster for the linkedlist (no shifting), and I want finished orders to be removed fast so they aren’t distracting.

Design Decisions

Both ArrayList and LinkedList implementations have pros and cons.

Neither is strictly better than the other.

Some major objectives of this course:

Evaluating pros and cons

Deciding on a design

Defending that design decision

Especially when there's more than one possible answer.