Lecture 1: Welcome!

CSE 373: Data Structures and Algorithms

Agenda

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

Welcome!

- The world is *still* BANANAS!
- You are not expected to be at your best this quarter and that is OK
- We are in this together
- Humans first, students second
- Patience, vulnerability, compassion

"We acknowledge that we are on the traditional land of the first people of Seattle, the Duwamish People past and present and honor with gratitude the land itself and the Duwamish Tribe."

https://www.realrentduwamish.org/

Waitlist/ Overloads

- I have told CSE the more the merrier, but technically I 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

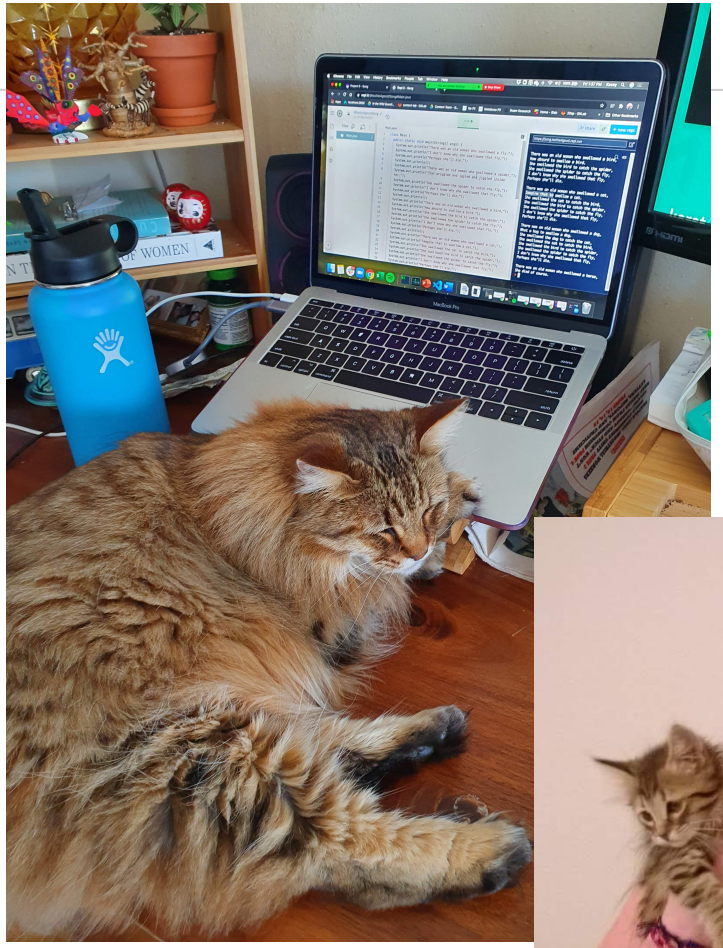
Software Engineer @ Karat

Kasey has to go to her “real job” after this

High School Teacher @ Franklin High

champk@cs.washington.edu

@techie4good



Iron Fist No Mercy
“Mercy” for short

Course Overview

Course Goals

- Design data structures and algorithms by implementing and maintaining invariants.
- Analyze the runtime and design values of data structures and algorithms.
- Critique the application of data structures and algorithms towards complex problems.
- Prepare for technical interviews

Course Topics

- Data Structures and ADTs: lists, stacks, queues, sets, dictionaries, arrays, linked lists, trees, hash tables, priority queues, binary heaps and disjoint sets
- Algorithm analysis: Big O Notation, asymptotic analysis, P and NP complexity
- Sorting algorithms: selection, insertion, merge, quick, more…
- Graphs and graph algorithms: graph search, shortest path, minimum spanning trees

Course Components

Learning Components

- Lectures
  - Recorded
  - Please come hang out with us
- Lecture Participation Polls
  - Graded on participation NOT correctness
- Exercises
  - Sets of conceptual problems distributed via Gradescope
- Projects
  - Programming assignments distributed via GitLab
- "Exams"
  - 2 this quarter
  - "Test like questions" without time limit
  - Groups, notes, internet searches allowed, but no staff help
- Office Hours
  - Please come hang out with us!

Course Tools

- Class webpage
  - Central location for all information
- Course canvas
  - Gradebook
- Zoom
  - Lectures
- Poll Everywhere
  - Lecture participation
- Gradescope
  - Exercise distribution and submission
- Discussion board
  - Get help
- GitLab
  - Project file distribution and submission
- Anonymous Feedback Tool
  - Tell us how it's going

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

Discussion Board

- 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 ☺

Course Policies

Turn In Policies

- 7 late days per student
  - use for projects or exercises
  - use up to 3 per assignment unless you speak to Kasey
- Assignments
  - Solo or groups of 2
  - Projects out/in on Wednesdays
  - Exercises out/in on Fridays
  - after all late days used up -5% for each 24-hour period turned in late
- Exams
  - Solo or groups of 2
  - Open note/open book, no staff help
  - Will have 1 week to turn in
    - **No late exams accepted**
- Participation Extra Credit
  - Poll everywhere open at start of lecture
  - Due before start of next lecture
  - **No late polls will be accepted**

Grade Breakdown

- Programming Projects (40%)
- Written Exercises (30%)
- Exam I (15%)
- Exam II (15%)
- Participation (EC round up to 0.05)

Grade to GPA Minimums

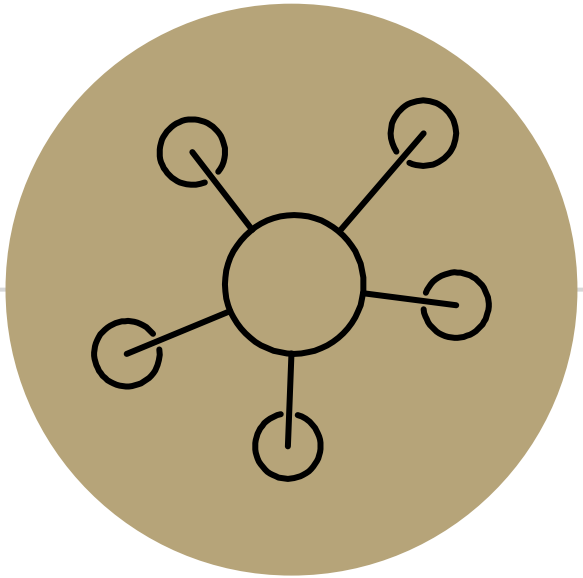
- 95% -> 3.5
- 85% -> 3.0
- 75% -> 2.5
- 65% -> 2.0
- 50% -> 0.7

Academic Misconduct

- Don't share your code
- Don't look at others code
- Don't "step by step"
- DO talk to one another about concepts and approaches
- DO look things up on the internet
- No posting code on discussion board or ANYWHERE online
- We do run MOSS

Accommodations and Extenuating Circumstances

- Make sure you get the support you are entitled to via DRS
  - If you're having issues with DRS system reach out to Kasey
- When in doubt, reach out!



Questions?

Clarification on syllabus, General complaining/moaning

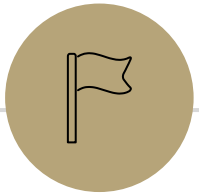
What is this class about?

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

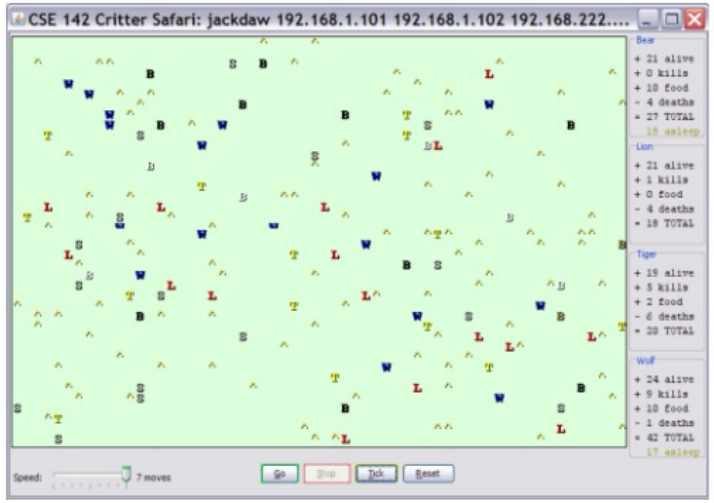
- A series of precise instructions to produce to a specific outcome
- Examples from CSE 14X: binary search, merge sort, recursive backtracking

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)
```



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)

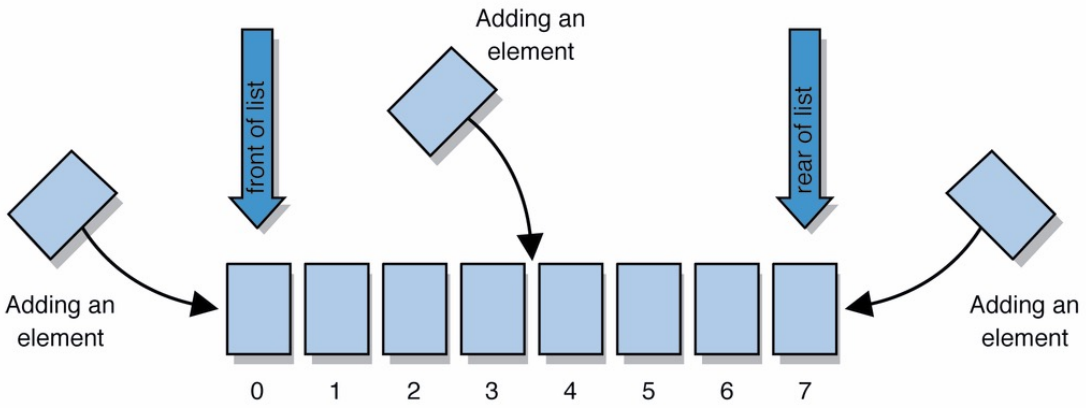
Abstract Data Types (ADT)

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

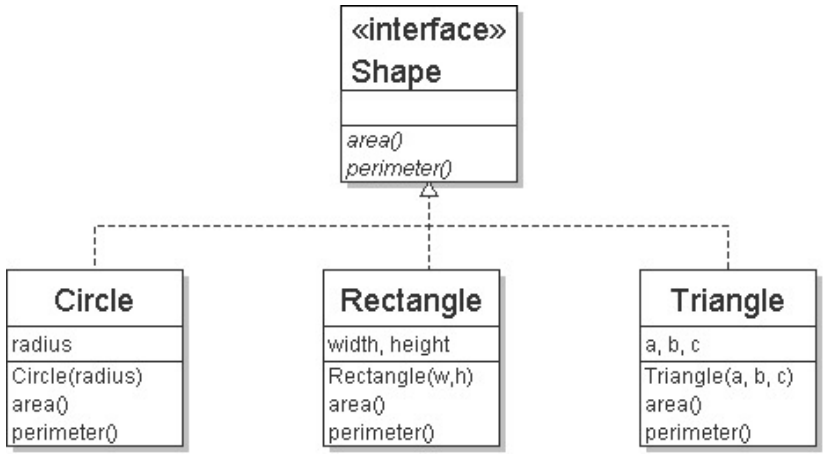
interface: A construct in Java that defines a set of methods that a class promises to implement

- 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

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 size defined as the number of elements in the list

```
List<String> names = new ArrayList<>();
names.add("Anish");
names.add("Amanda");
names.add(0, "Brian");
```

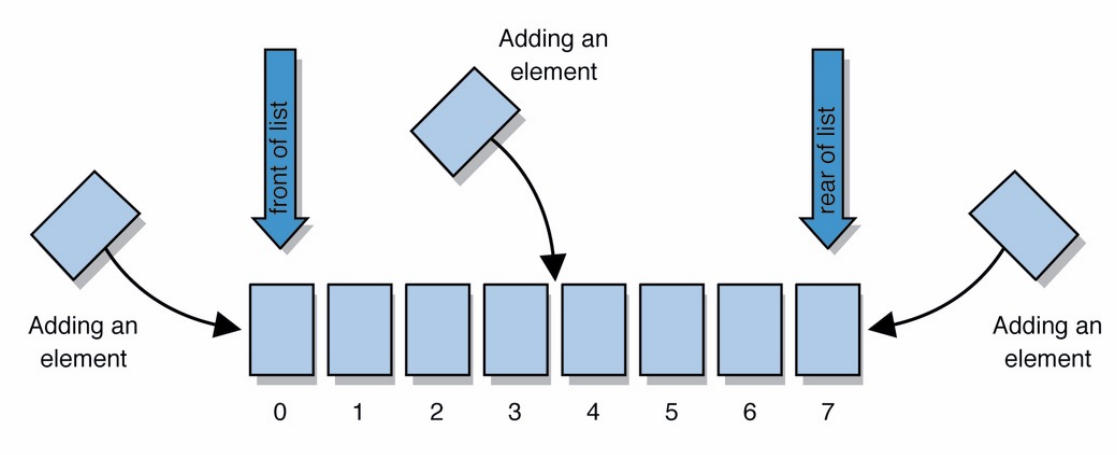
Case Study: The List ADT

list: a collection storing an ordered sequence of elements.

- Each item is accessible by an index.
- A list has a size defined as the number of elements in the list

Expected Behavior:

- **get(index):** returns the item at the given index
- **set(value, index):** sets the item at the given index to the given value
- **append(value):** adds the given item to the end of the list
- **insert(value, index):** insert the given item at the given index maintaining order
- **delete(index):** removes the item at the given index maintaining order
- **size():** returns the number of elements in the list



Case Study: List Implementations

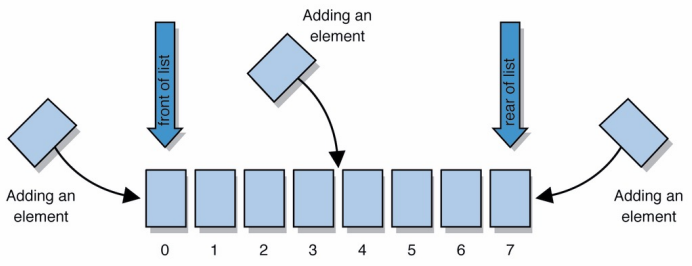
List ADT

state

Set of ordered items
Count of items

behavior

get(index) return item at index
set(item, index) replace item at index
append(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

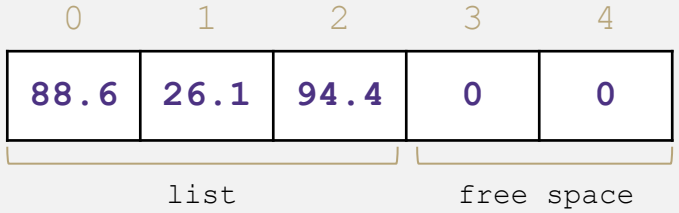
ArrayList<E>

state

data[]
size

behavior

get return data[index]
set data[index] = value
append 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

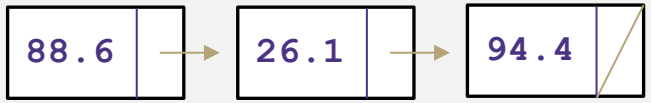
LinkedList<E>

state

Node front
size

behavior

get loop until index, return node's value
set loop until index, update node's value
append 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



Implementing ArrayList

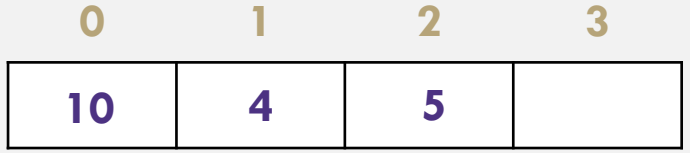
ArrayList<E>

state
- data[]
- size

behavior
- get return data[index]
- set data[index] = value
- append 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

`insert(element, index)` with shifting

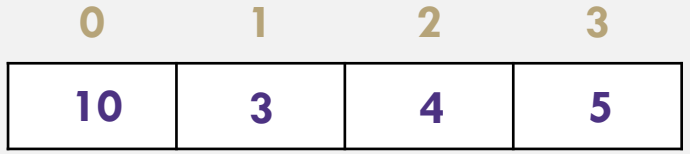
`insert(10, 0)`



`numberOfItems` = 4

`delete(index)` with shifting

`delete(0)`



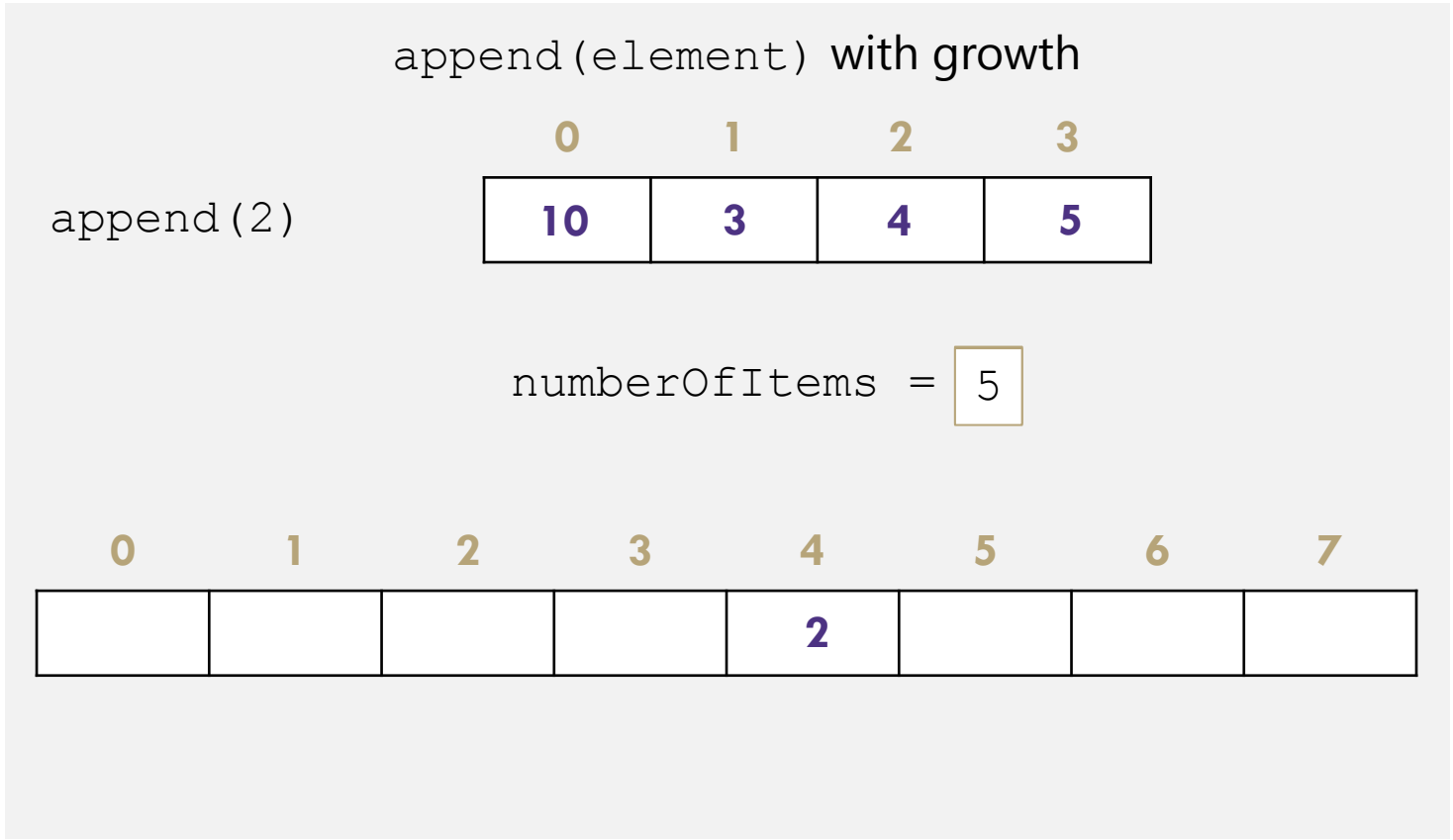
`numberOfItems` = 3

Implementing ArrayList

ArrayList<E>

state
- data[]
- size

behavior
- get return data[index]
- set data[index] = value
- append 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



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