Lecture 1: Intro to ADTs

CSE 373: Data Structures and Algorithms

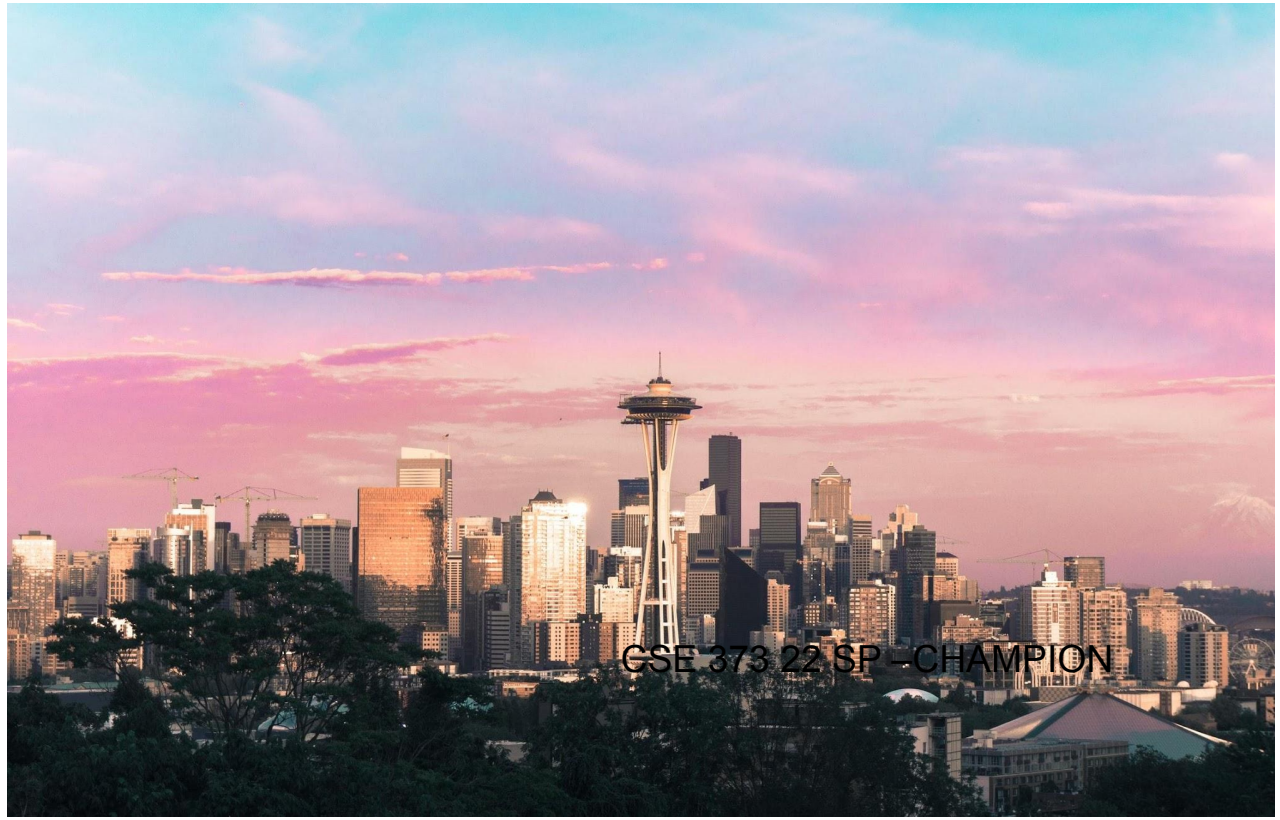
Agenda

Introductions
Syllabus
Dust off data structure cobwebs
Meet the ADT
Questions

Welcome!

This class will be complex and quick moving, but we hope to provide lots of support and opportunities to connect as a community.

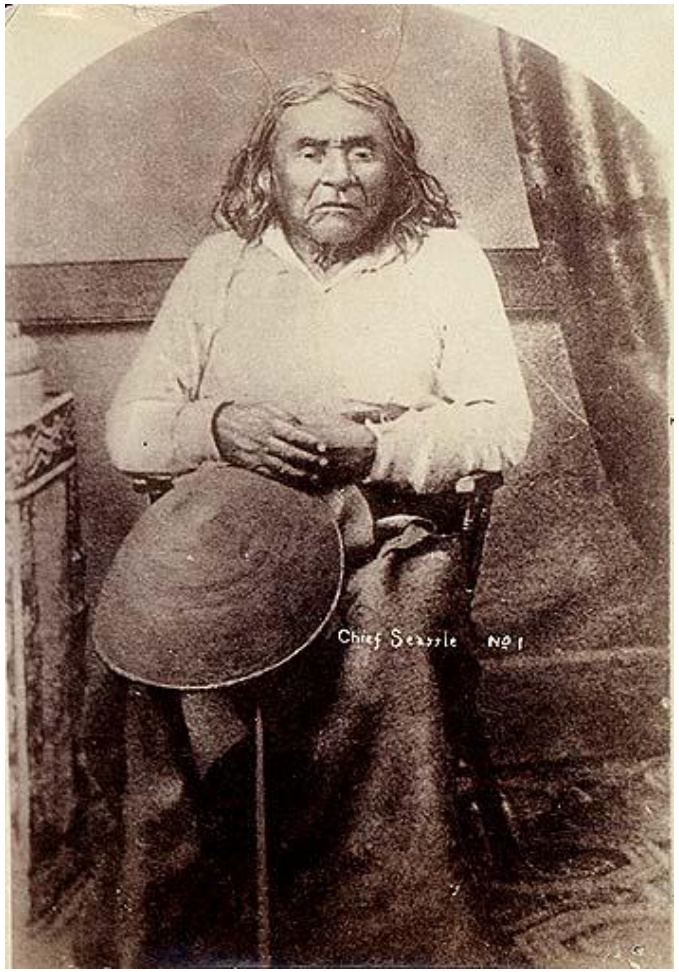
- Humans first, students second
- Patience, vulnerability, compassion
- Learn to build technology that supports all humans



Land Acknowledgement

"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/



Course Overview

Course Goals

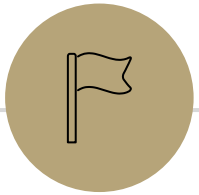
- Design data structures and algorithms by implementing and maintaining invariants.
- Analyze the runtime and design values of data structures and algorithms
- Critique the application of data structures and algorithms towards complex problems
- Prepare for technical interviews

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, Dynamic Programming optimization
- Sorting algorithms: selection, insertion, merge, quick, more…
- Graphs and graph algorithms: graph search, shortest path, minimum spanning trees

Waitlist / Overloads

- I have told CSE the more the merrier, but technically I have no control over these things :/
- Email ugrad-adviser@cs.washington.edu for all registration questions
- Many students move around, likely a spot will open
- Keep coming to lecture!



Introductions

Syllabus

Dust off data structure cobwebs

Meet the ADT

Questions

Hello! I am Kasey Champion (she/her)

Technical Program Manager @ Google

- Kasey has to go to her "real job" during the day (L)

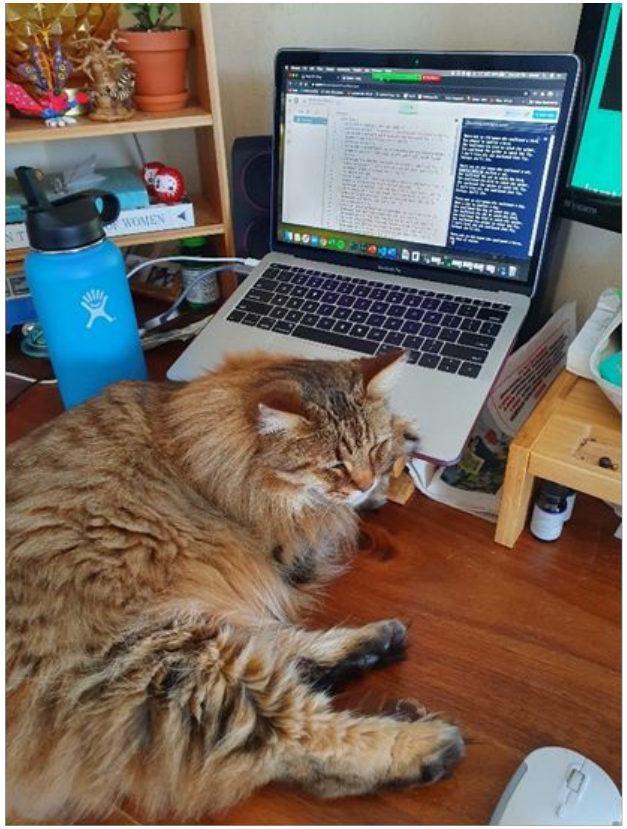
Previously:

- Technical Interview Content Team Lead @ Karat
- Software Engineer @ Microsoft

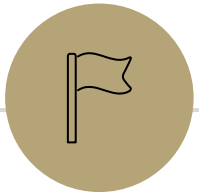
Electrical Engineering and Computing @ UW

champk@cs.washington.edu

bit.ly/kasey1on1s



If we Zoom, you will probably meet Mercy



Introductions

Syllabus

Dust Off Data Structure Cobwebs

Meet the ADT

Questions

Course Components

Learning Components

- Lectures
  - Recorded on Panopto, recordings on canvas
  - In-Person- Please come hang out with us
  - Each day we will cover 1 problem for weekly exercises
  - Lecture polls for EC participation
- Exercises
  - Sets of conceptual problems distributed via Gradescope
- Projects
  - Programming assignments distributed via GitLab
- Assessments
  - **4/28** In lecture Friday Week 5
  - **5/26** In lecture Friday Week 9
  - Resubmissions after you receive your grade can earn up to ½ missed points back
- Office Hours
  - TA and Kasey help to on exercises, programming assignments, course topics
  - We are here to help you move forward, not to debug or fix your code
  - 10 min per question limit

Course Components
Course Tools

- Class Webpage
  - cs.washington.edu/373
  - Central location for all information
- Course Canvas
  - Gradebook
  - Panopto lecture recordings
- Slido
- Lecture participation and questions
- EdStem
  - Course discussion board
  - Lecture companion lessons
- Gradescope
  - Exercise distribution and submission
- GitLab
  - Project file distribution and submission
- Anonymous Feedback Tool
  - Tell us how it's going

Course Policies

Turn In Policies

- 5 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 3
  - Projects out/in on Wednesdays
  - Exercises out/in on Mondays
  - after all late days used up -5% for each 24-hour period turned in late
- Exams
  - In person during lecture
  - Open note/open book, no staff help
  - Grades to be returned wednesday following
  - Resubmission due 1 week after grades posted
  - **No late exam resubmissions accepted**

Course Policies Cont.

- Grade Breakdown
  - Programming Projects (40%)
  - Written Exercises (30%)
  - Exam I (15%)
  - Exam II (15%)
  - Participation (EC round up to 0.05)
- 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 boards or **ANYWHERE** online
  - We do run MOSS
  - ChatGPT – it is ok to use this to **inspire** your solutions, but if you submit code you did not write and do not understand that is academic misconduct
- Accomodations and Extenuating Circumstance
  - 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!

A Note About Transitioning Back to In-Person

We are all figuring this out as we go!

Section

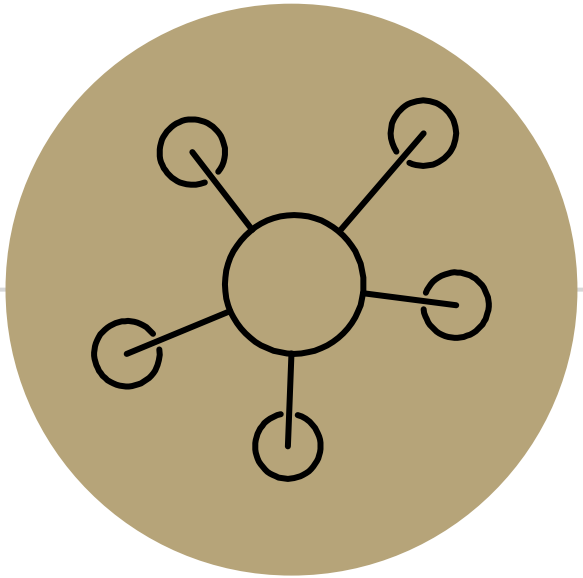
- Similar to lecture
- Please be prepared to work with other students
- Discuss in small groups when working on problems

Office Hours

- Both online and in-person meetings, logistics coming later this week
- Feel free to use this to meet and engage with one another (but make sure you are not sharing code, only ideas!)
- Please be prepared to share your screen
- Turn on mic

Let us know what works!

- Share what you've seen elsewhere
- Use the anonymous feedback form
- Always happy to take suggestions / feedback ☺️



Questions?

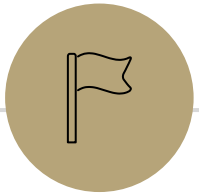
Clarification on syllabus, General complaining/moaning

What is this class about?

CSE 142 – INTRO TO PROGRAMMING	CSE 143 – OBJECT ORIENTED PROGRAMMING	CSE 373 – DATA STRUCTURES AND ALGORITHMS
– Java syntax – Methods – Variables – Parameters & Returns – File IO – Scanner – Arrays	– Classes and Interfaces – Methods, variables and conditionals – Loops and recursion – Linked lists and binary trees – Sorting and Searching – O(n) analysis	– Design decisions – Design analysis – Implementations of data structures – Debugging and testing – Abstract Data Types – Code Modeling – Complexity Analysis – Software Engineering Practices
Vocab	**Grammar**	**Creative Writing**

Introductions

Syllabus



Dust off Data Structure Cobwebs

Meet the ADT

Questions

Basic Definitions

Data Structure

- A way of organizing and storing data
- Examples from CSE 14X/12X: arrays, linked lists, stacks, queues, trees

Algorithm

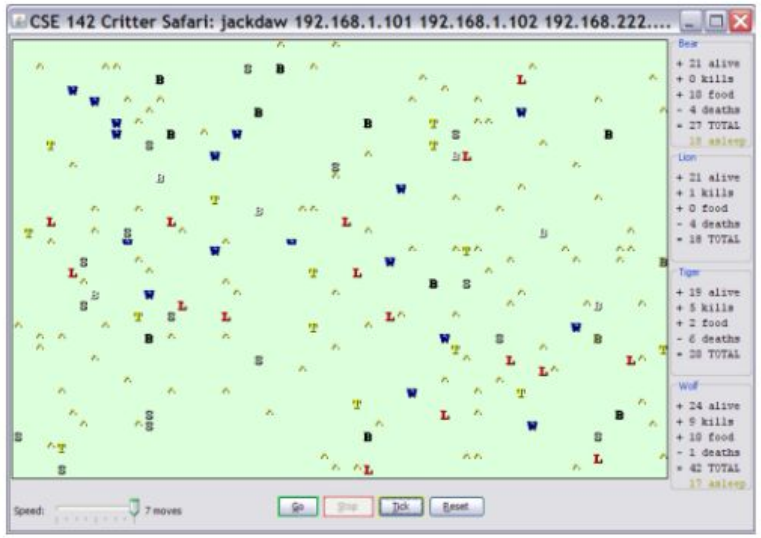
- A series of precise instructions to produce to a specific outcome
- Examples from CSE 14X/12X: binary search, merge sort, recursive backtracking

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

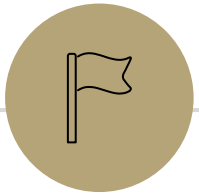
1. Ant

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)

Introductions

Syllabus

Dust off Data Structure Cobwebs



Meet the ADT

Questions

Abstract Data Types (ADT)

Abstract Data Type

- A type of organization of data that we define through its behavior and operations
  - Defines the input and outputs, not the implementations

Invented in 1974 by Barbara Liskov

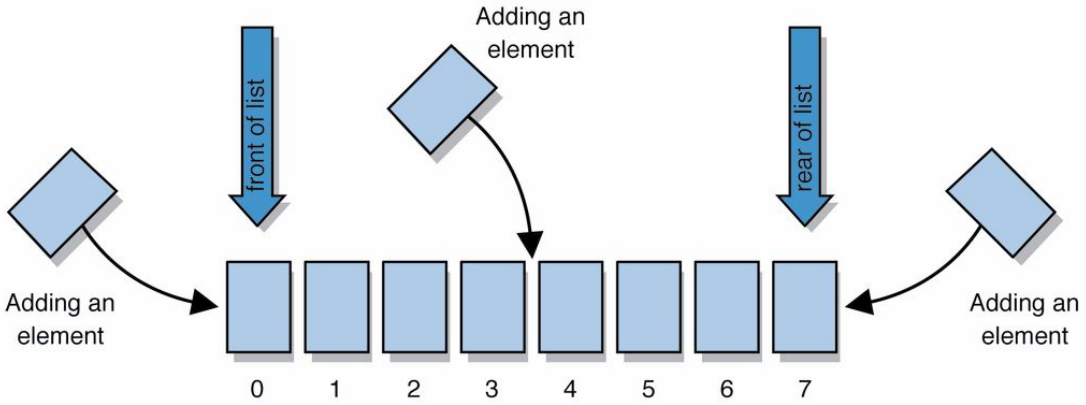
> *What we desire from an abstraction is a mechanism which permits the expression of relevant details and the suppression of irrelevant details. In the case of programming, the use which may be made of an abstraction is relevant; the way in which the abstraction is implemented is irrelevant. — Barbara Liskov*

Source Article

Abstract Data Types (ADT): List Example

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
- the ADT of a list can be implemented many ways through different Data Structures
  - 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

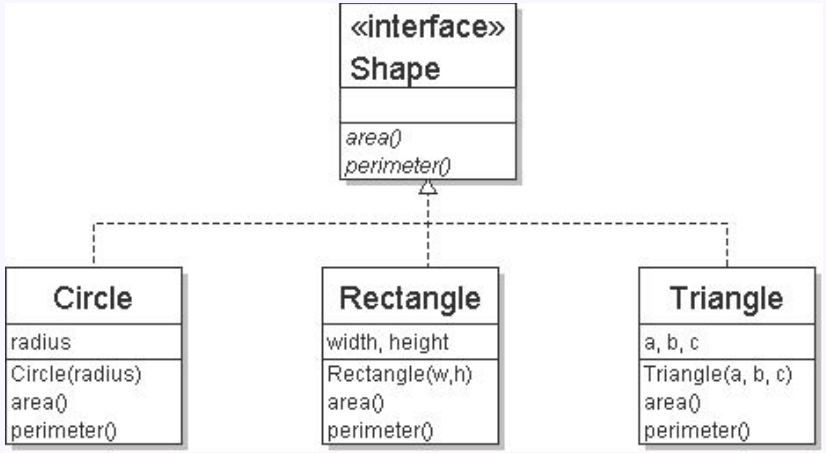
```
public interface name {
    public type name(type name, …, type name );
    public type name(type name, …, type name );
    …
}
```

- 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/certifications:
  - "I'm 'certified' as a Shape, because I implement the Shape interface. This assures you I know **how** to compute my area and perimeter."
  - "I'm 'certified' as a CPA accountant. This assures you I know **how** to do taxes, audits, and consulting."

Example

```
// Describes features common to all
// shapes.

public interface Shape {
      public double area();
      public double perimeter();
}
```



Review: Interfaces: List Example

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

In terms of ADTs, interfaces help us make sure that our implementations of that ADT are doing what they need to

For example, we could define an interface for the ADT `List<E>` and any class that implements it must have implementations for all of the defined methods

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …

    // many more methods
}
```

note: this is not how the `List<E>` interface actually looks, as in reality it extends another interface

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 and Data Structures: (Loose) Analogy

Abstract Data Type (ADT)

Mode of Transportation
Must be able to move
Must be able to be steered

Data Structures

Car	**Airplane**	**Bike**
Tires	Engines/wings	Wheels
Steering wheel	Control column	Handlebars

ADTs and Data Structures: List Example

List – Abstract Data Type (ADT)

```
// Describes features common to all lists.

public interface List<E> {
    public E get(int index);
    public void set(E element, int index);
    public void append(E element);
    public E remove(int index);
    …
}
```

ArrayIntList – Data Structure

```
public class ArrayIntList extends List<E>{
    private int[] list;
    private int size;

    public ArrayIntList(){
        //initialize fields
    }

    public int get(int index){
        return list[index];
    }

    public void set(E element, int index){
        list[index] = element;
    }
    …
}
```

ADTs we'll discuss this quarter

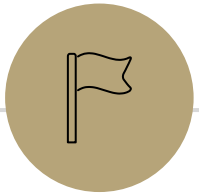
- List: an ordered sequence of elements
- Set: an unordered collection of elements
- Map: a collection of "keys" and associated "values"
- Stack: a sequence of elements that can only go in or out from one end
- Queue: a sequence of elements that go in one end and exit the other
- Priority Queue: a sequence of elements that is ordered by "priority"
- Graph: a collection of points/vertices and edges between points
- Disjoint Set: a collection of sets of elements with no overlap

Introductions

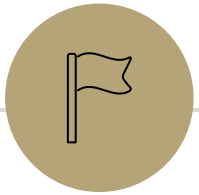
Syllabus

Data Structures and Algorithms

Meet the ADT



Questions?



That’s all!