Lecture 1: CSE 373

Data Structures and Algorithms

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Whitaker Brand, Stuart Reges, Zora Fung, Justin Hsia, and many others for sample slides and materials ...

Agenda

-Introductions
-Administrative stuff
-Class overview
-Review some 143 concepts
-Meet the ADT

Waitlist

- No Overloads
- Many students move around, likely a spot will open
- Email cse373@cs.washington.edu for all registration requests/questions

Hello!

Shrirang (Shri) Mare

Postdoc in CSE, Security & Privacy
Before UW: PhD at Dartmouth, Software Developer at IBM
shri@cs.washington.edu
CSE 360
Office Hours: Wednesdays 3:30-6:30

Course Goals

At the end of this class, you should be able to…

- Implement your own data structures
- Figure out which data structure AND implementation is best to solve a problem
- Write tests to be confident that your implementation are correct
- Work collaboratively with others on code
- Use with professional software engineering tools

Tell me your course goals: Pre-Course Survey

Communication

Website

- Schedule, course calendar, policies, material, assignments, etc.

Canvas

- Grades will be posted here

Google Discussion Board

- Announcements made here
- Ask and answer questions – staff will monitor and contribute

Office Hours

- Spread throughout the week, by appointment

Email

- To staff: cse373-staff@cs.washington.edu
- To individuals: Include “[CSE-373]” in your subject

Anonymous feedback

- Comments about anything related to the course where you would feel better not attaching your name

Grade Break Down

Homework (65%)
- Projects (50%)
  - Partners encouraged
- Written Assignments (15%)
  - Must be individual

Exams (35%)
- Midterm Exam – Friday Nov 2 in class (15%)
- Final Exam – Tuesday Dec 11 (20%)

Deadlines and Student Conduct

Late policies

- 3 late day “tokens” for the quarter, max 2 per submission
- For pair projects, both partners will need to use their late day(s)
- Need to get things done on time – difficult to catch up!

Academic Integrity

- Name your collaborators
- Credit your sources
- I will trust you implicitly and will follow up if that trust is violated
- In short: don’t attempt to gain credit for something you didn’t do and don’t help others do so either
- This does ***not*** mean suffer in silence – can still learn from the course staff and peers

Class Style

Please come to lecture and participate

- Collaboration
- Demos
- Participate:
  - Answer questions – don't worry about it being right or wrong
  - Ask questions! Point out mistakes!

Sections

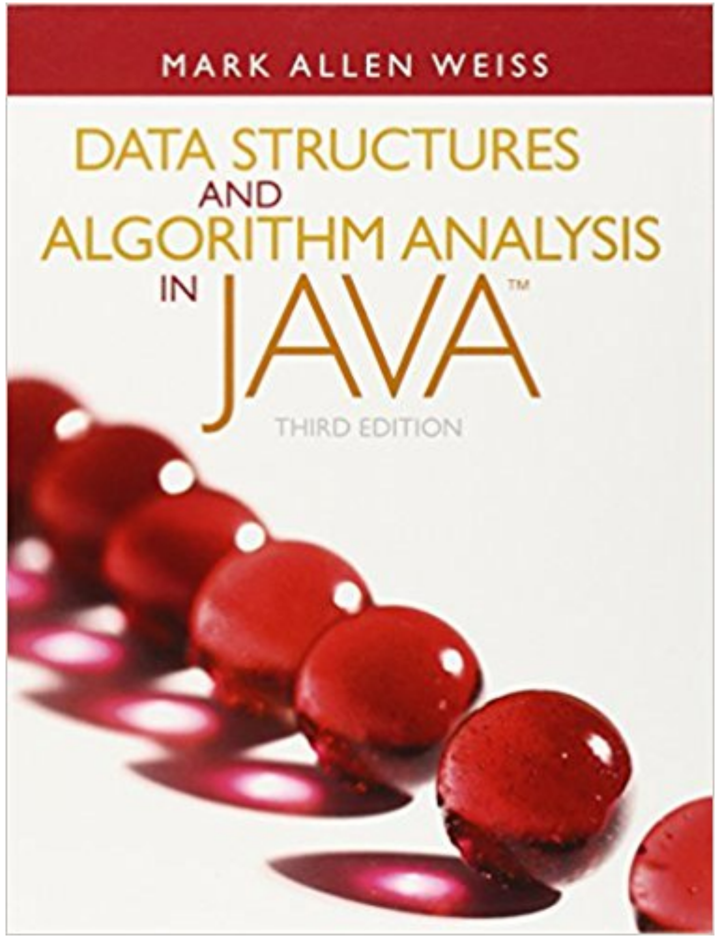
- TAs = heroes
- Practice problems
- Sections start next week

Getting help

- The internets
- Each other
- Office hours and Sections

Textbook

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



Hooked on Gadgets

Gadgets reduce focus and learning

- Bursts of info (*e.g.* emails, IMs, etc.) are *addictive*
- Heavy multitaskers have more trouble focusing and shutting out irrelevant information
  - http://www.npr.org/2016/04/17/474525392/attention-students-put-your-laptops-away
- Seriously, you will learn more if you use **paper** instead!!!

Non-disruptive use okay

- NO audio allowed (mute phones & computers)
- Stick to side and back seats
- Stop/move if asked by fellow student

Questions?

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-base development

Data Structures and Algorithms

What are they anyway?

Basic Definitions

Data Structure

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

Algorithm

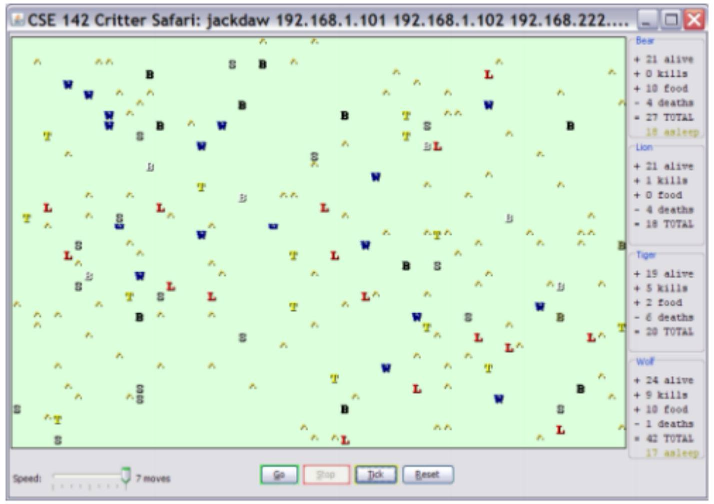
- A series of precise instructions used to perform a task
- Examples from CSE 14X: binary search, merge sort, recursive backtracking

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
  18 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)

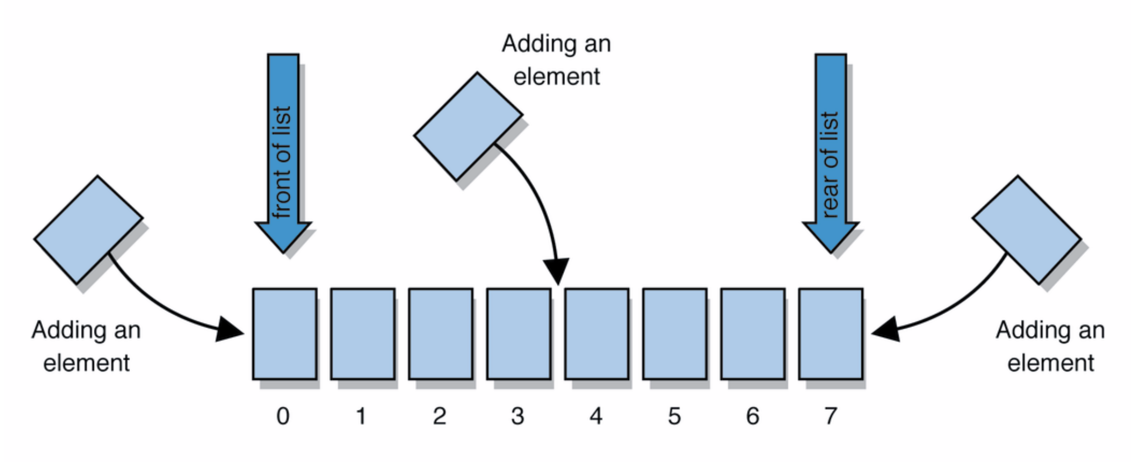
Abstract Data types

- A definition for expected operations and behavior

Start with the operations you want to do then define how those operations will play out on whatever data is being stored

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

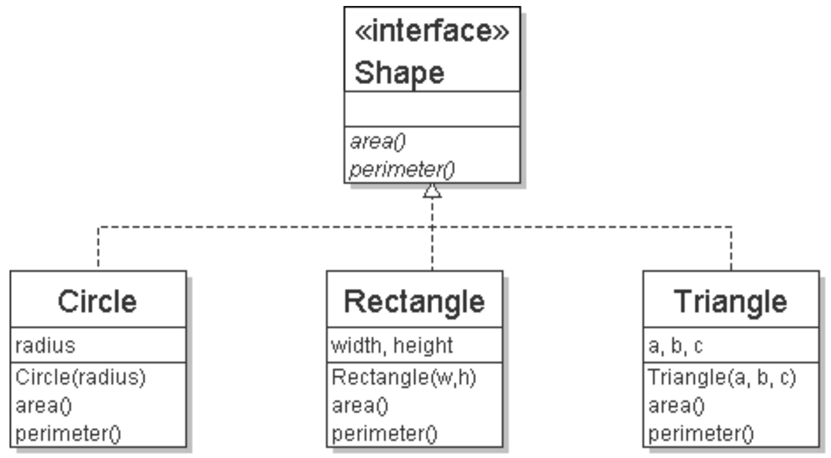
interface: A list of methods that a class promises to implement.

- 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!

You used:

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

List of ADTs

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

Case Study: The List ADT

list: stores an ordered sequence of information.

- Each item is accessible by an index.
- Lists have a variable size as items can be added and removed

Supported Operations:

- **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

- ArrayList

- Linked List