**CSE 332 Data Abstractions:**

*Introduction and ADTs*

Kate Deibel

Summer 2012
Welcome!

We have 9 weeks to learn fundamental data structures and algorithms for organizing and processing information

- Classic data structures and algorithms: queues, trees, graphs, sorting, etc.
- Rigorously analyze their efficiency
- Determine when to use them
- Parallelism and concurrency (!)
Today in Class

- Course mechanics
- What this course is about
  - And how it fits into the CSE curriculum
- What is an ADT?
- Review of Stacks and Queues
- Mystery Topics!? 
Concise to-do list

In next 48 hours, you should:

- Adjust class email-list settings
- Do homework 0 (worth 5 bonus pts)
- Read all course policies
- Read/skim Chapters 1 & 3 of Weiss book
  - Relevant to Project 1, due next week
  - Will start Chapter 2 on Wednesday
Socket wrench... scalpel... snarky comments...

COURSE MECHANICS
Instructor: Kate Deibel

- PhD in CSE (2011), University of Washington
- Research: Digital literacies, Educational Technologies, Assistive technologies, Disability and education
- Office: CSE 210
- Hours: TBD or drop-by
- E-mail: deibel@cs or @uw

Not me but my cute calico Susie
Teaching Assistant: David Swanson

- Let's let him introduce himself...
- E-mail: swansond@cs

Not David but Susie again. Isn't she cute?
D-E-I-B-E-L

- Pronunciation: DIE-BULL

- Spelling: Decibel minus the ‘c’
When in doubt...

- Consult the course webpage
  
  http://www.cs.washington.edu/education/courses/cse332/12su/

Or, if you want the quicker URL:

  http://www.cs.washington.edu/332
Communication

- Course email list: cse332a_su12@u
  - You are already subscribed (your @uw e-mail)
  - You must get announcements sent there
  - Fairly low traffic

- Course staff: cse332-staff@cs or Kate's and David's individual emails

- Discussion board
  - For appropriate discussions; TAs will monitor
  - Optional but can be enlightening

- Anonymous feedback link
  - If you don’t tell me (good or bad), I don’t know
Course meetings

- Lecture (Kate)
  - Materials posted usually before class (95% guarantee) to aid your note-taking
  - Lectures focus on key ideas & proofs
  - Some interactive problem-solving

- Section (David)
  - Often focus on software (Java features, programming tools, project/HW issues)
  - Reinforce key issues from lecture
  - Answer homework questions, etc.
  - An important part of the course (not optional)
NOTICE!!!

- Locations for one or more quiz sections will likely change
  - Goal is to have both in the same room or at least the same building
  - Will announce over course e-mail list before Thursday
  - Website will update when we know
Office Hours

- David's Office Hours
  - TBD but will students for time
- Kate's Office Hours
  - TBD after David's are set
  - I frequently hold open-door hours:
    
    *If my door is open, come on in!*
Course materials

- Textbook: Weiss 3rd Edition in Java
  - Good read, but only responsible for lecture/section/hw topics
  - Will assign homework problems from it
  - 3rd edition improves on 2nd, but we’ll support the 2nd

- Core Java book: A good Java reference (there may be others)
  - Don’t struggle Googling for features you don’t understand
  - Same book recommended for CSE331

- Parallelism / concurrency units use a free notes written by Dan Grossman (linked on website)
Course Work

- 8 written/typed homeworks (25%)
  - Due at end of lecture the day it is due
  - No late homeworks accepted
- 3 programming projects (25%)
  - Projects have phases (parts)
  - First phase of Project 1 due next week (TBD)
  - Use Java (see this week’s section)
  - Two 24-hour late-days for the quarter
- Midterm Exam (20%)
- Final Exam (30%)
Collaboration & Academic Integrity

- Read the course policy very carefully to understand how you can and cannot get/provide help to/from others
- Be proactive and always explain (when you submit) any unconventional action on your part when it happens
Respect Policy

- If you respect me, I will respect you
- I am here to teach you and help you learn about data abstractions
- I make a promise to have good lectures, polished assignments, etc. on time and in good humor
- In return, you should be
  - Respectful in lab and lecture
  - Do not cheat
Academic Accommodations (formal)

To request personal academic accommodations due to a disability, please contact Disability Resources for Students: 448 Schmitz, 206-543-8924 (or 206-543-8925 for TTY).

If you have a letter from DRS indicating that you have a disability which requires academic accommodations, please present the letter to me so we can discuss how to meet your needs for this course.
Academic Accommodations (proper)

- My goal is for you to learn productively
- If you have problems, ask me or a TA
- Accommodations:
  - We are not mean
  - We understand that life happens beyond this class, this major, this university, ...
  - We can make reasonable accommodations for individual students
  - This offer is open for everyone
- Just talk to us...
Unsolicited Advice

- Get to class on time!
- Learn this stuff
  - You need it for so many later classes/jobs
  - Falling behind only makes more work for you
- Have fun
  - So much easier to be motivated and learn
  - Get used to my bad jokes
  - Yes, they really are that bad
  - If you don't laugh, they just get worse
It's not about teaching penguins to limbo...

**WHAT THIS CLASS IS ABOUT?**
Data Structures + Threads

- About 70% of the course is a “classic data-structures course”
  - Timeless, essential stuff
  - Core data structures and algorithms that underlie most software
  - How to analyze algorithms
- Plus a serious first treatment of programming with multiple threads
  - Parallelism: Use multiple processors
  - Concurrency: Access to shared resources
  - Connections to the classic material
Where 332 fits

- Most common pre-req for 400-level courses
- Essential stuff for many internships too!

June 18, 2012
CSE 332 Data Abstractions, Summer 2012
What 332 is about

- Deeply understand the basic structures used in all software
  - Understand the data structures and trade-offs
  - Analyze the algorithms that use them (math!)
  - Learn how to pick “the right thing for the job”
- Experience the purposes and headaches of multithreading
- Practice design, analysis, and implementation
  - The elegant interplay of “theory” and “engineering” at the core of computer science
Goals

- Be able to **make good design choices** as a developer, project manager, etc.
  - Reason in terms of the general abstractions that come up in all non-trivial software (and many non-software) systems
- Be able to **justify and communicate** your design decisions
Views on this course

- Prof. Steve Seitz (graphics):  
  - 100-level and some 300-level courses teach how to do stuff  
  - 332 teaches really cool ways to do stuff  
  - 400 level courses teach how to do really cool stuff  

- Prof. James Fogarty (HCI):  
  - Computers are fricking insane  
  - Raw power can enable bad solutions to many problems  
  - This course is about how to attack non-trivial problems where it actually matters how you solve them
Views on this course

- Prof. Dan Grossman (prog. langs.): Three years from now this course will seem like it was a waste of your time because you can’t imagine not “just knowing” every main concept in it
  - Key abstractions computer scientists and engineers use almost every day
  - A big piece of what separates us from others
My View on the Course

- This is the class where you begin to think like a computer scientist
  - You stop thinking in Java or C++ code
  - You start thinking that this is a hashtable problem, a linked list problem, etc.
  - You realize that little assumptions make big differences in performance
  - You realize there is no absolutely best solution for a problem
Data structures, ADTs, etc. (sorry, no weird joke here)

**TERMINOLOGY**
Data structures

[Often highly *non-obvious*] ways to organize information to enable *efficient* computation over that information

- Key goal of the next lecture is introducing *asymptotic analysis* to *precisely* and *generally* describe efficient use of time and space

A data structure supports certain *operations*, each with a:
- **Meaning**: what does the operation do/return
- **Performance**: how efficient is the operation

Examples:
- **List** with operations *insert* and *delete*
- **Stack** with operations *push* and *pop*
Trade-offs

- A data structure strives to provide many useful, efficient operations

- But there are unavoidable trade-offs:
  - Time performance vs. space usage
  - Getting one operation to be more efficient makes others less efficient
  - Generality vs. simplicity vs. performance

- That is why there are many data structures and educated CSEers internalize their main trade-offs and techniques
  - And recognize logarithmic < linear < quadratic < exponential
Terminology

- **Algorithm**
  - A high level, language-independent description of a step-by-step process

- **Abstract Data Type (ADT)**
  - Mathematical description of a “thing” with set of operations

- **Data structure**
  - A specific family of algorithms for implementing an ADT

- **Implementation** of a data structure
  - A specific implementation in a specific language on a specific machine (both matter!)
Example: Stacks

- The **Stack** ADT supports operations:
  - `isEmpty`: have there been the same number of pops as pushes
  - `push`: takes an item
  - `pop`: raises an error if `isEmpty`, else returns the most-recently pushed item not yet returned by a `pop`
  - ... (possibly more operations)

- A Stack **data structure** could use a linked-list or an array or something else, and associated **algorithms** for the operations

- One **implementation** is in the library `java.util.Stack`
The Stack is a Useful Abstraction

- It arises all the time in programming (e.g., see Weiss 3.6.3)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: $3 \ 4 \ + \ 5 \ *$
  - Clever: Infix $((3+4) \ * \ 5)$ to postfix conversion
- We can code up a reusable library
- We can communicate in high-level terms
  “Use a stack and push numbers, popping for operators...” rather than, “create a linked list and add a node when...”
The Queue ADT

- Operations
  - create
  - destroy
  - enqueue
  - dequeue
  - is_empty

- Just like a stack except:
  - Stack: LIFO (last-in-first-out)
  - Queue: FIFO (first-in-first-out)

- Just as useful and ubiquitous
Get in line right now for the best offers!

*LET'S MAKE A QUEUE DATA STRUCTURE!*
## Circular Array Queue Data Structure

<table>
<thead>
<tr>
<th>Q: 0</th>
<th>b c d e f</th>
<th>size - 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>front</td>
<td>back</td>
<td></td>
</tr>
</tbody>
</table>

### Code Snippets

```java
// Basic idea only!
enqueue(x) {
    Q[back] = x;
    back = (back + 1) % size
}
```

```java
// Basic idea only!
dequeue() {
    x = Q[front];
    front = (front + 1) % size;
    return x;
}
```

- What if queue is empty?
  - Enqueue?
  - Dequeue?
- What if array is full?
- How to test for empty?
- What is the complexity of the operations?
- Can you find the kth element in the queue?
Linked List Queue Data Structure

// Basic idea only!
enqueue(x) {
    back.next = new Node(x);
    back = back.next;
}

dequeue() {
    x = front.item;
    front = front.next;
    return x;
}

- What if queue is empty?
  - Enqueue?
  - Dequeue?

- Can list be full?
- How to test for empty?
- What is the complexity of the operations?
- Can you find the kth element in the queue?
Circular Array vs. Linked List

Array:
- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast
- Constant-time access to kth element
- For operation insertAtPosition, must shift all later elements
  - Not in Queue ADT

List:
- Always just enough space
- But more space per element
- Operations very simple / fast
- No constant-time access to kth element
- For operation insertAtPosition must traverse all earlier elements
  - Not in Queue ADT
The Stack ADT

Operations:
- create
- destroy
- push
- pop
- top
- is_empty

Can also be implemented with an array or a linked list
- This is Project 1!
- Like queues, type of elements is irrelevant
  - Ideal for Java’s generic types (section and Project 1B)
Conclusions

- Welcome again!
- This will be a fun class.
- Read Chapter 1-3 for Wednesday
  - Chapter 1 is about Java
  - Chapter 3 is what we talked about today
  - Chapter 2 is discussed on Wednesday