Stacks and Queues

There's a lot we can say about "One Dimensional data structures"
- Here's another way of looking at it.

What if we want to maintain a collection of items that we're both adding to and removing from, but we want to add and remove in a specific order?

A **STACK** is something where you add items, and the only item that you're allowed to remove is the last thing you added.
- Typically 3 operations: push(E e), E pop(), E peek()
- LIFO, FILO (Last in-first out, first in last out)
    - Personally, I always mix up these acronyms.

Example: Your function stack.
- You might be in a function f, then it calls function g. You can't start working on f again until you're done evaluating g! And so on and so forth. And everything is in main - your main function is at the very bottom of the stack.
- Intuitive Examples: You're in a conversation and you go off on a tangent. Then you want to close that tangent and come back to the original conversation. But - you're tangent may itself have a tangent and you need to retrace your steps.
- A dream within a dream within a dream

Show examples of push and pop operations in action

Stacks are pretty fundamental to computation.
- Again, the function stack
- Tracing you're location in a more complex data structure when you've followed multiple links (show a picture)
    - So in general, we're not in a doubly linked list, and we can't go back to where we came from, but if we maintain a stack then we can
- Markup languages (show them HTML tags)
- Programming languages in general (parentheses and brackets)
- So we'll be using stacks throughout the rest of your assignments, and in fact you are already using stacks.

A **QUEUE** is a data structure that has the opposite rule, but it's the rule that we follow more intuitively. First in, Last out. In other words, we Americans call it waiting your turn and first-come-first-served. The British just call it a queue.
- FIFO (first-in, first-out)
- Alternatively, first come, first served

So a queue has 2 operations:

- enqueue(E e)
- E dequeue()

Show examples of enqueue and dequeue in action.

Examples of when are queues used?

- Maybe you can have one system (thread) adding to a queue (work that needs to be done) and then another one doing the work sequentially.
- Data Pipeline - I need these jobs to run sequentially, so put them in a queue
- Web Servers + Messaging: Sometimes you have a server that does a job (maybe it handles database management or outbound communication) so you have one server added to its queue, and then that server can go back to what it's doing.
- Input (keyboard) stored in a queue. Need to process letters typed in the order they came in!

Implementing Stacks and Queues in memory:

- Both are 1-D data structures, and we know 2 ways to implement 1-D data structures
- Arrays and LinkedLists
- Draw a square with 4 quadrants here
    - What it does (ADT) Stack and Queue
    - How it does it (implementation) LinkedList and Array

	ADT = Stack	ADT = Queue
Implementation: Array	ArrayStack	ArrayQueue
Implementation: Linked List	LinkedStack	LinkedQueue

They are given in the goodrich book here:

ArrayStack:
https://github.com/rysharprules/Data-Structures-and-Algorithms-in-Java-6th-Edition/blob/master/src/dsa6/chapter_06/ArrayStack.java

LinkedStack:
https://github.com/rysharprules/Data-Structures-and-Algorithms-in-Java-6th-Edition/blob/master/src/dsa6/chapter_06/LinkedStack.java

ArrayQueue:
https://github.com/rysharprules/Data-Structures-and-Algorithms-in-Java-6th-Edition/blob/master/src/dsa6/chapter_06/ArrayQueue.java

LinkedQueue:
https://github.com/rysharprules/Data-Structures-and-Algorithms-in-Java-6th-Edition/blob/master/src/dsa6/chapter_06/LinkedQueue.java

THE MOST INTERESTING ONE

- That’s the array queue.
- We’re using a Circular Array to arrange the elements. When they get pushed onto the end of the allocated array, we start from position 0 again (assuming that those first items have already been dequeued)
- Illustrate how this works (book)