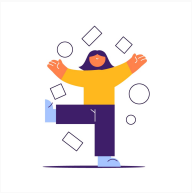
Lec 21: Queues

Sam McCauley

April 13, 2026

Admin



- Midterm Friday
  - Lab Wednesday will be a review session
  - Practice midterm posted; let me know if you find any issues
  - I'm planning a cleanup round on the posted code and slides later today; let me know if you find issues there as well
- Apply to be a TA!!
- Any questions?

Queues

Queues

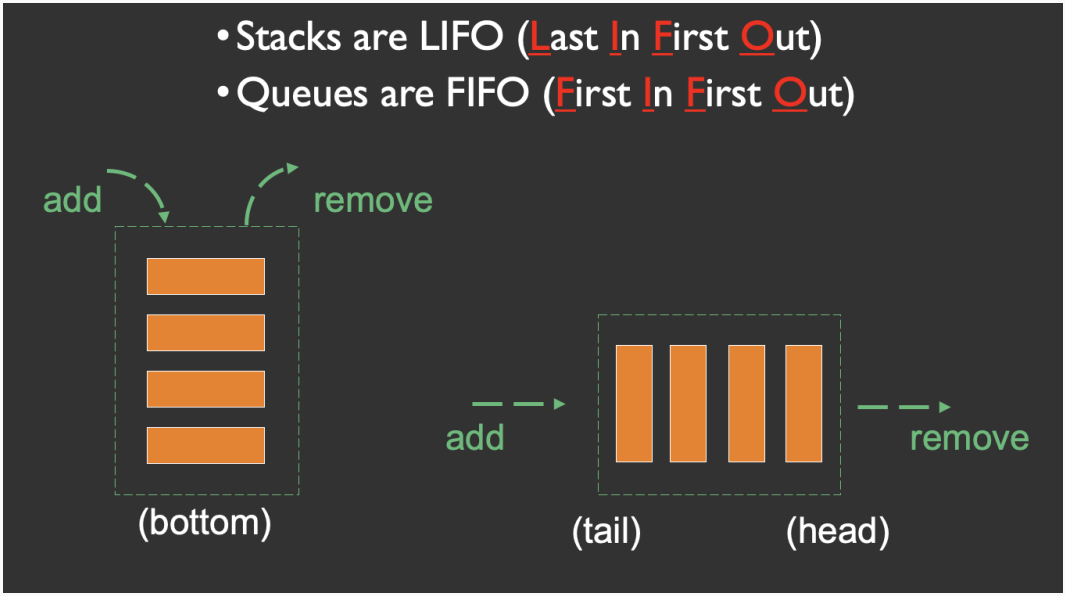
- Same idea as stacks: can only access one element
- Stacks are FILO (**F**irst **I**n **L**ast **O**ut)
- Queues are FIFO (**F**irst **I**n **F**irst **O**ut)

Queues

- Think of a queue as waiting in line
- The first to join the queue is the first to leave

Stacks vs Queues

- Stacks are LIFO (Last In First Out)
- Queues are FIFO (First In First Out)



Queue Operations

You *do* need to remember the `push`/`pop` stack vocabulary. You do *not* need to remember `enqueue`/`dequeue`.

- `enqueue()` or `add`: insert a value at the back of the queue
  - Think `addLast()`
- `dequeue()` or `remove`: remove and return the value from the front of the queue
  - Think `removeFirst()`
- (You can think `addFirst()`/`removeLast()` too.)
- `peek()`: access the first value of the queue without removing it

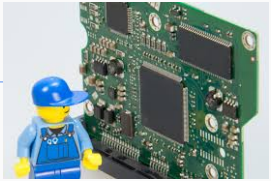
Queue Interface

```
1  public interface Queue<E> {
2      public void add(E item);
3      public E remove();
4      public E peek();
5  }
```

`Queue<E>` actually requires a few more methods which we won't discuss; they're all very similar to the above three.

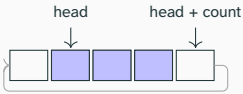
- Example: `element()` is the same as `peek()`, but throws an exception rather than returning `null` if the queue is empty.

How to Implement a Queue?



- What data structures can we use?
  - Array
  - ArrayList
  - Singly Linked List
  - Doubly Linked List
- In pairs: how should each be implemented? What is the performance of `enqueue` and `dequeue` for each?

Queue using an Array

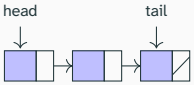


- Has a max number of elements it can store
- Keeps two `ints` in addition to the array: `head` and `count`
- Key idea: we *wrap around* the array as new items are enqueued and old items are dequeued
- Cost for `enqueue()`?
  - $O(1)$
- Cost for `dequeue()`?
  - $O(1)$

Queue using an ArrayList

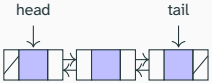
- To `enqueue` call `add(elem)`; to `dequeue` call `remove(0)`
- Time for `enqueue()`?
  - $O(1)$
- Time for `dequeue()`?
  - $O(n)$ (this is terrible!)
- Practice at home: what would happen if we instead added to the beginning, and removed from the end? Would performance improve?

Queue Using a Singly Linked List



- We want efficient `addLast` and `removeFirst`
- Singly linked lists have inefficient `addLast`
    - Side note: it's easy to modify so that we get $O(1)$ for both using singly linked nodes by adding a `tail` pointer. Let's do this on the board!

Queue Using a Doubly Linked List



- Let's consider a doubly linked list for the sake of discussion (only downside vs Singly Linked List: slightly wasteful for space)

- Time for `enqueue()`?
  - $O(1)$

- Time for `dequeue()`?
  - $O(1)$

Java Library Queue

- `Queue` is an *interface* in the Java library
- Several classes implement queue; main one is `LinkedList<E>`
  - Note that the Java Library `LinkedList<E>` is a *doubly* linked list
  - `add()` calls `addLast()` and `remove()` calls `removeFirst()` in `LinkedList<E>` so it's a queue!
- I think an interface is not the right choice here: it would make more sense to have `Queue<E>` be a class that inherits from `LinkedList<E>` (that's how `Stack<E>` works)
  - Not everything that implements `add()` and `remove()` is first-in-first-out
  - The Java designers appear to disagree with me on this

The List Interface

Printing Positive Numbers in a List

The following code prints all of the positive numbers in an `ArrayList<Integer>`:

```
1  public void printPositive(ArrayList<Integer> list) {
2      for(int i = 0; i < list.size(); i++) {
3          int item = list.get(i);
4          if(item > 0) {
5              System.out.println(item);
6          }
7      }
8  }
```

What happens if we instead want to print all of the positive numbers in a `LinkedList<Integer>`?

Printing Positive Numbers in a List

The following code prints all of the positive numbers in a `LinkedList<Integer>`:

```
1  public void printPositive(LinkedList<Integer> list) {
2      for(int i = 0; i < list.size(); i++) {
3          int item = list.get(i);
4          if(item > 0) {
5              System.out.println(item);
6          }
7      }
8  }
```

Literally nothing changed other than the type! Can we write a single method that can handle both data structures? What should we use?

Interfaces

- Interfaces help us out when we want a single method to handle different types of objects with the same functionality

- `ArrayList<E>` and `LinkedList<E>` both implement the `List<E>` interface

`List<E>` Interface Methods (partial list)

`add(E item)`	append `item` to the end
`add(int i, E item)`	insert `item` at index `i`
`get(int i)`	return element at index `i`
`set(int i, E item)`	replace element at index `i` with `item`
`remove(int i)`	remove and return element at index `i`
`size()`	return number of elements
`isEmpty()`	return `true` if the list has no elements
`contains(Object o)`	return `true` if `o` is in the list
`indexOf(Object o)`	return index of first occurrence of `o`
`clear()`	remove all elements

Interfaces and Inheritance

- A `Stack<E>` inherits from `ArrayList<E>`. Does that mean it also implements `List<E>`?

- Yes! If class B inherits from class A, then class B implements every method that class A implements

    - Makes sense—class B must have all of these methods implemented, since it has all of A's methods

Interfaces and Inheritance

- Interfaces can `extend` (inherit from) other interfaces
- Remember: an interface is just a list of required methods
- Works like you'd expect: "copies over" the other interface's methods
- To implement a subinterface, a class must implement every method in the subinterface, and all methods in the superinterface

Implementing Multiple Interfaces

- A class may implement multiple interfaces
    - It just means that it meets the requirement (has all required methods) for all of the listed interfaces
- A class may only extend *at most one* other class. You cannot inherit from multiple classes
- Let's look at how the Java library version of `ArrayList` is declared. It inherits from an abstract class, and implements several interfaces

Printing Positive Numbers in a List

The following code prints all of the positive numbers in any `List<Integer>`:

```
1  public void printPositive(List<Integer> list) {
2      for(int i = 0; i < list.size(); i++) {
3          int item = list.get(i);
4          if(item > 0) {
5              System.out.println(item);
6          }
7      }
8  }
```

What is the problem with this code? (It works! It just can be slow...)

Answer: this code takes $O(n^2)$ time for a linked list!

The Issue with Linked Lists

- Any method that access all items of a linked list using `get()` will take $O(n^2)$ time

- Problem: each time we call `get()` we start over at the beginning of the list