

LEC 04

CSE 122**Queues**

Questions during Class?
Raise hand or send here

sli.do #cse122

**BEFORE WE START*****Talk to your neighbors:***

*What is your favorite place to eat
on the Ave?*

Respond on sli.do!**Instructor:** Brett Wortzman

TAs:	Archit	Daniel	Naomi	Sreshta
	Blake P.	David	Neha	Suyash
	Blake R.	Hanna	Nicole	Trisha
	Cady	Isis	Rio	Wesley
	Carson	Jun	Rohan	Yash
	Cole	Landon	Shreyank	
	Connor	Michelle	Somya	

Music: 🎸 [CSE 122 26au Lecture Tunes](#) 🎵

Lecture Outline

- **Announcements** 
- Review: ADTs; Queues
- Programming Practice
- Commenting/Implementation Details

Announcements

- Programming Project 0 (P0) due tomorrow by 11:59pm
- Computer Quiz 0 now [available for scheduling](#)
 - Will run Friday 10/16 through Friday 10/23
 - See [policies and procedures](#) on Ed
- Paper Quiz 0 coming up on Tuesday, October 27 (in section)
 - More details soon, but like computer quizzes except on paper
- Brett's [Office Hours posted](#)
 - Mon 12:30-1:30, Thur 12:00-1:00 (except 11/5 and 12/10)

Lecture Outline

- Announcements
- **Review: ADTs; Queues** 
- Programming Practice
- Commenting/Implementation Details

(PCM) Abstract Data Types

- **Abstract Data Type (ADT):** A specification of a collection of data and the operations that can be performed on it.
 - Describes what a collection does, not how it does it (not implementation!)
 - Think of it as an ✨ idea ✨ of a data type
- We don't know exactly how a list or queue is implemented, and we don't need to!
 - Only need to understand high-level idea of what a collection does
 - **List:** ordered collection, access any element by index, add/remove anywhere
 - **Queue:** ordered collection, only add to back/remove from front

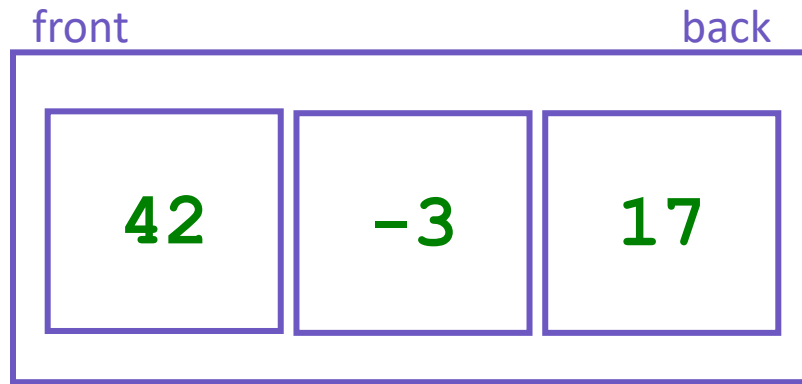
Wait, ADT? Interfaces?

- **Abstract Data Type (ADT):** A *description of the idea* of a data structure including what operations are available on it and how those operations should behave.
 - The concept of a list.
- **Interface:** Java construct that lets programmers *specify what methods a class should have*.
 - The `List` interface in Java.
- **Implementation:** *Concrete code* that meets the specified interface.
 - The `ArrayList` and `LinkedList` classes that implement the `List` interface.

(PCM) Queues

- Queue: Retrieves elements **in First-In, First-Out ("FIFO") order**
 - Client can only *add to the end of the queue*, and can only *examine/remove the front of the queue*.
 - Elements are stored in order of insertion but don't have indexes.
- Basic Queue operations:
 - **add** (enqueue): Add an element to the back.
 - **remove** (dequeue): Remove the front element.
 - **peek**: Examine the front element.
 - Java allows certain other operations, but *you should not use them in 122!*

(PCM) Programming with Queues



<code>add (value)</code>	places given value at back of queue
<code>remove ()</code>	removes value from front of queue and returns it; throws a <code>NoSuchElementException</code> if queue is empty
<code>peek ()</code>	returns front value from queue without removing it; returns <code>null</code> if queue is empty
<code>size ()</code>	returns number of elements in queue
<code>isEmpty ()</code>	returns <code>true</code> if queue has no elements

```
➔ Queue<Integer> q = new LinkedList<Integer> ();  
➔ q.add(42);  
➔ q.add(-3);  
➔ q.add(17);  
➔ System.out.println(q.remove());
```

 **IMPORTANT:** When constructing a queue you must use a new **LinkedList** object instead of a new `Queue` object. (More on that with Interfaces.)

Queues in Computer Science

- Operating systems:
 - Queue of print jobs to send to the printer
 - Queue of programs / processes to be run
 - Queue of network data packets to send
- Computer Architecture
 - Miss status/handling register (MSHR) queue
 - Instruction fetch queue
 - Issue queue
 - Instruction pipeline in general!
- Programming:
 - Modeling a line of customers or clients
 - Storing a queue of computations to be performed in order
- Real world examples:
 - People on an escalator or waiting in a line
 - Cars at a gas station (or on an assembly line)

But *Why?*

- Lists can do everything queues can and more.
- So why use a queue?
- Similar questions:
 - Why not always use `double` instead of `int`?
 - Why not always use `String` instead of `char`?
 - Why not always use `while` instead of `for`?
- Makes the intent clearer
- Avoids unintended operations
- Can have efficiency benefits

Lecture Outline

- Announcements
- Review: ADTs; Queues
- **Programming Practice** ◀
- Commenting/Implementation Details

Lecture Outline

- Announcements
- Review: ADTs; Queues
- Programming Practice
- **Commenting/Implementation Details** ◀

Commenting/Implementation Details

- In CSE 122, we use header comments as general *documentation*
 - Describing what the *client* needs to know about the program/method
- See the [Commenting Guide](#) for example and guidelines
- Should avoid providing *implementation details*
 - Things that are not relevant to the *client's* understanding
 - The “how” instead of the “what”
 - Note: this is not universal, but it is a 122 expectation

Implementation Details

```
// B: Uses a for-loop to go from 0 up to but not including numCount and
//     puts the running sums in an ArrayList.
//     For example, if numCount is 4, the list will contain [0, 1, 3, 6].
// R: sums - the list of sums being returned
// P: numCount - the number of terms to sum
public static List<Integer> sumNumbers(int numCount) {
    List<Integer> sums = new ArrayList<>();
    int total = 0;
    for (int i = 0; i < numCount; i++) {
        total += i;
        sums.add(total);
    }

    return sums;
}
```

Implementation Details

```
// B: Puts the running sums from 0 up to but not including numCount in a list.  
//     For example, if numCount is 4, the list will contain [0, 1, 3, 6].  
// R: the list of running sums  
// P: numCount - the number of terms to sum  
public static List<Integer> sumNumbers(int numCount) {  
    List<Integer> sums = new ArrayList<>();  
    int total = 0;  
    for (int i = 0; i < numCount; i++) {  
        total += i;  
        sums.add(total);  
    }  
  
    return sums;  
}
```