Present/Report on:

- What is your data structure?
    - Queues - A queue is an interface that extends the Collections interface. A queue follows a FIFO (First in, First Out) methodology using the .add() and .remove() methods. .add() allows one to insert an element at the end of a linear data structure, while the .remove() method removes an element from the start of a linear data structure. For a queue implementation, the .add() method is often referred to as Enqueue, and the .remove() method is called Dequeue.
    - Queues use **head** and **tail** elements to point to the first and last positions of the queue to aid in adding and removal of data.
- Are there different implementations? What are they?
    - Array, Stack, LinkedList, ArrayList
- What is your data structure useful for?
    - Where would you want to use it vs not want to use it?
        - You want to use a queue if you want to get elements out in the same order they entered the queue. Given that a queue follows that FIFO methodology, a queue data structure would be a better option for simulating a waiting line or assembly line in comparison to a stack.
        - It is possible to implement a queue using a stack, and vice versa
    - Are there certain implementations of your data structure that are more useful than others?
        - ArrayList - useful for dynamic sizing and shifting of data after head removal
        - LinkedList - also dynamically resizes, can easily move an item from one LinkedList queue to another using the existing LinkedList pointer
        - Array - Unlike an Arraylist or LinedList, this data structure would be useful if the number of items is fixed. It requires changing the size of the array whenever the array is filled past its original defined size.
        - Stack

Implementation

- What version of the data structure did you implement?
    - ArrayList
- How did you implement it?
    - We used an arraylist, primarily to handle dynamic sizing of the queue
        - (read: we are lazy and didn't want to have to write more code)
    - To keep track of the front and back of the queue, we used the private variables head and tail, respectively
    - We implemented each of the core functions of a queue, listed below
- What are the most basic methods needed to access elements stored?
    - The basic methods that we can use to access elements stored in the queue is with a .get() method, if we want to get a certain element. If we want to get the head of the queue, or the next element that will be incremented from the queue we can use the .peek() method.
- How do you go about adding or deleting elements?
    - To add an element, the .enqueue() method was used.
    - To remove an element, the .dequeue() method was used.

Queue Methods

Enqueue

- Check if queue is full (not needed for ArrayList)
- If not, increment tail and add data to new tail position

Dequeue

- Check if queue is empty
- If not, return data located at head, then increment head

Peek

- Return the next value to be removed (head) without removing it from the queue

Display

- Print the contents of the queue to the console

Poll

- Returns and removes the data at the head, or returns null if queue is empty