Data Structures and Algorithms

Lecture 4: Queues

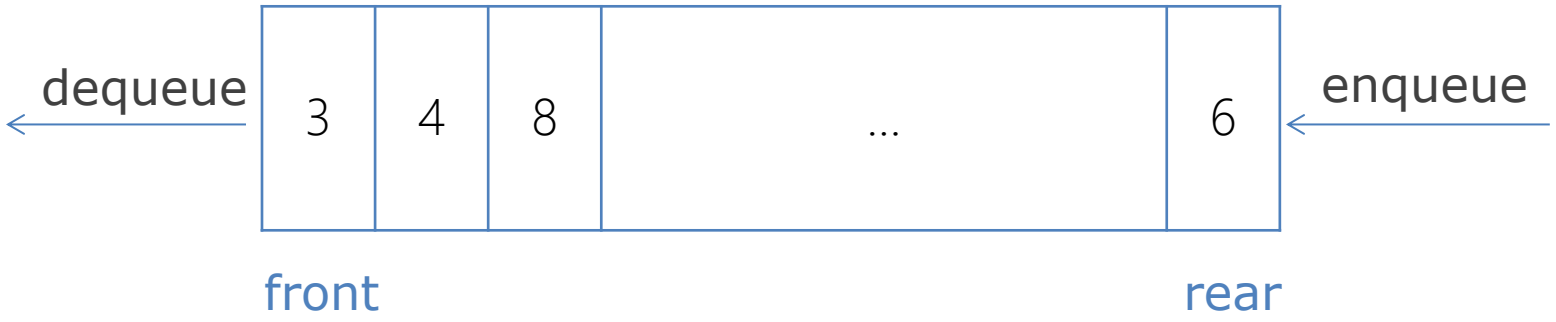
Department of Computer Science & Technology
United International College

Outline

- Queue ADT
- Basic operations of queue
  - enqueue, dequeue
- Applications of queue
- Implementation of queue
  - Array
  - Linked list

Queue ADT

- Like a stack, a *queue* is also a list.
- However, with a queue
  - insertion is done at one end
    - The rear
  - while deletion is performed at the other end.
    - The front



Queue Animation

- http://liveexample.pearsoncmg.com/liang/animation/animation.html
- Queues are known as FIFO (First In, First Out) lists.

Enqueue and Dequeue

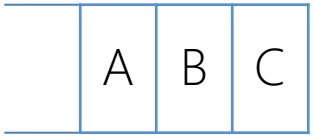
- Primary operations: Enqueue and Dequeue
- Enqueue
  - insert an element at the rear of the queue
- Dequeue
  - remove an element from the front of the queue

Enqueue and Dequeue

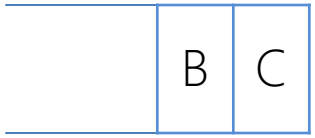
Empty queue

Enqueue(A)

Enqueue(B)



Enqueue(C)



Dequeue()

Dequeue()

Queue Applications

- Printer Queue
- Web Crawler
- System Buffer
- Any sequence maintained in a FIFO order

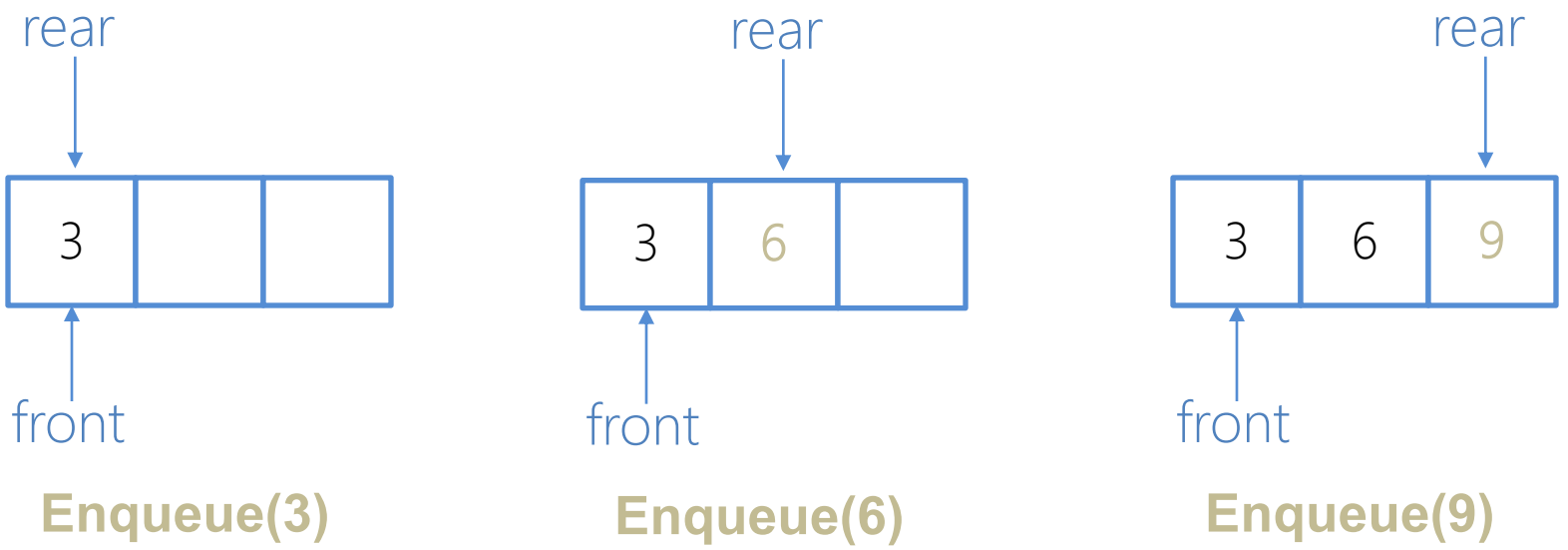
Implementation of Queue

- Recall the reason why we usually
  - implement a list using links?
  - implement a stack using array?

Topic		Array	Linked List
Efficiency	Enqueue		
	Dequeue		
space			

Queue Implementation Using Array

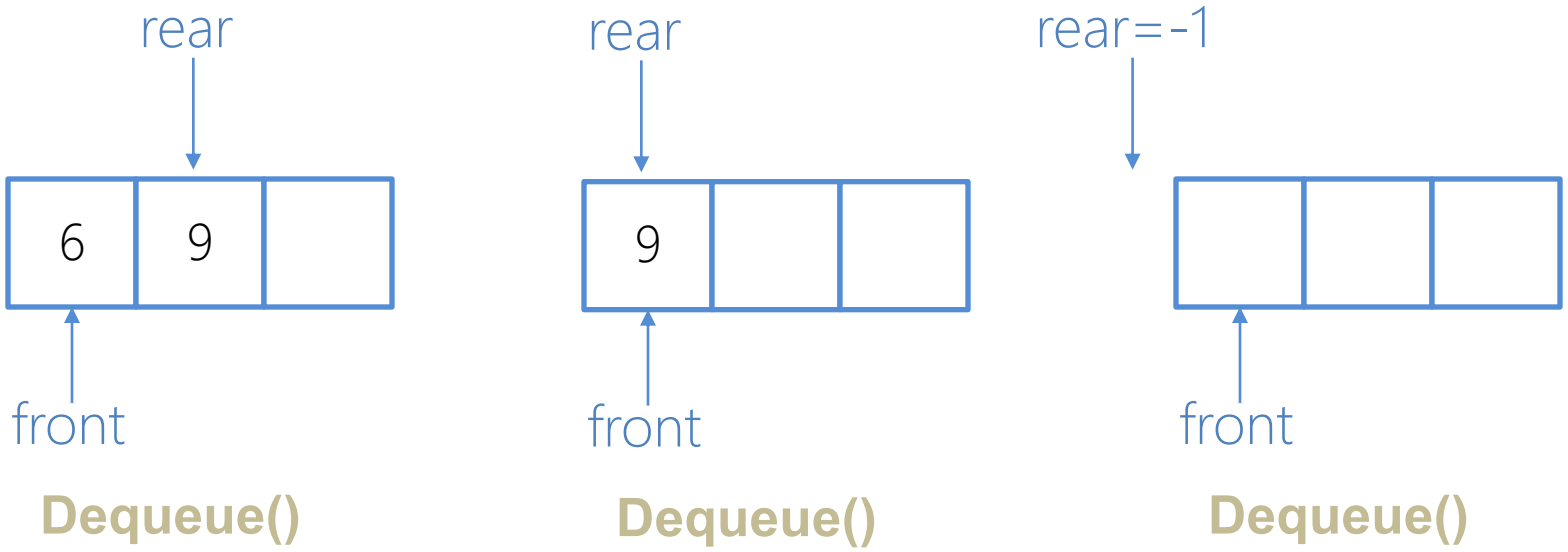
- There are several different algorithms to implement Enqueue and Dequeue
- Naive way: Enqueue
  - the front index is always fixed
  - the rear index moves forward in the array.




10
10
10

Queue Implementation Using Array

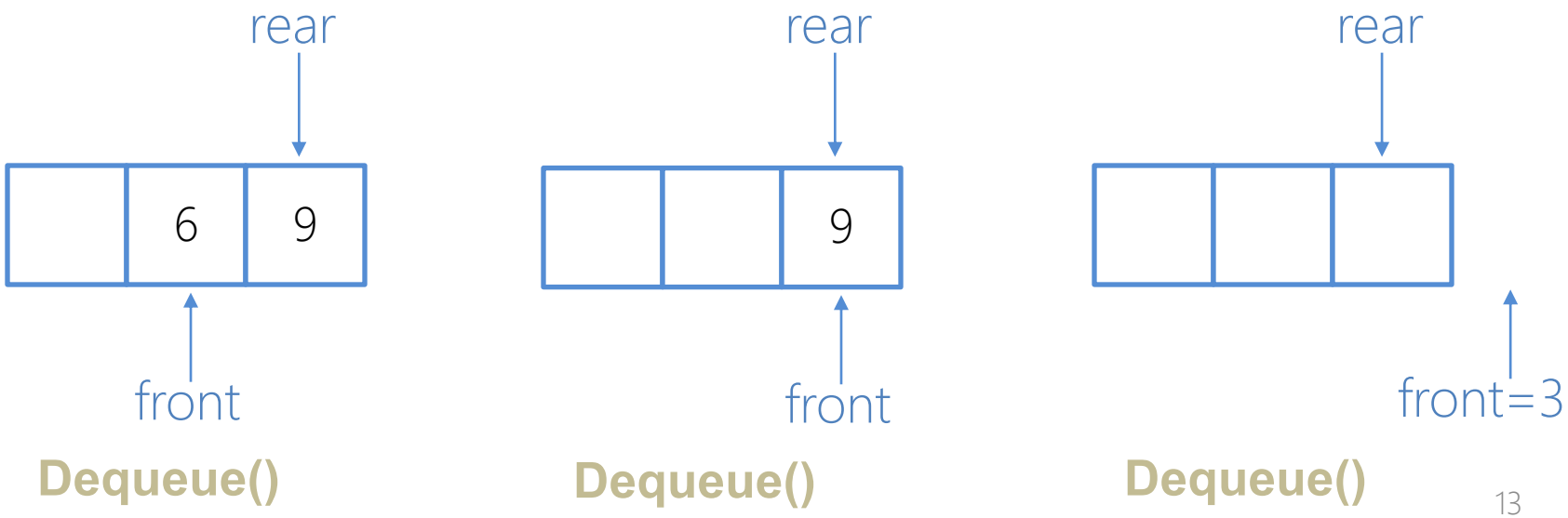
- Naive way: Dequeue
  - the front index is fixed
  - the element at the front the queue is removed
  - Move all the elements after it by one position.



NO GOOD

A Better Array Implementation

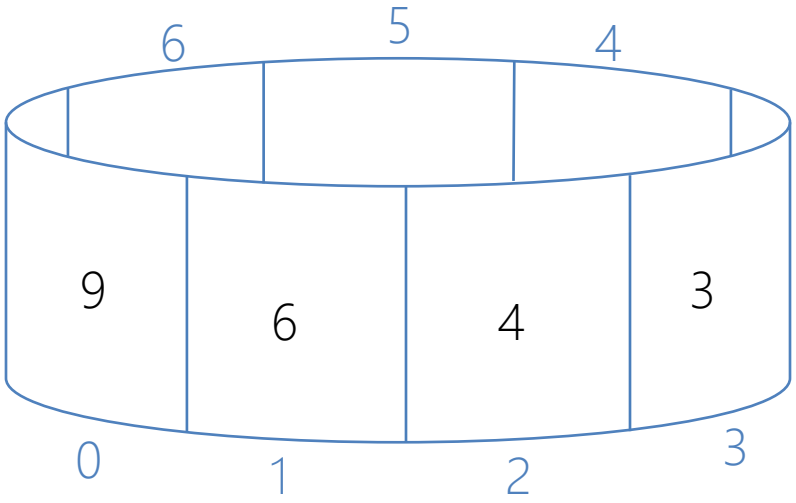
- When an item is enqueued, the rear index moves forward.
- When an item is dequeued, the front index also moves forward by one element



Efficient…but what is the **PROBLEM?**

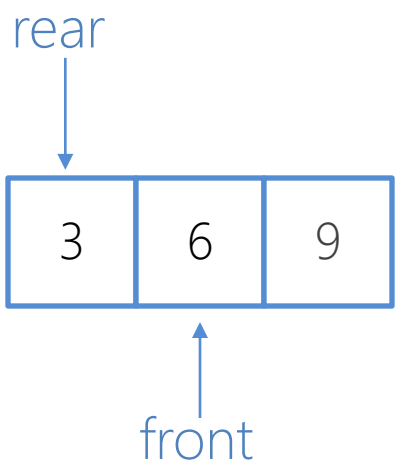
Final Solution: A Circular Array

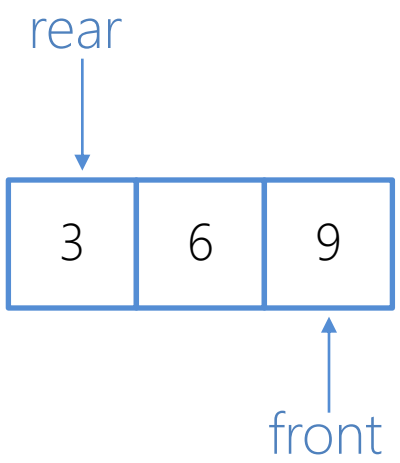
- Circular array
  - When an element moves past the end of a circular array, it wraps around to the beginning



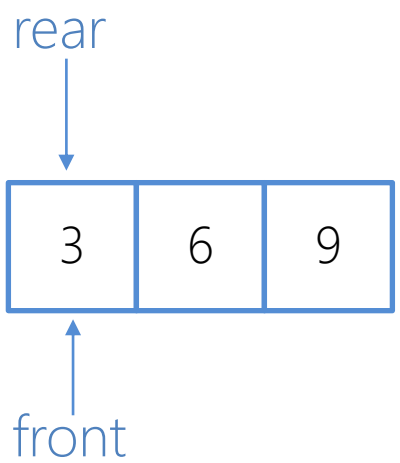
Index Growth: 0 → 1 → 2 → 3 → 4 → 5 → 6 → 0 → …

Dequeue in a Circular Array

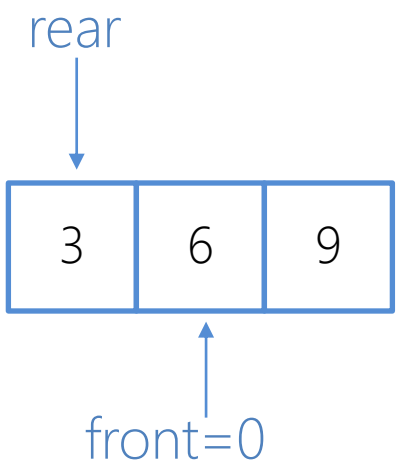




Dequeue()

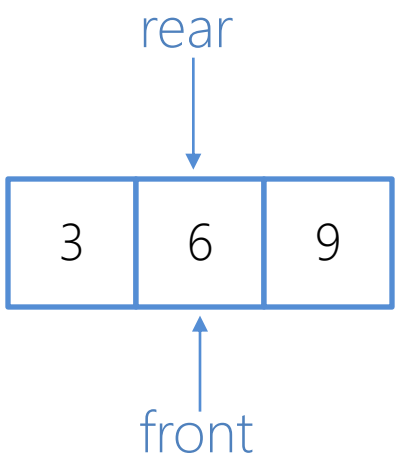


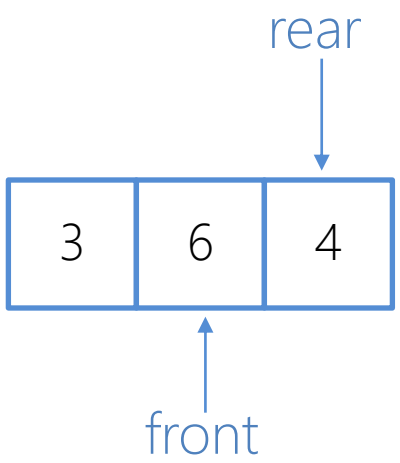
Dequeue()



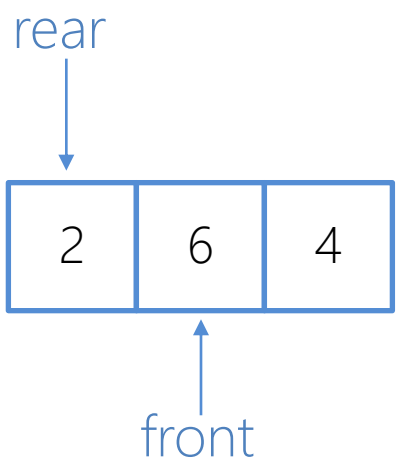
Dequeue()

Enqueue in a Circular Array

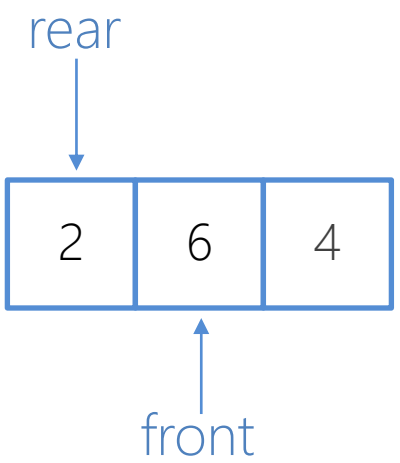




Enqueue(4)



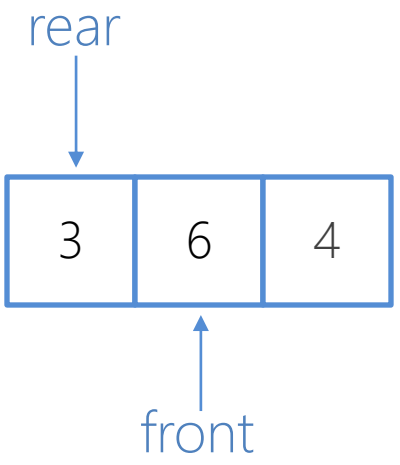
Enqueue(2)



Enqueue(1) Fails

Question to Ponder

- How to detect an empty or full queue using a circular array?
  - Empty
    - rear is one position before front
  - Full
    - rear is one position before front
  - Is this queue empty or full?



Question to Ponder

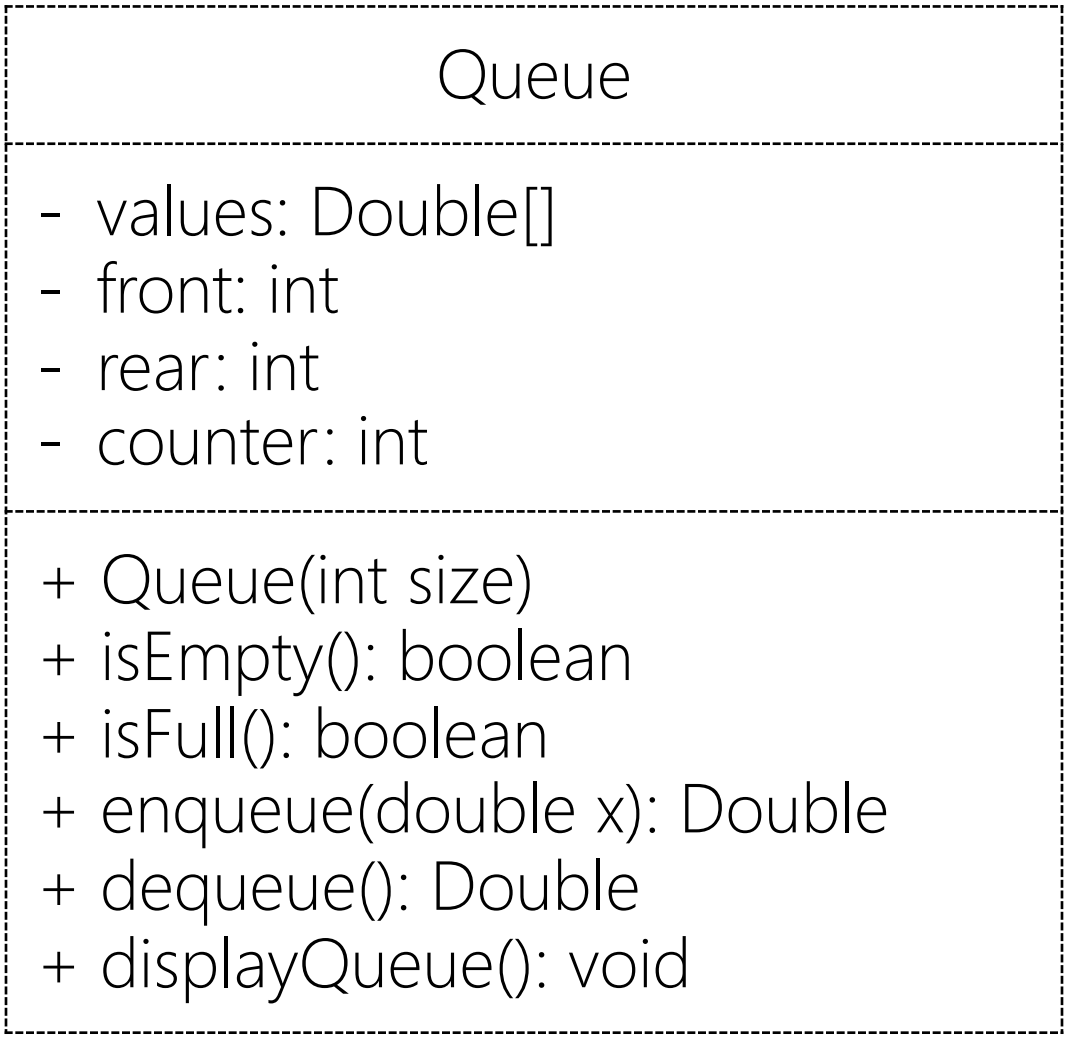
- How to detect an empty or full queue, using a circular array algorithm?

Use a counter which records number of elements in the queue.

Queue Implementation

- Data are stored in an array
  - values: the array of data items. The data can be of any type but we use Double for demonstration
  - front: index of the front
  - rear: index of the rear
  - counter: number of elements in the queue

Queue Implementation



Methods

- public Queue(int size)
  - Creates an empty queue whose capacity is *size*
- public boolean isEmpty()
  - Returns true if the queue is empty and false otherwise
- public boolean isFull()
  - Returns true if the queue is full and false otherwise

Methods

- public Double enqueue(double x)
  - Adds a new element with value *x* after the rear of the queue
  - Returns the new element if the operation is successful and null otherwise
- public Double dequeue()
  - Removes and returns the (old) front element of the queue
  - Returns null if the operation fails
- public void displayQueue()

```
front-> |        -2.0000        |
        |         3.0000        |
        |         1.0000        |
        |         8.0000        |<-rear
```

The Constructor

```
public Queue(int size) {
        values = new Double[size];
        front = 0;
        rear = -1;
        counter = 0;
}
```

Why?
Any other valid initialization values?

Enqueue

```
public Double enqueue(double x) {
        if(isFull())
                return null;
        rear = (rear + 1) % values.length;
        values[rear] = Double.valueOf(x);
        counter ++;
        return values[rear];
}
```

Circular index increase

Using Queue

```
public static void main(String[] args) {
    Queue myQueue = new Queue(4);
    System.out.println(myQueue.isEmpty());
    myQueue.enqueue(-2);
    myQueue.enqueue(3);
    myQueue.enqueue(1);
    System.out.println("The queue has 3 items: -2, 3, 1");
    myQueue.displayQueue();
    myQueue.enqueue(8);
    myQueue.enqueue(6);
    System.out.println("The queue has 4 items: -2, 3, 1, 8");
    System.out.println(myQueue.isFull());
    myQueue.displayQueue();
    myQueue.dequeue();
    myQueue.dequeue();
    System.out.println("The queue has 2 items: 1, 8");
    myQueue.displayQueue();
    myQueue.dequeue();
    myQueue.dequeue();
    myQueue.dequeue();
    System.out.println("The queue is empty:");
    myQueue.displayQueue();
}
```

Using Queue

```
public static void main(String[] args) {
    Queue myQueue = new Queue(4);
    System.out.println(myQueue.isEmpty());
    myQueue.enqueue(-2);
    myQueue.enqueue(3);
    myQueue.enqueue(1);
    System.out.println("The queue h
    myQueue.displayQueue();
    myQueue.enqueue(8);
    myQueue.enqueue(6);
    System.out.println("The queue h
    System.out.println(myQueue.isFu
    myQueue.displayQueue();
    myQueue.dequeue();
    myQueue.dequeue();
    System.out.println("The queue h
    myQueue.displayQueue();
    myQueue.dequeue();
    myQueue.dequeue();
    myQueue.dequeue();
    System.out.println("The queue is empty:");
    myQueue.displayQueue();
}
```

```
true
The queue has 3 items: -2, 3, 1
front-> |        -2.0000        |
        |         3.0000        |
        |         1.0000        |<-rear
The queue has 4 items: -2, 3, 1, 8
true
front-> |        -2.0000        |
        |         3.0000        |
        |         1.0000        |
        |         8.0000        |<-rear
The queue has 2 items: 1, 8
front-> |         1.0000        |
        |         8.0000        |<-rear
The queue is empty:
Empty queue!
```

Note

- The main function on page 26-27 is for demonstration.
- When you write your own main function, you should design testing cases like the ones you've learnt in the OOP course.

Task

- Complete *Queue.java* which implements the queue class
  - The class is defined on slide 21.
  - A *main* function has been given for the class which tests 5 functions: enqueue, dequeue, isEmpty, isFull, displayQueue
- Submit *Queue.java* to iSpace.