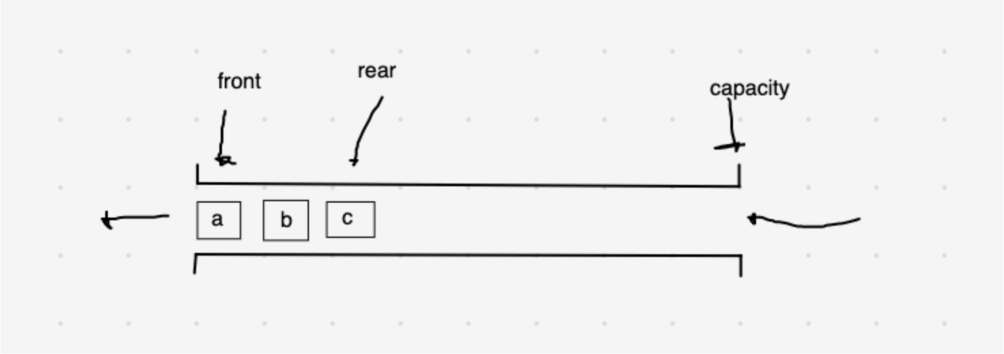
Queue

Queue values

- A FIFO (first-in first-out) collection of elements.
- Also known as a line-up.



- Can overflow/underflow similar to a stack.

Queue API

enqueue

Description	Add an element to the rear of the queue.
Signature	`void enqueue(int x)`
Preconditions	Queue is not full.
Returns	None.

dequeue

Description	Remove the front element of the queue.
Signature	`int dequeue()`
Preconditions	Queue is not empty.
Returns	The removed element.

front

Description	Check the front element in the queue.
Signature	`int front()`
Preconditions	Queue is not empty.
Returns	The front element.

is-empty

Description	Determine if the queue is empty.
Signature	`boolean isEmpty()`
Preconditions	None.
Returns	True if the queue is empty, false otherwise.

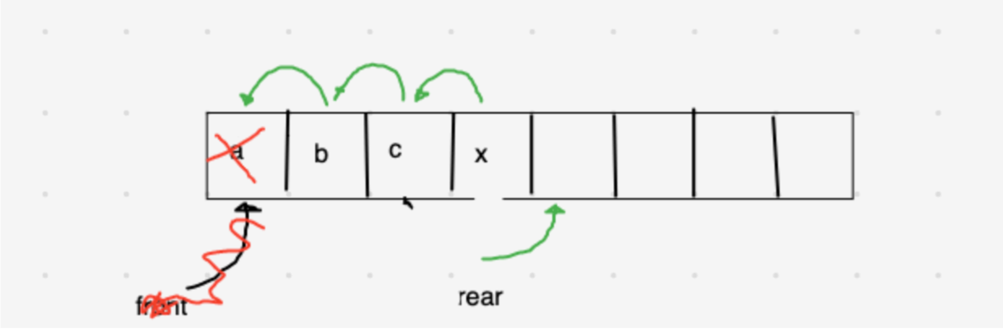
is-full

Description	Determine if the queue is full.
Signature	`boolean isFull()`
Preconditions	None.
Returns	True if the queue is full, false otherwise.

Queue implementation

Idea 1

- Use the stack implementation idea: one field to track the rear.
- But the dequeue can't be the same: we need to shift the elements down…



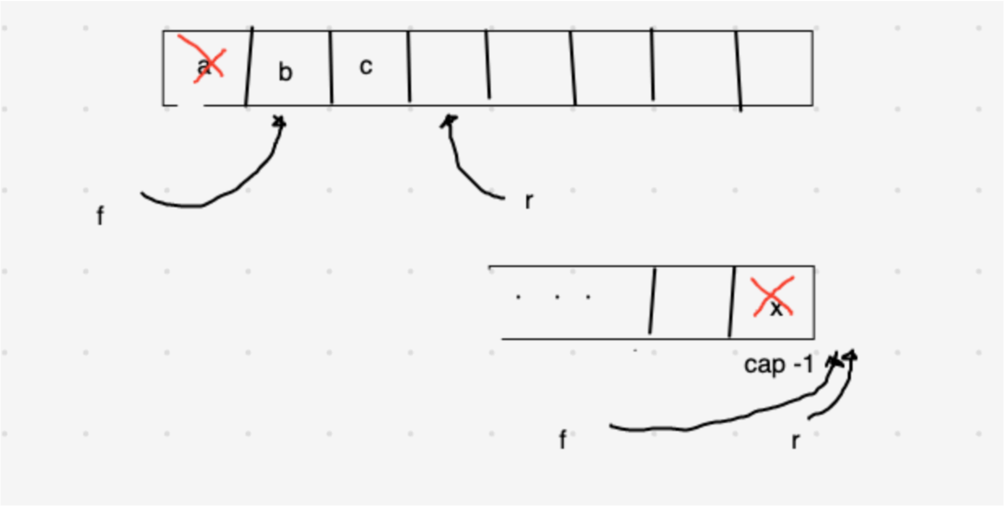
- We can do better!

Idea 2

- Add a `front` field

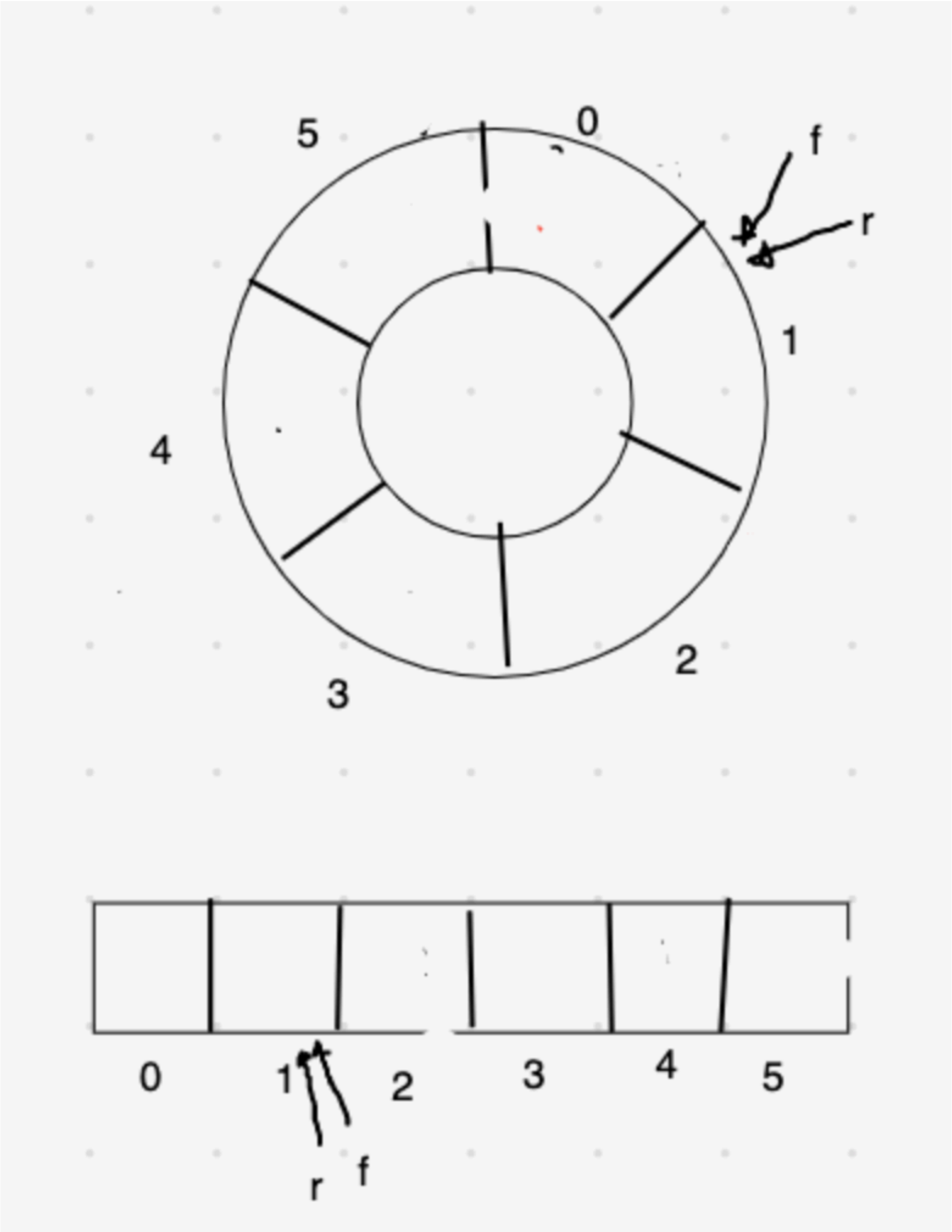
```
public class IntQueue {

    private int[] elements;
    private int rear;
    private int front;

    // Creates an empty stack
    public IntQueue(int capacity) {
         elements = new int[capacity];
         front = rear = 0;
    }

    public void enqeueue(int x) {
         if(isFull())
             throw new QueueOVerflowException();
         elements[rear++] = x;
    }

    public int dequeue() {
         if(isEmpty())
            throw new QueueUnderflowException();
         return elements[front++];
    }

    ...

}
```

- Problem: eventually we will end with both `front` and `rear` stuck at the end of the array:



Idea 3

- Use a ”circular array”, sometimes called a circular buffer.


5
0
f
r
1
4
2
3
0
1
2
3
4
5
r
f

Exercise: Implement the queue API using a circular array

Sample Solution

```
public class IntQueue {

    private int[] elements;
    private int rear;
    private int front;
    private boolean empty;

    // Creates an empty stack
    public IntQueue(int capacity) {
         elements = new int[capacity];
         front = rear = 0;
         empty = true;
    }

    public void enqeueue(int x) {
         if(isFull())
             throw new QueueOVerflowException();
         elements[rear] = x;
         rear++;
         if(rear >= element.length)
             rear = 0;
         // or: rear = (rear + 1) % element.length
         empty = false;
    }

    public int dequeue() {
         if(isEmpty())
            throw new QueueUnderflowException();
         int tmp = elements[front];
         front++;
         if(front >= element.length)
             front = 0;
         // or: front = (front + 1) % element.length
         if(front == rear)
            empty = true;
         return tmp;
    }
    public boolean isFull() {
        return front == rear && !empty;
    }

    public boolean isEmpty() {
        return empty;
    }

}
```