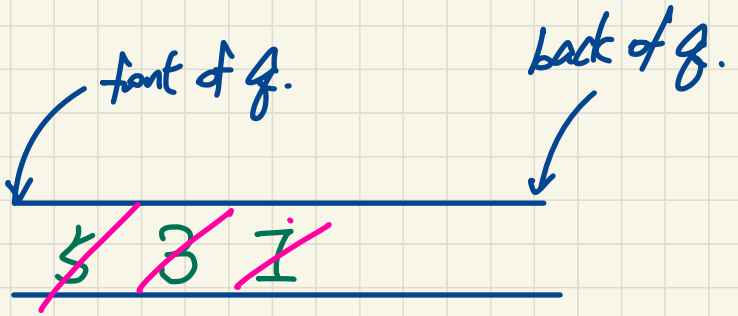
Queue ADT: Illustration

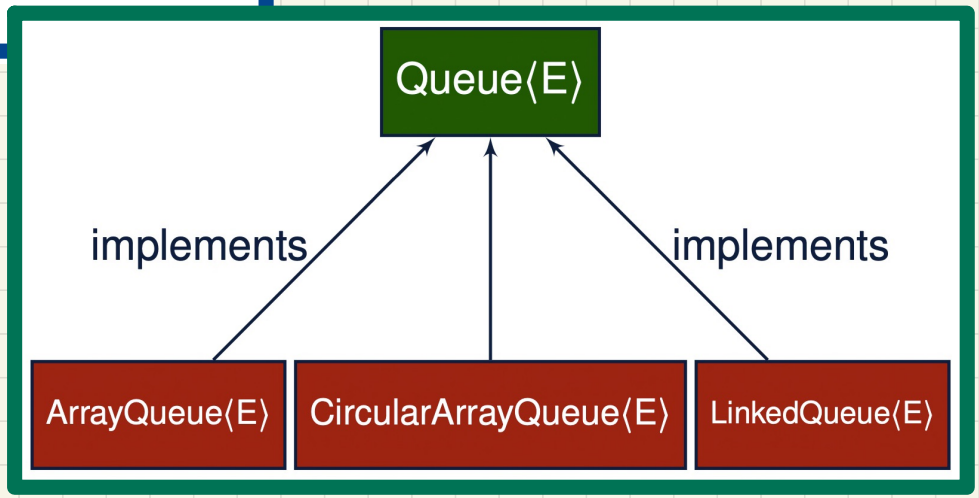
	isEmpty	size	first
new queue	T	0	n.a.
enqueue(5)	F	1	5
enqueue(3)	F	2	5
enqueue(1)	F	3	5
dequeue rtn. 5	F	2	3
dequeue rtn. 3	F	1	1
dequeue rtn. 1	T	0	n.a.



→ First-In First-Out (FIFO)

Implementing the Queue ADT in Java: Architecture

```
public interface Queue<E> {
   public int size();
   public boolean isEmpty();
   public E first();
   public void enqueue(E e);
   public E dequeue();
}
```



Implementing the Queue ADT using an Array

```
public class ArrayQueue<E> implements Queue<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int r; /* rear index */
  public ArrayQueue() {
    data = (E[]) new Object[MAX_CAPACITY];
    r = -1;
  }
  public int size() { return (r + 1); }
  public boolean isEmpty() { return (r == -1); }
  public E first() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[0]; }
  }
  public void enqueue(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { r ++; data[r] = e; }
  }
  public E dequeue() {
    if (isEmpty()) { /* Precondition Violated */ }
    else {
      E result = data[0];
      for (int i = 0; i < r; i ++) { data[i] = data[i + 1]; }
      data[r] = null; r --;
      return result;
    }
  }
}
```

O(1)

O(1)

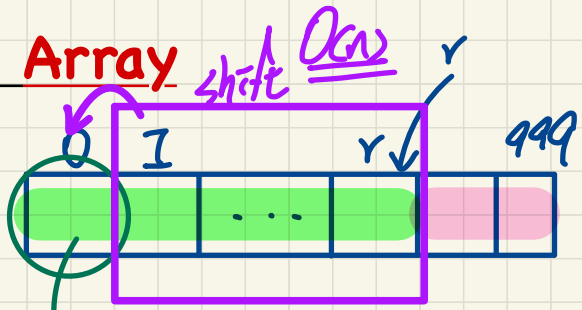
O(1)

front index

O(1)

O(n)

shifting "2nd item" and onwards to the left by one position



limitation: no resizing.

to improve this, we need to be flexible about where the front index is ⇒ Circular array.

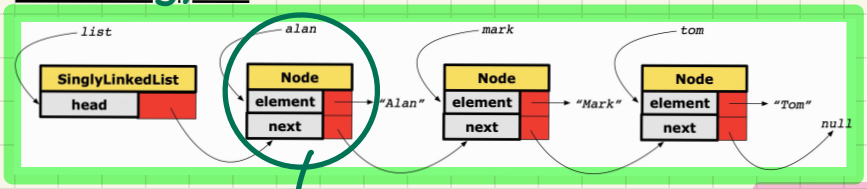
Implementing the Queue ADT using a SLL

```
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list;
  ...
}
```

O(n)
① use SI instead
② use DLL instead

Queue Method	Singly-Linked List Method	
	Strategy 1	Strategy 2
size	list.size	
isEmpty	list.isEmpty	
first	list.first	list.last
enqueue	list.addLast	list.addFirst
dequeue	list.removeFirst	list.removeLast

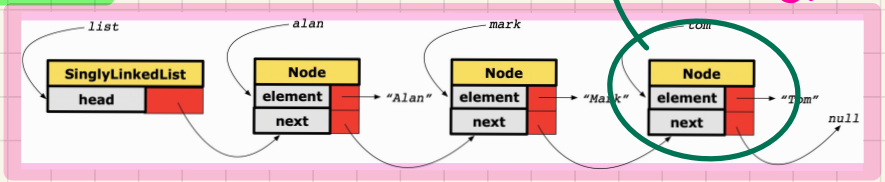
Strategy 1



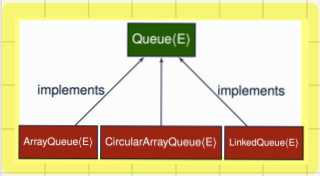
first of queue

first of queue.

Strategy 2



Stack ADT: Testing Alternative Implementations




```
public class ArrayQueue<E> implements Queue<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int r = -1; /* rear index */
  public ArrayQueue() {
    data = (E[]) new Object[MAX_CAPACITY];
    r = -1;
  }
  public int size() { return (r + 1); }
  public boolean isEmpty() { return (r == -1); }
  public E first() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[0]; }
  }
  public void enqueue(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { r ++; data[r] = e; }
  }
  public E dequeue() {
    if (isEmpty()) { /* Precondition Violated */ }
    else {
      E result = data[0];
      for (int i = 0; i < r; i ++) { data[i] = data[i + 1]; }
      data[r] = null; r --;
      return result;
    }
  }
}
```

dynamic binding.

Polymorphism

```
@Test
public void testPolymorphicQueues() {
  Queue<String> q = new ArrayQueue<>();
  q.enqueue("Alan"); /* dynamic binding */
  q.enqueue("Mark"); /* dynamic binding */
  q.enqueue("Tom"); /* dynamic binding */
  assertTrue(q.size() == 3 && !q.isEmpty());
  assertEquals("Alan", q.first());

  q = new LinkedQueue<>();
  q.enqueue("Alan"); /* dynamic binding */
  q.enqueue("Mark"); /* dynamic binding */
  q.enqueue("Tom"); /* dynamic binding */
  assertTrue(q.size() == 3 && !q.isEmpty());
  assertEquals("Alan", q.first());
}
```

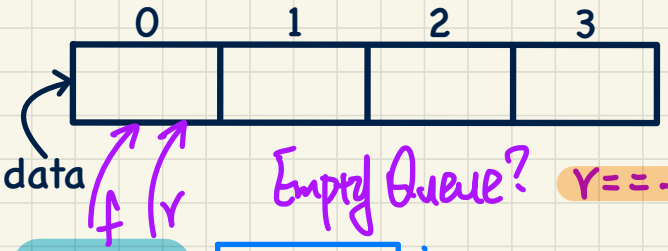
size of Queue vs. size of array | % modulo

Implementing the Queue ADT using a Circular Array

1. fix-sized (no resizing)
2. flexible for performing "dequeue"

Assume: A circular array of length 4.

Phase 0: Empty Queue q



f=0;
r=0;

Empty Queue? r==f

Phase 1: enqueue 3 elements

- data[r] = item;
- r++;

Queue Full? (r+1)%N == f
when r points to is the only empty slot.

(r+1)%4 = 3

[f, r-1] = (r-1)-f+1 = r-f

empty slots before f
slots before this index

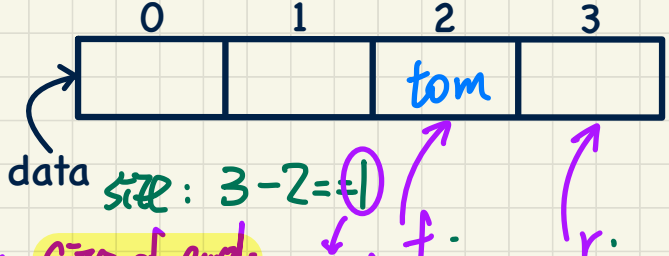
size: r>=f → r-f

3 = 1+(4-2)

size: r<f → r+(N-f)

Phase 2: dequeue 2 times

data[f]=null; f++;



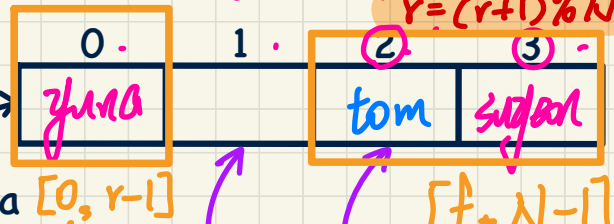
size: 3-2=1

size of array: 4 (fixed) → size of 4.

(3+1)%4 = 0

Phase 3: enqueue 2 elements

data[r] = item; ~~r++;~~ r=(r+1)%N



[0, r-1] → (r-1)-0+1 = r

[f, N-1] → (N-1)-f+1 = N-f