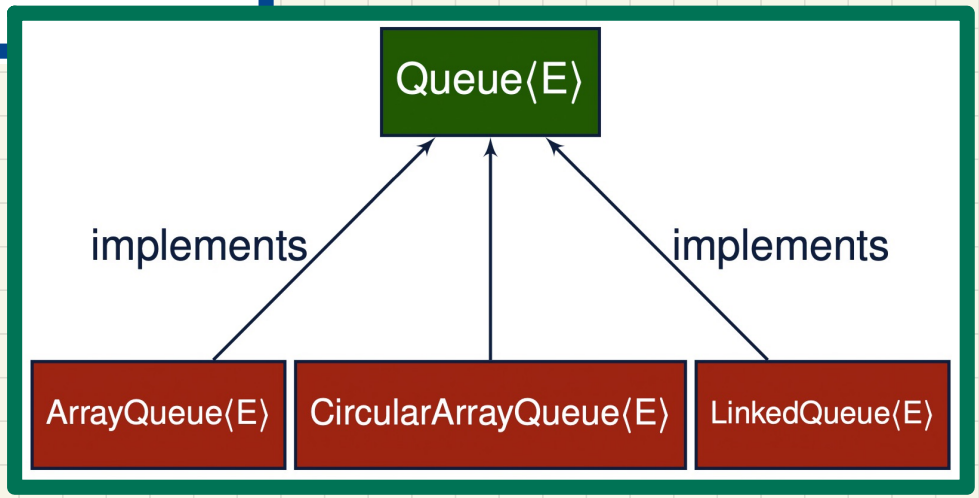
Queue ADT: Illustration

	isEmpty	size	first
new queue			
enqueue(5)			
enqueue(3)			
enqueue(1)			
dequeue			
dequeue			
dequeue			

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

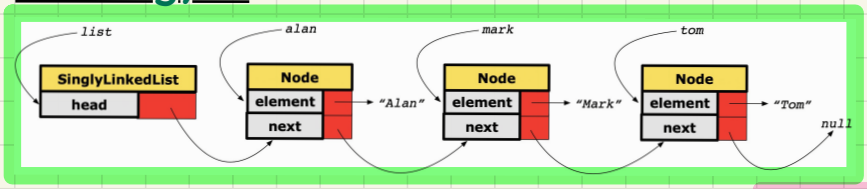
```
public class ArrayQueue<E> implements Queue<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int r; /* rear index */
  public ArrayQueue() {
    data = (E[]) new Object[MAX_CAPACITY];
    r = -1;
  }
  public int size() { return (r + 1); }
  public boolean isEmpty() { return (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;
    }
  }
}
```

Implementing the Queue ADT using a SLL

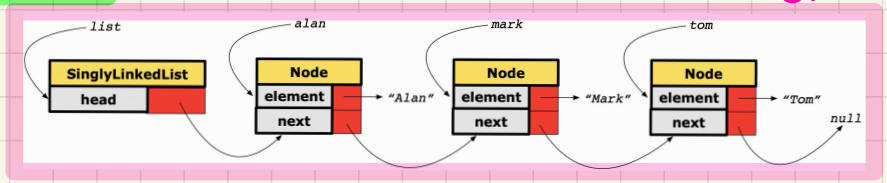
```
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list;
  ...
}
```

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



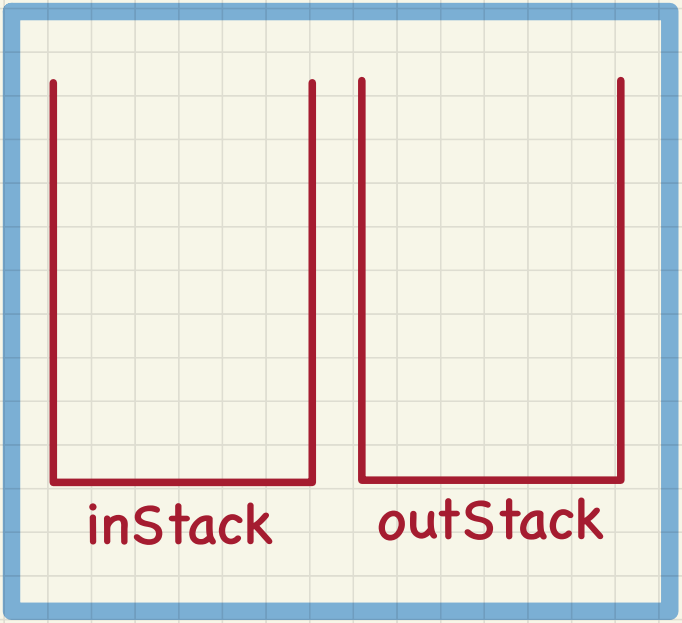
Strategy 2



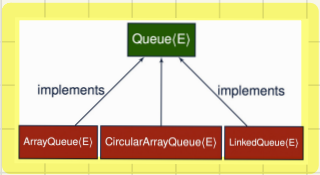
Exercise: Implementing a Queue using Two Stacks

Queue Operation:

```
q.enqueue("alan");
q.enqueue("mark");
q.enqueue("tom");
String front = q.dequeue();
front = q.dequeue();
front = q.dequeue();
```



Queue 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;
    }
  }
}
```

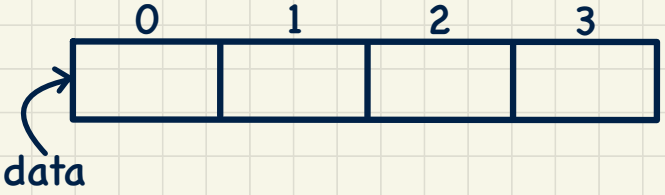
```
@Test
public void testPolymorphicQueues() {
  Queue<String> q = new ArrayQueue<>();
  q.enqueue("Alan"); /* dynamic binding */
  q.enqueue("Mark"); /* dynamic binding */
  q.enqueue("Tom"); /* dynamic binding */
  assertTrue(q.size() == 3 && !q.isEmpty());
  assertEquals("Alan", q.first());

  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());
}
```

Implementing the Queue ADT using a Circular Array

Assume: A circular array of length 4.

Phase 0: Empty Queue q



Phase 1: enqueue 3 elements

Phase 2: dequeue 2 times

Phase 3: enqueue 2 elements