CSCI 3230 Data Structures

Linear Data Structures

Weitian Tong, Ph.D.

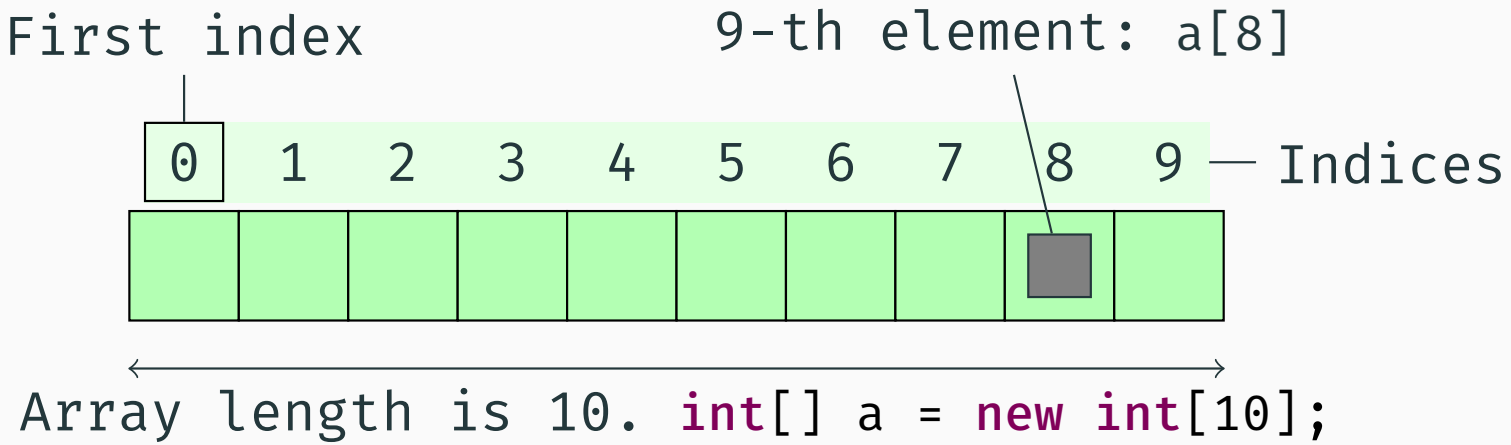
Department of Computer Science
Georgia Southern University
Website: www.weitiantong.com
Email:wtong@georgiasouthern.edu

Table of contents

Arrays

Array

Array: a fixed-size sequential collection of elements of the same type



- array size is fixed at the time of array's construction
- array elements are placed contiguously in memory
- indexing into a component takes $O(1)$ time

Array: continue ...

- Java arrays are homogeneous
  - all components must be of the same (object or primitive) type.
  - but, an array of an object type can contain objects of any respective subtype
- An array is itself an object
  - it is allocated dynamically by means of new
  - it is automatically deallocated when no longer referred to
- When an array is first created, **all values are automatically initialized** with
  - 0, for an array of int[] or double[] type
  - false, for a boolean[] array
  - null, for an array of objects

Array: Limitations

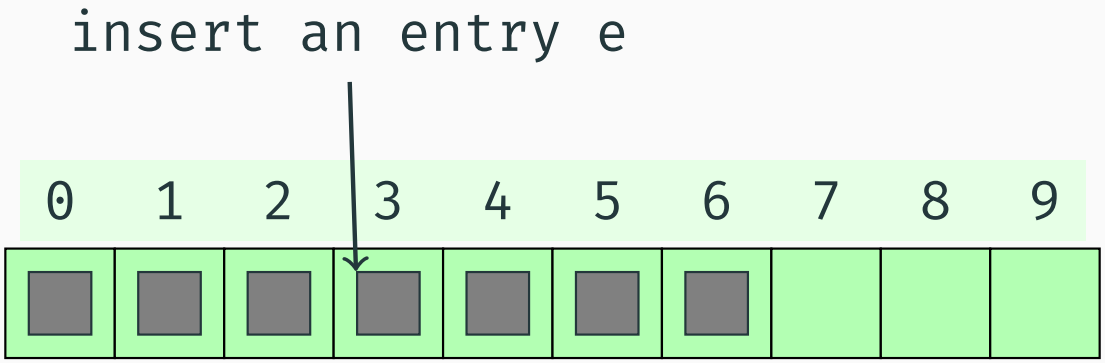
Limitations of Arrays:

- Static data structure
- Insertion / deletion in an array is time consuming ($O(n)$, why?)

Array: Limitations

Limitations of Arrays:

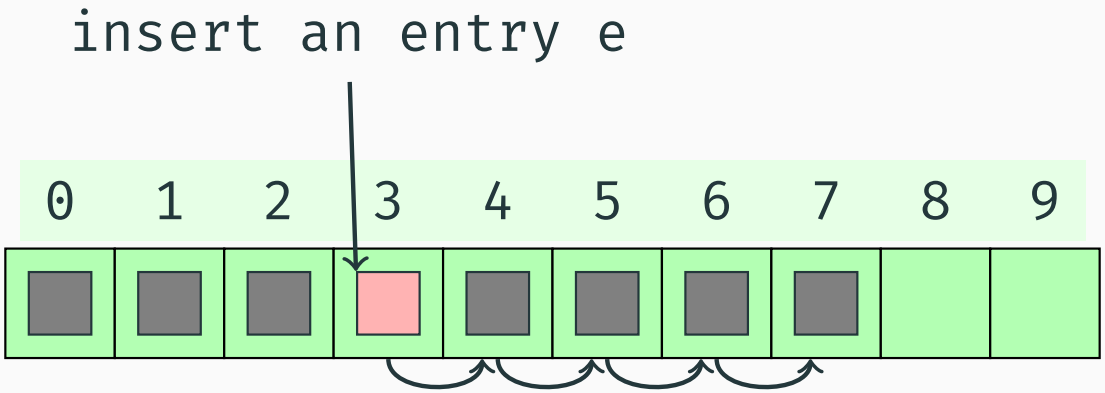
- Static data structure
- Insertion / deletion in an array is time consuming ($O(n)$, why?)



Array: Limitations

Limitations of Arrays:

- Static data structure
- Insertion / deletion in an array is time consuming ($O(n)$, why?)



ArrayLists

ArrayList

ArrayList: dynamic sized arrays in Java that implement List interface.

```
1  ArrayList {
2    size();
3    isEmpty();
4    add(int I, E e);
5    get(int i);
6    remove(int i);
7    set(int I, E e);
8  }
```

ArrayList

ArrayList: dynamic sized arrays in Java that implement List interface.

```
1 ArrayList {
2   size();
3   isEmpty();
4   add(int I, E e);
5   get(int i);
6   remove(int i);
7   set(int I, E e);
8 }
```

In the array based implementation

- The space used by the data structure is $O(n)$
- **size**, **isEmpty**, **get** and **set** run in $O(1)$ time
- **add** and **remove** run in $O(n)$ time as these two operations need to shift forward and backward on the array.

ArrayList: add(element)

When the array is full, instead of throwing an exception, we could replace the array with a larger one.

```
1  Algorithm add(element):
2  if n = N then
3      newArray = new array of size newN
4      for i = 0 to N - 1 do
5          newArray[i] = array[i]
6      array = newArray
7      n = n + 1
8      array[n] = element
```

ArrayList: add(element)

When the array is full, instead of throwing an exception, we could replace the array with a larger one.

```
1  Algorithm add(element):
2  if n = N then
3      newArray = new array of size newN
4      for i = 0 to N - 1 do
5          newArray[i] = array[i]
6      array = newArray
7      n = n + 1
8      array[n] = element
```

- **Incremental strategy**: increase the size by a constant c
- **Doubling strategy**: double the size

Incremental Strategy Analysis (optional)

n	N	
0	0	
1	c	Extend array
2	c	
⋮	⋮	
c	c	
c + 1	2c	Extend array
⋮	⋮	
2c	2c	
2c + 1	3c	Extend array
⋮	⋮	
n	kc	$(k = \lceil n/c \rceil)$

Replace the array $k = \lceil n/c \rceil$ times

The total time $T(n)$ is proportional to

$$n+c+2c+3c+4c+\ldots+kc = n+ck(k+1)/2$$

Since c is a constant,
$T(n)$ is $O(n + k^2)$, *i.e.*, $O(n^2)$

The amortized time of an **add(element)** operation is $O(n)$

Doubling Strategy Analysis (optional)

n	N	
0	0	
1	1	Extend array
2	2	Extend array
3	4	Extend array
4	4	
5	8	Extend array
6	8	
7	8	
⋮	⋮	
n	2^k	$(k = \lceil \log n \rceil)$

Replace the array $k = \lceil \log n \rceil$ times

The total time $T(n)$ is proportional to

$$n+1+2+4+8+\ldots+2^k = n+2^{k+1}-1 \leq 5n$$

Thus $T(n)$ is $O(n)$

The amortized time of an **add(element)** operation is $O(1)$

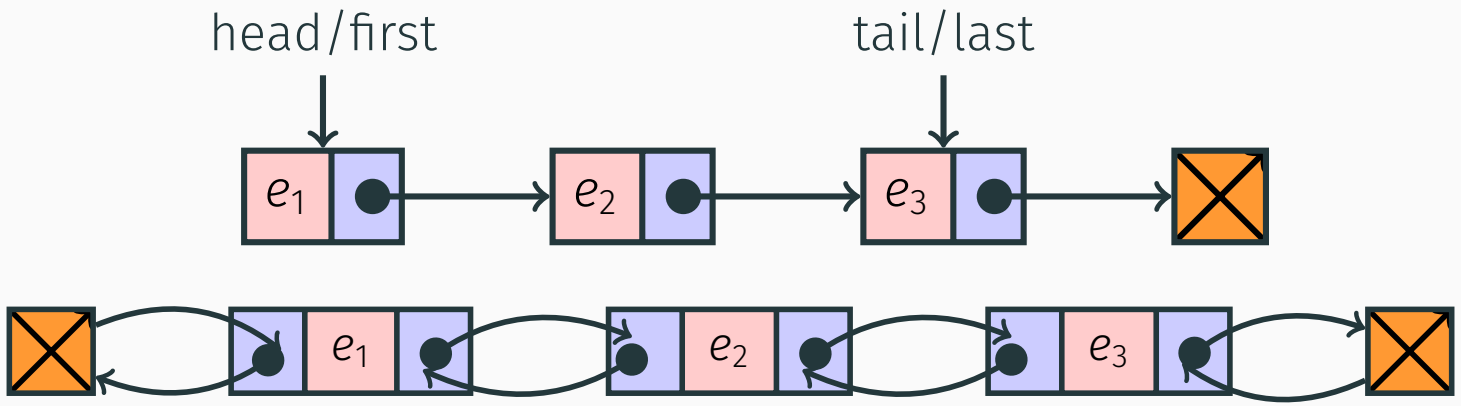
Linked Lists

Linked Lists

A concrete data structure consisting of a sequence of nodes

Each **node** stores

- element
- link to the next/previous node



Singly Linked List vs **Doubly Linked List**

SinglyLinkedList.java, DoublyLinkedList.java

Linked Lists: Node class

```
1  private static class Node<E> {
2      private E element;
3      private Node<E> next;
4      // private Node<E> prev;
5      public Node(E e, Node<E> n) {
6        element = e;
7        next = n;
8        // prev = p;
9      }
10     public E getElement() { return element; }
11     public Node<E> getNext() { return next; }
12     // public Node<E> getPrev() { return prev; }
13     public void setNext(Node<E> n) { next = n; }
14     // public void setPrev(Node<E> p) { prev = p; }
15 }
```

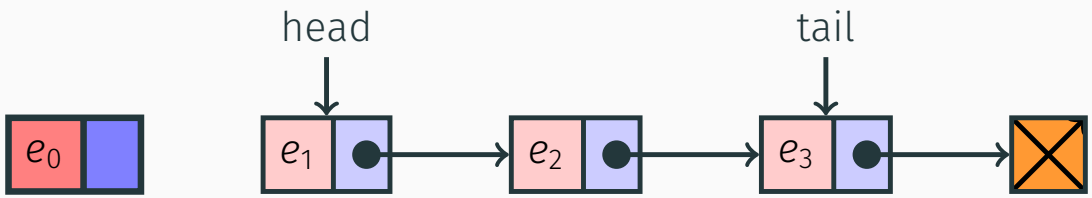
Linked Lists

Singly Linked List

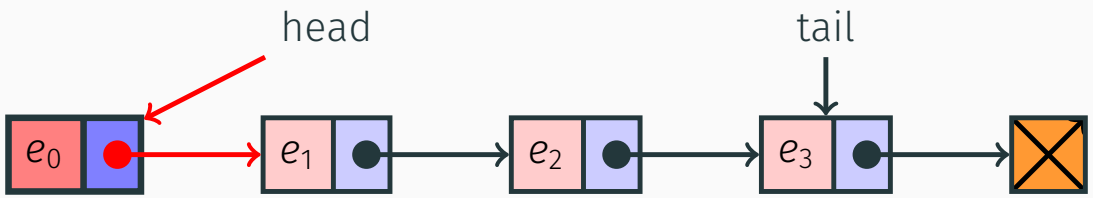
Singly Linked List: ADT

```
1  private Node<E> head = null;
2  private Node<E> tail = null;
3  /** Number of nodes in the list */
4  private int size = 0;
5
6  public int size()
7  public boolean isEmpty()
8  public E first()
9  public E last()
10 public void addFirst(E e)
11 public void addLast(E e)
12 public E removeFirst()
```

Singly Linked List: addFirst(E e), removeFirst()

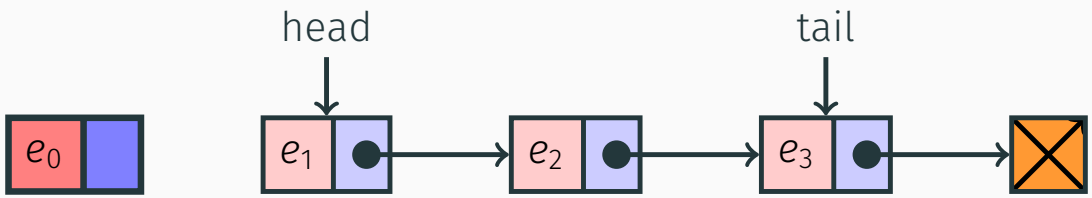


Singly Linked List: addFirst(E e), removeFirst()



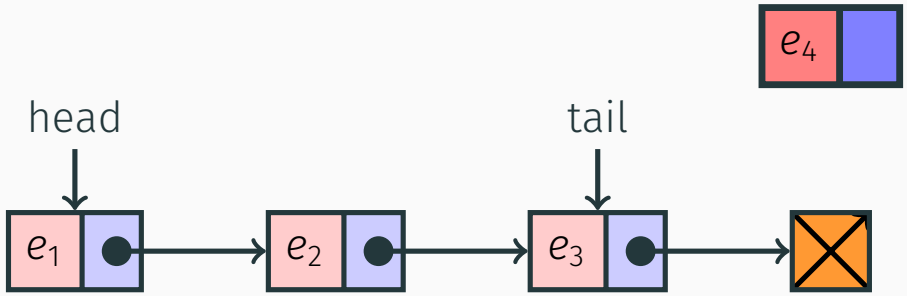

```
1  public void addFirst(E e) {
2    head = new Node<>(e, head);
3    if (size == 0)
4      tail = head;
5    size++;
6  }
```

Singly Linked List: addFirst(E e), removeFirst()

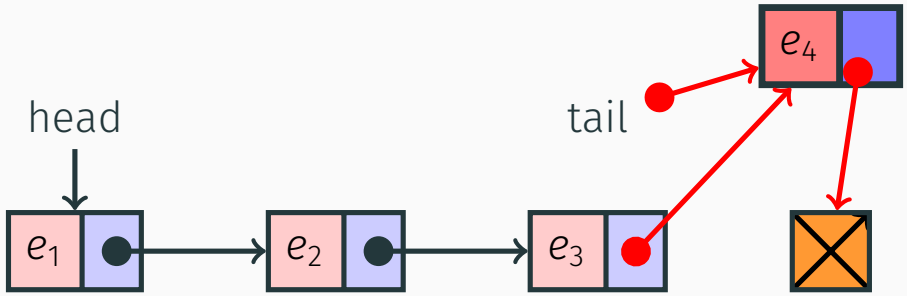



```
1  public E removeFirst() {
2    if (isEmpty()) return null;
3    E answer = head.getElement();
4    head = head.getNext();
5    size--;
6    if (size == 0)
7      tail = null;
8    return answer;
9  }
```

Singly Linked List: addLast(E e), removeLast()

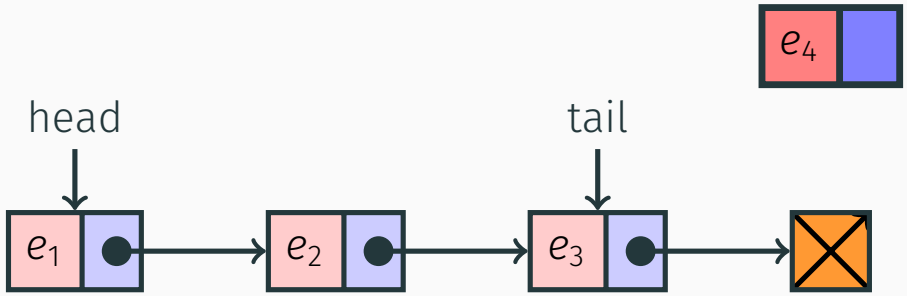


Singly Linked List: addLast(E e), removeLast()




```
1  public void addLast(E e) {
2    Node<E> newest = new Node<>(e, null);
3    if (isEmpty()) // previously empty list
4      head = newest;
5    else // new node after existing tail
6      tail.setNext(newest);
7    tail = newest; // new node becomes the tail
8    size++;
9  }
```

Singly Linked List: addLast(E e), removeLast()



There is no constant-time way to update the tail to point to the previous node

Why?

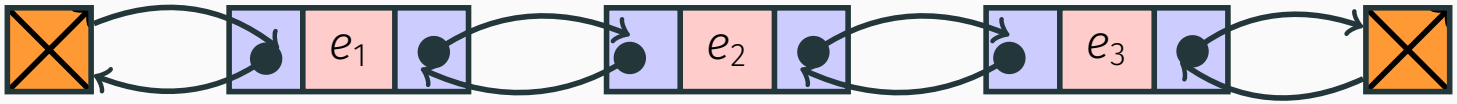
Linked Lists

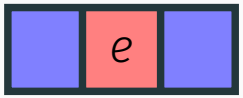
Doubly Linked List

Doubly Linked List: ADT

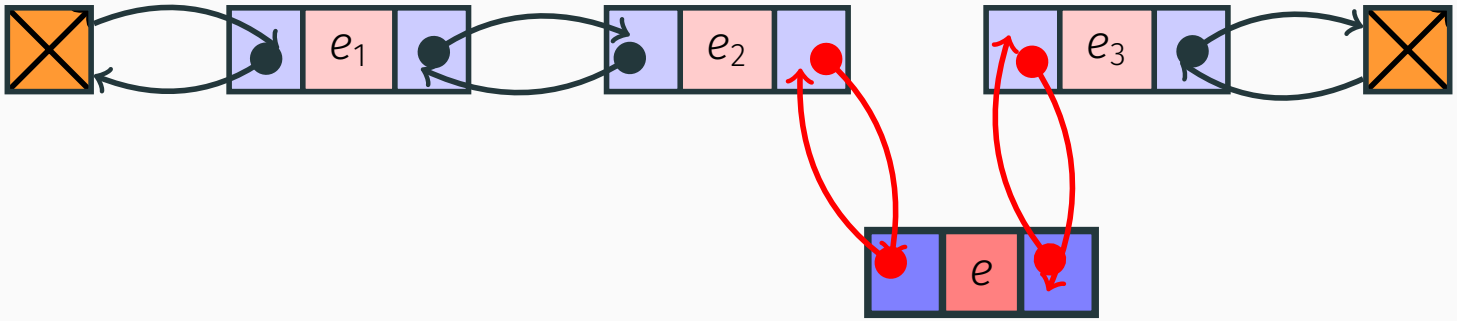
```
1  private Node<E> head = null;
2  private Node<E> tail = null;
3  private int size = 0;
4
5  public int size()
6  public boolean isEmpty()
7  public E first()
8  public E last()
9  public void addFirst(E e)
10 public void addLast(E e)
11 public E removeFirst()
12 public E removeLast()
13 private void addBetween
14   (E e, Node<E> predecessor, Node<E> successor)
15 private E remove(Node<E> node)
```

Doubly Linked List: addBetween and remove

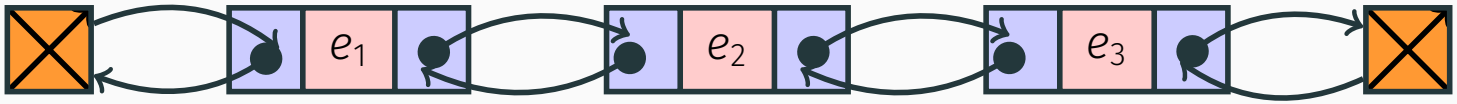


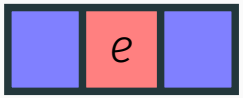


Doubly Linked List: addBetween and remove



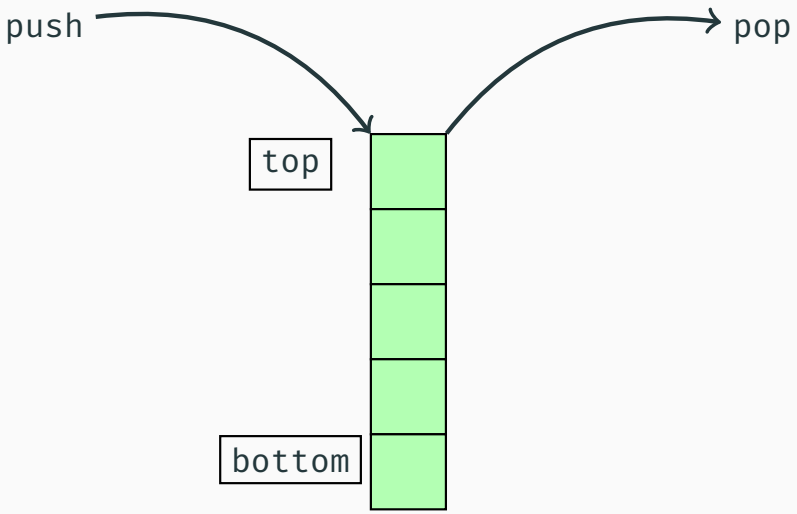
Doubly Linked List: addBetween and remove





Stacks

Stack



- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out scheme

Stack Interface

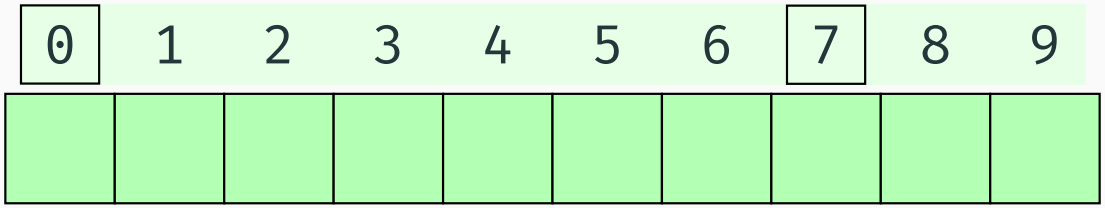
```
1  public interface Stack<E> {
2    int size();
3    boolean isEmpty();
4    void push(E e);
5    E top();
6    E pop();
7  }
```

Note: java.util.Stack provides push and pop, but differs in other respects.

ArrayStack.java, LinkedStack.java

Array-based Stack: ArrayStack.java

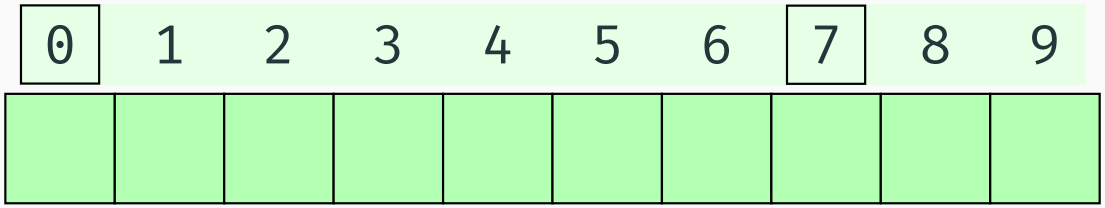
```
1  public class ArrayStack<E> implements Stack<E> {
2    private E[] data;
3    private int t = -1;
4    // index of the top element in stack
5    ...
6    public int size()
7    public boolean isEmpty()
8    public void push(E e)
9    public E pop()
10 }
```



Which side is “bottom”?

Array-based Stack: ArrayStack.java

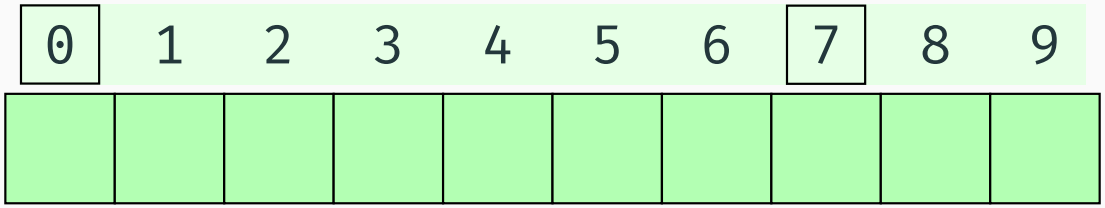
```
1  public class ArrayStack<E> implements Stack<E> {
2    private E[] data;
3    private int t = -1;
4    // index of the top element in stack
5    ...
6    public int size()
7    public boolean isEmpty()
8    public void push(E e)
9    public E pop()
10 }
```



Which side is “bottom”? bottom ..... top

Array-based Stack: ArrayStack.java

```
1  public class ArrayStack<E> implements Stack<E> {
2    private E[] data;
3    private int t = -1;
4    // index of the top element in stack
5    ...
6    public int size()
7    public boolean isEmpty()
8    public void push(E e)
9    public E pop()
10 }
```

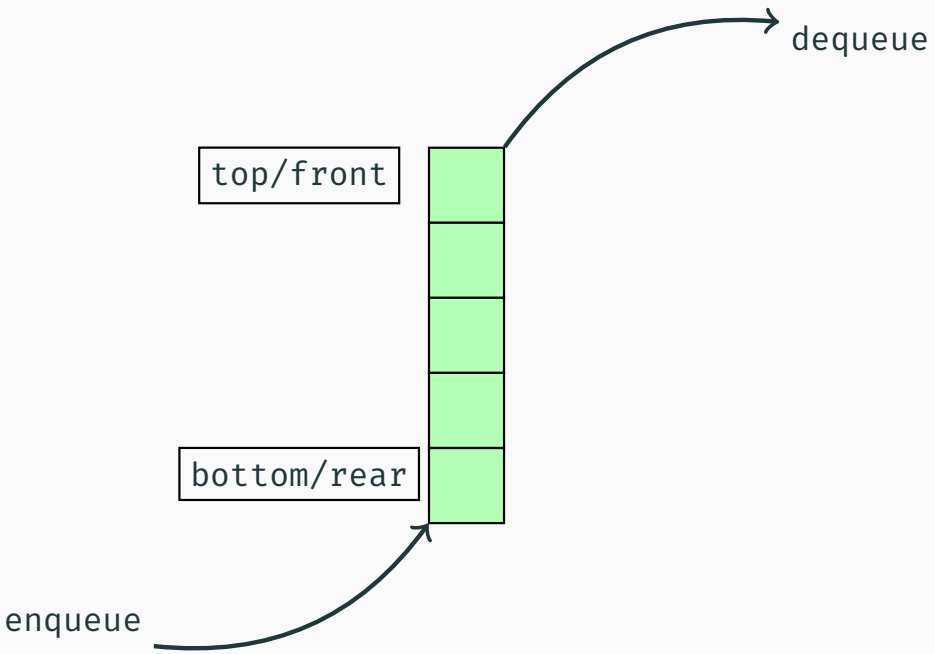


Which side is “bottom”? bottom ..... top

- **Performance**: each operation runs in time $O(1)$
- **Limitations**: fixed size

Queues

Queue



- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme

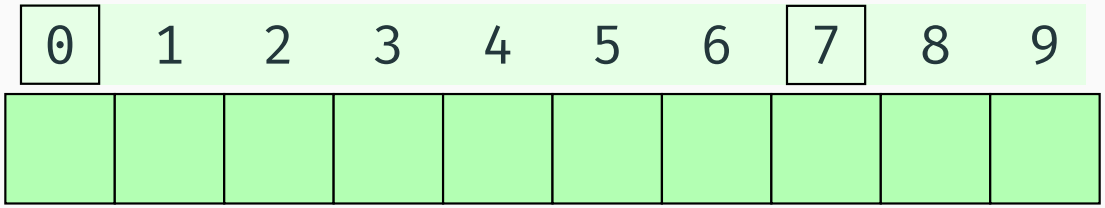
Queue Interface

```
1  public interface Queue<E> {
2    int size();
3    boolean isEmpty();
4    void enqueue(E e);
5    E first();
6    E dequeue();
7  }
```

ArrayQueue.java, LinkedQueue.java

Array-based Queue

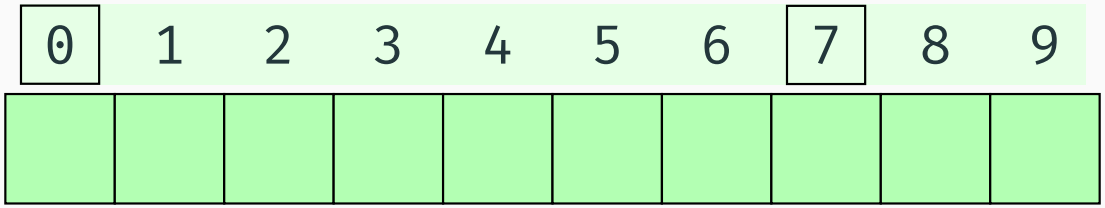
```
1  public class ArrayQueue<E> implements Queue<E> {
2    /** Generic array used for storage of queue elements. */
3    private E[] data;
4    /** Index of the top element of the queue in the array. */
5    private int f = 0;
6   /** Current number of elements in the queue. */
7    private int sz = 0;
8    ...
9    public int size()
10   public boolean isEmpty()
11   public void enqueue(E e) throws IllegalStateException
12   public E first()
13   public E dequeue()
14 }
```



Which side is “front”?

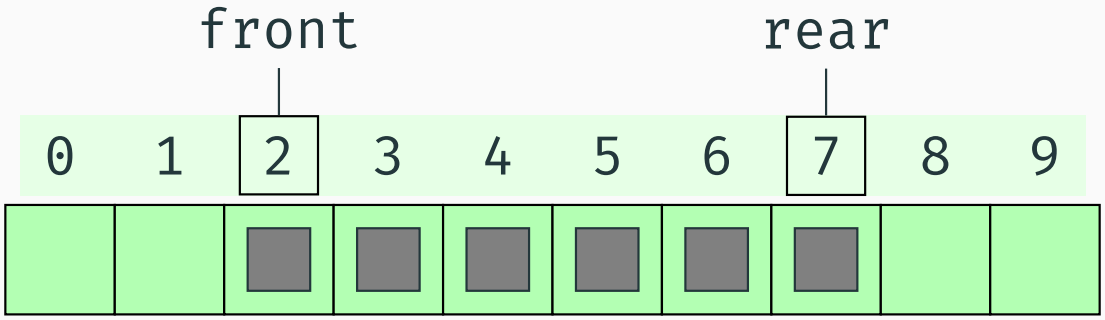
Array-based Queue

```
1  public class ArrayQueue<E> implements Queue<E> {
2    /** Generic array used for storage of queue elements. */
3    private E[] data;
4    /** Index of the top element of the queue in the array. */
5    private int f = 0;
6   /** Current number of elements in the queue. */
7    private int sz = 0;
8    ...
9    public int size()
10   public boolean isEmpty()
11   public void enqueue(E e) throws IllegalStateException
12   public E first()
13   public E dequeue()
14 }
```



Which side is “front”? front ..... rear

Array-based Queue



mod operation

```
1  Algorithm enqueue(o):
2  if size() = N - 1 then
3    throw
       IllegalStateException
4  else
5    r = (f + sz) mod N
6    Q[r] = o
7    sz = (sz + 1)
```

```
1  Algorithm dequeue():
2  if isEmpty() then
3    return null
4  else
5    o = Q[f]
6    f = (f + 1) mod N
7    sz = (sz - 1)
8    return o
```

- **Performance**: each operation runs in time $O(1)$
- **Limitations**: fixed size

Thank you!

Questions?