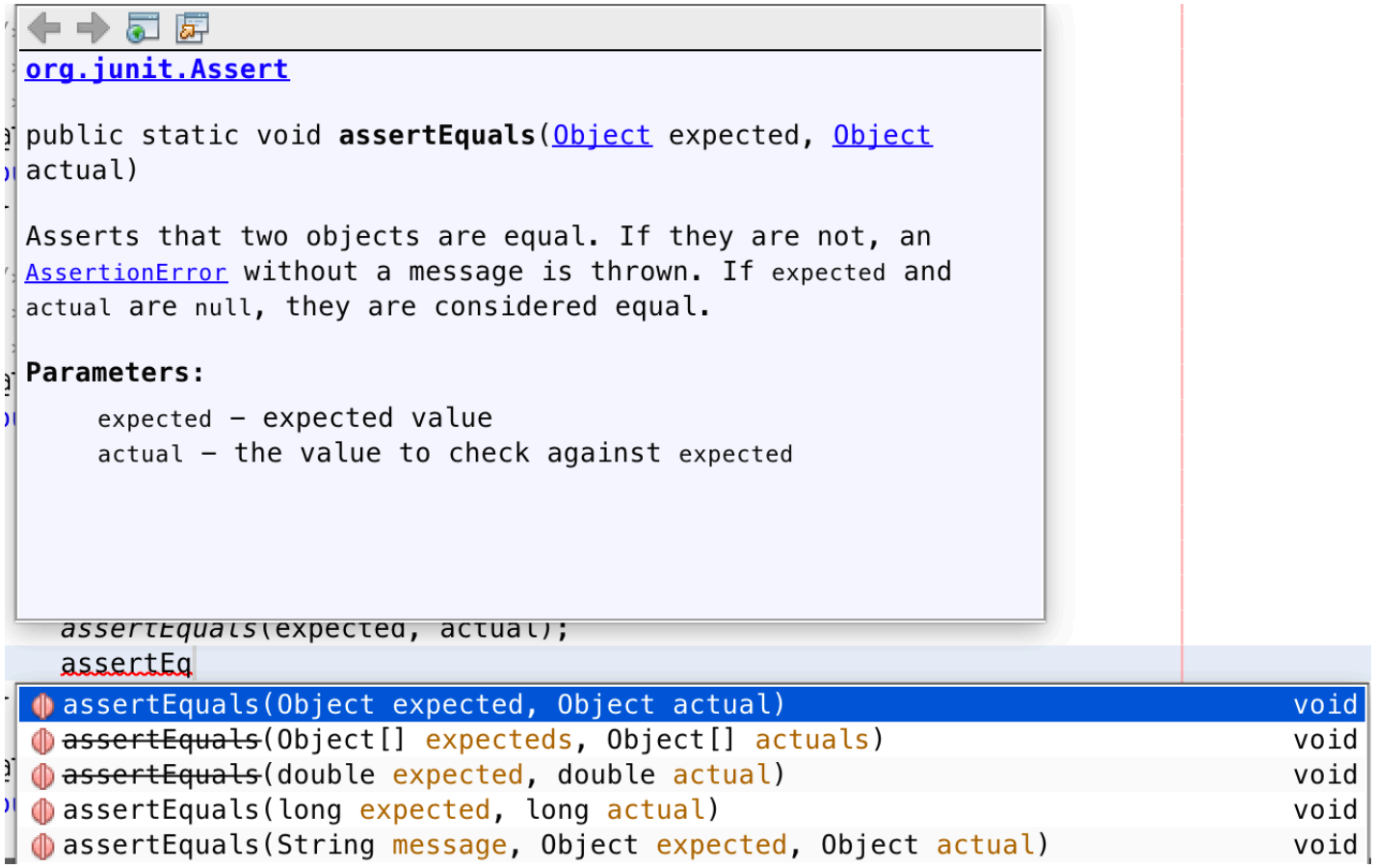
You are writing JUnit tests now?



it's time to talk about Java equality...

== isn't always equal?

In Java, == does the expected for primitives.

```
int a = 26;
int b = 26;
// a == b is true
```

```
int a = 13;
int b = 26;
// a == b is false
```

Comparing two references checks if they are pointing to the same object

```
Patient p1 = new Patient("Marion", 100);
Patient p2 = new Patient("Marion", 100);
Patient p3 = p1;
// p1 == p2  is false
// p1 == p3  is true
```

Not pointing to the same object? not ==

The equals() method

```
class Patient {
    String name;
    int height;
}
```

We decide that two Patients are equal() when they have the same <u>name</u> and <u>height</u>

the code that does that...

```
public boolean equals(Object o) {
        if (o instanceof Patient){
                Patient op = (Patient) o;
                return this.height==op.height &&
                        this.name.equals(op.name);
        } else {
                return false;
        }
}
```

String.equals compares the content of the String instead of references

Every Java class already has an invisible `equals` method defined. But you have to *override* it with your own if you want to do something smarter like compare the fields.

Secondary new things in this snippet of code

- instanceof to check if o is a Patient
- casting o from Object to Patient

Peer instruction

```
boolean equals(Object o) {
        if (o instanceof Cat) {
                Cat c = (Cat) o;
                return this.breed.equals(o.breed);
        }
        return false;
}
```

Object o1 = new Object();
Object o2 = new Cat("Siamese");
Cat o3 = new Cat("Tabby");
Cat o4 = new Cat("Siamese");
Cat o5 = o2;

Which are true statements?

A. o1 == o2
B. o2 == o4
C. o2 == o5
D. o4==o5
E. o2.equals(o4)
F. o2.equals(o5)
G. o4.equals(o5)
H. o2.equals(o3)

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+

Midterm 1

- Next Tuesday 9/19 in class, 75 minutes
- What is on it?
  - Anything you've practiced including HW3 (incl. quizzes, peer instructions, HWs, labs)
- Is there any practice/review?
  - Pre-lab is midterm1_sp17
  - Section on 9/21 is a review session
- Notes allowed on exam?
  - 1 double-sided 8.5"x11" sheet of notes

CS 2230
CS II: Data structures

Meeting 9: More ADTs: queues

Brandon Myers

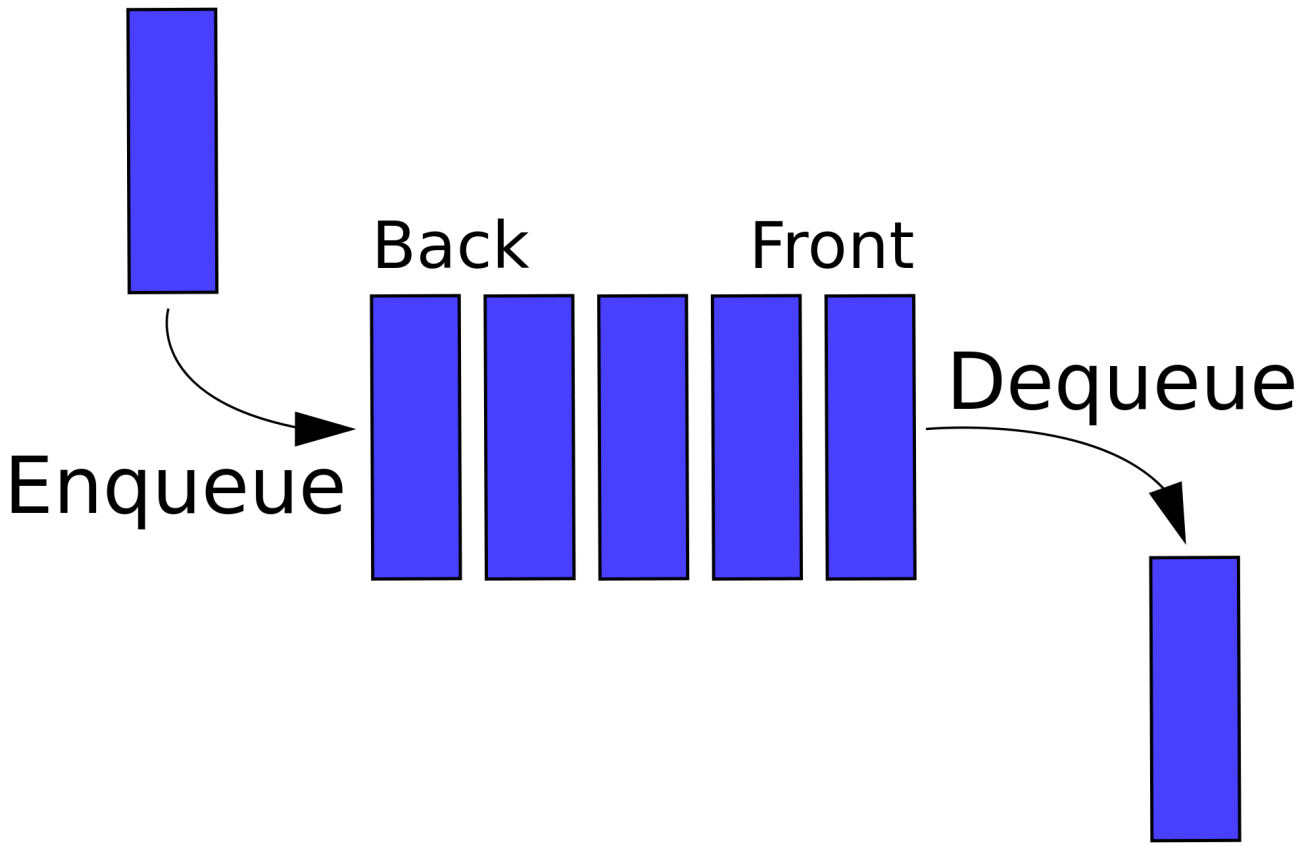
University of Iowa

Today’s big ideas

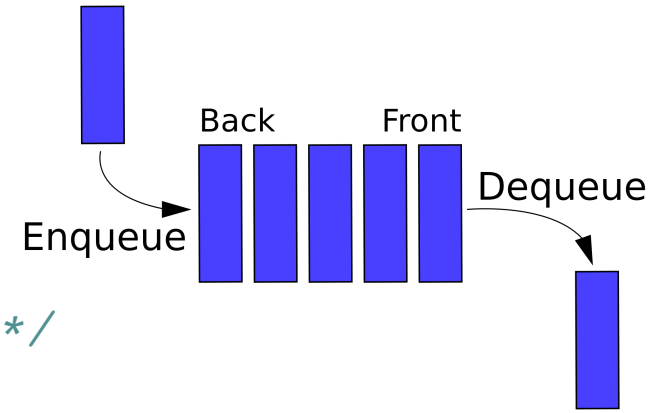
- Examine and implement two more Abstract Data Types: Stack and Queue

Queue ADT

Queues use **FIFO** order
First In First Out



https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/2000px-Data_Queue.svg.png




```
public interface Queue {
    /* Insert element at back of queue */
    public void enqueue(Object ele);


    /* Remove element from front of queue
    and return it
    */
    public Object dequeue();


    /* Return the element at the front of queue */
    public Object peek();
}
```

Peer instruction

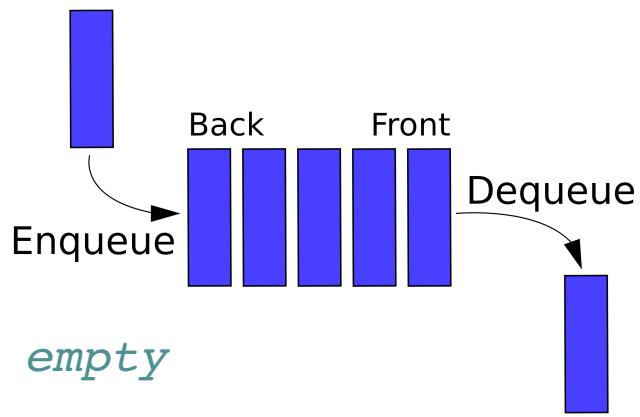
```
Queue x = <instantiate a queue>;
x.enqueue(100);
x.enqueue(22);
System.out.println(x.peek());
x.enqueue(50);
x.dequeue();
```

What is does the abstract state of the `Queue x` look like after the code runs? (front on Queue is left side)

A. [100, 22, 50]
B. [50, 22, 100]
C. [22, 50, 100]
D. [22, 50]
E. [50, 22]
F. [100, 22]
G. [22, 100]

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+

```java
public interface Queue {
    /* Insert element at back of queue */
    public void enqueue(Object ele);


    /* Remove element from front of queue
    and return it. Returns null if queue is empty
    */
    public Object dequeue();


    /* Return the element at the front of queue. Returns null
    if queue is empty */
    public Object peek();
}
```



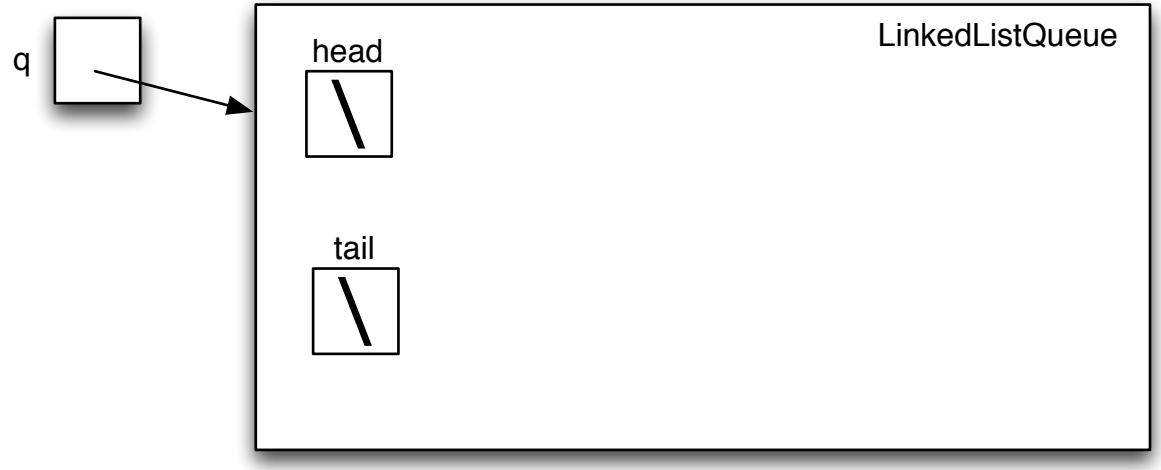
Give one way you might implement the Queue ADT:

- how would the be data stored?
- how would you find the front? the back?

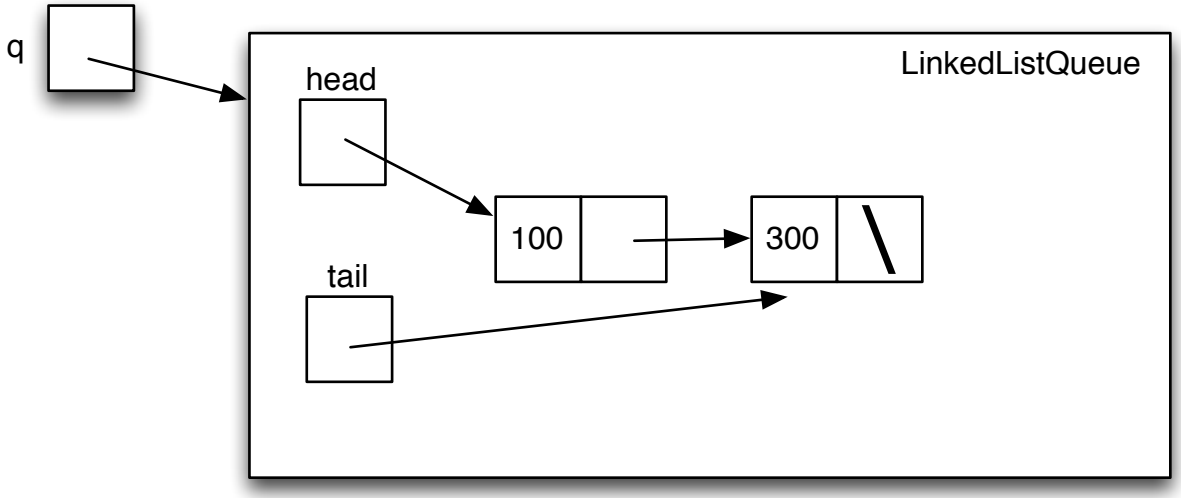
https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+

LinkedListQueue

```
Queue q = new LinkedListQueue();
```



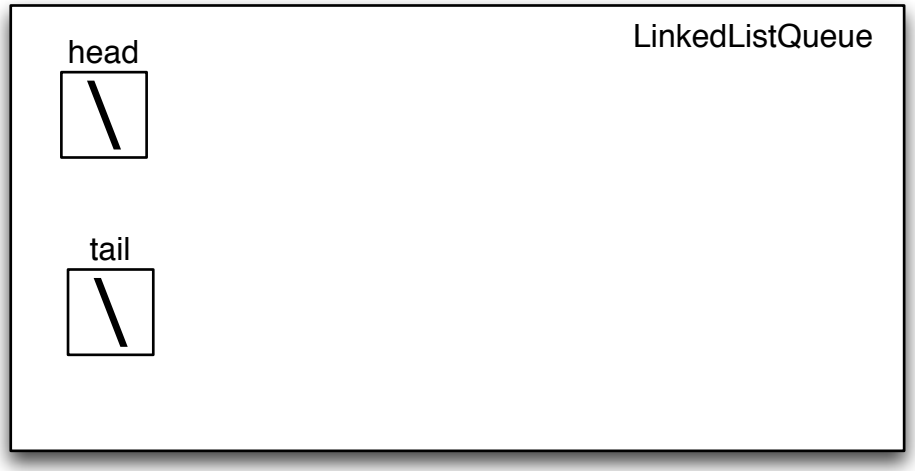

```
q.enqueue(100);
q.enqueue(300);
```



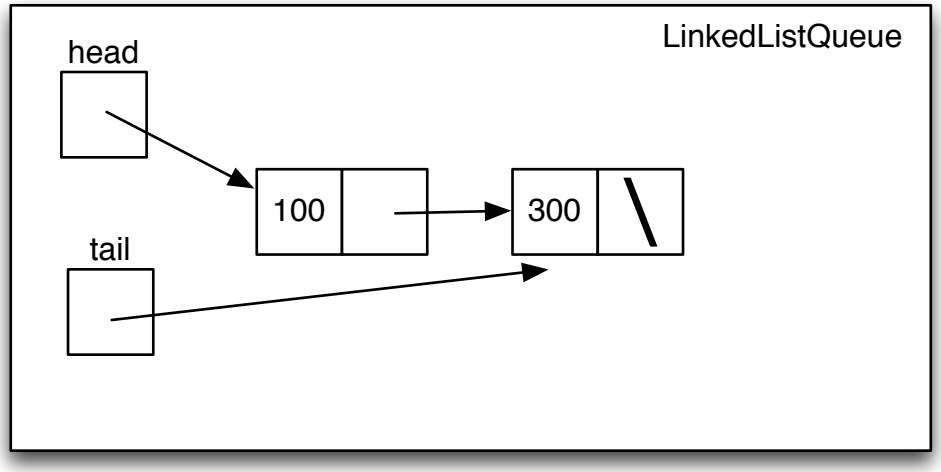

```
q.dequeue();
```

draw the LinkedListQueue now

Empty case []

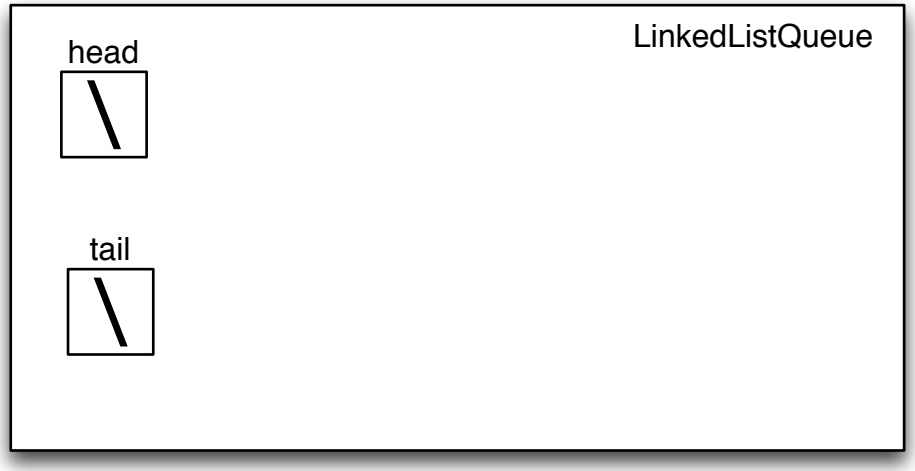


Non-empty case [100, 300]

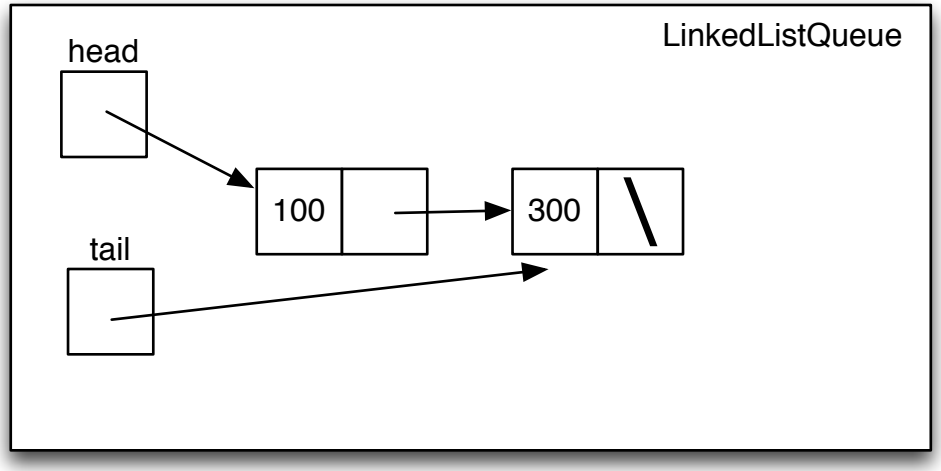



```
/* Return the element at the front of queue. Returns null
if queue is empty */
@Override
public Object peek() {

}
```

Empty case []



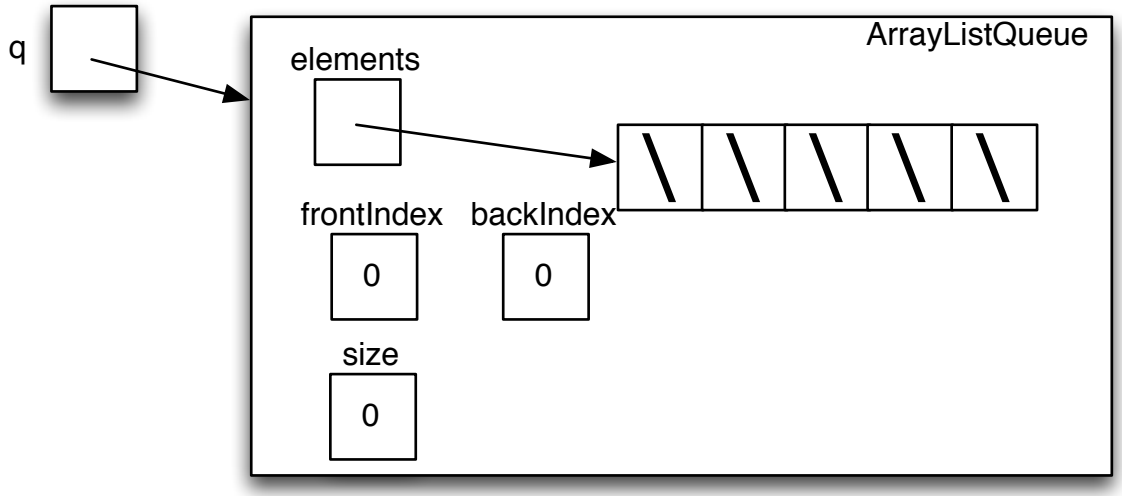
Non-empty case [100, 300]



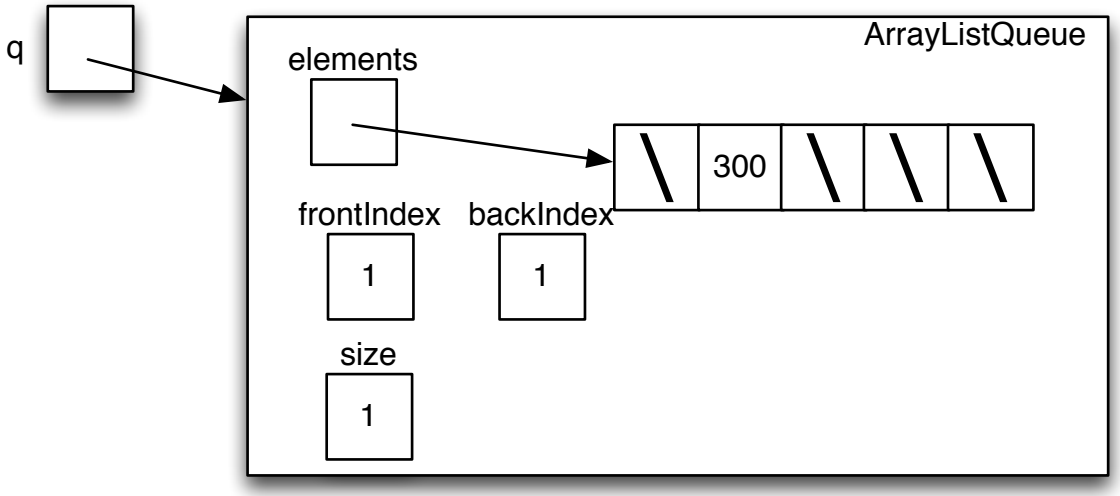

```
/* Remove element from front of queue
and return it. Returns null if queue is empty
*/
public Object dequeue() {



}
```

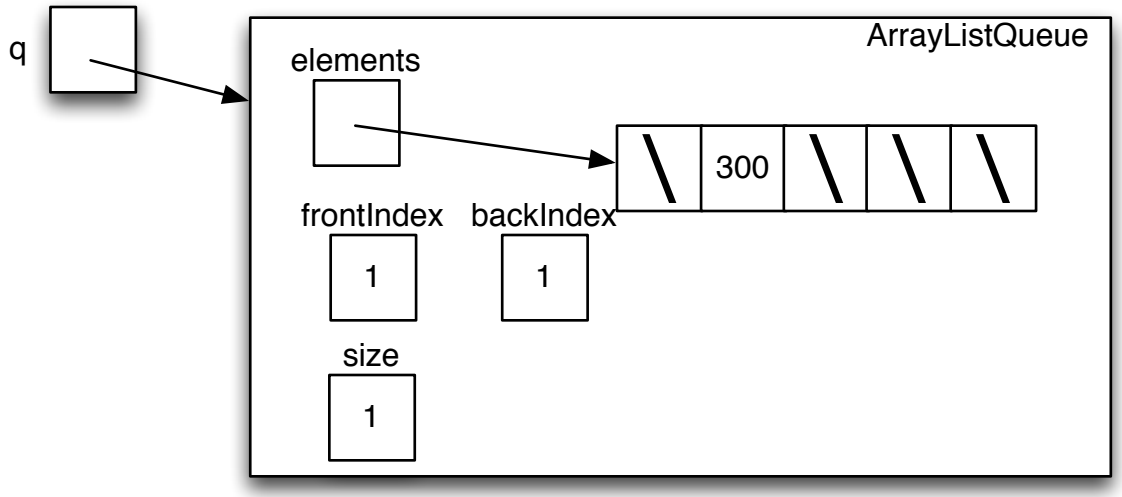
ArrayListQueue

```
Queue q = new ArrayListQueue();
```



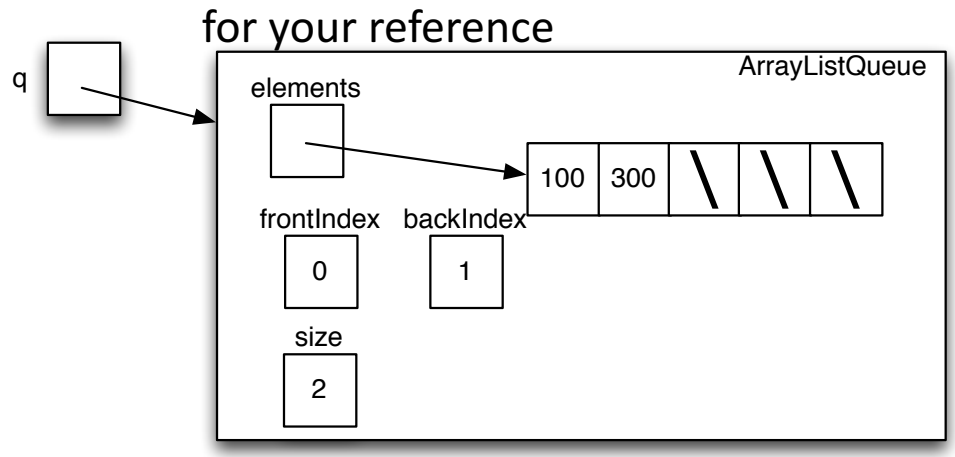

```
q.enqueue(100);
q.enqueue(300);
q,dequeue();
```






```
q.enqueue(400)
q.dequeue();
```

draw the ArrayListQueue now



How should we check if an ArrayListQueue is empty?

A. elements == null

B. frontIndex==backIndex

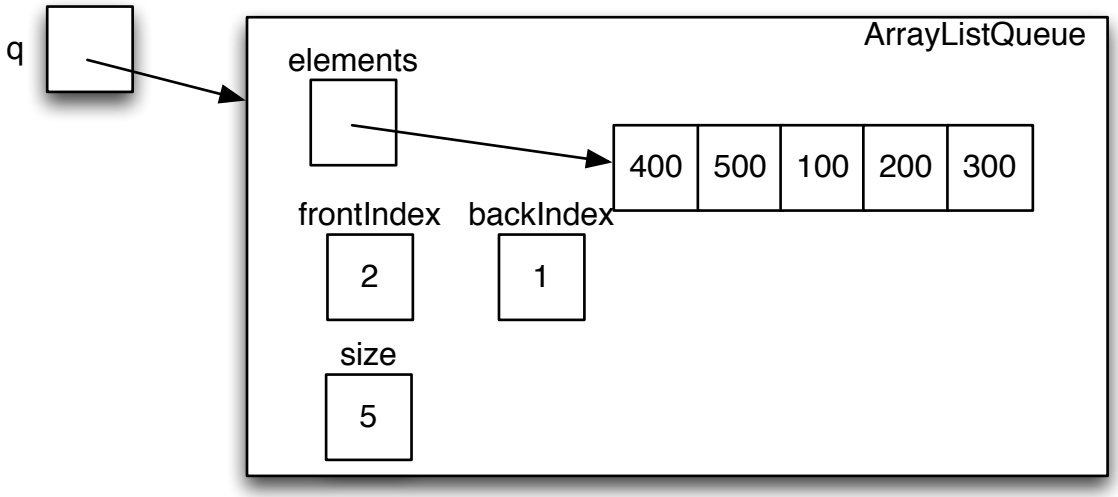
C. size==0

D. all items in the elements array are null

E. frontIndex > backIndex

F. frontIndex < backIndex

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+




```
q.enqueue(600);
```

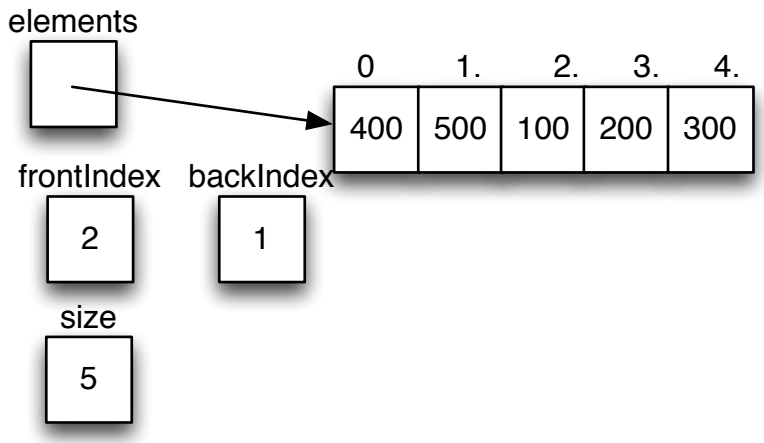
What should we do in this case?

```
public interface Queue {
    /* Insert element at back of queue */
    public void enqueue(Object ele);

    /* Remove element from front of queue
    and return it. Returns null if queue is
    empty
    */
    public Object dequeue();

    /* Return the element at the front of
    queue. Returns null if queue is empty */
    public Object peek();
}
```

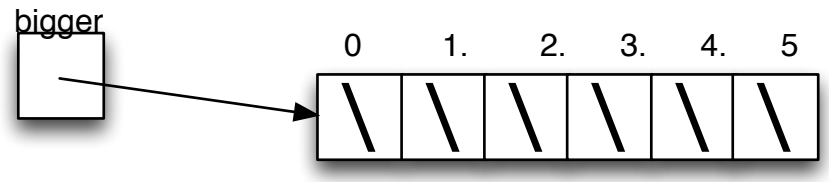
https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+

One solution to full array:
resize and move over the front

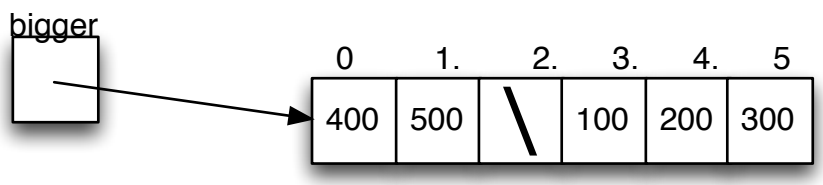
1. enqueue(600) to this



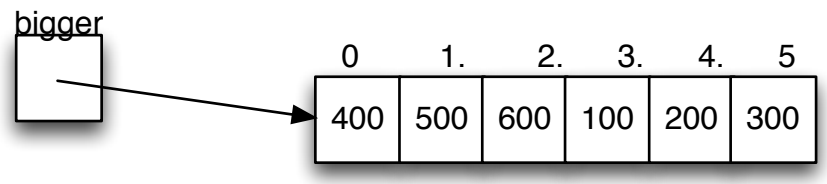
2. allocate array of length size+1



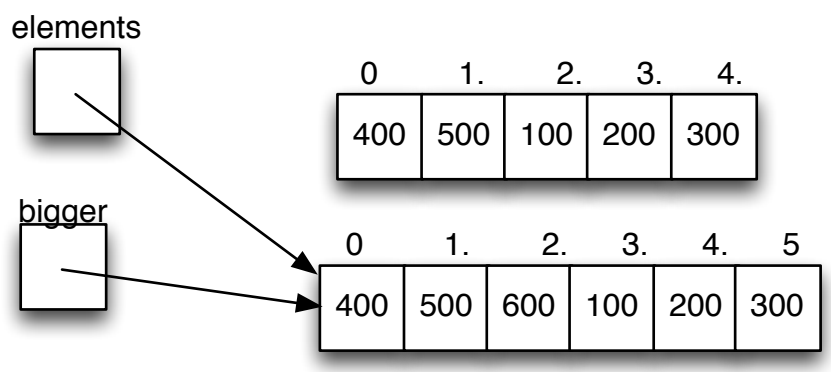
3. starting at frontIndex, copy element i to i+1; when you get to size, start at 0 and copy element i to i



4. copy the new element to backIndex+1

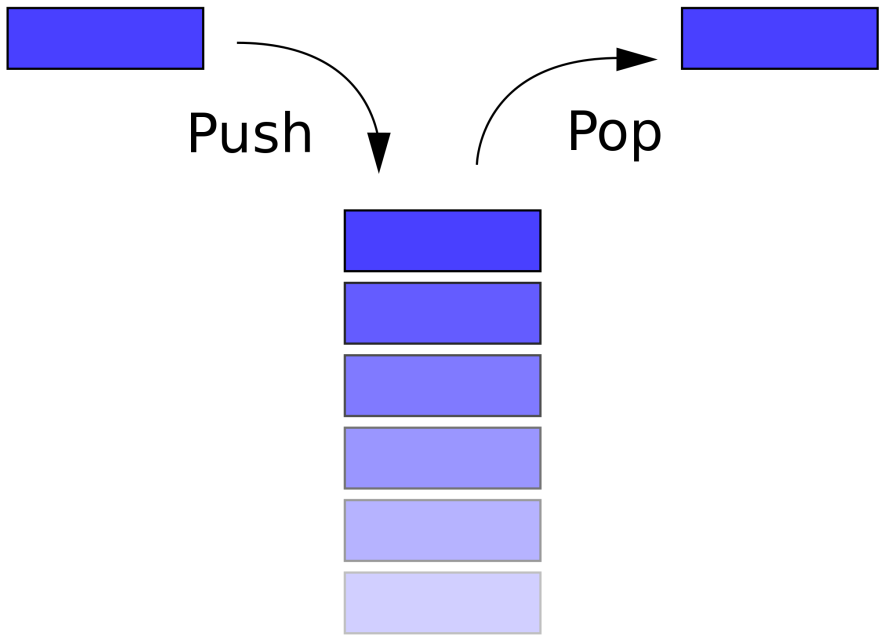


5. point `elements` to `bigger`

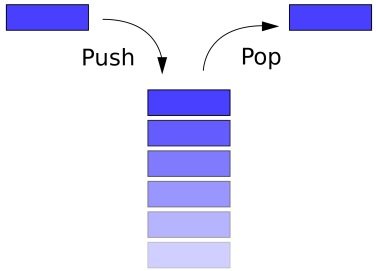


Stack ADT

Stacks use **LIFO** order
Last In First Out

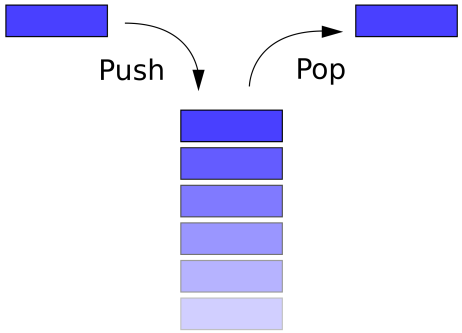


Real-life example where...



- a ***specific*** real-life process/situation/etc behaves like a stack
- or, a ***specific*** application where a stack data structure would be useful

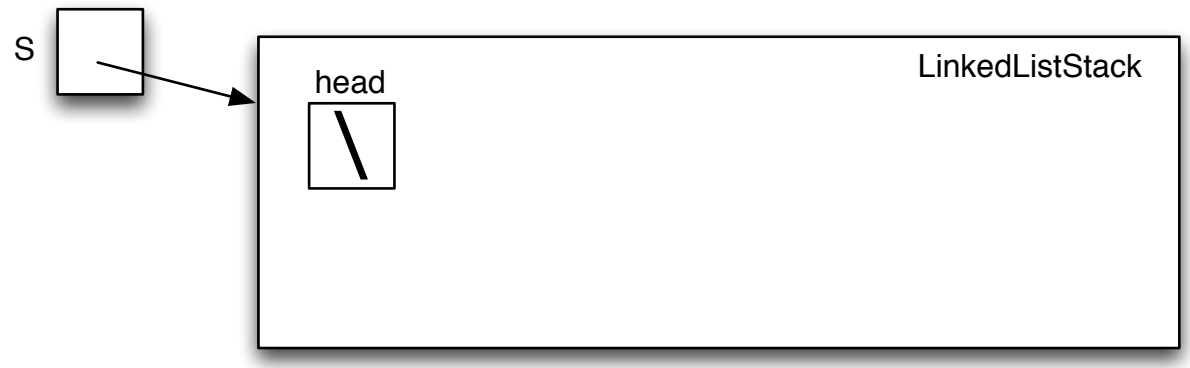
https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+



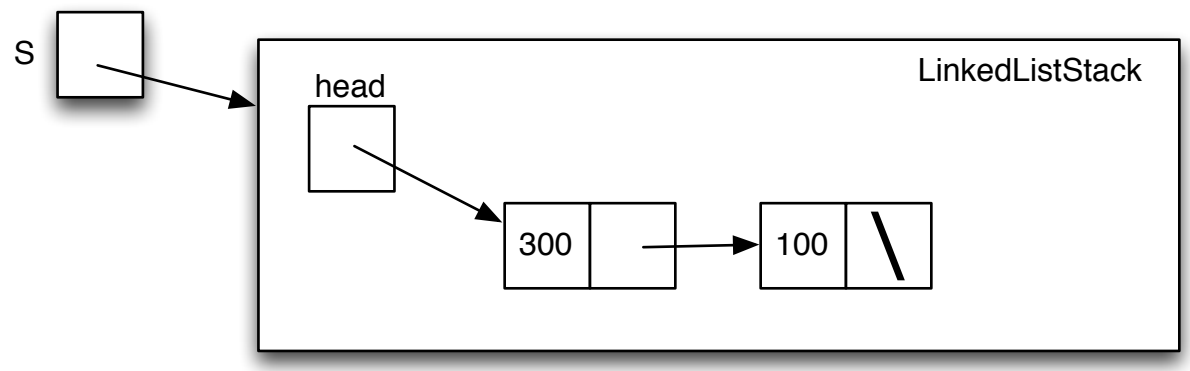

```
public interface Stack {
    /* put the element on the top of the stack */
    public void push(Object ele);


    /* remove the element on top of the stack
    and return it; Returns null if stack is empty
    */
    public Object pop();


    /* return the element on top of the stack; returns null
    if stack is empty */
    public Object peek();
}
```

LinkedListStack

```
Stack s = new LinkedListStack();
```




```
s.push(100);
s.push(300);
```

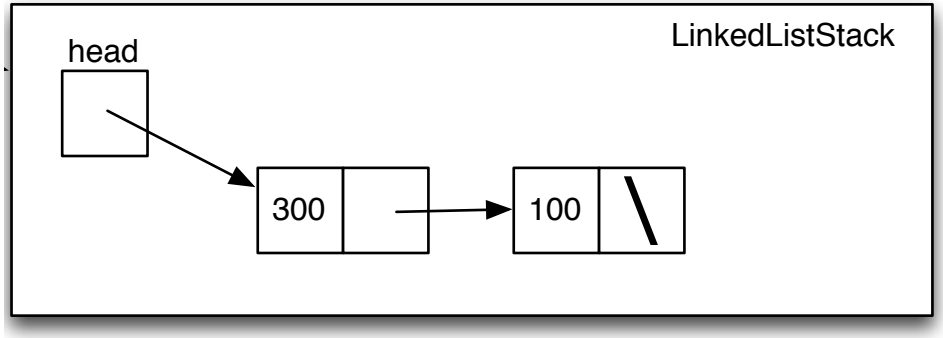



```
s.pop();
```

draw the LinkedListStack now

Empty case []

Non-empty case [300, 100]




```
/* remove the element on top of the stack
and return it; Returns null if stack is empty
*/
public Object pop();

}
```

Today’s big ideas

- Examine and implement two more Abstract Data Types: Stack and Queue