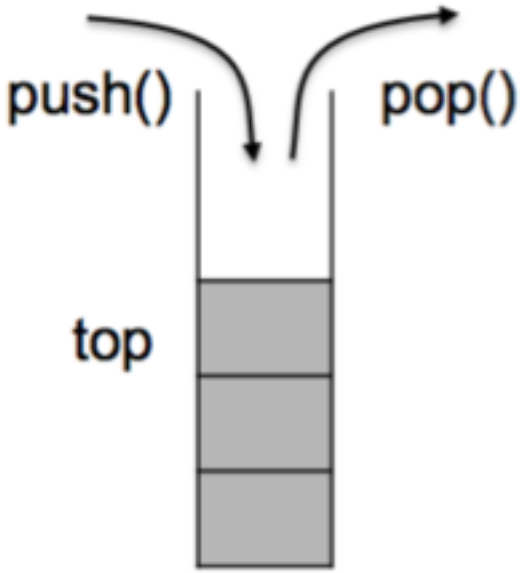
CMSC 132: Object-Oriented Programming II

Stack and Queue

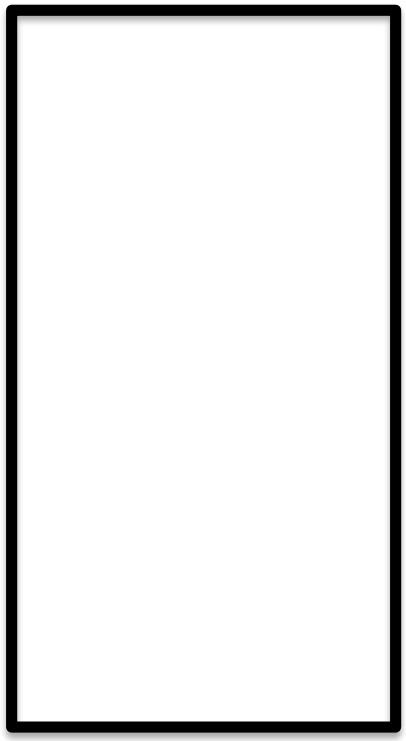
Stack

- Allows access to only the last item inserted.
- An item is inserted or removed from the stack from one end called the "top" of the stack.
- This mechanism is called Last-In-First-Out (LIFO).



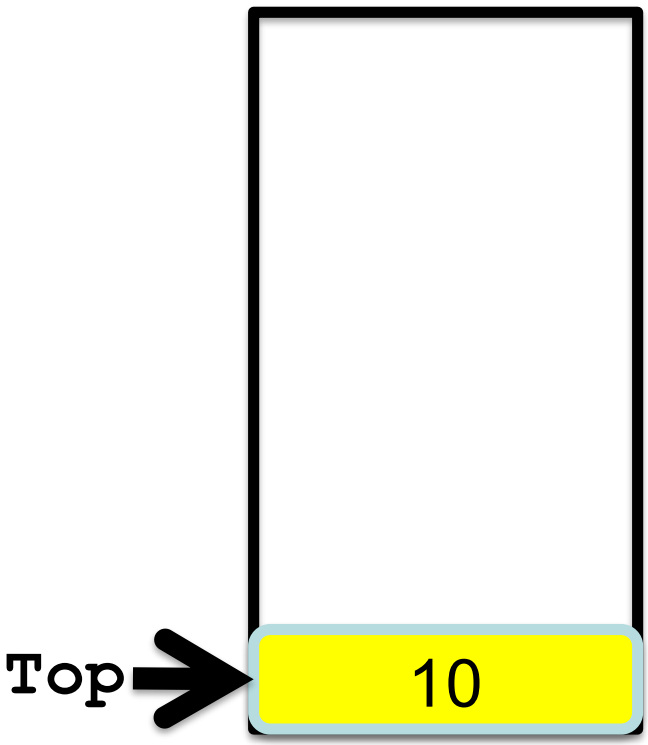
Stack Example

- Empty Stack



Stack Example

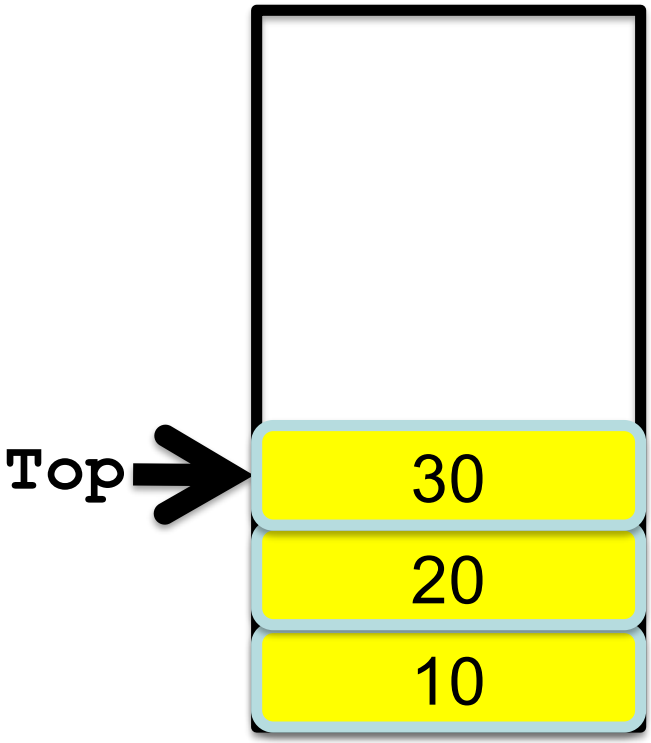
- Stack



Push 10

Stack Example

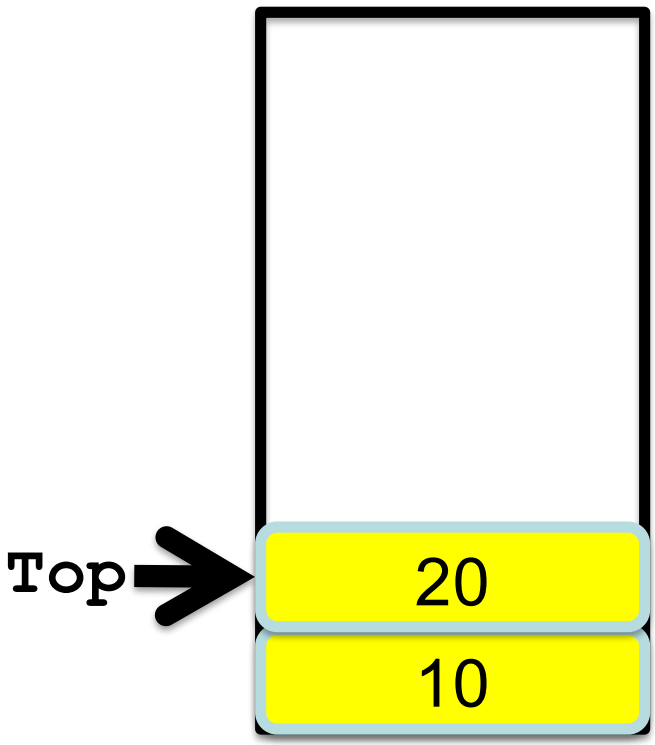
- Stack




```
Push 10

Push 20

Push 30
```

Stack Example

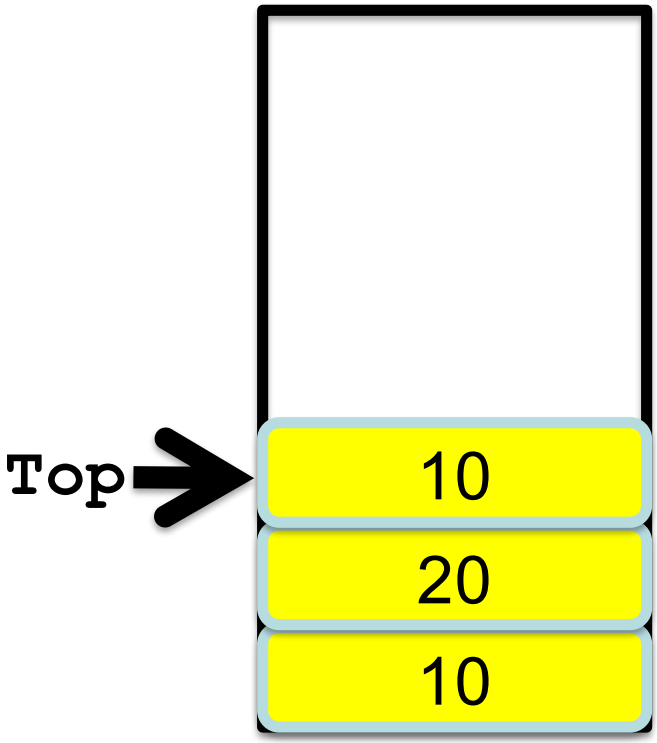
- Stack




```
Push 10

Push 20

Push 30

Pop
```

Stack Example

- Stack




```
Push 10

Push 20

Push 30

Pop

Push 10
```

Applications

- JVM stack machine
- Function Call
- Expression Evaluation
- Pushdown Automata
- Recursive → Iterative Function Conversion

Stack Interface

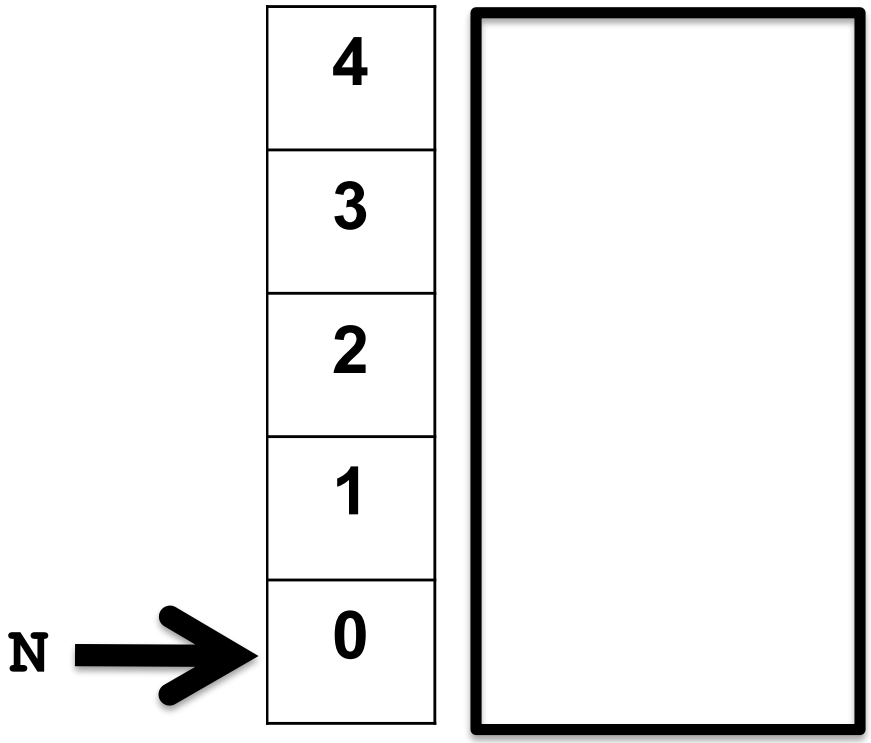
```
public interface Stack<T> extends Iterable<T>{
  void push(T t);
  T pop();// throws EmptyStackException;
  T peek() ;//throws EmptyStackException;
  boolean isEmpty();
  int size();
}
```

Implement Stack Using an Array

```
public class ArrayStack<T> implements Stack<T> {
    private T[] items;
    private int N; // number of elements in the stack
    public ArrayStack(){
        items = (T[])new Object[2];
    }
}
```

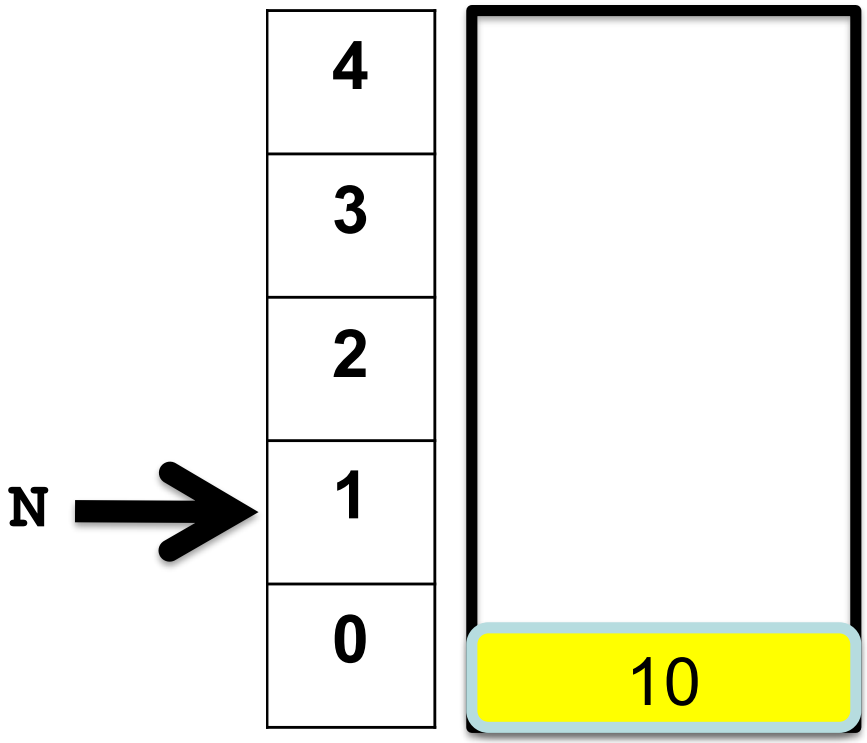
Stack Example

- Empty Stack

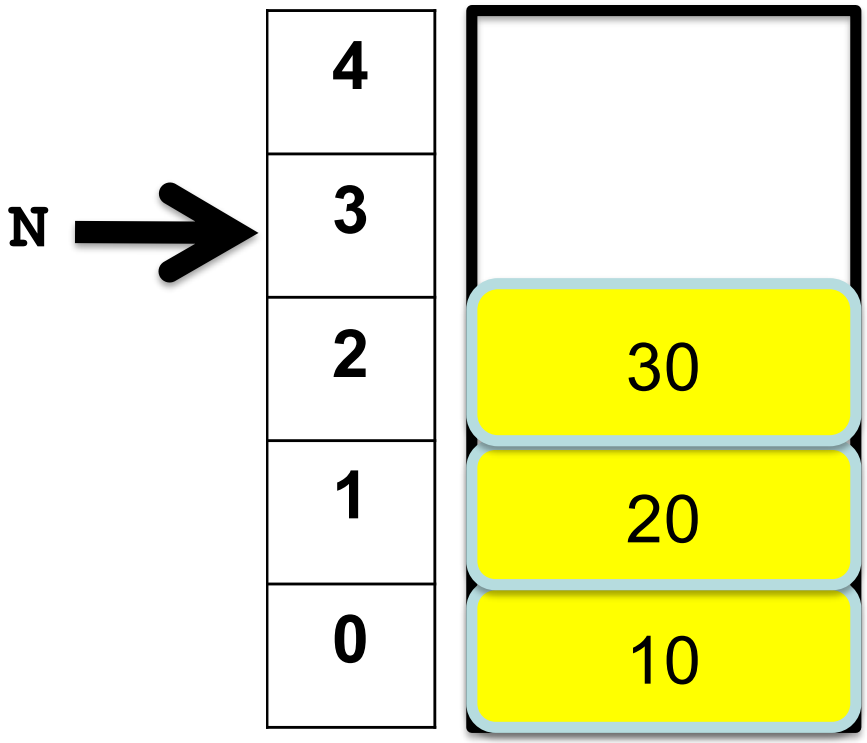


Pushing an item

Push 10



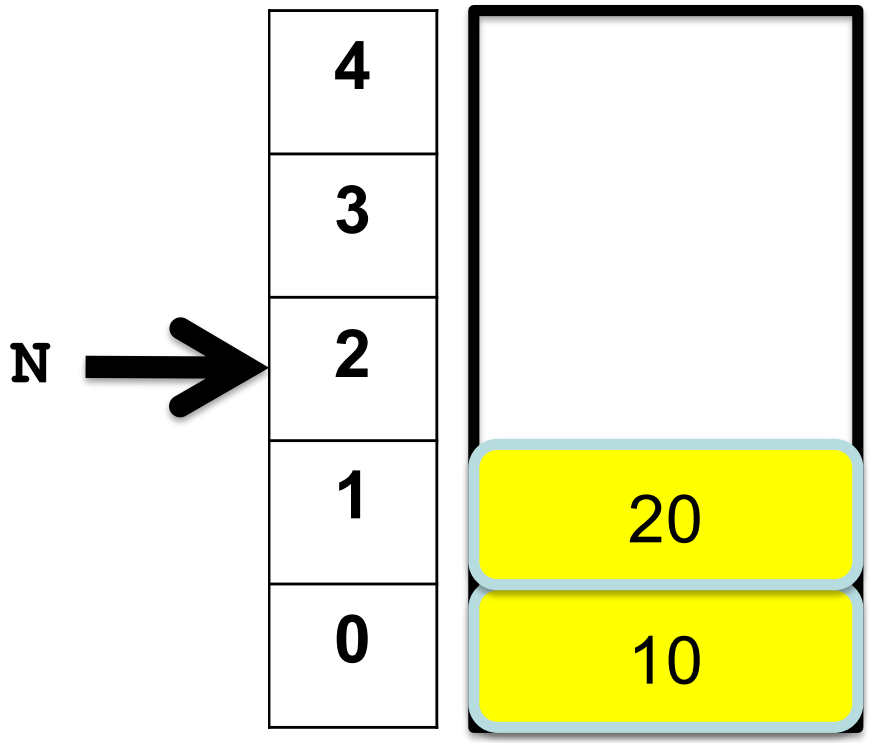
Pushing more items




```
Push 10

Push 20

Push 30
```

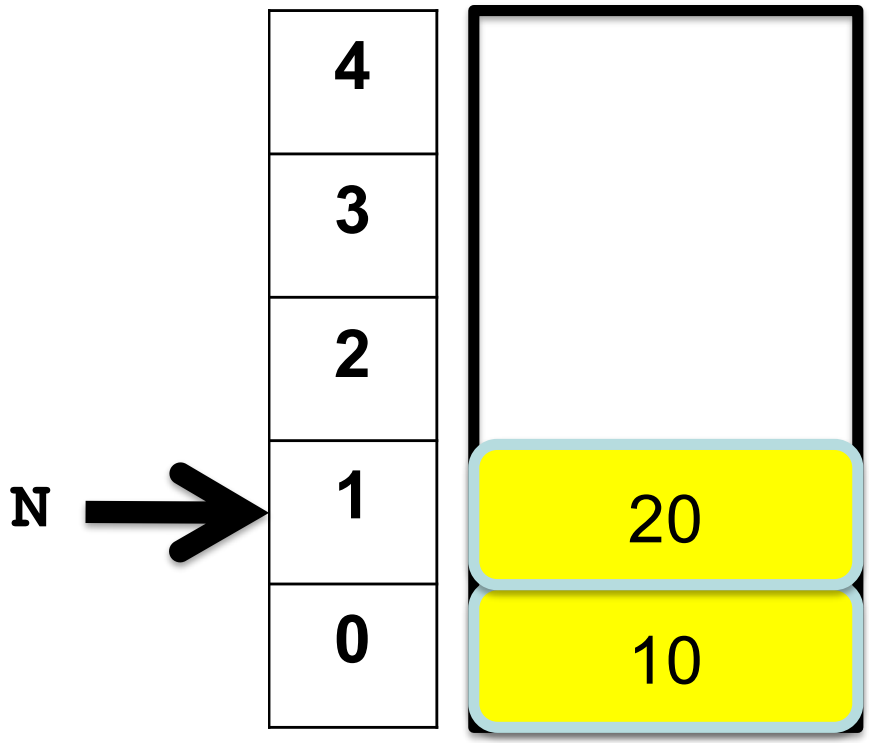
What if the array is full?

Removing an item




```
Push 10

Push 20

Push 30

Pop
```

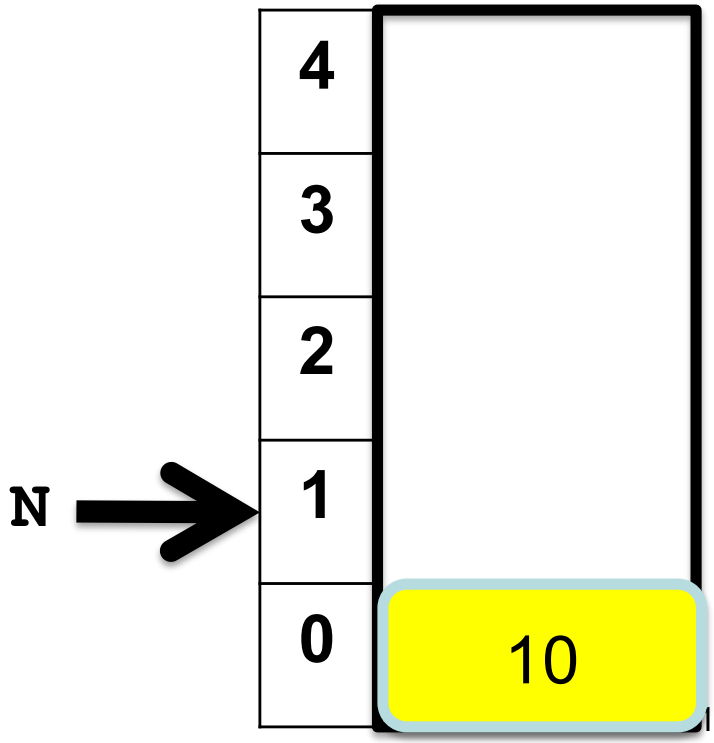
Removing an item




```
Push 10

Push 20

Push 30

Pop

Pop
```

Array Implementation: push

```
public void push(T item){
  if(N == items.length){
    resize(2 * items.length);
  }
  items[N++] = item;
}
```



Array Implementation: pop

```
public T pop(){
  if(isEmpty()) throw new NoSuchElementException();
  T item = items[--N];
  items[N] = null;
  return item;
}
```

Array Implementation: peek

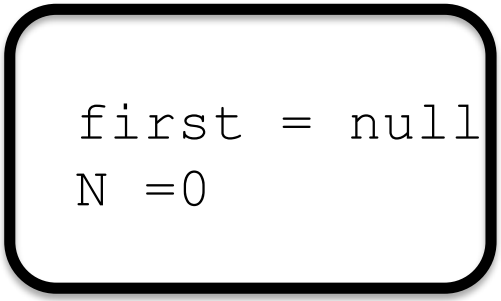
```
public T peek(){
  if(isEmpty())
    throw new NoSuchElementException("Stack Underflow");
   return items[N-1];
}
```

Implement Stack Using a Linked List

```
public class LinkedStack<T> implements Stack<T> {
  private int N;
  private Node first;

  private class Node{
    private T data;
    private Node next;
    Node(T item){
      data = item;
    }
}
```

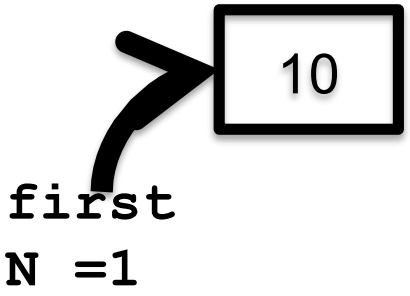
Stack Example

- Empty Stack



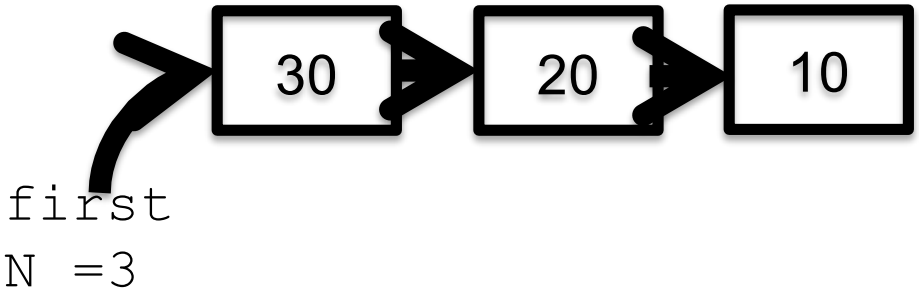
Pushing an item

Push 10




```
public void push(T item){
   Node old = first;
   first = new Node(item);
   first.next = old;
   N++;
}
```

Pushing more items



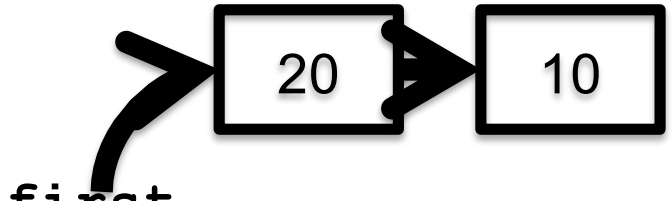
Push 10

Push 20

Push 30

```
public void push(T item){
  Node old = first;
  first = new Node(item);
  first.next = old;
  N++;
}
```

Pushing more items




```
Push 10

Push 20

Push 30

Pop
```

```
public T pop(){
  if(isEmpty()){
    throw new NoSuchElementException();
  }
  T item = first.data;
  first = first.next;
  N--;
  return item;
}
```

Other methods

Peek

```
public T peek(){
  if(isEmpty()){throw new NoSuchElementException();}
  return first.data;
}
```

size

```
public int size(){
  return N;
}
```

isEmpty

```
public boolean isEmpty(){
  return first == null;
}
```

Testing the Stack

```
public static void main(String[] args) {
    LinkedStack<Integer> ls = new LinkedStack();
    for(int i = 1; i <= 7; i++){
        ls.push(i);
    }
    System.out.println("size:" + ls.size());
    System.out.println("\n");
    while(!ls.isEmpty()){
        System.out.print(ls.peek()+",");
        System.out.print(ls.pop()+",");
    }
}
```

```
Size:7
7,7,6,6,5,5,4,4,3,3,2,2,1,1
```

Queue

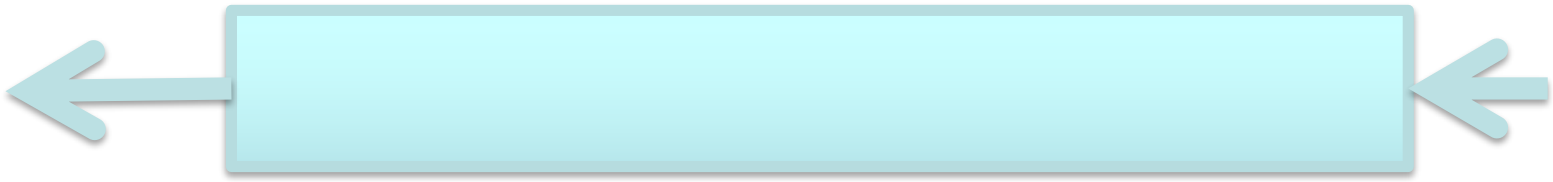
- Queue is an ADT data structure similar to stack, except that the first item to be inserted is the first one to be removed.
- This mechanism is called First-In-First-Out (FIFO).
- Placing an item in a queue: enqueue.
- Removing an item from a queue: dequeue
- Applications: printer queue, keystroke queue, etc.

Queue


```
public interface Queue<T> extends Iterable<T> {
  void enqueue(T t);
  T dequeue();// throws EmptyStackException;
  T peek() ;//throws EmptyStackException;
  boolean isEmpty();
  int size();
}
```

Queue Example

Empty Queue



Adding an item

Empty Queue

enqueue 10

Adding more items

Empty Queue

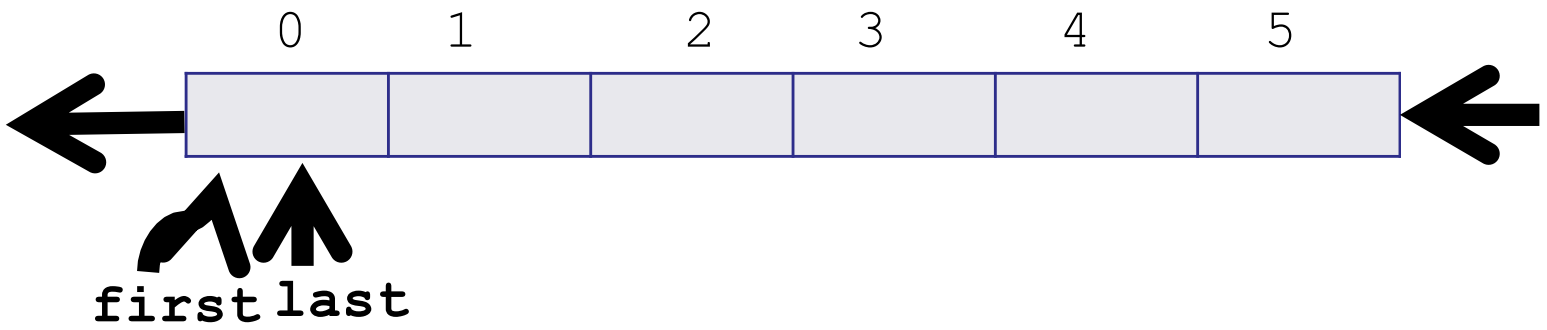

```
enqueue 10
enqueue 20
enqueue 30
```

Adding more items

Empty Queue

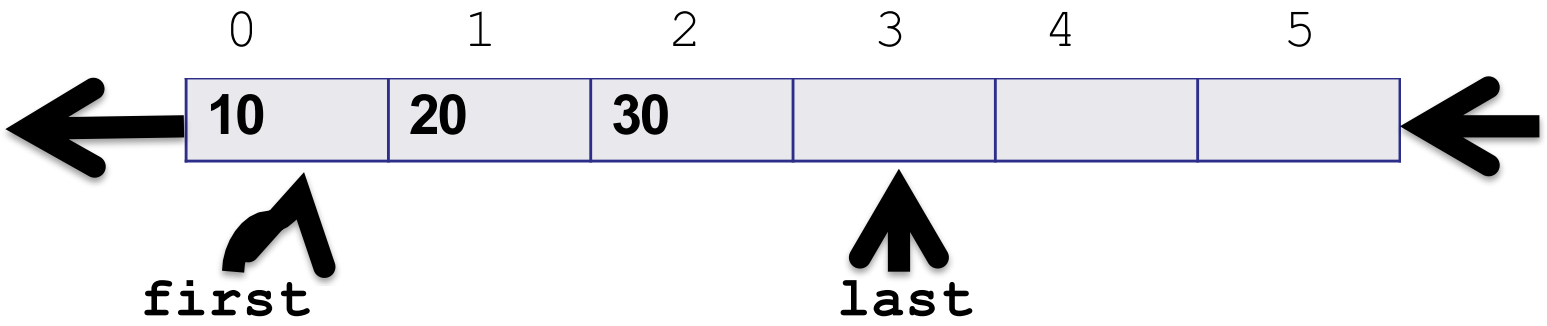

```
enqueue 10
enqueue 20
enqueue 30
dequeue
```

Circular Array Implementation



N = 0

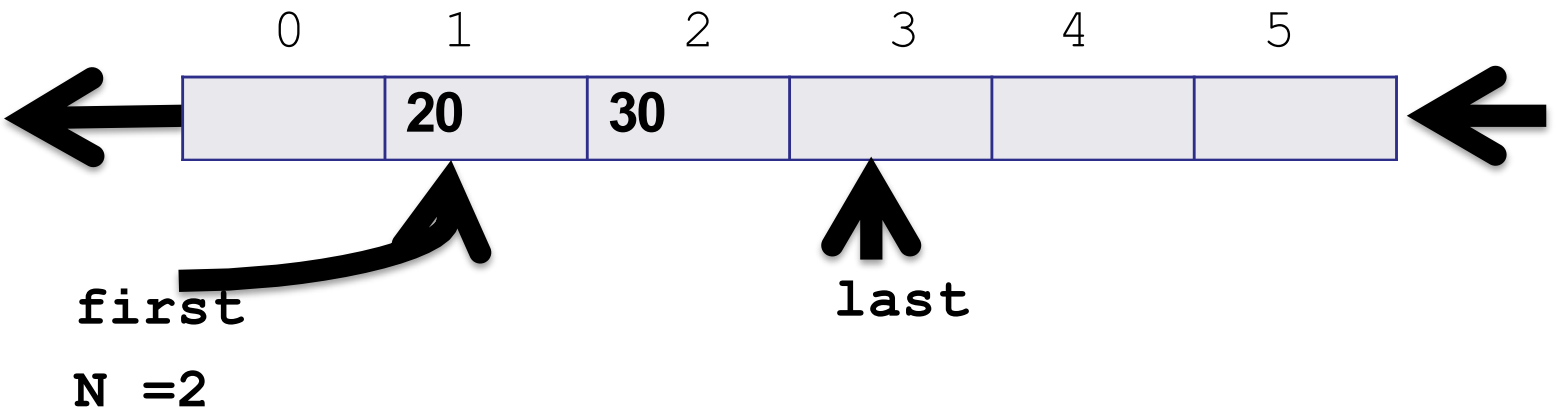
Circular Array Implementation



N =3

```
enqueue 10
enqueue 20
enqueue 30
```

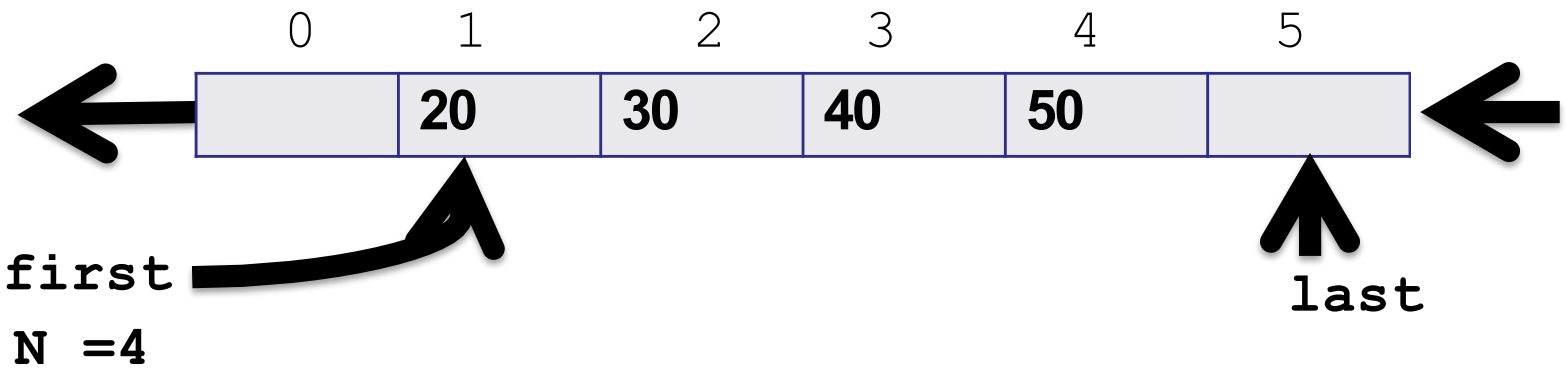
Circular Array Implementation



N =2

```
enqueue 10        dequeue
enqueue 20
enqueue 30
```

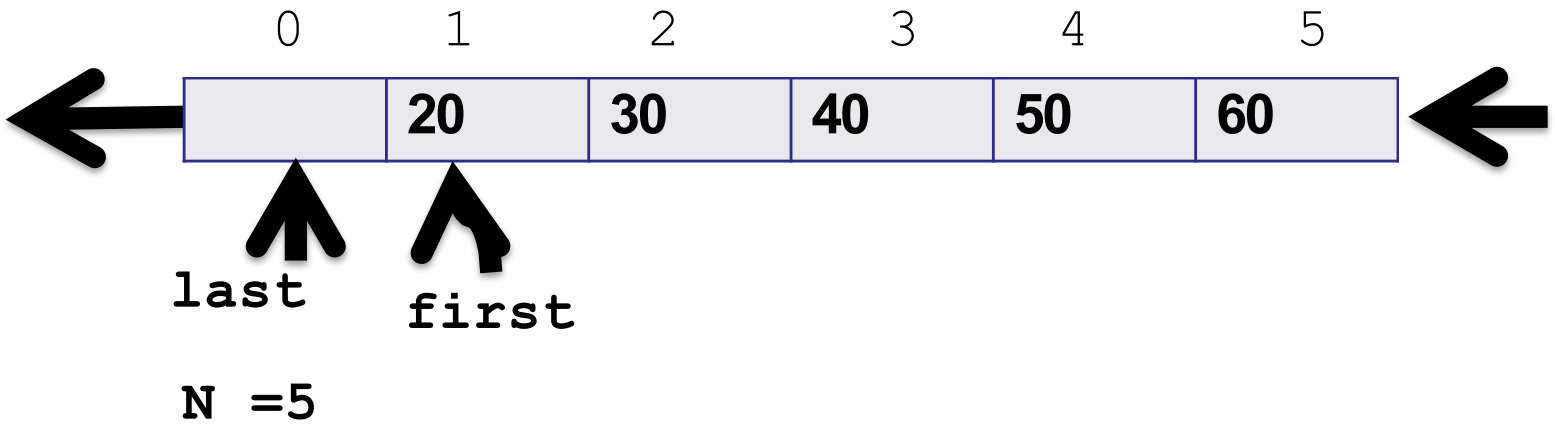
Circular Array Implementation




```
enqueue 10
enqueue 20
enqueue 30
```

```
dequeue
enqueue 40
enqueue 50
```

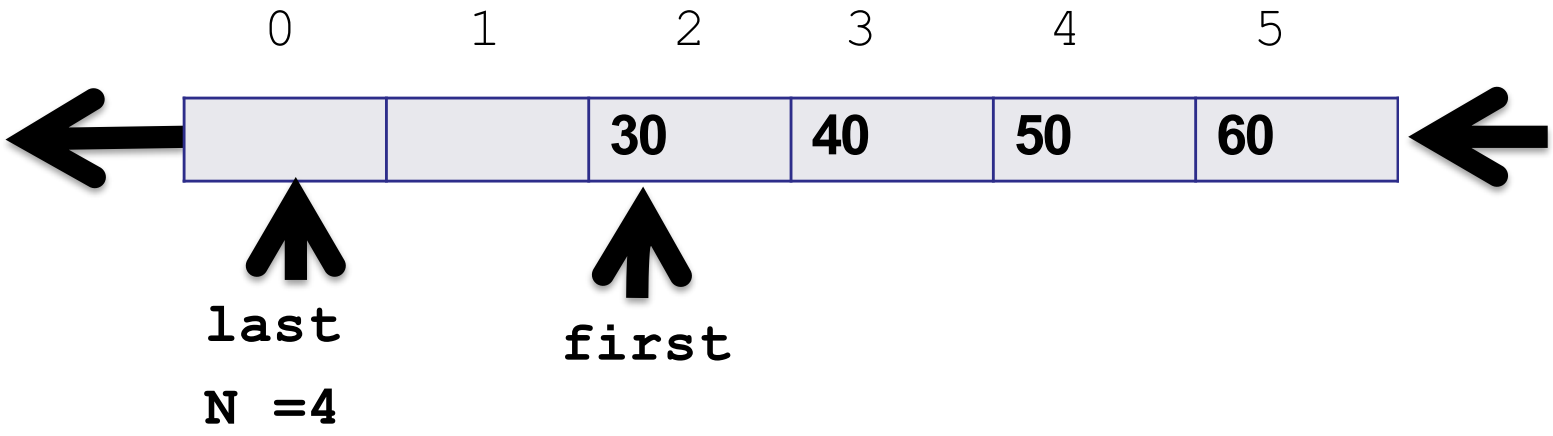
Circular Array Implementation




```
enqueue 10
enqueue 20
enqueue 30
```

```
dequeue
enqueue 40
enqueue 50
enqueue 60
```

Circular Array Implementation

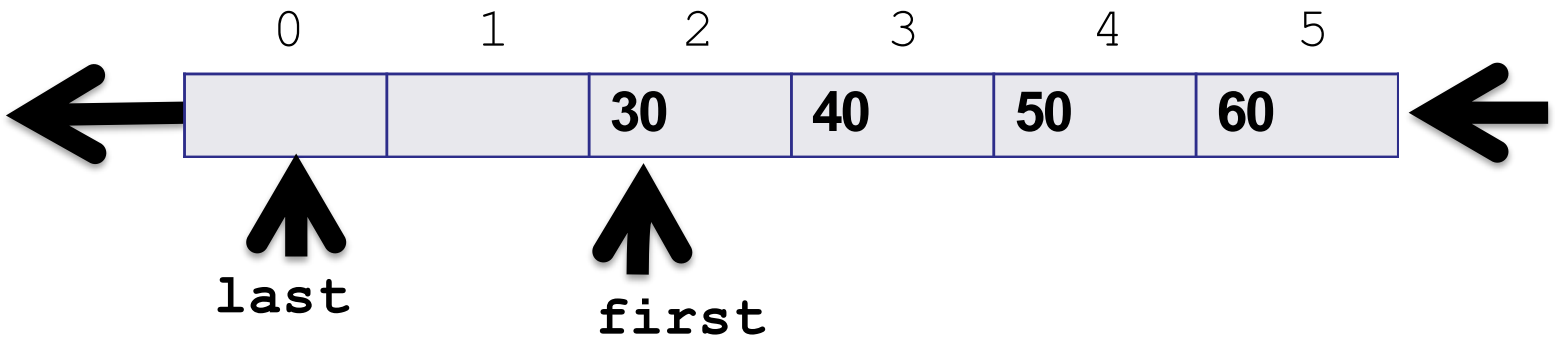



```
enqueue 10
enqueue 20
enqueue 30
```

```
dequeue
enqueue 40
enqueue 50
enqueue 60
dequeue
```

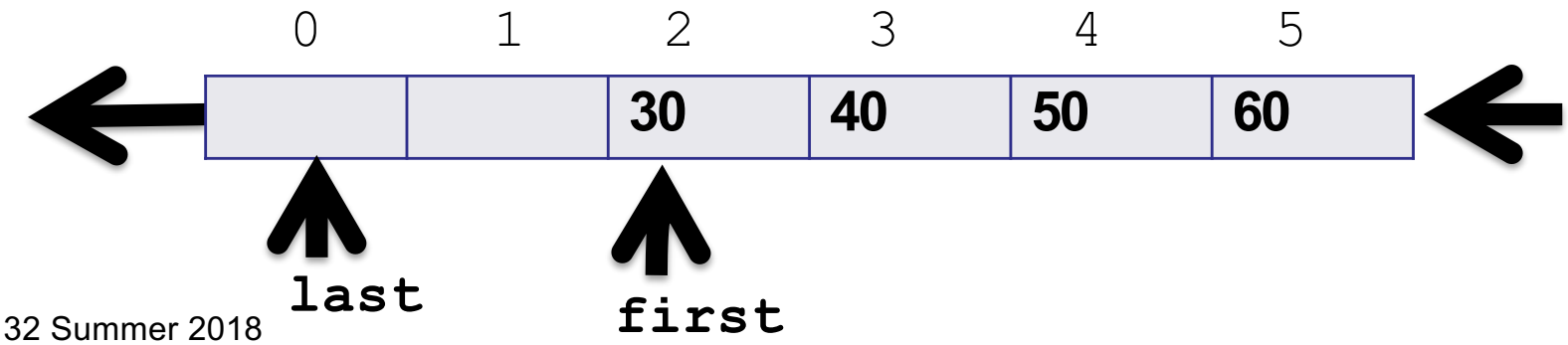
Implementing push (enqueue)

```
public void enqueue(E item) {
   if (N == q.length) resize(2*q.length);
   q[last++] = item;
   if (last == q.length) last = 0;  //wrap-around
   N++;
}
```



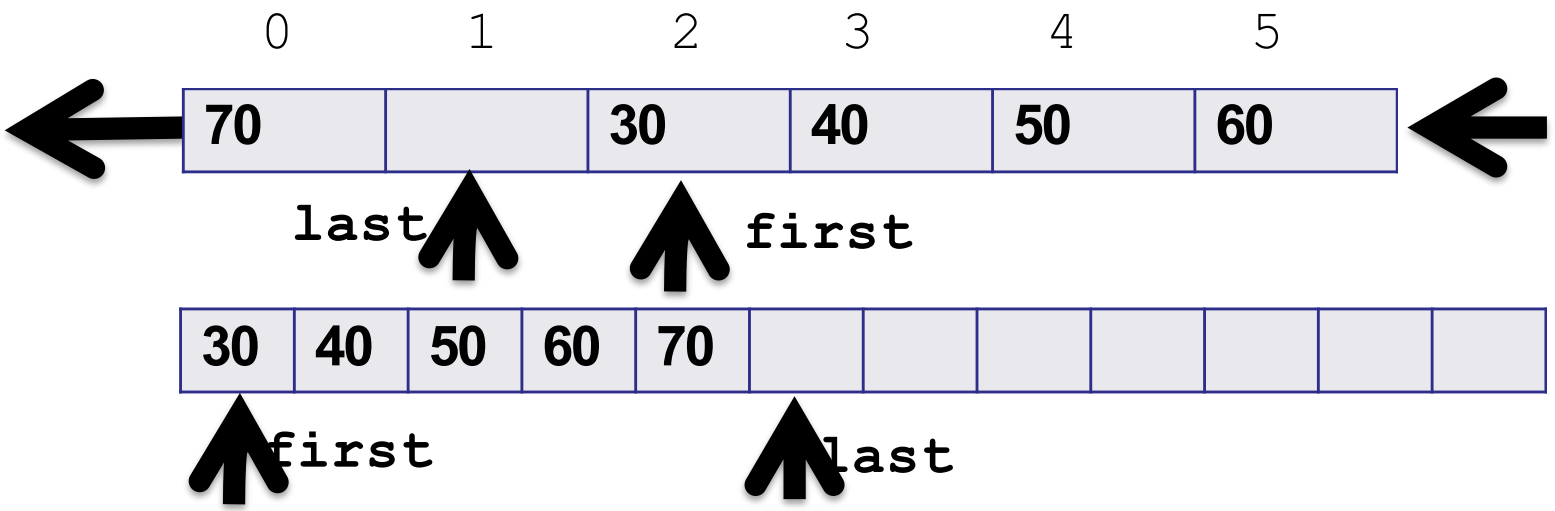
Implementing pop (dequeue)

```
public E dequeue() {
   if(isEmpty()) throw
      new NoSuchElementException("Queue underflow");
   E item = q[first];
   q[first] = null;
   N--;
   first++;
   if (first == q.length) first = 0;// wrap-around
// shrink size of array if necessary
   if (N > 0 && N == q.length/4)
         resize(q.length/2);
    return item;
}
```

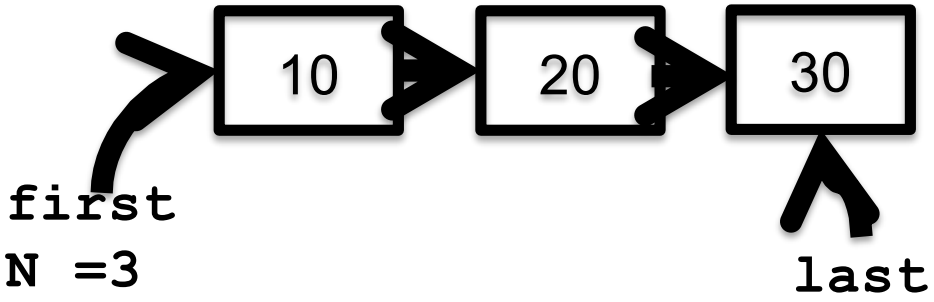


Implementing resize

```
private void resize(int max){
  assert max >= N;
  E[] temp = (E[]) new Object[max];
  for (int i = 0; i < N; i++) {
    temp[i] = q[(first + i) % q.length];
  }
  q = temp;
  first = 0;
  last = N;
}
```



Linked List Queue



- Add a new item
  - New node is added to tail (last.next)
- Remove a item
  - Remove the first node (first = first.next)