# Implementing Stacks and Queues for Lab 3

Over the rest of the semester we will implement a variety of structures. We usually want to be able to say “new X” and get a structure of type X, so that means X needs to be a class.  We will usually want to make the process of creating a structure separate from adding data to it, so we can’t just use null to represent an empty structure.  For each structure the questions are: how do we create an empty structure?  How do we  add data to the structure?  How do we remove data from the structure?

For dynamic structures we will usually make the Node structure as a class nested within the larger class.  For example, if we wanted to make a queue class we might start

```
public class MyQueue<E> {
        protected class Node {
                E data;
                Node next;
                // Node constructors and methods come here
        }
        Node head, back etc.  // Queue variables
        // Queue constructors and methods  come here
}
```

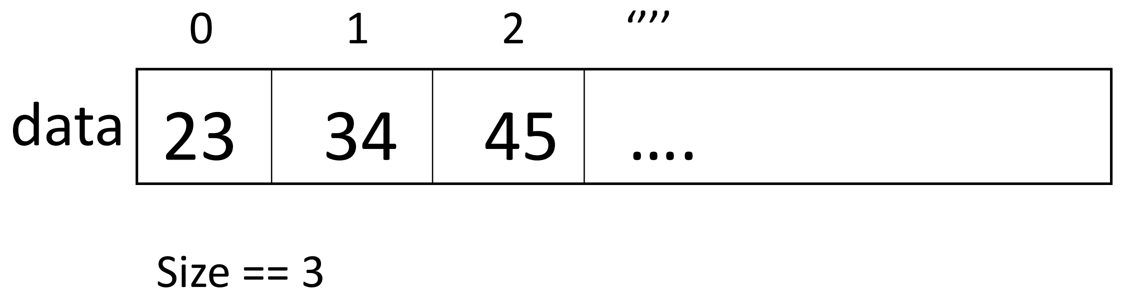
## Stacks

Remember that stacks are "Last in, first out" structures.

For an ArrayList implementation we could do this:

> To construct a Stack we will make an ArrayList *data*. The stack's *push*( ) operation corresponds to the ArrayList's *data.add*( ). The Stack's *pop* operation corresponds to data.remove(size()-1). The Stack's size() operation is just the ArrayList's size() operation; the Stack is empty when the ArrayList is empty.

For example, if we create a new Stack and push 23, 34, and 45 on it in that order, it will look like this:

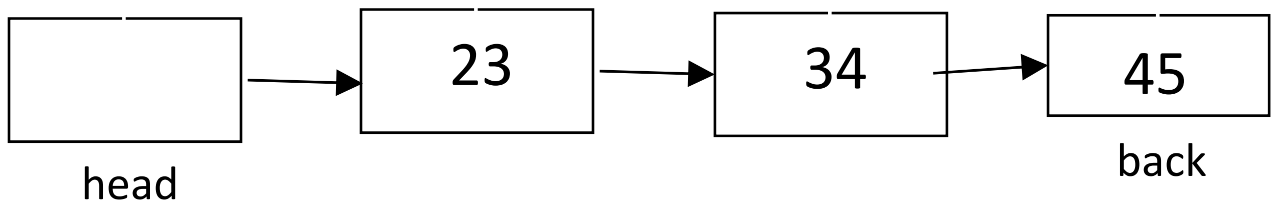


Our stack implementation will start:

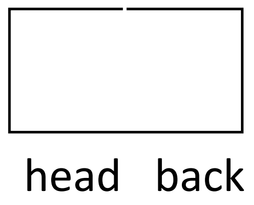
```
public class MyStack<E> implements StackADT<E>{
        private ArrayList<E> data;

        public MyStack() {
                data = new ArrayList<E> ();
        }

        public void Push( E element ){
                data.add(element);
        }
        ….
}
```

Queues are “First in, first out” structures.  Data enters at the back of the structure and leaves from the front.

The next slide shows a picture of a linked implementation of a queue.



We start an empty Queue with just the head node:



We dequeue with head.next = head.next.next.  We enqueue(x) with

```
Node p = new Node(x);
back.next = p;
back= p;
```

The only special case is that when we dequeue the last node in the queue we need to set   back=head

```
public class MyQueue<E> implements QueueADT<E> {
        class Node {
                E data;
                Node next;
                Node( E elt) {
                        data = elt;
                        next = null;
                }
                Node() {
                        this( null );
                }
        }
        Node head, back;
        int size;
        // continued next  slide
        public MyQueue( ) {
                head = new Node();
                back = head;
                size = 0;
        }

        public enqueue(E x ) {
                Node p = new Node(x);
                back.next = p;
                back = p;
                size += 1;
        }


        // etc
}
```