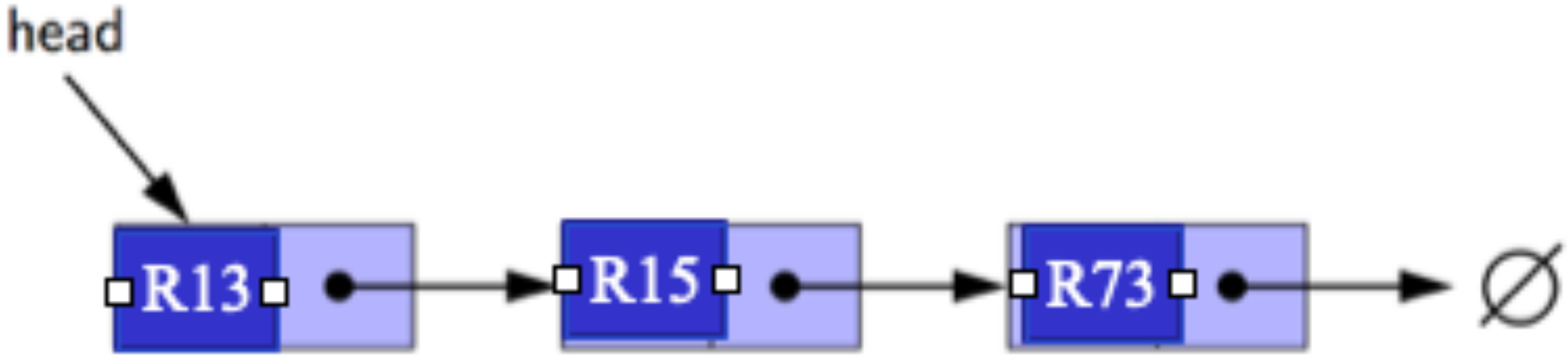
Linked Lists[2]

Java Aside -- Static Methods

- Only use when output depends only on parameters
    - Or main()
        - whcih really does depend on on args

In Pictures



Linked List Interface

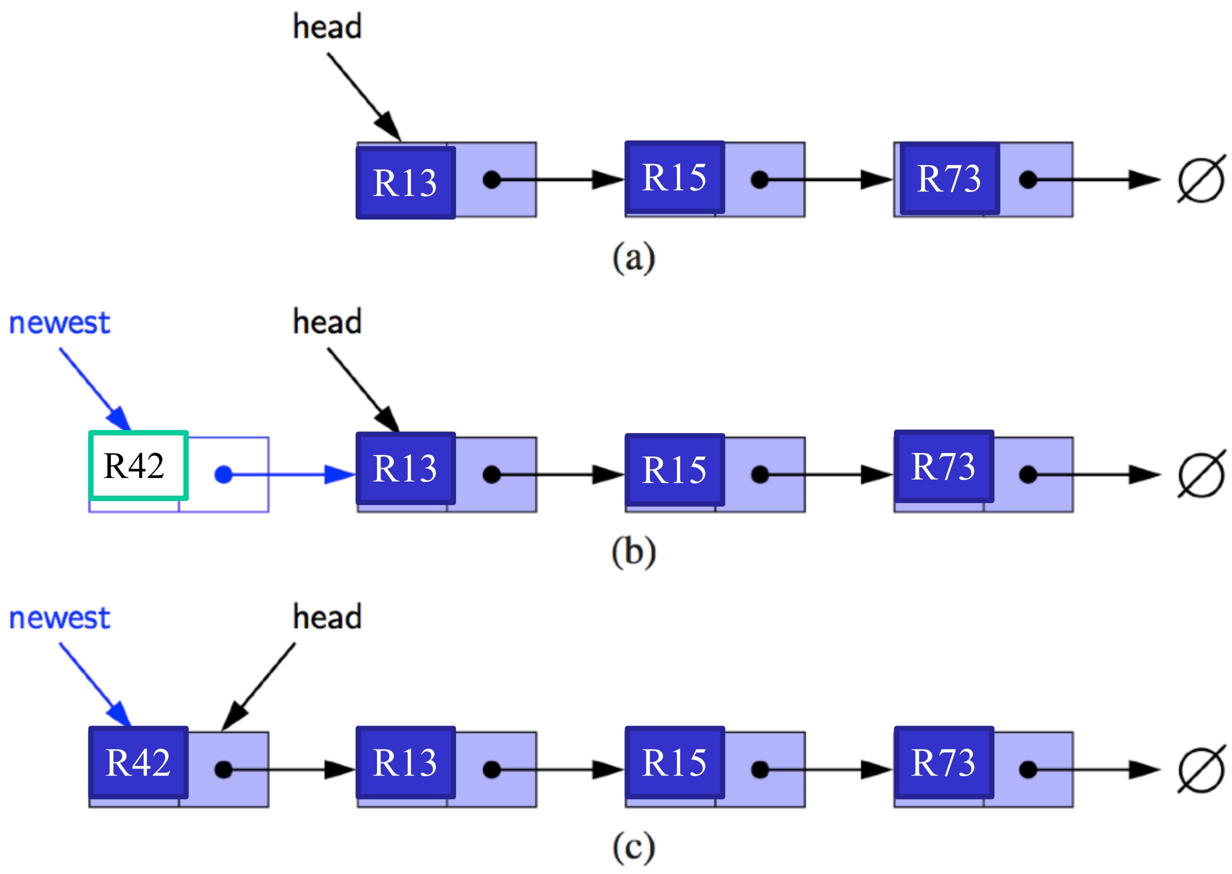
```
public interface LinkedListInterface<M>
{
   int size();
   boolean isEmpty();
   M first();
   M last();
   void addLast(M c);
   void addFirst(M c);
   M removeFirst();
   M removeLast();
   boolean remove(M r);
}
```

Starting Point

```
public class SingleLinkedList<J> implements LinkedListInterface<J>
{
    protected class Node<H> {
        public H data;
        public Node<H> next;
        public Node(H data) {
           this.data = data;
           this.next = null;
       }
    }
    int size = 0;
    protected Node<J> head = null;
    protected Node<J> tail = null;
```

Inserting at the Head

1. create a new node
2. have new node point to old head
3. update head to point to new node



write addFirst at chalkboard

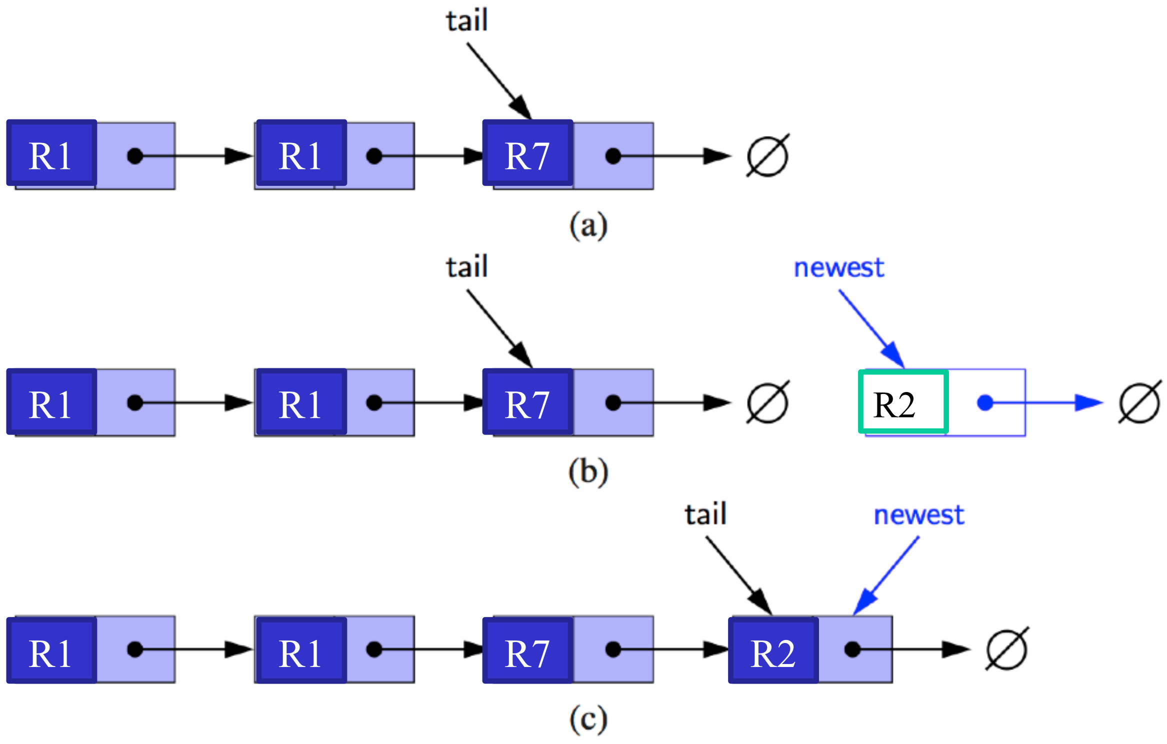
Add first -- code

```
protected Node<J> head = null;
protected class Node<H> { ... }


public void addFirst(J c) {




}
```

Printing a LL

Inserting at the Tail

1. Get to the end
    1. O(n)
    2. Save time, add an instance variable "tail"
2. Create a new node
3. Have new node point to null
4. have old last node point to new node
5. update tail to point to new node



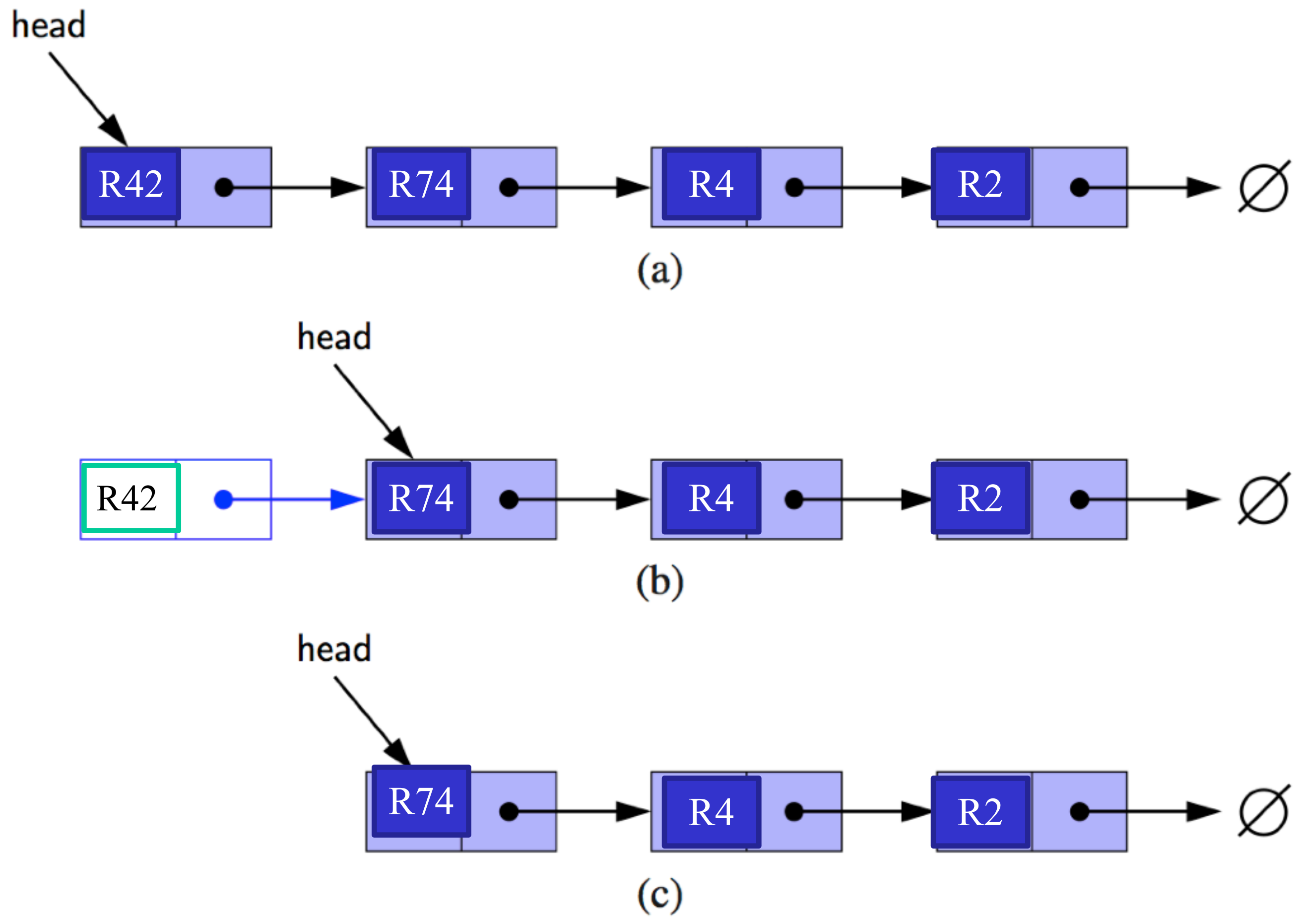
`void addLast(J c);`

addLast() WITHOUT tail pointer

```
private Node<J> lastNode() {









}
public void addLast(J c) {









}
```

Removing at the Head

1. update head to point to next node in the list
2. allow "garbage collector" to reclaim the former first node



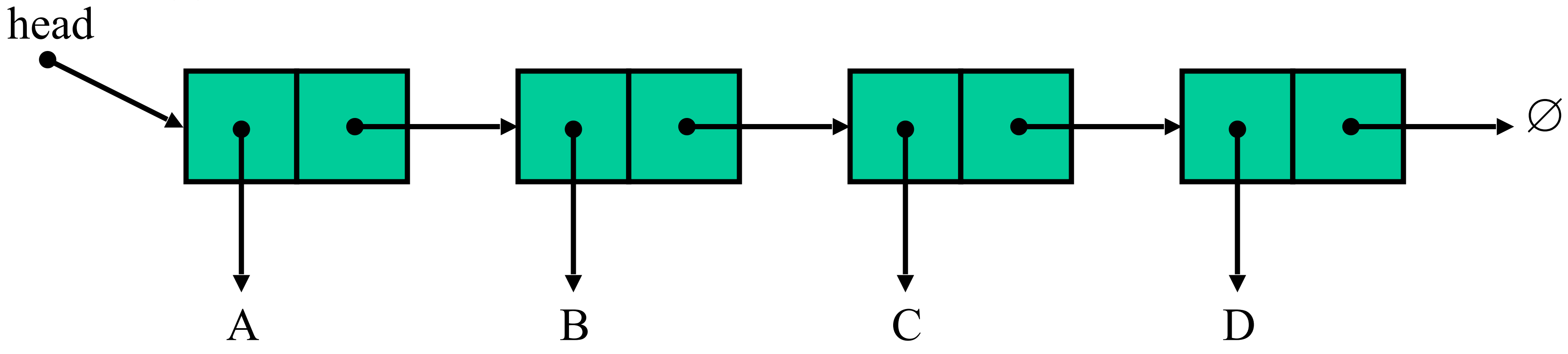
Deletion
remove the first node

1. Check if the list is empty
2. Remember the data in the current head
3. Set the head to the next item in the list
4. return data that you remembered

```
public J removeFirst() {
    if (head == null)
        return;
    J tmp = head.data;
    head = head.next;
    return tmp.data;
}
```

removeLast()

- Problem
  - How do you remove the last
  - Can we use the lastNode utility function?
    - Not exactly, because to remove D we need to do things to C
    - Cannot go backwards!!
- So, need to search forward in list to find the node before the last node



Remove Last

- Problem, to remove the last item need to do things to the next to last item so the "tail" pointer is not enough!
  - Why?
- What are the "easy" cases?
- curr.next.next
  - !!!
  - ???

```
public J removeLast() {
    if (head == null)
        return null;
    if (head.next == null) {
        // only one item in list
        J tmp = head.data;
        head = null;
        tail=null;
        size=0;
        return tmp;
    }
    Node<J> curr = head;
    while (curr.next.next != null) {
        curr = curr.next;
    }
    J tmp = curr.next.data;
    curr.next = null;
    size--;
    return tmp;
}
```

Linked Lists and Queues

```
public class QueueLL<Q> implements QueueInterface<Q>{
    private SingleLinkedList<Q> underlyingLL;
    public QueueLL() {
        underlyingLL = new SingleLinkedList<Q>();
    }
    @Override
    public int size() {
        return underlyingLL.size();
    }
    @Override
    public boolean isEmpty() {
        return underlyingLL.isEmpty();
    }
public Q peek() { return underlyingLL.first(); }
public boolean enqueue(Q e) { underlyingLL.addLast(e); return true; }
public Q dequeue() { return underlyingLL.removeFirst(); }
public void clear() { underlyingLL == new SingleLinkedList<Q>(); }
```

Linked Lists and Stacks

- Similar to Queue
- Add and remove on the head side of the LL!!

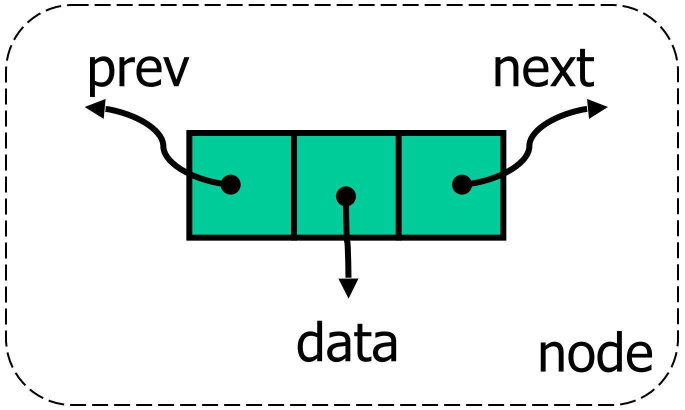
Musings on singly linked lists

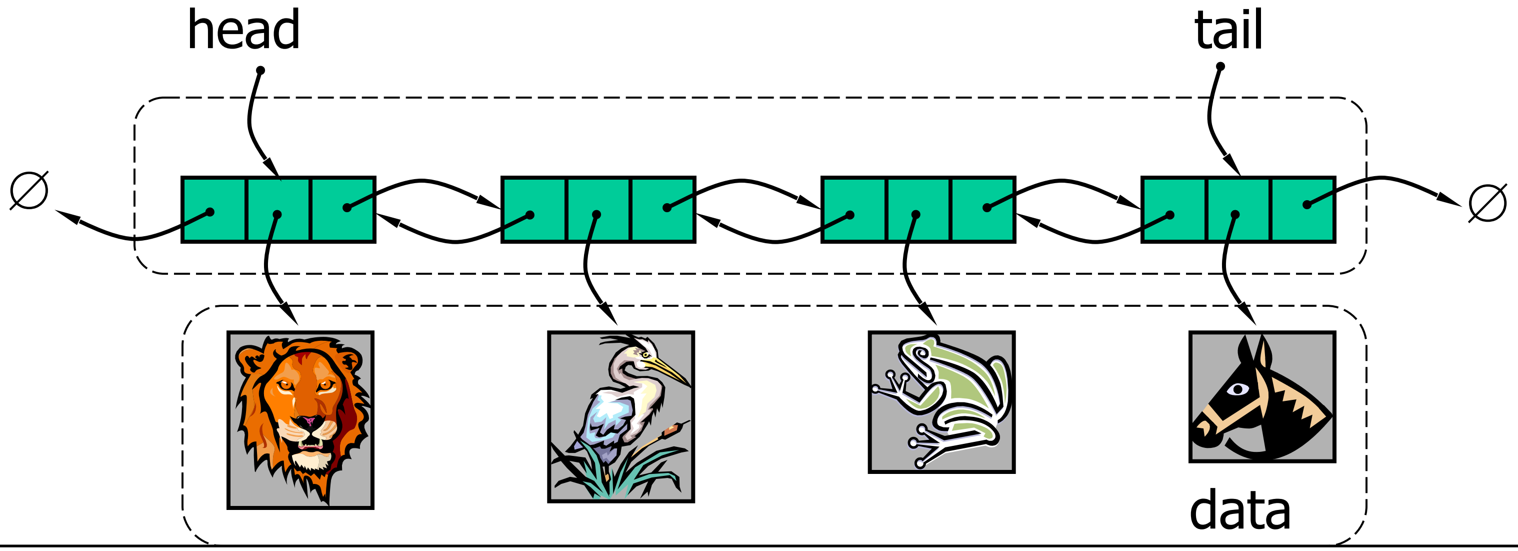
- The whole remove last method is a pain
  - and it is slow O(n)
- Not being able to go backward is a pain
  - Linked lists are a pain

- Can't do anything about linked lists be ing a pain

Doubly Linked List

- Can be traversed forward and backward
- Nodes store an extra reference





Double Linked List interface

```
public interface LinkedListInterfaceComp<E extends Comparable<E>> {
    int size();
    boolean isEmpty();
    E first();
    E last();
    void addLast(E c);
    void addFirst(E c);
    E removeFirst();
    E removeLast();
    E remove(E r);
    boolean contains(E iD);
}
```

This could also be applied to a single linked list (or an array list) or …

Node & DLL start

```
public class DoubleLinkedList<T extends Comparable<T>> implements
LinkedListInterfaceComp<T> {
    protected class Node<V extends Comparable<V>> {
       public V data;
       public Node<V> next;
       public Node<V> prev;
       public Node(V data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }
   private Node<T> head = null;
   private Node<T> tail = null;
   private int size = 0;
```

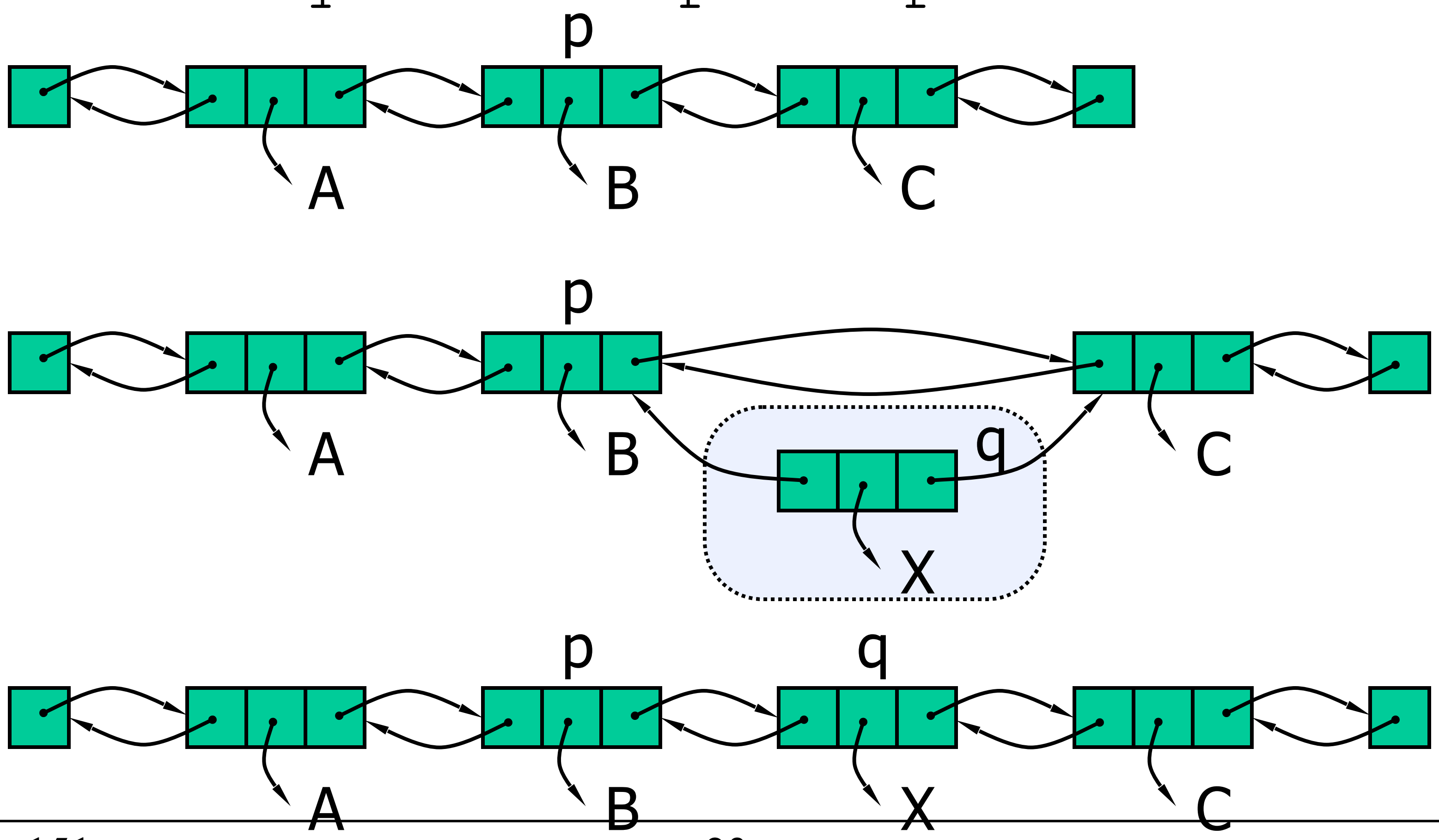
Basics

```
@Override
public int size() {
     return size;
 }
@Override
 public boolean isEmpty() {
     return size == 0;
 }

@Override
    public T first() {
     if (head == null)
         return null;
     return head.data;
 }
@Override
 public T last() {
     if (head == null)
         return null;
     return tail.data;
 }
```

Insertion: AddFirst, AddLast

Add Between

- Insert `q` between `p` and `p.next`



Add Between

```
protected void addBtw(T c, Node prev, Node next) {
    Node newest = new Node<>(c);
    prev.next = newest;
    next.prev = newest;
     newest.prev = prev;
     newest.next = next;
    size++;
}
```

Why not public?? Is this enough?

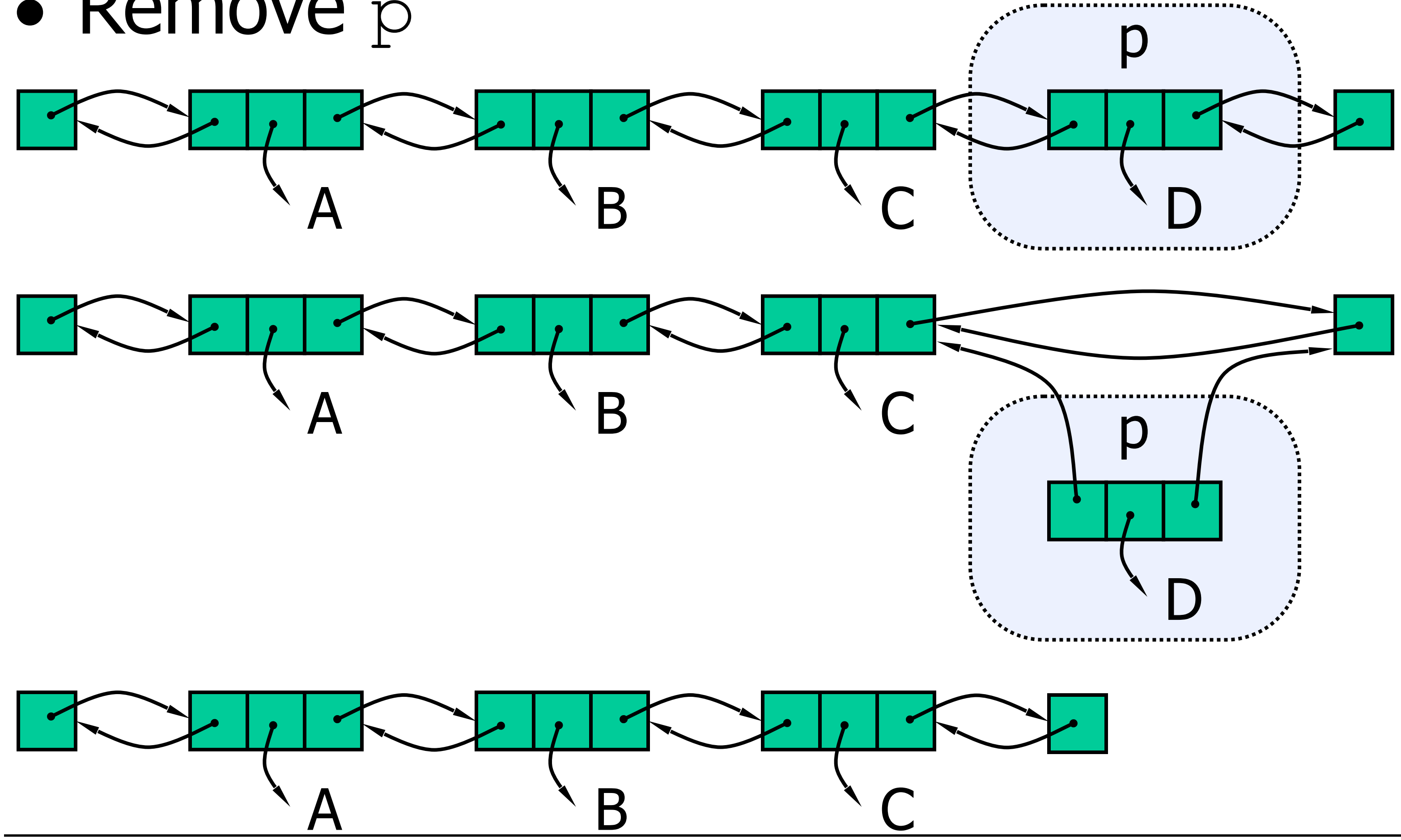
Deletion — last element

```
public T removeLast() {
        if (head == null)
            return null;
        T rtn = tail.data;
        if (head == tail) {
            head = null;
            size = 0;
            tail = null;
            return rtn;
        }
        tail = tail.prev;
        tail.next = null;
        size--;
        return rtn;
    }
```

Deleting the first element is very similar

Deletion

- Remove `p`


p
A
B
C
D
A
B
C
p
D
A
B
C