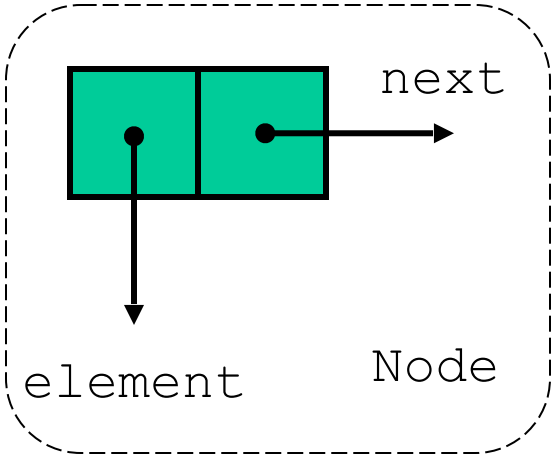
CS206

Generic Linked Lists

Generic Linked List

- A linked list not locked into one type of object




```
private class Node<E> {
   public E element;
   public Node<E> next;
   public Node(E element, Node<E>
next) {
        this.element = element;
        this.next = next;
     }
```

Basics

```
private Node<E> head = null;
private Node<E> tail = null;
private int size = 0;
public int size() {return size;}
public boolean isEmpty() {return size == 0;}

public E first() throws NoSuchElementException {
  if (isEmpty()) {
      throw new NoSuchElementException("There are no rabbits");
      }
  else {
      return head.element;}
}
```

Insertion & Deletion

```
public void addLast(T c) {
     Node<T> newest = new Node<>(c, null);
     if (isEmpty()) {
         head = newest;
     } else {
         tail.next = newest;
     }
     tail = newest;
     size++;
 }

public void addFirst(T c) {
     Node<T> newest = new Node<>(c, head);
     if (isEmpty()) {
         head = newest;
         tail = newest;
     }
     head = newest;
     size++;
 }
```

Find

Recall from RabbitLinkedList

Problem

```
public Rabbit find(String id)
{
    Node curr = head;
    while (curr!=null)
    {
        if (curr.data.getId().equals(id))
        {
       return curr.data;
        }
        curr=curr.next;
    }
    return null;
}
```

Need a generic way to compare!!!

The Comparable Interface

```
public interface Comparable<T>
  72: {
  73:   /**
  74:    * Compares this object with another, and returns a numerical result based
  75:    * on the comparison.  If the result is negative, this object sorts less
  76:    * than the other; if 0, the two are equal, and if positive, this object
  77:    * sorts greater than the other.  To translate this into boolean, simply
  78:    * perform <code>o1.compareTo(o2) <em>&lt;op&gt;</em> 0</code>, where op
  79:    * is one of &lt;, &lt;=, =, !=, &gt;, or &gt;=.
  80:    *
  81:    * (deleted more)    *
  92:    * @param o the object to be compared
  93:    * @return an integer describing the comparison
  94:    * @throws NullPointerException if o is null
  95:    * @throws ClassCastException if o cannot be compared
  96:    */
  97:   int compareTo(T o);
  98: }
```

Short story: return 0 if equal, negative if less, positive if greater

Comparable example
Integer and String

```
public class ComparableEx {
    public static void main(String[] args) {
        Integer i5 = new Integer(50);
        Integer i3 = new Integer(30);
        Integer j5 = new Integer(50);
        System.out.println("i5:" + i5 + "  i3:" + i3 + "  j5:" + j5);
        System.out.println("i3.compareTo(i5) " + i3.compareTo(i5));
        System.out.println("i5.compareTo(i3) " + i5.compareTo(i3));
        System.out.println("i5.compareTo(j5) " + i5.compareTo(j5));
        System.out.println("i5.equals(j5) " + i5.equals(j5));
        System.out.println("(i5 == j5) " + (i5 == j5));

        String abc = "abc";
        String def = "def";
        String abc1 = new String("abc");
        System.out.println("abc:" + abc + "  def:" + def + "  abd0:" + abc1);
        System.out.println("abc.compareTo(def) " + abc.compareTo(def));
        System.out.println("def.compareTo(abc) " + def.compareTo(abc));
        System.out.println("abc.compareTo(abc0) " + abc.compareTo(abc1));
        System.out.println("abc.equals(abc0) " + abc.equals(abc1));
        System.out.println("abc == abc0 " + (abc == abc1));
    }
}
```

Generic with Comparable

```
public interface LinkedListInterface<E extends Comparable<E>>
{ // also replace Rabbit with E
}
```

```
    private class Node<T extends Comparable<T>> {
        /** The data item in the node. An instance of rabbit */
        public Comparable<T> data;
        /** The next item in the linked list */
        public Node<T> next;

        /**
         * Node constructor. Takes a T and another node
         */
        public Node(Comparable<T> data, Node<T> next) {
            this.data = data;
            this.next = next;
        }
    }
```

With these changes this linked list class requires that any class being stored within implement the comparable interface

Find with Comparable


```
@Override
@SuppressWarnings("unchecked")
public T find(T rr) {
    Node<T> curr = head;
    while (curr != null) {
        if (0 == curr.data.compareTo(rr)) {
            return (T) curr.data;
        }
        curr = curr.next;
    }
    return null;
}
```

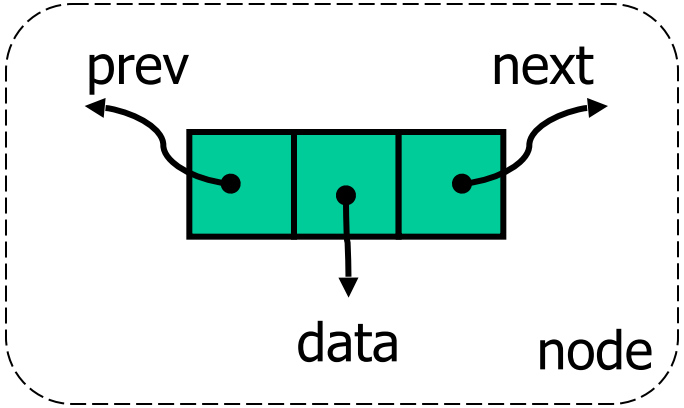
Comparable Rabbit

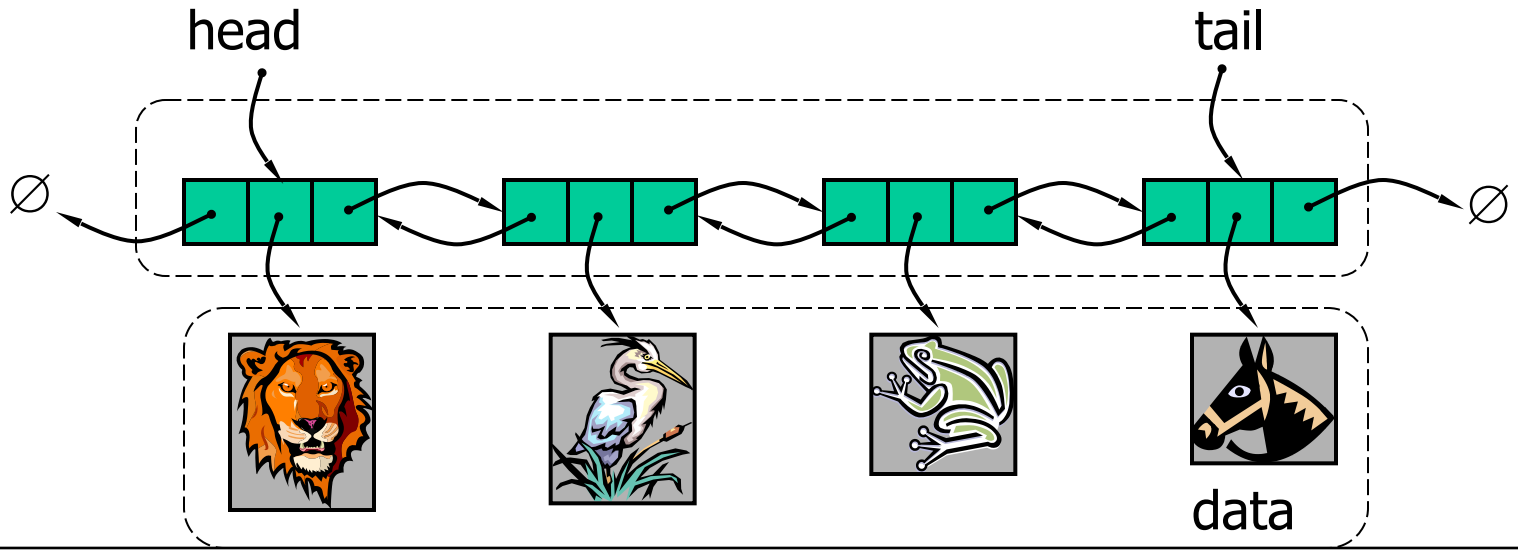
```
public class Rabbit implements Comparable<Rabbit>
{
    // otherwise unchanged
    @Override
     public int compareTo(Rabbit r)
     {
         return iD.compareTo(r.getId());
     }
}
```

Goto unix and run

Doubly Linked List

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





Double Linked List interface

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

This is identical to the generic single linked list!!!

Node & DLL start

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

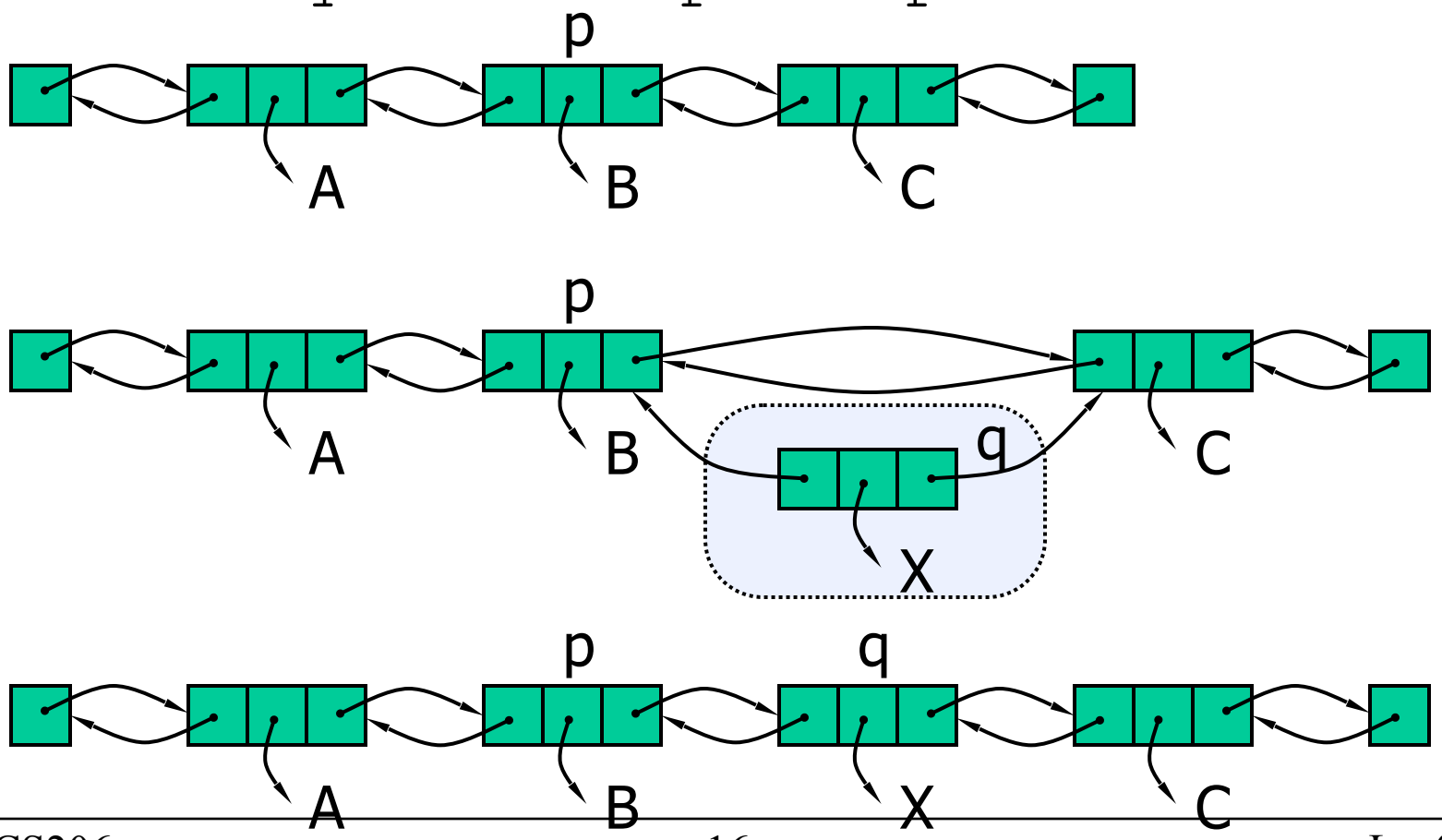
Basics

```
    @Override
    public int size() {
        return size;
     }
    @Override
     public boolean isEmpty() {
        return size == 0;
     }
     @Override
     @SuppressWarnings("unchecked")
     public T first() throws NoSuchElementException {
        if (head == null)
            throw new NoSuchElementException("Nothing");
        return (T)head.data;
     }
     @Override
     @SuppressWarnings("unchecked")
     public T last() throws NoSuchElementException {
        if (head == null)
            throw new NoSuchElementException("Nothing");
        return (T)tail.data;
     }
```

Insertion: AddFirst, AddLast

Add Between

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



Add Between

```
public void addBtw(Rabbit c, Node prev, Node next)
{
    Node newest = new Node(c, prev, next);
    prev.next = newest;
    next.prev = newest;
    size++;
}
```

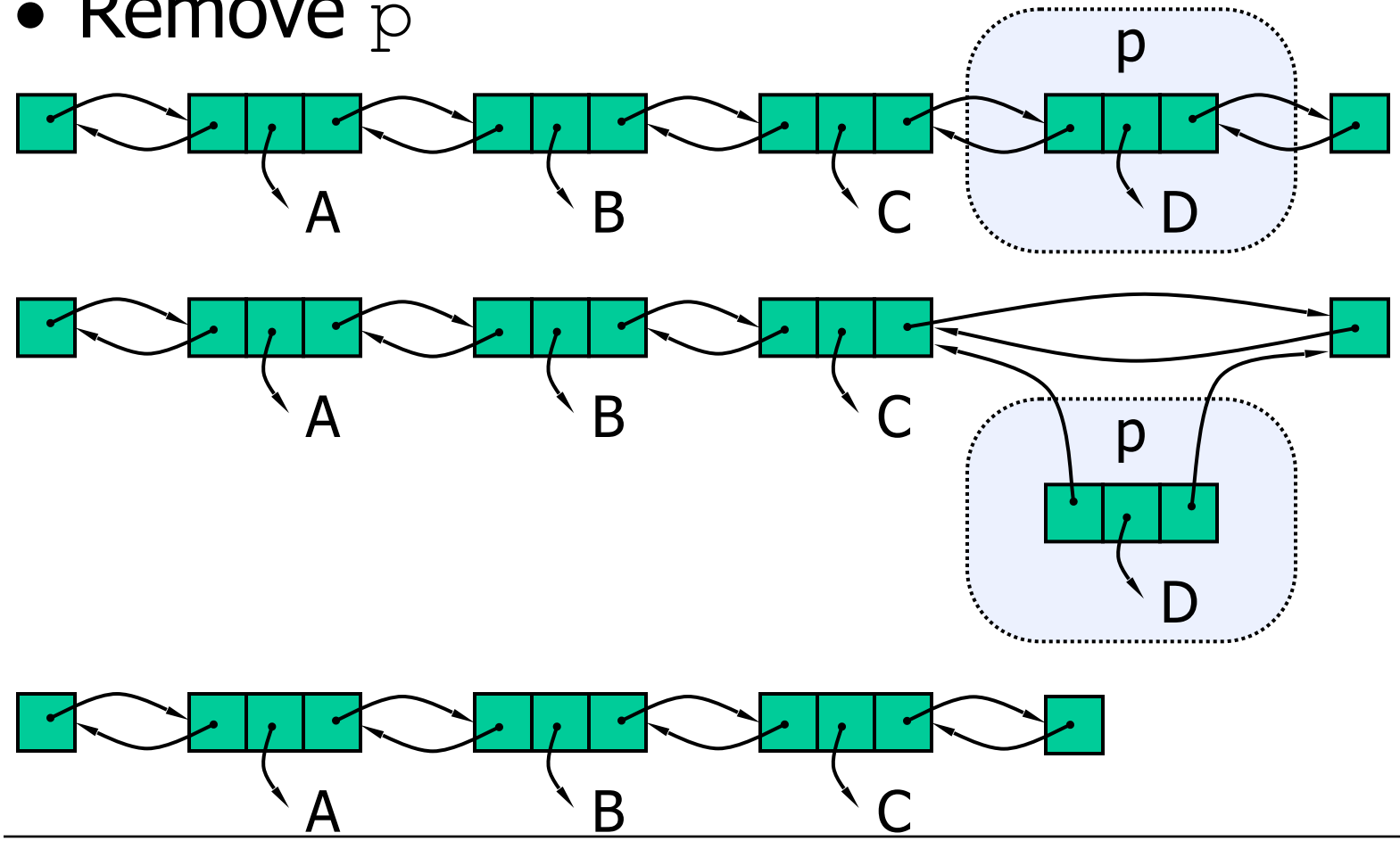
Problems??

Deletion — first element

```
    @Override
    @SuppressWarnings("unchecked")
    public T removeFirst() {
        if (head == null)
            return null;
        Comparable<T> rtn = head.data;
        head = head.next;
        if (head == null)
            tail = null;
        else
            head.prev = null;
        size--;
        return (T) rtn;
    }
```

Deletion

- Remove `p`



Deletion

```
@Override
    public T remove(T r) {
        // Do something much like find, but need to trak the previous node
        Node<T> curr = head;
        while (curr != null) {
            if (0 == curr.data.compareTo(r)) {
                break;
            }
            curr = curr.next;
        }
        if (curr == null) {
            // 1. the rabbit was not found
            return null;
        }
        size--;
        if (curr.prev != null)
            curr.prev.next = curr.next;
        if (curr.next != null)
            curr.next.prev = curr.prev;
        if (curr == tail)
            tail = curr.prev;
        return r;
    }
```

Sorted Linked Lists

```
public class SortedDLL<T extends Comparable<T>> extends
DoubleLinkedList<T> {

    @Override
    public void addLast(T t) {
    }

    @Override
    public void addFirst(T t) {
    }

    @SuppressWarnings("unchecked")
    public void addSorted(Comparable<T> t) {
      // lots of thought here
      }
```