

20-0: Linked Lists

- From Hands-on on Monday:
 - `removeAt(int index)`
 - `remove(Object o)`

20-1: `removeAt`

```
public class LinkedList
{
    private ListNode head;
    public void removeAt(int index)
    {
        if (index == 0)
        {
            head = head.next;
        }
        else
        {
            ListNode tmp = head;
            for (int i = 0; i < index - 1; i++)
            {
                tmp = tmp.next;
            }
            tmp.next = tmp.next.next;
        }
    }
}
```

20-2: `remove`

```
public class LinkedList
{
    public void remove(Object o)
    {
        if (head != null)
        {
            if (head.data.equals(o))
            {
                head = head.next;
            }
            else
            {
                ListNode tmp;
                for (tmp = head;
                     tmp.next != null && !tmp.next.data.equals(o);
                     tmp = tmp.next);
                if (tmp.next != null)
                {
                    tmp.next = tmp.next.next;
                }
            }
        }
    }
}
```

20-3: `remove`

```
public class LinkedList
{
    public void remove(Object o)
    {
        if (head != null)
        {
            if (head.data.equals(o))
            {
                head = head.next;
            }
            else
            {
                for (ListNode tmp = head; tmp.next != null; tmp = tmp.next)
                {
                    if (tmp.next.data.equals(o))
                    {
                        tmp.next = tmp.next.next;
                        return;
                    }
                }
            }
        }
    }
}
```

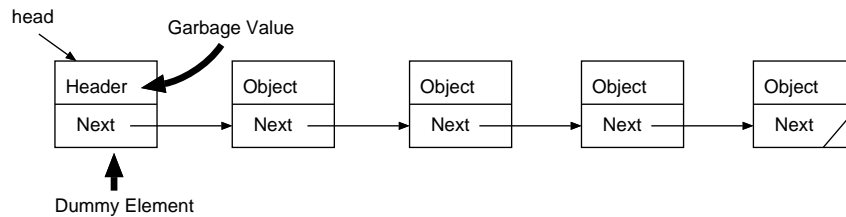
20-4: Special Cases

- Two special cases:

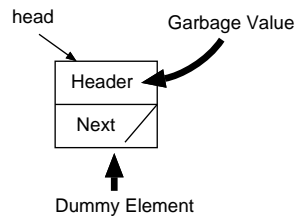
- Removing from an empty list
- Removing the first element in the list
- We can remove these special cases by using a dummy element

20-5: Dummy Element

Linked List with Dummy element



Empty List



20-6: Dummy Element

```
class LinkedList
{
    ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    // Other methods ...
}
```

20-7: find, Dummy Element

```
class LinkedList
{
    ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    boolean find(Object o) { ... }
}
```

20-8: find, Dummy Element

```
class LinkedList
{
    ListNode head;

    boolean find(Object o)
    {
        for (ListNode tmp = head.next; tmp != null; tmp = tmp.next)
        {
            if (tmp.data.equals(o))
            {
                return true;
            }
        }
        return false;
    }
}
```

20-9: RemoveAt, Dummy Element

```
public class LinkedList
{
    private ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    public void removeAt(int index) { ... }
}

```

20-10: RemoveAt, Dummy Element

```
public class LinkedList
{
    private ListNode head;
    public void removeAt(int index)
    {
        ListNode tmp = head;
        for (int i = 0; i < index; i++)
        {
            tmp = tmp.next;
        }
        tmp.next = tmp.next.next;
    }
}

```

20-11: remove, Dummy Element

```
public class LinkedList
{
    private ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    public void remove(int index) { ... }
}

```

20-12: remove, Dummy Element

```
public class LinkedList
{
    private ListNode head;
    public void remove(Object o)
    {
        ListNode tmp;
        for (tmp = head;
             tmp.next != null && !tmp.next.data.equals(o);
             tmp = tmp.next);
        if (tmp.next != null)
        {
            tmp.next = tmp.next.next;
        }
    }
}

```

20-13: remove, Dummy Element

```
public class LinkedList
{
    private ListNode head;
    public void remove(Object o)
    {
        for (ListNode tmp = head; tmp.next != null; tmp = tmp.next)
            if (tmp.next.data.equals(o))
            {
                tmp.next = tmp.next.next;
                return;
            }
    }
}

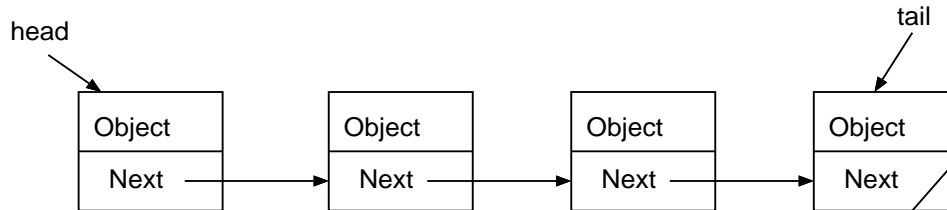
```

20-14: Speed up Appending

- To append an element to the end of a linked list, need to move a temp pointer to the end of the list
- How could we do this more quickly

- We can add things to our data structure ...

20-15: Tail Pointer



20-16: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void insert(Object o) { ... }
}
```

20-17: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void insert(Object o)
    {
        head = new ListNode(o, head);
        if (tail == null)
            tail = head;
    }
}
```

20-18: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void append(Object o) { ... }
}
```

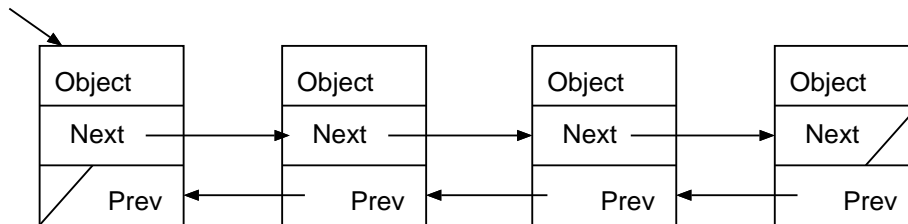
20-19: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

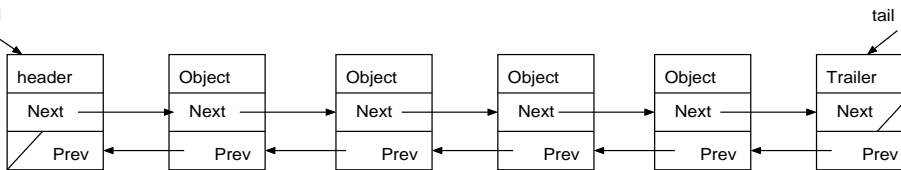
    public void append(Object o)
    {
        if (tail != null)
        {
            tail.next = new ListNode(o);
            tail = tail.next;
        }
        else
        {
            head = new ListNode(o, head);
            tail = head;
        }
    }
}
```

20-20: **Doubly Linked List**

head

20-21: **Doubly Linked List**

head

20-22: **Doubly Linked List**

- Advantages
 - Don't need to be "one off" to do deletions
 - Don't need as many special cases (especially with dummy header and trailer nodes)
- Disadvantages
 - Uses more space (two pointers per node)
 - Need to update both previous/next when changing list

20-23: **Midterm**

- Recursion
- Polymorphism
- Exceptions
- Reverse

20-24: **Lab – Doubly Linked Lists**

(Switch to web browser for assignment)