Building Java Programs

Chapter 16
Lecture 16-3: Complex Linked List Code

reading: 16.2 – 16.3
addSorted

- Write a method `addSorted` that accepts an `int` as a parameter and adds it to a sorted list in sorted order.

- Before `addSorted(17)`:

```
front = [ ]
data = -4
next = [ ]
data = 8
next = [ ]
data = 22
next = [ ]
```

- After `addSorted(17)`:

```
front = [ ]
data = -4
next = [ ]
data = 8
next = [ ]
data = 17
next = [ ]
data = 22
next = [ ]
```
The common case

- Adding to the middle of a list:
  \[ \text{addSorted}(17) \]

Which references must be changed?
What sort of loop do we need?
When should the loop stop?
First attempt

- An incorrect loop:

```java
ListNode current = front;
while (current.data < value) {
    current = current.next;
}
```

- What is wrong with this code?
  - The loop stops too late to affect the list in the right way.
Recall: changing a list

- There are only two ways to change a linked list:
  - Change the value of `front` (modify the front of the list)
  - Change the value of `<node>.next` (modify middle or end of list to point somewhere else)

- Implications:
  - To add in the middle, need a reference to the `previous` node
  - Front is often a special case
Key idea: peeking ahead

- Corrected version of the loop:

```java
ListNode current = front;
while (current.next.data < value) {
    current = current.next;
}
```

- This time the loop stops in the right place.
Another case to handle

- Adding to the end of a list: `addSorted(42)`

```
front =  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
```

Exception in thread "main": `java.lang.NullPointerException`

- Why does our code crash?
- What can we change to fix this case?
Multiple loop tests

• A correction to our loop:

```java
ListNode current = front;
while (current.next != null &&
       current.next.data < value) {
    current = current.next;
}
```

- We must check for a `next` of `null` before we check its `.data`. 
Third case to handle

- Adding to the front of a list:
  \texttt{addSorted(-10)}

What will our code do in this case?
What can we change to fix it?
Handling the front

- Another correction to our code:

```java
if (value <= front.data) {
    // insert at front of list
    front = new ListNode(value, front);
} else {
    // insert in middle of list
    ListNode current = front;
    while (current.next != null &&
            current.next.data < value) {
        current = current.next;
    }
}
```

- Does our code now handle every possible case?
Fourth case to handle

- Adding to (the front of) an empty list:
  \[ \text{addSorted}(42) \]

What will our code do in this case?
What can we change to fix it?
// Adds given value to list in sorted order.
// Precondition: Existing elements are sorted
public void addSorted(int value) {
    if (front == null || value <= front.data) {
        // insert at front of list
        front = new ListNode(value, front);
    } else {
        // insert in middle of list
        ListNode current = front;
        while (current.next != null &&
               current.next.data < value) {
            current = current.next;
        }
    }
}
Common special cases

- **middle**: "typical" case in the middle of an existing list
- **back**: special case at the back of an existing list
- **front**: special case at the front of an existing list
- **empty**: special case of an empty list
Other list features

- Add the following methods to the `LinkedList`:
  - `size`
  - `isEmpty`
  - `clear`
  - `toString`
  - `indexOf`
  - `contains`

- Add a `size` field to the list to return its size more efficiently.

- Add preconditions and exception tests to appropriate methods.