More Linked Lists

reading: 16.2 - 16.3

slides created by Marty Stepp
http://www.cs.washington.edu/143/
Conceptual questions

- What is the difference between a `LinkedIntList` and a `ListNode`?

- What is the difference between an empty list and a `null` list?
  - How do you create each one?

- Why are the fields of `ListNode` public? Is this bad style?

- What effect does this code have on a `LinkedIntList`?

  ```java
  ListNode current = front;
  current = null;
  ```
Conceptual answers

• A list consists of 0 to many node objects.
  – Each node holds a single data element value.

• null list: $\text{LinkedIntList list = null;}$
  empty list: $\text{LinkedIntList list = new LinkedIntList();}$

• It's okay that the node fields are public, because client code never directly interacts with ListNode objects.

• The code doesn't change the list.
  You can change a list only in one of the following two ways:
  – Modify its front field value.
  – Modify the next reference of a node in the list.
// Removes and returns the list's first value.
public int remove() {
    ...
}

- How do we remove the front node from a list?
- Does it matter what the list's contents are before the remove?
Removing front element

• Before removing front element:

• After first removal:

• After second removal:
// Removes and returns the first value.
// Throws a NoSuchElementException on empty list.
public int remove() {
    if (front == null) {
        throw new NoSuchElementException();
    } else {
        int result = front.data;
        front = front.next;
        return result;
    }
}
Implementing `remove` (2)

// Removes value at given index from list.
// Precondition: 0 <= index < size
public void remove(int index) {
    ...
}

- How do we remove any node in general from a list?
- Does it matter what the list's contents are before the remove?
Removing from a list

• Before removing element at index 1:

  front = element 0

  element 0
  data: 42
  next: 

  data: -3
  next: 

  element 2
  data: 20
  next: 

• After:

  front = element 0

  element 0
  data: 42
  next: 

  element 1
  data: 20
  next: 

Removing from the front

- Before removing element at index 0:
  - Front = 42
  - Next data: 20
  - Next data: -3
  - Next data: 20

- After:
  - Front = -3
  - Next data: 20
  - Next data: 20
Removing the only element

- We must change the front field to store `null` instead of a node.
- Do we need a special case to handle this?
// Removes value at given index from list.
// Precondition: 0 <= index < size()
public void remove(int index) {
    if (index == 0) {
        // special case: removing first element
        front = front.next;
    } else {
        // removing from elsewhere in the list
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = current.next.next;
    }
}
Exercise

• Write a method `addSorted` that accepts an integer value as a parameter and adds that value to a sorted list in sorted order.

  – Before `addSorted(17)`:

  front = 
  
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 0

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 1

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 2

  – After `addSorted(17)`:

  front = 
  
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 0

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 1

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 2

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
  
  element 3
The common case

- Adding to the middle of a list: 
  `addSorted(17)`

<table>
<thead>
<tr>
<th>front</th>
<th>data</th>
<th>next</th>
<th>data</th>
<th>next</th>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-4</td>
<td></td>
<td>8</td>
<td></td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

- 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.
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:
  \texttt{addSorted(42)}

Front: 

\begin{tabular}{|c|c|c|c|}
\hline
\text{data} & \text{next} & \text{data} & \text{next} \\
\hline
-4 & & 8 & \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|}
\hline
\text{data} & \text{next} \\
\hline
22 & \\
\hline
\end{tabular}

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

- Why does our code crash?
- What can we change to fix this case?
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:
  \texttt{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;
        }
    }
}
Other list features

• Add the following methods to the `LinkedIntList`:
  - 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.