Building Java Programs

Complex Linked List Code

reading: 16.2 – 16.3
First order of business: bug check.
**LinkedIntList(int n)**

- Write a constructor for `LinkedIntList` that accepts an `int n` parameter and makes a list of the number from 0 to `n`.
  - `new LinkedIntList(3)`: 
    ```
    front = |
    data | next |
    3    |      |
    2    |      |
    1    |      |
    0    |```
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 =` (empty)
- `data = -4`, `next =` (empty)
- `data = 8`, `next =` (empty)
- `data = 22`, `next =` (empty)

**After `addSorted(17)`:**

- `front =` (empty)
- `data = -4`, `next =` (empty)
- `data = 8`, `next =` (empty)
- `data = 17`, `next =` (empty)
- `data = 22`, `next =` (empty)
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.
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 = element 0
    data: -4
    next:

    data: 8
    next:

    data: 22

    data: next
```

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`. 
Adding to the front of a list:
\textit{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?
Final version of code

// 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 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