What Are We Doing Again?

We’re building an alternative data structure to an ArrayList with different efficiencies.

Today’s Main Goals:
- Get more familiarity with LinkedLists
- Write more LinkedList methods
- Learn how to “protect” against NullPointerExceptions

A New LinkedList Constructor

First Attempt
```java
public LinkedList(int n) {
    /* Current State */
    ListNode current = this.front;
    for (int i = 1; i <= n; i++) {
        current = current.next;
    }
}
```

Remember, to edit a LinkedList, we MUST edit one of the following:
- front, or
- node.next (for some ListNode node)

In our code above, we edit current, which is neither.

Second Attempt
```java
public LinkedList(int n) {
    /* Current State */
    if (n > 0) {
        this.front = new ListNode(1);
        ListNode current = this.front;
        for (int i = 1; i <= n; i++) {
            current.next = new ListNode(i);
            current = current.next;
        }
    }
}
```

What kind of loop should we use?
A for loop, because we have numbers we want to put in the list.

What cases should we worry about?
We’re creating the list; so, there aren’t really “cases”.

A New LinkedList Constructor: Another Solution

This other solution works by going backwards. Before, we were editing the next fields. Here, we edit the front field instead:

```java
Different Solution!
public LinkedList(int n) {
    /* Current State */
    for (int i = n; i > 0; i--) {
        this.front = new ListNode(i, null);
    }
    this.front = new ListNode(1, next);
}
```

Implementing addSorted

Write a method addSorted(int value) that adds value to a sorted LinkedIntList and keeps it sorted. For example, if we call addSorted(10) on the following LinkedIntList.

```
front
5 4 3 2 1
```

We would get:

```
front
5 4 3 2 1 10
```

As always, we should approach this by considering the separate cases (and then drawing pictures):
- We're supposed to insert at the front
- We're supposed to insert in the middle
- We're supposed to insert at the back

Case: Middle

An Incorrect Solution

```
public void addSorted(int value) { //Say value = 10...
    ListNode current = this.front;
    while (current.data < value) {
        current = current.next;
    }
    //the while loop continues...
}
```

Uh Oh! We went too far! We needed the next field BEFORE us.

Fixing the Problem

```
public void addSorted(int value) { //Say value = 10...
    ListNode current = this.front;
    while (current.data < value) {
        current = current.next;
    }
    current.next = new ListNode(value, next);
}
```

Does this cover all the cases?

Case: End

Adding At The End?

```
public void addSorted(int value) { //Say value = 40...
    ListNode current = this.front;
    while (current.next.data < value) {
        current = current.next;
    }
    //the while loop continues...
}
```

We fell off the end of the LinkedList.

Idea: Make sure current.next exists.
Our current code only sets current to a new ListNode. Importantly, this never updates front; so, we lose the new node.

```java
public void addSorted(int value) {
    if (value < front.data) {
        ListNode next = front;
        front = new ListNode(value, next);
    } else {
        while (current.next != null && current.next.data < value) {
            current = current.next;
        }
        ListNode next = current.next;
        current.next = new ListNode(value, next);
    }
}
```

Have we covered all of our cases now?

With LinkedList code, every time we make a test (if, while, etc.), we need to make sure we're protected. Our current code is:

```java
public void addSorted(int value) {
    if (value < front.data) {
        ListNode next = front;
        front = new ListNode(value, next);
    } else {
        while (current.next != null && current.next.data < value) {
            current = current.next;
        }
        ListNode next = current.next;
        current.next = new ListNode(value, next);
    }
}
```

We're "protected" if we know we won't get a NullPointerException when trying the test. So, consider our tests:
- value < front.data
- current.next != null && current.next.data < value

So, Are We Protected?

Nope! What happens if front == null? We try to get the value of front.data, and get a NullPointerException. The fix:

```java
public void addSorted(int value) {
    if (front == null || value < front.data) {
        front = new ListNode(value, null);
    } else {
        while (current.next != null && current.next.data < value) {
            current = current.next;
        }
        current.next = new ListNode(value, next);
    }
}
```

Helpfully, this fix actually handles the empty list case correctly!

Make sure to try all the cases:
- Empty List
- Front of Non-empty List
- Middle of Non-empty List
- Back of Non-empty List

To Edit a LinkedList, the assignment must look like:
- this.front = <something>; or
- node.next = <something>; (for some ListNode node in the list)

When protecting your conditionals! Make sure that nothing can accidentally be null.

When protecting your conditionals, make sure the less complicated check goes first.