Linked Lists I
1. Get more familiar with LinkedNodes

2. Learn how to run through the values of a LinkedList

3. Learn how LinkedIntList is implemented

4. Learn about the different cases to deal with for LinkedLists
Quick Note: When I say “does that make sense?”…

- If it does make sense, yell “yes”

- Otherwise, say nothing.
1. Get more familiar with LinkedNodes
2. Learn how to run through the values of a LinkedList
3. Learn how LinkedIntList is implemented
4. Learn about the different cases to deal with for LinkedLists
Another **LinkedNode** Example

**Before:**

```
     list
       ↓
      1 → 2 → 4 → 3
  (0) (1) (4) (3)

list2
     ↓
      3 → 4
  (2) (3)
```

**After:**

```
     list
       ↓
      1 → 2 → 4 → 3
  (0) (1) (4) (3)

list2
     ↓
      4 → 2 → 3
  (4) (2) (3)
```

How many **LinkedNode**s are there in the before picture?

There are **FOUR**. Each box is a **LinkedNode**.

How many references to **LinkedNode**s are there?

There are **SIX**. Every arrow is a reference to a **LinkedNode**.
Another **LinkedNode** Example (Solution)

Before:

```
list
④↓ ①
③ ②
1 2 3 4
```

```
list2
⑤↓ ③
② ①
3 4
```

After:

```
list
④↓ ①
③ ②
1 2 4 3
```

```
list2
⑤↓ ③
② ①
3 4
```

1. `list.next.next = list2.next`
2. `list2.next.next = list2;`
3. `list2.next = null;`
1. Get more familiar with LinkedNodes

2. Learn how to run through the values of a LinkedList

3. Learn how LinkedIntList is implemented

4. Learn about the different cases to deal with for LinkedLists
Printing a LinkedList

Now, note that we can use a variable to keep track of where we are:

```java
System.out.println(list.data);
list = list.next;
System.out.println(list.data);
list = list.next;
System.out.println(list.data);
list = list.next;
System.out.println(list.data);
```

---

Printing a LinkedList Manually

```java
System.out.println(list.data);  
list = list.next;  
System.out.println(list.data);  
list = list.next;  
System.out.println(list.data);  
list = list.next;  
System.out.println(list.data);  
```
What if our list has 1000 nodes? That would be horrible to write.

```java
ListNode list = ...; // List with 1000 nodes
while (list != null) {
    System.out.println(list.data);
    list = list.next;
}
```

But that destroys the list; so, use a temporary variable instead:

```java
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```
We can use for loops in a similar way to with ArrayLists to run through LinkedLists!

### Traversing an ArrayList

```java
for (int i = 0; i < arrayList.size(); i++) {
    System.out.println(arrayList.get(i));
}
```

### Traversing an LinkedList

```java
for (ListNode current = linkedList; current != null; current = current.next) {
    System.out.println(current.data);
}
```

<table>
<thead>
<tr>
<th>Description</th>
<th>ArrayList Code</th>
<th>LinkedList Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to front of list</td>
<td>int i = 0;</td>
<td>ListNode current = list;</td>
</tr>
<tr>
<td>Test for more elements</td>
<td>i &lt; list.size()</td>
<td>current != null</td>
</tr>
<tr>
<td>Current value</td>
<td>list.get(i)</td>
<td>current.data</td>
</tr>
<tr>
<td>Go to next element</td>
<td>i++;</td>
<td>current = current.next;</td>
</tr>
</tbody>
</table>
1. Get more familiar with LinkedNodes
2. Learn how to run through the values of a LinkedList
3. Learn how LinkedIntList is implemented
4. Learn about the different cases to deal with for LinkedLists
No generics (only stores ints)

Fewer methods: add(value), add(index, value), get(index), set(index, value), size(), isEmpty(), remove(index), indexOf(value), contains(value), toString()

This is the same idea as when we implemented ArrayIntList!
What fields does our `LinkedIntList` need?

**A reference to the front of the list**

```
public class LinkedIntList {
    private ListNode front;

    public LinkedIntList() {
        front = null;
    }

    // ...
}
```
**Buggy toString()**

```java
public String toString() {
    String result = "["
    ListNode current = this.front;
    while (current != null) {
        result += current.data + ", ";
        current = current.next;
    }
    return result + "]";
}
```

**Our toString() puts a trailing comma. Fix it by stopping one early:**

**Fixed toString()**

```java
public String toString() {
    String result = "["
    ListNode current = this.front;
    while (current != null && current.next != null) {
        result += current.data + ", ";
        current = current.next;
    }
    if (current != null) {
        result += current.data;
    }
    return result + "]";
}
```
1. Get more familiar with LinkedNodes
2. Learn how to run through the values of a LinkedList
3. Learn how LinkedList is implemented
4. Learn about the different cases to deal with for LinkedLists
Modifying LinkedLists

Writing a LinkedList Method

1. Identify cases to consider...
   - Front/Empty
   - Middle
   - End

2. Draw pictures for each case

3. Write each case separately

front 1  2  3  ...  9  10  11  ...  42 end
LinkedIntList add() *(Empty Case)*

Cases to consider:
- Add to empty list
- Add to non-empty list

Add To An Empty List

What does an empty list look like?

```java
public void add(int value) {
    /* If the list is empty... */
    if (this.front == null) {
        this.front = new ListNode(value);
    }
    /* Other Cases ... */
}
```
Add To A Non-Empty List

Consider a non-empty list:

```
front
↓
1 → 2 → 3 → ... → 100
```

/* Idea: We want to change the red arrow.
Loop until we’re at the last node. */

```java
ListNode current = this.front;

while (current != null) {
    current = current.next;
}

current = new ListNode(value);
```
Add To A Non-Empty List (Fixed)

Consider a non-empty list:

```
front

1 -> 2 -> 3 -> ... -> 100
```

/* Idea: We want to change the red arrow. */
Loop until we’re at the node before the last node */
```
ListNode current = this.front;
```
```
while (current.next != null) {
    current = current.next;
}
```
```
current.next = new ListNode(value);
```
There are only two ways to modify a LinkedList:

- **Change front**

  \[ \text{front}\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \text{...changing front...} \]

  \[ (\ [1,2,3] ; \ \text{...changing front...} \rightarrow [2,3] ) \]

- **Change current.next for some ListNode, current**

  \[ \text{front}\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \text{...changing .next...} \]

  \[ (\ [1,2,3] ; \ \text{...changing .next...} \rightarrow [1,3] ) \]

**Setting “current” does NOTHING!**
public int get(int index) {
    ListNode current = front;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    return current.data;
}
Some LinkedList Tips!

- Be able to deal with before-and-after LinkedNode pictures

- Know how to loop through a LinkedList
  - Use a while loop.
  - Don’t forget to create a ListNode current variable so we don’t destroy the original list.
  - Don’t forget to update the current variable.

- Understand differences and similarities between ArrayList and LinkedList
  - They both have the same functionality (add, remove, etc.)
  - But they’re implemented differently (array vs. ListNodes)

- With LinkedLists, you often have to stop one node before the one you want.

- DO NOT start coding LinkedList problems without drawing pictures first.