CSE 143
Computer Programming II

Outline

1. Get more familiar with `LinkedNode`s
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 `LinkedList`s

Does That Make Sense?

Quick Note: When I say “does that make sense?”...

- If it does make sense, yell "yes"
- Otherwise, say nothing.

Another `LinkedNode` Example

Before:

```
1 2 3 4
0 1 2 3
list
4
list2
5
```

After:

```
1 2 4 3
0 1 2
list
4
list2
5
```

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
  1 2 3
list2
  0 1 2
```

After:

```
list
  1 2 4 3
list2
  0 1 2 3
```

1. list.next.next = list2.next
2. list2.next.next = list2;
3. list2.next = null;

Outline

0. Get more familiar with LinkedNodes
1. Learn how to run through the values of a LinkedList
2. Learn how LinkedIntList is implemented
3. Learn about the different cases to deal with for LinkedLists

Printing a LinkedList

Printing a LinkedList Manually

```
1. System.out.println(list.data);
2. System.out.println(list.next.data);
3. System.out.println(list.next.next.data);
```

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

```
1. System.out.println(list.data);
2. list = list.next;
3. System.out.println(list.data);
4. list = list.next;
5. System.out.println(list.data);
6. list = list.next;
```

Printing a Big LinkedList

```
while (list != null) {
    System.out.println(list.data);
    list = list.next;
}
```

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

```
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

LinkedList vs. ArrayList

We can use for loops in a similar way to with ArrayLists to run through LinkedLists!

<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; i &lt; list.size(); i++ {</td>
<td>ListNode current = list; current =</td>
</tr>
<tr>
<td></td>
<td>System.out.println(arrayList.get(i));</td>
<td>(current != null) {}</td>
</tr>
<tr>
<td>Test for more elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current value</td>
<td>int i = 0; current = current.data;</td>
<td></td>
</tr>
<tr>
<td>Go to next element</td>
<td>i++;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list.get(i)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i++;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i++;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i++;</td>
<td></td>
</tr>
</tbody>
</table>

Outline

0. Get more familiar with LinkedNodes
1. Learn how to run through the values of a LinkedList
2. Learn how LinkedIntList is implemented
3. Learn about the different cases to deal with for LinkedLists
LinkedIntList

- 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 ArrayList!

LinkedIntList Fields

What fields does our LinkedIntList need?

A reference to the front of the list

LinkedIntList v1

```java
public class LinkedIntList {
    private ListNode front;

    public LinkedIntList() {
        front = null;
    }
}
```

Outline

1. Get more familiar with LinkedLists
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

Modifying LinkedLists

Writing a LinkedList Method

- Identify cases to consider...
  - Front/Empty
  - Middle
  - End
- Draw pictures for each case
- Write each case separately

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 (this.front == null) {
        this.front = new ListNode(value);
    }
}
```
Add To A Non-Empty List (Fixed)
Consider a non-empty list:

```
1 2 3 . . .
```

```java
/* Idea: We want to change the red arrow. */
ListNode current = this.front;
while (current.next != null) {
    current = current.next;
}
```

```
1 2 3 . . .
```

```java
current.next = new ListNode(value);
```

```
1 2 3 . . .
```

Working with LinkedLists
There are only two ways to modify a LinkedList:

- Change front

```
1 2 3 . . . changing front...
```

```
[1, 2, 3]; ...changing front... [2, 3]
```

- Change current.next for some ListNode, current

```
1 2 3 . . . changing .next...
```

```
[1, 2, 3]; ...changing .next... [1, 3]
```

Setting “current” does NOTHING!

Some LinkedList Tips!
- Be able to deal with before-and-after LinkedList 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. ListNode)
- With LinkedLists, you often have to stop one node before the one you want.
- DO NOT start coding LinkedList problems without drawing pictures first.