Linked List operations

Assume some elements are already in the list (so don't have to check if list is empty)

Assume LinkedList of Strings (applies to any LinkedList)

Pseudocode

```
| "a" | → | "b" | → | null |
↑     |   |     |   |      |
first |   | last |
```

1.addToEnd(Object e)

Create new Node: Node newNode = new Node ("new")
Update current last: last.next = newNode
Update last: last = newNode

```
| "a" | → | "b" | → | null |
↑     |   |     |   |      |
first |   | last |
```

Time: O(1)

2. remove <last>
(3) contains (object e) (Scan at most n nodes)

Node current = first
while (current != null)
    if (current.element == e) // pseudocode for equals
        return true
    current = current.next

return false

Time: O(n)

(4) indexof (Object e) (Scan and keep counter).

```
0 1 2

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th></th>
</tr>
</thead>
</table>

Node current = first
while (current != null)
    if (current.element == e)
        return count
    else
        current = current.next
        count++

return -1
```

Time: O(n)

ex: indexof("c") 0 1 "a" 2 "b" 3 "c"
6) **insertAfterNode(Node n1, Object e)**

/inserts e after n1 (shifts elements to the right)/

![Diagram of a linked list with a new node inserted after n1]

```java
Node newNode = new Node(e);
newNode.next = n1.next;
null.next = newNode;
```

*Note: This order matters*

```java
Time: O(1)
```

6) **get (int index)** //Returns the element at location index

- Keep Node current, counter, scan Objects until count == index, return current (instead of count)

```java
counter = 0
current = first
while (current != null)
    if (count == index)
        return current.element
    current = current.next
    count++
return error!
```

⇒ Time: O(n)