Problem with arrays + Array Lists:
- For add, remove in the middle of the list, elements must be shifted.

Reason: For both, elements are stored adjacent in memory.

In memory:

Instead, let's make elements non-adjacent.

But now we lose ordering.

So what additional info can we keep with each element?

⇒ Pointers to next element!

Linked List - series of nodes, not always adjacent in memory.
Requires: import java.util.LinkedList;

```java
LinkedList<String> list = new LinkedList<String>();
```

How is a LinkedList implemented?

2 Classes: Node, LinkedList.

Node: (already built into Java!)

```java
public class Node {
    Object element;
    Node next; //something else

    /* Constructor */
    public Node (Object e) {
        this.element = e;
    }
}
```

```java
public class LinkedList {

    /* Constructor */
    public LinkedList () {
    }
```
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" \rightarrow "b" \rightarrow null  
↑ (next)  ↑  
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" \rightarrow "b" \rightarrow null  
↑  
first
```

```
↑  "new"  ↑
last
```

Time: O(1)

2. remove <Later>
contains("me")

\[
\text{current} \downarrow \rightarrow \text{current} \downarrow \rightarrow \ldots \rightarrow \text{"me"} \downarrow \rightarrow \text{null} \uparrow
\]

last

first

(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).

\[
\begin{array}{c}
0 & 1 & 2 & \ldots & n-1 \\
\text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \ldots \rightarrow \text{null} \\
\end{array}
\]

Node current = first

while (current != null)

if (current.element == e)

return count

// else (not needed)

current = current.next

count++

return -1

Time: O(n)

ex: indexOf("c")

\[
\begin{array}{c}
0 & 1 & 2 \\
\text{a} \rightarrow \text{b} \rightarrow \text{c} \\
\end{array}
\]
5. `insertAfterNode(Node nl, Object e)`

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

```
[Node] \rightarrow [Node] \rightarrow \ldots \rightarrow (n) \rightarrow \ldots \rightarrow [Node] \rightarrow \text{null}
```

```
| first | \rightarrow | \text{null} |
```

```
\text{newNode}
```

*Note: This matters*

```
Node newNode = new Node(e)
newNode.next = nl.next
nl.next = newNode
```

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)

```
counter = 0
current = first
```

```
while (current != null)
    if (count == index)
        return current.element
    current = current.next
    count++
```

`return error!` → Time: O(n)