What about general remove()?

remove(Node n)

 Somehow need to access n's previous node.

Let's keep this into! Doubly Linked List

public class Node {
    Object element;
    Node next;
    Node prev;
}

Now, remove (Node n):

n.prev.next = n.next
n.next.prev = n.prev
Let's compare worst-case runtimes

<table>
<thead>
<tr>
<th></th>
<th>addToEnd</th>
<th>remove</th>
<th>contains</th>
<th>insert</th>
<th>get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arraylist</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Doubly LL</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Array</strong></td>
<td>$O(1)$</td>
<td>$O(1)$ (no shift)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Arrays, Arraylists, LinkedLists have many functionalities. What if we didn't need so many?
Motivation for stacks:

- function/method calls:
  - `main( ...)`
  - `method1()`
    - //more code
  - `method2()`
    - //more code
  - `method3()`
    - `method2()`
    - `method1()`
    - `main()`

Eventually need to execute 'more code' for each method. How to keep track of which method to return to? What operations do we need?

Want: `main -> method1 -> method2 -> method3`

Never need to access anything from middle of list.
Stacks: insertions, deletions performed at top (LIFO)

- Pile of heavy textbooks
- Garage of trucks
- Not possible to insert/remove in the middle
- Back button on browsers

LIFO: Last-In-First-Out

Operations:
- push(e): insert e at top
- pop(): remove top element
- top(): view top element
- isEmpty(): returns true if empty
- size(): returns number of elements

Stack <Object-type> s = new Stack <Object-type>();

s.push(a)
s.push(b)
s.push(c)
s.pop()
s.pop()
s.push(d)

One Application: Balancing parentheses

[ () ] ok, [ ( ] ) is wrong!
Two types of errors: (1) type of parenthesis don't match
(2) # of opening parenthesis ≠ # of closed parenthesis

1. Read string (of parentheses, brackets)
2. For each char c in string
3. if c is '(', '[', or '{':
   4. push(c)
5. else if c is ')', ']', or '}'
6. if stack.isEmpty() => error!
7. else
8. if c does not match stack.pop() => error!
9. else
10. if !stack.isEmpty() => error!

Example: [ ( ] ) ?

\[
\begin{array}{c}
( \\
[ \\
\end{array}
\Rightarrow \begin{array}{c}
( \neq ] \Rightarrow \text{error} \\
[ \leftrightarrow ) \Rightarrow \text{error} \\
\text{empty, } \exists \Rightarrow \text{error}
\end{array}
\]

Array Implementation of Stack
Store: 1D array (stack) represents the stack
Integer: top of stack: index of top element, initially -1

Assume non-empty

- top(): return stack[top of stack]
- push(e): top of stack ++, stack[top of stack] = e
- pop(): top of stack--
  return stack[top of stack+1]