6. `insertAfterNode(Node n1, Object e)`
   /inserts e after n1 (shifts elements to the right)

   ```plaintext
   first  \rightarrow  n1  \rightarrow  \ldots  \rightarrow  null
   ^        \rightarrow  \uparrow
   \text{newNode}
   ```

   ```plaintext
   Node newNode = new Node(e)
   newNode.next = n1.next
   n1.next = newNode
   ```

   *note* This order matters

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

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

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

   \(\text{Time: } O(n)\)
What about general remove()?

```java
remove(Node n)
```

Somehow need to access n's previous node.

Let's keep this info! **Doubly Linked List**

```java
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>ArrayList</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Doubly LL</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Array</td>
<td>$O(n)$</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()
method3()
//more code

method3()

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.
Ch 3.6

Stacks: insertions, deletions performed at top (LIFO)

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

LIFO: Last-In-First-Out

Operations: push(object e): insert e at top of stack
pop(): remove & return top element
top(): return (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()  \[ \begin{array}{c}
\text{C} \\
\text{B} \\
\text{A}
\end{array} \]
\[ \Rightarrow \begin{array}{c}
\text{D} \\
\text{B} \\
\text{A}
\end{array} \]
s.pop()  
s.pop()  
s.push(d)

<Sample Code>

One Application: Balancing parentheses

[()] OK ✓  [)] is not OK!

2 types of errors: (1) type of parenth's don't match
   \# opening parenth's ≠ \# closing parenth's
1. Create empty stack.
2. Read string (of parentheses, brackets)
3. For each char c in string
  4. if c is '(', '[', or 'I':
     5. stack.push(c)
  6. else if c is ')', ']', or 'I':
     7. if stack.isEmpty() \(\Rightarrow \) error \(\text{// closing > opening}\)
     8. else
     9. if c does not match stack.pop() \(\Rightarrow \) error \(\text{// type mismatch}\)
    10. \(\Rightarrow \)
    11. if !stack.isEmpty() \(\Rightarrow \) error \(\text{// opening > closing}\)

ex: \(\left( ( \right) \) \(\Rightarrow \)

\[
\begin{array}{c}
( & \Rightarrow \text{error} \\
[ & \Rightarrow \text{error} \\
] & \Rightarrow \text{error}
\end{array}
\]

empty \(\Rightarrow \) error

---

Array Implementation of Stack

Data Fields: Object \(\text{[]}\) stack = new Object[100000] \(\Rightarrow \)

int: topOfStack \(\Rightarrow \) index of top element \(\Rightarrow \) initially \(-1\)

\(\ast\) Assume non-empty \(\Rightarrow \)

\(\text{// topOfStack}\)

(\(\Rightarrow \)

0 \(\Rightarrow \) 1 \(\Rightarrow \) 2 \(\Rightarrow \)

0 \(\Rightarrow \) b \(\Rightarrow \) c \(\Rightarrow \) x \(\Rightarrow \) y \(\Rightarrow \)

\(\Rightarrow \)