CSCI 261
Computer Science II
Review: Java's LinkedList\{E\} Class

- Java's LinkedList class
  - Must import java.util.LinkedList
  - Like ArrayList, it implements Java's List interface

- Our work on SinglyLinkedList largely implement's Java's version.
  - Java's has the following optimizations:
    - Tail reference (remember why tails are important)
    - An iterator (thumb/bookmark) to accelerate getNode()
    - Is doubly-linked: each Node knows next and previous (why?)
  - (Use Java's version in your programming assignments)
Today:

- Lists (including arrays) are very common data structures
  - So common, in fact, that other data structures are built on top of Lists

- This lecture focuses on two such data structures:
  - Stacks and Queues
  - Two of the most widely-used data structures in the field
  - Both are *more restrictive* Lists
  - Chap 4 in the K&W book
Outline

- Stacks
  - Application of Stack: ParenChecker
- Queues
- Priority Queues
  - Implementation
- Conclusion
Stacks

- A **stack** is a last-in-first-out (LIFO) List!
  - We have many real-world examples
  - Note: the last thing *pushed* on the stack is the first thing *popped* off
Stacks (Cont.)

- A stack is a last-in-first-out (LIFO) List!

- ![Diagram of a stack with push and pop operations](Image)
  - `push(…)`
  - `pop()`
  - `top of the stack`
Java has a Stack<E> class that can be used.

- Here's its API
- Feel free to use this for assignments, exams.

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean isEmpty()</td>
<td>Returns true if the stack is empty, and false otherwise</td>
</tr>
<tr>
<td>public E peek()</td>
<td>Returns a reference to the top element without removing it. Throws java.util.EmptyStackException if stack was empty prior to popping</td>
</tr>
<tr>
<td>public E pop()</td>
<td>Remove the top element and return it. Throws java.util.EmptyStackException if stack was empty prior to popping</td>
</tr>
<tr>
<td>public E push(E item)</td>
<td>Pushes the item on top of the stack, and return a reference to it</td>
</tr>
</tbody>
</table>
Usage Example

- Import the Stack class first

```
import java.util.Stack;
```

- Sample usage:

```
public void doStuff() {
    Stack<String> names = new Stack<>();
    names.push("Rich");
    names.push("Mary");
    names.push("Ross");
    names.push("Ally");

    while (!names.isEmpty()) {
        System.out.println(names.pop());
    }
}
```
What Good Is a Stack?

- Lots of uses:
  - Palindrome checker
  - Call-stacks in language implementation
  - Matching parentheses checker
  - Calculators: evaluating mathematical expressions

- Since it's a more restrictive list, why not just use a List?
  - You *could*, but...
  - Cleaner API => cleaner code
  - Better thought-process for solving the problem
    - Don't get bogged down with all of Lists' *other* methods
A palindrome is a String that can be read forward and backward

- Easy ones: "mom", "noon", "kayak", "Gig"
- Longer ones: "avid diva", "are we not drawn onward to new era"

Given a string `str`, use a stack to determine whether `str` is a palindrome.

- 2 min: Think about algorithm design
- 4 min: Find a partner to share ideas, and code it up!
  - Running time, $T(n)$?
  - Share code with class
## The String API May Help!

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public char charAt(int index)</code></td>
<td>Returns the character at the given index</td>
</tr>
<tr>
<td><code>public boolean equals(String other)</code></td>
<td>Checks content equality with given String</td>
</tr>
<tr>
<td><code>public boolean equalsIgnoreCase(String other)</code></td>
<td>Same as above, but case-insensitive</td>
</tr>
<tr>
<td><code>public int indexOf(String str)</code></td>
<td>Returns starting position of <code>str</code> if found, or <code>-1</code> if not found</td>
</tr>
<tr>
<td><code>public int length()</code></td>
<td>Gets the length of the String</td>
</tr>
<tr>
<td><code>public String replace(String target, String rep)</code></td>
<td>Returns this String resulting from replacing <code>target</code> with <code>rep</code></td>
</tr>
<tr>
<td><code>public String substring(int begin, int end)</code></td>
<td>Returns this String starting from position <code>begin</code>, ending at position <code>end-1</code></td>
</tr>
<tr>
<td><code>public String toLowerCase()</code></td>
<td>Returns this String in lower case</td>
</tr>
<tr>
<td><code>public String toUpperCase()</code></td>
<td>Returns this String in upper case</td>
</tr>
</tbody>
</table>