Outline

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

- Design-decision time

- Let's recall the behavior of stacks:
  - `pop()`, `push()`, `peek()` are all done to one end of the list
  - No methods access middle of the list
If we used a LinkedList to implement a Stack

- We can either use head or tail of the list to represent top of stack
- Know why the following running times are what they are

<table>
<thead>
<tr>
<th>Signature</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean isEmpty()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E peek()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E pop()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E push(E item)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
If we used an ArrayList to implement a Stack

- Must use the tail of the ArrayList to represent top of stack!

<table>
<thead>
<tr>
<th>Signature</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean isEmpty()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E peek()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E pop()</td>
<td>O(1)</td>
</tr>
<tr>
<td>public E push(E item)</td>
<td>O(1) amortized</td>
</tr>
</tbody>
</table>
A Practical Application: ParenChecker

Languages and calculators must accept arbitrary expressions

• One thing you'd need to do would be to ensure the parentheses match!
  - Or else, it's an illegal expression!

Legal expressions:

4 + 5 - 1

Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

1 + ((a / b) - 5 * c)

Illegal expressions:

) 

((4 + 5) - 1

Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2);
Example: Legal Expression

1 + ((a / b) - 5 * c)
Example: Illegal Expression

1 + ((a / b) - 5 * c)/(a + b)
ParenChecker Idea

- Input: An expression is a String

- Run through each character in the String
  - If it's anything besides '(' or ')' then ignore it
  - If it's an open paren '('
    - Push it on the Stack
  - If it's a close-paren ')'
    - Pop an open-paren off the stack if possible
    - If not possible, then there was one fewer open-paren up to now -- return false

- After you've seen all characters in the String,
  - Return true if stack is empty, false otherwise
Outline

- Stacks
  - Implementation of the Stack Class
  - Application of Stack: Infix Evaluation
- Queues
- Priority Queues
  - Implementation
  - Using Java's PriorityQueue
  - Comparator
- Conclusion
Useful Application: Expression Solver

- In practice algebraic expressions usually follow *infix notation*
  - As we know, parentheses explicitly determine operator precedence, but are often implied.
  - Remember this ambiguous expression from summer 2019?
    
    \[ \frac{8}{2} \times \left( 2 + 2 \right) \]
    
    This expression is interpreted as:
    
    \[ \left( \frac{8}{2} \right) \times \left( 2 + 2 \right) = 16 \]
    
    or as:
    
    \[ \left( \frac{8}{2} \right) \times \left( 2 + 2 \right) = 1 \]
    
    Wouldn't have confused so many people if the person used parentheses in the first place!
How Do We Calculate Infix Expressions?

- **Problem:** Assuming infix expressions are given with parentheses, write an algorithm to evaluate them.

\[
\left( \left( \left( 2 \times 5 \right) + 4 \right) \right) / \left( \left( 3 \times 2 \right) + 1 \right)
\]

- **Dijkstra's ("dikes-truh") Algorithm:**
  - Assume symbols are separated by space
  - Examine a symbol from left to right
    - If ",", ignore
    - If number, push it onto **value stack**
    - If operator, push it onto **operator stack**
    - If ",", pop operator stack, and pop two values. Apply operator on values, and push result onto **value stack**

**Professor Edsger Dijkstra**
(1930 - 2002)
Outline

- Stacks
- Queues
- Priority Queues
  - Implementation
- Conclusion
Queues

- A **queue** is a first-in-first-out (FIFO) List
  - One of the most widely-used data structures
  - *e.g.*, modern operating systems
  - *e.g.*, simulations

- **Operations**
  - The item **dequeued** (or **polled**) is the oldest item in the queue
  - The item **enqueued** (or **offered**) is the youngest item in the queue
Queues (Cont.)

- A *queue* is a first-in-first-out (FIFO) List!
Java's Queue\<E\> Interface

- Java has a Queue\<E\> Interface
  - This tells you there are multiple kinds of Queues in practice
  - [https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html](https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public boolean offer(E item)</code></td>
<td>Inserts the specified element in the tail of the queue, returning <code>true</code> if successful, and <code>false</code> otherwise.</td>
</tr>
<tr>
<td><code>public E remove()</code></td>
<td>Removes item at the head of the queue, and returns it. Throws <code>java.util.NoSuchElementException</code> if queue is empty.</td>
</tr>
<tr>
<td><code>public E poll()</code></td>
<td>Removes item at the head of the queue, and returns it. Returns <code>null</code> if queue is empty.</td>
</tr>
<tr>
<td><code>public E peek()</code></td>
<td>Returns item at head of the queue without removing it. Returns <code>null</code> if queue is empty.</td>
</tr>
<tr>
<td><code>public E element()</code></td>
<td>Returns item at head of the queue without removing it. Throws <code>java.util.NoSuchElementException</code> if queue is empty.</td>
</tr>
</tbody>
</table>