11.1 Circular Linked Lists

- A circular list has the tail point back to the head instead of NULL.
- This affects insertion, deletion & traversals.
- Traversals cannot use "while not NULL"
- Also can't use "while not head"
- Because you start at the head.
- Use a do-while loop.
- Set ptr to head.
- Do:
  - Traversal action.
  - Set ptr to head's next.
- While ptr is not head.

Insertion

- Head insertion.
  - Tail must be updated to point to new head.

Empty list insertion

- Creates one element list.
- Element is both head & tail.
- Must point back to self.

Deletion

- Must check for one element list to transition back to empty list.
- Cannot rely on head = head->getNext()
- "next" of one element list is itself.

11.3 Doubly Linked Lists

- Adds a previous pointer to the list node.
- Points back to the element before this one.

- Adds a "tail" pointer so List class can traverse list forward & back now.

- Also makes deletion easier because you don't have to find-previous.

Insertion

- Head insertion.
  - Set new node's previous to NULL.
  - Set new node's next to head.
  - Set head's previous to new node.
  - Set head to new node.

- Other insertions - inserting after prev.
  - Create var called next that is pointing to prev->getNext().

- So have vars: prev, next, new node.

- Set prev's next to new node.
  - If prev is tail:
    - Set tail to new node.
  - Else:
    - Set next's prev to new node.
    - Set new node's next to next.
    - Set new node's prev to prev.

Deletion
head deletion
  set tmp to head
  set head to head->getNext()
  set head's prev to NULL
  delete tmp

tail deletion
  set tmp to tail
  set tail to tail->getPrev()
  set tail's next to NULL
  delete tmp

other deletions
  set tmp to node to delete
  set next to node->getNext()
  set prev to node->getPrev()
  set prev's next to next
  set next's prev to prev
  delete tmp