Implementing Queue w/ Circularly Linked List

→ rely on intuition that the queue has a head & tail, but w/ 'next' reference of the tail linked to the 'head'

```
class CircularQueue:    (only writing some methods)
  def first(self):
    if self._is_empty():
      raise Empty
    head = self._tail._next
    return head._element

  def dequeue(self):
    oldhead = self._tail._next
    if self._size == 1:
      self._tail = None
    else
      self._tail._next = oldhead._next
    self._size -= 1
    return oldhead._element

  def enqueue(self, e):
    newest = self._Node(e, None)
    if self.is_empty()
      raise Empty
    else
      newest._next = self._tail._next
      self._tail._next = newest
    self._tail = newest
    self._size += 1

  def rotate(self):
    if self.size > 0:
      self._tail = self._tail._next
```

Doubly-Linked Lists

→ linked list in which each node keeps an explicit reference to the node before & after

→ these doubly-linked lists allow greater variety of O(1) time update operations, including insertion & deletion at arbitrary parts of the list

Header/Trailer Sentinels

→ in order to avoid special cases at boundaries of doubly-linked lists, it adds special nodes at both ends:

- header: node at beginning of the list
- trailer: node at end of list
- sentinels: the dummy nodes previously mentioned, do not store elements