```
class CircularQueue:
  """Queue implementation using circularly linked list for storage."""

  #-------------------------------------------------------------------------
----------
  # nested _Node class
  class _Node:
    """Lightweight, nonpublic class for storing a singly linked node."""
    __slots__ = '_element', '_next'         # streamline memory usage

    def __init__(self, element, next):
      self._element = element
      self._next = next

  # end of _Node class
  #-------------------------------------------------------------------------
----------

  def __init__(self):
    """Create an empty queue."""
    self._tail = None                       # will represent tail of queue
    self._size = 0                          # number of queue elements

  def __len__(self):
    """Return the number of elements in the queue."""
    return self._size

  def is_empty(self):
    """Return True if the queue is empty."""
    return self._size == 0

  def first(self):
    """Return (but do not remove) the element at the front of the queue.
    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    head = self._tail._next
    return head._element

  def dequeue(self):
    """Remove and return the first element of the queue (i.e., FIFO).
    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
```

```
      raise Empty('Queue is empty')
    oldhead = self._tail._next
    if self._size == 1:                    # removing only element
      self._tail = None                    # queue becomes empty
    else:
      self._tail._next = oldhead._next     # bypass the old head
    self._size -= 1
    return oldhead._element

  def enqueue(self, e):
    """Add an element to the back of queue."""
    newest = self._Node(e, None)           # node will be new tail node
    if self.is_empty():
      newest._next = newest                # initialize circularly
    else:
      newest._next = self._tail._next      # new node points to head
      self._tail._next = newest            # old tail points to new node
    self._tail = newest                    # new node becomes the tail
    self._size += 1
```