## Implementing a Queue w/ Singly-Linked List

```
class LinkedQueue:
  class _Node:
    implemented prior
  def __init__(self):
    self._head = None
    self._tail = None
    self._size = 0
  def __len__(self):
    return self._size
  def is_empty(self):
    return self._size == 0
  def first(self):
    if self.is_empty()
      raise Empty()
    return self._head._element
```

```
  def dequeue(self):
    if self.is_empty():
      raise Empty()
    answer = self._head._element
    self._head = self._head._next
    self._size -= 1
    if self.is_empty()
      self._tail = None
    return answer

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

### Circularly Linked Lists

→ we can have the tail of the list use its 'next' reference to point back to the 'head' of the list; called a circularly linked list

### Round-Robin Scheduler

→ iterates thru collection of elements in a circular fashion and services each element by performing a given action on it

→ for example: to fairly allocate a resource that is shared by a collection of clients.

→ round robin scheduler can be implemented using general queue ADT by repeatedly performing the following steps on the queue:

1.) e = Q.dequeue()
2.) service element e
3.) Q.enqueue(e)