Queues

What is a queue?
- Stacks reverse the order of items added to them
  - Last In First Out
  - What if we want to preserve the order in which items are added?
- Solution: queue
  - First In First Out: items removed in the same order they’re added
  - Similar to a line (i.e. a queue) at the bank, supermarket, etc.
- Uses in computer science include
  - Buffering (keyboard, network, etc.)
  - Simulations
  - Lots of other stuff

Operations on queues
- Same kinds of operations as stacks, but slightly different results
- Create
- Enqueue: add to the tail of the queue
- Dequeue: remove from the head of the queue
- Peek: look at the element at the head of the queue
- IsEmpty: tell whether the queue is empty
- DequeueAll: clear all elements from the queue

Queue using circular linked list
- Last element in list points back to the first one
- Enqueue (adding an element)
  - newNode.next = lastNode.next
  - lastNode.next = newNode
  - lastNode = newNode
- Dequeue (removing an element)
  - firstNode = lastNode.next
  - lastNode.next = firstNode.next
- Peek (look at first element)
  - firstNode = lastNode.next

Details on queues with linked lists
- Some methods can throw QueueException (like StackException)
  - Enqueue() on an empty queue
  - Peek() on an empty queue
- Enqueue() on an empty queue is a bit different
  - lastNode = newNode
  - newNode.next = newFirstNode
- Dequeue() on a queue with exactly one element is different
  - firstNode = lastNode
  - lastNode = null
- DequeueAll() can be done by lastNode = null

Queues with arrays?
- As with stacks, queues can be implemented with arrays
- Naïve implementation
  - Insert at top of array
  - Remove from bottom of array (element 0) and shift array contents down one place
  - Problem: this can be slow for large arrays!
- Better implementation: circular array
  - Keep track of start and end of queue
  - Queue “wraps around” the end of the array
  - Use modular arithmetic for array indexes
  - Space-efficient and fast
Circular arrays for queues

- Enqueue
  - queueArray[back] = item
  - count++; back = (back+1) % max_queue
- Dequeue
  - item = queueArray[front]
  - count--; front = (front+1) % max_queue
- Wraps around when front or back reaches max_queue
- NOTE: this implementation is slightly different from that in Chapter 7

Details on queues with arrays

- Some methods can throw QueueException, as with linked list queues
  - Dequeue() on an empty queue
  - Peek() on an empty queue
- Queue is empty when count==0
- If count==max_queue, queue is full
- If count==0, queue is empty
- Array-based queue can fill up!
  - Enqueue() can throw a QueueException if count==max_queue
  - Make the queue array large enough to avoid this
- DequeueAll() can be done by setting front=0, last=0, count=0
  - Same code as used to initialize an array-based queue...

Implementing queues (and stacks)

- Three choices for implementing queue ADT
  - List ADT
  - Array (circular)
  - Linked list (circular)
- List ADT is simpler: less code to write
- Array
  - Fixed maximum size
  - Low overhead (no link references)
- Linked list
  - Grows to any size
  - Requires more space for a given number of elements
- In languages other than Java, allocating and deleting elements is an issue
  - This favors arrays, which don’t need to allocate and delete very frequently (array methods don’t call new)

Queue application: simulations

- Computers often used to simulate behavior
  - Customers at a bank
  - Requests serviced by a roomful of Web servers
  - Traffic on roadways
- All of these simulations consist of events
  - An event occurs at a given time, determined by the model used in the simulation
  - Events could include
    - Car N enters Highway 1 at Morissey
    - Car N switches lanes at mile marker X
    - Car N leaves freeway at 41st Avenue
    - Simulation must keep track of thousands of events
      - Events ordered by the time they occur
      - Must process events in time order
      - Use a queue!

Sample simulation: supermarket

- N checkout lines
  - Each is FIFO
  - Each line serves the shopper at the front
  - Time to service determined by simulation
- Shopper may choose a line
  - Simulation decides how rapidly shoppers arrive
  - Simulation decides which line a shopper picks
  - Shonest line “Express” line?
  - Test different strategies
- Questions to answer:
  - How many lines should there be?
  - How should a shopper pick the best line?

Simulating a supermarket

- Each line is ordered by time
  - Customer at front of line is next to finish (in that line)
  - Amount of time to finish determined by simulation
- Simulation picks next to finish from front of all queues
  - Advances “time” to t
  - Dequeues the customer who finishes at time t
  - This repeats as long as simulation runs
- More advanced simulations may have more complex queueing
  - Time spent in each aisle
  - Time spent looking for items
  - Even more detail...