Outline of topics:

- Review. Abstraction: Complexity (Correctness?) versus Efficiency
  Illustrated last time for Stack of doubles – java.util.Stack versus “roll-my-own” versus in-line implementation
- Stacks and Queues

Stacks and queues

Stacks: Last In First Out (LIFO)  
(Often drawn as a well)

- top of stack

Create an empty stack
Push an item on the stack
Pop the top-most item
Check whether the stack isEmpty

 Peek at the top-most item
- Check whether the stack isFull

Queues: First In First Out (FIFO)  
(Usually drawn horizontally)

- front of queue
- rear of queue

Create an empty queue
Enqueue an item at the back of the queue
Dequeue the front-most item
Check whether the queue isEmpty

 Peek at the front of the queue
Check whether the queue isFull

- Abstraction – the size constraint should be respected in a LL implementation as well
- What if we do a pop() on an empty Stack?

  Preconditions represent a contract between user and the ADT. (Implementation need not be robust beyond the preconditions.)

- Implementation: arrays and Linked Lists

  - Did in class: Stack in array; Queue in LL
  - Generics in Java; create an array of Object and then cast
  - Discuss efficiency issues:

    1) Top of stack in array should be to the right (otherwise, each push, pop, is Θ(n))
    2) Queue in linked list: having each element point to the one in front of it is inefficient (the queue [a,b,c] should be stored as a-->b-->c rather than a<--b<--c)
    3) Enqueue in array should wrap-around (otherwise, Θ(n) worst-case)