Data Structures revisited

- Linked lists and arrays and ArrayLists and ...
  > Linear structures, operations include insert, delete, traverse, ...
  > Advantages and trade-offs include ...

- We want to move toward structures that support very efficient insertion and lookup, lists can't do better than $O(n)$ for one of these: consider binary search and insert for arrays, or insert and lookup for linked lists

- Interlude: two linear structures that facilitate certain algorithms: Stack and Queue
  > Restricted access linear structures

Stack: What problems does it solve?

- Stacks are used to avoid recursion, a stack can replace the implicit/actual stack of functions called recursively
- Stacks are used to evaluate arithmetic expressions, to implement compilers, to implement interpreters
  > The Java Virtual Machine (JVM) is a stack-based machine
  > Postscript is a stack-based language
  > Stacks are used to evaluate arithmetic expressions in many languages
- Small set of operations: LIFO or last in is first out access
  > Operations: push, pop, top, create, clear, size
  > More in postscript, e.g., swap, dup, rotate, ...

Simple stack example

- Stack is part of java.util.Collections hierarchy
  > It's an OO abomination, extends Vector (like ArrayList)
  > Should be implemented using Vector
  > Doesn't model "is-a" inheritance
  > what does pop do? What does push do?

```java
Stack<String> s = new Stack<String>();
s.push("panda");
s.push("grizzly");
s.push("brown");
System.out.println("size = " + s.size());
System.out.println(s.peek());
String str = s.pop();
System.out.println("size = " + s.size());
System.out.println(s.peek());
System.out.println(s.peek());
```

Implementation is very simple

- Extends Vector, so simply wraps Vector/ArrayList methods in better names
  > push==add, pop==remove
  > Note: code below for ArrayList, Vector is actually used
  > Stack is generic, so Object replaced by generic reference

```java
public Object push(Object o){
    add(o);
    return o;
}
public Object pop(Object o){
    return remove(size()-1);
}
```

```java
public Object push(Object o){
    add(o);
    return o;
}
public Object pop(Object o){
    return remove(size()-1);
}
```
Uses rather than "is-a"

- Suppose there's a private ArrayList myStorage
  - Doesn't extend Vector, simply uses Vector/ArrayList
  - Disadvantages of this approach?
    - Synchronization issues

```java
public Object push(Object o) {
    myStorage.add(o);
    return o;
}
public Object pop(Object o) {
    return myStorage.remove(size() - 1);
}
```

Postfix, prefix, and infix notation

- Postfix notation used in some HP calculators
  - No parentheses needed, precedence rules still respected
  - Read expression
    - For number/operand: push
    - For operator: pop, pop, operate, push

- See Postfix.java for example code, key ideas:
  - Use StringTokenizer, handy tool for parsing
  - Note: Exceptions thrown, what are these?

- What about prefix and infix notations, advantages?

Exceptions

- Exceptions are raised or thrown in exceptional cases
  - Bad indexes, null pointers, illegal arguments, ...
  - File not found, URL malformed, ...

- Runtime exceptions aren't meant to be handled or caught
  - Bad index in array, don't try to handle this in code
  - Null pointer stops your program, don't code that way!

- Other exceptions must be caught or rethrown
  - See FileNotFoundException and IOException in Scanner class implementation
  - RuntimeException extends Exception, catch not required

Prefix notation in action

- Scheme/LISP and other functional languages tend to use a prefix notation

```scheme
(define (square x) (* x x))
```

```scheme
(define (expt b n)
  (if (= n 0)
      1
      (* b (expt b (- n 1)))))
```
Postfix notation in action
- Practical example of use of stack abstraction
- Put operator after operands in expression
  - Use stack to evaluate
    - operand: push onto stack
    - operator: pop operands push result
- PostScript is a stack language mostly used for printing
  - drawing an X with two equivalent sets of code
    ```
    %!
    200 200 moveto
    100 100 rlineto
    200 300 moveto
    100 -100 rlineto
    stroke showpage
    %!
    100 -100 200 300 100 100 200 200 moveto rlineto moveto rlineto
    stroke showpage
    ```

Queue: another linear ADT
- FIFO: first in, first out, used in many applications
  - Scheduling jobs/processes on a computer
  - Tenting policy?
  - Computer simulations
- Common operations: add (back), remove (front), peek ??
  - `java.util.Queue` is an interface (jdk5)
    - `offer(E)`, `remove()`, `peek()`, `size()`
  - `java.util.LinkedList` implements the interface
    - `add()`, `addLast()`, `getFirst()`, `removeFirst()`
- Downside of using `LinkedList` as queue
  - Can access middle elements, remove last, etc. why?

Stack and Queue implementations
- Different implementations of queue (and stack) aren’t really interesting from an algorithmic standpoint
  - Complexity is the same, performance may change (why?)
  - Use `ArrayList`, growable array, `Vector`, linked list, ...
    - Any sequential structure
- As we’ll see `java.util.LinkedList` is good basis for all
  - In Java 5, `LinkedList` implements the `Queue` interface, low-level linked lists facilitate (circular list!)
- `ArrayList` for queue is tricky, `ring buffer` implementation,
  - add but wrap-around if possible before growing
  - Tricky to get right (exercise left to reader)

Using linear data structures
- We’ve studied arrays, stacks, queues, which to use?
  - It depends on the application
  - `ArrayList` is multipurpose, why not always use it?
    - Make it clear to programmer what’s being done
    - Other reasons?
- Other linear ADTs exist
  - List: add-to-front, add-to-back, insert anywhere, iterate
    - Alternative: `create`, `head`, `tail`, Lisp or
    - `Linked-list` nodes are concrete implementation
  - Deque: add-to-front, add-to-back, random access
    - Why is this “better” than an `ArrayList`?
    - How to implement?
Maria Klawe
Chair of Computer Science at UBC, Dean of Engineering at Princeton, President of Harvey Mudd College, ACM Fellow...

Klawe's personal interests include painting, long distance running, hiking, kayaking, juggling and playing electric guitar. She describes herself as "crazy about mathematics" and enjoys playing video games.

"I personally believe that the most important thing we have to do today is use technology to address societal problems, especially in developing regions"