Abstract Data Types

Some structures for data are so universal that we try to describe them in language- and implementation-independent way in terms of the operations for manipulating them. These are called Abstract Data Types or ADTs. They predate classes and object-oriented programming. ADTs are the original technique for abstracting data and they are still useful today.

In Java ADTs are usually represented as interfaces -- lists of method signature that need to be implemented for the data structure.

Stacks

Here is one ADT:

A stack is a data structure that implements the "LIFO" protocol -- (Last In, First Out). Data is removed from the stack in the reversal of the order in which it is entered.

The basic stack operations are

Push(x) -- add x to the stack.

Pop() -- removes the most recent unpopped addition to the stack.

Top() -- returns the top element on the stack without removing it from the stack.

isEmpty() -- returns true if the stack is empty.

Note that in the standard stack protocol the only item visible is the top element of the stack.

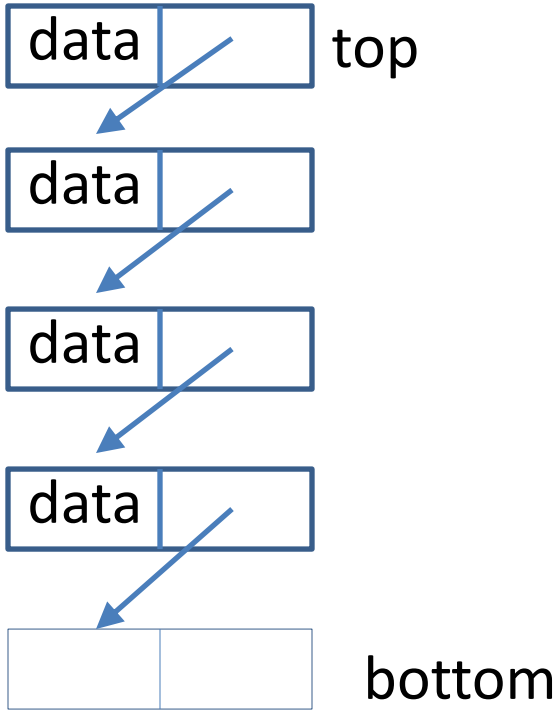
Stacks are used everywhere

- Almost all processors have a stack to support function calls
- Web browsers that interpret HTML tags use stacks
- Any system that has undo commands makes use of stacks
- There are many, many other applications

Here is a Java interface for a Stack ADT:

```
public interface StackADT<E> {
        void push(E item);
        E pop() throws EmptyStacksException;
        E top() throws NoSuchElementException
        int size();
         boolean isEmpty();
        void clear();
}
```

Here is a picture for a linked structure that implements stacks:



How would you initialize or construct a Stack? How would you write Push(), Pop(), Top() and IsEmpty()??

Queues

Here is another ADT -- a Queue. Queues implement the FIFO protocol -- First In, First Out.

The word "queue" comes from French, where it means "tail". During the French Revolution people were forced to wait in many long lines, and these lines became known as "queues". The word passed into English early in the 19th Century as a reference to a line of people waiting for something.

The Queue ADT works like a line of people waiting for a teller in a bank. One joins the Queue at the end, and exits from the Queue at the front.

The Queue ADT operations are

Enqueue(x) -- adds x to the end of the queue

Dequeue() -- removes the item at the front of the queue

Front() -- returns the item at the front of the queue

IsEmpty() -- returns true if the queue is empty.

As with stacks, there are lots of practical applications of queues.

- The operating system has a scheduler that keeps a queue of processes waiting for resources.
- Printers keep a queue of jobs waiting to print.
- Graphics cards keep a buffer queue that holds all of the drawing commands that have been issued but not yet executed.

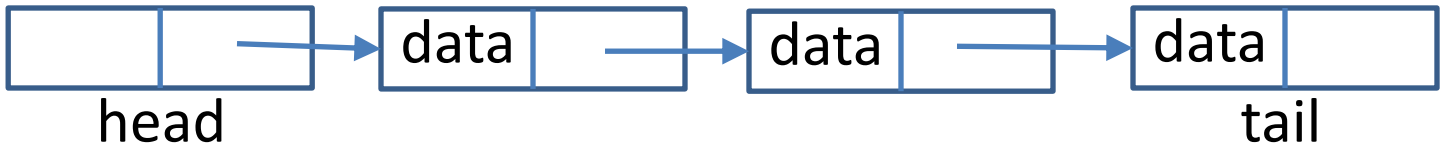
Here is a Java interface for the Queue ADT:

```
public interface QueueADT<E>
        void enqueue(E item);
        E dequeue() throws NoSuchElementException;
        E front() throws NoSuchElementException;
        int size();
        boolean isEmpty();
        void clear();
}
```

In Lab 3 we will use ArrayLists as an underlying structure to create a MyStack class that implements the Stack interface.

How will that work for Stacks? What will Push(x) correspond to? What about Pop()?

Here is a picture of a linked Queue structure:



How would you construct an empty Queue, and write Enqueue(), Dequeue(), Front() and IsEmpty() to go with this picture?