Intro to Stacks and Queues

We'll spend some time looking at stacks, and then briefly touch on queues.

Stacks are good for:
When things need to be done forwards and backwards
Recursion and backtracking, such as storing a solution to a maze problem
Some card games where you have real "stacks" of cards, as in gin

Queues are good for:
When things just have to wait for enough input to come in before processing
For instance when you have distinct parts of your program working together as in: Producer/consumer, Reader/writer, Input/output, Client/server

Both are linear structures with limited access.
Stacks have push() and pop() operations that access only the top of the stack. We may also find a use for a "peek" operation if we want to see what's on top of the stack without removing that object.

Queues have two operations – enqueue() for inserting an object on one end; dequeue() for deleting an object from the other end.

In a nutshell, stacks and queues are just special kinds of array-like aggregations of objects. (It turns out that queues are often implemented in a circular manner, but we'll save that topic for another day.)

Let's look at stacks in particular. We'll start with a Stack interface. At a reasonable minimum, we'd like these methods in any stack implementation: push, pop, peek, size, toString. But at the same time, we don't want to cheat – we should not want anything like an indexOf or any method that uses an index. These would not make sense with a stack.

Possible implementations of stacks:

- ArrayList
- LinkedList
- API Stack ! (We got lucky this time – there already is a built-in Stack. But strangely enough, no built-in Queue. So it pays to understand how to do some things by ourselves.)

How would you implement our 5 methods given a particular underlying representation such as an ArrayList? Hint: toString should print the "top" of stack first, so it should start out similar to peek and pop.

Now that we have a Stack interface and implementation(s), we need an application! There are many applications of stacks, but a good first example would be the problem of detecting balanced parentheses in a string. Here's the idea:

Push every '('
Pop a '(' when you see a ')'
Ignore all other characters

The output should say whether the input string is good or bad. How can we tell, using the stack?