My understanding of classes (OO objects) in scam have the syntax of

```
(define stk (Stack))
(stk 'push x)
(stk 'pop)
(stk 'peek)
```

And if you still want us to use

```
(define stk (Stack))
push stk x
(pop stk)
speek stk
```

then push, pop, etc... are not methods but rather functions correct?

---

Yes, these are functional stacks, no change of state. When a stack normally would be modified (say push and pop), a new stack is returned instead. Same for a queue.

---

Is it safe to assume we can modify the code from notes section and replace set with define

---

I'm pretty sure replacing set with define is not going to work (although I can't say for sure).
Subject: Re: Exercise 3  
Posted by tmurphy2 on Fri, 07 Oct 2016 20:46:36 GMT  
View Forum Message <> Reply to Message

"All methods must work in amortized constant time"

Does this mean functions like "append" and "length" cannot be used? I'm not entirely sure how they are implemented in Scam/Scheme.

Subject: Re: Exercise 3  
Posted by lusth on Sat, 08 Oct 2016 13:17:03 GMT  
View Forum Message <> Reply to Message

Append and length are theta(n). You could use these if you can guarantee enough constant time operations to average out their linear costs.

Subject: Re: Exercise 3  
Posted by padietl on Sat, 15 Oct 2016 19:13:46 GMT  
View Forum Message <> Reply to Message

In this:

```
(define (loop stack queue)
  (define x (readInt))
  (if (eof?)
      (list stack queue)
      (loop (push stack x) (enqueue queue x))
  )
)

(define (popper s)
  (cond
    (!= (ssize s) 0)
    (inspect (speek s))
    (popper (pop s))
  )
)

(define (dequeuer q)
  (cond
    (!= (qsize q) 0)
    (inspect (speek q))
    (dequeuer (dequeue q))
  )
)
```
In dequeuer you need to change (inspect (speek q)) to (inspect (qpeek q))

Also in this:

(define oldstream (setPort (open "data.ints" :read)))
(define data (loop (Stack) (Queue)))
(popper (car data))
(dequeuer (cadr data))
(setPort oldStream)

:read needs to be 'read
If stack is literally an alias for a list as in

(define (Stack) nil)

Then I'd think just returning the result of consing the new value onto the current list would be equivalent to creating a new Stack.

That's exactly my thinking, I just want to make sure it's ok with Dr. Lusth

When our code is being tested, are we guaranteed that nothing will be passed in when creating a stack and/or queue? I'm debating on making mine variadic to help with run time.

My stack and queue classes are variadic and I believe all of the methods run in amortized constant time, is it ok for our classes to be variadic?

lusth wrote on Thu, 29 September 2016 06:15
Yes, these are functional stacks, no change of state. When a stack normally would be modified (say push and pop), a new stack is returned instead. Same for a queue.

In test script for assignment 3,
Are we allowed to implement like this?

Does it suppose to be something like

   (inspect (((((Stack) 'push 2) 'push 1) 'pop) 'peek))
   (printin "It should be 1")

---

Subject: Re: Exercise 3
Posted by jrmelton on Sat, 22 Oct 2016 05:20:12 GMT
View Forum Message <> Reply to Message

All of the "methods" are actually functions. Meaning they are called with a Stack/Queue as an argument. You've implemented it where they are methods, not functions.

---

Subject: Re: Exercise 3
Posted by lusth on Sat, 22 Oct 2016 20:20:39 GMT
View Forum Message <> Reply to Message

padietl wrote on Mon, 17 October 2016 17:11

If stack is literally an alias for a list as in

   (define (Stack) nil)

Then I'd think just returning the result of consing the new value onto the current list would be equivalent to creating a new Stack.

It would be fine, except I don't see how you calculate the size of the stack in constant time.