The ADT ‘Queue’

```
module Queue ( Queue, emptyQueue, isEmptyQueue, enQueue, deQueue, front ) where

import STACK

-------------------------------------------------------------------------------
-- I N T E R F A C E  :  P U B L I C
-------------------------------------------------------------------------------

-- Queue a : a first-in first-out collection of items of type ‘a’

-------------------------------------------------------------------------------

-- emptyQueue : the empty queue

emptyQueue :: Queue a

-------------------------------------------------------------------------------

-- isEmptyQueue q : is queue ‘q’ empty ?

isEmptyQueue :: Queue a -> Bool

-------------------------------------------------------------------------------

-- enQueue x q : the queue formed by placing item ‘x’ at the back of queue ‘q’

enQueue :: a -> Queue a -> Queue a

-------------------------------------------------------------------------------

-- dequeue q : the queue formed by removing its front item
--             from the non-empty queue ‘q’

deQueue :: Queue a -> Queue a

-------------------------------------------------------------------------------

-- front q : the front item of the non-empty queue ‘q’

front :: Queue a -> a

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-- I M P L E M E N T A T I O N  :  P R I V A T E
-------------------------------------------------------------------------------

data Queue a = Q ( Stack a ) ( Stack a )

-------------------------------------------------------------------------------

emptyQueue = Q emptyStack emptyStack

-------------------------------------------------------------------------------

isEmptyQueue ( Q sf sb ) = isEmptyStack sf && isEmptyStack sb

-------------------------------------------------------------------------------

enQueue x ( Q sf sb ) = Q sf ( push x sb )

-------------------------------------------------------------------------------

deQueue ( Q sf sb ) = if isEmptyQueue ( Q sf sb ) then
                         error "Queue : ’deQueue’ called on empty queue"
                      else
                      if isEmptyStack sf then
                         Q ( pop ( invertStack sb ) ) emptyStack
                      else
                         Q ( pop sf ) sb

-------------------------------------------------------------------------------

front ( Q sf sb ) = if isEmptyQueue ( Q sf sb ) then
                       error "Queue : ’front’ called on empty queue"
                    else
                    if isEmptyStack sf then
                       top ( invertStack sb )
                    else
                       top sf

-------------------------------------------------------------------------------
```

The ADT ‘QUEUE’

```
module QUEUE ( module Queue, listToQueue, queueToList ) where

import Queue

-------------------------------------------------------------------------------
-- I N T E R F A C E  :  P U B L I C  :  all exports of module 'Queue', plus :
-------------------------------------------------------------------------------

-- listToQueue xs : the queue composed of the items of list 'xs',
--                  arranged so that the first item of 'xs' is at the front

listToQueue :: [ a ] -> Queue a

-------------------------------------------------------------------------------

-- queueToList q : the list composed of the items of queue 'q',
--                 arranged so that the front item of 'q' is first

queueToList :: Queue a -> [ a ]

-------------------------------------------------------------------------------
-- I M P L E M E N T A T I O N  :  P R I V A T E
-------------------------------------------------------------------------------

listToQueue xs = listToQueue' xs emptyQueue

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

-- listToQueue' xs q : the queue 'q' with the items of list 'xs'
--                     enqueued onto it, in order from first to last

listToQueue' :: [ a ] -> Queue a -> Queue a

listToQueue' [ ]       q = q
listToQueue' ( x : xs ) q = listToQueue' xs ( enQueue x q )

-------------------------------------------------------------------------------

queueToList q = if isEmptyQueue q then [ ]
                                  else front q : queueToList ( deQueue q )

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

$ ghci Queue.hs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 3] Compiling Stack            ( Stack.hs, interpreted )
[2 of 3] Compiling STACK            ( STACK.hs, interpreted )
[3 of 3] Compiling Queue            ( Queue.hs, interpreted )
Ok, modules loaded: Queue, STACK, Stack.

Queue> isEmptyQueue emptyQueue
True

Queue> isEmptyQueue ( enQueue 'a' emptyQueue )
False

Queue> front ( deQueue ( enQueue 'b' ( enQueue 'a' emptyQueue ) ) )
'b'

Queue> front emptyQueue
*** Exception: Queue : 'front' called on empty queue

Queue> front ( deQueue emptyQueue )
*** Exception: Queue : 'deQueue' called on empty queue

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

$ ghci QUEUE.hs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 4] Compiling Stack            ( Stack.hs, interpreted )
[2 of 4] Compiling STACK            ( STACK.hs, interpreted )
[3 of 4] Compiling Queue            ( Queue.hs, interpreted )
[4 of 4] Compiling QUEUE            ( QUEUE.hs, interpreted )
Ok, modules loaded: QUEUE, Queue, STACK, Stack.

QUEUE> queueToList ( listToQueue [ 1 .. 5 ] )
[1,2,3,4,5]

QUEUE> queueToList ( listToQueue [ 'a', 'b', 'c' ] )
"abc"

-------------------------------------------------------------------------------
```