Programming Assignment 2: Randomized Queues and Deques | queues.zip

Help Center

Submission	
Submission time	Wed-16-Sep 14:33:25
Raw Score	100.19 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  37/37 tests passed
Memory:       54/53 tests passed
Timing:       62/62 tests passed

Aggregate score: 100.19% [Correctness: 65%, Memory: 10%, Timing: 25
%, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 4.5K Sep 16 21:33 Deque.java
-rw-r--r-- 1 2.9K Sep 16 21:33 RandomizedQueue.java
-rw-r--r-- 1  846 Sep 16 21:33 Subset.java
-rw-r--r-- 1 3.0K Sep 16 21:33 studentSubmission.zip


********************************************************************
***********
```

```
*  compiling
**********************************************************************
**********


% javac Deque.java
*-----------------------------------------------------------


================================================================

% javac RandomizedQueue.java
*-----------------------------------------------------------
Note: RandomizedQueue.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

================================================================

% javac Subset.java
*-----------------------------------------------------------


================================================================


% checkstyle *.java
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
================================================================


Testing the APIs of your programs.
*-----------------------------------------------------------
Deque:

RandomizedQueue:

Subset:

================================================================
```

```
*****************************************************************
**********
*  correctness
*****************************************************************
**********

Testing methods in Deque
*-----------------------------------------------------------
Running 16 total tests.

Tests 1-6 make random calls to addFirst(), addLast(), removeFirst()
,
removeLast(), isEmpty(), and size(). The probabilities of each
operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: Calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 3: Calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
==> passed
```

```
Test 4: Calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
==> passed

Test 5: Calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 6: Calls to addFirst(), addLast(), removeFirst(),
        removeLast(), isEmpty(), and size().
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  * 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  * 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
==> passed

Test 7: Removing from an empty deque
  *  removeFirst()
  *  removeLast()
==> passed

Test 8: Create multiple deque objects at the same time
==> passed

Test 9: Check iterator() after calls only to addFirst()
```

```
==> passed

Test 10: Check iterator() after intermixed calls to addFirst(), add
Last(),
         removeFirst(), and removeLast()
==> passed

Test 11: Create two nested iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 12: Create two parallel iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 13: Create Deque objects of different parameterized types
==> passed

Test 14: Check that addFirst() and addLast() each throw a NullPoint
erException
         when inserting null items
==> passed

Test 15: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 16: Check iterator() when Deque is empty
==> passed


Total: 16/16 tests passed!

================================================================

Testing methods in RandomizedQueue
*-----------------------------------------------------------
Running 18 total tests.

Tests 1-4 make random calls to enqueue(), dequeue(), sample(),
isEmpty(), and size(). The probabilities of each operation are
(p1, p2, p3, p4, p5), respectively.
```

```
Test 1: Calls to enqueue() and size().
  *     5 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *    50 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *   500 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *  1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to enqueue() and dequeue().
  *     5 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *    50 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *   500 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *  1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *     5 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *    50 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *   500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *  1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
==> passed

Test 3: Calls to enqueue(), sample(), and size().
  *     5 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *    50 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *   500 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *  1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *     5 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *    50 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *   500 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *  1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
==> passed

Test 4: Calls to enqueue(), dequeue(), sample(), isEmpty(), and siz
e().
  *     5 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *    50 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *   500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *  1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *     5 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *    50 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *   500 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *  1000 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
==> passed

Test 5: dequeue() and sample() from an empty randomized queue
  *  dequeue()
```

```
  *  sample()
==> passed

Test 6: Create multiple randomized queue objects at the same time
==> passed

Test 7: Check that iterator() returns correct items after a sequenc
e of
        enqueue() operations
==> passed

Test 8: Check that iterator() returns correct items after sequence
of enqueue()
        and dequeue() operations
==> passed

Test 9: Create two nested iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 10: Create two parallel iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 11: Create two iterators over different randomized queues
==> passed

Test 12: Create RandomizedQueue objects of different parameterized
types
==> passed

Test 13: Check randomness of sample() by enqueueing strings, repeat
edly calling
         sample(), and counting the frequency of each value.
  *  Enqueue strings A to C and sampling 3000 times
  *  Enqueue strings A to E and sampling 5000 times
  *  Enqueue strings A to H and sampling 8000 times
  *  Enqueue strings A to J and sampling 10000 times
==> passed

Test 14: Check randomness of dequeue() by enqueueing items, repeate
dly calling
```

```
        dequeue() until a specific enqueued string appears.
  *  Enqueue strings A to C and call dequeue() until B is dequeued;
repeat 3000 times
  *  Enqueue strings A to E and call dequeue() until E is dequeued;
repeat 5000 times
  *  Enqueue strings A to H and call dequeue() until C is dequeued;
repeat 8000 times
  *  Enqueue strings A to J and call dequeue() until A is dequeued;
repeat 10000 times
==> passed

Test 15: Check randomness of iterator() by enqueueing strings, gett
ing an iterator()
        and repeatedly calling next() until a specific enqueued st
ring appears.
  *  Enqueue strings A to C, create iterator(), and call next() unt
il B is returned;
     Repeat 3000 times
  *  Enqueue strings A to E, create iterator(), and call next() unt
il D is returned;
     Repeat 5000 times
  *  Enqueue strings A to H, create iterator(), and call next() unt
il B is returned;
     Repeat 8000 times
  *  Enqueue strings A to J, create iterator(), and call next() unt
il B is returned;
     Repeat 10000 times
==> passed

Test 16: Check that NullPointerException is thrown when inserting n
ull items
==> passed

Test 17: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 18: Check iterator() when RandomizedQueue is empty
==> passed


Total: 18/18 tests passed!

================================================================
```

```
*****************************************************************
**********
*  correctness (substituting reference RandomizedQueue.java and Deq
ue.java)
*****************************************************************
**********

Testing methods in Subset
*-----------------------------------------------------------
Tests 1-3 call the main() function directly, resetting standard inp
ut
before each call.

Running 3 total tests.

Test 1: assignment inputs

% echo "A B C D E F G H I" | java Subset 3
[student solution]
D
F
B

% echo "A B C D E F G H I" | java Subset 3
[student solution]
I
G
H

% echo "AA BB BB BB BB BB CC CC " | java Subset 8
[student solution]
CC
AA
BB
CC
BB
BB
BB
BB

==> passed

Test 2: various inputs
```

```
% echo "A B C D E F G H I" | java Subset 1
[student solution]
E

% echo "A B C D E F G H I" | java Subset 5
[student solution]
I
B
A
H
F

% echo "A B C D E F G H I" | java Subset 5
[student solution]
E
A
C
G
I

% echo "A B C D E F G H I" | java Subset 9
[student solution]
A
H
E
I
B
D
F
C
G

% echo "A B C D E F G H I" | java Subset 0
[student solution]

% echo "it was the best of times it was the worst of times" | java
Subset 10
[student solution]
was
it
times
best
the
```

```
times
was
worst
it
the

% echo "It was the best of times, it was the worst of times, it was
..." | java Subset 10
[student solution]
absolute
Doctor's
introduce
flashed
to
of
with.
that
and
all

% echo "AA BB BB BB BB BB CC CC " | java Subset 7
[student solution]
CC
BB
AA
BB
BB
BB
CC

==> passed

Test 3: check that subsets are uniformly random
  * 1000 subsets of size 1 from subset10.txt
  * 250 subsets of size 4 from subset10.txt
  * 600 subsets of size 1 from subset6.txt
  * 300 subsets of size 2 from subset6.txt
  * 800 subsets of size 1 from subset8.txt
  * 160 subsets of size 5 from subset8.txt
==> passed

Total: 3/3 tests passed!

================================================================
```

```
*-----------------------------------------------------------*
**********
*  memory
*-----------------------------------------------------------*
**********

Computing memory of Subset
*-----------------------------------------------------------
Running 2 total tests.

Test 1: Check that only one Deque or RandomizedQueue object is crea
ted
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 50
==> passed

Test 2: Check that the maximum size of any Deque or RandomizedQueue
object
        created is <= N
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed

Test 3 (bonus): Check that maximum size of any or Deque or Randomiz
edQueue object
                created is <= k
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed
```

```
Total: 3/2 tests passed!

================================================================



*****************************************************************
***********
*  memory
*****************************************************************
***********

Computing memory of Deque
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a deque
containing N items is 48N + 192.

Running 28 total tests.

Test 1a-1e: Total memory usage after inserting N items,
            where N is a power of 2.

                 N        bytes
-----------------------------------------------------------
=> passed        8          424
=> passed       64         3112
=> passed      256        12328
=> passed     1024        49192
=> passed     4096       196648
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 2a-2e: Total memory usage after inserting N+1 items,
            where N is a power of 2.

                 N        bytes
-----------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
```

```
=> passed     4096       196696
==> 5/5 tests passed

Memory after adding N = 2^i + 1 items: 48.00 N + 40.00   (R^2 = 1.0
00)



Test 3a-3e: Total memory usage after inserting 2N+1 items
            and deleting N items, where N is a power of 2.

                 N        bytes
-------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 4a-4e: Total memory usage after inserting N items and then
            deleting all but one item, where N is a power of 2.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------
=> passed        8           88
=> passed       64           88
=> passed      256           88
=> passed     1024           88
=> passed     4096           88
==> 5/5 tests passed

Memory after adding N = 2^i items: 88.00   (R^2 = 1.000)



Test 5a-5e: Total memory usage of iterator after inserting N items.
            (should not grow with N or be too large of a constant)
```

```
                 N        bytes
-------------------------------------------------------------
=> passed        8           32
=> passed       64           32
=> passed      256           32
=> passed     1024           32
=> passed     4096           32
==> 5/5 tests passed

Memory of iterator after adding N = 2^i items: 32.00   (R^2 = 1.000
)



Test 6a: Insert N strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFir
st()
         and addLast() are (p1, p2), respectively. The probabilitie
s of
         removeFirst() and removeLast() are (q1, q2), respectively
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5,
0.5)
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast()
,
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
  * 100 random operations (0.8, 0.0, 0.0, 0.2)
  * 100 random operations (0.0, 0.8, 0.2, 0.0)
  * 100 random operations (0.0, 0.8, 0.0, 0.2)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 100 random operations (0.2, 0.2, 0.3, 0.3)
==> passed
```

```
Test 7: Worst-case constant memory allocated or deallocated
        per deque operation?
  * 128 random operations
  * 256 random operations
  * 512 random operations
==> passed

Total: 28/28 tests passed!

================================================================



Computing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a
randomized queue containing N items is 48N + 192.

Running 23 total tests.

Test 1a-1d: Total memory usage after inserting N integers.

                 N        bytes
-------------------------------------------------------------
=> passed       64          568
=> passed      256         2104
=> passed     1024         8248
=> passed     4096        32824
==> 4/4 tests passed


Memory: 8.00 N + 56.00   (R^2 = 1.000)



Test 2a-2d: Total memory usage after inserting N+1 items.

                 N        bytes
-------------------------------------------------------------
=> passed       64         1080
=> passed      256         4152
=> passed     1024        16440
=> passed     4096        65592
```

```
==> 4/4 tests passed


Memory: 16.00 N + 40.00   (R^2 = 1.000)



Test 3a-3d: Total memory usage after inserting 2N+1 items, and
            then deleting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64         2104
=> passed      256         8248
=> passed     1024        32824
=> passed     4096       131128
==> 4/4 tests passed


Memory: 32.00 N + 24.00   (R^2 = 1.000)


Test 4a-4d: Total memory usage after inserting N items, and
            then deleting all but one item.

                 N        bytes
-------------------------------------------------------
=> passed       64           72
=> passed      256           72
=> passed     1024           72
=> passed     4096           72
==> 4/4 tests passed


Memory: 72.00   (R^2 = 1.000)



Test 5a-5d: Total memory usage of iterator after inserting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64          576
```

```
=> passed      256         2112
=> passed     1024         8256
=> passed     4096        32832
==> 4/4 tests passed


Memory: 8.00 N + 64.00   (R^2 = 1.000)



Test 6a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 7: Insert T items into queue; then iterate over queue and chec
k
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Total: 23/23 tests passed!

================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********
```

```
Timing Deque
*------------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to addFirst(), addLast(), removeFirst()
,
             and removeLast().

                   N  seconds
-----------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
=> passed     1024000     0.14
=> passed     2048000     0.21
==> 12/12 tests passed


Test 2a-2g:  Create deque of N objects, then iterate over the N obj
ects
             by calling next() and hasNext().

                   N  seconds
-----------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.02
=> passed     2048000     0.06
==> 12/12 tests passed
```

```
Test 3a-3g:  Create deque of N objects, then interleave N calls eac
h to
             removeFirst()/removeLast() and addFirst()/addLast().

                    N  seconds
----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.01
=> passed       32767     0.01
=> passed       32768     0.01
=> passed       32769     0.01
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

```
Timing RandomizedQueue
*----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to enqueue(), sample(), dequeue(),
             isEmpty(), and size().

                    N  seconds
----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.02
=> passed      256000     0.03
=> passed      512000     0.07
=> passed     1024000     0.16
=> passed     2048000     0.39
==> 12/12 tests passed
```

```
Test 2a-2g:  Create randomized queue of N objects, then iterate
             over the N objects by calling next() and hasNext().

                    N  seconds
----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.01
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.04
=> passed     1024000     0.07
=> passed     2048000     0.17
==> 12/12 tests passed


Test 3a-3g:  Create randomized queue of N objects, then interleave
             N calls each to dequeue() and enqueue().

                    N  seconds
----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.00
=> passed       32767     0.01
=> passed       32768     0.00
=> passed       32769     0.00
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

Submission	
Submission time	Wed-16-Sep 13:50:48
Raw Score	100.00 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  37/37 tests passed
Memory:       53/53 tests passed
Timing:       62/62 tests passed

Aggregate score: 100.00% [Correctness: 65%, Memory: 10%, Timing: 25
%, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 4.5K Sep 16 20:51 Deque.java
-rw-r--r-- 1 2.9K Sep 16 20:51 RandomizedQueue.java
-rw-r--r-- 1  379 Sep 16 20:51 Subset.java
-rw-r--r-- 1 2.8K Sep 16 20:51 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********


% javac Deque.java
*-----------------------------------------------------------
```

```
================================================================

% javac RandomizedQueue.java
*-----------------------------------------------------------
Note: RandomizedQueue.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

================================================================

% javac Subset.java
*-----------------------------------------------------------

================================================================



% checkstyle *.java
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
================================================================


Testing the APIs of your programs.
*-----------------------------------------------------------
Deque:

RandomizedQueue:

Subset:

================================================================


**********************************************************************
**********
*  correctness
**********************************************************************
**********

Testing methods in Deque
```

```
Testing methods in Deque
*----------------------------------------------------------
Running 16 total tests.

Tests 1-6 make random calls to addFirst(), addLast(), removeFirst()
,
removeLast(), isEmpty(), and size(). The probabilities of each
operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: Calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 3: Calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
==> passed

Test 4: Calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
```

```
  *   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
==> passed

Test 5: Calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 6: Calls to addFirst(), addLast(), removeFirst(),
        removeLast(), isEmpty(), and size().
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  * 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  * 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
==> passed

Test 7: Removing from an empty deque
  *  removeFirst()
  *  removeLast()
==> passed

Test 8: Create multiple deque objects at the same time
==> passed

Test 9: Check iterator() after calls only to addFirst()
==> passed

Test 10: Check iterator() after intermixed calls to addFirst(), add
Last(),
         removeFirst(), and removeLast()
==> passed

Test 11: Create two nested iterators to same deque
```

```
Test 11: Create two nested iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 12: Create two parallel iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 13: Create Deque objects of different parameterized types
==> passed

Test 14: Check that addFirst() and addLast() each throw a NullPoint
erException
         when inserting null items
==> passed

Test 15: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 16: Check iterator() when Deque is empty
==> passed


Total: 16/16 tests passed!

================================================================


Testing methods in RandomizedQueue
*-----------------------------------------------------------
Running 18 total tests.

Tests 1-4 make random calls to enqueue(), dequeue(), sample(),
isEmpty(), and size(). The probabilities of each operation are
(p1, p2, p3, p4, p5), respectively.

Test 1: Calls to enqueue() and size().
  *     5 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *    50 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *   500 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *  1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
==> passed
```

```
Test 2: Calls to enqueue() and dequeue().
  *     5 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *    50 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *   500 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *  1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *     5 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *    50 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *   500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *  1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
==> passed

Test 3: Calls to enqueue(), sample(), and size().
  *     5 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *    50 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *   500 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *  1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *     5 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *    50 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *   500 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *  1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
==> passed
```

```
Test 4: Calls to enqueue(), dequeue(), sample(), isEmpty(), and siz
e().
  *     5 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *    50 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *   500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *  1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *     5 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *    50 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *   500 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *  1000 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
==> passed

Test 5: dequeue() and sample() from an empty randomized queue
  *  dequeue()
  *  sample()
==> passed

Test 6: Create multiple randomized queue objects at the same time
==> passed

Test 7: Check that iterator() returns correct items after a sequenc
e of
```

```
e of
        enqueue() operations
==> passed

Test 8: Check that iterator() returns correct items after sequence
of enqueue()
        and dequeue() operations
==> passed

Test 9: Create two nested iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 10: Create two parallel iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 11: Create two iterators over different randomized queues
==> passed

Test 12: Create RandomizedQueue objects of different parameterized
types
==> passed

Test 13: Check randomness of sample() by enqueueing strings, repeat
edly calling
         sample(), and counting the frequency of each value.
  *  Enqueue strings A to C and sampling 3000 times
  *  Enqueue strings A to E and sampling 5000 times
  *  Enqueue strings A to H and sampling 8000 times
  *  Enqueue strings A to J and sampling 10000 times
==> passed

Test 14: Check randomness of dequeue() by enqueueing items, repeate
dly calling
         dequeue() until a specific enqueued string appears.
  *  Enqueue strings A to C and call dequeue() until A is dequeued;
repeat 3000 times
  *  Enqueue strings A to E and call dequeue() until A is dequeued;
repeat 5000 times
  *  Enqueue strings A to H and call dequeue() until F is dequeued;
repeat 8000 times
```

```
  *  Enqueue strings A to J and call dequeue() until D is dequeued,
repeat 10000 times
==> passed

Test 15: Check randomness of iterator() by enqueueing strings, gett
ing an iterator()
         and repeatedly calling next() until a specific enqueued st
ring appears.
  *  Enqueue strings A to C, create iterator(), and call next() unt
il A is returned;
     Repeat 3000 times
  *  Enqueue strings A to E, create iterator(), and call next() unt
il B is returned;
     Repeat 5000 times
  *  Enqueue strings A to H, create iterator(), and call next() unt
il H is returned;
     Repeat 8000 times
  *  Enqueue strings A to J, create iterator(), and call next() unt
il D is returned;
     Repeat 10000 times
==> passed

Test 16: Check that NullPointerException is thrown when inserting n
ull items
==> passed

Test 17: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 18: Check iterator() when RandomizedQueue is empty
==> passed


Total: 18/18 tests passed!

================================================================

*****************************************************************
***********
*  correctness (substituting reference RandomizedQueue.java and Deq
ue.java)
*****************************************************************
***********
```

```
Testing methods in Subset
*-----------------------------------------------------------
Tests 1-3 call the main() function directly, resetting standard inp
ut
before each call.

Running 3 total tests.

Test 1: assignment inputs

% echo "A B C D E F G H I" | java Subset 3
[student solution]
H
I
E

% echo "A B C D E F G H I" | java Subset 3
[student solution]
D
I
E
```

```
% echo "AA BB BB BB BB BB CC CC " | java Subset 8
[student solution]
CC
BB
BB
AA
BB
BB
CC
BB

==> passed

Test 2: various inputs

% echo "A B C D E F G H I" | java Subset 1
[student solution]
G

% echo "A B C D E F G H I" | java Subset 5
[student solution]
```

```
E
G
H
C
D

% echo "A B C D E F G H I" | java Subset 5
[student solution]
C
E
A
F
G

% echo "A B C D E F G H I" | java Subset 9
[student solution]
E
H
I
D
C
B
G
F
A

% echo "A B C D E F G H I" | java Subset 0
[student solution]

% echo "it was the best of times it was the worst of times" | java
Subset 10
[student solution]
was
times
the
it
of
it
the
times
best
of

% echo "It was the best of times, it was the worst of times, it was
```

```
...  | java Subset 10
[student solution]
in
a
integrity,
its
the
to
possessions,
black
and
That

% echo "AA BB BB BB BB BB CC CC " | java Subset 7
[student solution]
BB
BB
CC
AA
BB
BB
CC

==> passed

Test 3: check that subsets are uniformly random
  * 1000 subsets of size 1 from subset10.txt
  * 250 subsets of size 4 from subset10.txt
  * 600 subsets of size 1 from subset6.txt
  * 300 subsets of size 2 from subset6.txt
  * 800 subsets of size 1 from subset8.txt
  * 160 subsets of size 5 from subset8.txt
==> passed

Total: 3/3 tests passed!

================================================================

*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Subset
```

```
Computing memory of Subset
*-----------------------------------------------------------
Running 2 total tests.

Test 1: Check that only one Deque or RandomizedQueue object is crea
ted
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 50
==> passed

Test 2: Check that the maximum size of any Deque or RandomizedQueue
object
        created is <= N
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed

Test 3 (bonus): Check that maximum size of any or Deque or Randomiz
edQueue object
                created is <= k
  * filename = tale.txt, N = 138653, k = 5
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 500
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 5000
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50000
    - max size of RandomizedQueue object = 138653
==> FAILED

Total: 2/2 tests passed!
```

```
================================================================


*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Deque
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a deque
containing N items is 48N + 192.

Running 28 total tests.

Test 1a-1e: Total memory usage after inserting N items,
            where N is a power of 2.

                 N        bytes
----------------------------------------------------------
=> passed        8          424
=> passed       64         3112
=> passed      256        12328
=> passed     1024        49192
=> passed     4096       196648
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 2a-2e: Total memory usage after inserting N+1 items,
            where N is a power of 2.

                 N        bytes
----------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed
```

```
Memory after adding N = 2^i + 1 items: 48.00 N + 40.00   (R^2 = 1.0
00)



Test 3a-3e: Total memory usage after inserting 2N+1 items
            and deleting N items, where N is a power of 2.

                 N        bytes
-------------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 4a-4e: Total memory usage after inserting N items and then
            deleting all but one item, where N is a power of 2.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------------
=> passed        8           88
=> passed       64           88
=> passed      256           88
=> passed     1024           88
=> passed     4096           88
==> 5/5 tests passed

Memory after adding N = 2^i items: 88.00   (R^2 = 1.000)



Test 5a-5e: Total memory usage of iterator after inserting N items.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------------
=> passed        8           32
```

```
=> passed        8           32
=> passed       64           32
=> passed      256           32
=> passed     1024           32
=> passed     4096           32
==> 5/5 tests passed

Memory of iterator after adding N = 2^i items: 32.00   (R^2 = 1.000
)



Test 6a: Insert N strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFir
st()
         and addLast() are (p1, p2), respectively. The probabilitie
s of
         removeFirst() and removeLast() are (q1, q2), respectively
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5,
0.5)
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast()
,
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
  * 100 random operations (0.8, 0.0, 0.0, 0.2)
  * 100 random operations (0.0, 0.8, 0.2, 0.0)
  * 100 random operations (0.0, 0.8, 0.0, 0.2)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 100 random operations (0.2, 0.2, 0.3, 0.3)
==> passed

Test 7: Worst-case constant memory allocated or deallocated
```

```
        per deque operation?
  * 128 random operations
  * 256 random operations
  * 512 random operations
==> passed

Total: 28/28 tests passed!

================================================================



Computing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a
randomized queue containing N items is 48N + 192.

Running 23 total tests.

Test 1a-1d: Total memory usage after inserting N integers.

                N        bytes
-----------------------------------------------------------
=> passed       64          568
=> passed      256         2104
=> passed     1024         8248
=> passed     4096        32824
==> 4/4 tests passed


Memory: 8.00 N + 56.00   (R^2 = 1.000)



Test 2a-2d: Total memory usage after inserting N+1 items.

                N        bytes
-----------------------------------------------------------
=> passed       64         1080
=> passed      256         4152
=> passed     1024        16440
=> passed     4096        65592
==> 4/4 tests passed
```

```
Memory: 16.00 N + 40.00   (R^2 = 1.000)



Test 3a-3d: Total memory usage after inserting 2N+1 items, and
            then deleting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64         2104
=> passed      256         8248
=> passed     1024        32824
=> passed     4096       131128
==> 4/4 tests passed


Memory: 32.00 N + 24.00   (R^2 = 1.000)



Test 4a-4d: Total memory usage after inserting N items, and
            then deleting all but one item.

                 N        bytes
-------------------------------------------------------
=> passed       64           72
=> passed      256           72
=> passed     1024           72
=> passed     4096           72
==> 4/4 tests passed


Memory: 72.00   (R^2 = 1.000)



Test 5a-5d: Total memory usage of iterator after inserting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64          576
=> passed      256         2112
=> passed     1024         8256
=> passed     4096        32832
```

```
=> passed     4096        32832
==> 4/4 tests passed


Memory: 8.00 N + 64.00   (R^2 = 1.000)



Test 6a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 7: Insert T items into queue; then iterate over queue and chec
k
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Total: 23/23 tests passed!

================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********

Timing Deque
*-----------------------------------------------------------
Running 31 total tests.
```

```
Running 31 total tests.

Test 1a-1g:  N random calls to addFirst(), addLast(), removeFirst()
,
             and removeLast().

                    N  seconds
-----------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
=> passed     1024000     0.14
=> passed     2048000     0.23
==> 12/12 tests passed


Test 2a-2g:  Create deque of N objects, then iterate over the N obj
ects
             by calling next() and hasNext().

                    N  seconds
-----------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.02
=> passed     2048000     0.05
==> 12/12 tests passed


Test 3a-3g:  Create deque of N objects, then interleave N calls eac
```

```
Test 3a-3g:  Create deque of N objects, then interleave N calls each h to
             removeFirst()/removeLast() and addFirst()/addLast().

                    N  seconds
----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.01
=> passed       32767     0.01
=> passed       32768     0.01
=> passed       32769     0.01
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================



Timing RandomizedQueue
*----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to enqueue(), sample(), dequeue(),
             isEmpty(), and size().

                    N  seconds
----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.02
=> passed      256000     0.03
=> passed      512000     0.06
=> passed     1024000     0.14
=> passed     2048000     0.36
==> 12/12 tests passed


Test 2a-2g:  Create randomized queue of N objects, then iterate
```

```
Test 2a-2g:  Create randomized queue of N objects, then iterate
             over the N objects by calling next() and hasNext().

                    N  seconds
----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.01
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.04
=> passed     1024000     0.07
=> passed     2048000     0.16
==> 12/12 tests passed


Test 3a-3g:  Create randomized queue of N objects, then interleave
             N calls each to dequeue() and enqueue().

                    N  seconds
----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.00
=> passed       32767     0.01
=> passed       32768     0.00
=> passed       32769     0.00
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

Submission

Submission	Wed-16-Sep 13:44:17

time	
Raw Score	99.25 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  37/37 tests passed
Memory:       49/53 tests passed
Timing:       62/62 tests passed

Aggregate score: 99.25% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 4.5K Sep 16 20:44 Deque.java
-rw-r--r-- 1 2.8K Sep 16 20:44 RandomizedQueue.java
-rw-r--r-- 1  379 Sep 16 20:44 Subset.java
-rw-r--r-- 1 2.8K Sep 16 20:44 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********


% javac Deque.java
*-----------------------------------------------------------


================================================================
```

```
% javac RandomizedQueue.java
*-----------------------------------------------------------
Note: RandomizedQueue.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

================================================================

% javac Subset.java
*-----------------------------------------------------------

================================================================



% checkstyle *.java
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
================================================================


Testing the APIs of your programs.
*-----------------------------------------------------------
Deque:

RandomizedQueue:

Subset:

================================================================


**********************************************************************
**********
*  correctness
**********************************************************************
**********

Testing methods in Deque
*-----------------------------------------------------------
Running 16 total tests.
```

```
Tests 1-6 make random calls to addFirst(), addLast(), removeFirst()
,
removeLast(), isEmpty(), and size(). The probabilities of each
operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: Calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 3: Calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
==> passed

Test 4: Calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
```

```
==> passed

Test 5: Calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
==> passed

Test 6: Calls to addFirst(), addLast(), removeFirst(),
        removeLast(), isEmpty(), and size().
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  * 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  *  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  * 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
==> passed

Test 7: Removing from an empty deque
  *  removeFirst()
  *  removeLast()
==> passed

Test 8: Create multiple deque objects at the same time
==> passed

Test 9: Check iterator() after calls only to addFirst()
==> passed

Test 10: Check iterator() after intermixed calls to addFirst(), add
Last(),
         removeFirst(), and removeLast()
==> passed

Test 11: Create two nested iterators to same deque
  *  N = 10
  *  N = 1000
```

```
==> passed

Test 12: Create two parallel iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 13: Create Deque objects of different parameterized types
==> passed

Test 14: Check that addFirst() and addLast() each throw a NullPoint
erException
         when inserting null items
==> passed

Test 15: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 16: Check iterator() when Deque is empty
==> passed

Total: 16/16 tests passed!

================================================================

Testing methods in RandomizedQueue
*-----------------------------------------------------------
Running 18 total tests.

Tests 1-4 make random calls to enqueue(), dequeue(), sample(),
isEmpty(), and size(). The probabilities of each operation are
(p1, p2, p3, p4, p5), respectively.

Test 1: Calls to enqueue() and size().
  *     5 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *    50 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *   500 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *  1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to enqueue() and dequeue().
  *     5 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
```

```
  *    50 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *   500 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *  1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *     5 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *    50 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *   500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *  1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
==> passed

Test 3: Calls to enqueue(), sample(), and size().
  *     5 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *    50 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *   500 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *  1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *     5 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *    50 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *   500 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *  1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
==> passed

Test 4: Calls to enqueue(), dequeue(), sample(), isEmpty(), and siz
e().
  *     5 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *    50 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *   500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *  1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *     5 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *    50 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *   500 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *  1000 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
==> passed

Test 5: dequeue() and sample() from an empty randomized queue
  *  dequeue()
  *  sample()
==> passed

Test 6: Create multiple randomized queue objects at the same time
==> passed

Test 7: Check that iterator() returns correct items after a sequenc
e of
        enqueue() operations
==> passed
```

```
Test 8: Check that iterator() returns correct items after sequence
of enqueue()
        and dequeue() operations
==> passed

Test 9: Create two nested iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 10: Create two parallel iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 11: Create two iterators over different randomized queues
==> passed

Test 12: Create RandomizedQueue objects of different parameterized
types
==> passed

Test 13: Check randomness of sample() by enqueueing strings, repeat
edly calling
         sample(), and counting the frequency of each value.
  *  Enqueue strings A to C and sampling 3000 times
  *  Enqueue strings A to E and sampling 5000 times
  *  Enqueue strings A to H and sampling 8000 times
  *  Enqueue strings A to J and sampling 10000 times
==> passed

Test 14: Check randomness of dequeue() by enqueueing items, repeate
dly calling
         dequeue() until a specific enqueued string appears.
  *  Enqueue strings A to C and call dequeue() until A is dequeued;
repeat 3000 times
  *  Enqueue strings A to E and call dequeue() until E is dequeued;
repeat 5000 times
  *  Enqueue strings A to H and call dequeue() until B is dequeued;
repeat 8000 times
  *  Enqueue strings A to J and call dequeue() until F is dequeued;
repeat 10000 times
==> passed
```

```
Test 15: Check randomness of iterator() by enqueueing strings, gett
ing an iterator()
         and repeatedly calling next() until a specific enqueued st
ring appears.
  *  Enqueue strings A to C, create iterator(), and call next() unt
il B is returned;
     Repeat 3000 times
  *  Enqueue strings A to E, create iterator(), and call next() unt
il D is returned;
     Repeat 5000 times
  *  Enqueue strings A to H, create iterator(), and call next() unt
il B is returned;
     Repeat 8000 times
  *  Enqueue strings A to J, create iterator(), and call next() unt
il F is returned;
     Repeat 10000 times
==> passed

Test 16: Check that NullPointerException is thrown when inserting n
ull items
==> passed

Test 17: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 18: Check iterator() when RandomizedQueue is empty
==> passed


Total: 18/18 tests passed!

================================================================

*****************************************************************
**********
*  correctness (substituting reference RandomizedQueue.java and Deq
ue.java)
*****************************************************************
**********

Testing methods in Subset
*-----------------------------------------------------------
```

```
Tests 1-3 call the main() function directly, resetting standard inp
ut
before each call.

Running 3 total tests.

Test 1: assignment inputs

% echo "A B C D E F G H I" | java Subset 3
[student solution]
A
H
I

% echo "A B C D E F G H I" | java Subset 3
[student solution]
H
C
A

% echo "AA BB BB BB BB BB CC CC " | java Subset 8
[student solution]
CC
BB
CC
BB
BB
BB
AA
BB

==> passed

Test 2: various inputs

% echo "A B C D E F G H I" | java Subset 1
[student solution]
I

% echo "A B C D E F G H I" | java Subset 5
[student solution]
E
F
I
```

```
C
G

% echo "A B C D E F G H I" | java Subset 5
[student solution]
H
F
C
E
G

% echo "A B C D E F G H I" | java Subset 9
[student solution]
C
F
A
B
E
G
D
I
H

% echo "A B C D E F G H I" | java Subset 0
[student solution]

% echo "it was the best of times it was the worst of times" | java
Subset 10
[student solution]
of
times
it
it
times
the
the
was
worst
was

% echo "It was the best of times, it was the worst of times, it was
..." | java Subset 10
[student solution]
child,
```

```
and
knee
so
action,
course.
babble
Mr.
packing,
his

% echo "AA BB BB BB BB BB CC CC " | java Subset 7
[student solution]
CC
BB
BB
BB
BB
AA
BB

==> passed

Test 3: check that subsets are uniformly random
  * 1000 subsets of size 1 from subset10.txt
  * 250 subsets of size 4 from subset10.txt
  * 600 subsets of size 1 from subset6.txt
  * 300 subsets of size 2 from subset6.txt
  * 800 subsets of size 1 from subset8.txt
  * 160 subsets of size 5 from subset8.txt
==> passed

Total: 3/3 tests passed!

================================================================

**********************************************************************
**********
*  memory
**********************************************************************
**********

Computing memory of Subset
*-----------------------------------------------------
Running 2 total tests.
```

```
Test 1: Check that only one Deque or RandomizedQueue object is crea
ted
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 50
==> passed

Test 2: Check that the maximum size of any Deque or RandomizedQueue
object
        created is <= N
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed

Test 3 (bonus): Check that maximum size of any or Deque or Randomiz
edQueue object
                created is <= k
  * filename = tale.txt, N = 138653, k = 5
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 500
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 5000
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50000
    - max size of RandomizedQueue object = 138653
==> FAILED

Total: 2/2 tests passed!

================================================================
```

```
*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Deque
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a deque
containing N items is 48N + 192.

Running 28 total tests.

Test 1a-1e: Total memory usage after inserting N items,
            where N is a power of 2.

                 N        bytes
-----------------------------------------------------------
=> passed         8          424
=> passed        64         3112
=> passed       256        12328
=> passed      1024        49192
=> passed      4096       196648
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 2a-2e: Total memory usage after inserting N+1 items,
            where N is a power of 2.

                 N        bytes
-----------------------------------------------------------
=> passed         8          472
=> passed        64         3160
=> passed       256        12376
=> passed      1024        49240
=> passed      4096       196696
==> 5/5 tests passed

Memory after adding N = 2^i + 1 items: 48.00 N + 40.00   (R^2 = 1.0
00)
```

```
Test 3a-3e: Total memory usage after inserting 2N+1 items
            and deleting N items, where N is a power of 2.

                 N        bytes
-------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 4a-4e: Total memory usage after inserting N items and then
            deleting all but one item, where N is a power of 2.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------
=> passed        8           88
=> passed       64           88
=> passed      256           88
=> passed     1024           88
=> passed     4096           88
==> 5/5 tests passed

Memory after adding N = 2^i items: 88.00   (R^2 = 1.000)



Test 5a-5e: Total memory usage of iterator after inserting N items.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------
=> passed        8           32
=> passed       64           32
=> passed      256           32
```

```
=> passed     1024           32
=> passed     4096           32
==> 5/5 tests passed

Memory of iterator after adding N = 2^i items: 32.00   (R^2 = 1.000
)



Test 6a: Insert N strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFir
st()
         and addLast() are (p1, p2), respectively. The probabilitie
s of
         removeFirst() and removeLast() are (q1, q2), respectively
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5,
0.5)
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast()
,
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
  * 100 random operations (0.8, 0.0, 0.0, 0.2)
  * 100 random operations (0.0, 0.8, 0.2, 0.0)
  * 100 random operations (0.0, 0.8, 0.0, 0.2)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 100 random operations (0.2, 0.2, 0.3, 0.3)
==> passed

Test 7: Worst-case constant memory allocated or deallocated
        per deque operation?
  * 128 random operations
  * 256 random operations
```

```
  * 512 random operations
==> passed

Total: 28/28 tests passed!

================================================================



Computing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a
randomized queue containing N items is 48N + 192.

Running 23 total tests.

Test 1a-1d: Total memory usage after inserting N integers.

                 N        bytes
-------------------------------------------------------------
=> passed       64          568
=> passed      256         2104
=> passed     1024         8248
=> passed     4096        32824
==> 4/4 tests passed


Memory: 8.00 N + 56.00   (R^2 = 1.000)



Test 2a-2d: Total memory usage after inserting N+1 items.

                 N        bytes
-------------------------------------------------------------
=> passed       64         1080
=> passed      256         4152
=> passed     1024        16440
=> passed     4096        65592
==> 4/4 tests passed


Memory: 16.00 N + 40.00   (R^2 = 1.000)
```

```
Test 3a-3d: Total memory usage after inserting 2N+1 items, and
            then deleting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64         2104
=> passed      256         8248
=> passed     1024        32824
=> passed     4096       131128
==> 4/4 tests passed


Memory: 32.00 N + 24.00   (R^2 = 1.000)



Test 4a-4d: Total memory usage after inserting N items, and
            then deleting all but one item.

                 N        bytes
-------------------------------------------------------
=> FAILED       64          568   (2.4x)
=> FAILED      256         2104   (8.8x)
=> FAILED     1024         8248  (34.4x)
=> FAILED     4096        32824 (136.8x)
==> 0/4 tests passed


Memory: 11965.82   (R^2 = 0.000)



Test 5a-5d: Total memory usage of iterator after inserting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64          576
=> passed      256         2112
=> passed     1024         8256
=> passed     4096        32832
==> 4/4 tests passed
```

```
Memory: 8.00 N + 64.00   (R^2 = 1.000)



Test 6a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 7: Insert T items into queue; then iterate over queue and chec
k
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Total: 19/23 tests passed!

================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********

Timing Deque
*----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to addFirst(), addLast(), removeFirst()
```

```
,
             and removeLast().

                    N  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
=> passed     1024000     0.14
=> passed     2048000     0.21
==> 12/12 tests passed


Test 2a-2g:  Create deque of N objects, then iterate over the N obj
ects
             by calling next() and hasNext().

                    N  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.02
=> passed     2048000     0.05
==> 12/12 tests passed


Test 3a-3g:  Create deque of N objects, then interleave N calls eac
h to
             removeFirst()/removeLast() and addFirst()/addLast().
```

```
                     N  seconds
-----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.01
=> passed       32767     0.01
=> passed       32768     0.01
=> passed       32769     0.01
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================



Timing RandomizedQueue
*-----------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to enqueue(), sample(), dequeue(),
             isEmpty(), and size().

                     N  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.02
=> passed      256000     0.04
=> passed      512000     0.06
=> passed     1024000     0.14
=> passed     2048000     0.40
==> 12/12 tests passed


Test 2a-2g:  Create randomized queue of N objects, then iterate
             over the N objects by calling next() and hasNext().
```

```
                    N  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.01
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.04
=> passed     1024000     0.07
=> passed     2048000     0.20
==> 12/12 tests passed


Test 3a-3g:  Create randomized queue of N objects, then interleave
             N calls each to dequeue() and enqueue().

                    N  seconds
-----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.00
=> passed       32767     0.01
=> passed       32768     0.00
=> passed       32769     0.00
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

Submission	
Submission time	Wed-16-Sep 13:38:15
Raw Score	83.25 / 100.00

Feedback

See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        FAILED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  28/37 tests passed
Memory:       48/53 tests passed
Timing:       62/62 tests passed

Aggregate score: 83.25% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 4.5K Sep 16 20:38 Deque.java
-rw-r--r-- 1 2.8K Sep 16 20:38 RandomizedQueue.java
-rw-r--r-- 1  410 Sep 16 20:38 Subset.java
-rw-r--r-- 1 2.8K Sep 16 20:38 studentSubmission.zip


********************************************************************
**********
*  compiling
********************************************************************
**********


% javac Deque.java
*-----------------------------------------------------------


================================================================


% javac RandomizedQueue.java
*-----------------------------------------------------------
Note: RandomizedQueue.java uses unchecked or unsafe operations.
```

```
Note: Recompile with -Xlint:unchecked for details.

================================================================

% javac Subset.java
*----------------------------------------------------------


================================================================



% checkstyle *.java
*----------------------------------------------------------
Deque.java:123:30: ',' is not followed by whitespace.
Deque.java:123:34: ',' is not followed by whitespace.
Checkstyle ends with 2 errors.
================================================================


% findbugs *.class
*----------------------------------------------------------
================================================================


Testing the APIs of your programs.
*----------------------------------------------------------
Deque:

RandomizedQueue:

Subset:

================================================================


**********************************************************************
**********
*  correctness
**********************************************************************
**********

Testing methods in Deque
*----------------------------------------------------------
Running 16 total tests.
```

```
Tests 1-6 make random calls to addFirst(), addLast(), removeFirst()
,
removeLast(), isEmpty(), and size(). The probabilities of each
operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: Calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
    - student   isEmpty() returned false
    - reference isEmpty() returned true
    - sequence of dequeue operations was:
        deque.addFirst(0)
        deque.removeFirst()     ==> 0
        deque.isEmpty()

  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
    - student   isEmpty() returned false
    - reference isEmpty() returned true
    - sequence of dequeue operations was:
        deque.addFirst(0)
        deque.isEmpty()
        deque.isEmpty()
        deque.removeFirst()     ==> 0
        deque.isEmpty()

  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
    - student   isEmpty() returned false
    - reference isEmpty() returned true
    - sequence of dequeue operations was:
        deque.isEmpty()
        deque.isEmpty()
        deque.isEmpty()
        deque.isEmpty()
```

```
        deque.isEmpty()
        deque.isEmpty()
        deque.isEmpty()
        deque.addFirst(7)
        deque.removeFirst()     ==> 7
        deque.isEmpty()

  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
    - student   isEmpty() returned false
    - reference isEmpty() returned true
    - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addFirst(1)
        deque.addFirst(2)
        deque.removeFirst()     ==> 2
        deque.removeFirst()     ==> 1
        deque.addFirst(5)
        deque.removeFirst()     ==> 5
        deque.isEmpty()

==> FAILED

Test 3: Calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  *   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
     java.lang.NullPointerException
     Deque$Node.access$200(Deque.java:11)
     Deque.removeLast(Deque.java:79)
     TestDeque.random(TestDeque.java:87)
     TestDeque.test3(TestDeque.java:185)
     TestDeque.main(TestDeque.java:741)

    - sequence of dequeue operations was:
        deque.addFirst(0)
        deque.removeLast()      ==> 0
        deque.addFirst(2)
        deque.removeLast()
  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
     java.lang.NullPointerException
     Deque$Node.access$200(Deque.java:11)
```

```
    Deque.removeLast(Deque.java:79)
    TestDeque.random(TestDeque.java:87)
    TestDeque.test3(TestDeque.java:186)
    TestDeque.main(TestDeque.java:741)

  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addFirst(1)
        deque.removeLast()      ==> 1
        deque.addFirst(3)
        deque.removeLast()
* 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
    java.lang.NullPointerException
    Deque$Node.access$200(Deque.java:11)
    Deque.removeLast(Deque.java:79)
    TestDeque.random(TestDeque.java:87)
    TestDeque.test3(TestDeque.java:187)
    TestDeque.main(TestDeque.java:741)

  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addFirst(1)
        deque.isEmpty()
        deque.removeLast()      ==> 1
        deque.addFirst(4)
        deque.removeLast()
==> FAILED

Test 4: Calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
    - student   isEmpty() returned false
    - reference isEmpty() returned true
    - sequence of dequeue operations was:
        deque.isEmpty()
        deque.isEmpty()
        deque.addLast(2)
        deque.removeLast()      ==> 2
        deque.isEmpty()

  *   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
```

```
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
       deque.isEmpty()
       deque.isEmpty()
       deque.addLast(2)
       deque.isEmpty()
       deque.removeLast()      ==> 2
       deque.isEmpty()

*  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
       deque.addLast(0)
       deque.isEmpty()
       deque.removeLast()      ==> 0
       deque.isEmpty()

* 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
       deque.addLast(0)
       deque.removeLast()      ==> 0
       deque.isEmpty()

==> FAILED

Test 5: Calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  * 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
     java.lang.NullPointerException
     Deque$Node.access$300(Deque.java:11)
     Deque.removeFirst(Deque.java:65)
     TestDeque.random(TestDeque.java:69)
     TestDeque.test5(TestDeque.java:217)
     TestDeque.main(TestDeque.java:743)

    - sequence of dequeue operations was:
```

```
        deque.addLast(0)
        deque.isEmpty()
        deque.removeFirst()     ==> 0
        deque.addLast(3)
        deque.removeFirst()
*  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addLast(1)
        deque.removeFirst()     ==> 1
        deque.isEmpty()

* 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addLast(1)
        deque.removeFirst()     ==> 1
        deque.isEmpty()

==> FAILED

Test 6: Calls to addFirst(), addLast(), removeFirst(),
        removeLast(), isEmpty(), and size().
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
    - student   size() returned 3
    - reference size() returned 2
    - sequence of dequeue operations was:
        deque.addFirst(0)
        deque.addLast(1)
        deque.addLast(2)
        deque.removeFirst()     ==> 0
        deque.size()

  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
    - student   size() returned 18
    - reference size() returned 17

  * 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
    - student   size() returned 15
```

```
  - reference size() returned 14

*    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
*   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  - student   isEmpty() returned false
  - reference isEmpty() returned true
  - sequence of dequeue operations was:
       deque.addLast(0)
       deque.size()
       deque.isEmpty()
       deque.removeFirst()     ==> 0
       deque.isEmpty()

*  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  - student   size() returned 3
  - reference size() returned 1
  - sequence of dequeue operations was:
       deque.isEmpty()
       deque.isEmpty()
       deque.size()
       deque.addFirst(3)
       deque.isEmpty()
       deque.removeLast()      ==> 3
       deque.addLast(6)
       deque.addLast(7)
       deque.isEmpty()
       deque.removeLast()      ==> 7
       deque.isEmpty()
       deque.size()

* 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
   java.lang.NullPointerException
   Deque$Node.access$300(Deque.java:11)
   Deque.removeFirst(Deque.java:65)
   TestDeque.random(TestDeque.java:69)
   TestDeque.test6(TestDeque.java:237)
   TestDeque.main(TestDeque.java:744)

  - sequence of dequeue operations was:
       deque.isEmpty()
       deque.addFirst(1)
       deque.removeFirst()     ==> 1
       deque.addLast(3)
       deque.addLast(4)
```

```
         deque.removeFirst()
==> FAILED

Test 7: Removing from an empty deque
  *  removeFirst()
  *  removeLast()
==> passed

Test 8: Create multiple deque objects at the same time
  size() returns wrong result
    - student   = 10
    - reference = 9
  Failed on 0th removeFirst() operation in deque 1
  size() returns wrong result
    - student   = 1000
    - reference = 999
  Failed on 0th removeFirst() operation in deque 1
==> FAILED

Test 9: Check iterator() after calls only to addFirst()
==> passed

Test 10: Check iterator() after intermixed calls to addFirst(), add
Last(),
         removeFirst(), and removeLast()
==> passed

Test 11: Create two nested iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 12: Create two parallel iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 13: Create Deque objects of different parameterized types
==> passed

Test 14: Check that addFirst() and addLast() each throw a NullPoint
erException
         when inserting null items
==> passed
```

```
Test 15: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 16: Check iterator() when Deque is empty
==> passed


Total: 10/16 tests passed!

================================================================

Testing methods in RandomizedQueue
*-----------------------------------------------------------
Running 18 total tests.

Tests 1-4 make random calls to enqueue(), dequeue(), sample(),
isEmpty(), and size(). The probabilities of each operation are
(p1, p2, p3, p4, p5), respectively.

Test 1: Calls to enqueue() and size().
  *     5 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *    50 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *   500 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *  1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to enqueue() and dequeue().
  *     5 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *    50 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *   500 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *  1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *     5 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *    50 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *   500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *  1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
==> passed

Test 3: Calls to enqueue(), sample(), and size().
  *     5 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *    50 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *   500 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *  1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
```

```
  *     5 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *    50 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *   500 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *  1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
==> passed

Test 4: Calls to enqueue(), dequeue(), sample(), isEmpty(), and siz
e().
  *     5 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *    50 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *   500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *  1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *     5 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *    50 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *   500 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *  1000 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
==> passed

Test 5: dequeue() and sample() from an empty randomized queue
  *  dequeue()
  *  sample()
==> passed

Test 6: Create multiple randomized queue objects at the same time
==> passed

Test 7: Check that iterator() returns correct items after a sequenc
e of
        enqueue() operations
==> passed

Test 8: Check that iterator() returns correct items after sequence
of enqueue()
        and dequeue() operations
==> passed

Test 9: Create two nested iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 10: Create two parallel iterators over same randomized queue
  *  N = 10
  *  N = 1000
```

```
==> passed

Test 11: Create two iterators over different randomized queues
==> passed

Test 12: Create RandomizedQueue objects of different parameterized
types
==> passed

Test 13: Check randomness of sample() by enqueueing strings, repeat
edly calling
         sample(), and counting the frequency of each value.
  *  Enqueue strings A to C and sampling 3000 times
  *  Enqueue strings A to E and sampling 5000 times
  *  Enqueue strings A to H and sampling 8000 times
  *  Enqueue strings A to J and sampling 10000 times
==> passed

Test 14: Check randomness of dequeue() by enqueueing items, repeate
dly calling
         dequeue() until a specific enqueued string appears.
  *  Enqueue strings A to C and call dequeue() until A is dequeued;
repeat 3000 times
  *  Enqueue strings A to E and call dequeue() until E is dequeued;
repeat 5000 times
  *  Enqueue strings A to H and call dequeue() until C is dequeued;
repeat 8000 times
  *  Enqueue strings A to J and call dequeue() until G is dequeued;
repeat 10000 times
==> passed

Test 15: Check randomness of iterator() by enqueueing strings, gett
ing an iterator()
         and repeatedly calling next() until a specific enqueued st
ring appears.
  *  Enqueue strings A to C, create iterator(), and call next() unt
il C is returned;
     Repeat 3000 times
  *  Enqueue strings A to E, create iterator(), and call next() unt
il C is returned;
     Repeat 5000 times
  *  Enqueue strings A to H, create iterator(), and call next() unt
il E is returned;
     Repeat 8000 times
```

```
  *  Enqueue strings A to J, create iterator(), and call next() unt
il J is returned;
     Repeat 10000 times
==> passed

Test 16: Check that NullPointerException is thrown when inserting n
ull items
==> passed

Test 17: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 18: Check iterator() when RandomizedQueue is empty
==> passed


Total: 18/18 tests passed!

================================================================

*****************************************************************
***********
*  correctness (substituting reference RandomizedQueue.java and Deq
ue.java)
*****************************************************************
***********

Testing methods in Subset
*-----------------------------------------------------------
Tests 1-3 call the main() function directly, resetting standard inp
ut
before each call.

Running 3 total tests.

Test 1: assignment inputs

% echo "A B C D E F G H I" | java Subset 3
[student solution]
3
F
H
I
```

```
Error: Output contains invalid string: '3'
% echo "A B C D E F G H I" | java Subset 3
[student solution]
3
D
E
F

Error: Output contains invalid string: '3'
% echo "AA BB BB BB BB BB CC CC " | java Subset 8
[student solution]
8
BB
CC
BB
AA
BB
BB
CC
BB

Error: Output contains invalid string: '8'
==> FAILED

Test 2: various inputs

% echo "A B C D E F G H I" | java Subset 1
[student solution]
1
H

Error: Output contains invalid string: '1'
% echo "A B C D E F G H I" | java Subset 5
[student solution]
5
B
E
C
H
I

Error: Output contains invalid string: '5'
% echo "A B C D E F G H I" | java Subset 5
```

```
[student solution]
5
E
I
B
C
G

Error: Output contains invalid string: '5'
% echo "A B C D E F G H I" | java Subset 9
[student solution]
9
D
A
E
F
H
I
C
G
B

Error: Output contains invalid string: '9'
% echo "A B C D E F G H I" | java Subset 0
[student solution]
0

Error: Standard output is not empty

% echo "it was the best of times it was the worst of times" | java
Subset 10
[student solution]
10
the
best
it
was
the
of
it
of
times
was
```

```
Error: Output contains invalid string: '10'
% echo "It was the best of times, it was the worst of times, it was
..." | java Subset 10
[student solution]
10
or
glass.
thing,
an
meanwhile,
at
the
is
every
the

Error: Output contains invalid string: '10'
% echo "AA BB BB BB BB BB CC CC " | java Subset 7
[student solution]
7
BB
CC
BB
AA
BB
BB
BB

Error: Output contains invalid string: '7'
==> FAILED

Test 3: check that subsets are uniformly random
  * 1000 subsets of size 1 from subset10.txt
    -  Outputs a string not in input: '1'
  * 250 subsets of size 4 from subset10.txt
    -  Outputs a string not in input: '4'
  * 600 subsets of size 1 from subset6.txt
    -  Outputs a string not in input: '1'
  * 300 subsets of size 2 from subset6.txt
    -  Outputs a string not in input: '2'
  * 800 subsets of size 1 from subset8.txt
    -  Outputs a string not in input: '1'
  * 160 subsets of size 5 from subset8.txt
    -  Outputs a string not in input: '5'
```

```
==> FAILED

Total: 0/3 tests passed!

================================================================

****************************************************************
**********
*  memory
****************************************************************
**********

Computing memory of Subset
*-----------------------------------------------------------
Running 2 total tests.

Test 1: Check that only one Deque or RandomizedQueue object is crea
ted
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 50
==> passed

Test 2: Check that the maximum size of any Deque or RandomizedQueue
object
        created is <= N
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed

Test 3 (bonus): Check that maximum size of any or Deque or Randomiz
edQueue object
                created is <= k
  * filename = tale.txt, N = 138653, k = 5
    - max size of RandomizedQueue object = 138653
```

```
  * filename = tale.txt, N = 138653, k = 50
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 500
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 5000
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50000
    - max size of RandomizedQueue object = 138653
==> FAILED

Total: 2/2 tests passed!

================================================================



*****************************************************************
***********
*  memory
*****************************************************************
***********

Computing memory of Deque
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a deque
containing N items is 48N + 192.

Running 28 total tests.

Test 1a-1e: Total memory usage after inserting N items,
            where N is a power of 2.

                 N        bytes
-----------------------------------------------------------
=> passed         8          424
=> passed        64         3112
=> passed       256        12328
=> passed      1024        49192
=> passed      4096       196648
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)
```

```
Test 2a-2e: Total memory usage after inserting N+1 items,
            where N is a power of 2.

                 N        bytes
-------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory after adding N = 2^i + 1 items: 48.00 N + 40.00   (R^2 = 1.0
00)



Test 3a-3e: Total memory usage after inserting 2N+1 items
            and deleting N items, where N is a power of 2.

                 N        bytes
-------------------------------------------------------
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 4a-4e: Total memory usage after inserting N items and then
            deleting all but one item, where N is a power of 2.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------
=> passed        8           88
=> passed       64           88
=> passed      256           88
=> passed     1024           88
```

```
=> passed     4096           88
==> 5/5 tests passed

Memory after adding N = 2^i items: 88.00   (R^2 = 1.000)



Test 5a-5e: Total memory usage of iterator after inserting N items.
            (should not grow with N or be too large of a constant)

                 N        bytes
-------------------------------------------------------------
=> passed        8           32
=> passed       64           32
=> passed      256           32
=> passed     1024           32
=> passed     4096           32
==> 5/5 tests passed

Memory of iterator after adding N = 2^i items: 32.00   (R^2 = 1.000
)



Test 6a: Insert N strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFir
st()
         and addLast() are (p1, p2), respectively. The probabilitie
s of
         removeFirst() and removeLast() are (q1, q2), respectively
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0,
0.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0,
1.0)
  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5,
0.5)
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast()
```

```
,
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
  * 100 random operations (0.8, 0.0, 0.0, 0.2)
  * 100 random operations (0.0, 0.8, 0.2, 0.0)
  * 100 random operations (0.0, 0.8, 0.0, 0.2)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 100 random operations (0.2, 0.2, 0.3, 0.3)
     java.lang.NullPointerException
     Deque$Node.access$200(Deque.java:11)
     Deque.removeLast(Deque.java:79)
     MemoryOfDeque.loiter(MemoryOfDeque.java:533)
     MemoryOfDeque.test6b(MemoryOfDeque.java:605)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addLast("JRLUVFSXXZ")
         deque.removeLast()      ==> JRLUVFSXXZ
         deque.addFirst("RAAQNJANZU")
         deque.addFirst("QYSWIQJWVN")
         deque.removeLast()

==> FAILED

Test 7: Worst-case constant memory allocated or deallocated
         per deque operation?
  * 128 random operations
  * 256 random operations
  * 512 random operations
==> passed

Total: 27/28 tests passed!

================================================================



Computing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a
randomized queue containing N items is 48N + 192.

Running 23 total tests.
```

```
Test 1a-1d: Total memory usage after inserting N integers.

                 N        bytes
-------------------------------------------------------
=> passed       64          568
=> passed      256         2104
=> passed     1024         8248
=> passed     4096        32824
==> 4/4 tests passed


Memory: 8.00 N + 56.00   (R^2 = 1.000)



Test 2a-2d: Total memory usage after inserting N+1 items.

                 N        bytes
-------------------------------------------------------
=> passed       64         1080
=> passed      256         4152
=> passed     1024        16440
=> passed     4096        65592
==> 4/4 tests passed


Memory: 16.00 N + 40.00   (R^2 = 1.000)



Test 3a-3d: Total memory usage after inserting 2N+1 items, and
            then deleting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64         2104
=> passed      256         8248
=> passed     1024        32824
=> passed     4096       131128
==> 4/4 tests passed


Memory: 32.00 N + 24.00   (R^2 = 1.000)
```

```
Test 4a-4d: Total memory usage after inserting N items, and
            then deleting all but one item.

                 N        bytes
-----------------------------------------------------------
=> FAILED       64          568   (2.4x)
=> FAILED      256         2104   (8.8x)
=> FAILED     1024         8248  (34.4x)
=> FAILED     4096        32824 (136.8x)
==> 0/4 tests passed


Memory: 11965.82   (R^2 = 0.000)



Test 5a-5d: Total memory usage of iterator after inserting N items.

                 N        bytes
-----------------------------------------------------------
=> passed       64          576
=> passed      256         2112
=> passed     1024         8256
=> passed     4096        32832
==> 4/4 tests passed


Memory: 8.00 N + 64.00   (R^2 = 1.000)



Test 6a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
```

```
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 7: Insert T items into queue; then iterate over queue and chec
k
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Total: 19/23 tests passed!

================================================================



**************************************************************
***********
*  timing
**************************************************************
***********

Timing Deque
*-----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to addFirst(), addLast(), removeFirst()
,
             and removeLast().

                   N  seconds
-----------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
```

```
=> passed     1024000     0.14
=> passed     2048000     0.22
==> 12/12 tests passed


Test 2a-2g:  Create deque of N objects, then iterate over the N obj
ects
             by calling next() and hasNext().

                   N  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.02
=> passed     2048000     0.05
==> 12/12 tests passed


Test 3a-3g:  Create deque of N objects, then interleave N calls eac
h to
             removeFirst()/removeLast() and addFirst()/addLast().

                    N  seconds
---------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.01
=> passed       32767     0.01
=> passed       32768     0.01
=> passed       32769     0.01
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

```
Timing RandomizedQueue
*-----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to enqueue(), sample(), dequeue(),
             isEmpty(), and size().

                    N  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.02
=> passed      256000     0.03
=> passed      512000     0.06
=> passed     1024000     0.14
=> passed     2048000     0.36
==> 12/12 tests passed


Test 2a-2g:  Create randomized queue of N objects, then iterate
             over the N objects by calling next() and hasNext().

                    N  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.01
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.04
=> passed     1024000     0.08
=> passed     2048000     0.17
==> 12/12 tests passed
```

```
Test 3a-3g:  Create randomized queue of N objects, then interleave
             N calls each to dequeue() and enqueue().

                     N  seconds
-----------------------------------
=> passed         1025     0.00
=> passed         2049     0.00
=> passed         4097     0.00
=> passed        16385     0.00
=> passed        32767     0.00
=> passed        32768     0.00
=> passed        32769     0.00
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```

Submission	
Submission time	Wed-16-Sep 13:17:27
Raw Score	68.24 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        FAILED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  25/37 tests passed
Memory:       37/53 tests passed
Timing:       43/62 tests passed

Aggregate score: 68.24% [Correctness: 65%, Memory: 10%, Timing: 25%
```

```
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 16K
-rw-r--r-- 1 3.8K Sep 16 20:17 Deque.java
-rw-r--r-- 1 2.8K Sep 16 20:17 RandomizedQueue.java
-rw-r--r-- 1  410 Sep 16 20:17 Subset.java
-rw-r--r-- 1 2.7K Sep 16 20:17 studentSubmission.zip


================================================================
===========
*  compiling
================================================================
===========


% javac Deque.java
*-----------------------------------------------------------


================================================================


% javac RandomizedQueue.java
*-----------------------------------------------------------
Note: RandomizedQueue.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

================================================================


% javac Subset.java
*-----------------------------------------------------------


================================================================




% checkstyle *.java
*-----------------------------------------------------------
Deque.java:103:30: ',' is not followed by whitespace.
Deque.java:103:34: ',' is not followed by whitespace.
```

```
RandomizedQueue.java:14:5: Constructor definition in wrong order. T
he order should be [static variables, instance variables, construct
ors, methods].
RandomizedQueue.java:74:18: '!' is followed by whitespace.
RandomizedQueue.java:82:28: ',' is not followed by whitespace.
RandomizedQueue.java:82:32: ',' is not followed by whitespace.
RandomizedQueue.java:82:36: ',' is not followed by whitespace.
RandomizedQueue.java:82:40: ',' is not followed by whitespace.
Checkstyle ends with 8 errors.
================================================================


% findbugs *.class
*---------------------------------------------------------
M P URF_UNREAD_FIELD UrF: Unread field: Deque$Node.prevNode  At Deq
ue.java:[line 11]
Warnings generated: 1
================================================================


Testing the APIs of your programs.
*---------------------------------------------------------
Deque:

RandomizedQueue:

Subset:

================================================================


**********************************************************************
**********
*  correctness
**********************************************************************
**********

Testing methods in Deque
*---------------------------------------------------------
Running 16 total tests.

Tests 1-6 make random calls to addFirst(), addLast(), removeFirst()
,
removeLast(), isEmpty(), and size(). The probabilities of each
```

```
operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: Calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
    - failed on operation 2 of 50
    - student   removeFirst() returned null
    - reference removeFirst() returned 1
    - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.removeFirst()     ==> null

  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
    - failed on operation 7 of 500
    - student   removeFirst() returned null
    - reference removeFirst() returned 6
    - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.addFirst(2)
         deque.addFirst(3)
         deque.addFirst(4)
         deque.addFirst(5)
         deque.addFirst(6)
         deque.removeFirst()     ==> null

  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0)
    - failed on operation 6 of 1000
    - student   removeFirst() returned null
    - reference removeFirst() returned 5
    - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.addFirst(2)
         deque.addFirst(3)
         deque.addFirst(4)
```

```
      deque.addFirst(5)
      deque.removeFirst()     ==> null

*    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 3 of 5
  - student   removeFirst() returned null
  - reference removeFirst() returned 2
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.isEmpty()
      deque.addFirst(2)
      deque.removeFirst()     ==> null

*   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 1 of 50
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addFirst(0)
      deque.removeFirst()     ==> null

*  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 2 of 500
  - student   removeFirst() returned null
  - reference removeFirst() returned 1
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.addFirst(1)
      deque.removeFirst()     ==> null

* 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 1 of 1000
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addFirst(0)
      deque.removeFirst()     ==> null

==> FAILED

Test 3: Calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  - failed on operation 6 of 50
```

```
  - student   removeLast() returned null
  - reference removeLast() returned 0
  - sequence of dequeue operations was:
       deque.addFirst(0)
       deque.addFirst(1)
       deque.addFirst(2)
       deque.addFirst(3)
       deque.addFirst(4)
       deque.addFirst(5)
       deque.removeLast()      ==> null

*  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  - failed on operation 21 of 500
  - student   removeLast() returned null
  - reference removeLast() returned 0

* 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
  - failed on operation 1 of 1000
  - student   removeLast() returned null
  - reference removeLast() returned 0
  - sequence of dequeue operations was:
       deque.addFirst(0)
       deque.removeLast()      ==> null

*    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 1 of 5
  - student   removeLast() returned null
  - reference removeLast() returned 0
  - sequence of dequeue operations was:
       deque.addFirst(0)
       deque.removeLast()      ==> null

*   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 2 of 50
  - student   removeLast() returned null
  - reference removeLast() returned 1
  - sequence of dequeue operations was:
       deque.isEmpty()
       deque.addFirst(1)
       deque.removeLast()      ==> null

*  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 5 of 500
  - student   removeLast() returned null
```

```
    - reference removeLast() returned 0
    - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.addFirst(2)
         deque.addFirst(3)
         deque.addFirst(4)
         deque.removeLast()      ==> null

  * 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
    - failed on operation 2 of 1000
    - student   removeLast() returned null
    - reference removeLast() returned 1
    - sequence of dequeue operations was:
         deque.isEmpty()
         deque.addFirst(1)
         deque.removeLast()      ==> null

==> FAILED

Test 4: Calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
    - failed on operation 2 of 5
    - student   removeLast() returned null
    - reference removeLast() returned 1
    - sequence of dequeue operations was:
         deque.addLast(0)
         deque.addLast(1)
         deque.removeLast()      ==> null

  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
    - failed on operation 11 of 50
    - student   removeLast() returned null
    - reference removeLast() returned 9
    - sequence of dequeue operations was:
         deque.addLast(0)
         deque.addLast(1)
         deque.addLast(2)
         deque.addLast(3)
         deque.addLast(4)
         deque.addLast(5)
         deque.addLast(6)
         deque.addLast(7)
         deque.addLast(8)
```

```
        deque.addLast(9)
        deque.isEmpty()
        deque.removeLast()      ==> null

*  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  - failed on operation 9 of 500
  - student   removeLast() returned null
  - reference removeLast() returned 8
  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addLast(1)
        deque.addLast(2)
        deque.addLast(3)
        deque.addLast(4)
        deque.addLast(5)
        deque.addLast(6)
        deque.addLast(7)
        deque.addLast(8)
        deque.removeLast()      ==> null

* 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0)
  - failed on operation 7 of 1000
  - student   removeLast() returned null
  - reference removeLast() returned 6
  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addLast(1)
        deque.addLast(2)
        deque.addLast(3)
        deque.addLast(4)
        deque.addLast(5)
        deque.addLast(6)
        deque.removeLast()      ==> null

*    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 2 of 5
  - student   removeLast() returned null
  - reference removeLast() returned 1
  - sequence of dequeue operations was:
        deque.isEmpty()
        deque.addLast(1)
        deque.removeLast()      ==> null

*   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
```

```
  - failed on operation 1 of 50
  - student   removeLast() returned null
  - reference removeLast() returned 0
  - sequence of dequeue operations was:
      deque.addLast(0)
      deque.removeLast()      ==> null

*  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 2 of 500
  - student   removeLast() returned null
  - reference removeLast() returned 1
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.addLast(1)
      deque.removeLast()      ==> null

* 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0)
  - failed on operation 6 of 1000
  - student   removeLast() returned null
  - reference removeLast() returned 5
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.isEmpty()
      deque.isEmpty()
      deque.isEmpty()
      deque.addLast(5)
      deque.removeLast()      ==> null

==> FAILED

Test 5: Calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
    - failed on operation 14 of 50
    - student   removeFirst() returned null
    - reference removeFirst() returned 0

  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
    - failed on operation 3 of 500
    - student   removeFirst() returned null
    - reference removeFirst() returned 0
    - sequence of dequeue operations was:
        deque.addLast(0)
```

```
      deque.addLast(1)
      deque.addLast(2)
      deque.removeFirst()     ==> null

* 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0)
  - failed on operation 15 of 1000
  - student   removeFirst() returned null
  - reference removeFirst() returned 0

*    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 1 of 5
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addLast(0)
      deque.removeFirst()     ==> null

*   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 2 of 50
  - student   removeFirst() returned null
  - reference removeFirst() returned 1
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.addLast(1)
      deque.removeFirst()     ==> null

*  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 2 of 500
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addLast(0)
      deque.addLast(1)
      deque.removeFirst()     ==> null

* 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
  - failed on operation 5 of 1000
  - student   removeFirst() returned null
  - reference removeFirst() returned 4
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.isEmpty()
      deque.isEmpty()
      deque.isEmpty()
```

```
         deque.addLast(4)
         deque.removeFirst()     ==> null

==> FAILED

Test 6: Calls to addFirst(), addLast(), removeFirst(),
        removeLast(), isEmpty(), and size().
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
    - failed on operation 2 of 50
    - student   removeLast() returned null
    - reference removeLast() returned 0
    - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.removeLast()      ==> null

  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
    - failed on operation 1 of 500
    - student   removeLast() returned null
    - reference removeLast() returned 0
    - sequence of dequeue operations was:
         deque.addLast(0)
         deque.removeLast()      ==> null

* 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)
  - failed on operation 2 of 1000
  - student   removeFirst() returned null
  - reference removeFirst() returned 1
  - sequence of dequeue operations was:
         deque.addFirst(0)
         deque.addFirst(1)
         deque.removeFirst()     ==> null

*    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  - failed on operation 2 of 5
  - student   removeFirst() returned null
  - reference removeFirst() returned 1
  - sequence of dequeue operations was:
         deque.isEmpty()
         deque.addLast(1)
         deque.removeFirst()     ==> null

*   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
```

```
  - failed on operation 1 of 50
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addFirst(0)
      deque.removeFirst()     ==> null

*  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  - failed on operation 4 of 500
  - student   removeFirst() returned null
  - reference removeFirst() returned 0
  - sequence of dequeue operations was:
      deque.addLast(0)
      deque.isEmpty()
      deque.size()
      deque.size()
      deque.removeFirst()     ==> null

* 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
  - failed on operation 5 of 1000
  - student   removeLast() returned null
  - reference removeLast() returned 1
  - sequence of dequeue operations was:
      deque.isEmpty()
      deque.addFirst(1)
      deque.addFirst(2)
      deque.size()
      deque.isEmpty()
      deque.removeLast()      ==> null

==> FAILED

Test 7: Removing from an empty deque
  *  removeFirst()
     java.util.NoSuchElementException not thrown
  *  removeLast()
     java.util.NoSuchElementException not thrown
==> FAILED

Test 8: Create multiple deque objects at the same time
  Failed on 0th removeFirst() operation in deque 1: Returned null
  Failed on 0th removeFirst() operation in deque 1: Returned null
==> FAILED
```

```
Test 9: Check iterator() after calls only to addFirst()
==> passed

Test 10: Check iterator() after intermixed calls to addFirst(), add
Last(),
         removeFirst(), and removeLast()
    - failed on operation 4 of 200
    - student   removeLast() returned null
    - reference removeLast() returned 3
     -  sequence of dequeue operations was:
          deque.addFirst(1)
          deque.addFirst(2)
          deque.addLast(3)
          deque.removeLast()    ==> null

==> FAILED

Test 11: Create two nested iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 12: Create two parallel iterators to same deque
  *  N = 10
  *  N = 1000
==> passed

Test 13: Create Deque objects of different parameterized types
     java.lang.NullPointerException
     TestDeque.test13(TestDeque.java:614)
     TestDeque.main(TestDeque.java:751)

==> FAILED

Test 14: Check that addFirst() and addLast() each throw a NullPoint
erException
         when inserting null items
==> passed

Test 15: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 16: Check iterator() when Deque is empty
```

```
==> passed


Total: 7/16 tests passed!

================================================================

Testing methods in RandomizedQueue
*-----------------------------------------------------------
Running 18 total tests.

Tests 1-4 make random calls to enqueue(), dequeue(), sample(),
isEmpty(), and size(). The probabilities of each operation are
(p1, p2, p3, p4, p5), respectively.

Test 1: Calls to enqueue() and size().
  *     5 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *    50 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *   500 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
  *  1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 2: Calls to enqueue() and dequeue().
  *     5 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *    50 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *   500 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *  1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
  *     5 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *    50 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *   500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
  *  1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
==> passed

Test 3: Calls to enqueue(), sample(), and size().
  *     5 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *    50 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *   500 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *  1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1)
  *     5 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *    50 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *   500 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
  *  1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1)
==> passed
```

```
Test 4: Calls to enqueue(), dequeue(), sample(), isEmpty(), and siz
e().
  *     5 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *    50 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *   500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *  1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
  *     5 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *    50 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *   500 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
  *  1000 random calls (0.1, 0.1, 0.6, 0.1, 0.1)
==> passed

Test 5: dequeue() and sample() from an empty randomized queue
  *  dequeue()
  *  sample()
==> passed

Test 6: Create multiple randomized queue objects at the same time
==> passed

Test 7: Check that iterator() returns correct items after a sequenc
e of
        enqueue() operations
==> passed

Test 8: Check that iterator() returns correct items after sequence
of enqueue()
        and dequeue() operations
==> passed

Test 9: Create two nested iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 10: Create two parallel iterators over same randomized queue
  *  N = 10
  *  N = 1000
==> passed

Test 11: Create two iterators over different randomized queues
==> passed

Test 12: Create RandomizedQueue objects of different parameterized
```

```
types
==> passed

Test 13: Check randomness of sample() by enqueueing strings, repeat
edly calling
         sample(), and counting the frequency of each value.
  *  Enqueue strings A to C and sampling 3000 times
  *  Enqueue strings A to E and sampling 5000 times
  *  Enqueue strings A to H and sampling 8000 times
  *  Enqueue strings A to J and sampling 10000 times
==> passed

Test 14: Check randomness of dequeue() by enqueueing items, repeate
dly calling
         dequeue() until a specific enqueued string appears.
  *  Enqueue strings A to C and call dequeue() until C is dequeued;
repeat 3000 times
  *  Enqueue strings A to E and call dequeue() until C is dequeued;
repeat 5000 times
  *  Enqueue strings A to H and call dequeue() until H is dequeued;
repeat 8000 times
  *  Enqueue strings A to J and call dequeue() until D is dequeued;
repeat 10000 times
==> passed

Test 15: Check randomness of iterator() by enqueueing strings, gett
ing an iterator()
         and repeatedly calling next() until a specific enqueued st
ring appears.
  *  Enqueue strings A to C, create iterator(), and call next() unt
il A is returned;
     Repeat 3000 times
  *  Enqueue strings A to E, create iterator(), and call next() unt
il D is returned;
     Repeat 5000 times
  *  Enqueue strings A to H, create iterator(), and call next() unt
il G is returned;
     Repeat 8000 times
  *  Enqueue strings A to J, create iterator(), and call next() unt
il D is returned;
     Repeat 10000 times
==> passed

Test 16: Check that NullPointerException is thrown when inserting n
```

```
ull items
==> passed

Test 17: Check that remove() and next() throw the specified excepti
ons in iterator()
==> passed

Test 18: Check iterator() when RandomizedQueue is empty
==> passed


Total: 18/18 tests passed!

================================================================

*****************************************************************
**********
*  correctness (substituting reference RandomizedQueue.java and Deq
ue.java)
*****************************************************************
**********

Testing methods in Subset
*-----------------------------------------------------------
Tests 1-3 call the main() function directly, resetting standard inp
ut
before each call.

Running 3 total tests.

Test 1: assignment inputs

% echo "A B C D E F G H I" | java Subset 3
[student solution]
3
C
B
D

Error: Output contains invalid string: '3'
% echo "A B C D E F G H I" | java Subset 3
[student solution]
3
G
```

```
D
B

Error: Output contains invalid string: '3'
% echo "AA BB BB BB BB BB CC CC " | java Subset 8
[student solution]
8
BB
CC
BB
CC
BB
BB
BB
AA

Error: Output contains invalid string: '8'
==> FAILED

Test 2: various inputs

% echo "A B C D E F G H I" | java Subset 1
[student solution]
1
C

Error: Output contains invalid string: '1'
% echo "A B C D E F G H I" | java Subset 5
[student solution]
5
B
F
C
A
I

Error: Output contains invalid string: '5'
% echo "A B C D E F G H I" | java Subset 5
[student solution]
5
I
G
F
C
```

```
E

Error: Output contains invalid string: '5'
% echo "A B C D E F G H I" | java Subset 9
[student solution]
9
C
I
D
A
F
B
H
G
E

Error: Output contains invalid string: '9'
% echo "A B C D E F G H I" | java Subset 0
[student solution]
0

Error: Standard output is not empty

% echo "it was the best of times it was the worst of times" | java
Subset 10
[student solution]
10
the
times
it
was
times
worst
it
of
was
best

Error: Output contains invalid string: '10'
% echo "It was the best of times, it was the worst of times, it was
..." | java Subset 10
[student solution]
10
was
```

```
to
The
the
said
dream,
the
jury
reclining
to

Error: Output contains invalid string: '10'
% echo "AA BB BB BB BB BB CC CC " | java Subset 7
[student solution]
7
BB
BB
BB
BB
AA
CC
BB

Error: Output contains invalid string: '7'
==> FAILED

Test 3: check that subsets are uniformly random
  * 1000 subsets of size 1 from subset10.txt
    -  Outputs a string not in input: '1'
  * 250 subsets of size 4 from subset10.txt
    -  Outputs a string not in input: '4'
  * 600 subsets of size 1 from subset6.txt
    -  Outputs a string not in input: '1'
  * 300 subsets of size 2 from subset6.txt
    -  Outputs a string not in input: '2'
  * 800 subsets of size 1 from subset8.txt
    -  Outputs a string not in input: '1'
  * 160 subsets of size 5 from subset8.txt
    -  Outputs a string not in input: '5'
==> FAILED

Total: 0/3 tests passed!

================================================================
```

```
*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Subset
*-----------------------------------------------------------
Running 2 total tests.

Test 1: Check that only one Deque or RandomizedQueue object is crea
ted
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 50
==> passed

Test 2: Check that the maximum size of any Deque or RandomizedQueue
object
        created is <= N
  * filename = subset9.txt, N = 9, k = 1
  * filename = subset9.txt, N = 9, k = 2
  * filename = subset9.txt, N = 9, k = 4
  * filename = tinyTale.txt, N = 12, k = 10
  * filename = tale.txt, N = 138653, k = 5
  * filename = tale.txt, N = 138653, k = 50
  * filename = tale.txt, N = 138653, k = 500
  * filename = tale.txt, N = 138653, k = 5000
  * filename = tale.txt, N = 138653, k = 50000
==> passed

Test 3 (bonus): Check that maximum size of any or Deque or Randomiz
edQueue object
                created is <= k
  * filename = tale.txt, N = 138653, k = 5
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 50
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 500
    - max size of RandomizedQueue object = 138653
  * filename = tale.txt, N = 138653, k = 5000
    - max size of RandomizedQueue object = 138653
```

```
  * filename = tale.txt, N = 138653, k = 50000
    - max size of RandomizedQueue object = 138653
==> FAILED

Total: 2/2 tests passed!

================================================================



*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Deque
*-----------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a deque
containing N items is 48N + 192.

Running 28 total tests.

Test 1a-1e: Total memory usage after inserting N items,
            where N is a power of 2.

                N        bytes
-------------------------------------------------------
=> passed        8          424
=> passed       64         3112
=> passed      256        12328
=> passed     1024        49192
=> passed     4096       196648
==> 5/5 tests passed

Memory: 48.00 N + 40.00   (R^2 = 1.000)



Test 2a-2e: Total memory usage after inserting N+1 items,
            where N is a power of 2.

                N        bytes
-------------------------------------------------------
```

```
=> passed        8          472
=> passed       64         3160
=> passed      256        12376
=> passed     1024        49240
=> passed     4096       196696
==> 5/5 tests passed

Memory after adding N = 2^i + 1 items: 48.00 N + 40.00   (R^2 = 1.0
00)



Test 3a-3e: Total memory usage after inserting 2N+1 items
            and deleting N items, where N is a power of 2.

                N        bytes
-------------------------------------------------------------
=> FAILED       8          856   (1.4x)
=> FAILED      64         6232   (1.9x)
=> FAILED     256        24664   (2.0x)
=> FAILED    1024        98392   (2.0x)
=> FAILED    4096       393304   (2.0x)
==> 0/5 tests passed

Memory: 96.00 N + -8.00   (R^2 = 1.000)



Test 4a-4e: Total memory usage after inserting N items and then
            deleting all but one item, where N is a power of 2.
            (should not grow with N or be too large of a constant)

                N        bytes
-------------------------------------------------------------
=> FAILED       8          424   (1.8x)
=> FAILED      64         3112  (13.0x)
=> FAILED     256        12328  (51.4x)
=> FAILED    1024        49192 (205.0x)
=> FAILED    4096       196648 (819.4x)
==> 0/5 tests passed

Memory after adding N = 2^i items: 71498.91   (R^2 = 0.000)
```

```
Test 5a-5e: Total memory usage of iterator after inserting N items.
            (should not grow with N or be too large of a constant)

                  N        bytes
-------------------------------------------------------------
=> passed         8           32
=> passed        64           32
=> passed       256           32
=> passed      1024           32
=> passed      4096           32
==> 5/5 tests passed

Memory of iterator after adding N = 2^i items: 32.00   (R^2 = 1.000
)



Test 6a: Insert N strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFir
st()
         and addLast() are (p1, p2), respectively. The probabilitie
s of
          removeFirst() and removeLast() are (q1, q2), respectively
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0,
0.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)

MemoryOfDeque.loiterInsertionsBeforeDeletions(MemoryOfDeque.java:43
6)
     MemoryOfDeque.test6a(MemoryOfDeque.java:474)
     MemoryOfDeque.main(MemoryOfDeque.java:715)

  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0,
```

```
1.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)

MemoryOfDeque.loiterInsertionsBeforeDeletions(MemoryOfDeque.java:43
6)
     MemoryOfDeque.test6a(MemoryOfDeque.java:475)
     MemoryOfDeque.main(MemoryOfDeque.java:715)

  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0,
0.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)

MemoryOfDeque.loiterInsertionsBeforeDeletions(MemoryOfDeque.java:43
6)
     MemoryOfDeque.test6a(MemoryOfDeque.java:476)
     MemoryOfDeque.main(MemoryOfDeque.java:715)

  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0,
1.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)
```

```
com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)

MemoryOfDeque.loiterInsertionsBeforeDeletions(MemoryOfDeque.java:43
6)
     MemoryOfDeque.test6a(MemoryOfDeque.java:477)
     MemoryOfDeque.main(MemoryOfDeque.java:715)

  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5,
0.5)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)

MemoryOfDeque.loiterInsertionsBeforeDeletions(MemoryOfDeque.java:43
6)
     MemoryOfDeque.test6a(MemoryOfDeque.java:478)
     MemoryOfDeque.main(MemoryOfDeque.java:715)

==> FAILED

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast()
,
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)
```

```
com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:600)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addFirst("BRMMVZQIPY")
         deque.removeFirst()     ==> null

  * 100 random operations (0.8, 0.0, 0.0, 0.2)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:601)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addFirst("CGCGKGBXKP")
         deque.addFirst("WYHXOERJKU")
         deque.addFirst("XPEZUBUQVO")
         deque.addFirst("PHNDXUGJSK")
         deque.addFirst("VXSORKWLVE")
         deque.addFirst("UISGMZIIJR")
         deque.addFirst("QYMHZOYSHQ")
         deque.addFirst("GRFHZEDEDA")
         deque.addFirst("TRTQEBCMXQ")
         deque.removeLast()      ==> null

  * 100 random operations (0.0, 0.8, 0.2, 0.0)
     java.lang.NullPointerException: null passed as 'objectToSize'
```

```
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:602)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addLast("PKTEIBTEWM")
         deque.addLast("XHTYDNRFYD")
         deque.removeFirst()     ==> null

  * 100 random operations (0.0, 0.8, 0.0, 0.2)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:603)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addLast("NVXABZYTGC")
         deque.removeLast()      ==> null

  * 100 random operations (0.4, 0.4, 0.1, 0.1)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
```

```
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:604)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addLast("TWNNEZZDQY")
         deque.addLast("DINMGPHVDK")
         deque.addFirst("JNJLBWCZFF")
         deque.addFirst("FCCNKLDRZE")
         deque.removeLast()      ==> null

  * 100 random operations (0.2, 0.2, 0.3, 0.3)
     java.lang.NullPointerException: null passed as 'objectToSize'
in getObjectSize

sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImp
l.java:188)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.jav
a:178)

com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.j
ava:165)
     MemoryOfDeque.loiter(MemoryOfDeque.java:546)
     MemoryOfDeque.test6b(MemoryOfDeque.java:605)
     MemoryOfDeque.main(MemoryOfDeque.java:716)

    - sequence of dequeue operations was:
         deque.addLast("CFHESJUSHU")
         deque.removeFirst()     ==> null

==> FAILED

Test 7: Worst-case constant memory allocated or deallocated
        per deque operation?
  * 128 random operations
  * 256 random operations
```

```
  * 512 random operations
==> passed

Total: 16/28 tests passed!

================================================================



Computing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a
randomized queue containing N items is 48N + 192.

Running 23 total tests.

Test 1a-1d: Total memory usage after inserting N integers.

                 N        bytes
-------------------------------------------------------------
=> passed       64          568
=> passed      256         2104
=> passed     1024         8248
=> passed     4096        32824
==> 4/4 tests passed


Memory: 8.00 N + 56.00   (R^2 = 1.000)



Test 2a-2d: Total memory usage after inserting N+1 items.

                 N        bytes
-------------------------------------------------------------
=> passed       64         1080
=> passed      256         4152
=> passed     1024        16440
=> passed     4096        65592
==> 4/4 tests passed


Memory: 16.00 N + 40.00   (R^2 = 1.000)
```

```
Test 3a-3d: Total memory usage after inserting 2N+1 items, and
            then deleting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64         2104
=> passed      256         8248
=> passed     1024        32824
=> passed     4096       131128
==> 4/4 tests passed


Memory: 32.00 N + 24.00   (R^2 = 1.000)



Test 4a-4d: Total memory usage after inserting N items, and
            then deleting all but one item.

                 N        bytes
-------------------------------------------------------
=> FAILED       64          568   (2.4x)
=> FAILED      256         2104   (8.8x)
=> FAILED     1024         8248  (34.4x)
=> FAILED     4096        32824 (136.8x)
==> 0/4 tests passed


Memory: 11965.82   (R^2 = 0.000)



Test 5a-5d: Total memory usage of iterator after inserting N items.

                 N        bytes
-------------------------------------------------------
=> passed       64          576
=> passed      256         2112
=> passed     1024         8256
=> passed     4096        32832
==> 4/4 tests passed
```

```
Memory: 8.00 N + 64.00   (R^2 = 1.000)



Test 6a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 7: Insert T items into queue; then iterate over queue and chec
k
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Total: 19/23 tests passed!

================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********

Timing Deque
*-----------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to addFirst(), addLast(), removeFirst()
```

```
,
             and removeLast().

                    N  seconds
------------------------------
     java.lang.NullPointerException
     TimeDeque.run(TimeDeque.java:21)
     TimeDeque.test1(TimeDeque.java:105)
     TimeDeque.main(TimeDeque.java:201)

     java.lang.NullPointerException
     TimeDeque.run(TimeDeque.java:20)
     TimeDeque.test1(TimeDeque.java:106)
     TimeDeque.main(TimeDeque.java:201)

=> FAILED        1024 Infinity
   [ Most likely one of your operations is not constant time. ]
==> 0/12 tests passed


Test 2a-2g:  Create deque of N objects, then iterate over the N obj
ects
             by calling next() and hasNext().

                   N  seconds
------------------------------
=> passed         1024     0.00
=> passed         2048     0.00
=> passed         4096     0.00
=> passed         8192     0.00
=> passed        16384     0.00
=> passed        32768     0.00
=> passed        65536     0.01
=> passed       128000     0.01
=> passed       256000     0.01
=> passed       512000     0.03
=> passed      1024000     0.03
=> passed      2048000     0.05
==> 12/12 tests passed


Test 3a-3g:  Create deque of N objects, then interleave N calls eac
h to
             removeFirst()/removeLast() and addFirst()/addLast().
```

```
                    N  seconds
-----------------------------------
  Failed on 1th operation of 1025: returned null on call to removeF
irst() or removeLast()
  Failed on 1th operation of 1025: returned null on call to removeF
irst() or removeLast()
=> FAILED       1025     Test did not complete due to an exceptio
n.
==> 0/7 tests passed

Total: 12/31 tests passed!
================================================================



Timing RandomizedQueue
*----------------------------------------------------------
Running 31 total tests.

Test 1a-1g:  N random calls to enqueue(), sample(), dequeue(),
             isEmpty(), and size().

                    N  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.01
=> passed       65536     0.01
=> passed      128000     0.03
=> passed      256000     0.03
=> passed      512000     0.07
=> passed     1024000     0.17
=> passed     2048000     0.36
==> 12/12 tests passed


Test 2a-2g:  Create randomized queue of N objects, then iterate
             over the N objects by calling next() and hasNext().

                    N  seconds
```

```
---------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.01
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.04
=> passed     1024000     0.07
=> passed     2048000     0.17
==> 12/12 tests passed


Test 3a-3g:  Create randomized queue of N objects, then interleave
             N calls each to dequeue() and enqueue().

                   N  seconds
---------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed       16385     0.00
=> passed       32767     0.01
=> passed       32768     0.00
=> passed       32769     0.00
==> 7/7 tests passed

Total: 31/31 tests passed!
================================================================
```