Programming Assignment: Deques and Randomized Queues

✔ Passed · 80/100 points

Deadline The assignment was due on April 1, 11:59 PM PDT
You can still pass this assignment before the course ends.

Instructions

My submission

Discussions

✖ Cancel

Upload Files and Submit

To upload a file, click the part below. Then, submit the files. You can submit as many times as you like. You do not need to upload all parts in order to submit.

Deques and Randomized Queues

queues.zip
80/100

Submit

Your Submissions

Date	Score	Passed?
✔ 2 April 2018 at 10:16 PM	80/100	Yes
Deques and Randomized Queues	80/100	Hide grader output

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

ASSESSMENT SUMMARY

Compilation:  PASSED (0 errors, 2 warnings)
API:          PASSED

Findbugs:     FAILED (1 warning)
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 223 warnings)

Correctness:  30/43 tests passed
Memory:       102/105 tests passed
Timing:       123/136 tests passed

Aggregate score: 79.66%
[Compilation: 5%, API: 5%, Findbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:
----------------------------------
1.2K Apr  3 05:16 Deque.java
 343 Apr  3 05:16 Permutation.java
3.3K Apr  3 05:16 RandomizedQueue.java

================================================================
*  COMPILING
================================================================

% javac Deque.java
*-----------------------------------------------------------
% javac RandomizedQueue.java
*-----------------------------------------------------------
RandomizedQueue.java:13: warning: [unchecked] unchecked cast
                randomQueue = (Item[]) new Object[1];
                              ^
```

✔ 2 April 2018 at 10:11 PM	79/100	No
✔ 2 April 2018 at 10:09 PM	79/100	No
✔ 1 April 2018 at 11:35 PM	79/100	No
✔ 1 April 2018 at 11:17 PM	78/100	No
✔ 1 April 2018 at 11:11 PM	10/100	No
✔ 1 April 2018 at 11:08 PM	5/100	No
✔ 31 March 2018 at 7:10 PM	0/100	No
✔ 31 March 2018 at 7:06 PM	0/100	No
✔ 30 March 2018 at 11:47 PM	0/100	No

30 March 2018 at 11:32 PM 0/100 No

```
  found:    Object[]
  where Item is a type-variable:
    Item extends Object declared in class RandomizedQueue
RandomizedQueue.java:38: warning: [unchecked] unchecked cast
                Item[] copy=(Item[]) new Object[capacity];
                                     ^
  required: Item[]
  found:    Object[]
  where Item is a type-variable:
    Item extends Object declared in class RandomizedQueue
2 warnings

% javac Permutation.java
*-----------------------------------------------------------


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


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

RandomizedQueue:

Permutation:

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


********************************************************************************
*  CHECKING STYLE AND COMMON BUG PATTERNS
********************************************************************************


% findbugs *.class
*-----------------------------------------------------------
L D NP_LOAD_OF_KNOWN_NULL_VALUE NP: The variable 'sentinel' is known to be null at this point due to an earlier check against null.
Warnings generated: 1


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


% pmd .
*-----------------------------------------------------------
Deque.java:9: Can you replace the instance (or static) variable 'first' with a local variable? [SingularField]
Deque.java:9: Can you replace the instance (or static) variable 'last' with a local variable? [SingularField]
Deque.java:9: The private instance (or static) variable 'header' can be made 'final'; it is initialized only in the declaration or
Deque.java:9: The private instance (or static) variable 'trailer' can be made 'final'; it is initialized only in the declaration o
Deque.java:103: Unnecessary use of fully qualified name 'java.util.NoSuchElementException' due to existing import 'java.util.NoSuch
Deque.java:107: There appears to be a spurious semicolon. [EmptyStatementNotInLoop]
Deque.java:121: Unnecessary use of fully qualified name 'java.util.NoSuchElementException' due to existing import 'java.util.NoSuch
RandomizedQueue.java:46: Unnecessary use of fully qualified name 'java.util.NoSuchElementException' due to existing import 'java.ut
RandomizedQueue.java:61: Unnecessary use of fully qualified name 'java.util.NoSuchElementException' due to existing import 'java.ut
PMD ends with 9 warnings.


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


% checkstyle *.java
*-----------------------------------------------------------
[WARN] Deque.java:5:1: File contains tab characters (this is the first occurrence). Configure your editor to replace tabs with spac
[WARN] Deque.java:8:21: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:8:22: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:9:25: ',' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:9:33: ',' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:9:39: ',' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:19:28: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:20:17: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:20:18: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:21:17: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:21:18: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:22:17: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:22:18: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:52:19: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:53:22: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:53:23: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:54:15: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:54:16: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:55:16: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:55:17: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:61:29: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:62:21: '==' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:62:23: '==' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:65:22: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:70:36: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:71:11: '//' or '/*' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:72:11: 'if' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:72:16: '==' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:72:18: '==' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:72:23: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:75:22: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:75:23: '=' is not followed by whitespace. [WhitespaceAround]
[WARN] Deque.java:76:11: '//' or '/*' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:77:14: The local (or parameter) variable 'first' has the same name as an instance variable. Use a different name.
```

```
[WARN] Deque.java:77:19: '=' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:80:11: '//' or '/*' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:87:11: 'if' is not followed by whitespace. [WhitespaceAfter]
[WARN] Deque.java:87:16: '==' is not preceded with whitespace. [WhitespaceAround]
[WARN] Deque.java:87:18: '==' is not followed by whitespace. [WhitespaceAround]
...
Checkstyle ends with 0 errors and 223 warnings.

% custom checkstyle checks for Deque.java
*-----------------------------------------------------------

% custom checkstyle checks for RandomizedQueue.java
*-----------------------------------------------------------

% custom checkstyle checks for Permutation.java
*-----------------------------------------------------------


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


********************************************************************************
*  TESTING CORRECTNESS
********************************************************************************

Testing correctness of 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: check random 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: check random 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: check random 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)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:470)
    TestDeque.main(TestDeque.java:831)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.addFirst(0)
         deque.removeLast()      ==> 0
         deque.addFirst(2)
         deque.addFirst(3)
         deque.addFirst(4)
         deque.addFirst(5)
         deque.addFirst(6)
         deque.addFirst(7)
         deque.isEmpty()         ==> false
         deque.isEmpty()         ==> false
         deque.removeLast()

  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:471)
    TestDeque.main(TestDeque.java:831)


  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:472)
    TestDeque.main(TestDeque.java:831)


  *    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.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
```

```
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:474)
    TestDeque.main(TestDeque.java:831)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.addFirst(0)
         deque.removeLast()      ==> 0
         deque.isEmpty()         ==> true
         deque.addFirst(3)
         deque.removeLast()

  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:475)
    TestDeque.main(TestDeque.java:831)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.isEmpty()         ==> true
         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.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test3(TestDeque.java:476)
    TestDeque.main(TestDeque.java:831)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.addFirst(0)
         deque.removeLast()      ==> 0
         deque.addFirst(2)
         deque.removeLast()

==> FAILED

Test 4: check random 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: check random 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.util.NoSuchElementException

    Deque.removeFirst(Deque.java:121)
    TestDeque.random(TestDeque.java:87)
    TestDeque.test5(TestDeque.java:506)
    TestDeque.main(TestDeque.java:833)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.isEmpty()         ==> true
         deque.isEmpty()         ==> true
         deque.isEmpty()         ==> true
         deque.addLast(3)
         deque.removeFirst()     ==> 3
         deque.isEmpty()         ==> true
         deque.isEmpty()         ==> true
         deque.addLast(7)
         deque.removeFirst()

  *  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
    java.util.NoSuchElementException

    Deque.removeFirst(Deque.java:121)
    TestDeque.random(TestDeque.java:87)
    TestDeque.test5(TestDeque.java:507)
    TestDeque.main(TestDeque.java:833)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.isEmpty()         ==> true
         deque.addLast(1)
         deque.removeFirst()     ==> 1
         deque.addLast(3)
         deque.isEmpty()         ==> false
         deque.isEmpty()         ==> false
```

```
         deque.removeFirst()

  * 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0)
    java.util.NoSuchElementException

    Deque.removeFirst(Deque.java:121)
    TestDeque.random(TestDeque.java:87)
    TestDeque.test5(TestDeque.java:508)
    TestDeque.main(TestDeque.java:833)

    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.isEmpty()         ==> true
         deque.addLast(1)
         deque.removeFirst()     ==> 1
         deque.addLast(3)
         deque.removeFirst()

==> FAILED

Test 6: check random 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)
    - failed on operation 9 of 50
    - student   removeFirst() returned null
    - reference removeFirst() returned 5
    - sequence of operations was:
         Deque<Integer> deque = new Deque<Integer>()
         deque.addFirst(0)
         deque.removeLast()      ==> 0
         deque.isEmpty()         ==> true
         deque.isEmpty()         ==> true
         deque.addFirst(4)
         deque.addFirst(5)
         deque.isEmpty()         ==> false
         deque.addLast(7)
         deque.removeLast()      ==> 7
         deque.removeFirst()     ==> null

  *  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test6(TestDeque.java:525)
    TestDeque.main(TestDeque.java:834)


  * 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1)
    java.util.NoSuchElementException

    Deque.removeLast(Deque.java:103)
    TestDeque.random(TestDeque.java:106)
    TestDeque.test6(TestDeque.java:526)
    TestDeque.main(TestDeque.java:834)


==> FAILED

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

Test 8: check whether two Deque objects can be created at the same time
==> passed

Test 9: check iterator() after n calls to addFirst()
  * n = 10
  * n = 50
==> passed

Test 10: check iterator() after each of m intermixed calls to
         addFirst(), addLast(), removeFirst(), and removeLast()
  * m = 20
  * m = 50
    - failed on operation 12 of 50
    - student   removeLast() returned null
    - reference removeLast() returned 11

  * m = 100
  * m = 1000
==> FAILED

Test 11: create two nested iterators to same deque
  * n = 10
  * n = 50
==> passed

Test 12: create two parallel iterators to same deque
==> passed

Test 13: create Deque objects of different parameterized types
```

```
==> passed

Test 14: call addFirst() and addLast() with null argument
==> passed

Test 15: check that remove() and next() throw the specified exceptions in iterator()
==> passed

Test 16: call iterator() when the deque is empty
==> passed


Total: 12/16 tests passed!


================================================================
Testing correctness of 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: check random 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: check random 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)
    - failed on operation 25 of 500
    - dequeue() returned null

  * 1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1)
    - failed on operation 21 of 1000
    - dequeue() returned null

  *    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)
    - failed on operation 17 of 50
    - dequeue() returned null

  *  500 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
    - failed on operation 40 of 500
    - dequeue() returned null

  * 1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1)
    - failed on operation 5 of 1000
    - dequeue() returned null

    - sequence of randomized queue operations was:
        RandomizedQueue<Integer> rq = new RandomizedQueue<Integer>()
        rq.enqueue(95)
        rq.dequeue()     ==> 95
        rq.enqueue(707)
        rq.enqueue(426)
        rq.dequeue()     ==> 707
        rq.dequeue()     ==> null
==> FAILED

Test 3: check random 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: check random calls to enqueue(), dequeue(), sample(), isEmpty(), and size()
  *    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)
    - failed on operation 17 of 50
    - sample() returned null

  *  500 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
    - failed on operation 35 of 500
    - dequeue() returned null

  * 1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1)
    - failed on operation 21 of 1000
    - dequeue() returned null

  *    5 random calls (0.1, 0.6, 0.1, 0.1, 0.1)
  *   50 random calls (0.1, 0.6, 0.1, 0.1, 0.1)
  *  500 random calls (0.1, 0.6, 0.1, 0.1, 0.1)
    - failed on operation 152 of 500
    - dequeue() returned null

  * 1000 random calls (0.1, 0.6, 0.1, 0.1, 0.1)
    - failed on operation 46 of 1000
```

```
    - dequeue() returned null

==> FAILED

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

Test 6: create multiple randomized queue objects at the same time
  * n = 10
    - failed on dequeue() operation 2 of 10
    - dequeue() returned null

  * n = 100
    - failed on dequeue() operation 7 of 100
    - dequeue() returned null

==> FAILED

Test 7: check that iterator() returns correct items after a sequence
        of n enqueue() operations
  * n = 10
  * n = 50
==> passed

Test 8: check that iterator() returns correct items after sequence
        of m enqueue() and dequeue() operations
  * m = 10
    - number of non-null entries in student   solution: 5
    - number of non-null entries in reference solution: 8
    - number of null     entries in student   solution: 3
    - number of null     entries in reference solution: 0
    - 3 missing entries in student solution, including: '6'

  * m = 1000
    - number of non-null entries in student   solution: 153
    - number of non-null entries in reference solution: 530
    - number of null     entries in student   solution: 349
    - number of null     entries in reference solution: 0
    - 377 missing entries in student solution, including: '995'

==> FAILED

Test 9: create two nested iterators over the same randomized queue
  * n = 10
    java.lang.NullPointerException

    TestRandomizedQueue.checkTwoNestedIterators(TestRandomizedQueue.java:325)
    TestRandomizedQueue.test9(TestRandomizedQueue.java:910)
    TestRandomizedQueue.main(TestRandomizedQueue.java:1191)

  * n = 50
    java.lang.NullPointerException

    TestRandomizedQueue.checkTwoNestedIterators(TestRandomizedQueue.java:325)
    TestRandomizedQueue.test9(TestRandomizedQueue.java:911)
    TestRandomizedQueue.main(TestRandomizedQueue.java:1191)

==> FAILED

Test 10: create two parallel iterators over the same randomized queue
  * n = 10
    - student   iterator 1 hasNext() = true
    - student   iterator 2 hasNext() = true
    - reference iterator   hasNext() = false

  * n = 50
    - two iterators return the same sequence of values
    - they should return the same set of values but in a
      different order

==> FAILED

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 n items, repeatedly calling
         sample(), and counting the frequency of each item
  * n = 3, trials = 12000
  * n = 5, trials = 12000
  * n = 8, trials = 12000
  * n = 10, trials = 12000
==> passed

Test 14: check randomness of dequeue() by enqueueing n items, dequeueing n items,
         and seeing whether each of the n! permutations is equally likely
  * n = 2, trials = 12000
    - dequeue() returned null
    - failed on operation 1 of 12000

  * n = 3, trials = 12000
    - dequeue() returned null
    - failed on operation 1 of 12000
```

```
  * n = 4, trials = 12000
    - dequeue() returned null
    - failed on operation 1 of 12000

  * n = 5, trials = 12000
    - dequeue() returned null
    - failed on operation 1 of 12000

==> FAILED

Test 15: check randomness of iterator() by enqueueing n items, iterating over those
         n items, and seeing whether each of the n! permutations is equally likely
  * n = 2, trials = 12000
  * n = 3, trials = 12000
    - next() returned null
    - failed on operation 1 of 12000

  * n = 4, trials = 12000
  * n = 5, trials = 12000
    - next() returned null
    - failed on operation 5 of 12000

==> FAILED

Test 16: call enqueue() with a null argument
==> passed

Test 17: check that remove() and next() throw the specified exceptions in iterator()
==> passed

Test 18: call iterator() when randomized queue is empty
    - hasNext() returns true

==> FAILED


Total: 9/18 tests passed!


================================================================
********************************************************************************
*  TESTING CORRECTNESS (substituting reference RandomizedQueue and Deque)
********************************************************************************

Testing correctness of Permutation
*-----------------------------------------------------------
Tests 1-5 call the main() function directly, resetting standard input
before each call.

Running 9 total tests.

Test 1a: check formatting for sample inputs from assignment specification
  % java Permutation 3 < distinct.txt
  G
  B
  I

  % java Permutation 3 < distinct.txt
  H
  F
  B

  % java Permutation 8 < duplicates.txt
  BB
  AA
  BB
  BB
  BB
  CC
  BB
  CC

==> passed

Test 1b: check formatting for other inputs
  % java Permutation 8 < mediumTale.txt
  of
  age
  foolishness
  times
  age
  of
  the
  of

  % java Permutation 0 < distinct.txt
  [no output]

==> passed

Test 2: check that main() reads all data from standard input
  * filename = distinct.txt, k = 3
  * filename = distinct.txt, k = 3
  * filename = duplicates.txt, k = 8
  * filename = mediumTale.txt, k = 8
==> passed

Test 3a: check that main() prints each item from the sequence at most once
```

```
        (for inputs with no duplicate strings)
  * filename = distinct.txt, k = 3
  * filename = distinct.txt, k = 1
  * filename = distinct.txt, k = 9
  * filename = permutation6.txt, k = 6
  * filename = permutation10.txt, k = 10
==> passed

Test 3b: check that main() prints each item from the sequence at most once
        (for inputs with duplicate strings)
  * filename = duplicates.txt, k = 8
  * filename = duplicates.txt, k = 3
  * filename = permutation8.txt, k = 6
  * filename = permutation8.txt, k = 2
  * filename = tinyTale.txt, k = 10
==> passed

Test 3c: check that main() prints each item from the sequence at most once
        (for inputs with newlines)
  * filename = mediumTale.txt, k = 10
  * filename = mediumTale.txt, k = 20
  * filename = tale.txt, k = 10
  * filename = tale.txt, k = 50
==> passed

Test 4: check main() when k = 0
  * filename = distinct.txt, k = 0
  * filename = distinct.txt, k = 0
==> passed

Test 5a: check that permutations are uniformly random
        (for inputs with no duplicate strings)
  * filename = permutation4.txt, k = 1
  * filename = permutation4.txt, k = 2
  * filename = permutation4.txt, k = 3
  * filename = permutation4.txt, k = 4
  * filename = permutation6.txt, k = 2
==> passed

Test 5b: check that permutations are uniformly random
        (for inputs with duplicate strings)
  * filename = permutation5.txt, k = 1
  * filename = permutation5.txt, k = 2
  * filename = permutation5.txt, k = 3
  * filename = duplicates.txt, k = 3
  * filename = permutation8.txt, k = 2
==> passed

Total: 9/9 tests passed!


================================================================
********************************************************************************
*  TIMING (substituting reference RandomizedQueue and Deque)
********************************************************************************

Timing Permutation
*-----------------------------------------------------------
Running 23 total tests.

Test 1: count calls to methods in StdIn
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 2: count calls to methods in Deque and RandomizedQueue
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 3: count calls to methods in StdRandom
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 4: Time main() with k = 5, for inputs containing n random strings

                    n  seconds
-------------------------------
=> passed        1000     0.00
=> passed        2000     0.00
=> passed        4000     0.00
=> passed        8000     0.01
=> passed       16000     0.01
=> passed       32000     0.01
=> passed       64000     0.02
```

```
=> passed      128000     0.05
=> passed      256000     0.22
=> passed      512000     0.27
==> 10/10 tests passed


Test 5: Time main() with k = 1000, for inputs containing n random strings

                    n  seconds
------------------------------
=> passed        1000     0.00
=> passed        2000     0.00
=> passed        4000     0.00
=> passed        8000     0.00
=> passed       16000     0.01
=> passed       32000     0.01
=> passed       64000     0.02
=> passed      128000     0.05
=> passed      256000     0.09
=> passed      512000     0.18
==> 10/10 tests passed


Total: 23/23 tests passed!


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



********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Permutation
*-----------------------------------------------------------
Running 2 total tests.

Test 1: check that only one Deque or RandomizedQueue object is created
  * filename = distinct.txt, n = 9, k = 1
  * filename = distinct.txt, n = 9, k = 2
  * filename = distinct.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 between k and n
  * filename = distinct.txt, n = 9, k = 1
  * filename = distinct.txt, n = 9, k = 2
  * filename = distinct.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 RandomizedQueue object
                created is equal to 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
********************************************************************************

Analyzing 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          536
=> passed       64         3224
=> passed      256        12440
=> passed     1024        49304
=> passed     4096       196760
==> 5/5 tests passed

Memory: 48.00 n + 152.00   (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          584
=> passed       64         3272
=> passed      256        12488
=> passed     1024        49352
=> passed     4096       196808
==> 5/5 tests passed

Memory after adding n = 2^i + 1 items: 48.00 n + 152.00   (R^2 = 1.000)



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          632
=> passed       64         3320
=> passed      256        12536
=> passed     1024        49400
=> passed     4096       196856
==> 5/5 tests passed

Memory: 48.00 n + 200.00   (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          248
=> passed       64          248
=> passed      256          248
=> passed     1024          248
=> passed     4096          248
==> 5/5 tests passed

Memory after adding n = 2^i items: 248.00   (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 addFirst()
         and addLast() are (p1, p2), respectively. The probabilities 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.util.NoSuchElementException

    Deque.removeFirst(Deque.java:121)
    MemoryOfDeque.loiter(MemoryOfDeque.java:553)
    MemoryOfDeque.test6b(MemoryOfDeque.java:631)
    MemoryOfDeque.main(MemoryOfDeque.java:747)

    - sequence of operations was:
         deque.addFirst("XTDOUBDRDN")
         deque.addFirst("RKSOHDHOUH")
         deque.removeFirst()     ==> RKSOHDHOUH
         deque.removeFirst()     ==> XTDOUBDRDN
         deque.addLast("CNGZVOQVLU")
         deque.removeLast()      ==> CNGZVOQVLU
         deque.addLast("RJUVBHUYOU")
         deque.removeFirst()

==> 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!

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



Analyzing memory of RandomizedQueue
*-----------------------------------------------------------
For tests 1-5, the maximum amount of memory allowed for
a RandomizedQueue containing n items is 48n + 192.

Test 1a-1i: Total memory usage after inserting n items
            when n is a power of 2.

                 n        bytes
-------------------------------------------------------------
=> passed       32          312
=> passed       64          568
=> passed      128         1080
=> passed      256         2104
=> passed      512         4152
=> passed     1024         8248
=> passed     2048        16440
=> passed     4096        32824
=> passed     8192        65592
==> 9/9 tests passed

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


Test 2a-2i: Total memory usage after inserting n items,
            when n is one more than a power of 2.

                 n        bytes
-------------------------------------------------------------
=> passed       33          568
=> passed       65         1080
=> passed      129         2104
=> passed      257         4152
=> passed      513         8248
=> passed     1025        16440
=> passed     2049        32824
=> passed     4097        65592
=> passed     8193       131128
==> 9/9 tests passed

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


Test 3a-3i: Total memory usage after inserting 2n-1 items, and then
            deleting n-1 items, when n is one more than a power of 2.

                 n        bytes
-------------------------------------------------------------
=> passed       33         1080
=> passed       65         2104
=> passed      129         4152
=> passed      257         8248
=> passed      513        16440
=> passed     1025        32824
=> passed     2049        65592
=> passed     4097       131128
=> passed     8193       262200
==> 9/9 tests passed

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


Test 4a-4i: Total memory usage after inserting n items, deleting n items,
            then inserting n times, when n is a power of 2.
```

```
                 n        bytes
-------------------------------------------------------------
=> passed       32          312
=> passed       64          568
=> passed      128         1080
=> passed      256         2104
=> passed      512         4152
=> passed     1024         8248
=> passed     2048        16440
=> passed     4096        32824
=> passed     8192        65592
==> 9/9 tests passed

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


Test 5a-5i: Total memory usage after inserting n items,
            and then deleting all but one item.

                 n        bytes
-------------------------------------------------------------
=> passed       32           48
=> passed       64           48
=> passed      128           48
=> passed      256           48
=> passed      512           48
=> passed     1024           48
=> passed     2048           48
=> passed     4096           48
=> passed     8192           48
==> 9/9 tests passed

Memory: 48.00   (R^2 = 1.000)


Test 6a-6d: Total memory usage of iterator after inserting n items.

                 n        bytes
-------------------------------------------------------------
=> passed       32           32
=> passed       64           32
=> passed      128           32
=> passed      256           32
=> passed      512           32
=> passed     1024           32
=> passed     2048           32
=> passed     4096           32
=> passed     8192           32
==> 9/9 tests passed

Memory: 32.00   (R^2 = 1.000)


Test 7a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
    java.lang.NullPointerException: null passed as 'objectToSize' in getObjectSize

    sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImpl.java:188)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.java:178)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.java:165)
    MemoryOfRandomizedQueue.test7a(MemoryOfRandomizedQueue.java:493)
    MemoryOfRandomizedQueue.main(MemoryOfRandomizedQueue.java:740)

Test 7b: 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)
    java.lang.NullPointerException: null passed as 'objectToSize' in getObjectSize

    sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImpl.java:188)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.java:178)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.java:165)
    MemoryOfRandomizedQueue.loiter(MemoryOfRandomizedQueue.java:424)
    MemoryOfRandomizedQueue.test7b(MemoryOfRandomizedQueue.java:530)
    MemoryOfRandomizedQueue.main(MemoryOfRandomizedQueue.java:741)


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

    sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImpl.java:188)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.java:178)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.java:165)
    MemoryOfRandomizedQueue.loiter(MemoryOfRandomizedQueue.java:424)
    MemoryOfRandomizedQueue.test7b(MemoryOfRandomizedQueue.java:531)
    MemoryOfRandomizedQueue.main(MemoryOfRandomizedQueue.java:741)

    - sequence of operations was:
         rq.enqueue("PMPPSYUXZQ")
         rq.enqueue("KQVBIXORQY")
         rq.dequeue()     ==> "PMPPSYUXZQ"
         rq.dequeue()     ==> "null"

  * 200 random operations (0.6, 0.2, 0.2)
    java.lang.NullPointerException: null passed as 'objectToSize' in getObjectSize

    sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImpl.java:188)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.java:178)
```

```
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.java:165)
    MemoryOfRandomizedQueue.loiter(MemoryOfRandomizedQueue.java:424)
    MemoryOfRandomizedQueue.test7b(MemoryOfRandomizedQueue.java:532)
    MemoryOfRandomizedQueue.main(MemoryOfRandomizedQueue.java:741)


  * 200 random operations (0.2, 0.4, 0.4)
    java.lang.NullPointerException: null passed as 'objectToSize' in getObjectSize

    sun.instrument.InstrumentationImpl.getObjectSize(InstrumentationImpl.java:188)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOf0(MemoryUtil.java:178)
    com.javamex.classmexer.MemoryUtil.deepMemoryUsageOfAll(MemoryUtil.java:165)
    MemoryOfRandomizedQueue.loiter(MemoryOfRandomizedQueue.java:424)
    MemoryOfRandomizedQueue.test7b(MemoryOfRandomizedQueue.java:533)
    MemoryOfRandomizedQueue.main(MemoryOfRandomizedQueue.java:741)


==> FAILED

Test 8: Insert T items into queue; then iterate over queue and check
        that worst-case constant memory is allocated or deallocated
        per iterator operation.
  * T = 64
  * T = 128
  * T = 256
==> passed

Test 9: Total memory usage after inserting n items, seeking to identify
        values of n where memory usage is minimized as a function of n.

                 n        bytes
------------------------------------------------------------
=> passed        8          120
=> passed       16          184
=> passed       32          312
=> passed       64          568
=> passed      128         1080
=> passed      256         2104
=> passed      512         4152
=> passed     1024         8248
=> passed     2048        16440
==> 9/9 tests passed

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


Test 10: Total memory usage after inserting 4096 items, then successively
         deleting items, seeking values of n where memory usage is maximized
         as a function of n

                 n        bytes
------------------------------------------------------------
=> passed     2049        65592
=> passed     1025        32824
=> passed      513        16440
=> passed      257         8248
=> passed      129         4152
=> passed       65         2104
=> passed       33         1080
=> passed       17          544
=> passed        9          288
==> 9/9 tests passed

Memory: -0.00 n^2 + 32.22 n + 4.81   (R^2 = 1.000)


Min observed memory for RandomizedQueue: -0.00 n^2 + 32.22 n + 4.81   (R^2 = 1.000)
Max observed memory for RandomizedQueue: 32.00 n + 24.00   (R^2 = 1.000)

Running 75 total tests.

Total: 73/75 tests passed!

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



********************************************************************************
*  TIMING
********************************************************************************

Timing Deque
*------------------------------------------------------------
Running 55 total tests.

Test 1a-1g: make n random calls to addFirst(), removeFirst(), isEmpty(), and size()
            with probabilities (0.7, 0.1, 0.1, 0.1)

                   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     0.02
=> passed      512000     0.04
=> passed     1024000     0.08
=> passed     2048000     0.12
==> 12/12 tests passed


Test 2a-2g: make n random calls to addLast(), removeLast(), isEmpty(), and size(),
            with probabilities (0.7, 0.1, 0.1, 0.1)

                    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     0.01
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.04
=> passed     2048000     0.07
==> 12/12 tests passed


Test 3a-3g: make n random calls to addFirst(), addLast(), removeFirst(), removeLast(),
            isEmpty(), and size() with probabilities (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)

                    n  seconds
------------------------------
    java.lang.NullPointerException

    Deque.addFirst(Deque.java:82)
    TimeDeque.timeRandomOperations(TimeDeque.java:28)
    TimeDeque.timeRandomOperations(TimeDeque.java:133)
    TimeDeque.test3(TimeDeque.java:193)
    TimeDeque.main(TimeDeque.java:288)

=> FAILED        1024   Test did not complete due to an exception.

==> 0/12 tests passed


Test 4a-4g: make n calls to addFirst(); iterate over the n items 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     0.00
=> passed      512000     0.01
=> passed     1024000     0.06
=> passed     2048000     0.17
==> 12/12 tests passed


Test 5a-5g: make n calls to addFirst()/addLast(); interleave n calls each to
            removeFirst()/removeLast() and addFirst()/addLast()

                    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: 43/55 tests passed!


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



Timing RandomizedQueue
*-----------------------------------------------------------
Running 58 total tests.

Test 1: make n calls to enqueue(); make n calls to dequeue();
        count calls to StdRandom
  * n = 10
  * n = 100
  * n = 1000
==> passed

Test 2: make n calls to enqueue(); make n calls to sample();
```

```
       count calls to StdRandom
  * n = 10
  * n = 100
  * n = 1000
==> passed

Test 3: make n calls to enqueue(); iterate over the n items;
       count calls to StdRandom
  * n = 10
    - iteration should call StdRandom() at most once per item
    - number of items                              = 10
    - number of elementary StdRandom() operations = 16

  * n = 100
    - iteration should call StdRandom() at most once per item
    - number of items                              = 100
    - number of elementary StdRandom() operations = 128

  * n = 1000
    - iteration should call StdRandom() at most once per item
    - number of items                              = 1000
    - number of elementary StdRandom() operations = 1024

==> FAILED

Test 4a-g: make n random calls to enqueue(), sample(), dequeue(), isEmpty(),
           and size() with probabilities (0.2, 0.2, 0.2, 0.2, 0.2)

                    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     0.01
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.04
=> passed     2048000     0.09
==> 12/12 tests passed


Test 5a-g: make n random calls to enqueue(), sample(), dequeue(), isEmpty(),
           and size() with probabilities (0.6, 0.1, 0.1, 0.1, 0.1)

                    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     0.01
=> passed      256000     0.02
=> passed      512000     0.03
=> passed     1024000     0.07
=> passed     2048000     0.15
==> 12/12 tests passed


Test 6a-g: make n random calls to enqueue(), sample(), dequeue(), isEmpty(),
           and size() with probabilities (0.1, 0.1, 0.6, 0.1, 0.1)

                    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     0.01
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.04
=> passed     2048000     0.08
==> 12/12 tests passed


Test 7a-g: make n calls to enqueue(); iterate over the n items
           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     0.02
=> passed      512000     0.04
=> passed     1024000     0.07
=> passed     2048000     0.12
==> 12/12 tests passed


Test 8a-g: make n calls to enqueue(); 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: 57/58 tests passed!


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