

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

## ASSESSMENT SUMMARY

Compilation: PASSED (0 errors, 3 warnings)

API: PASSED

SpotBugs: PASSED

PMD: PASSED

Checkstyle: PASSED

Correctness: 49/49 tests passed

Memory: 124/124 tests passed

Timing: 193/193 tests passed

Aggregate score: 100.00%

[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]

## ASSESSMENT DETAILS

The following files were submitted:

```
-----  
3.8K May  9 17:33 Deque.java  
 690 May  9 17:33 Permutation.java  
3.6K May  9 17:33 RandomizedQueue.java
```

```
*****  
* COMPILING  
*****
```

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

```
% javac RandomizedQueue.java  
-----
```

```
RandomizedQueue.java:20: warning: [unchecked] unchecked cast  
    itemList = (Item[]) new Object[1];  
           ^
```

```
required: Item[]  
found:   Object[]  
where Item is a type-variable:  
  Item extends Object declared in class RandomizedQueue
```

```
RandomizedQueue.java:65: warning: [unchecked] unchecked cast  
    Item[] copy = (Item[]) new Object[n];  
           ^
```

```
required: Item[]  
found:   Object[]  
where Item is a type-variable:  
  Item extends Object declared in class RandomizedQueue
```

```
RandomizedQueue.java:91: warning: [unchecked] unchecked cast  
    copy = (Item[]) new Object[tail];  
           ^
```

```
required: Item[]  
found:   Object[]  
where Item is a type-variable:  
  Item extends Object declared in class RandomizedQueue
```

```
Item extends Object declared in class RandomizedQueue  
3 warnings
```

```
% javac Permutation.java
```

```
=====
```

```
Checking the APIs of your programs.
```

```
*-----  
Deque:
```

```
RandomizedQueue:
```

```
Permutation:
```

```
=====
```

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

```
% spotbugs *.class
```

```
*-----  
% pmd .
```

```
*-----  
% checkstyle *.java
```

```
*-----  
% 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 19 total tests.
```

```
Tests 1–8 make random intermixed calls to addFirst(), addLast(),  
removeFirst(), removeLast(), isEmpty(), and size(), and iterator().  
The probabilities of each operation are (p1, p2, p3, p4, p5, p6, p7),  
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, 0.0)  
\* 50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)  
\* 500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)  
\* 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)  
==> 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, 0.0)  
\* 50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)  
\* 500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)  
\* 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)  
\* 5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)  
\* 50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)  
\* 500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)  
\* 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 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, 0.0)  
\* 50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)  
\* 500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)  
\* 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)  
\* 5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)  
\* 50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)  
\* 500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)  
\* 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)  
==> passed

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

Test 6: check random calls to addFirst(), removeFirst(), and iterator()  
\* 5 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)  
\* 50 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)  
\* 500 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)  
\* 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)  
\* 5 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)  
\* 50 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)  
\* 500 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)  
\* 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)  
==> passed

Test 7: check random calls to all methods except iterator()  
\* 5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)  
\* 50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)

```
* 500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
* 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
==> passed
```

```
Test 8: check random calls to all methods, including iterator()
* 5 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
* 50 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
* 500 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
* 1000 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
* 5 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
* 50 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
* 500 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
* 1000 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
==> passed
```

```
Test 9: check removeFirst() and removeLast() from an empty deque
* removeFirst()
* removeLast()
==> passed
```

```
Test 10: check whether two Deque objects can be created at the same time
* n = 10
* n = 1000
==> passed
```

```
Test 11: check iterator() after n calls to addFirst()
* n = 10
* n = 50
==> passed
```

```
Test 12: check iterator() after random calls to addFirst(), addLast(),
removeFirst(), and removeLast() with probabilities (p1, p2, p3, p4)
* 20 random operations (0.8, 0.0, 0.2, 0.0)
* 20 random operations (0.8, 0.0, 0.0, 0.2)
* 20 random operations (0.0, 0.8, 0.0, 0.2)
* 20 random operations (0.0, 0.8, 0.2, 0.0)
* 20 random operations (0.4, 0.4, 0.1, 0.1)
* 20 random operations (0.2, 0.0, 0.8, 0.0)
* 20 random operations (0.2, 0.0, 0.0, 0.8)
* 20 random operations (0.0, 0.2, 0.0, 0.8)
* 20 random operations (0.0, 0.2, 0.8, 0.0)
* 20 random operations (0.1, 0.1, 0.4, 0.4)
* 100 random operations (0.4, 0.4, 0.1, 0.1)
* 1000 random operations (0.4, 0.4, 0.1, 0.1)
==> passed
```

```
Test 13: create two nested iterators to same deque of size n
* n = 10
* n = 50
==> passed
```

```
Test 14: create two parallel iterators to same deque of size n
* n = 10
* n = 50
==> passed
```

```
Test 15: create an iterator and check calls to next() and hasNext()
* 10 consecutive calls to hasNext() on a deque of size 10
* 10 consecutive calls to next() on a deque of size 10
* 50 random intermixed calls to next() and hasNext() on a deque of size 10
* 1000 random intermixed calls to next() and hasNext() on a deque of size 100
==> passed
```

```
Test 16: create Deque objects of different parameterized types
==> passed
```

```
Test 17: call addFirst() and addLast() with null argument
==> passed
```

Test 18: check that remove() and next() throw the specified exceptions in iterator()  
==> passed

Test 19: call iterator() when the deque is empty  
==> passed

Total: 19/19 tests passed!

=====

Testing correctness of RandomizedQueue

\*-----

Running 21 total tests.

Tests 1–6 make random intermixed calls to enqueue(), dequeue(), sample(), isEmpty(), size(), and iterator(). The probabilities of each operation are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: check random calls to enqueue() and size()  
\* 5 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)  
\* 50 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)  
\* 500 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)  
\* 1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)

==> passed

Test 2: check random calls to enqueue() and dequeue()  
\* 5 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)  
\* 50 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)  
\* 500 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)  
\* 1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)  
\* 5 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)  
\* 50 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)  
\* 500 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)  
\* 1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)

==> passed

Test 3: check random calls to enqueue() and sample()  
\* 5 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)  
\* 50 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)  
\* 500 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)  
\* 1000 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)  
\* 5 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)  
\* 50 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)  
\* 500 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)  
\* 1000 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)

==> passed

Test 4: check random calls to enqueue() and iterator()  
\* 5 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)  
\* 50 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)  
\* 500 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)  
\* 1000 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)

==> passed

Test 5: check random calls to all methods except iterator()  
\* 5 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 50 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 500 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)  
\* 5 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)  
\* 50 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)  
\* 500 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)  
\* 1000 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)

==> passed

Test 6: check random calls to all methods, including iterator()  
\* 5 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)  
\* 50 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)

```
* 500 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
* 1000 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
* 5 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
* 50 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
* 500 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
* 1000 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
==> passed
```

Test 7: call dequeue() and sample() from an empty randomized queue

```
* dequeue()
* sample()
==> passed
```

Test 8: create multiple randomized queue objects at the same time

```
* n = 10
* n = 100
==> passed
```

Test 9: check that iterator() returns correct items after a sequence  
of n enqueue() operations

```
* n = 10
* n = 50
==> passed
```

Test 10: check that iterator() returns correct items after intermixed  
sequence of m enqueue() and dequeue() operations

```
* m = 10
* m = 1000
==> passed
```

Test 11: create two nested iterators over the same randomized queue of size n

```
* n = 10
* n = 50
==> passed
```

Test 12: create two parallel iterators over the same randomized queue of size n

```
* n = 10
* n = 50
==> passed
```

Test 13: create two iterators over different randomized queues,  
each of length 10

```
==> passed
```

Test 14: create an iterator and check calls to next() and hasNext()

```
* 10 consecutive calls to hasNext() on a deque of size 10
* 10 consecutive calls to next() on a deque of size 10
* 50 random intermixed calls to next() and hasNext() on a deque of size 10
* 1000 random intermixed calls to next() and hasNext() on a deque of size 100
==> passed
```

Test 15: create RandomizedQueue objects of different parameterized types

```
==> passed
```

Test 16: 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 17: check randomness of dequeue() by enqueueing n items, dequeuing n items,  
and seeing whether each of the n! permutations is equally likely

```
* n = 2, trials = 12000
* n = 3, trials = 12000
* n = 4, trials = 12000
* n = 5, trials = 12000
==> passed
```

```
Test 18: 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
* n = 4, trials = 12000
* n = 5, trials = 12000
==> passed
```

```
Test 19: call enqueue() with a null argument
==> passed
```

```
Test 20: check that remove() and next() throw the specified exceptions in iterator()
==> passed
```

```
Test 21: call iterator() when randomized queue is empty
==> passed
```

Total: 21/21 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
C
G
E
```

```
% java Permutation 3 < distinct.txt
B
E
F
```

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

```
==> passed
```

```
Test 1b: check formatting for other inputs
```

```
% java Permutation 8 < mediumTale.txt
age
the
it
times
was
best
was
the
```

```
% 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	0.00
=> passed	16000	0.00
=> passed	32000	0.01
=> passed	64000	0.02
=> passed	128000	0.04
=> passed	256000	0.07
=> passed	512000	0.15
==> 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	0.00
=> passed	32000	0.01
=> passed	64000	0.02
=> passed	128000	0.04
=> passed	256000	0.07
=> passed	512000	0.15
==> 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 49 total tests.

Test 1a-1i: total memory usage after inserting n items,

where n is a power of 2

	n	bytes
=> passed	32	1576
=> passed	64	3112
=> passed	128	6184
=> passed	256	12328
=> passed	512	24616
=> passed	1024	49192
=> passed	2048	98344
=> passed	4096	196648
=> passed	8192	393256
==>	9/9 tests passed	

Memory: 48.00 n + 40.00 (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	1624
=> passed	65	3160
=> passed	129	6232
=> passed	257	12376
=> passed	513	24664
=> passed	1025	49240
=> passed	2049	98392
=> passed	4097	196696
=> passed	8193	393304
==>	9/9 tests passed	

Memory: 48.00 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	1624
=> passed	65	3160
=> passed	129	6232
=> passed	257	12376
=> passed	513	24664
=> passed	1025	49240
=> passed	2049	98392
=> passed	4097	196696
=> passed	8193	393304
==>	9/9 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  
(should not grow with n or be too large of a constant).

	n	bytes
=> passed	32	88
=> passed	64	88
=> passed	128	88
=> passed	256	88
=> passed	512	88
=> passed	1024	88
=> passed	2048	88

```
=> passed      4096          88  
=> passed      8192          88  
==> 9/9 tests passed
```

Memory: 88.00 (R<sup>2</sup> = 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	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<sup>2</sup> = 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 (p<sub>1</sub>, p<sub>2</sub>), respectively. The probabilities of  
removeFirst() and removeLast() are (q<sub>1</sub>, q<sub>2</sub>), 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 (p<sub>1</sub>, p<sub>2</sub>, p<sub>3</sub>, p<sub>4</sub>),  
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: Perform m random add/remove operations in the deque and check  
that only constant memory is allocated/deallocated per operation

- \* m = 128
- \* m = 256
- \* m = 512

==> passed

Test 8: Insert m items into deque; then iterate over deque and check  
that only constant memory is allocated/deallocated per operation

- \* m = 64
- \* m = 128
- \* m = 256

==> passed

Min observed memory for Deque: 48.00 n + 40.00 (R<sup>2</sup> = 1.000)

Max observed memory for Deque: 48.00 n + 40.00 (R<sup>2</sup> = 1.000)

Total: 49/49 tests passed!

=====  
Analyzing memory of RandomizedQueue

\*  
For Tests 1–5, the maximum amount of memory allowed for  
a RandomizedQueue containing n items is  $48n + 192$ .

For Test 6, the maximum amount of memory allowed for  
a RandomizedQueue iterator over n items is  $8n + 72$ .

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

	n	bytes
=> passed	32	320
=> passed	64	576
=> passed	128	1088
=> passed	256	2112
=> passed	512	4160
=> passed	1024	8256
=> passed	2048	16448
=> passed	4096	32832
=> passed	8192	65600
==>	9/9 tests passed	

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

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

	n	bytes
=> passed	33	576
=> passed	65	1088
=> passed	129	2112
=> passed	257	4160
=> passed	513	8256
=> passed	1025	16448
=> passed	2049	32832
=> passed	4097	65600
=> passed	8193	131136
==>	9/9 tests passed	

Memory:  $16.00 n + 48.00$  ( $R^2 = 1.000$ )

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	1088
=> passed	65	2112
=> passed	129	4160
=> passed	257	8256
=> passed	513	16448
=> passed	1025	32832
=> passed	2049	65600
=> passed	4097	131136
=> passed	8193	262208
==>	9/9 tests passed	

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

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	320
=> passed	64	576
=> passed	128	1088
=> passed	256	2112
=> passed	512	4160
=> passed	1024	8256
=> passed	2048	16448
=> passed	4096	32832
=> passed	8192	65600
==>	9/9 tests passed	

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

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

	n	bytes
=> passed	32	80
=> passed	64	80
=> passed	128	80
=> passed	256	80
=> passed	512	80
=> passed	1024	80
=> passed	2048	80
=> passed	4096	80
=> passed	8192	80
==>	9/9 tests passed	

Memory: 80.00 ( $R^2 = 1.000$ )

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

	n	bytes
=> passed	32	328
=> passed	64	584
=> passed	128	1096
=> passed	256	2120
=> passed	512	4168
=> passed	1024	8264
=> passed	2048	16456
=> passed	4096	32840
=> passed	8192	65608
==>	9/9 tests passed	

Memory: 8.00 n + 72.00 ( $R^2 = 1.000$ )

Test 7a: Insert 100 strings; delete them one at a time, checking for loitering after each deletion.

==> passed

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)
- \* 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 8: Insert  $m$  items into queue; then iterate over deque and check  
that only constant memory is allocated/deallocated per operation  
\*  $m = 64$   
\*  $m = 128$   
\*  $m = 256$   
==> 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	32	320
=> passed	64	576
=> passed	128	1088
=> passed	256	2112
=> passed	512	4160
=> passed	1024	8256
=> passed	2048	16448
==>	7/7 tests passed	

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

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	65600
=> passed	1025	32832
=> passed	513	16448
=> passed	257	8256
=> passed	129	4160
=> passed	65	2112
=> passed	33	1088
=> passed	17	576
=> passed	9	320
==>	9/9 tests passed	

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

Min observed memory for RandomizedQueue:  $8.00 n + 64.00$  ( $R^2 = 1.000$ )  
Max observed memory for RandomizedQueue:  $32.00 n + 32.00$  ( $R^2 = 1.000$ )

Running 73 total tests.

Total: 73/73 tests passed!

=====

\*\*\*\*\*  
\* TIMING  
\*\*\*\*\*

Timing Deque

=====

Running 103 total tests.

Test 1a-1k: make  $n$  calls to addFirst() followed by  $n$  calls to removeFirst()  
 $n$  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    0.02
==> 11/11 tests passed
```

Test 2a-2k: make n calls to addLast() followed by n calls to 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    0.00
=> passed     65536    0.00
=> passed    128000    0.01
=> passed   256000    0.00
=> passed   512000    0.01
=> passed  1024000    0.02
==> 11/11 tests passed
```

Test 3a-3k: make n calls to addFirst() followed by n calls to 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    0.00
=> passed     65536    0.00
=> passed    128000    0.00
=> passed   256000    0.00
=> passed   512000    0.01
=> passed  1024000    0.02
==> 11/11 tests passed
```

Test 4a-4k: make n calls to addLast() followed by n calls to removeFirst()  
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    0.02
==> 11/11 tests passed
```

Test 5a-5g: 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    0.00
=> passed  128000    0.00
=> passed  256000    0.01
=> passed  512000    0.01
=> passed 1024000    0.03
=> passed 2048000    0.06
==> 12/12 tests passed
```

Test 6a-6g: 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	0.00
=> passed	256000	0.01
=> passed	512000	0.02
=> passed	1024000	0.03
=> passed	2048000	0.06
==>	12/12 tests passed	

Test 7a-7g: 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
=> 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	0.02
=> passed	1024000	0.04
=> passed	2048000	0.07
==>	12/12 tests passed	

Test 8a-8g: 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    0.02
=> passed     2048000    0.04
==> 12/12 tests passed
```

Test 9a-9k: make n calls to addFirst()/addLast(); interleave n calls each to removeFirst(), removeLast(), addFirst(), and addLast()

n	seconds	
=> passed	1025	0.00
=> passed	2049	0.00
=> passed	4097	0.00
=> passed	8193	0.00
=> passed	16385	0.00
=> passed	32769	0.00
=> passed	65537	0.01
=> passed	128001	0.01
=> passed	256001	0.02
=> passed	512001	0.03
=> passed	1024001	0.05
==> 11/11 tests passed		

Total: 103/103 tests passed!

=====

Timing RandomizedQueue

```
*-----  
Running 67 total tests.
```

Test 1: make n calls to enqueue() followed by n calls to dequeue();  
count calls to StdRandom

- \* n = 10
- \* n = 100
- \* n = 1000

```
==> passed
```

Test 2: make n calls to enqueue() followed by n calls to sample();  
count calls to StdRandom

- \* n = 10
- \* n = 100
- \* n = 1000

```
==> passed
```

Test 3: make n calls to enqueue() and iterate over the n items;  
count calls to StdRandom

- \* n = 10
- \* n = 100
- \* n = 1000

```
==> passed
```

Test 4a-k: make n calls to enqueue() followed by n calls to dequeue()

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    0.01
=> passed      1024000   0.03
==> 11/11 tests passed
```

Test 5a-k: make n calls to enqueue() followed by 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	0.01
=> passed	128000	0.01
=> passed	256000	0.02
=> passed	512000	0.04
=> passed	1024000	0.09
==> 11/11 tests passed		

Test 6a-k: make n calls to enqueue() followed by 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
==> 11/11 tests passed		

Test 7a-k: make n calls to enqueue() followed by 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	0.02
=> passed	512000	0.04
=> passed	1024000	0.12
==> 11/11 tests passed		

Test 8a-k: make n calls to enqueue() followed by n calls each to 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    0.01  
=> passed  256000    0.02  
=> passed  512000    0.03  
=> passed 1024000    0.07  
==> 11/11 tests passed
```

Test 9a-i: make 100 calls to enqueue; 99 calls to dequeue;  
n calls to enqueue(); then call dequeue() three times,  
followed by enqueue() three times, and repeat n times.

```
-----  
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    0.02  
=> passed  256000    0.05  
==> 9/9 tests passed
```

Total: 67/67 tests passed!

  

```
=====
```