Frequently Asked Questions (General)

Should I use arrays or linked lists in my implementations? In general we don't tell you *how* to implement your data structures—you can use arrays, linked lists, or maybe even invent your own new structure, provided you abide by the specified time and space requirements.

How serious are you about not calling external library functions (other than those that are specifically permitted)? You will receive a substantial deduction. The goal of this assignment is to implement data types from first principles, using resizing arrays and linked lists. We also require you to use `StdIn` (instead of java.util.Scanner ☕) because we will intercept the calls to `StdIn` in our testing.

Can I add extra public methods to the `Deque` or `RandomizedQueue` APIs? Can I use different names for the methods? No. You must implement the API exactly as specified. As usual, extra private methods are permitted.

What should my deque (or randomized queue) iterator do if the deque (or randomized queue) is structurally modified at any time after the iterator is created (but before it is done iterating)? You don't need to worry about this in your solution. An industrial-strength solution (used in the Java libraries) is to make the iterator *fail-fast*: throw a java.util.ConcurrentModificationException ☕ as soon as this is detected.

Why does the following code lead to a `generic array creation` compile-time error when `Item` is a generic type parameter?

```
Item[] a = new Item[1];
```

Java prohibits the creation of arrays of generic types. See the Q+A in Section 1.3 for a brief discussion. Instead, use a cast.

```
Item[] a = (Item[]) new Object[1];
```

Unfortunately, this leads to an unavoidable compiler warning.

The compiler says that my program uses an unchecked cast. Is this warning OK? If it is the generic-array-creation warning described in the previous question, that is OK. However, *you should not receive any other compiler warnings on this assignment (and you should not receive any compiler warnings on any other assignment).*

Checkstyle complains that my nested class' instance variables must be private and have accessor methods that are not private. Do I need to make them private? No, but there's no harm in doing so. The access modifier of a private nested class' instance variable is irrelevant—regardless of its access modifier, it can be accessed anywhere in the file. (Of course, the enclosing class' instance variables should be private.)

Can a nested class have a constructor and instance variables? Yes, just like any other class. For example, see `ReverseArrayIterator` in ResizingArrayStack.java ☕.

Will I lose points for loitering? Yes. See p. 137 of the textbook for a discussion of loitering. Loitering is maintaining a useless reference to an object that could otherwise be garbage collected.

Frequently Asked Questions (Randomized Queue)

What does "uniformly at random" mean? If there are n items in the randomized queue, then you should choose each one with probability $1/n$, up to the randomness of `StdRandom.uniform()`, independent of past decisions. You can generate a pseudo-random integer between 0 and $n - 1$ using `StdRandom.uniform(n)`.

How can I rearrange the entries of an array in uniformly random order? Use `StdRandom.shuffle()`—it implements the Knuth shuffle (see page 32 of the textbook) and runs in linear time. Note that depending on your implementation, you may not need to call this method.

Can repeated calls to `sample()` in a randomized queue return the same item more than once? Yes, since you are sampling without removing the item. This is also known as "sampling with replacement."

Should two iterators to the same randomized queue return the items in the same order? No, each iterator should have a different random order. This is what "independent iterators" means.

Why do I need to allocate $\Theta(n)$ extra memory in the iterator constructor? You need to store the random order associated with each iterator.

Why is it called a randomized queue if the items are not removed in first-in first-out order? This is the name commonly used for this policy in queueing theory.

Frequently Asked Questions (Permutation)

What assumptions can I make about the input to `Permutation`? Standard input can contain any sequence of strings. You may assume that there is one integer command-line argument k and it is between 0 and the number of strings on standard input.

Testing

Develop unit tests as you write each method and constructor to allow for testing. As an example for `Deque`, you know that if you call `addFirst()` with the numbers 1 through n in ascending order, then call `removeLast()` n times, you should see the numbers 1 through n in ascending order. As soon as you have those two methods written, you can write a unit test for these methods.

Test intermixed sequence of operations. Sometimes you want to test two methods in isolation, as above. But, you also need to make sure that all of the methods work together with one another.

Test your iterator. Test that multiple iterators can be used simultaneously and operate independently of one another. For example, the following code fragment uses two nested iterators:

```
int n = 5;
RandomizedQueue<Integer> queue = new RandomizedQueue<Integer>();
for (int i = 0; i < n; i++)
    queue.enqueue(i);
for (int a : queue) {
    for (int b : queue)
        StdOut.print(a + "-" + b + " ");
    StdOut.println();
}
```

It should produce a result like the following:

```
0-2 0-1 0-3 0-0 0-4
1-2 1-1 1-4 1-0 1-3
2-1 2-4 2-0 2-3 2-2
4-2 4-3 4-4 4-1 4-0
3-2 3-3 3-1 3-0 3-4
```

Make sure to test what happens when your data structures are emptied. A common bug is for something to go wrong when your data structure goes from non-empty to empty and then back to non-empty. Make sure to include this in your tests.

Don't rely solely on the autograder for debugging. Write your own unit tests; it's good practice in this course and beyond.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps. These same steps apply to each of the two data types that you will be implementing.