Goal: Implement a double-ended queue (deque) using a doubly-linked list and a random queue using a resizing array.

Part I: Warmup Problems

The problems in this part of the assignment are intended to give you solid practice on concepts (using and defining data types) needed to solve the problems in Part II.

Problem 1. (*Iterable Primes*) Implement an immutable, iterable data type called `Primes` to iterate over the first n primes. The data type must support the following API:

Primes	
`Primes(int n)`	constructs a `Primes` object given the number of primes needed
`Iterator<Integer> iterator()`	returns an iterator to iterate over the first n primes

```
>_ ~/workspace/collections
$ javac -d out src/Primes.java
$ java Primes 10
2
3
5
7
11
13
17
19
23
29
```

Problem 2. (*Min Max*) Implement a library called `MinMax` with functions `min()` and `max()` that each accept a reference `first` to the first node in a linked list of integers and return the minimum and the maximum values respectively.

```
>_ ~/workspace/collections
$ javac -d out src/MinMax.java
$ java MinMax
min(first) == StdStats.min(items)? true
max(first) == StdStats.max(items)? true
```

Part II: Collections

Problem 3. (*Deque*) A double-ended queue or deque (pronounced "deck") is a generalization of a stack and a queue that supports adding and removing items from either the front or the back of the data structure. Create a generic, iterable data type called `LinkedDeque` that uses a doubly-linked list to implement the following deque API:

LinkedDeque	
`LinkedDeque()`	constructs an empty deque
`boolean isEmpty()`	returns `true` if this deque empty, and `false` otherwise
`int size()`	returns the number of items on this deque
`void addFirst(T item)`	adds `item` to the front of this deque
`void addLast(T item)`	adds `item` to the back of this deque
`T peekFirst()`	returns the item at the front of this deque
`T removeFirst()`	removes and returns the item at the front of this deque
`T peekLast()`	returns the item at the back of this deque
`T removeLast()`	removes and returns the item at the back of this deque
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this deque from front to back
`String toString()`	returns a string representation of this deque

Corner Cases

- The `add*()` methods should throw a `NullPointerException("item is null")` if *item* is `null`.
- The `peek*()` and `remove*()` methods should throw a `NoSuchElementException("Deque is empty")` if the deque is empty.
- The `next()` method in the deque iterator shoud throw a `NoSuchElementException("Iterator is empty")` if there are no more items to iterate.

Performance Requirements

- The constructor and methods in `LinkedDeque` and `DequeIterator` should run in time $T(n) \sim 1$.

```
>_ ~/workspace/collections
$ javac -d out src/LinkedDeque.java
$ java LinkedDeque
Filling the deque...
The deque (364 characters): There is grandeur in this view of life, with its several powers, having been originally
breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law
of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being,
evolved. ~ Charles Darwin, The Origin of Species
Emptying the deque...
deque.isEmpty()? true
```

Problem 4. (*Random Queue*) A random queue is similar to a stack or queue, except that the item removed is chosen uniformly at random from items in the data structure. Create a generic, iterable data type called `ResizingArrayRandomQueue` that uses a resizing array to implement the following random queue API:

ResizingArrayRandomQueue	
`ResizingArrayRandomQueue()`	constructs an empty random queue
`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds *item* to the end of this queue
`T sample()`	returns a random item from this queue
`T dequeue()`	removes and returns a random item from this queue
`Iterator<T> iterator()`	returns an independent† iterator to iterate over the items in this queue in random order
`String toString()`	returns a string representation of this queue

† The order of two or more iterators on the same randomized queue must be mutually independent, ie, each iterator must maintain its own random order.

Corner Cases

- The `enqueue()` method should throw a `NullPointerException("item is null")` if *item* is `null`.
- The `sample()` and `dequeue()` methods should throw a `NoSuchElementException("Random queue is empty")` if the random queue is empty.
- The `next()` method in the random queue iterator shoud throw a `NoSuchElementException("Iterator is empty")` if there are no more items to iterate.

Performance Requirements

- The constructor and methods in `ResizingArrayRandomQueue` should run in time $T(n) \sim 1$.
- The constructor in `RandomQueueIterator` should run in time $T(n) \sim n$.
- The methods in `RandomQueueIterator` should run in time $T(n) \sim 1$.

```
>_ ~/workspace/collections
$ javac -d out src/ResizingArrayRandomQueue.java
$ java ResizingArrayRandomQueue
sum       = 5081434
iterSumQ  = 5081434
dequeSumQ = 5081434
iterSumQ + dequeSumQ == 2 * sum? true
```

Files to Submit:

1. `Primes.java`
2. `MinMax.java`
3. `LinkedDeque.java`
4. `ResizingArrayRandomQueue.java`
5. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond *Basic Data Structures*.
- Your code follows good programming principles (ie, it is clean and well-organized; uses meaningful variable names; and includes useful comments).
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, you must include in nore more than 100 words: a short, high-level description of the problem; your approach to solve it; and any issues you encountered and if/how you managed to solve them.

Acknowledgement: Part II of this assignment is an adaptation of the Deques and Randomized Queues assignment developed at Princeton University by Kevin Wayne.