This document only contains the description of the project and the project problems. For the programming exercises on concepts related to the project, please refer to the project checklist .

Goal The purpose of this project is to implement elementary data structures using arrays and linked lists, and to introduce you to generics and iterators.

Problem 1. (*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(Item item)`	adds `item` to the front of this deque
`void addLast(Item item)`	adds `item` to the back of this deque
`Item peekFirst()`	returns the item at the front of this deque
`Item removeFirst()`	removes and returns the item at the front of this deque
`Item peekFirst()`	returns the item at the back of this deque
`Item removeLast()`	removes and returns the item at the back of this deque
`Iterator<Item> iterator()`	returns an iterator to iterate over the items in this deque from front to back

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 exhausted")` if there are no more items to iterate.

Performance Requirements

- The constructor and each method should run in time $T(n) \sim 1$.
- The constructor and methods in the deque iterator should run in time $T(n) \sim 1$.

~/workspace/project2

```
$ 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 2. (*Sorting Strings*) Implement a program `Sort.java` that accepts strings from standard input, stores them in a `LinkedDeque` data structure, sorts the deque, and writes the sorted strings to standard output.

Performance Requirements

- Your implementation should run in time $T(n) \sim n^2$, where n is the number of input strings.

```
>_ ~/workspace/project2
$ java Sort
A B R A C A D A B R A
<ctrl-d>
A
A
A
A
A
B
B
C
D
R
R
```

Problem 3. (*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(Item item)`	adds *item* to the end of this queue
`Item sample()`	returns a random item from this queue
`Item dequeue()`	removes and returns a random item from this queue
`Iterator<Item> iterator()`	returns an independent† iterator to iterate over the items in this queue in random order

† 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 exhausted")` if there are no more items to iterate.

Performance Requirements

- The constructor and each method should run in amortized time $T(n) \sim 1$.
- The constructor in the random queue iterator should run in time $T(n) \sim n$.
- The methods in the random queue iterator should run in time $T(n) \sim 1$.

```
>_ ~/workspace/project2
$ java ResizingArrayRandomQueue
sum       = 5081434
iterSumQ  = 5081434
dequeSumQ = 5081434
iterSumQ + dequeSumQ == 2 * sum? true
```

Problem 4. (*Sampling Integers*) Implement a program `Sample.java` that accepts *lo* (int), *hi* (int), *k* (int), and *mode* (String) as command-line arguments, uses a random queue to sample *k* integers from the interval $[lo, hi]$, and writes the samples to standard output. The sampling must be done with replacement if *mode* is "+", and without replacement if *mode* is "-". You may assume that $k \leq hi - lo + 1$.

Corner Cases

- The program should throw a `IllegalArgumentException("Illegal mode")` if *mode* is different from "+" or "-".

Performance Requirements

- The program should run in time $T(k, n) \sim kn$ in the worst case (sampling without replacement), where k is the sample size and n is the length of the sampling interval.

```
>_ ~/workspace/project2
$ java Sample 1 5 5 +
3
3
5
4
1
$ java Sample 1 5 5 -
2
3
1
4
5
```

Acknowledgements This project is an adaptation of the Deques and Randomized Queues assignment developed at Princeton University by Kevin Wayne.