

Data Structures and Algorithms in Java

Assignment 4 (Collections) Discussion

Problem 1 (Deque)

Create a generic, iterable data type called `Deque` that uses a doubly-linked list to implement the following deque API

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

Problem 1 (Deque)

× ~/workspace/collections

```
$ javac -d out src/LinkedDeque.java
```

```
$ java LinkedDeque
```

```
Filling the deque...
```

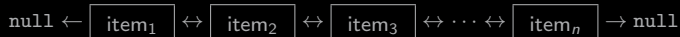
```
The deque (364 characters): There is grandeur in this view of life, with its  
several powers, having been originally breathed into a few forms or into one;  
and that, whilst this planet has gone cycling on according to the fixed law  
of gravity, from so simple a beginning endless forms most beautiful and most  
wonderful have been, and are being, evolved. ~ Charles Darwin, The Origin  
of Species
```

```
Emptying the deque...
```

```
deque.isEmpty()? true
```

Problem 1 (Deque)

Use a doubly-linked list `Node` to implement the API — each node in the list stores a generic `item`, and references `next` and `prev` to the next and previous nodes in the list



Instance variables (`LinkedDeque`)

- Reference to the front of the deque, `Node first`
- Reference to the back of the deque, `Node last`
- Size of the deque, `int n`

`LinkedDeque()`

- Initialize instance variables to appropriate values

`boolean isEmpty()`

- Return whether the deque is empty or not

`int size()`

- Return the size of the deque

Problem 1 (Deque)

`void addFirst(T item)`

- Add the given item to the front of the deque
- Increment `n` by one
- If this is the first item that is being added, both `first` and `last` must point to the same node

`void addLast(T item)`

- Add the given item to the back of the deque
- Increment `n` by one
- If this is the first item that is being added, both `first` and `last` must point to the same node

`T peekFirst()`

- Return the item at the front of the deque

`T removeFirst()`

- Remove and return the item at the front of the deque
- Decrement `n` by one
- If this is the last item that is being removed, both `first` and `last` must point to null

`T peekLast()`

- Return the item at the back of the deque

Problem 1 (Deque)

`T removeLast()`

- Remove and return the item at the back of the deque
- Decrement `n` by one
- If this is the last item that is being removed, both `first` and `last` must point to null

`Iterator<T> iterator()`

- Return an object of type `DequeIterator`

`LinkedDeque::DequeIterator`

- Instance variable
 - Reference to current node in the iterator, `Node current`

`DequeIterator()`

- Initialize instance variable appropriately

`boolean hasNext()`

- Return whether the iterator has more items to iterate or not

`T next()`

- Return the item in `current` and advance `current` to the next node

Problem 2 (Random Queue)

Create a generic, iterable data type called `ResizingArrayRandomQueue` that uses a resizing array to implement the following random queue API

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

Problem 2 (Random Queue)

× ~/workspace/collections

```
$ javac -d out src/ResizingArrayRandomQueue.java
```

```
$ java ResizingArrayRandomQueue
```

```
sum          = 5081434
```

```
iterSumQ     = 5081434
```

```
dequeSumQ    = 5081434
```

```
iterSumQ + dequeSumQ == 2 * sum? true
```


Problem 2 (Random Queue)

Use a resizing array to implement the API

Instance variables (ResizingArrayRandomQueue)

- Array to store the items of queue, `Item[] q`
- Size of the queue, `int n`

`ResizingArrayRandomQueue()`

- Initialize instance variables appropriately — create `q` with an initial capacity of 2

`boolean isEmpty()`

- Return whether the queue is empty or not

`int size()`

- Return the size of the queue

`void enqueue(T item)`

- If `q` is at full capacity, resize it to twice its current capacity
- Insert the given item in `q` at index `n`
- Increment `n` by one

Problem 2 (Random Queue)

`T sample()`

- Return `q[r]`, where `r` is a random integer from the interval $[0, n)$

`T dequeue()`

- Save `q[r]` in `item`, where `r` is a random integer from the interval $[0, n)$
- Set `q[r]` to `q[n - 1]` and `q[n - 1]` to null
- Decrement `n` by one
- If `q` is at quarter capacity, resize it to half its current capacity
- Return `item`

`Iterator<T> iterator()`

- Return an object of type `RandomQueueIterator`

Problem 2 (Random Queue)

ResizingArrayRandomQueue::RandomQueueIterator

- Instance variables
 - Array to store the items of q, T[] items
 - Index of the current item in items, int current
- RandomQueueIterator()
 - Create items with capacity n
 - Copy the n items from q into items
 - Shuffle items
 - Initialize current appropriately
- boolean hasNext()
 - Return whether the iterator has more items to iterate or not
- T next()
 - Return the item in items at index current and advance current by one