# Data Structures and Algorithms in Java

Assignment 4 (Collections) Discussion

## Part I (Warmup Problems) · Problem 1 (Iterable Primes)

Implement an immutable, iterable data type called `Primes` to iterate over the first *n* primes and supports 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
```

## **Part I (Warmup Problems)** · Problem 1 (Iterable Primes)

Instance variables

- Number of primes needed, `int n`

`Primes(int n)`

- Initialize instance variables

`Iterator<Integer> iterator()`

- Return an object of type `PrimesIterator`

`Primes::PrimesIterator`

- Instance variables
    - Number of primes returned so far, `int count`
    - The next prime, `int p`
- `PrimesIterator()`
    - Initialize instance variables
- `boolean hasNext()`
    - Return `true` if there are more primes to be interated over, and `false` otherwise
- `Integer next()`
    - As long as `p` is a prime (use `isPrime()`)
        - Increment `p` by one
    - Return `p` and increment `p` by one

# **Part I (Warmup Problems)** · 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 I (Warmup Problems) · Problem 2 (Min Max)

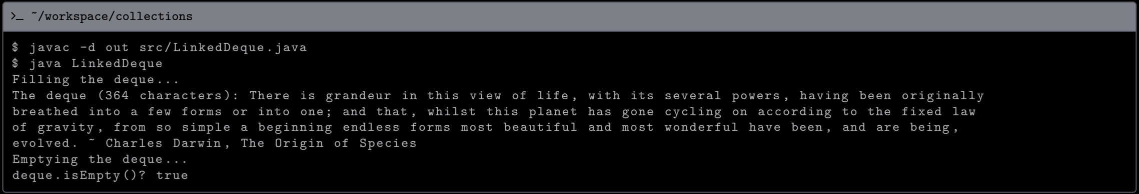
```
static int min(Node first)
    - Set min (int) to Integer.MAX_VALUE
    - For each x (Node) in the linked list first
        - If x.item is less than min, update min to x.item
    - Return min

static int max(Node first)
    - Set max (int) to Integer.MIN_VALUE
    - For each x (Node) in the linked list first
        - If x.item is greater than max, update max to x.item
    - Return max
```

# **Part II (Collections)** · Problem 3 (Deque)

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 |

# Part II (Collections) · Problem 3 (Deque)



```
$ 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
```

## **Part II (Collections)** · Problem 3 (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

$$\texttt{null} \leftarrow \boxed{\text{item}_1} \leftrightarrow \boxed{\text{item}_2} \leftrightarrow \boxed{\text{item}_3} \leftrightarrow \cdots \leftrightarrow \boxed{\text{item}_n} \rightarrow \texttt{null}$$

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

`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

## **Part II (Collections)** · Problem 3 (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

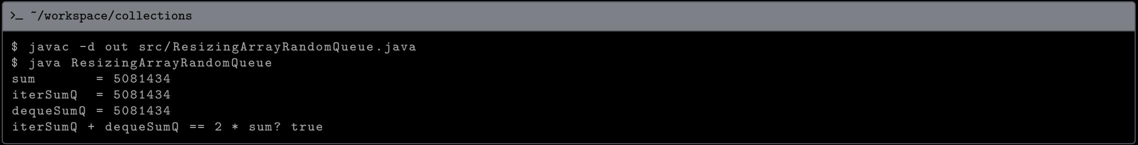
## **Part II (Collections)** · Problem 4 (Random Queue)

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 |

# **Part II (Collections)** · Problem 4 (Random Queue)



>_ ~/workspace/collections

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

## **Part II (Collections)** · Problem 4 (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

## **Part II (Collections)** · Problem 4 (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`

## **Part II (Collections)** · Problem 4 (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