Project 2 (Deques and Randomized Queues) Checklist

Prologue

Project goal: implement generic and iterable data structures, such as double-ended and randomized queues, using arrays and linked lists

Files:

- ⇝ `project2.pdf` (project writeup)
- ⇝ `project2_checklist.pdf` (checklist)
- ⇝ `project2.zip` (starter files for the exercises/problems, and `report.txt` file for the project report)

Exercises

Exercise 1. (*Iterable Binary Strings*) Implement an immutable, iterable data type called `BinaryStrings` to systematically iterate over binary strings of length n. The data type must support the following API:

`BinaryStrings`	
`BinaryStrings(int n)`	constructs an iterable `BinaryStrings` object given the length of binary strings needed
`Iterator<String> iterator()`	returns an iterator to iterate over binary strings of length n

```
>_ ~/workspace/project2

$ java BinaryStrings 3
000
001
010
011
100
101
110
111
```

Exercises

BinaryStrings.java

```
import java.util.Iterator;

import stdlib.StdOut;

// An immutable data type to systematically iterate over binary strings of length n.
public class BinaryStrings implements Iterable<String> {
    private int n; // need all binary strings of length n

    // Constructs a BinaryStrings object given the length of binary strings needed.
    public BinaryStrings(int n) {
        ...
    }

    // Returns an iterator to iterate over binary strings of length n.
    public Iterator<String> iterator() {
        ...
    }

    // Binary strings iterator.
    private class BinaryStringsIterator implements Iterator<String> {
        private int count; // number of binary strings returned so far
        private int p;     // current number in decimal

        // Constructs an iterator.
        public BinaryStringsIterator() {
            ...
        }

        // Returns true if there are anymore binary strings to be iterated, and false otherwise.
        public boolean hasNext() {
            ...
        }

        // Returns the next binary string.
        public String next() {
```

Exercises

BinaryStrings.java

```
            ...
        }

        // Remove is not supported.
        public void remove() {
            throw new UnsupportedOperationException("remove() is not supported");
        }

        // Returns the n-bit binary representation of x.
        private String binary(int x) {
            String s = Integer.toBinaryString(x);
            int padding = n - s.length();
            for (int i = 1; i <= padding; i++) {
                s = "0" + s;
            }
            return s;
        }
    }

    // Unit tests the data type. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (String s : new BinaryStrings(n)) {
            StdOut.println(s);
        }
    }
}
```

Exercises

Exercise 2. (*Iterable Primes*) Implement an immutable, iterable data type called `Primes` to systematically 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/project2

$ java Primes 10
2
3
5
7
11
13
17
19
23
29
```

Exercises

Primes.java

```
import java.util.Iterator;

import stdlib.StdOut;

// An immutable data type to systematically iterate over the first n primes.
public class Primes implements Iterable<Integer> {
    private int n; // need first n primes

    // Constructs a Primes object given the number of primes needed.
    public Primes(int n) {
        ...
    }

    // Returns an iterator to iterate over the first n primes.
    public Iterator<Integer> iterator() {
        ...
    }

    // Primes iterator.
    private class PrimesIterator implements Iterator<Integer> {
        private int count; // number of primes returned so far
        private int p;     // current prime

        // Constructs an iterator.
        public PrimesIterator() {
            ...
        }

        // Returns true if there are anymore primes to be iterated, and false otherwise.
        public boolean hasNext() {
            ...
        }

        // Returns the next prime.
        public Integer next() {
```

Exercises

Primes.java

```
            // Increment count by 1.
            ...

            // As long as p is not prime, increment p by 1.
            ...

            // Return current value of p and increment it by 1.
            ...
        }

        // Remove is not supported.
        public void remove() {
            throw new UnsupportedOperationException("remove() is not supported");
        }

        // Returns true if x is a prime, and false otherwise.
        private boolean isPrime(int x) {
            for (int i = 2; i <= x / i; i++) {
                if (x % i == 0) {
                    return false;
                }
            }
            return true;
        }
    }

    // Unit tests the data type. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i : new Primes(n)) {
            StdOut.println(i);
        }
    }
}
```

Exercises

Exercise 3. (*Min Max*) Implement a library called `MinMax` with static methods `min()` and `max()` that accept a reference `first` to the first node in a linked list of integer-valued items and return the minimum and the maximum values respectively.

```
>_ ~/workspace/project2
```

```
$ java MinMax
min(first) == StdStats.min(items)? true
max(first) == StdStats.max(items)? true
```

Exercises

MinMax.java

```
import stdlib.StdOut;
import stdlib.StdRandom;
import stdlib.StdStats;

public class MinMax {
    // Returns the minimum value in the given linked list.
    public static int min(Node first) {
        // Set min to the largest integer.
        ...

        // Compare each element in linked list with min and if it is smaller, update min.
        ...

        // Return min.
        ...
    }

    // Returns the maximum value in the given linked list.
    public static int max(Node first) {
        // Set max to the smallest integer.
        ...

        // Compare each element in linked list with max and if it is larger, update max.
        ...

        // Return max.
        ...
    }

    // A data type to represent a linked list. Each node in the list stores an integer item and a
    // reference to the next node in the list.
    protected static class Node {
        protected int item;  // the item
        protected Node next; // the next node
    }
```

Exercises

MinMax.java

```
    // Unit tests the library. [DO NOT EDIT]
    public static void main(String[] args) {
        int[] items = new int[1000];
        for (int i = 0; i < 1000; i++) {
            items[i] = StdRandom.uniform(-10000, 10000);
        }
        Node first = null;
        for (int item : items) {
            Node oldfirst = first;
            first = new Node();
            first.item = item;
            first.next = oldfirst;
        }
        StdOut.println("min(first) == StdStats.min(items)? " + (min(first) == StdStats.min(items)));
        StdOut.println("max(first) == StdStats.max(items)? " + (max(first) == StdStats.max(items)));
    }
}
```

Exercises

Exercise 4. (*Text Editor Buffer*) Implement a data type called `Buffer` to represent a buffer in a text editor. The data type must support the following API:

Buffer

`Buffer()`	creates an empty buffer
`void insert(char c)`	inserts `c` at the cursor position
`char delete()`	deletes and returns the character immediately ahead of the cursor
`void left(int k)`	moves the cursor `k` positions to the left
`void right(int k)`	moves the cursor `k` positions to the right
`int size()`	returns the number of characters in this buffer
`String toString()`	returns a string representation of this buffer with the `"|"` character (not part of the buffer) at the cursor position

>_ ~/workspace/project2

```
$ java Buffer
|There is grandeur in this view of life, with its several powers,having been originally breathed by the
Creator 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
```

Hint: Use two stacks `left` and `right` to store the characters to the left and right of the cursor, with the characters on top of the stacks being the ones immediately to its left and right.

Exercises

Buffer.java

```
import dsa.LinkedStack;
import stdlib.StdOut;

// A data type to represent a text editor buffer.
public class Buffer {
    protected LinkedStack<Character> left;  // chars left of cursor
    protected LinkedStack<Character> right; // chars right of cursor

    // Creates an empty buffer.
    public Buffer() {
        ...
    }

    // Inserts c at the cursor position.
    public void insert(char c) {
        ...
    }

    // Deletes and returns the character immediately ahead of the cursor.
    public char delete() {
        ...
    }

    // Moves the cursor k positions to the left.
    public void left(int k) {
        ...
    }

    // Moves the cursor k positions to the right.
    public void right(int k) {
        ...
    }

    // Returns the number of characters in this buffer.
    public int size() {
```

Exercises

Buffer.java

```
    ...
}

// Returns a string representation of the buffer with the "|" character (not part of the buffer)
// at the cursor position.
public String toString() {
    ...

    // Push chars from left into a temporary stack.
    ...

    // Append chars from temporary stack to sb.
    ...

    // Append "|" to sb.
    ...

    // Append chars from right to sb.
    ...

    // Return the string from sb.
    ...
}

// Unit tests the data type (DO NOT EDIT).
public static void main(String[] args) {
    Buffer buf = new Buffer();
    String s = "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";
    for (int i = 0; i < s.length(); i++) {
        buf.insert(s.charAt(i));
    }
```

Exercises

Buffer.java

```
        buf.left(buf.size());
        buf.right(97);
        s = "by the Creator ";
        for (int i = 0; i < s.length(); i++) {
            buf.insert(s.charAt(i));
        }
        buf.right(228);
        buf.delete();
        buf.insert('-');
        buf.insert('-');
        buf.left(342);
        StdOut.println(buf);
    }
}
```

Exercises

Exercise 5. (*Josephus Problem*) In the Josephus problem from antiquity, n people are in dire straits and agree to the following strategy to reduce the population. They arrange themselves in a circle (at positions numbered from 1 to n) and proceed around the circle, eliminating every mth person until only one person is left. Legend has it that Josephus figured out where to sit to avoid being eliminated. Implement a program `Josephus.java` that accepts n (int) and m (int) as command-line arguments, and writes to standard output the order in which people are eliminated (and thus would show Josephus where to sit in the circle).

```
>_ ~/workspace/project2

$ java Josephus 7 2
2
4
6
1
5
3
7
```

Exercises

Josephus.java

```
import dsa.LinkedQueue;
import stdlib.StdOut;

public class Josephus {
    // Entry point.
    public static void main(String[] args) {
        // Accept n (int) and m (int) as command-line arguments.
        ...

        // Create a queue q and enqueue integers 1, 2, ..., n.
        ...

        // Set i to 0. As long as q is not empty: increment i; dequeue an element (say pos); if m
        // divides i, write pos to standard output, otherwise enqueue pos to q.
        ...
    }
}
```

Problems

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Problems

Problem 1. (*Deque*)

Hints:

- 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
  - 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

Problems

- `void addFirst(Item item)`
  - Add the given item to the front of the deque
  - Increment `n` by one
- `void addLast(Item item)`
  - Add the given item to the back of the deque
  - Increment `n` by one
- `Item peekFirst()`
  - Return the item at the front of the deque
- `Item removeFirst()`
  - Remove and return the item at the front of the deque
  - Decrement `n` by one
- `Item peekLast()`
  - Return the item at the back of the deque

Problems

- `Item removeLast()`
  - Remove and return the item at the back of the deque
  - Decrement `n` by one
- `Iterator<Item> 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
  - `Item next()`
    - Return the item in `current` and advance `current` to the next node

Problem 2. (*Sorting Strings*)

Hints:

- ⇝ Create a deque d
- ⇝ For each word w read from standard input
  - ⇝ Add w to the front of d if it is less† than the first word in d
  - ⇝ Add w to the back of d if it is greater† than the last word in d
  - ⇝ Otherwise, remove words that are less than w from the front of d and store them in a temporary stack s; add w to the front of d; and add words from s also to the front of d
- ⇝ Write the words from d to standard output

† Use the helper method `boolean less(String v, String w)` to test if a string `v` is less than a string `w`

Problems

Problem 3. (*Random Queue*)

Hints:

- ⇝ Use a resizing array to implement the API
- ⇝ Instance variables
  - ⇝ 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

Problems

- `void enqueue(Item 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
- `Item sample()`
  - Return `q[r]`, where `r` is a random integer from the interval `[0, n)`
- `Item 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`
  - If `q` is at quarter capacity, resize it to half its current capacity
  - Decrement `n` by one
  - Return `item`
- `Iterator<Item> iterator()`
  - Return an object of type `RandomQueueIterator`

Problems

- `ResizingArrayRandomQueue` :: `RandomQueueIterator()`
  - Instance variables
    - Array to store the items of `q`, `Item[] 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
  - `Item next()`
    - Return the item in `items` at index `current` and advance `current` by one

Problem 4. (*Sampling Integers*)

Hints:

- ⇝ Accept *lo* (int), *hi* (int), *k* (int), and *mode* (String) as command-line arguments
- ⇝ Create a random queue *q* containing integers from the interval $[lo, hi]$
- ⇝ If *mode* is "+" (sampling with replacement), sample and write *k* integers from *q* to standard output
- ⇝ If *mode* is "-" (sampling without replacement), dequeue and write *k* integers from *q* to standard output

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include:

- ⇝ Time (in hours) spent on the project
- ⇝ Difficulty level (1: very easy; 5: very difficult) of the project
- ⇝ A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- ⇝ Acknowledgement of any help you received
- ⇝ Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files:

- ⇝ Make sure your programs meet the style requirements by running the following command on the terminal

```
>_ ~/workspace/project2
$ check_style src/*.java
```

- ⇝ Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and time complexities
- ⇝ Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit:

1. `BinaryStrings.java`
2. `Primes.java`
3. `MinMax.java`
4. `Buffer.java`
5. `Josephus.java`
6. `LinkedDeque.java`
7. `Sort.java`
8. `ResizingArrayRandomQueue.java`
9. `Sample.java`
10. `report.txt`