Jason Boccuti
CISC 213
Lab 3

Pseudocode

Deque add front/rear and poll (remove) front/rear methods

```
function addFront(key)
      newNode = new Node(key)

      if isEmpty()
            front = newNode
            rear = newNode
      else
            newNode.next = front
            front.prev = newNode
            front = newNode
      size++
```

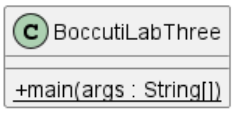
```
function addRear(key)
      newNode = new Node(key)

      if isEmpty()
            front = newNode
            rear = newNode
      else
            newNode.prev = rear
            rear.next = newNode
            rear = newNode
      size++
```

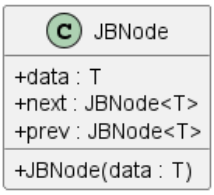
```
function pollFront(key)
      if isEmpty()
            print("Can't poll empty deque")
      else
            front = front.next
            size-
```

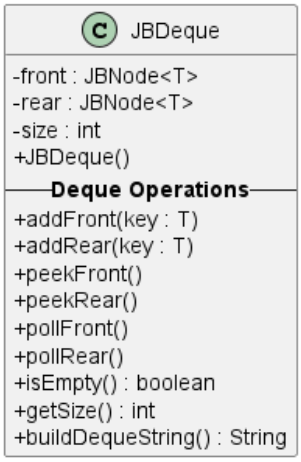
```
function pollRear(key)
      if isEmpty()
            print("Can't poll empty deque")
      else
            rear = rear.prev
            size-
```

Jason Boccuti
CISC 213
Lab 3

UML Class Diagram + Console Output








```
--------------------------------------
[DEQUE size = 0] FRONT --> [] <-- REAR
--------------------------------------
Deque is empty, can't peek front
Deque is empty, can't peek rear
Can't poll front of deque as deque is empty
Can't poll rear of deque as deque is empty
Adding "Apple" to front of deque:
-----------------------------------------
[DEQUE size = 1] FRONT --> [Apple] <-- REAR
-----------------------------------------
Adding "Banana" to rear of deque:
-------------------------------------------------
[DEQUE size = 2] FRONT --> [Apple, Banana] <-- REAR
-------------------------------------------------
Adding "Orange" to front of deque:
---------------------------------------------------------
[DEQUE size = 3] FRONT --> [Orange, Apple, Banana] <-- REAR
---------------------------------------------------------
Adding "Kiwi" to rear of deque:
-----------------------------------------------------------------
[DEQUE size = 4] FRONT --> [Orange, Apple, Banana, Kiwi] <-- REAR
-----------------------------------------------------------------
Peeking front of deque, found node: "Orange"
Peeking rear of deque, found node: "Kiwi"
Polling (removing) front of deque: "Orange"
---------------------------------------------------------
[DEQUE size = 3] FRONT --> [Apple, Banana, Kiwi] <-- REAR
---------------------------------------------------------
Peeking front of deque, found node: "Apple"
Polling (removing) rear of deque: "Kiwi"
-------------------------------------------------
[DEQUE size = 2] FRONT --> [Apple, Banana] <-- REAR
-------------------------------------------------
Peeking rear of deque, found node: "Banana"
Adding "Mango" to front of deque:
---------------------------------------------------------
[DEQUE size = 3] FRONT --> [Mango, Apple, Banana] <-- REAR
---------------------------------------------------------
Adding "Plum" to front of deque:
-----------------------------------------------------------
[DEQUE size = 4] FRONT --> [Plum, Mango, Apple, Banana] <-- REAR
-----------------------------------------------------------
Adding "Blueberries" to rear of deque:
-------------------------------------------------------------------------
[DEQUE size = 5] FRONT --> [Plum, Mango, Apple, Banana, Blueberries] <-- REAR
-------------------------------------------------------------------------
```

Jason Boccuti
CISC 213
Lab 3

Source Code

Source files also uploaded on Canvas

BoccutiLabThree.java

```java
package dev.boccuti.www.cisc213.lab3;

/**
 * Implementation of Lab #3 for CISC 213 that implements a deque using a Doubly
 * Linked List design and avoids using any Java built-in classes for Deque.
 *
 * @author Jason Boccuti | jason@boccuti.dev
 */
public class BoccutiLabThree {

    /**
     * Main method that produces the testing output for the Lab.
     *
     * @param args
     */
    public static void main(String args[]) {
        JBDeque<String> deque = new JBDeque<String>();

        System.out.println(deque.buildDequeString());

        // Test peaking empty deque
        deque.peekFront();
        deque.peekRear();
        // Test polling empty deque
        deque.pollFront();
        deque.pollRear();
        // Add to front and rear
        deque.addFront("Apple");
        deque.addRear("Banana");
        deque.addFront("Orange");
        deque.addRear("Kiwi");
        // Peek front and rear
        deque.peekFront();
        deque.peekRear();
        // Poll front and rear
        deque.pollFront();
```

```
        deque.peekFront();
        deque.pollRear();
        deque.peekRear();
        // Add to front and rear to ensure nothing broke
        deque.addFront("Mango");
        deque.addFront("Plum");
        deque.addRear("Blueberries");
    }
}
```

JBNode.java

```
package dev.boccuti.www.cisc213.lab3;

/**
 * Node implementation for a Doubly Linked List that utilizes generics for type.
 *
 * @author Jason Boccuti | jason@boccuti.dev
 */
public class JBNode<T> {

    public T data;
    public JBNode<T> next;
    public JBNode<T> prev;

    /**
     * Constructor for Node class, initializes the provided data and sets the
     * next/prev pointers to null;
     *
     * @param data
     */
    public JBNode(T data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
```

Jason Boccuti
CISC 213
Lab 3

JBDeque.java

```java
package dev.boccuti.www.cisc213.lab3;

/**
 * Implementation of a Deque in Java using a Doubly Linked List that supports
 * adding/removing/peeking from the front and rear of the queue.
 *
 * @author Jason Boccuti | jason@boccuti.dev
 */
public class JBDeque<T> {

    private JBNode<T> front;
    private JBNode<T> rear;
    private int size;

    /**
     * Constructor of the Deque that sets the front/rear to null and size to 0.
     */
    public JBDeque() {
        this.front = null;
        this.rear = null;
        this.size = 0;
    }

    /**
     * Add a node to the front of the deque.
     *
     * @param key data for the node
     */
    public void addFront(T key) {
        JBNode<T> newNode = new JBNode<T>(key);

        // If the deque is empty, the new node becomes the front + rear
        if (this.isEmpty()) {
            this.front = newNode;
            this.rear = newNode;
        }
        // Else, the new node's next is set to the current front and the current front's
        // prev is set to the new node, then the front is officially set to the new node
        else {
```

Jason Boccuti
CISC 213
Lab 3

```
            newNode.next = this.front;
            this.front.prev = newNode;
            this.front = newNode;
        }
        // Increase the size of the deque
        this.size++;

        System.out.println("Adding \"" + key + "\" to front of deque:\n" +
this.buildDequeString());
    }

    /**
     * Add a node to the rear of the deque
     *
     * @param key data for the node
     */
    public void addRear(T key) {
        JBNode<T> newNode = new JBNode<T>(key);

        // If the deque is empty, the new node becomes the front + rear
        if (this.isEmpty()) {
            this.front = newNode;
            this.rear = newNode;
        }
        // Else, the new node's prev is set to the current rear and the current rear's
        // next is set to the new node, then the rear is officially set to the new node
        else {
            newNode.prev = this.rear;
            this.rear.next = newNode;
            this.rear = newNode;
        }
        // Increase the size of the deque
        this.size++;

        System.out.println("Adding \"" + key + "\" to rear of deque:\n" +
this.buildDequeString());
    }

    /**
```

Jason Boccuti
CISC 213
Lab 3

```
     * Peek the data in the front node of the deque.
     */
    public void peekFront() {
        // If the deque is empty, can't peek
        if (this.isEmpty()) {
            System.out.println("Deque is empty, can't peek front");
        }
        // Else, peek the front node
        else {
            System.out.println("Peeking front of deque, found node: \"" + this.front.data +
"\"");
        }
    }

    /**
     * Peek the data in the rear node of the deque.
     */
    public void peekRear() {
        // If the deque is empty, can't peek
        if (this.isEmpty()) {
            System.out.println("Deque is empty, can't peek rear");
        }
        // Else, peek the rear node
        else {
            System.out.println("Peeking rear of deque, found node: \"" + this.rear.data +
"\"");
        }
    }

    /**
     * Poll (remove) the front node of the deque.
     */
    public void pollFront() {
        // If the deque is empty, can't poll
        if (this.isEmpty()) {
            System.out.println("Can't poll front of deque as deque is empty");
        }
        // Else, poll (remove) the front node by setting the front pointer to the
        // current front node's next
```

Jason Boccuti
CISC 213
Lab 3

```
        else {
            System.out.println("Polling (removing) front of deque: \"" + this.front.data +
"\"");
            // Remove Front
            this.front = this.front.next;
            // Decrease the size of the deque
            this.size--;
            System.out.println(this.buildDequeString());
        }
    }

    /**
     * Poll (remove) the rear node of the deque.
     */
    public void pollRear() {
        // If the deque is empty, can't poll
        if (this.isEmpty()) {
            System.out.println("Can't poll rear of deque as deque is empty");
        }
        // Else, poll (remove) the rear node by setting the rear pointer to the current
        // rear node's prev
        else {
            System.out.println("Polling (removing) rear of deque: \"" + this.rear.data +
"\"");
            // Remove rear
            this.rear = this.rear.prev;
            // Decrease the size of the deque
            this.size--;
            System.out.println(this.buildDequeString());
        }
    }

    /**
     * Method to check if the deque is currently empty;
     *
     * @return a boolean of whether the deque is empty or not
     */
    public boolean isEmpty() {
        if (size == 0) {
```

Jason Boccuti
CISC 213
Lab 3

```
            return true;
        }
        return false;
    }

    /**
     * Method to check the size of the deque.
     *
     * @return the deque's size
     */
    public int getSize() {
        return this.size;
    }

    /**
     * Build a string representation of the deque to show the front, rear and the
     * order of all the nodes.
     *
     * @return the string to be displayed
     */
    public String buildDequeString() {
        String dequeStr = "";
        JBNode<T> currNode = this.front;

        dequeStr += "[DEQUE size = " + this.size + "] FRONT --> [";

        for (int i = 0; i < this.size; i++) {

            if (i != 0 && i != this.size) {
                dequeStr += ", ";
            }
            dequeStr += currNode.data;
            currNode = currNode.next;
        }

        dequeStr += "] <-- REAR";

        String spacer = "-".repeat(dequeStr.length());
```

```
        dequeStr = spacer + "\n" + dequeStr + "\n" + spacer;

        return dequeStr;
    }
}
```