Assignment 6, Jeffrey Wan Module 6

1. A deque (pronounced deck) is an ordered set of items from which items may be delete at either end and into which items maybe inserted at either end. Call the two ends left and right. This is an access restricted structure since no insertions or deletions can happen other than at the ends. Implement the deque as a doubly-linked list (not circular, no header). Write insertLeft and DeleteRight.

So with any node you can get to the node to the left and right of it. No header node. There's a next and previous with each node. There's a tail node and that points to the head and a next. Head has a next and a previous that points to the tail.

```
class public class Dequeue {
        private class Node {
                String data;
                Node next;
                Node previous;
        }

        Node head
        Node tail

        public insertLeft (Node newNode){
                next = head.next
                next.previous = newNode
                newNode.next = head
                newNode.previous = tail
                tail.next = newNode
                head = newNode
        }

        public deleteRight (Node newNode){
                previous = tail.previous
                previous.next = head
                head.previous = previous
                tail.next = null
                tail.previous = null
        }
}
```

2. Implement a deque from problem 1 as a doubly-linked circular list with a header. Write InsertRight and DeleteLeft.

So the tail points to the header and a previous node, and the header points to a next node which is head and a previous node which is the tail. It wraps around in a doubly-linked circular list with a header.

```
public class  Deque2
    # this node has a previous and a next pointer so it’s doubly-linked
    class Node
        public String data
        public Node previous
        public Node next


    private class Header
        public Node head
        public Node tail
        public int size

    private Header header;

    # constructor
    public Deque2() {
        header = new Header();
        header.head = null
        header.tail = null
        header.size = 0
    }

    public void insertRight(String data)
        Node newNode = new Node()
        newNode.data = data
        if (header.head == null)
            newNode.previous = newNode
            newNode.next = newNode
            header.head = newNode
            header.tail = newNode
            header.size = 1
        } else {
            newNode.previous = header.tail
            newNode.next = header.head
            header.tail = newNode
            header.size++
        }
    }

    public Node deleteLeft()
        if (header.head == null)
            return null

        else
            nodeToReturn = header.head;
            header.head.previous = null
            # connect tail with the head’s next node
            header.tail.next = header.head.next
            header.head.next = null
            header.head = header.head.next
            header.head.previous = header.tail
            header.size--
            return nodeToReturn
```

3. I did my best and tried rewatching the videos but ultimately was a tad confused. Show me a good solution to this? Are there ever full solutions given to homeworks? I see the rubric but I could use more fleshed out solutions. Also where are quiz answers?

Write a set of routines for implementing several stacks and queues within a single array. Hint: Look at the lecture material on the hybrid implementation.

stacks and queues in single array.two stacks and two queues.

```
public class HybridArray(int size) {
    public int stack1top = null
    public int stack2top = null
    public int queue1tail = null
    public int queue1tail = null
    public int queue2head = null
    public int queue2head = null
    # 2 because each space holds the data and the position in the array
    public freeSpace = new Array[size][2]

    # set everything to null to denote
    HybridArray {
        for (i = 0, i <= size, i++) {
            for elem in freeSpace {
                # index 0 is the data, index 1 is the index in the subarray.
                elem = [null, i]
            }
        }
    }

    public Array getFirstFreeSpace() {
        space = null
        # get first null space
        for elem in freeSpace {
            if elem[0] = null;
                space = elem
                break;
        }
        return space
    }

    public stack1push(elem) {
        space = getFirstFreeSpace()

        if stack1top = null {
            # add elem to first free space, keep the free space index
            space[0] = elem
            # set stack1top to index
            stack1top = elem[1]
        } else {
            # get element at top of stack
            top = freeSpace[stack1top]
            space[0] = elem
            # point current top to new top
            top[1] = space[1]
        }
    }

    stack1pop(elem) {
        # this should be the top of the stack
        elem = freeSpace[stack1top]
        item = elem[0].copy()
        # set data to null to return back to free space
        elem[0] = null

        return item
    }

    #stack2 is the same

    # add to tail
    queue1add {
        space = getFirstFreeSpace()

        if stack1top = null {
            # add elem to first free space, keep the free space index
            space[0] = elem
            # set queue1tail to index
            queue1tail = elem[1]
            queue1head = elem[1]
        } else {
            # get element at tail of queue
            tail = freeSpace[queue1tail]
            space[0] = elem
            # point current tail to new tail
            tail[1] = space[1]
        }
    }


    # remove head
    queue1remove {
        item = freeSpace[queue1head]
        elem = item[0].copy()
        #set data to null to return back to free space
        elem[0] = null
        return elem
    }

}
```