Assignment 1

Jordan Murray

Due: February 15, 2019

1 Stack

The StackMurray class, implements a Stack data structure using the NodeMurray Linked List.

```
Line 9: private NodeMurray myHead;
```

This class has a null constructor which sets the Head value to null

```
Line 13: myHead = null;
```

The next method of this class is called isEmpty(). This checks if there are any values in the current stack. It will return true if the head of the stack is equal to null. Otherwise, the stack is not empty.

```
Line 17: public boolean isEmpty(){
Line 18:    boolean ans = false;
Line 19:      if(myHead == null)
Line 20:         ans = true;
Line 21:
Line 22:    return ans;
Line 23: }
```

The next method of this class is called peek(). This method is used to get the data of the head of the stack. It checks if the stack is not empty and if it is not, it will get the data of the head and if it is empty, it will return a space char.

```
Line 26: public char peek(){
```

```
Line 27:    if(!isEmpty()){
Line 28:       return myHead.getData();
Line 29:    }else{
Line 30:        return ' ';
Line 31: }
```

The next method is traverse(). This method has one variable of the type NodeMurray and sets the value equal to myHead which is the top of the stack. Then it loops through all of the nodes in the stack without popping them, gets all of the values of every node, and prints them out. This method was very helpful when debugging my program, because it printed the whole stack, so I could see what values were on it.

```
Line 35: public void traverse(){
Line 36:    NodeMurray currentNode = myHead;
Line 37:    while(currentNode !=null){
Line 38:        System.out.print(currentNode.getData());
Line 39:       currentNode = currentNode.getNext();
Line 40:    }
Line 41:    System.out.println();
Line 42: }
```

The next method of the StackMurray class is push(). This method's job is to add a new node onto a stack. It has one parameter which is a char value. This char value becomes the data of the new node. Once the data has been set, the new node's "next" value gets set to the previous head of the stack, and then the new node becomes the new head of the stack.

```
Line 45: public void push(char value){
Line 46:    NodeMurray newGuy = new NodeMurray();
Line 47:    newGuy.setData(value);
Line 48:    newGuy.setNext(myHead);
Line 49:    myHead = newGuy;
Line 50: }
```

The last method of this class is called pop(). This method is used to remove a node from the top of the stack. First it checks if the stack is empty, so it doesn't try to remove something that isn't there. If it is not empty, then it stores the data in a variable, and sets the head to the current heads "next" value. It then returns the variable ans which holds the data of the node that was just removed from the stack.

```
Line 54: public char pop(){
Line 55:    char ans = 0;
Line 56:    if(!isEmpty()){
Line 57:       ans = myHead.getData();
Line 58:        myHead = myHead.getNext();
Line 59:     }
Line 60:    return ans;
Line 61: }
```

2 Queue

The QueueMurray class, implements the queue data structure also using the NodeMurray Linked List.

Line 9: private NodeMurray myHead; Line 10: private NodeMurray myTail;

This class has a null constructor which sets the Head and Tail value to null

```
Line 14: myHead = null;
Line 15: myTail = null;
```

The next method of this class is called isEmpty(). This checks if there are any values in the current queue. It will return true if the tail of the stack is equal to null. Otherwise, the queue is not empty.

```
Line 19: public boolean isEmpty(){
Line 20:    boolean ans = false;
Line 21:    if(myTail == null)
Line 22:        ans = true;
Line 23:
Line 24:    return ans;
Line 25: }
```

The next method of this class is called peek(). This method is used to get the data of the head of the queue. It checks if the queue is not empty and if it is not, it will get the data of the head and if it is empty, it will return a space char.

```
public char peek(){
  if(!isEmpty()){
    return myHead.getData();
  }else{
     return ' ';
}
```

The next method is traverse(). This method has one variable of the type NodeMurray and sets the value equal to myHead which is the top of the stack. Then it loops through all of the nodes in the stack without popping them, gets all of the values of every node, and prints them out. This method was very helpful when debugging my program, because it printed the whole stack, so I could see what values were on it.

```
public void traverse(){
  NodeMurray currentNode = myHead;
  while(currentNode !=null){
    System.out.print(currentNode.getData());
    currentNode = currentNode.getNext();
  }
  System.out.println();
}
```

The next method of this class is enqueue. This is used to add a new node to the end of the queue. It takes in one parameter of the type char, and sets that value to the newly created node. If the queue is empty, then both the head and tail point to the new node because it will be the only value in the queue. If it is not empty, then the current tail's "next value gets set to the new node, and then the new node becomes the new tail or end of the queue.

```
public void enqueue(char value){
  NodeMurray newGuy = new NodeMurray();
  newGuy.setData(value);
  if(isEmpty()){
    myHead = newGuy;
```

```
        myTail = newGuy;
    }else{
        myTail.setNext(newGuy);
        myTail = newGuy;
    }
}
```

The last method in the QueueMurray Class is called dequeue. This method will take the head of the queue, and store the data in a variable. It then sets the head to the current head's "next" value. There is one condition where if the head and tail of the queue are equal, meaning there is only one item in the queue, then the tail should be set to null. It then returns the answer variable which holds the data of the node that was just removed from the queue.

```
public char dequeue(){
    char ans = myHead.getData();
    myHead = myHead.getNext();
    if(myHead == myTail){
        myTail = null;
    }
    return ans;
}
```