Assignment One – LaTeX Palindrome Paradise

Rocco Piccirillo
Rocco.Piccirillo1@Marist.edu

February 13, 2019

1 Stack

```
public class Stack
{

Node head;

  public void push(char data)
{
Node node = new Node();
node.data = data;
node.next = null;
node.next = head;
head = node;
}


public Node pop()
{
Node node;
     node = head;
head = head.next;
return node;
}
23
24
25 public void display()
26 {
27 Node node = head;
28 while(node.next != null)
29 {
30 System.out.println(node.data);
31 node = node.next;
32 }
33 System.out.print(node.data);
34 }
35
36 public Node peek()
37 {
38 Node node;
39 node = head;
40 System.out.print(node.data);
41 return head;
42 }
43 }
```

This right here is my stack class. The first code to mention is my instance of Node head; on line 4. That is so that I can reference my node class and have a head for the stack. Next up is my push method from line 6 to line 13. In this, whenever I added to the top of the stack, I created a new node. Whatever data the user passes in gets stored int through node.data = data;. I am changing the head location to a new object, and it will also be taking the original heads address. Node.next = head; means that I am giving the address of the prior address so it knows where to travel next. After this is my pop method, looking at line 16 to line 22 next. Pop is pretty simple but we are basically saying that the head node is being switched to the next node. So that the address of the first node is now the second nodes. This does leave the original node lying around though, but it doesn't affect run time all that much luckily. Shortly after is my show method or my print method from lines 25 to 34. We have to travel between each and every node through a while loop. While the next value is not empty, I printed the value of the node. Though, for the last node in the stack, there isn't a node after, so we have to print it manually outside of the loop. And after this, we shift to the next node until it is empty. The last method is the peek method from 36 to 42. This is the exact same as delete but instead of changing the old heads address, we are printing out the current head.

2 Queue

```
1    public class Queue {
2
```

```java
Node head;

public void enQueue(char data)
{

Node node = new Node();
node.data  = data;
node.next = null;

if(head == null)
{
head = node;

} else
{
Node n = head;
while(n.next != null)
{
n = n.next;
}
n.next = node;
}

}

public Node deQueue()
{
Node node;
node = head;
head = head.next;
return node;
}

public void show()
{
Node node = head;

while(node.next != null)
{
System.out.println(node.data);
node = node.next;
}
System.out.print(node.data);
}
```

```
49 public Node peek()
50 {
51 Node node;
52 node = head;
53 System.out.print(node.data);
54 return head;
55 }
56 }
```

The other important class is of course my Queue. Once again we are creating a node for the head on line 3. On line 5 to 26, we have our first method known as enQueue. This is sort of like append. First we are creating a new node and if there is no current head this node gets added as the head. After this we are adding in another node and adding it behind the original node we added. The first node gets the address of the second node so it knows where to travel next. Then from 28 to 34 we have deQueue. This is removing our head node, basically the same exact as pop where we change the address of the head node and leave it in the middle of space not being used by anything. Then from 36 to 46 we have show again, which keeps printing nodes until the address of the node is null. Though the last node has a null address so we print that one separately. Lastly we have peek which just prints the data from the head node instead of changing it to the second node. This is almost like deletion once again but we are printing. The value.