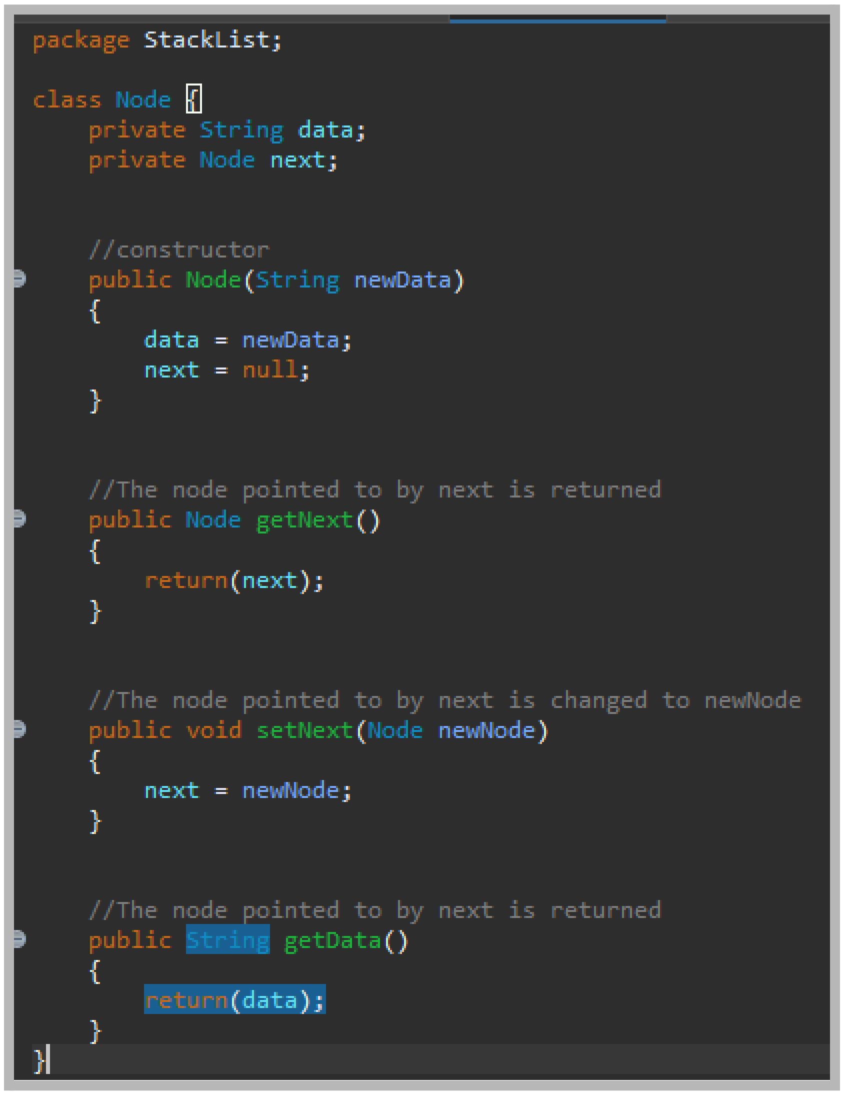
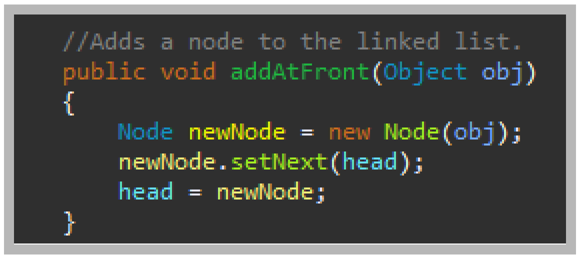
Credit Name: Advanced Algorithms and Data Structures
Assignment Name: StackList

How has your program changed from planning to coding to now? Please explain?



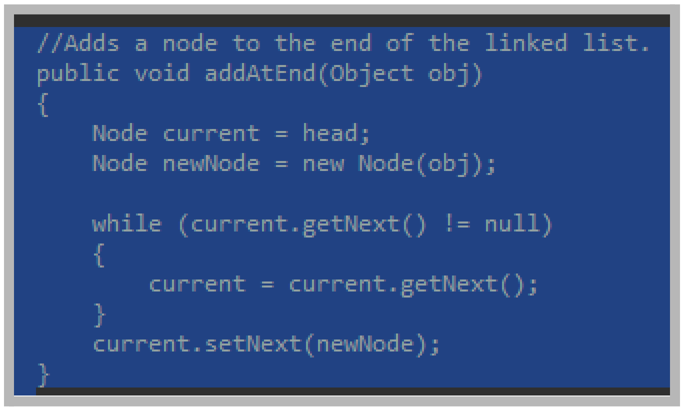
```java
package StackList;

class Node {
    private String data;
    private Node next;


    //constructor
    public Node(String newData)
    {
        data = newData;
        next = null;
    }


    //The node pointed to by next is returned
    public Node getNext()
    {
        return(next);
    }


    //The node pointed to by next is changed to newNode
    public void setNext(Node newNode)
    {
        next = newNode;
    }


    //The node pointed to by next is returned
    public String getData()
    {
        return(data);
    }
}
```

I started the StackList mastery by creating a Node class based on the LinkedList skillbuilder as it is something that I would need when implementing a linked list in the StackList class.



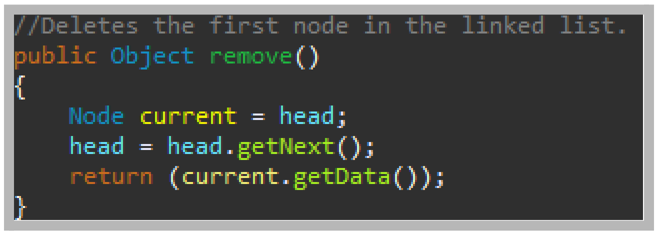
```
//Adds a node to the linked list.
public void addAtFront(Object obj)
{
    Node newNode = new Node(obj);
    newNode.setNext(head);
    head = newNode;
}
```

Next I started working on modifying the Linked List from the skillbuilder to only include what I need and add additional methods that are needed for a stack. But the first thing I modified was changing the item stored from String to Object and worked from there.



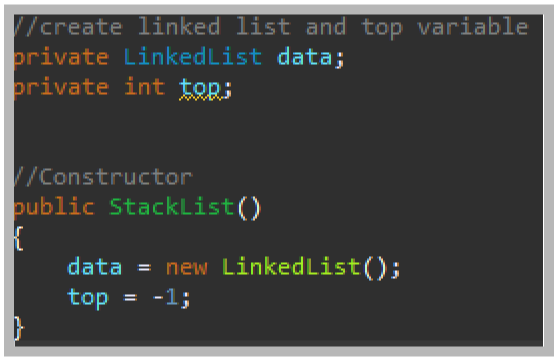
```
//Adds a node to the end of the linked list.
public void addAtEnd(Object obj)
{
    Node current = head;
    Node newNode = new Node(obj);

    while (current.getNext() != null)
    {
        current = current.getNext();
    }
    current.setNext(newNode);
}
```

I removed the addAtEnd method because stacks don’t allow you to do that.



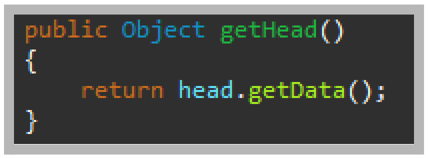
```
//Deletes the first node in the linked list.
public Object remove()
{
    Node current = head;
    head = head.getNext();
    return (current.getData());
}
```

Then I edited the remove method to move the top item in the list to the next one rather than search for an item, and I moved on to the StackList class.

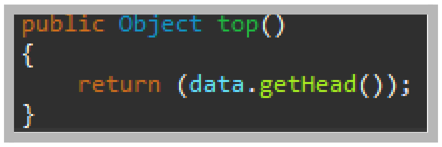


```
//create linked list and top variable
private LinkedList data;
private int top;


//Constructor
public StackList()
{
    data = new LinkedList();
    top = -1;
}
```

When starting on the StackList class I copied the code from Stack2 and started to modify it to include the LinkedList object and edited the constructor.

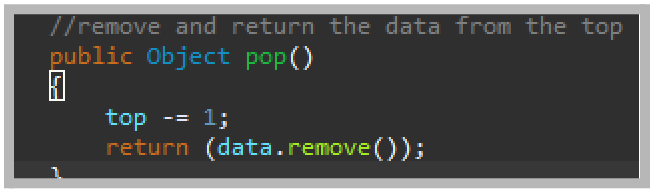


```
public Object getHead()
{
    return head.getData();
}
```

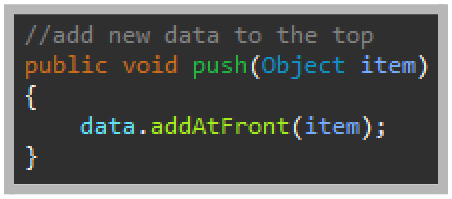


```
public Object top()
{
    return (data.getHead());
}
```

When I moved onto the top method I realized that I didn't have any way of isolating the top item in the linked list. I decided to go back to the LinkedList class to add a method to get the head node and used that for the top method in StackList.

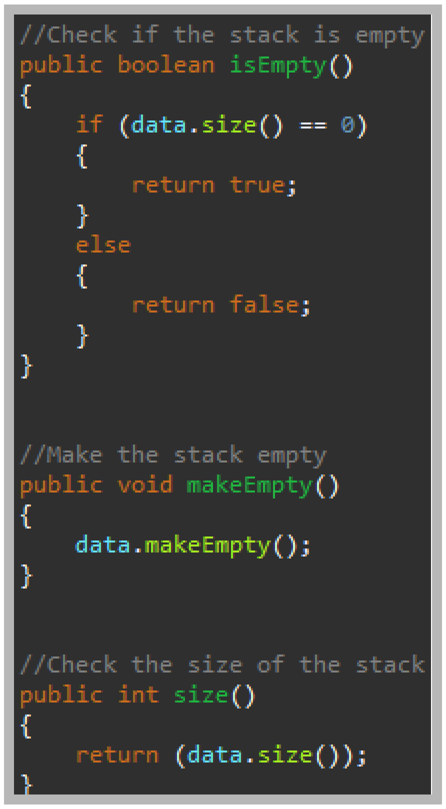


```
//remove and return the data from the top
public Object pop()
{
    top -= 1;
    return (data.remove());
```



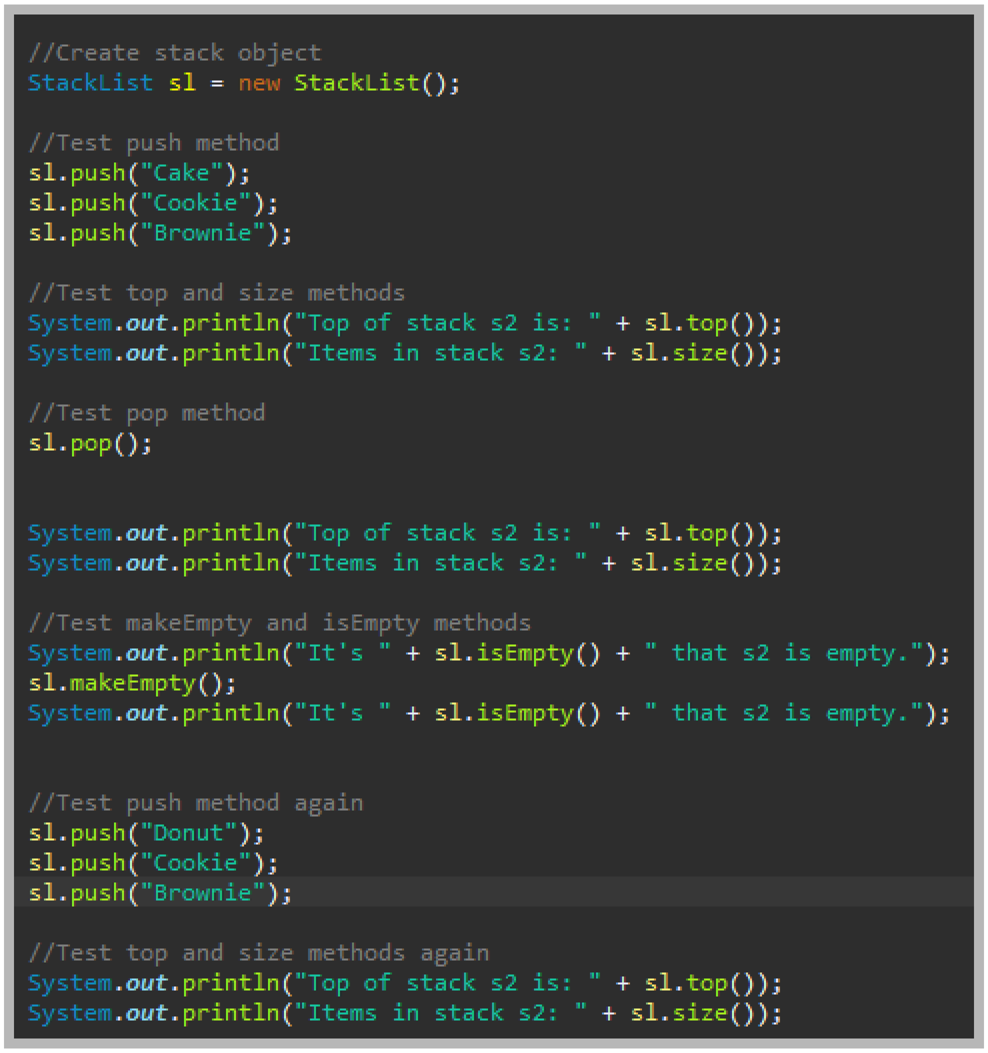
```
//add new data to the top
public void push(Object item)
{
    data.addAtFront(item);
}
```

Then I simply added the method for removing the top to pop and the addAtFront method to push.



```
//Check if the stack is empty
public boolean isEmpty()
{
    if (data.size() == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}


//Make the stack empty
public void makeEmpty()
{
    data.makeEmpty();
}


//Check the size of the stack
public int size()
{
    return (data.size());
}
```

I also got rid of the top variable because the way it was coded, it was no longer necessary. I also changed the isEmpty method to check for size instead of for top, changed make empty to use the other makeEmpty method, size to use the size method from LinkedList.



```java
//Create stack object
StackList sl = new StackList();

//Test push method
sl.push("Cake");
sl.push("Cookie");
sl.push("Brownie");

//Test top and size methods
System.out.println("Top of stack s2 is: " + sl.top());
System.out.println("Items in stack s2: " + sl.size());

//Test pop method
sl.pop();


System.out.println("Top of stack s2 is: " + sl.top());
System.out.println("Items in stack s2: " + sl.size());

//Test makeEmpty and isEmpty methods
System.out.println("It's " + sl.isEmpty() + " that s2 is empty.");
sl.makeEmpty();
System.out.println("It's " + sl.isEmpty() + " that s2 is empty.");


//Test push method again
sl.push("Donut");
sl.push("Cookie");
sl.push("Brownie");

//Test top and size methods again
System.out.println("Top of stack s2 is: " + sl.top());
System.out.println("Items in stack s2: " + sl.size());
```

Finally I copied the tester code from the Stack2Tester and added to it so that it also tests for pushing after the list was emptied.