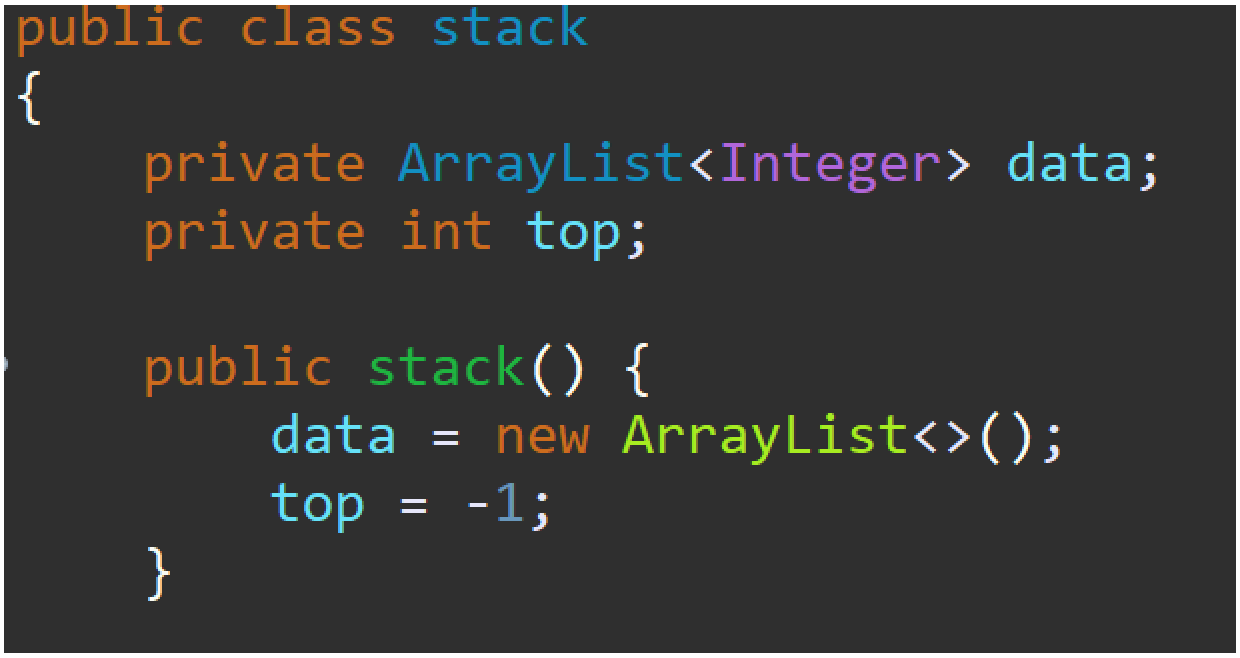
Credit Name: CSE 3020 Computer Science 4

Assignment: ParenChecker

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

There are 2 classes for this application: 1 defines our stack with a dynamic size. The other is the main program which checks that every open parenthesis has a corresponding closed parenthesis.

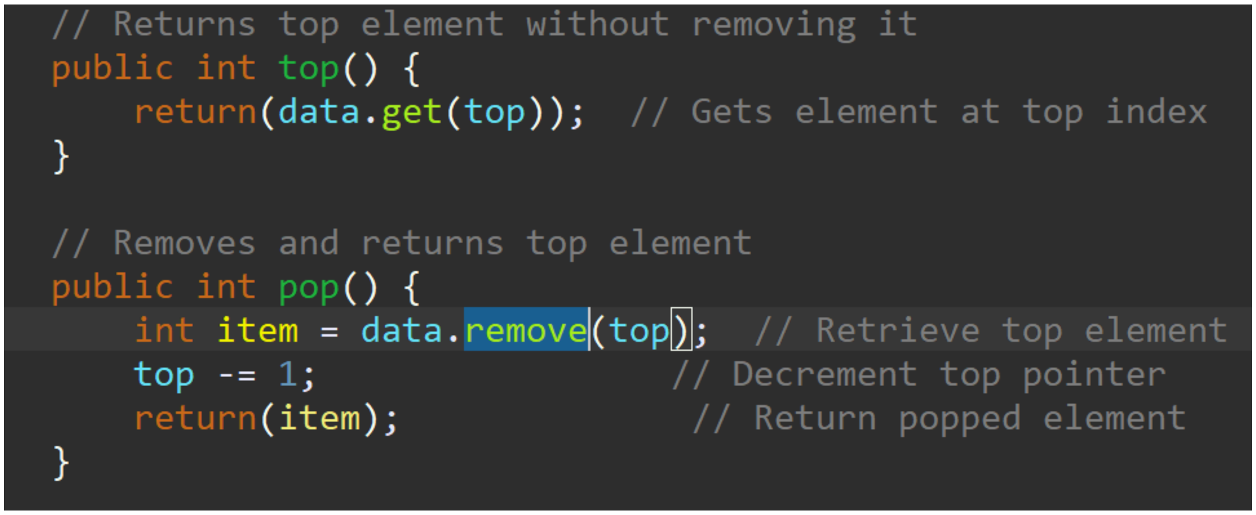
First, I designed the stack class.



```
public class stack
{
    private ArrayList<Integer> data;
    private int top;

    public stack() {
        data = new ArrayList<>();
        top = -1;
    }
```

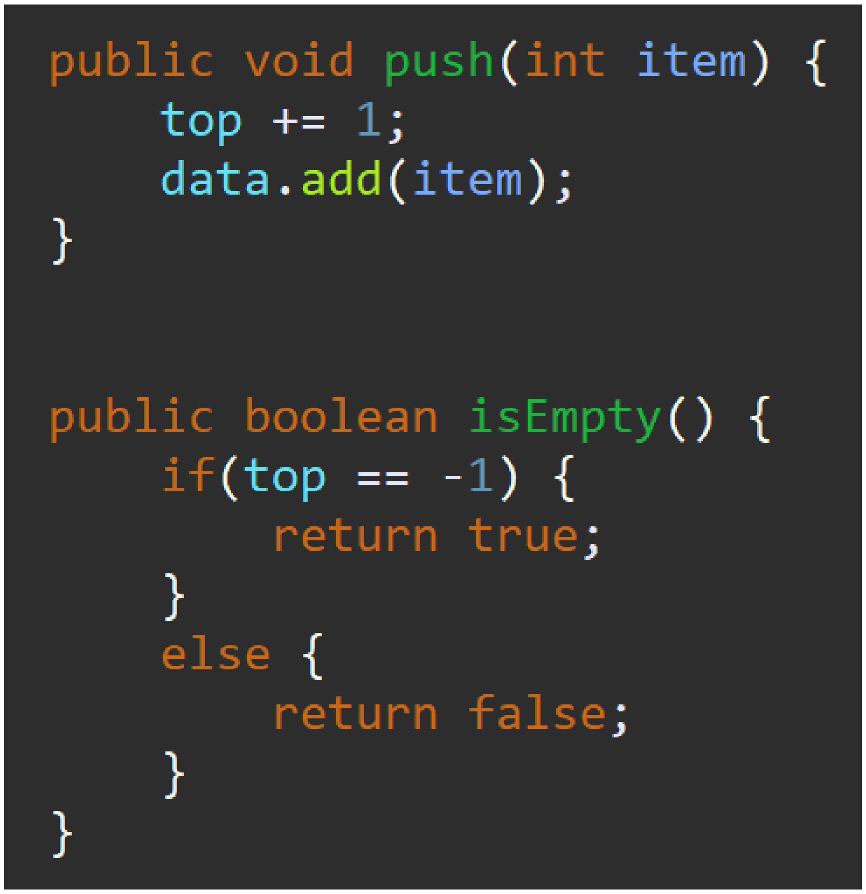
First i declare an ArrayList as the location where we will store data because it does not require any static and predefined item limits. I also declare a variable for the top value in the stack. It is an integer because our stack will contain the indexes of all open brackets in the expression.

In the constructor for stack, I initialized the ArrayList and set the value of the top value to be -1 when there are no entries in the stack.



```java
// Returns top element without removing it
public int top() {
    return(data.get(top));  // Gets element at top index
}

// Removes and returns top element
public int pop() {
    int item = data.remove(top);  // Retrieve top element
    top -= 1;                    // Decrement top pointer
    return(item);                 // Return popped element
}
```

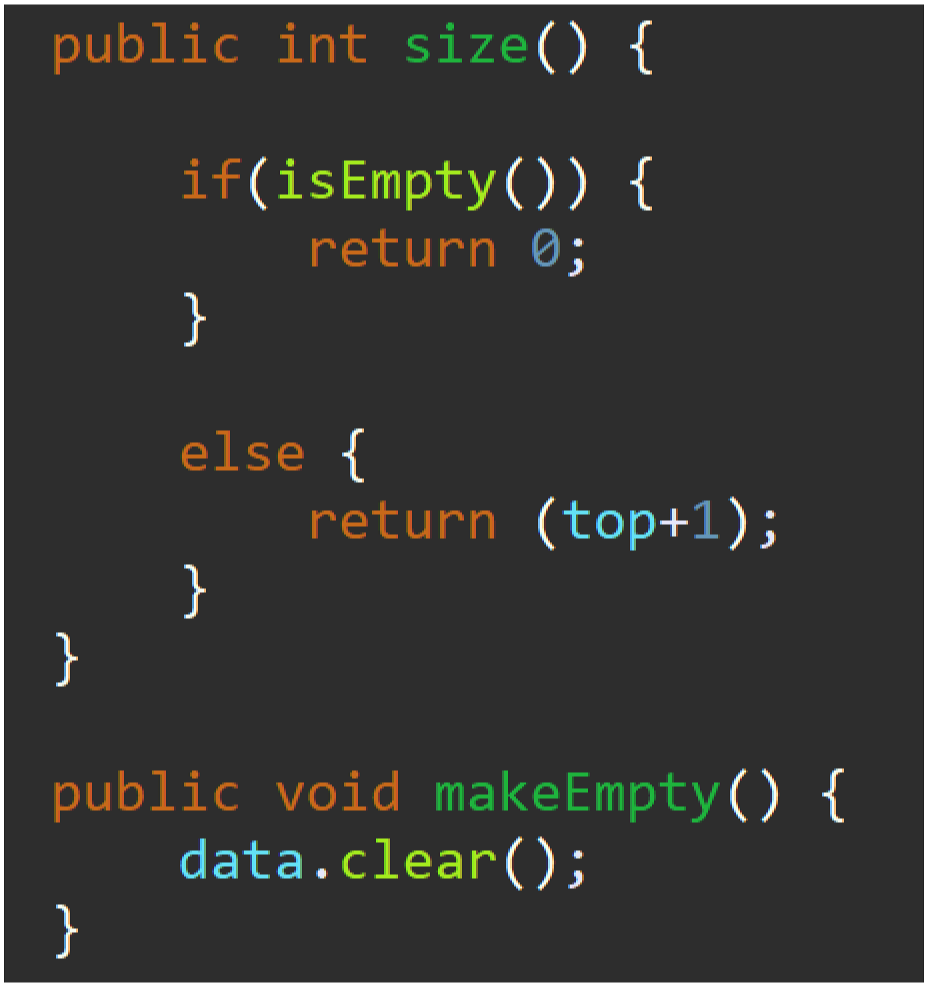
Our first 2 methods simply return the value of the top item and the second removes the item from the stack and returns the item that was removed.



```java
public void push(int item) {
    top += 1;
    data.add(item);
}


public boolean isEmpty() {
    if(top == -1) {
        return true;
    }
    else {
        return false;
    }
}
```

The push method adds a given value to the top of the stack. To do this we increment the top index value by 1 and add the item to the beginning of the ArrayList.

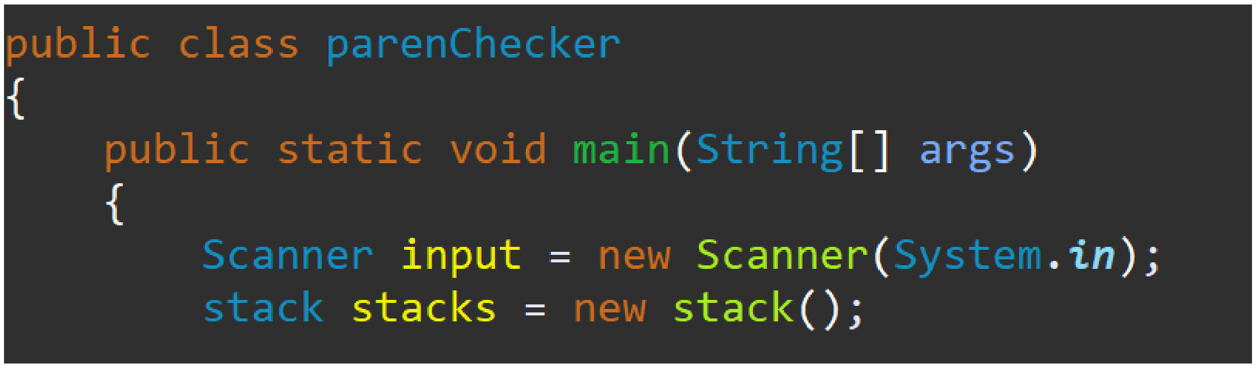
The isEmpty() method returns true if the top index value is -1 (this is the indicator we defined to highlight an empty stack) otherwise it returns false.



```
public int size() {

    if(isEmpty()) {
        return 0;
    }

    else {
        return (top+1);
    }
}

public void makeEmpty() {
    data.clear();
}
```

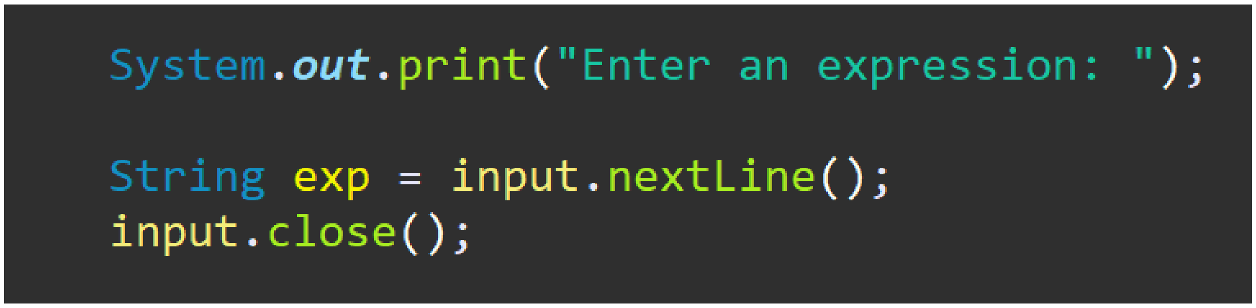
The last 2 methods are:
a method that returns the size of the stack. First we check if the stack is empty. If it is we return 0. Otherwise we return the value of the top index but we add 1 to account for the index starting from 0.
The last method simply clears the data in the list so it makes it empty.

The next class is the parenthesis checker.



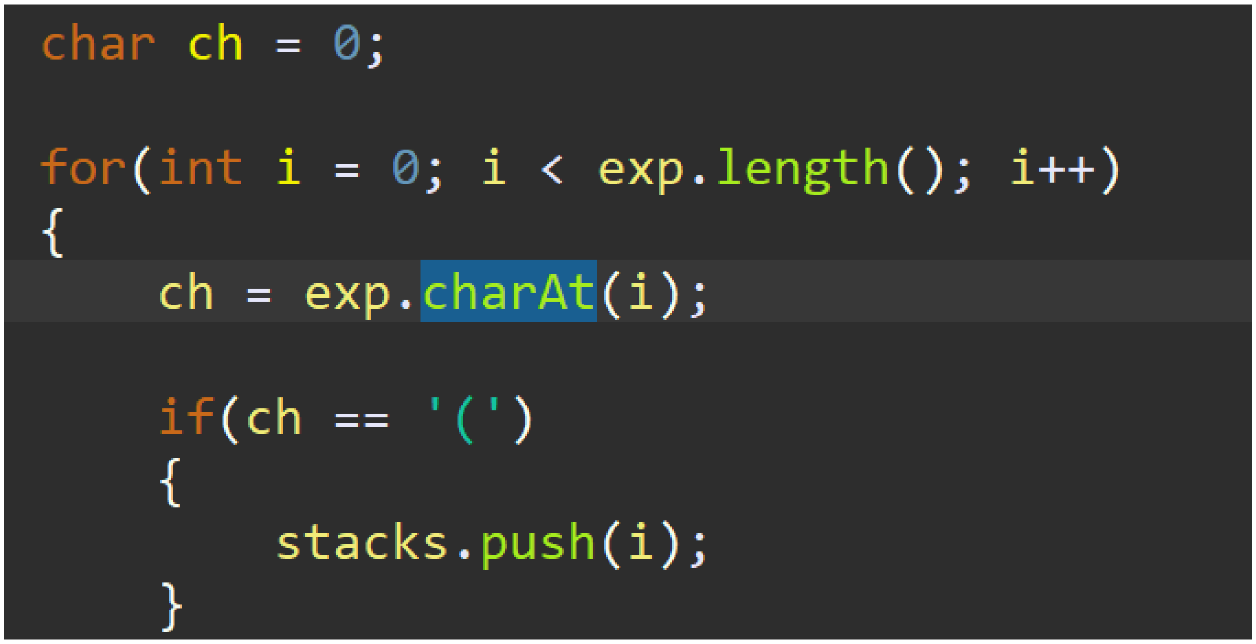
```
public class parenChecker
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        stack stacks = new stack();
```

We define the main method first. Then we initialize the scanner to receive user inputs for the expression. Then we also declare and initialize our stack.



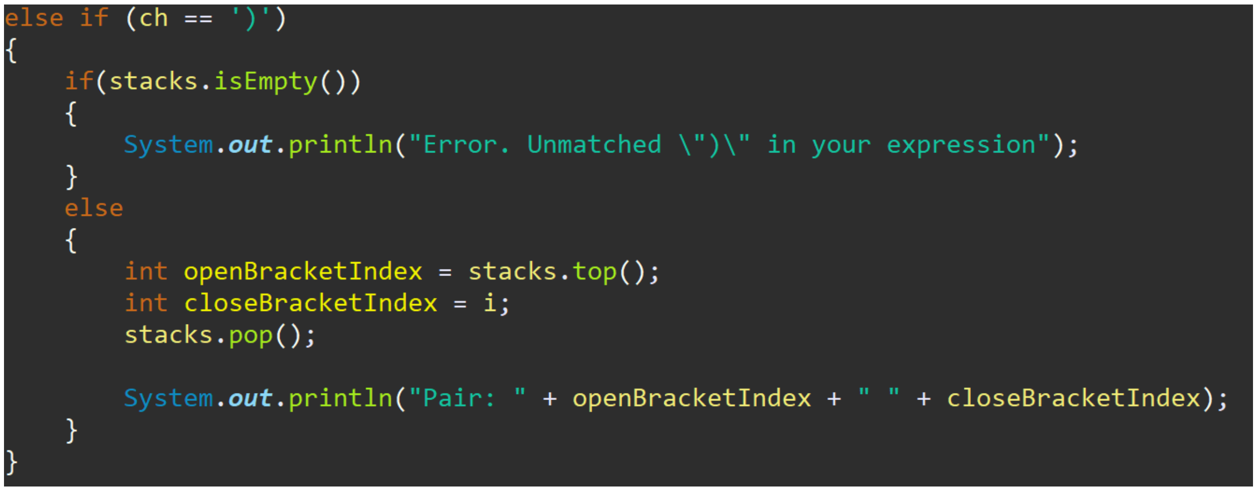
```java
    System.out.print("Enter an expression: ");

    String exp = input.nextLine();
    input.close();
```

We then ask the user to enter an expression and then we save their response which we will check.



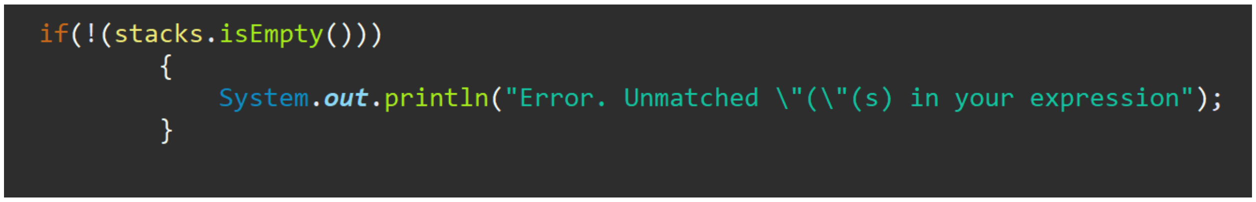
```java
 char ch = 0;

 for(int i = 0; i < exp.length(); i++)
 {
     ch = exp.charAt(i);

     if(ch == '(')
     {
         stacks.push(i);
     }
```

Then we declare and initialize a char variable to 0. This is because we will iterate through every character in the expression given by the user. If we find that one of the characters is an open parenthesis then we push its index to our stack.



```java
else if (ch == ')')
{
    if(stacks.isEmpty())
    {
        System.out.println("Error. Unmatched \")\" in your expression");
    }
    else
    {
        int openBracketIndex = stacks.top();
        int closeBracketIndex = i;
        stacks.pop();

        System.out.println("Pair: " + openBracketIndex + " " + closeBracketIndex);
    }
}
```

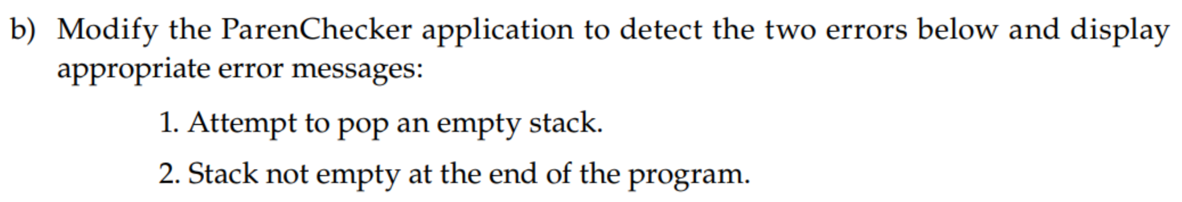
Otherwise if we find a closed bracket instead, we need to refer to the stack. We first check if the stack is empty(this is the first condition of part b for this mastery). If so, that means theres no corresponding open bracket for the closed one. Therefore we show a message that there is an unmatched ')' in the users expression. Otherwise if the stack is not empty. We get the index of the open bracket by getting the value of the top index, we get the closeBracketIndex which is simply the i-value where our else condition was satisfied. Then we remove the top value in the stack because we have already found it to be in a matching pair and then we display that pair below. The reason this works is because mathematically, the first bracket to be opened is the last to be closed or in other words, the last bracket to be opened is the first bracket to be closed.



```java
  if(!(stacks.isEmpty()))
          {
              System.out.println("Error. Unmatched \"(\"(s) in your expression");
          }
```

Lastly, we check if the stack is empty after we have iterated through the entire expression. If so we return a statement that says we have too many unmatched open brackets without closed expression. (this is the second part of part b for this mastery)

Therefore this mastery satisfies the required conditions:



b) Modify the ParenChecker application to detect the two errors below and display appropriate error messages:

1. Attempt to pop an empty stack.
2. Stack not empty at the end of the program.