

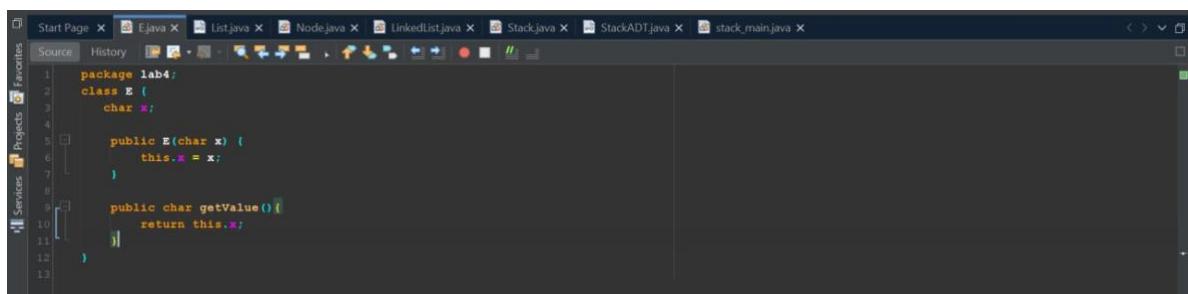
Name : Abdulmajeed Abdullah Almazmomi.

ID : 2240297.

Section : T66.

LAB : 5.

E class:

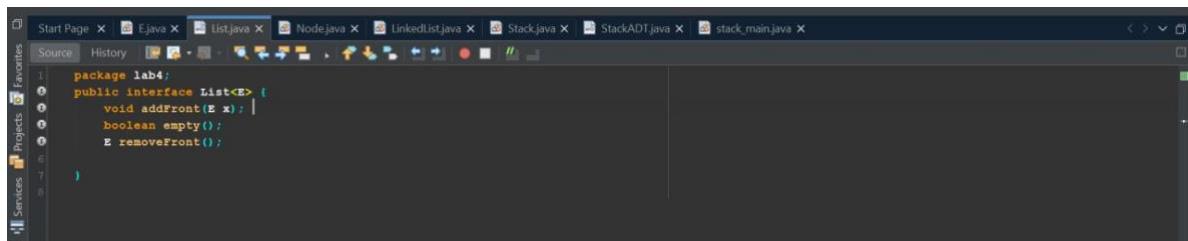


```
Start Page Ejava List.java Node.java LinkedList.java Stack.java StackADT.java stack_main.java

Source History

1 package lab4;
2 class E {
3     char x;
4
5     public E(char x) {
6         this.x = x;
7     }
8
9     public char getValue() {
10        return this.x;
11    }
12 }
13
```

List class:

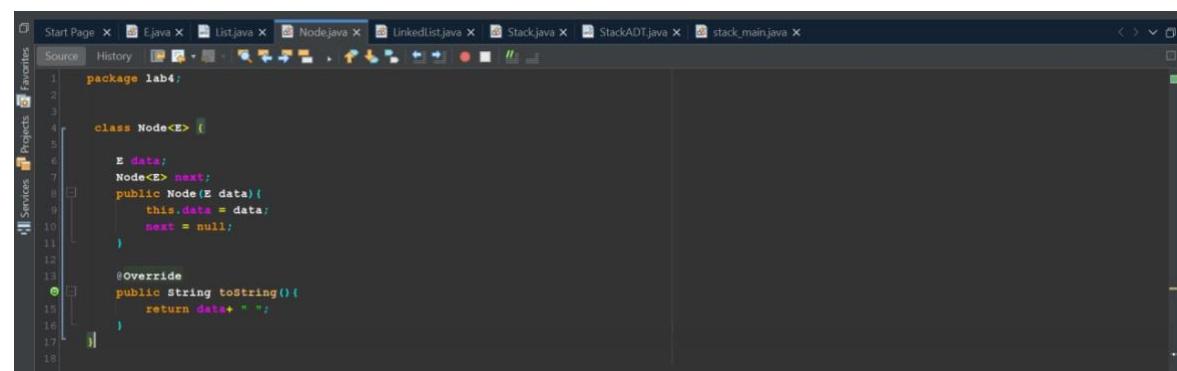


```
Start Page Ejava List.java Node.java LinkedList.java Stack.java StackADT.java stack_main.java

Source History

1 package lab4;
2 public interface List<E> {
3     void addFront(E x);
4     boolean empty();
5     E removeFront();
6 }
7
```

Node class:



```
Start Page Ejava List.java Node.java LinkedList.java Stack.java StackADT.java stack_main.java

Source History

1 package lab4;
2
3
4 class Node<E> {
5     E data;
6     Node<E> next;
7     public Node(E data) {
8         this.data = data;
9         next = null;
10    }
11
12    @Override
13    public String toString() {
14        return data+ " ";
15    }
16 }
17
18
```

Linkedlist class:

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows multiple tabs including "Start Page", "E.java", "List.java", "Node.java", "LinkedList.java", "Stack.java", "StackADT.java", and "stack_main.java".
- Left Sidebar:** Displays "Projects" and "Services" sections.
- Code Editor:** Contains the source code for the `LinkedList` class, which implements the `List<E>` interface. The code includes methods for getting the head node, adding front, checking if empty, and removing front.

```
1 package lab4;
2 public class LinkedList<E> implements List<E>{
3     private Node<E> head;
4
5     public E getHead()
6     {
7         if(empty())
8             return head.data;
9         return null;
10    }
11
12    public LinkedList()
13    {
14        head = null;
15    }
16
17    @Override
18    public void addFront(E x)
19    {
20        Node<E> node = new Node<E>(x);
21        node.next = head;
22        head = node;
23    }
24
25    @Override
26    public boolean empty()
27    {
28        if(head == null)
29            return true;
30        else
31            return false;
32    }
33
34    @Override
35    public E removeFront()
36    {
37        if(empty())
38            System.out.println("list is empty");
39        else
40        {
41            Node<E> temp = head;
42            head = head.next;
43            temp.next = null;
44            return temp.data;
45        }
46    }
47
48}
49
```

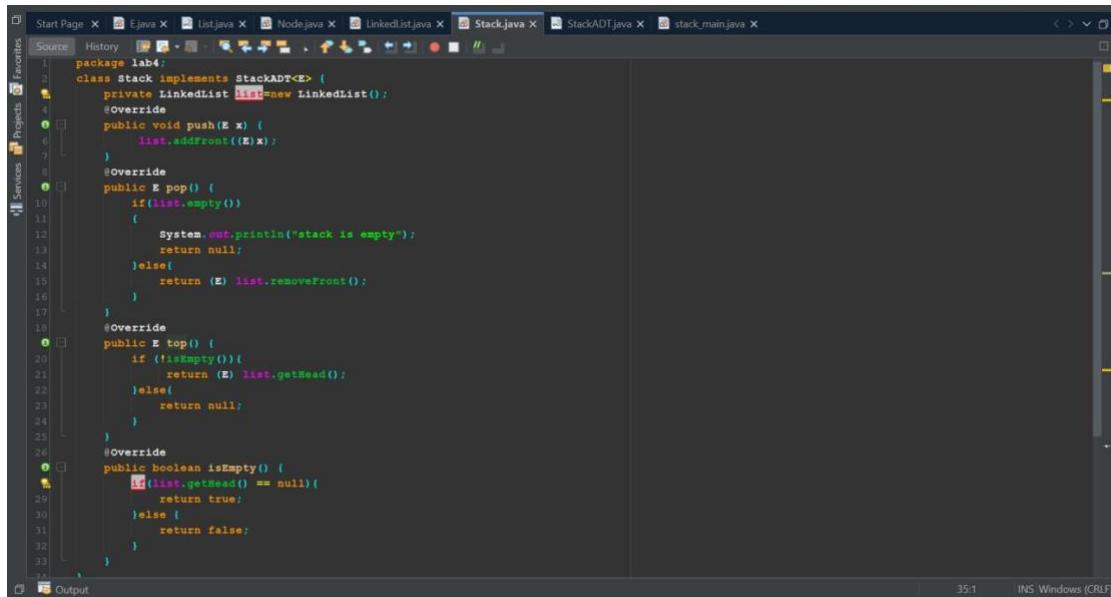
StackADT:

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows multiple tabs including "Start Page", "E.java", "List.java", "Node.java", "LinkedList.java", "Stack.java", "StackADT.java", and "stack_main.java".
- Left Sidebar:** Displays "Projects" and "Services" sections.
- Code Editor:** Contains the source code for the `StackADT` interface, which defines methods for pushing, popping, getting the top element, and checking if empty.

```
1 package lab4;
2 public interface StackADT<E>{
3     void push(E x);
4     E pop();
5     E top();
6     boolean isEmpty();
7 }
```

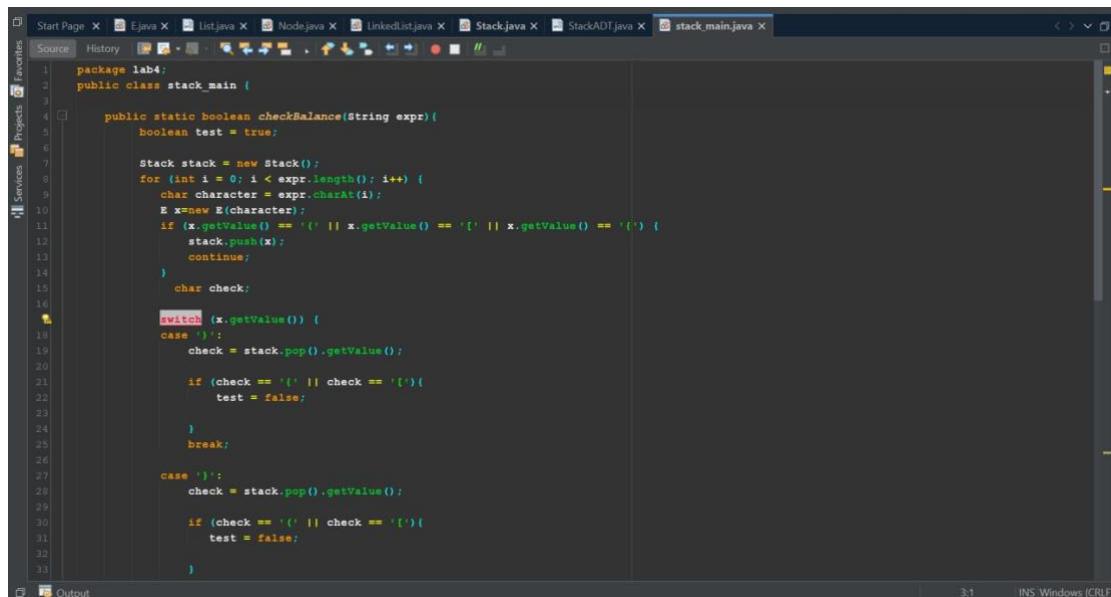
Stack class:



A screenshot of a Java IDE showing the code for the Stack class. The code implements the StackADT interface using a LinkedList. It includes methods for pushing and popping elements, getting the top element, and checking if the stack is empty.

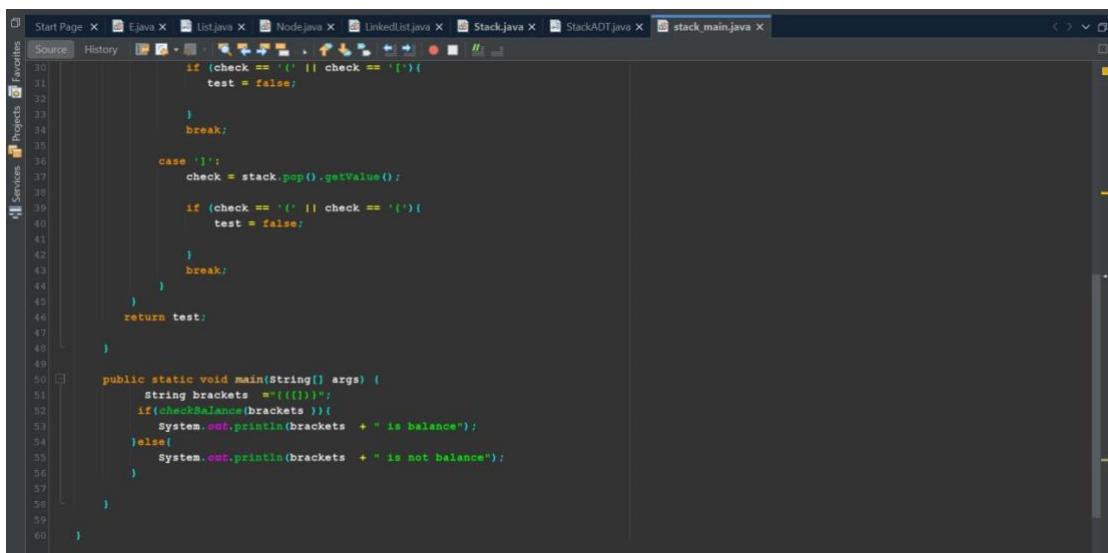
```
1 package lab4;
2 class Stack implements StackADT<E> {
3     private LinkedList<E> list = new LinkedList<E>();
4     @Override
5     public void push(E x) {
6         list.addFront((E)x);
7     }
8     @Override
9     public E pop() {
10        if(list.isEmpty())
11        {
12            System.out.println("stack is empty");
13            return null;
14        }
15        return (E) list.removeFront();
16    }
17    @Override
18    public E top() {
19        if (!isEmpty())
20            return (E) list.getHead();
21        else
22            return null;
23    }
24 }
25
26 @Override
27 public boolean isEmpty() {
28     if (list.getHead() == null)
29         return true;
30     else
31         return false;
32 }
```

Stack_main class:



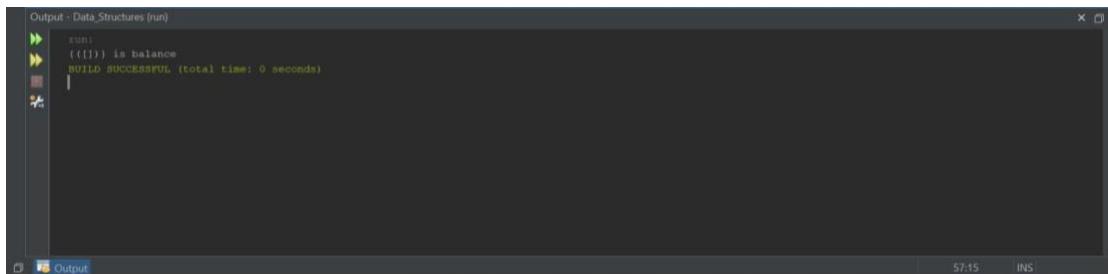
A screenshot of a Java IDE showing the stack_main.java file. This program contains a checkBalance method that uses a Stack to validate the balance of parentheses in a given expression.

```
1 package lab4;
2 public class stack_main {
3
4     public static boolean checkBalance(String expr) {
5         boolean test = true;
6
7         Stack stack = new Stack();
8         for (int i = 0; i < expr.length(); i++) {
9             char character = expr.charAt(i);
10            E x=new E(character);
11            if (x.getValue() == '(' || x.getValue() == '[' || x.getValue() == '{') {
12                stack.push(x);
13                continue;
14            }
15            char check;
16
17            switch (x.getValue()) {
18                case ')':
19                    check = stack.pop().getValue();
20
21                    if (check == '[' || check == '{') {
22                        test = false;
23
24                    }
25                    break;
26
27                case ']':
28                    check = stack.pop().getValue();
29
30                    if (check == '(' || check == '{') {
31                        test = false;
32
33                    }
34            }
35        }
36    }
37 }
```

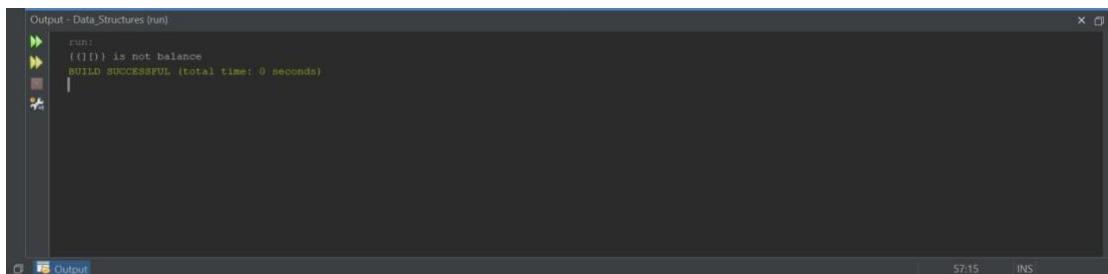


```
30     if (check == ')' || check == 'O'){
31         test = false;
32     }
33     break;
34
35     case '}':
36         check = stack.pop().getValue();
37
38         if (check == '{' || check == 'O'){
39             test = false;
40
41         }
42         break;
43
44     }
45
46     return test;
47
48 }
49
50 public static void main(String[] args) {
51     String brackets = "((()))";
52     if(checkBalance(brackets)){
53         System.out.println(brackets + " is balance");
54     }else{
55         System.out.println(brackets + " is not balance");
56     }
57
58 }
```

The output:



```
Output - Data_Structures (run)
▶ run:
((())) is balance
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
Output - Data_Structures (run)
▶ run:
(()) is not balance
BUILD SUCCESSFUL (total time: 0 seconds)
```