

Tejas Katkade

Brackets are tall punctuation marks used in matched pairs within text, to set apart or interject other text. Brackets refer to different types of brackets in different parts of the world and in different contexts.

Write a program which reads a String, which consists of alphabets [a-z, A-Z] and 3 types of brackets listed below:

1. Parentheses - ()
2. Square brackets - []
3. Braces or Curly brackets - {}

And determine whether every open bracket has a matching close bracket. If any open/close bracket doesn't have a matching close/open bracket or any extra open/close bracket then it is to be treated as invalid string.

Following are 3 examples of valid string:

- (the[is]{valid})
- {the(is[valid])}
- (this)(is)(valid)

Following are 4 examples of invalid string:

- (the[is]{invalid))
- (the[is]{invalid}}
- (this)(is){invalid)
- [this]{is}(invalid))

```
• (the[is]invalid))

New Tab Full Screen Java 16px Eclipse Word Wrap Shortout
Untitled x BracketMatch.java x
1 import java.util.Scanner;
2
3 class Stack {
4     int[] s;
5     int size;
6     int top = -1;
7     Stack() {
8         s = new int[40];
9     }
10
11     public boolean isEmpty() {
12
13         return top == -1;
14     }
15
16     public void push(int data) {
17         s[++top] = data;
18     }
19
20     public int pop() {
21         return s[top--];
22     }
23
24     public int peek() {
25         return s[top];
26     }
27 }
28
29 public class BracketMatch {

Ln: 1, Ch: 0, | Total Ln: 89

Compile Self Assessment Submit to Grader
```

```
• (the[is]invalid))

New Tab Full Screen Java 16px Eclipse Word Wrap Shortout
Untitled x BracketMatch.java x
30
31     public static boolean checkParenthesis(String str) {
32         Stack stack = new Stack();
33         int n = str.length();
34         for (int i=0; i<n; i++) {
35             if( str.charAt(i) == '[' || str.charAt(i) == '(' || str.charAt(i) == '{' ) {
36                 stack.push(str.charAt(i));
37             }
38             else if( str.charAt(i) >= 'a' && str.charAt(i) <= 'z' ||
39                     str.charAt(i) >= 'A' && str.charAt(i) <= 'Z' ){
40             }
41             else{
42                 if(stack.isEmpty()){
43                     return false;
44                 }
45                 if( (str.charAt(i) == ']' && (char)stack.peek() == '[' ) ||
46                     (str.charAt(i) == ')' && (char)stack.peek() == '(' ) ||
47                     (str.charAt(i) == '}' && (char)stack.peek() == '{' )
48                 ){
49                     stack.pop();
50                 }else{
51                     return false;
52                 }
53             }
54         }
55         if(stack.isEmpty()) {
56             return true;
57         }
58         return false;
59     }

Ln: 54, Ch: 24, | Total Ln: 78

Compile Self Assessment Submit to Grader
```

```

45-         if( (str.charAt(i) == '[' && (char)stack.peek() == '[') ||
46-             (str.charAt(i) == ')' && (char)stack.peek() == '(') ||
47-             (str.charAt(i) == '}' && (char)stack.peek() == '{') )
48-         ){
49-             stack.pop();
50-         }else{
51-             return false;
52-         }
53-     }
54- }
55- if(stack.isEmpty()) {
56-     return true;
57- }
58- return false;
59- }
60-
61- public static void main(String[] args){
62-     Scanner sc = new Scanner(System.in);
63-     int test = sc.nextInt();
64-     sc.nextLine();
65-     for(int i=0; i<test; i++) {
66-
67-         if(checkParenthesis(sc.nextLine())){
68-             System.out.println("TRUE");
69-         }else {
70-             System.out.println("FALSE");
71-         }
72-     }
73- }
74- }

```

Ln: 70, Ch: 17, | Total Ln: 75

Compile Self Assessment Submit to Grader

Parikshak Team  
parikshak[at]cdac[dot]in

Last updated: 04-Oct-2024 16:21

Write a program to create **Link-List** and different operations on it. The operation descriptions along with their code is given in the table below.

No	Operation Code	Operation Description
1	<b>AB</b>	Add the node at the beginning of the list.
2	<b>PR</b>	Print the link-list from head to end of the list, each node separated by space and terminated by a newline.
3	<b>AE</b>	Add the node at the end of the list.
4	<b>AMA</b>	Add the node at the middle after a particular node.
5	<b>AMB</b>	Add the node at the middle before a particular node.
6	<b>DN</b>	Delete the node.
7	<b>DNA</b>	Delete the node after a particular node.

Your program should take command as given above, corresponding input and perform the operations till the exit command ("EXIT") is given. Assume that 'node\_info' is a single integer  $0 < \text{node\_info} < 100$

#### Note

- Partial grading is enabled for this test. i.e. You will get partial marks for correct output for each input. Also each input is designed to test one or more of 'operations'
- So it is advisable to implement the functions (operations) incrementally and keep submitting the program it for grading once you have implemented and tested an operation.

New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java

```
1 import java.util.Scanner;
2 class SimpleLinkedList {
3
4     Node head;
5
6     static class Node {
7         int data;
8         Node next;
9
10        Node(int data) {
11            this.data = data;
12            this.next = null;
13        }
14    }
15    public Node createNode(int data) {
16        return new Node(data);
17    }
18
19    public void addB(int data) {
20        Node newNode = createNode(data);
21        if(head == null){
22            head = newNode;
23            return;
24        }
25
26        newNode.next = head;
27        head = newNode;
28    }
29
30    public void addE(int data) {
31        Node newNode = createNode(data);
32        if(head == null){
33            head = newNode;
34            return;
35        }
36    }
```

Ln: 30, Ch: 40, | Total Ln: 217

Close Test

New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java

```
29
30    public void addE(int data) {
31        Node newNode = createNode(data);
32        if(head == null){
33            head = newNode;
34            return;
35        }
36        Node temp = head;
37
38        while(temp.next != null) {
39            temp = temp.next;
40        }
41        temp.next = newNode;
42    }
43
44    public void addA(int key, int data) {
45        Node newNode = createNode(data);
46
47        if(head == null) {
48            return;
49        }
50
51        Node temp = head;
52        while(temp != null && temp.data != key){
53            temp = temp.next;
54        }
55        if(temp != null) {
56            if(temp.next != null) {
57                newNode.next = temp.next;
58            }
59            temp.next = newNode;
60        }
61    }
62
63    public void addB(int key, int data) {
```

Ln: 30, Ch: 40, | Total Ln: 217

Close Test

```
New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut
Untitled BracketMatch.java SimpleLinkedList.java
63
64 public void addB(int key, int data) {
65     Node newNode = createNode(data);
66     if(head == null) {
67         return;
68     }
69     Node temp = head;
70     Node prev = null;
71     while(temp != null && temp.data != key){
72         prev = temp;
73         temp = temp.next;
74     }
75     if(temp != null) {
76         if(temp == head) {
77             newNode.next = head;
78             head = newNode;
79         }else {
80             newNode.next = prev.next;
81             prev.next = newNode;
82         }
83     }
84 }
85 public void deleteNA(int key){
86     if(head == null) {
87         return;
88     }
89     Node temp = head;
90     while(temp != null && temp.data != key){
91         temp = temp.next;
92     }
93     if(temp.next != null) {
94         temp.next = temp.next.next;
95     }
96 }
97 public void deleteNB(int key){
98
```

Ln: 94, Ch: 9, | Total Ln: 212

Close Test

```
New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut
Untitled BracketMatch.java SimpleLinkedList.java
97 public void deleteNB(int key){
98     if(head == null) {
99         return;
100     }
101 }
102
103 Node temp = head;
104 Node prev = null;
105 while(temp.next != null && temp.next.data != key){
106     prev = temp;
107     temp = temp.next;
108 }
109
110 if(temp.next != null) {
111     if(temp == head) {
112         head = head.next;
113     }else {
114         prev.next = prev.next.next;
115     }
116 }
117 }
118
119 public void deleteN(int key){
120     if(head == null) {
121         return;
122     }
123 }
124
125 Node temp = head;
126 if(temp.data == key) {
127     head = head.next;
128 }
129
130 while(temp.next != null && temp.next.data != key){
131     temp = temp.next;
132 }
133
```

Ln: 94, Ch: 9, | Total Ln: 212

Close Test

New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java

```
119 public void deleteN(int key){
120
121     if(head == null) {
122         return;
123     }
124
125     Node temp = head;
126     if(temp.data == key) {
127         head = head.next;
128     }
129
130     while(temp.next != null && temp.next.data != key){
131         temp = temp.next;
132     }
133
134     if(temp.next != null) {
135         temp.next = temp.next.next;
136     }
137
138 }
139
140 public void print(){
141     if(head == null) {
142         return;
143     }
144
145     Node temp = head;
146     while(temp != null) {
147         System.out.print(temp.data+" ");
148         temp = temp.next;
149     }
150     System.out.println();
151 }
152
153 public static void main(String[] args) {
```

Ln: 94, Ch: 9, | Total Ln: 212

Close Test

New Tab Full Screen Java 13px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java

```
153 public static void main(String[] args) {
154     Scanner sc = new Scanner(System.in);
155     SimpleLinkedList ll = new SimpleLinkedList();
156     while(true){
157         String str = sc.next();
158         switch(str){
159             case "AB":
160                 {
161                     int data = sc.nextInt();
162                     sc.nextLine();
163                     ll.addB(data);
164                 }
165                 break;
166             case "AE":
167                 {
168                     int data = sc.nextInt();
169                     sc.nextLine();
170                     ll.addE(data);
171                 }
172                 break;
173             case "AMA":
174                 {
175                     int key = sc.nextInt();
176                     int data = sc.nextInt();
177                     sc.nextLine();
178                     ll.addA(key,data);
179                 }
180                 break;
181             case "AMB":
182                 {
183                     int key = sc.nextInt();
184                     int data = sc.nextInt();
185                     sc.nextLine();
186                     ll.addB(key,data);
187                 }
188         }
189     }
190 }
```

Ln: 94, Ch: 9, | Total Ln: 212

Close Test

```

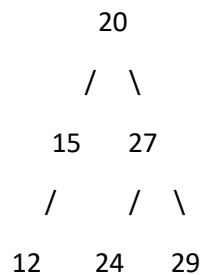
175         int key = sc.nextInt();
176         int data = sc.nextInt();
177         sc.nextLine();
178         ll.addA(key,data);
179     }
180     break;
181     case "AMB":
182     {
183         int key = sc.nextInt();
184         int data = sc.nextInt();
185         sc.nextLine();
186         ll.addB(key,data);
187     }
188     break;
189     case "PR":
190         ll.print();
191         break;
192     case "DN":
193         int data = sc.nextInt();
194         sc.nextLine();
195         ll.deleteN(data);
196         break;
197     case "DNA":
198     {
199         int key = sc.nextInt();
200         sc.nextLine();
201         ll.deleteA(key);
202     }
203     break;
204     case "EXIT":
205         return;
206 }
207 }
208 }
209 }

```

Ln: 206, Ch: 9, | Total Ln: 210

Close Test

Write a Java program to create a binary search tree, where a left child node is always less than its parent node and a right child node is always greater than the parent node. Create a tree by taking input of positive integers. Once the tree is created search a particular value in the tree and print the search path starting from root. Assume that the keys in all nodes are unique.



**fig1.**

In the Fig 1. tree search path as following for given node

20 => Root

24 => Root R L

25 => Not found

12 => Root L L

**Note:** R is for right, L is for Left

New Tab Full Screen Java 15px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java TreePath.java

```

1 import java.util.Scanner;
2
3 class BST{
4     Node root;
5     static class Node{
6         Node left;
7         int data;
8         Node right;
9
10        Node(int data) {
11            this.data = data;
12            this.left = null;
13            this.right = null;
14        }
15    }
16
17    public Node insert(Node root, int data) {
18        if(root == null) {
19            root = new Node(data);
20            return root;
21        }
22
23        if(data < root.data) {
24            root.left = insert(root.left, data);
25        }else{
26            root.right = insert(root.right, data);
27        }
28        return root;
29    }
30
    
```

New Tab Full Screen Java 15px Eclipse Word Wrap Shortcut

Untitled BracketMatch.java SimpleLinkedList.java TreePath.java

```

30
31 public void search(Node root, int data, String str) {
32     if(root == null) {
33         System.out.println("Not Found");
34         return;
35     }
36     if(root.data == data){
37         System.out.println(str);
38         return;
39     }
40
41     if(data < root.data) {
42         str = str + "L ";
43         search(root.left, data, str);
44     }else{
45         str = str + "R ";
46         search(root.right, data, str);
47     }
48 }
49 public void inOrder(Node root){
50     if(root == null){
51         return ;
52     }
53
54     inOrder(root.left);
55     System.out.print(root.data);
56     inOrder(root.right);
57 }
58 }
59 public class TreePath{
60     public static void main(String[] args) {
    
```



The screenshot shows the Eclipse IDE interface. At the top, there's a status bar with 'Time Left: 01:44:20' and a 'PARIKSHAK' logo with the tagline 'LET'S SOLVE IT'. The IDE has several tabs open: 'BracketMatch.java', 'SimpleLinkedList.java', and 'TreePath.java'. The 'TreePath.java' tab is active, displaying the following Java code:

```
55 System.out.print(root.data);
56 inOrder(root.right);
57 }
58 }
59 public class TreePath{
60 public static void main(String[] args ) {
61 Scanner sc = new Scanner(System.in);
62 int[] arr = new int[100];
63 BST bst = new BST();
64 int input = 0;
65 int m = 0;
66 while(input != -1){
67 input = sc.nextInt();
68 arr[m++] = input;
69 }
70 int j = 1;
71 bst.root = bst.insert(bst.root, arr[0]);
72 while(arr[j] != -1){
73 bst.insert(bst.root, arr[j++]);
74 }
75 int size = sc.nextInt();
76 int[] a = new int[size];
77 for(int i = 0; i < size; i++){
78 a[i] = sc.nextInt();
79 }
80 for(int i = 0; i < size; i++){
81 bst.search(bst.root, a[i], "Root ");
82 }
83 }
84 }
```

The status bar at the bottom right indicates 'Ln: 81, Ch: 9, | Total Ln: 85'.

Implement a circular queue in c++/java, to support "**enqueue**" and "**dequeue**" operation of a given size. Your program should support following commands as given below:

- **enqueue N** // insert N in the Queue, if queue is full print the output as "**FULL**"
- **dequeue** // Remove the first elements from the Queue and print the value if the Queue is empty print "**EMPTY**"
- **exit** // exit the program

assume the input will be always a positive integer.

```
New Tab Full Screen Java 15px Eclipse Word Wrap Shortcut
Untitled BracketMatch.java Untitled SimpleLinkedList.java TreePath.java CircularQ.java
1 import java.util.Scanner;
2 public class CircularQ{
3     int[] CQ;
4     int size;
5     int front;
6     int rear;
7     CircularQ(int size) {
8         this.size = size;
9         CQ = new int[size];
10        front = -1;
11        rear = -1;
12    }
13    public boolean isEmpty()
14    {
15        return front == -1;
16    }
17    public boolean isFull(){
18        return front == (rear+1)%size;
19    }
20    public void enqueue(int data) {
21        if(isFull()) {
22            System.out.println("FULL");
23            return;
24        }
25        if(front == -1){
26            front = 0;
27        }
28        rear = (rear+1)%size;
29        CQ[rear] = data;
30    }
}
```

Ln: 0, Ch: 25, | Total Ln: 73

Close Test

```
New Tab Full Screen Java 15px Eclipse Word Wrap Shortcut
Untitled BracketMatch.java Untitled SimpleLinkedList.java TreePath.java CircularQ.java
31 public void deque() {
32     if(isEmpty()) {
33         System.out.println("EMPTY");
34         return;
35     }
36     System.out.println(CQ[front]);
37     if(front == rear) {
38         front = -1;
39         rear = -1;
40     }else{
41         front = (front+1)%size;
42     }
43 }
44 public static void main(String[] args ){
45     Scanner sc = new Scanner(System.in);
46     int size = sc.nextInt();
47     CircularQ cq = new CircularQ(size);
48     sc.nextLine();
49     while(true){
50         String str = sc.next();
51         switch(str){
52             case "enqueue":
53                 int data = sc.nextInt();
54                 sc.nextLine();
55                 cq.enqueue(data);
56                 break;
57             case "deque":
58                 cq.deque();
59                 break;
60         }
}
```

Ln: 51, Ch: 24, | Total Ln: 67

Close Test

```
New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut
Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java

38     front = -1;
39     rear = -1;
40 }else{
41     front = (front+1)%size;
42 }
43 }
44 public static void main(String[] args ){
45
46     Scanner sc = new Scanner(System.in);
47     int size = sc.nextInt();
48     CircularQ cq = new CircularQ(size);
49     sc.nextLine();
50     while(true){
51         String str = sc.next();
52         switch(str){
53             case "enqueue":
54                 int data = sc.nextInt();
55                 sc.nextLine();
56                 cq.enqueue(data);
57                 break;
58             case "dequeue":
59                 cq.dequeue();
60                 break;
61             case "exit":
62                 return;
63         }
64     }
65 }
66
67 }
```

Ln: 51, Ch: 24, | Total Ln: 67

Close Test

- Input: "{([()])}"  
Output: Balanced
- Input: "([])"  
Output: Not Balanced

The screenshot shows the Eclipse IDE with the following code in the `Parenthesis.java` file:

```
1 import java.io.BufferedReader;
2 import java.io.InputStreamReader;
3 import java.io.IOException;
4
5 class Stack {
6     int[] s;
7     int size;
8     int top = -1;
9     Stack() {
10         s = new int[40];
11     }
12     public boolean isEmpty() {
13         return top == -1;
14     }
15     public void push(int data) {
16         s[++top] = data;
17     }
18     public int pop() {
19         return s[top--];
20     }
21     public int peek() {
22         return s[top];
23     }
24 }
25 public class Parenthesis {
26     public static boolean checkParenthesis(String str) {
27         Stack stack = new Stack();
28         int n = str.length();
29         for (int i=0; i<n; i++) {
30
31             if( str.charAt(i) == '[' || str.charAt(i) == '{' || str.charAt(i) == '(' )
32             {
33                 stack.push(str.charAt(i));
34             }
35         }
36     }
37 }
```

The status bar at the bottom right indicates "Ln: 19, Ch: 5, | Total Ln: 64". A "Close Test" button is visible in the bottom right corner.

The screenshot shows the Eclipse IDE with the following code in the `Parenthesis.java` file:

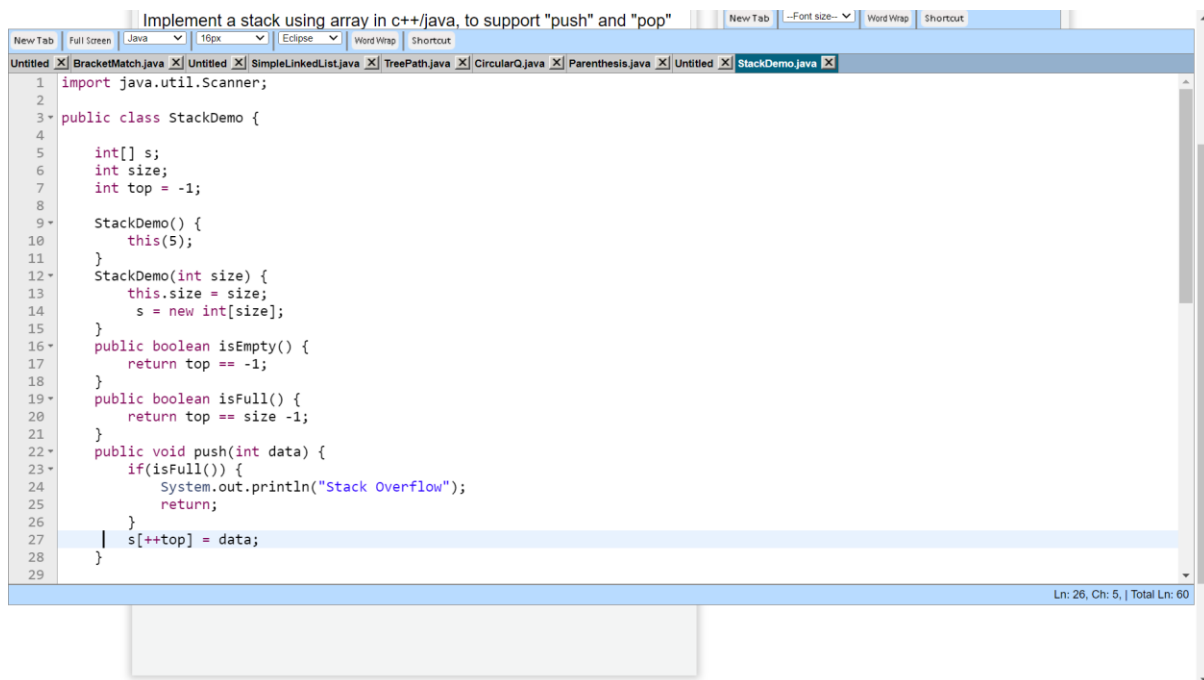
```
31 if( str.charAt(i) == '[' || str.charAt(i) == '{' || str.charAt(i) == '(' )
32 {
33     stack.push(str.charAt(i));
34 }else{
35
36     if(stack.isEmpty()) {
37         return false;
38     }
39
40     if( (str.charAt(i) == ']' && (char)stack.peek() == '[') ||
41         (str.charAt(i) == '}' && (char)stack.peek() == '{') ||
42         (str.charAt(i) == ')' && (char)stack.peek() == '(') )
43     {
44         stack.pop();
45     }else{
46         return false;
47     }
48 }
49
50 if(stack.isEmpty()) {
51     return true;
52 }
53 return false;
54 }
55
56 public static void main(String[] args) throws IOException{
57     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
58     if(checkParenthesis(br.readLine())){
59         System.out.println("Balanced");
60     }else {
61         System.out.println("Not Balanced");
62     }
63 }
64 }
```

The status bar at the bottom right indicates "Ln: 57, Ch: 33, | Total Ln: 64". A "Close Test" button is visible in the bottom right corner.

Implement a stack using array in c++/java, to support "push" and "pop" operation of a given size. Your program should support following commands as given below:

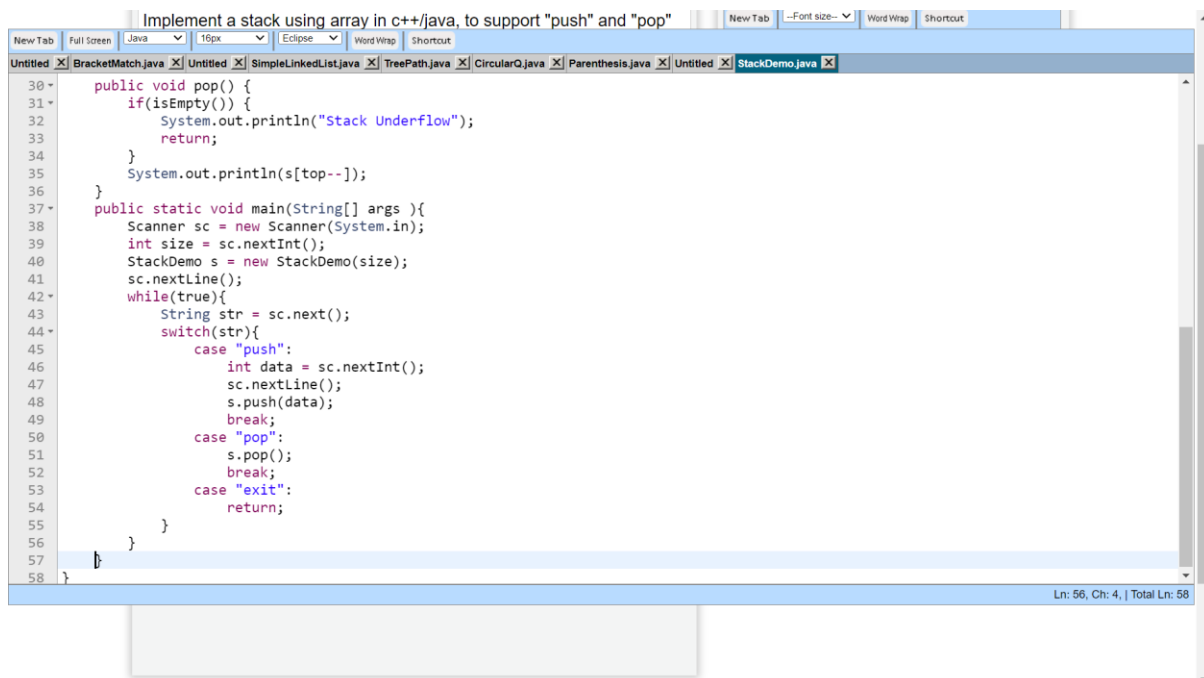
- push N // insert N in the stack, if stack is full print the output as "Stack Overflow"
- pop // Remove the last elements from the Stack and print the value and if the stack is empty print "Stack Underflow"
- exit // exit the program

assume the input will be always a positive integer.



```
1 import java.util.Scanner;
2
3 public class StackDemo {
4     int[] s;
5     int size;
6     int top = -1;
7
8     StackDemo() {
9         this(5);
10    }
11    StackDemo(int size) {
12        this.size = size;
13        s = new int[size];
14    }
15    public boolean isEmpty() {
16        return top == -1;
17    }
18    public boolean isFull() {
19        return top == size - 1;
20    }
21    public void push(int data) {
22        if(isFull()) {
23            System.out.println("Stack Overflow");
24            return;
25        }
26        s[++top] = data;
27    }
28 }
29
```

Ln: 26, Ch: 5, | Total Ln: 60



```
30 public void pop() {
31     if(isEmpty()) {
32         System.out.println("Stack Underflow");
33         return;
34     }
35     System.out.println(s[top--]);
36 }
37 public static void main(String[] args ){
38     Scanner sc = new Scanner(System.in);
39     int size = sc.nextInt();
40     StackDemo s = new StackDemo(size);
41     sc.nextLine();
42     while(true){
43         String str = sc.next();
44         switch(str){
45             case "push":
46                 int data = sc.nextInt();
47                 sc.nextLine();
48                 s.push(data);
49                 break;
50             case "pop":
51                 s.pop();
52                 break;
53             case "exit":
54                 return;
55         }
56     }
57 }
58
```

Ln: 56, Ch: 4, | Total Ln: 58