

```
1  import java.util.*;
2
3  class Node{
4
5      private int data;
6      private Node nextNodePtr;
7      private Node prevNodePtr;
8
9      public Node () {}
10
11     public void setData(int d){
12         data = d;
13     }
14
15     public int getData () {
16         return data;
17     }
18
19     public void setNextNodePtr (Node nodePtr) {
20         nextNodePtr = nodePtr;
21     }
22
23     public Node getNextNodePtr () {
24         return nextNodePtr;
25     }
26
27     public void setPrevNodePtr (Node nodePtr) {
28         prevNodePtr = nodePtr;
29     }
30
31     public Node getPrevNodePtr () {
32         return prevNodePtr;
33     }
34 }
35
36
37 class Stack{
38
39     private Node headPtr;
40     private Node tailPtr;
41
42     public Stack () {
43         headPtr = new Node ();
44         tailPtr = new Node ();
45         headPtr.setNextNodePtr (null);
46         tailPtr.setPrevNodePtr (null);
47     }
48
49     public Node getHeadPtr () {
50         return headPtr;
51     }
52
53     public Node getTailPtr () {
54         return tailPtr;
55     }
56
57     public boolean isEmpty () {
58
59         if (headPtr.getNextNodePtr () == null)
60             return true;
61
62         return false;
63     }
64 }
```

```

65
66     public void push(int data){
67
68         Node newNodePtr = new Node();
69         newNodePtr.setData(data);
70         newNodePtr.setNextNodePtr(null);
71
72         Node lastNodePtr = tailPtr.getPrevNodePtr();
73
74         if (lastNodePtr == null){
75
76             headPtr.setNextNodePtr(newNodePtr);
77             newNodePtr.setPrevNodePtr(null);
78
79         }
80         else{
81
82             lastNodePtr.setNextNodePtr(newNodePtr);
83             newNodePtr.setPrevNodePtr(lastNodePtr);
84
85         }
86
87         tailPtr.setPrevNodePtr(newNodePtr);
88
89     }
90
91     public int pop(){
92
93         Node lastNodePtr = tailPtr.getPrevNodePtr();
94         Node prevNodePtr = null;
95
96         int poppedData = -100000; //empty stack
97
98         if (lastNodePtr != null){
99             prevNodePtr = lastNodePtr.getPrevNodePtr();
100            poppedData = lastNodePtr.getData();
101        }
102        else
103            return poppedData;
104
105        if (prevNodePtr != null){
106            prevNodePtr.setNextNodePtr(null);
107            tailPtr.setPrevNodePtr(prevNodePtr);
108        }
109        else{
110            headPtr.setNextNodePtr(null);
111            tailPtr.setPrevNodePtr(null);
112        }
113
114        return poppedData;
115    }
116
117 }
118
119
120     public int peek(){
121
122         Node lastNodePtr = tailPtr.getPrevNodePtr();
123
124         if (lastNodePtr != null)
125             return lastNodePtr.getData();
126         else
127             return -100000; // empty stack
128

```

```

129     }
130
131
132
133
134 }
135
136
137 class DoublyLinkedList{
138
139     public static void main(String[] args){
140
141         Stack stack = new Stack();
142
143         String expression;
144
145         Scanner input = new Scanner(System.in);
146
147         System.out.print("Enter the expression to evaluate: ");
148         expression = input.nextLine();
149
150         StringTokenizer stk = new StringTokenizer(expression, " , ");
151
152         while (stk.hasMoreTokens()){
153
154             String token = stk.nextToken();
155
156             boolean isOperator = false;
157
158             if ( (token.equals("*")) || (token.equals("/")) || (token.equals("+")) || (token
159                 .equals("-")) )
160                 isOperator = true;
161
162             if (!isOperator){
163                 int val = Integer.parseInt(token);
164                 stack.push(val);
165             }
166
167             if (isOperator){
168
169                 int rightOperand = stack.pop();
170                 int leftOperand = stack.pop();
171
172                 if (token.equals("*")){
173                     int result = leftOperand * rightOperand;
174                     System.out.println("intermediate result: " + result);
175                     stack.push(result);
176                 }
177                 else if (token.equals("/")){
178                     int result = leftOperand / rightOperand;
179                     System.out.println("intermediate result: " + result);
180                     stack.push(result);
181                 }
182                 else if (token.equals("+")){
183                     int result = leftOperand + rightOperand;
184                     System.out.println("intermediate result: " + result);
185                     stack.push(result);
186                 }
187                 else if (token.equals("-")){
188                     int result = leftOperand - rightOperand;
189                     System.out.println("intermediate result: " + result);
190                     stack.push(result);
191                 }

```

```
192
193     }
194
195
196
197 }
198
199     System.out.println("final result: " + stack.pop() );
200
201 }
202
203 }
```

```
Enter the expression to evaluate: 2, 3, *, 1, 5, *, +, 4, -  
intermediate result: 6  
intermediate result: 5  
intermediate result: 11  
intermediate result: 7  
final result: 7
```
