```java
import java.util.*;

class Node{

    private char data;
    private Node nextNodePtr;
    private Node prevNodePtr;

    public Node(){}

    public void setData(char d){
        data = d;
    }

    public char getData(){
        return data;
    }

    public void setNextNodePtr(Node nodePtr){
        nextNodePtr = nodePtr;
    }

    public Node getNextNodePtr(){
        return nextNodePtr;
    }

    public void setPrevNodePtr(Node nodePtr){
        prevNodePtr = nodePtr;
    }

    public Node getPrevNodePtr(){
        return prevNodePtr;
    }

}

class Stack{

    private Node headPtr;
    private Node tailPtr;

    public  Stack(){
        headPtr = new Node();
        tailPtr = new Node();
        headPtr.setNextNodePtr(null);
        tailPtr.setPrevNodePtr(null);
    }

    public Node getHeadPtr(){
        return headPtr;
    }

    public  Node getTailPtr(){
        return tailPtr;
    }

    public  boolean isEmpty(){

        if (headPtr.getNextNodePtr() == null)
            return true;

        return false;
    }

```

```
65
66      public  void push(char 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
92      public char pop(){
93
94          Node lastNodePtr = tailPtr.getPrevNodePtr();
95          Node prevNodePtr = null;
96
97          char poppedData = '$'; //empty stack
98
99          if (lastNodePtr != null){
100             prevNodePtr = lastNodePtr.getPrevNodePtr();
101             poppedData = lastNodePtr.getData();
102         }
103         else
104             return poppedData;
105
106         if (prevNodePtr != null){
107             prevNodePtr.setNextNodePtr(null);
108             tailPtr.setPrevNodePtr(prevNodePtr);
109         }
110         else{
111             headPtr.setNextNodePtr(null);
112             tailPtr.setPrevNodePtr(null);
113         }
114
115         return poppedData;
116
117     }
118
119
120     public  char peek(){
121
122         Node lastNodePtr = tailPtr.getPrevNodePtr();
123
124         if (lastNodePtr != null)
125             return lastNodePtr.getData();
126         else
127             return '$'; //  empty stack
128
```

```
    }


}


class DoublyLinkedList{

  public static void main(String[] args){

    String expression;

    Scanner input = new Scanner(System.in);
    System.out.print("Enter an expression: ");
    expression = input.nextLine();

    Stack stack = new Stack();

    int index = 0;

    while (index < expression.length()){

        char symbol = expression.charAt(index);

        if (symbol == '{' || symbol == '(' || symbol == '['){
            stack.push(symbol);
            index++;
            continue;
        }
        else if (symbol == '}' || symbol == ')' || symbol == ']'){

            char topSymbol = stack.pop();
            if ( (topSymbol == '{' && symbol == '}') ||
                 (topSymbol == '(' && symbol == ')') ||
                 (topSymbol == '[' && symbol == ']') ){

                    index++;
                    continue;

            }
            else{

                System.out.println("parenthesis not balanced!!");
                return;

            }
        }
        else{

            System.out.println("Invalid symbol "+ symbol + " in the expression!!");
            return;
        }

    }

    System.out.println(expression + " is balanced!!");

   }

}
```

```
Enter an expression: {()[()]}
{()[()]} is balanced!!

Enter an expression: {{})
parenthesis not balanced!!

Enter an expression: {{}p}
Invalid symbol p in the expression!!
```