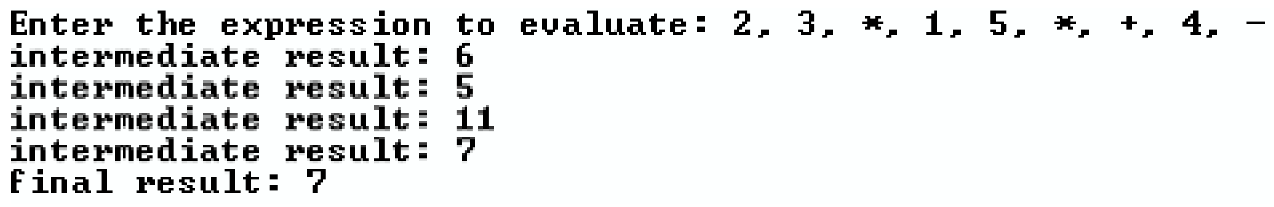
```java
import java.util.*;

class Node{

    private int data;
    private Node nextNodePtr;
    private Node prevNodePtr;

    public Node(){}

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

    public int 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;
    }


    public  void push(int data){

        Node newNodePtr = new Node();
        newNodePtr.setData(data);
        newNodePtr.setNextNodePtr(null);

        Node lastNodePtr = tailPtr.getPrevNodePtr();

        if (lastNodePtr == null){

            headPtr.setNextNodePtr(newNodePtr);
            newNodePtr.setPrevNodePtr(null);

        }
        else{

            lastNodePtr.setNextNodePtr(newNodePtr);
            newNodePtr.setPrevNodePtr(lastNodePtr);

        }

        tailPtr.setPrevNodePtr(newNodePtr);

    }


    public int pop(){

        Node lastNodePtr = tailPtr.getPrevNodePtr();
        Node prevNodePtr = null;

        int poppedData = -100000; //empty stack

        if (lastNodePtr != null){
            prevNodePtr = lastNodePtr.getPrevNodePtr();
            poppedData = lastNodePtr.getData();
        }
        else
            return poppedData;

        if (prevNodePtr != null){
            prevNodePtr.setNextNodePtr(null);
            tailPtr.setPrevNodePtr(prevNodePtr);
        }
        else{
            headPtr.setNextNodePtr(null);
            tailPtr.setPrevNodePtr(null);
        }

        return poppedData;

    }


    public  int peek(){

        Node lastNodePtr = tailPtr.getPrevNodePtr();

        if (lastNodePtr != null)
            return lastNodePtr.getData();
        else
            return -100000; //  empty stack

    }




}


class DoublyLinkedList{

   public static void main(String[] args){

    Stack stack = new Stack();

    String expression;

    Scanner input = new Scanner(System.in);

    System.out.print("Enter the expression to evaluate: ");
    expression = input.nextLine();

    StringTokenizer stk = new StringTokenizer(expression, ", ");

    while (stk.hasMoreTokens()){

        String token = stk.nextToken();

        boolean isOperator = false;

        if ( (token.equals("*")) || (token.equals("/")) || (token.equals("+")) || (token.equals("-")) )
            isOperator = true;

        if (!isOperator){
            int val = Integer.parseInt(token);
            stack.push(val);
        }


        if (isOperator){

            int rightOperand = stack.pop();
            int leftOperand = stack.pop();

            if (token.equals("*")){
                int result = leftOperand * rightOperand;
                System.out.println("intermediate result: " + result);
                stack.push(result);
            }
            else if (token.equals("/")){
                int result = leftOperand / rightOperand;
                System.out.println("intermediate result: " + result);
                stack.push(result);
            }
            else if (token.equals("+")){
                int result = leftOperand + rightOperand;
                System.out.println("intermediate result: " + result);
                stack.push(result);
            }
            else if (token.equals("-")){
                int result = leftOperand - rightOperand;
                System.out.println("intermediate result: " + result);
                stack.push(result);
            }

        }



    }

    System.out.println("final result: " + stack.pop() );

   }

}
```



```
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
```