스택 _ 계산기

1. 중위 표기법의 수식을 후위 표기법으로 변경
2. 후위 표기법의 수식을 스택을 이용하여 계산
3. 코드 구현

1. 중위 표기법의 수식을 후위 표기법으로 변경

1. **토큰 하나씩 가져오기**

- 토큰이 연산자면 : 연산자 우선순위에 따라 처리
- 토큰이 피연산자면 출력

연산자 우선순위

토큰	isp (in-stack)	icp(in-coming)
)	-	-
*, /	2	2
+, -	1	1
(	0	3

⇒ 새로운 연산자의 icp와 스택 최상단의 isp 비교

→ 새로운 연산자의 icp가 **더 큼** : 해당 연산자를 스택에 삽입

→ 새로운 연산자의 icp가 **작거나 같음** : 스택 상단에 있는 연산자 출력

⇒ (스택 맨 위에 있는 애가 이거면) 꺼낸다 / 쌓는다

토큰	꺼낸다	쌓는다
*, /	*, /	+, -, (,)
+, -	*, /, +, -	(,)
(		무조건 쌓는다.
)	(만날 때까지 꺼낸다	

⇒ 오른쪽 괄호 나오면 스택에서 왼쪽 괄호 나올 때까지 모든 연산자 꺼내어 출력하고, 왼쪽 괄호 나오면 삭제

2. **스택에 남아있는 연산자 모두 출력**

(예)

$(6 + 5 * (2 - 8)/2)$ **를 후위 표기법으로 바꾸고 싶어**

① (

→ 연산자니까 연산자 우선순위에 따라 처리해

→ 스택이 비어있고 '('는 무조건 쌓으니까 스택에 쌓아

현재 스택 : (

② 6

→ 피연산자니까 출력해

현재 출력 : 6

③ +

→ 연산자니까 연산자 우선순위에 따라 처리해

→ 스택 맨 위에 있는 애가 ‘(’니까 스택에 쌓아

현재 스택 : (+

④ 5

→ 피연산자니까 출력해

현재 출력 : 6 5

⑤ *

~~→ 연산자니까 연산자 우선순위에 따라 처리해~~

→ 스택 맨 위에 있는 애가 ‘+’니까 스택에 쌓아

현재 스택 : (+ *

⑥ (

→ ‘(’는 무조건 쌓으니까 스택에 쌓아

현재 스택 : (+ * (

⑦ 2

→ 현재 출력 : 6 5 2

⑧ -

→ 스택 맨 위에 있는 애가 ‘(’니까 스택에 쌓아

현재 스택 : (+ * (-

⑨ 8

→ 현재 출력 : 6 5 2 8

⑩)

→ ‘(’ 만날 때까지 꺼내서 출력하고 ‘(’는 삭제해

현재 스택 : (+ *

현재 출력 : 6 5 2 8 -

⑪ /

→ 스택 맨 위에 있는 애가 '*'니까 '*' 꺼내

→ 그 다음 애는 '+'니까 쌓아

현재 스택 : (+ /

현재 출력 : 6 5 2 8 - *

⑫ 2

→ 현재 출력 6 5 2 8 - * 2

⑬)

→ '(' 만날 때까지 꺼내서 출력하고 '('는 삭제해

현재 스택 :

현재 출력 : 6 5 2 8 - * 2 / +

⇒ 수식 끝나고 스택도 비었다! 끝 ! ㅠ

2. 후위 표기법의 수식을 스택을 이용하여 계산

1. **토큰 하나씩 가져오기**

- 토큰이 연산자면 : 필요한만큼의 피연산자 스택에서 pop하여 연산, 그 결과 다시 스택에 push
- 토큰이 피연산자면 스택에 push

2. **수식이 끝나면, 마지막으로 스택을 pop하여 출력**

(예)

아까 후위 표기법으로 바꾼 수식 계산해보쟈

6 5 2 8 - * 2 / +

① 6 5 2 8 스택에 쌓기

→ 현재 스택 : 6 5 2 8

② -

→ 연산자니까 스택에서 피연산자 두 번 pop() 하자

현재 스택 : 6 5 (2랑 8 빠짐)

→ 계산 결과 다시 스택에 넣자 (2-8=-6) (*스택에 먼저 있던 애가 앞으로 와야 함)

현재 스택 : 6 5 -6

③ *

→ 연산자니까 스택에서 피연산자 두 번 pop() 하자

현재 스택 : 6 (5랑 -6 빠짐)

→ 계산 결과 다시 스택에 넣자 (5 * (-6) = -30)

현재 스택 : 6 -30

④ 2

→ 현재 스택 : 6 -30 2

⑤ /

→ 연산자니까 스택에서 피연산자 두 번 pop() 하자

현재 스택 : 6 (-30이랑 2 빠짐)

→ 계산 결과 다시 스택에 넣자 (-30/2 = -15)

현재 스택 : 6 -15

⑥ +

→ 연산자니까 스택에서 피연산자 두 번 pop() 하자

현재 스택 : (6이랑 -15 빠짐)

→ 계산 결과 다시 스택에 넣자 (6+(-15) = -9)

현재 스택 : -9

⇒ 수식 끝났다. 스택 pop하자!

⇒ 결과 = -9

3. 코드 구현

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Stack;
import java.util.StringTokenizer;

public class InfixtoPostfix {
  public static void main(String[] args) {
    Stack<String> myStack = new Stack<String>();

    String infix = "( 6 + 5 * ( 2 - 8 ) / 2 )";
    StringTokenizer st = new StringTokenizer(infix);

    //isp
    HashMap<String, Integer> isp = new HashMap<String, Integer>();
    isp.put("(", 0);
    isp.put("+", 1);
    isp.put("-", 1);
    isp.put("*", 2);
    isp.put("/", 2);

    //icp
    HashMap<String, Integer> icp = new HashMap<String, Integer>();
    icp.put("+", 1);
    icp.put("-", 1);
    icp.put("*", 2);
    icp.put("/", 2);
    icp.put("(", 3);

    //출력 배열
    ArrayList<String> arr = new ArrayList<String>();

    while(st.hasMoreTokens()) {
      String token = st.nextToken();

      //토큰이 왼쪽 괄호 및 연산자
      if(token.equals("(") ||token.equals("/") ||token.equals("*")||token.equals("-")||token.equals("+")) {

        //스택이 비어있으면
        if(myStack.isEmpty()) {
          myStack.push(token);

        //스택이 비어있지 않으면
        }else {

          //우선순위 : top < token
          if(isp.get(myStack.peek()) < icp.get(token)) {
            myStack.push(token);
          }
          else {
          //우선순위 : top > token
            while(true) {
              if(isp.get(myStack.peek()) < icp.get(token)) {
                myStack.push(token);
                break;
              } //우선순위 : top < token 될때까지 stack pop
              arr.add(myStack.pop());
            }
          }
        }


      //토큰이 오른쪽 괄호
      } else if(token.equals(")")) {
        while(true) {
```

```
          if(myStack.peek().equals("(")) {
            myStack.pop();
            break;
          }
        arr.add(myStack.pop());
        }

      //토큰이 숫자
      } else {
        arr.add(token);
      }
    }
    if(!st.hasMoreTokens()) {
      while(true) {
        if(myStack.isEmpty()) {
          break;
        }
        arr.add(myStack.pop());
      }
    }

    System.out.println(arr);
//출력 : [6, 5, 2, 8, -, *, 2, /, +]
  }
}
```

```
//이제 계산
    int len = arr.size();
    Stack<Integer> sum = new Stack<Integer>();

    int i=0;
    while(i<len) {
      if(arr.get(i).equals("+")) {
        int p; int q;
        q = sum.pop();
        p = sum.pop();
        sum.add(p+q); //먼저 들어있던애가 앞으로
      }
      else if(arr.get(i).equals("-")) {
        int p; int q;
        q = sum.pop();
        p = sum.pop();
        sum.add(p-q);
      }
      else if(arr.get(i).equals("*")) {
        int p; int q;
        q = sum.pop();
        p = sum.pop();
        sum.add(p*q);
      }
      else if(arr.get(i).equals("/")) {
        int p; int q;
        q = sum.pop();
        p = sum.pop();
        sum.add(p/q);
      }
      else {
        sum.add(Integer.parseInt(arr.get(i)));
      }

      i++;
    }

    System.out.println(sum);
    // 출력 : -9
```