자료구조 실습 보고서

[제 8주]

제출일: 19.05.08
학번/이름: 201603867/조성환

1. 프로그램 설명서

1) 주요 알고리즘/ 자료구조/ 기타

stack에 입력을 순서대로 모두 적어서 순서대로 출력한다. stack은 순서대로 저장할 수 있어서 계산기 뿐 만 아니라 여러 프로그래밍에 매우 유용하다.

2) 함수 설명서

-AppController

run()
infixExpression변수에 inputExpression() 함수를 통해 값을 집어넣는다.
while문을 통해 try catch를 하는데,
while문의 조건은 입력 받은 string의 0번째 문자가 AppController의 endofcalculation인 ! 문자인지 아닌지를 검사하고 아니면 계속 while문을 실행한다.
try catch 내용은 calculator의 evalutate(infixExpression)를 실행시켜 그 결과를 result에 넣는다. 이후 AppView의 ouput을 통해 출력한다.
만약 오류가 있다면 해당 오류를 showCalculatorErrorMessage()함수를 통해 출력한다.
try catch를 실행하면 다시 infixExpression에 값을 입력받는다.
이후 !문자가 들어오면 종료 문구와 함께 종료를 한다.

inputExpression()
계산할 수식을 입력하라고 하는 출력문과 함께 AppView.inputLine()함수를 통해 입력을 받는다.

AppController()
AppController의 생성자이다. Calculator의 객체를 만들고 Appview에서 debugmode를 설정한다. 이 프로그램에서는 항상 true이다.

showCalculatorErrorMessage(CalculatorError anError)
에러 유형에 따른 메시지를 보여준다.
에러 유행은 CalculatorError Enum클래스에 있다.

-Calculator

evalutate(String anInfixExpression) throws CalculatorException
먼저 함수 설명 이전에 throws가 붙는데, 이는 오류가 생길 수 있는 부분이기 때문에 오류가 생긴다면 이 함수를 부른 해당 메소드 ,클래스에 오류가 생겼음을 알린다. 본 프로그램은 이 오류를 try catch를 통해 예외처리를 한다.
입력받은 infixExpression을 setInfixExpression()함수를 통해 Calculator class의 infixExpression에 세팅을 해준다. 이후 AppView.outputLineDebugMessage를 통해 디버깅

모드로 출력을 한다.
이후 if문이 실행되는데, 만약 infixExpression이 비어있으면 throw new CalculatorException(~_NoExpression))을 통해 아무것도 입력되지 않았음을 알린다.
CalculatorError를 infixError이름으로 만들어 infixToPostfix()의 리턴값을 넣어준다.
만약 infixError가 none의 값을 나타내면 정상적으로 실행을 한다. AppView.outputLineDebugMessage() 함수를 통해 postfixExpression을 보여주고 postfixCalculator().evaluate의 리턴값을 리턴한다. 이 리턴값은 AppController run에서 마지막으로 나타나는 result값이다.
infixError가 none 외의 값을 나타내면 thorw new CalculatorException(infixError)함수를 통해 error임을 나타낸다.

CalculatorError infixToPostfix()
postfixExpressionArray라는 infixExpression의 길이만큼의 배열을 만든다.
array.fill()통해 해당 배열을 매개변수로 채운다.
이후 3가지 token을 만든다. type은 Character이고 currentToken, poppedToken, topToken이다.

먼저 operatorStack을 clear()함수로 비운다.
for문을 통해 infixExpression의 길이만큼 돌린다.
배열의 해당 위치에 있는 요소를 currentToken에 넣는다.
진행은 2가지로 나뉜다. 숫자면 그대로 postfixExpressionArray에 추가 후 token을 프린트한다.
만약 숫자가 아니라면 문자에 해당하는 연산을 한다.

Token이 숫자면 postfixExprssionArray에 추가한다. 그리고 showTokenAndPostfixExpression()함수를 통해 입력받은 token과 현재 postfix의 식이 무엇인지 보여준다.
만약 숫자가 아니면 연산자 중 하나이므로 다음 연산으로 시작한다,
괄호가 있다면 괄호를 모두 담아야 하므로 ()에 대한 조건을 다룬다.
if문을 통해 만약) 문자이면 담는 것을 멈추고 출력을 시작한다.
왼쪽 괄호 (가 나타날 때까지 스택에서 꺼내어 출력이라는 출력문과 함께 popeedToken에 operratorStack에서 pop()한 후 token이 비거나 (문자일 때 까지 postfixExpressionArray에 담고 popped라는 출력문과 함께 출력한 문자와 postfixExpressionArray를 보여준다. 이후 (가 나올때 까지 보여준다.
만약 poppedToken이 null이면 (문자가 없는거기 때문에 calcuratorError.InfixError_MissingLeftParen을 통해 알려준다.

) 문자가 아니라면 일반 수식이므로 꺼내어 출력한다.
incomingprecedence 변수에 incomingprecedence로 우선순위를 검사한다
만약 0보다 작으면 -1, 오류이므로 unknownOperator error 리턴한다

아니라면 입력 연산자보다 순위가 높지않은 연산자를 stack에서 꺼낸다.
while문을 통해 postfix를 보여준다.
poppedToken이 비거나 안에 있는 연산자보다 우선순위가 낮을 때 까지 계속 뽑아서 postfixExpression으로 나타낸다.
우선순위를 만족하거나 topToken이없는, operatorStack에 문자가없으면 다음 연산을 한다.
만약 operatorStack이 꽉차면 ToolongExpression error를 나타내고 operatorStack.push()를 통해 해당 token을 opratorStack에 넣어준다.
showOperatorStack을 통해 pushe됨을 알려주고 stack을보여준다.
이후 infixExpression이 끝남을 알린다.
operatorStack이 안비어있다면 operatorStack에서 pop한다. 만약 (가 나온다면)괄호를 아 넣은거기 때문에 오류를 나타낸다. 뽑을때마다 showTokenAndPostfixExpression()을 통해 token과 postfixExpression을 보여준다.
마지막으로 postfixExpression을 문자로 나타내고 set을한 뒤 error가 none임을 나타낸다.

incomingprecedence(), instackprecedence token()의 우선순위를 숫자로 나타낸다, 만약 알수없다면 -1로 오류임을 알려준다.

showTokenAndPostfixExpression(), showTokenAndMessage()
token과 함께 operatorstack을 보여주거나 postfixexprssion을 보여준다.

showOperatorStack()
stack의 popall과 같은 역할을 한다. operatorStack을 bottom에서부터 top까지 elementAt()통해 순서대로 나열한다. 여기서 stack을 ArrayList<Charactor>로 강제 형변환을 해야 elementAt함수를 쓸 수 있다.

PostfixCaculator

evaluate(String aPostfixExpression) throws CalculatorException
만약 aPostfixExpression이 비어있다면 NoExpression error를 throw한다.
먼저 valueStack.clear()함수를 통해 valueStack을 비운다
token을 선언한 후
aPostfixExpression의 길이만큼 for문을 돌린다
token에 aPostfixExprssion의 current위치의 값을 넣는다
만약 token이 숫자면 valueStack에 넣는다. valueStack이 만약 isFull인 상태면 TooLongExpression error를 throw한다.
숫자가 아니면 연산자이다.
CalculatorError의 객체를 만들어 이 값에 executeBinaryOperator()의 반환값을 가져온다.
error가 None이 아니면 해당 error를 throw한다. 만약 성공적으로 연산을 하면 showTokenAndValueStack에 해당 token을 넣는다.
모든 값을 연산햇으므로 valueStack의 size는 1이어야 한다. 마지막 연산값을 리턴한다.

만약 아니라면 1개보다 많은 거므로 Toomanyvalues오류를 리턴한다.

executeBinaryOperator(char anOperator)
먼저 valueStack의 size가 2보다 작으면 TooFewValues error를 나타낸다.
첫 번째 pop을 operand1, 두 번째 pop을 operan2로 넣는다
anOperator가 무엇이냐에 따라 switch에 의해 해당 연산자에 따른 연산을 시작한다. 만약 하나라도 해당되는게 없다면 UnknownOperator error를 반환한다.
계산을 마치면 valueStack에 계산값을 넣고
Error_None을 반환한다.

showTokenAndValueStack(char aToken)
현재 valueStack의 size만큼 for문을 돌린다 현재 valueStack에서 현재 위치의 값을 출력한다. 마지막으로 Top을 통해 마지막 값을 출력했음을 알린다. 여기 역시 Stack의 종류를 모르기 때문에 ArrayList<Integer> 로 강제 형변환을 해야 elementAt()함수를 쓸 수 있다.

3) 종합 설명서

결국 stack을 이용한 계산기이다.
계산기는 infix로 먼저 계산을 해서 postfix로 바꾸는 과정에서 postfixExpressionArray에 하나씩 넣는다. infix계산이 끝나면 postfixExpressionArray에 완벽한 postfix식이 들어간다. 이를 evaluate에 넘긴다.
valueStack에 담겨 token에 따라 연산이 진행되고
마지막에 하나 남은 값을 AppController에 전달해 결과를 표기한다.

2. 프로그램 장단점/ 특이점 분석

계산기를 stack을 이용해서 구현했다. stack은 대부분의 프로그램에서 많이 사용한다. 컴파일러나 디스크 같은 경우 순서대로 저장한 값들을 임의의 순서로 다시 정렬한 후 용도에 맞게 순서를 재조정해서 임의로 값을 저장, 혹은 연산을 한다. 이런 stack은 순서대로 저장을 할 수 있으며 빼낼때도 순서가 있어서 프로그래밍시 값을 저장하기에 매우 훌륭한 자료구조이다.
이번 계산기도 역시 마찬가지다. 계산에는 순서가 중요한데, infix는 인간이 보기에 좋고 postfix는 기계가 계산하기에 좋다. 그래서 stack을 통해 infix값을 저장하고 이를 postfix로 바꾸는 과정이 필요하며, postfix의 식을 valuestack에 담아 하나씩 계산한다.

3. 실행 결과 분석

1) 입력과 출력

```
<<< 계산기 프로그램을 시작합니다. >>>

? 계산할 수식을 입력하시오 (종료하려면! 를 입력하시오): 3-5)
[Infix to Postfix] 3-5)
3 : (Postfix) 3
- : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>-  <Top>
5 : (Postfix) 35
) : (왼쪽 괄호가 나타날 때까지 스택에서 꺼내어 출력)
 : Popped [OperatorStack] <Bottom> <Top>
- : (Postfix) 35-
[오류] 왼쪽 괄호가 빠졌습니다.

? 계산할 수식을 입력하시오 (종료하려면! 를 입력하시오): 2^3^2
[Infix to Postfix] 2^3^2
2 : (Postfix) 2
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>^  <Top>
3 : (Postfix) 23
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>^ ^  <Top>
2 : (Postfix) 232
(End of infix expression: 스택에서 모든 연산자를 꺼내어 출력)
 : Popped [OperatorStack] <Bottom>^  <Top>
^ : (Postfix) 232^
 : Popped [OperatorStack] <Bottom> <Top>
^ : (Postfix) 232^^
[Evalutate Postfix]232^^
2 : ValueStack <Bottom> 2  <Top>
3 : ValueStack <Bottom> 2 3  <Top>
2 : ValueStack <Bottom> 2 3 2  <Top>
^ : ValueStack <Bottom> 2 9  <Top>
```

```
^ : ValueStack <Bottom> 512  <Top>
> 계산값: 512

? 계산할 수식을 입력하시오 (종료하려면! 를 입력하시오): (2+5+3^4)%5
[Infix to Postfix] (2+5+3^4)%5
( : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>(  <Top>
2 : (Postfix) 2
+ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>( +  <Top>
5 : (Postfix) 25
* : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>( + *  <Top>
3 : (Postfix) 253
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>( + * ^  <Top>
4 : (Postfix) 2534
) : (왼쪽 괄호가 나타날 때까지 스택에서 꺼내어 출력)
 : Popped [OperatorStack] <Bottom>( + *  <Top>
^ : (Postfix) 2534^
 : Popped [OperatorStack] <Bottom>( +  <Top>
* : (Postfix) 2534^*
 : Popped [OperatorStack] <Bottom>(  <Top>
+ : (Postfix) 2534^*+
 : Popped [OperatorStack] <Bottom> <Top>
% : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>%  <Top>
5 : (Postfix) 2534^*+5
(End of infix expression: 스택에서 모든 연산자를 꺼내어 출력)
 : Popped [OperatorStack] <Bottom> <Top>
```

```
% : (Postfix) 2534^*+5%
[Evalutate Postfix]2534^*+5%
2 : ValueStack <Bottom> 2  <Top>
5 : ValueStack <Bottom> 2 5  <Top>
3 : ValueStack <Bottom> 2 5 3  <Top>
4 : ValueStack <Bottom> 2 5 3 4  <Top>
^ : ValueStack <Bottom> 2 5 81  <Top>
* : ValueStack <Bottom> 2 405  <Top>
+ : ValueStack <Bottom> 407  <Top>
5 : ValueStack <Bottom> 407 5  <Top>
% : ValueStack <Bottom> 2  <Top>
> 계산값: 2

? 계산할 수식을 입력하시오 (종료하려면! 를 입력하시오): 8-1
[Infix to Postfix] 8-1
8 : (Postfix) 8
- : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
 : Pushed [OperatorStack] <Bottom>-  <Top>
1 : (Postfix) 81
(End of infix expression: 스택에서 모든 연산자를 꺼내어 출력)
 : Popped [OperatorStack] <Bottom> <Top>
- : (Postfix) 81-
[Evalutate Postfix]81-
8 : ValueStack <Bottom> 8  <Top>
1 : ValueStack <Bottom> 8 1  <Top>
- : ValueStack <Bottom> 7  <Top>
> 계산값: 7

? 계산할 수식을 입력하시오 (종료하려면! 를 입력하시오): !

<<< 계산기 프로그램을 종료합니다 >>>

Process finished with exit code 0
```

2) 결과 분석

오류에 대한 모든 예외처리를 했으므로 오류가 잡히며, 오류 없이 정상적인 입력에 대해서는

결과가 모두 나온다.

4. 소스 코드

Appcontroller

```
                AppView.outputLine( aMessage: "[오류] 연산자에 비해 연산값의 수가 적습니다.");
                break;
            case PostfixError_TooManyValues:
                AppView.outputLine( aMessage: "[오류] 연산자에 비해 연산값의 수가 많습니다.");
                break;
            case PostfixError_DivideByZero:
                AppView.outputLine( aMessage: "[오류] 나눗셈의 분모가 0 입니다.");
                break;
            case PostfixError_UnknownOperator:
                AppView.outputLine( aMessage: "[오류] 후위 계산식에 알 수 없는 연산자가 있습니다.");
                break;
            default:
                ;
        }
    }
    public void run(){
        AppView.outputLine( aMessage: "<<< 계산기 프로그램을 시작합니다. >>>");

        String infixExpression= this.inputExpression();
        while (infixExpression.charAt(0)!= AppController.END_OF_CALCULATION){
            try {
                int result= this.calculator().evaluate(infixExpression);
                AppView.outputLine( aMessage: "> 계산값: "+ result);
            } catch (CalculatorException exception){
                this.showCalculatorErrorMessage(exception.error());
            }
            infixExpression= this.inputExpression();
        }
```

```
        return AppView.inputLine();
    }

    private void showCalculatorErrorMessage(CalculatorError anError){
        switch (anError) {
            case InfixError_NoExpression:
                AppView.outputLine( aMessage: "[오류] 중위 계산식이 주어지지 않았습니다.");
                break;

            case InfixError_TooLongExpression:
                AppView.outputLine( aMessage: "[오류] 중위 계산식이 너무 길어 처리할 수 없습니다.");
                break;
            case InfixError_MissingLeftParen:
                AppView.outputLine( aMessage: "[오류] 왼쪽 괄호가 빠졌습니다.");
                break;
            case InfixError_MissingRightParen:
                AppView.outputLine( aMessage: "[오류] 오른쪽 괄호가 빠졌습니다.");
                break;
            case InfixError_UnknownOperator:
                AppView.outputLine( aMessage: "[오류] 중위 계산식에 알 수 없는 연산자가 있습니다.");
                break;
            case PostfixError_NoExpression:
                AppView.outputLine( aMessage: "[오류] 후위 계산식이 주어지지 않았습니다.");
                break;
            case PostfixError_TooLongExpression:
                AppView.outputLine( aMessage: "[오류] 후위 계산식이 너무 길어 처리할 수 없습니다.");
                break;
            case PostfixError_TooFewValues:
```

```java
package controller;

import ...

public class AppController {
    //constants
    private static final char END_OF_CALCULATION= '!';
    private static final boolean DEBUG_MODE= true;

    //private var
    private Calculator _calculator;

    //getter setter
    public Calculator calculator() { return _calculator; }
    public void setCalculator(Calculator newCalculator) { this._calculator = newCalculator; }

    //constructors
    public AppController(){
        this.setCalculator(new Calculator());
        AppView.setDebugMode(AppController.DEBUG_MODE);
    }

    //private methods
    private String inputExpression(){
        AppView.outputLine( aMessage: "");
        AppView.output(
                aMessage: "? 계산할 수식을 입력하시오 (종료하려면"
                + END_OF_CALCULATION+ " 를 입력하시오): "
        );
```

```java
            AppView.outputLine( aMessage: "");
            AppView.outputLine( aMessage: "<<< 계산기 프로그램을 종료합니다 >>> ");
        }
    }

```

AppView

```java
package view;

import java.util.Scanner;

public class AppView {
    //private instance var
    private static Scanner scanner= new Scanner(System.in);
    private static boolean debugMode= false;

    //constructors
    public AppView(){}

    //public methods
    //output
    public static boolean debugMode(){return debugMode;}
    public static void setDebugMode(boolean newDebugMode){
        debugMode= newDebugMode;
    }
    public static void outputDebugMessage(String aMessage){
        if (AppView.debugMode()){
            System.out.print(aMessage);
        }
    }
    public static void outputLineDebugMessage(String aMessage){
        if(AppView.debugMode()){
            System.out.println(aMessage);
        }
    }

    public static void output(String aMessage){ System.out.print(aMessage); }
```

```java
    public static void outputLine(String aMessage){ System.out.println(aMessage); }

    //input
    public static String inputLine(){
        String line= AppView.scanner.nextLine().trim();
        while (line.equals("")){
            line= AppView.scanner.nextLine().trim();
        }
        return line;
    }
}
```

Calculator

```java
package model;

import controller.CalculatorError;
import controller.CalculatorException;
import view.AppView;

import java.util.Arrays;

public class Calculator {
    //constants
    public static final int MAX_EXPRESSION_LENGTH= 100;

    //private instance var
    private Stack<Character> _operatorStack;
    private String _infixExpression;
    private String _postfixExpression;
    private PostfixCaculator _postfixCalcurator;

    //getter setter
    public Stack<Character> operatorStack() { return _operatorStack; }
    public void setOperatorStack(Stack<Character> newOperatorStack) { this._operatorStack = newOperatorStack; }

    public String infixExpression() { return _infixExpression; }
    public void setInfixExpression(String newInfixExpression) { this._infixExpression = newInfixExpression; }

    public String postfixExpression() { return _postfixExpression; }
    public void setPostfixExpression(String newPostfixExpression) { this._postfixExpression = newPostfixExpression; }

    public PostfixCaculator postfixCalcurator() { return _postfixCalcurator; }
    public void setPostfixCalcurator(PostfixCaculator newPostfixCalcurator) { this._postfixCalcurator = newPostfixCalcurator; }

    //Constructors
    public Calculator(){
        this.setOperatorStack(
                new ArrayList<Character>(Calculator.MAX_EXPRESSION_LENGTH));
        this.setPostfixCalcurator(
                new PostfixCaculator(Calculator.MAX_EXPRESSION_LENGTH));
    }

    //private methods
    private void showOperatorStack(String anOperationLabel){
        //작성
        AppView.outputDebugMessage( aMessage: " : "+anOperationLabel);
        AppView.outputDebugMessage( aMessage: " [OperatorStack] <Bottom>");
        for (int order= 0; order< this.operatorStack().size(); order++){
            AppView.outputDebugMessage( aMessage: ((ArrayList<Character>)this.operatorStack()).elementAt(order).toString()+ " ");
        }
        AppView.outputLineDebugMessage( aMessage: " <Top>");
    }
    private void showTokenAndPostfixExpression(char aToken, char[] aPostfixExpressionArray){
        AppView.outputDebugMessage( aMessage: aToken+ " : (Postfix) ");
        AppView.outputLineDebugMessage(new String(aPostfixExpressionArray));
    }
    private void showTokenAndMessage(char token, String message){
        //작성
        AppView.outputLineDebugMessage( aMessage: token+" : ("+message+")");
    }
@   private int inComingPrecedence(Character aToken){
```

```
58 @    private int inComingPrecedence(Character aToken){
59          switch (aToken.charValue()){
60              case '(': return 20; case ')': return 19;
61              case '^': return 17; case '*': return 13;
62              case '/': return 13; case '%': return 13;
63              case '+': return 12; case '-': return 12;
64              default:
65                  return -1; //알수없는 연산자
66          }
67      }
68 @    private int inStackPrecedence(Character aToken){
69          switch (aToken.charValue()){
70              case '(': return 0; case ')': return 19;
71              case '^': return 16; case '*': return 13;
72              case '/': return 13; case '%': return 13;
73              case '+': return 12; case '-': return 12;
74              default:
75                  return -1; //알수없는 연산자
76          }
77      }
78      private CalculatorError infixToPostfix(){
79          char[] postfixExpressionArray= new char[this.infixExpression().length()];
80          Arrays.fill(postfixExpressionArray,  val: ' ');
81
82          Character currentToken, poppedToken, topToken;
83          this.operatorStack().clear();
84          int p= 0;
85          for (int i= 0; i< this.infixExpression().length(); i++){
86              currentToken= this.infixExpression().charAt(i);
```

```
            if (Character.isDigit(currentToken.charValue())){
                postfixExpressionArray[p++]= currentToken;
                this.showTokenAndPostfixExpression(currentToken, postfixExpressionArray);
            } else {
                if (currentToken== ')'){
                    this.showTokenAndMessage(currentToken, message: "왼쪽 괄호가 나타날 때까지 스택에서 꺼내어 출력");
                    poppedToken= this.operatorStack().pop();
                    while (poppedToken!= null && poppedToken.charValue()!= '('){
                        postfixExpressionArray[p++]= poppedToken.charValue();
                        this.showOperatorStack( anOperationLabel: "Popped");
                        this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
                        poppedToken= this.operatorStack().pop();
                    }
                    if (poppedToken== null){
                        return CalculatorError.InfixError_MissingLeftParen;
                    }
                    this.showOperatorStack( anOperationLabel: "Popped");
                } else {
                    int inComingPrecedence= this.inComingPrecedence(currentToken.charValue());
                    if (inComingPrecedence< 0){
                        AppView.outputLineDebugMessage( aMessage: currentToken+ " : (Unknown Operator)");
                        return CalculatorError.InfixError_UnknownOperator;
                    }
                    this.showTokenAndMessage
                            (currentToken, message: "입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력");
                    topToken= this.operatorStack().peek();
                    while (topToken!= null &&
                            this.inStackPrecedence(topToken)>= inComingPrecedence){
```

```
                        poppedToken= this.operatorStack().pop();
                        postfixExpressionArray[p++]= poppedToken;
                        this.showOperatorStack( anOperationLabel: "Popped");
                        this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
                        topToken= this.operatorStack().peek();
                    }
                    if (this.operatorStack().isFull()){
                        this.showOperatorStack( anOperationLabel: "Fulled");
                        return CalculatorError.InfixError_TooLongExpression;
                    }
                    this.operatorStack().push(currentToken);
                    this.showOperatorStack( anOperationLabel: "Pushed");
                }
            }
        }
        AppView.outputLineDebugMessage( aMessage: "(End of infix expression: 스택에서 모든 연산자를 꺼내어 출력)");

        while (!this.operatorStack().isEmpty()){
            poppedToken= this.operatorStack().pop();
            this.showOperatorStack( anOperationLabel: "Popped");
            if (poppedToken== '('){
                return CalculatorError.InfixError_MissingRightParen;
            }
            postfixExpressionArray[p++]= poppedToken;
            this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
        }

        this.setPostfixExpression(new String(postfixExpressionArray).trim());
        return CalculatorError.InfixError_None;
```

```
        return CalculatorError.InfixError_None;
    }
    public int evaluate(String anInfixExpression) throws CalculatorException{
        this.setInfixExpression(anInfixExpression);
        AppView.outputLineDebugMessage( aMessage: "[Infix to Postfix] "+anInfixExpression);
        if (this.infixExpression()== null|| this.infixExpression().length()==0 ){
            throw new CalculatorException(CalculatorError.InfixError_NoExpression);
        }
        CalculatorError infixError= this.infixToPostfix();
        if (infixError== CalculatorError.InfixError_None){
            AppView.outputLineDebugMessage( aMessage: "[Evalutate Postfix]"+ this.postfixExpression());
            return this.postfixCalcurator().evaluate(this.postfixExpression());
        } else {
            throw new CalculatorException(infixError);
        }
    }
}
```

PostfixCalculator

```
package model;

import controller.CalculatorError;
import controller.CalculatorException;
import view.AppView;

public class PostfixCaculator {
    //private instance var
    private Stack<Integer> _valueStack;

    //getter setter
    public Stack<Integer> valueStack() { return _valueStack; }
    public void setValueStack(Stack<Integer> newValueStack) { this._valueStack = newValueStack; }

    //constructors
    public PostfixCaculator(int givenStackCapacity){
        setValueStack(new ArrayList<Integer>(givenStackCapacity));
    }

    //public methods
    public int evaluate(String aPosfixExpression) throws CalculatorException{
        if (aPosfixExpression== null || aPosfixExpression.length()== 0){
            throw new CalculatorException(CalculatorError.PostfixError_NoExpression);
        }
        this.valueStack().clear();
        char token;
        for (int current= 0; current< aPosfixExpression.length(); current++){
            token= aPosfixExpression.charAt(current);
            if (Character.isDigit(token)){
```

```
                int tokenValue= Character.getNumericValue(token);
                if (this.valueStack().isFull()){
                    //
                    throw new CalculatorException(CalculatorError.PostfixError_TooLongExpression);
                } else {
                    this.valueStack().push(Integer.valueOf(tokenValue));
                }
            } else {
                CalculatorError error= this.executeBinaryOperator(token);
                if (error!= CalculatorError.PostfixError_None){
                    throw new CalculatorException(error);
                }
            }
            this.showTokenAndValueStack(token);
        }
        if (this.valueStack().size()== 1){
            return (this.valueStack().pop().intValue());
        } else {
            throw new CalculatorException(CalculatorError.PostfixError_TooManyValues);
        }
    }
    private CalculatorError executeBinaryOperator(char anOperator){
        if (this.valueStack().size()< 2){
            return CalculatorError.PostfixError_TooFewValues;
        }
        int operand1= this.valueStack().pop().intValue();
        int operand2= this.valueStack().pop().intValue();
        int calculated= 0;
        switch (anOperator){
```

```
            case '^':
                calculated= (int) Math.pow((double)operand2, (double)operand1);
                break;
            case '+':
                calculated= operand2* operand1;
                break;
            case '/':
                if (operand1== 0){
                    AppView.outputLineDebugMessage
                            ( aMessage: anOperator+ " : (DivideByZero) "+ operand2+ " " + anOperator+ " "+ operand1 );
                    return CalculatorError.PostfixError_DivideByZero;
                } else {
                    calculated= operand2/ operand1;
                }
                break;
            case '%':
                if (operand1== 0){
                    AppView.outputLineDebugMessage
                            ( aMessage: anOperator+ " : (DivideByZero) "+ operand2+ " "+ anOperator+ " "+ operand1);
                    return CalculatorError.PostfixError_DivideByZero;
                } else {
                    calculated= operand2% operand1;
                }
                break;
            case '+':
                calculated= operand2+ operand1;
                break;
            case '-':
                calculated= operand2- operand1;
```

```
                calculated= operand2- operand1;
                break;
            default:
                return CalculatorError.PostfixError_UnknownOperator;
        }
        this.valueStack().push(Integer.valueOf(calculated));
        return CalculatorError.PostfixError_None;
    }

    private void showTokenAndValueStack(char aToken){
        AppView.outputDebugMessage( aMessage: aToken+ " : ValueStack <Bottom> ");
        for (int i= 0; i< this.valueStack().size(); i++){
            AppView.outputDebugMessage
                    ( aMessage: ((ArrayList<Integer>)this.valueStack()).elementAt(i).intValue()+ " ");
        }
        AppView.outputLineDebugMessage( aMessage: " <Top>");
    }
}

```