자료구조 실습 보고서

[제08주] 스택 - 수식계산

제출일 : 2021/05/03

학번/이름 : 201400875/김용준

주요 개념:

■ 보다 일반적인 계산기 프로그램을 작성하려면?

● 한 자리 이상의 정수의 입력은? 더 일반적으로, 실수 값의 입력은?

>> 입력값을 char[]로 저장을 하기 때문에 1개씩 밖에 인식을 못하지만, 입력값의 변환을 주고 입력된 값들 간의 구분만 가능하면 가능할 듯 싶다.

● 부호가 붙은 수의 입력은? 일반적으로 Unary Operator 처리는? 함수는? (삼각함수 등)

>> 자바 내의 함수를 사용하면 가능할 듯 싶다.

■ 수식의 표현:

● String 과 문자열 사이의 자료형의 변환이 필요하였다.

◆ 변환 방법은? (int)나 (String) 그리고 .trim을 사용하여 변환을 한다.

● 이러한 변환이 필요 없도록 하려면? 입력을 받을 때 값들의 자료형을 설정한다.

■ 오류 처리:

● 현재는 하나의 오류만 발생해도, 수식 계산을 멈추게 되어 있다.

● 될 수 있는 대로 많은 오류를 검사하게 하려면? 하나의 try문에 여러개의 catch를 넣는다?

● Try-Catch 사용법을 이해하자.

>> try {} - catch{} - function 구조일 때 try 구간에서 오류가 발생하면 그 후의 try쪽은 무시 catch를 실행한 후 function으로 넘어간다.

● Exception 사용법을 이해하자.

>> 예외가 발생했을 때 throws로 예외상황을 넘겨 준 다음에 try-catch에서 처리를 한다.

throw로 강제처리를 할 경우 그 즉시 처리를 한다.

출력1(일반모드)

```
<terminated> DS08_Main_201400875_감용준 [Java Application] C:₩Program Files₩Java₩jdk-16₩bin₩javaw.e
<<< 계산기 프로그램을 시작합니다 >>>

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3-8
> 계산값 : -5

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3-8+7
> 계산값 : 2

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 8/(7-5)*(9%5/2^(9-7))
> 계산값 : 4

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : ((9%7-6*8)*(2+5/3))
> 계산값 : -138

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 2^3^2
> 계산값 : 512

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : (2^3)^2
> 계산값 : 64

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : (2+5*3^4)%5
> 계산값 : 2

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : !

<<< 계산기 프로그램을 종료합니다 >>>
```

```
<terminated> DS08_Main_201400875_감용준 [Java Application] C:₩Program Files₩Java
<<< 계산기 프로그램을 시작합니다 >>>

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : ()
[오류] 후위 계산식이 주어지지 않았습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3+5)
[오류] 왼쪽 괄호가 빠졌습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : (3+5
[오류] 오른쪽 괄호가 빠졌습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3=5
[오류] 중위 계산식에 알 수 없는 연산자가 있습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3=5-
[오류] 중위 계산식에 알 수 없는 연산자가 있습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3+5-
[오류] 연산자에 비해 연산값의 수가 적습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : (3+5)2
[오류] 연산자에 비해 연산값의 수가 많습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3/(2-7%5)
[오류] 나눗셈의 분모가 0입니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : !

<<< 계산기 프로그램을 종료합니다 >>>
```

출력2(디버깅모드)

```
DS08_Main_201400875_감용준 [Java Application]
<<< 계산기 프로그램을 시작합니다 >>>

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3-8
[Infix to Postfix] 3-8
3 : (Postfix 수식으로 출력) 3
- : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
8 : (Postfix 수식으로 출력) 38
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
- : (Postfix 수식으로 출력) 38-
[Evaluate Postfix] 38-
3 : ValueStack <Bottom> 3 <Top>
8 : ValueStack <Bottom> 3 8 <Top>
- : ValueStack <Bottom> -5 <Top>
> 계산값 : -5
```

```
DS08_Main_201400875_감용준 [Java Application] C:₩Program Files₩Java₩jdk-16₩bin₩j
<<< 계산기 프로그램을 시작합니다 >>>

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3-8+7
[Infix to Postfix] 3-8+7
3 : (Postfix 수식으로 출력) 3
- : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
8 : (Postfix 수식으로 출력) 38
+ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
- : (Postfix 수식으로 출력) 38-
7 : (Postfix 수식으로 출력) 38-7
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
+ : (Postfix 수식으로 출력) 38-7+
[Evaluate Postfix] 38-7+
3 : ValueStack <Bottom> 3 <Top>
8 : ValueStack <Bottom> 3 8 <Top>
- : ValueStack <Bottom> -5 <Top>
7 : ValueStack <Bottom> -5 7 <Top>
+ : ValueStack <Bottom> 2 <Top>
> 계산값 : 2

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) :
```

```
? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 2^3^2
[Infix to Postfix] 2^3^2
2 : (Postfix 수식으로 출력) 2
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
3 : (Postfix 수식으로 출력) 23
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
2 : (Postfix 수식으로 출력) 232
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
^ : (Postfix 수식으로 출력) 232^
^ : (Postfix 수식으로 출력) 232^^
[Evaluate 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) : (2^3)^2
[Infix to Postfix] (2^3)^2
( : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
2 : (Postfix 수식으로 출력) 2
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
3 : (Postfix 수식으로 출력) 23
) : (왼쪽 괄호가 나타날때까지 스택에서 꺼내어 출력)
^ : (Postfix 수식으로 출력) 23^
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
2 : (Postfix 수식으로 출력) 23^2
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
^ : (Postfix 수식으로 출력) 23^2^
[Evaluate Postfix] 23^2^
2 : ValueStack <Bottom> 2 <Top>
3 : ValueStack <Bottom> 2 3 <Top>
^ : ValueStack <Bottom> 8 <Top>
2 : ValueStack <Bottom> 8 2 <Top>
^ : ValueStack <Bottom> 64 <Top>
> 계산값 : 64
```

```
? 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)
2 : (Postfix 수식으로 출력) 2
+ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
5 : (Postfix 수식으로 출력) 25
* : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
3 : (Postfix 수식으로 출력) 253
^ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
4 : (Postfix 수식으로 출력) 2534
) : (왼쪽 괄호가 나타날때까지 스택에서 꺼내어 출력)
^ : (Postfix 수식으로 출력) 2534^
* : (Postfix 수식으로 출력) 2534^*
+ : (Postfix 수식으로 출력) 2534^*+
% : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
5 : (Postfix 수식으로 출력) 2534^*+5
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
% : (Postfix 수식으로 출력) 2534^*+5%
[Evaluate 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) : 3+5)
[Infix to Postfix] 3+5)
3 : (Postfix 수식으로 출력) 3
+ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
5 : (Postfix 수식으로 출력) 35
) : (왼쪽 괄호가 나타날때까지 스택에서 꺼내어 출력)
+ : (Postfix 수식으로 출력) 35+
[오류] 왼쪽 괄호가 빠졌습니다.

? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : (3+5
[Infix to Postfix] (3+5
( : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
3 : (Postfix 수식으로 출력) 3
+ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
5 : (Postfix 수식으로 출력) 35
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
+ : (Postfix 수식으로 출력) 35+
[오류] 오른쪽 괄호가 빠졌습니다.
```

```
? 계산할 수식을 입력하시오 (종료하려면 ! 를 입력하시오) : 3/(2-7%5)
[Infix to Postfix] 3/(2-7%5)
3 : (Postfix 수식으로 출력) 3
/ : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
( : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
2 : (Postfix 수식으로 출력) 32
- : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
7 : (Postfix 수식으로 출력) 327
% : (입력 연산자보다 순위가 높지 않은 연산자를 스택에서 꺼내어 출력)
5 : (Postfix 수식으로 출력) 3275
) : (왼쪽 괄호가 나타날때까지 스택에서 꺼내어 출력)
% : (Postfix 수식으로 출력) 3275%
- : (Postfix 수식으로 출력) 3275%-
(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)
/ : (Postfix 수식으로 출력) 3275%-/
[Evaluate Postfix] 3275%-/
3 : ValueStack <Bottom> 3 <Top>
2 : ValueStack <Bottom> 3 2 <Top>
7 : ValueStack <Bottom> 3 2 7 <Top>
5 : ValueStack <Bottom> 3 2 7 5 <Top>
% : ValueStack <Bottom> 3 2 2 <Top>
- : ValueStack <Bottom> 3 0 <Top>
/ : (DivideByZero) 3 / 0[오류] 나눗셈의 분모가 0입니다.
```

소스코드

Main

```
public class DS08_Main_201400875_감용준 {
	public static void main (String[] args) {
	AppController appController = new AppController() ;
	// AppController 가 실질적인 main class 이다.

	appController.run() ;
	// 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
	}
}
```

AppView

```
import java.util.Scanner;

public class AppView {
	private static Scanner scanner = new Scanner(System.in);
	private static boolean debugMode = true;

	private AppView() {}

	public static void output(String message) {
			System.out.print(message);
	}
	public static void outputLine(String message) {
			System.out.println(message);
	}
	public static boolean debugMode() {
		return true;
	}
	public static void setDebugMode(boolean newDebugMode) {
		if(AppView.debugMode()) {
		}
	}
	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 String inputLine() {
		String line = AppView.scanner.nextLine().trim();
		while(line.equals("")) {
			line = AppView.scanner.nextLine().trim();
		}
		return line;
	}
}
```

AppController

```
public class AppController {
	private static final char END_OF_CALCULATION = '!';
	private static final boolean DEBUG_MODE = true;
	// 비공개 변수들
	private Calculator _calculator;

	// Getter/Setter
	private Calculator calculator() {
		return this._calculator;
	}
	private void setCalculator(Calculator newCalculator) {
		this._calculator = newCalculator;
	}
	// 생성자
	public AppController() {
		this.setCalculator(new Calculator());
		AppView.setDebugMode(AppController.DEBUG_MODE);
	}

	private String inputExpression() {
```

```
                AppView.outputLine("");
                AppView.output("? 계산할 수식을 입력하시오 (종료하려면 " + END_OF_CALCULATION + " 를 입력하시오) : ");
                return AppView.inputLine();
        }

        private void showCalculatroErrorMessage(CalculatorError anError) {
                switch(anError) {
                case InfixError_NoExpression:
                        AppView.outputLine("[오류] 중위 계산식이 주어지지 않았습니다.");
                        break;
                case InfixError_TooLongExpression:
                        AppView.outputLine("[오류] 중위 계산식이 너무 길어 처리할 수 없습니다.");
                        break;
                case InfixError_MissingLeftParen:
                        AppView.outputLine("[오류] 왼쪽 괄호가 빠졌습니다.");
                        break;
                case InfixError_MissingRightParen:
                        AppView.outputLine("[오류] 오른쪽 괄호가 빠졌습니다.");
                        break;
                case InfixError_UnknownOperator:
                        AppView.outputLine("[오류] 중위 계산식에 알 수 없는 연산자가 있습니다.");
                        break;
                case PostfixError_NoExpression:
                        AppView.outputLine("[오류] 후위 계산식이 주어지지 않았습니다.");
                        break;
                case PostfixError_TooLongExpression:
                        AppView.outputLine("[오류] 후위 계산식이 너무 길어 처리할 수 없습니다.");
                        break;
                case PostfixError_TooFewValues:
                        AppView.outputLine("[오류] 연산자에 비해 연산값의 수가 적습니다.");
                        break;
                case PostfixError_TooManyValues:
                        AppView.outputLine("[오류] 연산자에 비해 연산값의 수가 많습니다.");
                        break;
                case PostfixError_DivideByZero:
                        AppView.outputLine("[오류] 나눗셈의 분모가 0입니다.");
                        break;
                case PostfixError_UnknownOperator:
                        AppView.outputLine("[오류] 후위 계산식에 알 수 없는 연산자가 있습니다.");
                        break;
                default:
                        ;
                }
        }

        public void run() {
                AppView.outputLine("<<< 계산기 프로그램을 시작합니다 >>>");

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

Calculator

```
import java.util.Arrays;

public class Calculator {

        private static final int MAX_EXPRESSION_LENGTH = 100;

        private Stack<Character> _operatorStack;
        private String _infixExpression;
        private String _postfixExpression;
```

```
private PostfixCalculator _postfixCalculator;

private String infixExpression() {
        return this._infixExpression;
}
private void setInfixExpression(String newInfixExpression) {
        this._infixExpression = newInfixExpression;
}
private String postfixExpression() {
        return this._postfixExpression;
}
private void setPostfixExpression(String newPostfixExpression) {
        this._postfixExpression = newPostfixExpression;
}
private PostfixCalculator postfixCalculator() {
        return this._postfixCalculator;
}
private void setPostfixCalculator(PostfixCalculator newPostfixCalculator) {
        this._postfixCalculator = newPostfixCalculator;
}
private Stack<Character> operatorStack() {
        return this._operatorStack;
}
private void setOperatorStack(Stack<Character> newOperatorStack) {
        this._operatorStack = newOperatorStack;
}

public Calculator() {
        this.setOperatorStack(new ArrayList<Character>(Calculator.MAX_EXPRESSION_LENGTH));
        this.setPostfixCalculator(new PostfixCalculator(Calculator.MAX_EXPRESSION_LENGTH));
}
private void showOperatorStack (String anOperationLabel) {

}
private void showTokenAndPostfixExpression(char aToken, char[] aPostfixExpressionArray) {
        AppView.outputDebugMessage(aToken + " : (Postfix 수식으로 출력) ");
        AppView.outputLineDebugMessage(new String(aPostfixExpressionArray));
}
private void showTokenAndMessage(char aToken, String aMessage) {
        AppView.outputLineDebugMessage(aToken + " : (" + aMessage + ") ");
}
private int inComingPrecedence(Character aToken) {
        switch(aToken.charValue()) {
        case '(' : return 20;
        case ')' : return 19;
        case '^' : return 17;
        case '*' : return 13;
        case '/' : return 13;
        case '%' : return 13;
        case '+' : return 12;
        case '-' : return 12;
        default :
                return -1;
        }
}
private int inStackPrecedence(Character aToken) {
        switch(aToken.charValue()) {
        case '(' : return 0;
        case ')' : return 19;
        case '^' : return 16;
        case '*' : return 13;
        case '/' : return 13;
        case '%' : return 13;
        case '+' : return 12;
        case '-' : return 12;
        default :
                return -1;
        }
}

private CalculatorError infixToPostfix() {
        char[] postfixExpressionArray = new char[this.infixExpression().length()];
        Arrays.fill(postfixExpressionArray,' ');
```

```
		Character currentToken, poppedToken, topToken;
		this.operatorStack().clear();
		int p = 0;
		for(int i = 0; i < this.infixExpression().length(); i++) {
			currentToken = this.infixExpression().charAt(i);
			if((Character.isDigit(currentToken.charValue()))) {
				postfixExpressionArray[p++] = currentToken;
				this.showTokenAndPostfixExpression(currentToken, postfixExpressionArray);
			}
			else {
				if(currentToken == ')') {
					this.showTokenAndMessage(currentToken, "왼쪽 괄호가 나타날때까지 스택에서 꺼내어 출력");
					poppedToken = this.operatorStack().pop();
					while(poppedToken != null && poppedToken.charValue() != '(') {
						postfixExpressionArray[p++] = poppedToken.charValue();
						this.showOperatorStack("Popped");
						this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
						poppedToken = this.operatorStack().pop();
					}
					if(poppedToken == null) {
						return CalculatorError.InfixError_MissingLeftParen;
					}
					this.showOperatorStack("Popped");
				}
				else {
					int inComingPrecedence = this.inComingPrecedence(currentToken.charValue());
					if(inComingPrecedence < 0) {
						AppView.outputLineDebugMessage(currentToken + " : (Unknown Operator)");
						return CalculatorError.InfixError_UnknownOperator;
					}
					this.showTokenAndMessage(currentToken, "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("Popped");
						this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
						topToken = this.operatorStack().peek();
					}
					if(this.operatorStack().isFull()) {
						this.showOperatorStack("Fulled");
						return CalculatorError.InfixError_TooLongExpression;
					}
					this.operatorStack().push(currentToken);
					this.showOperatorStack("Pushed");
				}
			}
		}
		AppView.outputLineDebugMessage("(End of infix Expression : 스택에서 모든 연산자를 꺼내어 출력)");
		while(!this.operatorStack().isEmpty()) {
			poppedToken = this.operatorStack().pop();
			this.showOperatorStack("Popped");
			if(poppedToken == '(') {
				return CalculatorError.InfixError_MissingRightParen;
			}
			postfixExpressionArray[p++] = poppedToken;
			this.showTokenAndPostfixExpression(poppedToken, postfixExpressionArray);
		}
		this.setPostfixExpression(new String(postfixExpressionArray).trim());
		return CalculatorError.InfixError_None;
	}

	public int evaluate(String anInfixExpression) throws CalculatorException {
		this.setInfixExpression(anInfixExpression);
		AppView.outputLineDebugMessage("[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("[Evaluate Postfix] " + this.postfixExpression());
			return this.postfixCalculator().evaluate(this.postfixExpression());
		}
```

```
                else {
                        throw new CalculatorException(infixError);
                }
        }
}

PostfixCalculator
public class PostfixCalculator {
        private static final int DEFAULT_MAX_EXPRESSION_LENGTH = 100 ;
        // 인스턴스 변수
        private int _maxExpressionLength;
        private Stack<Integer> _valueStack;
        // Getter/Setter

        public int maxExpressionLength() {
                return this._maxExpressionLength;
        }
        private void setMaxExpressionLength (int newMaxExpressionLength) {
                this._maxExpressionLength = newMaxExpressionLength;
        }
        private Stack<Integer> valueStack() {
                return this._valueStack;
        }
        private void setValueStack (Stack<Integer> newValueStack) {
                this._valueStack = newValueStack;
        }
        public PostfixCalculator() {
                this (PostfixCalculator.DEFAULT_MAX_EXPRESSION_LENGTH);
        }
        public PostfixCalculator(int givenMaxExpressionLength) {
                this.setMaxExpressionLength (givenMaxExpressionLength);
                this.setValueStack(new ArrayList<Integer>(this.maxExpressionLength()));
        }
        public int evaluate(String aPostfixExpression) throws CalculatorException {
                if(aPostfixExpression == null || aPostfixExpression.length() == 0) {
                        throw new CalculatorException(CalculatorError.PostfixError_NoExpression);
                }
                this.valueStack().clear();
                char token;
                for(int current = 0; current < aPostfixExpression.length(); current++) {
                        token = aPostfixExpression.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.outputDebugMessage(anOperator + " : (DivideByZero) " + operand2 + " " + anOperator + " " +
operand1);
                                return CalculatorError.PostfixError_DivideByZero;
                        }
                        else {
                                calculated = operand2 / operand1;
                        }
                        break;
                case '%' :
                        if(operand1 == 0) {
                                AppView.outputDebugMessage(anOperator + " : (DivideByZero) " + operand2 + " " + anOperator + " " +
operand1);
                                return CalculatorError.PostfixError_DivideByZero;
                        }
                        else {
                                calculated = operand2 % operand1;
                        }
                        break;
                case '+' :
                        calculated = operand2 + operand1;
                        break;
                case '-' :
                        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(aToken + " : ValueStack <Bottom> ");
                for(int i = 0; i < this.valueStack().size(); i++) {
                        AppView.outputDebugMessage(((ArrayList<Integer>)this.valueStack()).elementAt(i).intValue() + " ");
                }
                AppView.outputLineDebugMessage("<Top>");
        }
}

CalculatorException
public class CalculatorException extends Exception {
        /**
         *
         */
        private static final long serialVersionUID = -4220992234192929051L;

        private CalculatorError _error;

        public CalculatorError error() {
                return this._error;
        }
        public void setError(CalculatorError newError) {
                this._error = newError;
        }
        public CalculatorException() {
                this.setError(CalculatorError.Undefined);
        }
        public CalculatorException(CalculatorError givenError) {
                this.setError(givenError);
        }
}
```

CalculatorError

```
public enum CalculatorError {
	Undefined,

	InfixError_None,
	InfixError_NoExpression,
	InfixError_TooLongExpression,
	InfixError_MissingLeftParen,
	InfixError_MissingRightParen,
	InfixError_UnknownOperator,

	PostfixError_None,
	PostfixError_NoExpression,
	PostfixError_TooLongExpression,
	PostfixError_TooFewValues,
	PostfixError_TooManyValues,
	PostfixError_DivideByZero,
	PostfixError_UnknownOperator,
}
```

Stack

```
public interface Stack<E> {
	public int size();
	public boolean isFull();
	public boolean isEmpty();

	public boolean push(E anElement);
	public E pop();
	public E peek();

	public void clear();
}
```

ArrayList

```
public class ArrayList<E extends Comparable<E>> implements Stack<E> {

	private static final int DEFAULT_CAPACITY = 5;

	private int _capacity;
	private int _size;
	private E[] _elements;

	public int capacity() {
		return this._capacity;
	}
	private void setCapacity(int newCapacity) {
		this._capacity = newCapacity;
	}
	@Override
	public int size() {
		return this._size;
	}
	private void setSize(int newSize) {
		this._size = newSize;
	}
	private E[] elements() {
		return this._elements;
	}
	private void setElements(E[] newElements) {
		this._elements = newElements;
	}

	public ArrayList() {
		this (ArrayList.DEFAULT_CAPACITY);
	}
	@SuppressWarnings("unchecked")
	public ArrayList(int givenCapacity) {
		this.setCapacity(givenCapacity);
		this.setElements((E[]) new Comparable[this.capacity()]);
	}

	private void makeRoomAt(int aPosition) {
		for(int i = this.size(); i > aPosition; i--) {
			this.elements()[i] = this.elements()[i-1];
```

```
		}
	}
	private void removeGapAt(int aPosition) {
		for(int i = aPosition + 1; i < this.size(); i++) {
			this.elements()[i-1] = this.elements()[i];
		}
		this.elements()[this.size()-1] = null;
	}

	@Override
	public boolean isEmpty() {
		return (this._size == 0);
	}
	@Override
	public boolean isFull() {
		return (this._size == this._capacity);
	}
	@Override
	public boolean push(E anElement) {
		return this.addToLast(anElement);
	}
	@Override
	public E pop() {
		return this.removeLast();
	}
	@Override
	public E peek() {
		if (this.isEmpty()) {
			return null;
		}
		else {
			return this.elementAt(this.size()-1); // Last element
		}
	}
	@Override
	public void clear() {
	}

	public boolean doesContain(E anElement) {
		return(this.orderOf(anElement) >= 0);
	}
	public int orderOf(E anElement) {
		int order = -1;
		for(int index = 0; index < this.size() && order < 0; index++) {
			if(this.elements()[index].equals(anElement)) {
				order = index;
			}
		}
		return order;
	}
	public E elementAt(int anOrder) {
		if(anOrder < 0 || anOrder >= this.size()) {
			return null;
		}
		else {
			return this.elements()[anOrder];
		}
	}
	public void setElementAt(int anOrder, E anElement) {
		if(anOrder < 0 || anOrder >= this.size()) {
			return ;
		}
		else {
			this.elements()[anOrder] = anElement;
		}
	}
	public boolean addTo(int anOrder, E anElement) {
		if(this.isFull()) {
			return false;
		}
		else if(anOrder < 0 || anOrder > this.size()) {
			return false;
		}
		else {
```

```
                        this.makeRoomAt(anOrder);
                        this.elements()[anOrder] = anElement;
                        this.setSize(this.size()+1);
                        return true;
                }
        }
        public boolean addToFirst(E anElement) {
                if(this.isFull()) {
                        return false;
                }
                else {
                        this.makeRoomAt(0);
                        this.elements()[0] = anElement;
                        this.setSize(this.size()+1);
                        return true;
                }
        }
        public boolean addToLast(E anElement) {
                if(this.isFull()) {
                        return false;
                }
                else {
                        this.elements()[this.size()] = anElement;
                        this.setSize(this.size()+1);
                        return true;
                }
        }
        public E removeFrom(int anOrder) {
                if(anOrder < 0 || anOrder >= this.size()) {
                        return null;
                }
                else {
                        E removedElement = this.elements()[anOrder];
                        this.removeGapAt(anOrder);
                        this.setSize(this.size()-1);
                        return removedElement;
                }
        }
        public E removeFirst() {
                if(this.isEmpty()) {
                        return null;
                }
                else {
                        E removedElement = this.elements()[0];
                        this.removeGapAt(0);
                        this.setSize(this.size()-1);
                        return removedElement;
                }
        }
        public E removeLast() {
                if(this.isEmpty()) {
                        return null;
                }
                else {
                        E removedElement = this.elements()[this.size()-1];
                        this.setSize(this.size()-1);
                        return removedElement;
                }
        }
}
```