과목 : 자료구조(가반)

교수 : 신용태 교수

이름 : 김병준

학번 : 20162448

1. 수식 계산
    - Result

```
====================== Stack Calculator ======================
 >> (150+60/2)*2+(78-20+60)+1
Postfix : 150 60 2 / + 2 * 78 20 - 60 + + 1 +
Result : 479.0
 >> 3 + 5 * 2
Postfix : 3 5 2 * +
Result : 13.0
 >> 123 + 456
Postfix : 123 456 +
Result : 579.0
 >> (13-22)/94-22-33
Postfix : 13 22 - 94 / 22 - 33 -
Result : -55.09574468085106
 >> exit

Process finished with exit code 255
```

2. 구문 검사

- Result

```
====================== Stack Calculator ======================
 >> 123(((3 + 2))
Exception in thread "main" java.lang.Exception: syntaxCondition(): '('Syntax error
    at StackCalculator.syntaxCondition(StackCalculator.java:227)
    at StackCalculator.main(StackCalculator.java:45)

Process finished with exit code 1
====================== Stack Calculator ======================
 >> 3 + a / 2
Exception in thread "main" java.lang.Exception: syntaxCondition(): There is not number in formular
    at StackCalculator.syntaxCondition(StackCalculator.java:212)
    at StackCalculator.main(StackCalculator.java:43)
====================== Stack Calculator ======================
 >> 12 = 33
Exception in thread "main" java.lang.Exception: syntaxCondition(): There is not number or unavailable character in formular
    at StackCalculator.syntaxCondition(StackCalculator.java:212)
    at StackCalculator.main(StackCalculator.java:43)

Process finished with exit code 1
```

3. 소스코드 (StackCalculator.java)

```
import java.util.Scanner;

abstract class Stack {
	int top = -1;
	int size = 100;
	boolean isEmpty() {
		return top == -1;
	}
}

class Operator extends Stack {
	private char[] stack = new char[size];
	void push(char op) {
		stack[++top] = op;
	}
	char pop() {
		return stack[top--];
	}
}

class Operand extends Stack {
	private double[] stack = new double[size];

	void push(double num) {
		stack[++top] = num;
	}
	double pop() {
		return stack[top--];
	}
}

public class StackCalculator {
	public static void main(String[] args) throws Exception {
		System.out.println("====================== Stack Calculator ======================");
		while (true) {
			Scanner scanner = new Scanner(System.in);
			System.out.print(" >> ");
			String expression = scanner.nextLine();
			if (expression.equals("exit")) System.exit(-1);
		       expression = Preprocessor(expression);
			if (syntaxCondition(expression)) {
				String post_expression = transPostfix(expression);
				double result = calculate(post_expression);
				System.out.println("Postfix : " + post_expression);
				System.out.println("Result : " + result);
			}
		}
	}

	private static String Preprocessor(String expression) {
		String[] split_expression = expression.split(" ");
		StringBuilder fit_expression = new StringBuilder();
		for(String exp : split_expression) {
```

```
            fit_expression.append(exp);
        }
        return fit_expression.toString();
    }

    private static String transPostfix(String expression) {
        StringBuilder post_expression = new StringBuilder();
        Operator operator = new Operator();
        for (int i = 0; i < expression.length(); i++) {
            char exp = expression.charAt(i);
            switch (exp) {
                case '(':
                case '{':
                case '[':
                    operator.push(exp);
                    break;
                case ')':
                case '}':
                case ']':
                    while (true) {
                        char op = operator.pop();
                        if (!(op == '(') && !(op == '{') && !(op == '[')) {
                            post_expression.append(op).append(" ");
                        } else {
                            break;
                        }
                    }
                    break;
                case '+':
                case '-':
                    while (true) {
                        if (operator.isEmpty()) {
                            break;
                        }
                        char op = operator.pop();
                        if (op == '+' || op == '-' || op == '*' || op == '/') {
                            post_expression.append(op).append(" ");
                        } else {
                            operator.push(op);
                            break;
                        }
                    }
                    operator.push(exp);
                    break;
                case '*':
                case '/':
                    while (true) {
                        if (operator.isEmpty()) {
                            break;
                        }
                        char op = operator.pop();
                        if (op == '*' || op == '/') {
                            post_expression.append(op).append(" ");
                        } else {
                            operator.push(op);
                            break;
```

```
                        }
                    }
                    operator.push(exp);
                    break;
                default:
                    post_expression.append(exp);
                    if(expression.length() != i + 1) {
                        char op = expression.charAt(i + 1);
                        if (op == '+' || op == '-' || op == '*' || op == '/'
                                || op == ')' || op == '}' || op == ']') {
                            post_expression.append(" ");
                        }
                    }
                    break;
            }
        }
        while (!operator.isEmpty()) {
            post_expression.append(" ").append(operator.pop());
        }
        return post_expression.toString();
    }

    private static Double calculate(String post_expression) {
        Operand operand = new Operand();
        String[] expressionArr = post_expression.split(" ");
        for (String exp : expressionArr) {
            try {
                double number = Double.parseDouble(exp);
                operand.push(number);
            } catch (NumberFormatException e) {
                double op1 = operand.pop();
                double op2 = operand.pop();
                switch (exp) {
                    case "+":
                        operand.push(op2 + op1);
                        break;
                    case "-":
                        operand.push(op2 - op1);
                        break;
                    case "*":
                        operand.push(op2 * op1);
                        break;
                    case "/":
                        operand.push(op2 / op1);
                        break;
                }
            }
        }
        return operand.pop();
    }

    private static boolean syntaxCondition(String expression) throws Exception {
        Operator operator = new Operator();
        int number_count = 0;
        for (int i = 0; i < expression.length(); i++) {
            char exp = expression.charAt(i);
```

```
                switch (exp) {
                    case '+':
                    case '-':
                    case '*':
                    case '/':
                        char op = expression.charAt(i + 1);
                        if (op == '+' || op == '-' || op == '*' || op == '/') {
                            throw new Exception("syntaxCondition(): Operator error");
                        }
                        break;
                    case '(':
                    case '{':
                    case '[':
                        operator.push(exp);
                        break;
                    case ')':
                        if (operator.isEmpty()) {
                            throw new Exception("syntaxCondition(): Syntax error");
                        } else {
                            if (!(operator.pop() == '(')) {
                                throw new Exception("syntaxCondition(): Syntax error");
                            }
                        }
                        break;
                    case '}':
                        if (operator.isEmpty()) {
                            throw new Exception("syntaxCondition(): Syntax error");
                        } else {
                            if (!(operator.pop() == '{')) {
                                throw new Exception("syntaxCondition(): Syntax error");
                            }
                        }
                        break;
                    case ']':
                        if (operator.isEmpty()) {
                            throw new Exception("syntaxCondition(): Syntax error");
                        } else {
                            if (!(operator.pop() == '[')) {
                                throw new Exception("syntaxCondition(): Syntax error");
                            }
                        }
                        break;
                    default:
                            if(exp < '0' || exp > '9')
                                throw new Exception("syntaxCondition(): There is not number or
unavailable character in formular");
                            number_count++;
                        break;
                }
            }
            if (number_count == 0 || number_count == 1) {
                throw new Exception("syntaxCondition(): There is no number or unavailable in formular");
            }
            while (!operator.isEmpty()) {
                char op = operator.pop();
                if (op == '+' || op == '-' || op == '*' || op == '/'
```

```
                    || op == '(' || op == '{' || op == '[') {
                throw new Exception("syntaxCondition(): '" + op + "'Syntax error");
            }
        }
        return true;
    }
}
```