자료구조 실습 보고서

[제08주] 스택 : 기본기능

2021년 04월 26일

201702039 오명주

1. 프로그램 설명서

(1) 프로그램의 전체 설계 구조

➔ MVC (Model – View – Controller) 구조

Model : 프로그램이 "무엇"을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 "보여주기 위한" 역할. 최종 사용자에게 "무엇"을 화면으로 보여줌.

Controller : 모델이 "어떻게" 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

➔ 스택 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- Stack(Interface) : 스택 변수와 추상 메소드를 구성한다.
- ArrayList<T> : 스택 배열 기능을 구성한다.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

- AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.

(2) 함수 설명서

➔ 주요 알고리즘

(1) Push

```
// 스택 수행 관련
private void pushToStack(char aCharForPush) {
    if (this.stack().isFull()) { // 스택이 가득 차있다면
        AppView.outputLine("(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다."); // 오류 출력문 출력
    } else {
        Character charObjectForAdd = Character.valueOf(aCharForPush); // 문자를 문자 객체로 변환
        if (this.stack().push(charObjectForAdd)) { // 스택에 push
            AppView.outputLine("[Push] 삽입된 원소는 '" + aCharForPush + "' 입니다 .");
        } else {
            AppView.outputLine("(오류) 스택에 넣는 동안에 오류가 발생하였습니다.");
        }
    }

}
```

Stack Interface에 정의된 추상 메소드를 ArrayList class에서 구현한다. Stack의 Top에 원소를 삽입하는 형태이며, 배열에서는 인덱스 size부분에 삽입하는 것에 해당한다.

- 문자 추가를 위해 aCharForPush의 char 형 문자를 입력 받는다.
- Stack이 가득 차 있다면 오류 출력문을 출력한다.
- 그것이 아니라면, aCharForPush 문자를 문자 객체로 변환하여 저장한다.
- Stack의 push 함수를 이용하여 삽입한다. 성공하면 true를 false라면 오류를 출력한다.

```
@Override
public boolean push(E anElement) {
    return this.addToLast(anElement); // 배열 마지막에 add
}
```

배열의 마지막에 원소를 추가하는 Stack을 구현한 ArrayList의 push 함수이다. addToLast 함수를 이용하여 배열의 마지막에 원소를 삽입한다.

(2) PopOne

```
private void popOne() { // 스택 Top의 원소를 하나 삭제하여 return하는 함수
    if (this.stack().isEmpty()) { // 스택이 비어있다면
        AppView.outputLine("[Pop.Empty] 스택에 삭제할 원소가 없습니다."); // 원소가 없다는 출력문 출력
    } else {
        Character poppedChar = this.stack().pop(); // pop한 결과를 poppedChar에 저장
        if (poppedChar == null) { // poppedChar에 대한 null 검사
            AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");
        } else {
            AppView.outputLine("[Pop] 삭제된 원소는 ' " + poppedChar + " ' 입니다.");
        }
    }
}
```

스택 Top의 원소를 하나 삭제하여 반환하는 함수이다.

- 먼저 스택이 비어있다면 오류 출력문을 출력한다.
- 그게 아니라면, Stack의 pop 함수를 이용하여 반환 받은 Top 원소를 poppedChar에 저장한다.
- poppedChar이 null이라면 오류 출력문을 출력한다.
- null이 아니라면 삭제된 원소에 대한 정보를 출력한다.

```
@Override
public E pop() {
    return this.removeLast(); // 배열 마지막 원소 remove
}
```

ArrayList의 pop 함수이다. removeLast 함수를 이용하여 구현하였다. Stack의 Top의 인덱스가 배열에서는 size 부분이기 때문에 마지막 원소를 삭제하는 removeLast를 이용한다.

(3) PopN

```
private void popN(int numberOfCharsToBePopped) { // 스택 Top의 원소를 numberOfCharsToBePopped만큼 삭제하여 return하는 함수
    if (numberOfCharsToBePopped == 0) { // 만약 입력받은 수가 0이라면
        AppView.outputLine("[Pops] 삭제할 원소의 개수가 0 개 입니다."); // 삭제횟수가 0
    } else {
        int count = 0;
        while (count < numberOfCharsToBePopped && (!this.stack().isEmpty())) { // 입력받은 수 만큼 반복
            Character poppedChar = this.stack().pop(); // 반복하여 pop을 수행
            if (poppedChar == null) {
                AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");

            } else {
                AppView.outputLine("[Pops] 삭제된 원소는 ' " + poppedChar + " '입니다.");
            }
            count++;
        }
        if (count < numberOfCharsToBePopped) { // 스택에 있는 원소의 수보다 입력받은 숫자가 더 큰 경우
            AppView.outputLine("[Pops.Empty] 스택에 더 이상 삭제할 원소가 없습니다."); // 오류문 출력
        }
    }
}
```

스택 Top의 원소를 입력 받은 수만큼 반복하여 삭제하여 반환하는 함수이다.

- 입력 받은 numberOfCharsToBePopped 만큼 반복하여 스택의 Top원소를 삭제한다.
- 인자가 0이면 예외처리한다.
- 입력 받은 수만큼 반복 && 스택이 empty가 아닌 경우 반복한다.
- Stack의 pop 함수를 이용하여 반환 받은 원소를 poppedChar 변수에 저장한다.
- poppedChar이 null이라면 오류문을 출력한다.
- 그게 아니라면 반복해서 poppedChar 변수에 대해 출력한다.
- 만약 입력 받은 수가 스택에 있는 원소의 수보다 크다면 오류문을 출력하기 위해 조건문을 이용하여 선언한다.

⇨ pop에 대한 설명은 popOne 함수 설명을 참조

(3) 종합 설명서

➔ 프로그램 실행 순서대로 설명해보자.

```
public class _DS08_201702039_오명주 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        AppController appController = new AppController() ;
        // AppController가 실질적인 main class 이다
        appController.run() ;
        //여기 main() 에서는 앱 실행이 시작되도록 해주는 일이 전부이다

    }

}
```

main에서 AppController 의 객체를 생성하여 run 한다. 프로그램을 실행한다.

```
public void run() {
    AppView.outputLine("<<< 스택 기능 확인 프로그램을 시작합니다 >>>");
    AppView.outputLine("");

    char input = this.inputChar(); // char 문자 입력받음
    while (input != '!') { // '!'를 입력받으면 종료
        this.countInputChar(); // 입력된문자 + 1
        if (Character.isAlphabetic(input)) { // 알파벳인지 검사
            this.pushToStack(input);
            this.countPushedChar();
        } else if ((Character.isDigit(input))) { // 숫자 문자인지 검사
            this.popN(Character.getNumericValue(input)); // 숫자 문자를 정수값으로 변환
        } else if (input == '-') { // '-'문자 입력받으면 popOne 실행
            this.popOne();
        } else if (input == '#') { // '#'문자 입력받으면 스택크기 출력
            this.showStackSize();
        } else if (input == '/') { // '/'문자 입력받으면 Bottom-Top까지의 스택 출력
            this.showAllFromBottom();
        } else if (input == '\\') { // '\'문자 입력받으면 Top-Bottom까지의 스택 출력
            this.showAllFromTop();
        } else if (input == '^') { // '^'문자 입력받으면 Top원소 peek하여 출력
            this.showTopElement();
        } else {
            AppView.outputLine("[Ignore] 의미 없은 문자가 입력되었습니다."); // 오류처리
            this.countIgnoredChar(); // 무시된문자 + 1
        }
        input = this.inputChar(); // 반복해서 실행
    }
    this.quitStackProcessing(); // 스택 비우는 함수
    this.showStatistics();
    AppView.outputLine("");
    AppView.outputLine("<<< 스택 기능 확인 프로그램을 종료합니다 >>> ");
}
```

- 먼저 수행할 문자를 inputChar 함수를 이용하여 입력 받는다.
- '!' 를 입력 받으면 종료한다. (종료조건)
- 입력 받은 문자에 따라 각각 다른 수행 처리를 해준다. 문자를 입력 받으면 countInputChar 를 +1 하여 입력된 문자를 count 한다.
- 입력 받은 문자가 알파벳이라면 스택에 삽입하는데 그 때 countPushedChar 변수를 +1 하여 삽입된 문자를 count 한다.
- 의미 없는 문자가 들어오면 (예:$,% 등) 예외처리를 해주는데 이 때 countIgnoredChar 변수를 +1 하여 무시된 문자를 count 한다.
- 만약 숫자 문자가 입력되었다면 숫자 문자를 정수값으로 변환하고 정수만큼 popN을 실행한다.
- 삽입된 문자가 '!'라면 quitStackProcessing 함수와 showStatistics 함수를 호출하고 프로그램을 종료한다.

```
private void quitStackProcessing() { // 스택 비우고 종료하는 함수
    AppView.outputLine("");
    AppView.outputLine("<스택을 비우고 사용을 종료합니다>");
    this.showAllFromBottom(); // 스택 Bottom-Top 순서로 출력
    this.popN(this.stack().size()); // 모든 원소 pop 삭제
}
```

스택을 비우고 종료하는 quitStackProcessing 함수이다. 스택을 비운다는 출력문을 출력하고 스택을 Bottom부터 Top까지 원소를 출력한다. 그리고 popN 함수에서 size만큼의 인자를 주어 size만큼 pop을 반복한다. 스택 모든 원소를 삭제하게 된다.

```
private void showStatistics() { // 통계를 출력하는 함수
    AppView.outputLine("");
    AppView.outputLine("<스택 사용 통계>");
    AppView.outputLine("- 입력된 문자는 " + this.inputChars() + " 개 입니다.");
    AppView.outputLine("- 정상 처리된 문자는 " + (this.inputChars() - this.ignoredChars()) + " 개 입니다.");
    AppView.outputLine("- 무시된 문자는 " + this.ignoredChars() + " 개 입니다.");
    AppView.outputLine("- 삽입된 문자는 " + this.pushedChars() + " 개 입니다.");
}
```

프로그램에 대한 통계를 출력하는 showStatistics 함수이다. 입력된 문자, 정상 처리된 문자, 무시된 문자, 삽입된 문자를 출력할 때는 inputChars, ignoredChars pushedChars를 이용한다. 프로그램 실행동안 count 하여 저장한다.

2. 프로그램 장단점 / 특이점 분석

➔ 장점

- 지난 주 과제를 통해 구현 되어있는 UnsortedArrayList에 대한 재사용으로 코드를 구현하여 편리하였다.
- 삽입도 addToLast, 삭제도 removeLast로 구현된 Last In First Out 구조여서 구현이 비교적 쉽다.
- 배열을 통해 구현하여서 인덱스를 통한 접근이 가능했다. 접근 가능성이 좋고 속도가 빠르다.
- ArrayList에 대한 정의를 하고 Stack으로 활용하니 구현이 편리하였다.
- LinkedList에 대한 Stack 구현도 쉽게 할 수 있다.

➔ 단점

- 데이터에 대한 접근이 Top 부분을 통해서만 가능하기 때문에 Bottom 원소나, 중간 원소에 대한 접근이 불가능하다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)

[입출력 결과]

```
<terminated> _DS08_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 26. 오후 7:43:25)
<<< 스택 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오: A
[Push] 삽입된 원소는 'A' 입니다 .
? 문자를 입력하시오: x
[Push] 삽입된 원소는 'x' 입니다 .
? 문자를 입력하시오: h
[Push] 삽입된 원소는 'h' 입니다 .
? 문자를 입력하시오: /
[Stack] <Bottom> A x h  <Top>
? 문자를 입력하시오: #
[Size] 스택에는 현재 3 개의 원소가 있습니다.
? 문자를 입력하시오: W
[Push] 삽입된 원소는 'W' 입니다 .
? 문자를 입력하시오: z
[Push] 삽입된 원소는 'z' 입니다 .
? 문자를 입력하시오: -
[Pop] 삭제된 원소는 ' z ' 입니다.
? 문자를 입력하시오: \
[Stack] <Top> W h x A  <Bottom>
? 문자를 입력하시오: ^
[Top] 스택의 Top 원소는 'W' 입니다.
? 문자를 입력하시오: 3
[Pops] 삭제된 원소는 ' W '입니다.
[Pops] 삭제된 원소는 ' h '입니다.
[Pops] 삭제된 원소는 ' x '입니다.
? 문자를 입력하시오: e
[Push] 삽입된 원소는 'e' 입니다 .
? 문자를 입력하시오: !

<스택을 비우고 사용을 종료합니다>
[Stack] <Bottom> A e  <Top>
[Pops] 삭제된 원소는 ' e '입니다.
[Pops] 삭제된 원소는 ' A '입니다.
```

```
<스택 사용 통계>
- 입력된 문자는 12 개 입니다.
- 정상 처리된 문자는 12 개 입니다.
- 무시된 문자는 0 개 입니다.
- 삽입된 문자는 6 개 입니다.

<<< 스택 기능 확인 프로그램을 종료합니다 >>>
```

(한번에 캡처가 안되어 나누어서 캡처한 점 참고 바랍니다.)

[예외 처리]

```
_DS08_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 26. 오후 7:46:01)
<<< 스택 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오: A
[Push] 삽입된 원소는 'A' 입니다 .
? 문자를 입력하시오:      r
[Push] 삽입된 원소는 'r' 입니다 .
? 문자를 입력하시오:

\
[Stack] <Top> r A  <Bottom>
```

➔ 입력에 공백이 있는 경우 -> 공백을 제거하고 인식

```
? 문자를 입력하시오: /
[Stack] <Bottom> A r  <Top>
? 문자를 입력하시오: t
[Push] 삽입된 원소는 't' 입니다 .
? 문자를 입력하시오: G
[Push] 삽입된 원소는 'G' 입니다 .
? 문자를 입력하시오: x
[Push] 삽입된 원소는 'x' 입니다 .
? 문자를 입력하시오: o
(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다.
```

➔ capacity 보다 더 입력한 경우

```
? 문자를 입력하시오: 6
[Pops] 삭제된 원소는 ' x '입니다.
[Pops] 삭제된 원소는 ' G '입니다.
[Pops] 삭제된 원소는 ' t '입니다.
[Pops] 삭제된 원소는 ' r '입니다.
[Pops] 삭제된 원소는 ' A '입니다.
[Pops.Empty] 스택에 더 이상 삭제할 원소가 없습니다.
```

➔ 스택 원소 수보다 삭제할 수가 많은 경우

```
? 문자를 입력하시오: %
[Ignore] 의미 없은 문자가 입력되었습니다.
? 문자를 입력하시오: $
[Ignore] 의미 없은 문자가 입력되었습니다.
? 문자를 입력하시오:
```

➔ 의미 없는 문자가 입력된 경우

(2) 결과 분석 (자신의 논리적 평가, 기타 느낀 점)

⇨ Stack은 Last In First Out 구조로, 마지막 입력이 먼저 출력되는 후입선출 구조이다.

⇨ ArrayList를 이용하여 구현하였는데 인덱스를 이용한 접근이 가능하여 접근성이 좋았으나 만약 배열이 가득 차면 스택 오버플로우가 발생할 수 있으며 배열을 더 크게 만들어 옮겨야 한다는 단점이 있다.

⇨ 반면, LinkedList를 이용하여 구현하면 스택의 크기를 자유롭게 구성할 수 있어 크기에 대한 제한이 없다.

⇨ 느낀점

- 자료구조 중 스택을 이용하는 법을 배웠는데 먼저 구조를 Array를 통해 정의해놓고 스택을 구현하니 편리하였다. LinkedList를 이용하여 구현한다고 해도 구조를 먼저 정의하고 구현하면 쉽게 구현할 것 같다.
- 스택을 이용할 수 있는 다양한 프로그램에 사용될 것이라고 생각된다.

4. 소스코드

```java
import java.util.Scanner;

public class AppView {

    private static Scanner scanner = new Scanner(System.in);

    // 생성자
    public AppView() {

    }

    // 출력 관련 함수
    // 한줄을 출력하는 함수 (한줄이 띄워지지않는다)
    public static void output(String message) {
        System.out.print(message); // 입력받은 message를 출력한다
    }

    // 한줄을 출력하는 함수 (한줄이 띄워진다)
    public static void outputLine(String message) {
        System.out.println(message); // 입력받은 message를 출력한다
    }

    // 입력 관련 함수
    public static char inputChar() {
        String line = AppView.scanner.nextLine().trim();
        while(line.equals("")) {
            line = AppView.scanner.nextLine().trim();
        }
        return line.charAt(0);
    }
}
```

[AppView Class]

```java
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();

}
```

[Stack - interface]

```
    // 비공개 변수들
    private ArrayList<Character> _stack;
    private int _inputChars; // 입력된 문자의 개수
    private int _pushedChars; // 삽입된 문자의 개수
    private int _ignoredChars; // 무시된 문자의 개수

    // Getters/Setters
    private ArrayList<Character> stack() {
        return this._stack;
    }

    private void setStack(ArrayList<Character> newStack) {
        this._stack = newStack;
    }

    private int inputChars() {
        return this._inputChars;
    }

    private void setInputChars(int newInputChars) {
        this._inputChars = newInputChars;
    }

    private int pushedChars() {
        return this._pushedChars;
    }

    private void setPushedChars(int newPushedChars) {
        this._pushedChars = newPushedChars;
    }

    private int ignoredChars() {
        return this._ignoredChars;
    }

    private void setIgnoredChars(int newIgnoredChars) {
        this._ignoredChars = newIgnoredChars;
    }

    // 생성자
    public AppController() {
        this.setStack(new ArrayList<Character>(AppController.STACK_CAPACITY));
        this.setInputChars(0);
        this.setPushedChars(0);
        this.setIgnoredChars(0);
    }

    // 비공개 함수
    // 횟수 계산
    private void countInputChar() { // 입력된 문자 + 1 count
        this.setInputChars(this.inputChars() + 1);
    }

    private void countIgnoredChar() { // 무시된 문자 + 1 count
        this.setIgnoredChars(this.ignoredChars() + 1);
    }

    private void countPushedChar() { // 삽입된 문자 + 1 count
        this.setPushedChars(this.pushedChars() + 1);
    }

    // 스택 요청 관련
    private void pushToStack(char aCharForPush) { // 스택 Top에 원소를 삽입하는 함수
        if (this.stack().isFull()) { // 스택이 가득 차있다면
            AppView.outputLine("(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다."); // 오류 출력은 별도
        } else {
            Character charObjectForAdd = Character.valueOf(aCharForPush); // 문자를 문자 객체로 변환
            if (this.stack().push(charObjectForAdd)) { // 스택에 push
                AppView.outputLine("[Push] 삽입된 원소는 '" + aCharForPush + "' 입니다.");
            } else {
                AppView.outputLine("(오류) 스택에 넣는 동안에 오류가 발생하였습니다.");
            }
        }

    }

    private void popOne() { // 스택 Top의 원소를 하나 삭제하며 return하는 함수
        if (this.stack().isEmpty()) { // 스택이 비어있다면
            AppView.outputLine("[Pop.Empty] 스택에 삭제할 원소가 없습니다."); // 원소가 없다는 출력은 별도
        } else {
            Character poppedChar = this.stack().pop(); // pop한 결과를 poppedChar에 저장
            if (poppedChar == null) { // poppedChar가 null 검사
                AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");
            } else {
                AppView.outputLine("[Pop] 삭제된 원소는 ' " + poppedChar + " ' 입니다.");
            }
        }
    }

    private void popN(int numberOfCharsToBePopped) { // 스택 Top의 원소를 numberOfCharsToBePopped만큼 삭제하며 return하는 함수
        if (numberOfCharsToBePopped == 0) { // 만약 입력받은 수가 0이라면
            AppView.outputLine("[Pops] 삭제할 원소의 개수가 0 개 입니다."); // 삭제될수가 0
        } else {
            int count = 0;
            while (count < numberOfCharsToBePopped && (!this.stack().isEmpty())) { // 입력받은 수 만큼 반복
                Character poppedChar = this.stack().pop(); // 반복하며 pop을 수행
                if (poppedChar == null) {
                    AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");

                } else {
                    AppView.outputLine("[Pops] 삭제된 원소는 ' " + poppedChar + " ' 입니다.");
                }
                count++;
            }
            if (count < numberOfCharsToBePopped) { // 스택에 있는 원소의 수보다 입력받은 숫자가 더 큰 경우
                AppView.outputLine("[Pops.Empty] 스택에 더 이상 삭제할 원소가 없습니다."); // 오류은 별도
            }
        }
    }

    private void quitStackProcessing() { // 스택 비우고 종료하는 함수
        AppView.outputLine("");
        AppView.outputLine("스택을 비우고 사용을 종료합니다>");
        this.showAllFromBottom(); // 스택 Bottom-Top 순서로 출력
        this.popN(this.stack().size()); // 모든 원소 pop 삭제
    }

    // 출력 관련
    private void showAllFromBottom() { // 스택의 모든 원소를 Bottom 부터 Top까지 출력한다
        AppView.output("[Stack] <Bottom> ");
        for (int order = 0; order < this.stack().size(); order++) {
            AppView.output(this.stack().elementAt(order).toString() + " ");
        }
        AppView.outputLine(" <Top>");
    }

    private void showAllFromTop() { // 스택의 모든 원소를 Top 부터 Bottom까지 출력한다
        AppView.output("[Stack] <Top> ");
        for (int order = this.stack().size() - 1; order >= 0; order--) {
            AppView.output(this.stack().elementAt(order).toString() + " ");
        }
        AppView.outputLine(" <Bottom>");
    }

    private void showTopElement() {
        if (this.stack().isEmpty()) {
            AppView.outputLine("[Top.Empty] 스택이 비어서 Top 원소가 존재하지 않습니다.");
        } else {
            AppView.outputLine("[Top] 스택의 Top 원소는 '" + this.stack().peek() + "' 입니다.");
        }

    }

    private void showStackSize() { // 스택 사이즈를 출력하는 함수
        AppView.outputLine("[Size] 스택에는 현재 " + this.stack().size() + " 개의 원소가 있습니다.");
    }

    private void showStatistics() { // 통계를 출력하는 함수
        AppView.outputLine("");
        AppView.outputLine("<스택 사용 통계>");
        AppView.outputLine("- 입력된 문자는 " + this.inputChars() + " 개 입니다.");
        AppView.outputLine("- 정상 처리된 문자는 " + (this.inputChars() - this.ignoredChars()) + " 개 입니다.");
        AppView.outputLine("- 무시된 문자는 " + this.ignoredChars() + " 개 입니다.");
        AppView.outputLine("- 삽입된 문자는 " + this.pushedChars() + " 개 입니다.");
    }

    // 입력 관련
    private char inputChar() {
        AppView.output("? 문자를 입력하시오: ");
        return AppView.inputChar();
    }

    public void run() {
        AppView.outputLine("<<< 스택 기능 확인 프로그램을 시작합니다 >>>");
        AppView.outputLine("");

        char input = this.inputChar(); // char 문자 입력받음
        while (input != '!') { // '!'을 입력받으면 종료
            this.countInputChar(); // 입력된문자 + 1
            if (Character.isAlphabetic(input)) { // 알파벳인지 검사
                this.pushToStack(input);
                this.countPushedChar(); // 삽입된문자 + 1
            } else if ((Character.isDigit(input))) { // 숫자 문자인지 검사
                this.popN(Character.getNumericValue(input)); // 숫자 문자를 정수값으로 변환
            } else if (input == '-') { // '-'문자 입력받으면 popOne 실행
                this.popOne();
            } else if (input == '#') { // '#'문자 입력받으면 스택크기 출력
                this.showStackSize();
            } else if (input == '/') { // '/'문자 입력받으면 Bottom-Top까지의 스택 출력
                this.showAllFromBottom();
            } else if (input == '\\') { // '\'문자 입력받으면 Top-Bottom까지의 스택 출력
                this.showAllFromTop();
            } else if (input == '^') { // '^'문자 입력받으면 Top원소 peek하여 출력
                this.showTopElement();
            } else {
                AppView.outputLine("[Ignore] 의미 없음 문자가 입력되었습니다."); // 오류처리
                this.countIgnoredChar(); // 무시된문자 + 1
            }
            input = this.inputChar(); // 반복하서 실행
        }
        this.quitStackProcessing(); // 스택 비우는 함수
        this.showStatistics();
        AppView.outputLine("");
        AppView.outputLine("<<< 스택 기능 확인 프로그램을 종료합니다 >>> ");
    }
}
```

[AppController]

```java
    // Constant
    private static final int DEFAULT_CAPACITY = 5;

    // private instance variables
    private int _capacity;
    private int _size;
    private E[] _elements;

    // Getters/Setters
    public int capacity() {
        return this._capacity;
    }

    private void setCapacity(int newCapacity) {
        this._capacity = newCapacity;
    }

    @Override
    public int size() {
        return this._size;
    }

    public void setSize(int newSize) {
        this._size = newSize;
    }

    private E[] elements() {
        return this._elements;
    }

    private void setElements(E[] newElements) {
        this._elements = newElements;
    }

    // Constructor
    public ArrayList() {
        this(ArrayList.DEFAULT_CAPACITY); // 다른 생성자 사용
    }

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

    // Private Methods
    private void makeRoomAt(int aPosition) {
        for (int i = this.size(); i > aPosition; i--) { // size-aPosition번복
            this.elements()[i] = this.elements()[i - 1]; // i-1번째 원소를 i에 저장
        }
    }

    private void removeGapAt(int aPosition) {
        for (int i = aPosition + 1; i < this.size(); i++) { // aPosition+1-size만복
            this.elements()[i - 1] = this.elements()[i]; // i번째 원소를 i-1에 저장
        }
        this.elements()[this.size() - 1] = null; // 마지막 원소 삭제
    }

    // public Methods
    @Override
    public boolean isFull() { // 배열이 가득 찼는지 확인
        return (this.capacity() == this.size());
    }

    @Override
    public boolean isEmpty() { // 배열이 비어있는가 확인
        return (this.size() == 0);
    }

    public boolean doesContain(E anElement) { // 존재 유무 확인
        return (this.orderOf(anElement) >= 0); // anElement가 orderOf로 인해 있다는게 확인되면 true를 반환
    }

    public int orderOf(E anElement) {
        int order = -1; // 순서 변수 -1 로 선언
        for (int index = 0; index < this.size() && order < 0; index++) { // 0~사이즈 & order이 -1일때 반복
            if (this.elements()[index].equals(anElement)) { // index번째 배열과 anElement가 같으면
                order = index; // order 를 index로 설정
            }
        }
        return order; // 같은 배열을 찾은 순서인 order를 반환
    }

    public E elementAt(int anOrder) { // anOrder번째 원소 반환
        if (anOrder < 0 || anOrder >= this.size()) { // anOrder이 범위 벗어나면
            return null; // null 반환
        } else {
            return this.elements()[anOrder]; // anOrder번째 원소 반환
        }
    }

    public void setElementsAt(int anOrder, E anElement) { // anOrder번째에 원소삽입
        if (anOrder < 0 || anOrder >= this.size()) { // 입력한 anOrder의 유효성 판단
            return;
        } else {
            this.elements()[anOrder] = anElement; // anOrder번째 배열 anElement로 설정
        }
    }

    public boolean addTo(E anElement, int anOrder) { // anOrder번째에 원소삽입
        if (this.isFull()) { // 배열이 가득 찼다면
            return false; // false를 반환
        } else if (anOrder < 0 || anOrder > this.size()) { // 입력한 anOrder의 유효성 판단
            return false; // 유효하지 않다면 false를 반환
        } else {
            this.makeRoomAt(anOrder); // anOrder 순서에 삽입할 자리 마련
            this.elements()[anOrder] = anElement; // 해당 순서에 원소 삽입
            this.setSize(this.size() + 1); // size set
            return true;
        }
    }

    public boolean addToFirst(E anElement) { // 배열 첫번째에 원소삽입
        return addTo(anElement, 0);
    }

    public boolean addToLast(E anElement) { // 배열 마지막에 원소삽입
        return addTo(anElement, this.size());
    }

    public E removeFrom(int anOrder) {
        if (anOrder < 0 || anOrder >= this.size()) { // 입력된 anOrder 유효성 판단
            return null; // 유효하지 않다면 null 반환
        } else {
            E removedElement = this.elements()[anOrder]; // 삭제될 원소를 removedElement에 저장
            this.removeGapAt(anOrder); // anOrder 이후 원소 한칸씩 앞으로 저장
            this.setSize(this.size() - 1); // size set
            return removedElement;
        }
    }

    public E removeFirst() { // 첫번째 원소 삭제
        return removeFrom(0);
    }

    public E removeLast() { // 마지막 원소 삭제
        return removeFrom(this.size() - 1);
    }

    @Override
    public boolean push(E anElement) {
        return this.addToLast(anElement); // 배열 마지막에 add
    }

    @Override
    public E pop() {
        return this.removeLast(); // 배열 마지막 원소 remove
    }

    @Override
    public E peek() {
        if (this.isEmpty()) { // 배열이 empty 라면
            return null; // null을 반환
        } else {
            return this.elementAt(this.size() - 1); // Last element
        }
    }

    @Override
    public void clear() {
        for (int i = 0; i < this.size(); i++) { // 배열의 size만큼 반복
            this.elements()[i] = null; // 모든 배열을 null 처리
        }
        this.setSize(0); // size를 0으로 set
    }
}
```

[ArrayList]