자료구조 실습 보고서

[제 07 주] 스택 - 기본 기능

제출일	2017/04/21
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입출력
    - 영문 대소문자를 스택에 삽입한다.
      - 만일 스택이 꽉 차 있으면 삽입이 불가능하다는 메세지를 출력한다.
    - !(느낌표)를 입력하면 스택을 모두 삭제하고 프로그램을 종료한다.
      - 삭제할 때마다 삭제된 원소가 무엇인지 알려주는 메세지를 출력한다.
    - 숫자를 입력하면 해당 수 만큼 스택에서 삭제한다.
      - 매번 삭제할 때마다 삭제된 원소가 무엇인지 알려주는 메세지를 출력한다.
    - -(하이픈)를 입력하면 스택의 가장 위의 원소를 삭제한다.
    - 삭제를 시도하였을 때, 스택이 비어있다면 관련 메세지를 출력한다.
    - #(샵)을 입력하면 스택 안의 원소의 갯수를 출력한다.
    - /(슬래시)를 입력하면 스택 구조의 바닥부터 차례대로 원소를 출력한다.
    - \(역슬래시)를 입력하면 스택 구조의 위부터 차례대로 원소를 출력한다.
    - ^(곡절부호)를 입력하면 스택 구조의 제일 위 원소를 출력한다.
    - 그 밖의 문자들을 입력하면 의미 없는 문자가 입력되었다는 메세지를 출력한 후 입력을 무시한다.

- 자료구조
  - 영문자를 저장하기 위한 Stack 형태 자료구조
  - Stack 형태의 자료구조를 구현하기 위한 Array 형태의 List
  - 알림 및 오류 등의 메세지를 처리하는 enum

1 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppController()	없음	없음	_appView 변수와 AppController 변수들을 초기화하는 생성자 메소드
	run()	없음	없음	학생들의 성적순 정렬 프로그램을 실행시키는 메소드
	void showMessage(MessageID aMessageID)	에러 메세지 혹은 알림	없음	에러 메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void showAllFromBottom()	없음	없음	스택 구조 내부의 원소를 바닥에서부터 차례대로 출력을 지시하는 메소드
	void showAllFromTop()	없음	없음	스택 구조 내부의 원소를 맨 위에서부터 차례대로 출력을 지시하는 메소드
	void showTopElement()	없음	없음	스택 구조에서 맨 위의 원소를 출력하는 메소드
	void showStackSize()	없음	없음	스택 구조 내부 원소의 총 갯수를 출력하는 메소드

	void countAdded()	없음	없음	스택에 집어넣은 원소를 카운트하는 메소드
	void countIgnored()	없음	없음	스택에 집어넣지 않고 무시한 갯수를 카운트 하는 메소드
	void countInputChar()	없음	없음	입력받은 문자들을 카운트하는 메소드
	void addToStack(char inputChar)	입력받는 문자값	없음	입력받은 문자값을 스택에 추가하고 삽입된 원소가 무엇인지 출력 후 카운팅하는 메소드
	void removeOne()	없음	없음	스택이 비어있지 않다면 스택의 맨 위의 원소값을 출력하여 삭제하는 메소드
	void removeN(int numberOfCharsToBeRemoved)	지울 원소의 갯수	없음	입력받은 숫자만큼 스택의 원소를 지운다.
	void conclusion()	없음	없음	프로그램이 종료 될 때, 스택 내부의 원소들을 모두 삭제하며 출력한다. 또한 입력받은 갯수와 무시한 수, 스택에 추가한 문자 수들을 출력지시한다.

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	int inputInt()	없음	정수값	입력 받은 정수를 리턴하는 메소드
	String inputString()	없음	문자열값	입력 받은 문자열을 리턴하는 메소드
	void outputMessage(String aMessage)	문자열	없음	전달 받은 문자열을 출력하는 메소드
	char inputCharacter()	없음	문자값	입력받은 알파벳 문자값을 리턴하는 메소드
	void outputAddedElement(char anElement)	알파벳 문자값	없음	삽입된 원소가 무엇인지 출력하는 메소드
	void outputStackElement(char anElement)	알파벳 문자값	없음	스택 내부 원소를 출력하는 메소드
	void outputTopElement(char anElement)	알파벳 문자값	없음	스택 내부의 제일 위에 있는 원소를 출력하는 메소드
	void outputStackSize(int aStackSize)	스택 내부의 총 원소 갯수	없음	스택 내부의 총 원소갯수를 출력하는 메소드
	void outputRemove(char anElement)	알파벳 문자값	없음	스택에서 삭제된 원소를 출력하는 메소드
	void outputRemoveN(int numberOfCharsToBeRemoved)	삭제해야할 스택 내부 원소의 개수	없음	삭제해야할 스택 내부 원소 개수를 출력하는 메소드
	void outputResult(int	입력받은 문자 갯수, 무시한 문자 갯수,	없음	입력받아서 무시되거나 추가된 문자들의 갯수를

	numberOfinputChars, int numberOfIgnoredChars, int numberOfAddedChars)	스택에 입력한 문자 갯수		출력하는 메소드

Class	interface Stack<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	boolean push(T anElement)	원소	boolean	스택에 원소를 삽입하고자 할 때 사용할 메소드
	T pop()	없음	없음	스택의 맨 위의 원소를 빼내어 삭제할 때 사용할 메소드
	T peek()	없음	없음	스택의 맨 위의 원소를 출력하고자 할 때 사용할 메소드

Class	ArrayList<T> implements Stack<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	ArrayList()	없음	없음	디폴트값으로 스택의 최대 용량과 스택 역할을 할 배열의 최대 값을 초기화하는 생성자 메소드
	ArrayList(int givenCapacity)	스택의 최대 용량 값	없음	입력 받은 값으로 스택의 최대 용량과 스택 역할을 할 배열의 최대값을 초기화 하는 생성자 메소드
	boolean isEmpty()	없음	boolean	스택이 비어있으면 true, 비어있지 않으면 false 를 리턴하는 메소드
	boolean isFull()	없음	boolean	스택이 꽉 차 있다면 true, 그렇지 않다면 false
	int size()	없음	스택 내 원소의 갯수	스택 내부의 원소의 총 갯수 출력
	boolean push(T anElement)	원소	스택에 삽입하면 true	스택이 꽉 차있는지 검사하고, 그렇지않으면 스택에 원소를 추가
	T pop()	없음	원소	스택 내 가장 위의 원소 순번을 줄이고 해당 원소를 반환
	T peek()	없음	원소	스택 내 가장 위의 원소를 리턴
	T elementAT(int aPosition)	원소가 존재하는 위치	원소	입력받은 위치에 존재하는 스택의 원소를 반환

2 종합 설명서

- 문자열을 입력하면 push()메소드가 실행되어 스택 내부에 원소를 집어넣는다.
- '#'을 입력하면 ArrayList 의 size()가 실행되어 스택 내부에 존재하는 원소의 총 개수를 리턴한다.
- '/'를 입력하면 ArrayList 의 elementAt()메소드를 이용하여 바닥부터 제일 위의 있는 원소를 차례대로 출력한다.

- '\' 를 입력하면 ArrayList 의 elementAt()메소드를 이용하여 거꾸로 위에서부터 아래로 스택 내부의 원소를 출력한다.
- '^' 를 입력하면 ArrayList 의 peek()을 이용하여 스택의 가장 상단의 원소를 출력한다.
- 숫자를 입력할 시 숫자만큼의 원소를 삭제한다.
  - 더이상 삭제할 원소가 없으면 알림 메세지를 출력한다.
- '!' 를 입력하면 스택 내부의 원소를 모두 출력하며 삭제 후 입력받아 정상 처리된 문자, 무시된 문자, 삽입된 문자를 출력하고 프로그램을 종료한다.

2 프로그램 장단점 분석

- 장점
  - List 형의 Stack 자료구조를 구현하였다.
  - 삽입된 원소를 바닥에서 위로 혹은 위에서 바닥으로 출력할 수 있다
  - Stack 내부의 원소의 총 갯수를 알 수 있다.
  - 원하는 갯수만큼의 원소를 한 번에 삭제할 수 있다.
  - 입력받은 문자수와 정상적으로 스택에 입력된 문자수, 무시된 문자수를 카운트하여 보여줄 수 있다.

- 단점
  - 원하는 위치에 원소를 추가할 수 없다.
  - 원하는 위치의 원소를 삭제할 수 없다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행 및 문자 입력

```
> 프로그램을 시작합니다.
[스택 사용을 시작합니다.]
- 문자를 입력하시오 : A
[Push] 삽입된 원소는 'A'입니다.
```

입력 - '#' 입력 시 스택 내부 원소의 총 개수 출력

```
- 문자를 입력하시오 : #
[Size] 스택에는 현재 2개의 원소가 있습니다.
```

입력 - '/' 입력 시 스택 내부의 바닥에서부터 최상위 원소 출력

```
- 문자를 입력하시오 : /
[Stack] <Bottom> A  x  h <Top>
```

입력 - '\' 입력 시 최상위에서부터 바닥까지 원소 출력

```
- 문자를 입력하시오 : \
[Stack] <Top> z  W  h  x  A <Bottom>
```

입력 - '-' 입력 시 원소 삭제 및 삭제된 원소 출력

```
- 문자를 입력하시오 : -
[Pop] 삭제된 원소는 'z'입니다.
```

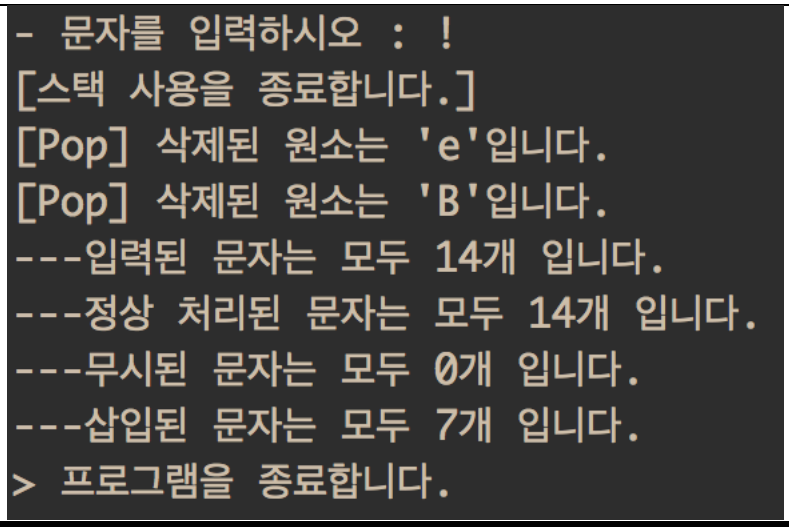
입력 - '^' 입력시 스택 내 최상위 원소 출력

```
- 문자를 입력하시오 : ^
[Stack] [Top] Top 원소는 'W'입니다.
```

입력 - 숫자 입력시 입력한 숫자 만큼의 원소 삭제.
삭제할 원소가 없을 시 알림 메세지 출력

```
- 문자를 입력하시오 : 5
[Pop] 삭제된 원소는 'W'입니다.
[Pop] 삭제된 원소는 'h'입니다.
[Pop] 삭제된 원소는 'x'입니다.
[Pop] 삭제된 원소는 'A'입니다.
[Empty] 스택에 삭제할 원소가 없습니다.
```

입력 - '!' 입력 시 프로그램 종료 및 스택 내부 원소 삭제, 처리된 문자 출력



2 결과 분석

- Last-In-First-out 개념의 Stack 자료구조를 구현하여 원소를 추가하면 차곡차곡 쌓이고, 원소를 꺼내려면 제일 위의 원소를 꺼내는 형태의 프로그램을 구현하였다.
- Stack 내부의 원소 의 총 개수를 출력할 수 있다.
- 스택 내부의 원소를 위에서 아래 혹은 아래에서 위의 방향으로 출력할 수 있다.
- 프로그램 종료 시 스택 내부 원소를 출력하며, 입력 관련 정보를 출력한다.
- 그러나, 원하는 위치에 삽입 혹은 삭제를 할 수 없다.

- 결론
  - Stack 구조를 구현해 볼 수 있다.
  - 원하는 위치에 삽입/삭제를 하려면 Stack 구조는 적절하지 않다.

4 소스 코드

Class	AppController

```
public class AppController {
	private AppView _appView;
	private ArrayList<Character> _arrayStack;
	private int _inputChars; //입력된 문자의 개수
	private int _ignoredChars; //무시된 문자의 개수
	private int _addedChars; //삽입된 문자의 개수

	public AppController() {
		this._appView = new AppView();
		this._inputChars = 0;
		this._ignoredChars = 0;
		this._addedChars = 0;
	}
	public void run() {
		this._arrayStack = new ArrayList<Character>();
		char input;

		this.showMessage(MessageID.Notice_StartProgram);
		this.showMessage(MessageID.Notice_StartMenu);
		input = this._appView.inputCharacter();
		while(input != '!') {
			this.countInputChar();
			if ((input >= 'A' && input <= 'Z') || (input >= 'a' && input <= 'z')) {
				this.addToStack(input);
			} else if (input>= '0' && input <= '9') {
				this.removeN(input - '0');
			} else if (input == '-') {
				this.removeOne();
			} else if (input == '#' ) {
				this.showStackSize();
			} else if (input == '/') {
				this.showAllFromBottom();
			} else if(input =='₩₩') {
				this.showAllFromTop();
			} else if (input =='^') {
				this.showTopElement();
			} else {
				this.showMessage(MessageID.Error_WrongMenu);
				this.countIgonored();
			}
			input = this._appView.inputCharacter();
		}
		this.showMessage(MessageID.Notice_EndMenu);
		this.conclusion();
		this.showMessage(MessageID.Notice_EndProgram);
	}


	private void showAllFromBottom() {
		this.showMessage(MessageID.Notice_ShowStack);
		this.showMessage(MessageID.Show_StartBottom);
```

```
		for(int index = 0; index < this._arrayStack.size(); index++) {
			this._appView.outputStackElement((char)this._arrayStack.elementAt(index));
		}
		this.showMessage(MessageID.Show_EndBottom);

	}
	private void showAllFromTop() {
		this.showMessage(MessageID.Notice_ShowStack);
		this.showMessage(MessageID.Show_StartTop);

		for(int index = (this._arrayStack.size() - 1) ; index > -1; index--) {
			this._appView.outputStackElement((char) this._arrayStack.elementAt(index));

		}
		this.showMessage(MessageID.Show_EndTop);

	}
	private void showTopElement() {
		this.showMessage(MessageID.Notice_ShowStack);
		this._appView.outputTopElement(this._arrayStack.peek());

	}
	private void showStackSize() {
		this._appView.outputStackSize(this._arrayStack.size());

	}
	private void countAdded() {
		this._addedChars++;

	}
	private void countIgonored() {
		this._ignoredChars++;

	}
	private void countInputChar() {
		this._inputChars++;

	}
	private void addToStack(char inputChar) {
		if (this._arrayStack.push(new Character(inputChar))) {
			this._appView.outputAddedElement(inputChar);
			this.countAdded();
		} else {
			this.showMessage(MessageID.Error_InputFull);
		}

	}
	private void removeOne() {
		if(this._arrayStack.isEmpty()) {
			this.showMessage(MessageID.Error_RemoveEmpty);
		} else {
			this._appView.outputRemove((char)this._arrayStack.pop());
		}

	}
	private void removeN(int numberOfCharsToBeRemoved) {
		for(int i = 0; i < numberOfCharsToBeRemoved; i++){
```

```
                        if(this._arrayStack.isEmpty()) {
                                this.showMessage(MessageID.Error_RemoveEmpty);
                                break;
                        } else {
                                this._appView.outputRemove((char)this._arrayStack.pop());
                        }
                }
        }
        private void conclusion() {
                for(int i = this._arrayStack.size() ; i > -1; i--) {
                        if(this._arrayStack.isEmpty()) {
                                break;
                        } else {
                        removeOne();
                        }
                }
                this._appView.outputResult(_inputChars, _ignoredChars, _addedChars);


        }
        private void showMessage(MessageID aMessage) {
                switch(aMessage) {
                case Notice_StartProgram :
                        this._appView.outputMessage("> 프로그램을 시작합니다.₩n");
                        break;
                case Notice_StartMenu :
                        this._appView.outputMessage("[스택 사용을 시작합니다.]₩n");
                        break;
                case Notice_EndMenu :
                        this._appView.outputMessage("[스택 사용을 종료합니다.]₩n");
                        break;
                case Notice_EndProgram :
                        this._appView.outputMessage("> 프로그램을 종료합니다.₩n");
                        break;
                case Notice_ShowStack :
                        this._appView.outputMessage("[Stack] ");
                        break;
                case Show_StartBottom :
                        this._appView.outputMessage("<Bottom>");
                        break;
                case Show_EndBottom:
                        this._appView.outputMessage("<Top>₩n");
                        break;
                case Show_StartTop :
                        this._appView.outputMessage("<Top>");
                        break;
                case Show_EndTop:
                        this._appView.outputMessage("<Bottom>₩n");
                        break;
                case Error_WrongMenu:
                        this._appView.outputMessage("[Error] 의미 없는 문자가 입력되었습니다.₩n");
                        break;
                case Error_InputFull:
                        this._appView.outputMessage("[Full] 스택이 꽉 차서 삽입이 불가능합니다.₩n");
                        break;
                case Error_RemoveEmpty:
                        this._appView.outputMessage("[Empty] 스택에 삭제할 원소가 없습니다.₩n");
                        break;
```

```
                default :
                        break;
                }
        }
}
```

Class	AppView

```
import java.util.Scanner;

public class AppView {
        private Scanner _scanner;

        public AppView() {
                this._scanner = new Scanner(System.in);
        }
        public int inputInt() {
                return Integer.parseInt(this._scanner.nextLine());
        }
        public String inputString() {
                return this._scanner.nextLine();
        }

        public char inputCharacter() {
                char element;
                System.out.print("- 문자를 입력하시오 : ");
                element = this.inputString().charAt(0);
                return element;
        }
        public void outputAddedElement(char anElement) {
                System.out.println("[Push] 삽입된 원소는 '" + anElement + "'입니다.");

        }
        public void outputStackElement(char anElement) {
                System.out.print(" " + anElement + " ");

        }
        public void outputTopElement(char anElement) {
                System.out.println("[Top] Top 원소는 '" + anElement + "'입니다.");

        }
        public void outputStackSize(int aStackSize) {
                System.out.println("[Size] 스택에는 현재 " + aStackSize+ "개의 원소가 있습니다.");

        }
        public void outputRemove(char anElement) {
                System.out.println("[Pop] 삭제된 원소는 '" + anElement + "'입니다.");

        }
        public void outputRemoveN(int numberOfCharsToBeRemoved) {

        }
        public void outputResult(int numberOfInputChars, int numberOfIgnoredChars,
                                                                int numberOfAddedChars) {
                System.out.println("---입력된 문자는 모두 " + numberOfInputChars + "개 입니다.");
```

```
                System.out.println("---정상 처리된 문자는 모두 " + (numberOfInputChars - numberOfIgnoredChars) + "개 입니다.");
                System.out.println("---무시된 문자는 모두 " + numberOfIgnoredChars + "개 입니다.");
                System.out.println("---삽입된 문자는 모두 " + numberOfAddedChars + "개 입니다.");


        }
        public void outputMessage (String aMessageString) {
                System.out.print(aMessageString);
        }



}
```

Class	Ban

```
public class ArrayList<T> implements Stack<T> {

        private static final int DEFAULT_CAPACITY = 5;
        private int _capacity;
        private int _top;
        private T[] _elements;



        public ArrayList() {
                this (DEFAULT_CAPACITY);
        }

        @SuppressWarnings("unchecked")
        public ArrayList(int givenCapacity) {
                this._capacity = givenCapacity;
                this._top = -1;
                this._elements = (T[]) new Object [ArrayList.DEFAULT_CAPACITY];
        }

        public boolean isEmpty() {
                return (this._top == -1);
        }

        public boolean isFull() {
                return (this._top+1 == this._capacity);
        }

        public int size() {
                return this._top+1;
        }


        @Override
        public boolean push(T anElement) {
                // TODO Auto-generated method stub
                if(isFull()) {
                        return false;
```

```
                } else {

                        this._top++;
                        this._elements[this._top] = anElement;

                }
                return true;
        }

        @Override
        public T pop() {
                // TODO Auto-generated method stub
                if(isEmpty()) {
                        return null;
                } else {
                        int i = this._top;
                        this._top--;
                        return this._elements[i];
                }

        }

        @Override
        public T peek() {
                // TODO Auto-generated method stub
                if(isEmpty()) {
                        return null;
                } else {
                        return this._elements[this._top];
                }

        }

        public void clear() {
                this._top = -1;

                for(int i = 0; i < this._capacity; i++) {
                        this._elements[i] = null; //집합의 모든 내용물을 비우는 작업
                }
        }

        public T elementAt(int aPosition) {
                return this._elements[aPosition];
        }



}
```

interface Class	Stack<T>

```
public interface Stack<T> {

        public boolean push (T anElement);
        public T pop();
```

```
	public T peek();

}
```