자료구조 실습 보고서

[제08주] 스택 - 기본기능

제출일 : 2021/04/25

학번/이름 : 201400875/김용준

주요 개념:

- Generic Type 의 개념, 사용법

>>> 제네릭 타입의 개념은 ‘클래스에서 사용할 타입을 클래스 외부에서 설정하는 타입’

- Interface 의 개념, 사용법
- Stack 의 개념, 사용법

>>> Stack의 개념은 사전적 의미로는 '쌓다', '더미'라는 뜻. 따라서 Stack은 데이터를 쌓는 형식으로 저장하는데 따라서 조회, 추가, 삭제 모두 가장 위에 있는 즉 가장 최근의 값에서 이루어진다. 스택 구조에서 가장 상단에 있는 데이터를 Top, 하단에 있는 데이터를 Bottom이라고 한다.

출력1

```
DS07_Main_201400875_김용준 (1) [Java Application] C:₩Prog
<<< 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 : #
[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 : p
(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다.
? 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 :
? 문자를 입력하시오 : 5
[Pops] 삭제된 원소는 'W' 입니다.
[Pops] 삭제된 원소는 'h' 입니다.
[Pops] 삭제된 원소는 'w' 입니다.
[Pops] 삭제된 원소는 'A' 입니다.
[Pops.Empth] 스택에 더이상 삭제할 원소가 없습니다.
? 문자를 입력하시오 : ^
[Top.Empty] 스택이 비어서 Top 원소가 존재하지 않습니다
? 문자를 입력하시오 : B
[Push] 삽입된 원소는 'B' 입니다.
? 문자를 입력하시오 : e
[Push] 삽입된 원소는 'e' 입니다.
? 문자를 입력하시오 : !

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

<스택 사용 통계>
- 입력된 문자는 15 개 입니다.
- 정상 처리된 문자는 15 개 입니다.
- 무시된 문자는 0 개 입니다.
- 삽입된 문자는 8 개 입니다.

<<< 스택 기능 확인 프로그램을 종료합니다
```

```
Console  Problems  Debug Shell
<terminated> DS07_Main_201400875_김용준 (1) [Java Applic
<<< 스택 기능 확인 프로그램을 시작합니다 >>>

? 문자를 입력하시오 : A
[Push] 삽입된 원소는 'A' 입니다.
? 문자를 입력하시오 : x
[Push] 삽입된 원소는 'x' 입니다.
? 문자를 입력하시오 : &
[Ignore] 의미 없는 문자가 입력되었습니다.
? 문자를 입력하시오 : \
[Stack] <Top> x A  <Bottom>
? 문자를 입력하시오 :

2
[Pops] 삭제된 원소는 'x' 입니다.
[Pops] 삭제된 원소는 'A' 입니다.
? 문자를 입력하시오 : -
[Pop.Empty] 스택에 삭제할 원소가 없습니다.
? 문자를 입력하시오 :     -
[Pop.Empty] 스택에 삭제할 원소가 없습니다.
? 문자를 입력하시오 : !

<스택을 비우고 사용을 종료합니다>
[Stack] <Bottom>  <Top>
[Pops] 삭제할 원소의 개수가 0개 입니다.

<스택 사용 통계>
- 입력된 문자는 7 개 입니다.
- 정상 처리된 문자는 6 개 입니다.
- 무시된 문자는 1 개 입니다.
- 삽입된 문자는 2 개 입니다.

<<< 스택 기능 확인 프로그램을 종료합니다
```

Main

```
public class DS07_Main_201400875_김용준 {

	public static void main(String[] args) {
		AppController appController = new AppController();
		appController.run();
	}
}
```

AppController

```
public class AppController {

	private static final int STACK_CAPACITY = 5;

	private ArrayList<Character> _stack;

	private int _inputChars;
	private int _pushedChars;
	private int _ignoredChars;

	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;
	}

	public AppController() {
		this.setStack(new ArrayList<Character>(AppController.STACK_CAPACITY));
		this.setInputChars(0);
		this.setPushedChars(0);
		this.setIgnoredChars(0);
	}
		// 횟수 계산
	private void countInputChar() {
		this.setInputChars(this.inputChars()+1);
	}
	private void countIgnoredChar() {
		this.setIgnoredChars(this.ignoredChars()+1);
	}
	private void countPushedChar() {
		this.setPushedChars(this.pushedChars()+1);
	}
		// 스택 수행 관련
	private void pushToStack(char aCharForPush) {
```

```
		if (this.stack().isFull()) {
			AppView.outputLine("(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다.");
		}
		else {
			Character charObjectForAdd = Character.valueOf(aCharForPush);
			if(this.stack().push(charObjectForAdd)) {
				AppView.outputLine("[Push] 삽입된 원소는 '" + aCharForPush + "' 입니다.");
			}
			else {
				AppView.outputLine("(오류) 스택에 넣는 동안에 오류가 발생하였습니다.");
			}
		}
	}
	private void popOne() {
		if(this.stack().isEmpty()) {
			AppView.outputLine("[Pop.Empty] 스택에 삭제할 원소가 없습니다.");
		}
		else {
			Character poppedChar = this.stack().pop();
			if(poppedChar == null) {
				AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");
			}
			else {
				AppView.outputLine("[Pop] 삭제된 원소는 '" + poppedChar + "' 입니다");
			}
		}
	}
	private void popN(int numberOfCharsToBePopped) {
		if(numberOfCharsToBePopped == 0) {
			AppView.outputLine("[Pops] 삭제할 원소의 개수가 0개 입니다.");
		}
		else {
			int count = 0;
			while(count < numberOfCharsToBePopped && (!this.stack().isEmpty())) {
				Character poppedChar = this.stack().pop();
				if(poppedChar == null) {
					AppView.outputLine("(오류) 스택에서 삭제하는 동안에 오류가 발생하였습니다.");
				}
				else {
					AppView.outputLine("[Pops] 삭제된 원소는 '" + poppedChar + "' 입니다.");
				}
				count++;
			}
			if(count < numberOfCharsToBePopped) {
				AppView.outputLine("[Pops.Empth] 스택에 더이상 삭제할 원소가 없습니다.");
			}
		}
	}
	private void quitStackProcessing() {
		AppView.outputLine("");
		AppView.outputLine("<스택을 비우고 사용을 종료합니다>");
		this.showAllFromBottom();
		this.popN(this.stack().size());
	}
		// 출력 관련
	private void showAllFromBottom() {
		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() {
	AppView.output("[Stack] <Top> ");
		for(int order = this.stack().size()-1; order > -1; 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().elementAt(this.stack().size()-1)+ "' 입니다.");
		}
	}
	private void showStackSize() {
		AppView.outputLine("[Size] 스택에는 현재 " +this.stack().size() + " 개의 원소가 있습니다.");
	}
	private void showStatistics() {
		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();
	}
	private void addToStack(char aCharForAdd) {
		if (this.stack().isFull()) {
			AppView.outputLine("(오류) 스택이 꽉 차서, 더 이상 넣을 수 없습니다.");
		}
		else {
			Character charObjectForAdd = Character.valueOf(aCharForAdd);
			if(this.stack().push(charObjectForAdd)) {
				AppView.outputLine("[Push] 삽입된 원소는 '" + aCharForAdd + "' 입니다.");
			}
			else {
				AppView.outputLine("(오류) 스택에 넣는 동안에 오류가 발생하였습니다.");
			}
		}
	}

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

		char input = this.inputChar();
		while(input != '!') {
			this.countInputChar();
			if(Character.isAlphabetic(input)) {
				this.pushToStack(input);
				this.countPushedChar();
			}
			else if(Character.isDigit(input)) {
				this.popN(Character.getNumericValue(input));
			}
			else if(input == '-') {
				this.popOne();
			}
			else if(input == '#') {
				this.showStackSize();
			}
			else if(input == '/') {
				this.showAllFromBottom();
			}
```

```
                else if(input == '\\') {
                        this.showAllFromTop();
                }
                else if(input == '^') {
                        this.showTopElement();
                }
                else {
                        AppView.outputLine("[Ignore] 의미 없는 문자가 입력되었습니다.");
                        this.countIgnoredChar();
                }
                input = this.inputChar();
        }
        this.quitStackProcessing();

        this.showStatistics();
        AppView.outputLine("");
        AppView.outputLine("<<< 스택 기능 확인 프로그램을 종료합니다");
    }
}
```

AppView

```
import java.util.Scanner;

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

    private AppView() {}

    public static void output(String message) {
                System.out.print(message);
    }
    public static void outputLine(String message) {
                System.out.println(message);
    }
    public static void outputDebugMessage(String message) {
        System.out.println(message);
    }
    public static char inputChar() {
        String line = AppView.scanner.nextLine().trim();
        while(line.equals("")) {
                line = AppView.scanner.nextLine().trim();
        }
        return line.charAt(0);
    }
}
```

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;
		}
	}
}
```

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();
}
```