자료구조 실습 보고서

[제 7 주] 스택 기본 기능

제출일 : 2015 - 4 - 16

201402395 이승희

1. 프로그램 설명서
    1) 주요 알고리즘 / 자료구조 / 기타

    : 한 번에 한 개의 문자를 입력 받으며, 입력 받은 문자는 스택에 push 된다. '!'가 입력되면 프로그램이 종료되며 스택의 모든 원소를 pop 한다. 매번 한 개의 문자를 입력 받을 때마다, 문자에 따라 정해진 일을 한다. 제네릭 클래스와 인터페이스를 사용해본다.

    2) 함수 설명서
        (1) AppController
            - private void showAllFromBottom()

              : 스택의 모든 원소를 BOTTOM 부터 TOP 까지 출력한다.
            - private void showAllFromTop()

              : 스택의 모든 원소를 TOP 부터 BOTTOM 까지 출력한다.
            - private void showTopElement()

              : this._arrayStack 의 peek()을 이용하여 top 원소를 얻어온 후 AppView 의 outputTopElement 를 이용하여 출력한다.
            - private void showStackSize()

              : this._arrayStack 의 size()를 이용하여 원소의 개수를 얻어온 후 AppView 의 outputStackSize 를 이용하여 출력한다.
            - private void countAdded()

              : 삽입된 문자의 수를 1 씩 증가시킨다.
            - private void countIgnorded()

              : 무시된 문자의 수를 1 씩 증가시킨다.
            - private void countInputChar()

              : 입력된 문자의 수를 1 씩 증가시킨다.
            - private void addToStack(char inputChar)

              : 문자를 스택에 삽입한다.
            - private void removeOne()

              : 문자 한 개를 스택에서 삭제한다.
            - private void removeN(int numOfCharToBeRemoved)

              : 원하는 숫자만큼 원소의 개수를 스택에서 삭제한다. 입력된 수가 원소의 개수보다 클 경우 Error Message 를 출력한다.
            - private void conclusion()

              : 스택 내부의 있는 모든 원소를 pop 한다. 이 때, AppView 의 outputResult 를 이용하여 입력된 문자, 정상 처리된 문자, 무시된 문자, 삽입된 문자의 개수를 출력한다.
        (2) AppView
            - public int inputInt()

              : 정수를 입력받는다.
            - public String inputString()

              : 문자열을 입력받는다.

- public char inputCharacter()
  : Character 한 개 만을 입력받는 함수다. inputString 함수를 이용하여 문자열의 첫 번째 문자를 반환한다.
- public void outputAddedElement(char anElement)
  : 삽입된 문자 anElement 를 출력한다.
- public void outputStackElement(char anElement)
  : 원소 anElement 를 출력한다.
- public void outputTopElement(char anElement)
  : Top 에 있는 원소 anElement 를 출력한다.
- public void outputStackSize(int aStackSize)
  : 스택의 size 를 출력한다.
- public void outputRemove(char anElement)
  : 제거된 원소 anElement 를 출력한다.
- public void outputResult(int aNumOfInputChars, int aNumOfIgnoredChars, int aNumOfAddedChars)
  : 입력된 문자, 정상 처리된 문자, 무시된 문자, 삽입된 문자의 개수를 출력한다. 이 때, 정상 처리된 문자는 입력된 문자에서 무시된 문자의 차이만큼이다.

(3) Stack
- public interface Stack<T>
  : 추상 메소드 public boolan push(T anElement), public T pop(). Public T peek()로만 이루어져 있는 클래스다.

(4) ArrayList
- public boolean isEmpty()
  : *top* 을 확인하여 비어있는지 확인한다.
- public boolean isFull()
  : top 을 확인하여 가득 차 있는지 확인한다.
- public int size()
  : top 을 확인하여 삽입된 원소의 길이를 확인한다.
- public Boolean push(T anElement)
  : 스택에 원소 anElement 를 삽입한다. 스택이 가득 차 있을 경우 false 를 반환하고, 가득 차 있지 않을 경우 top 을 1 증가시키고 this._element 에 anElement 를 삽입한다. 스택의 개념에 따라 Resize 를 고려하지 않는다.
- public T pop()
  : 스택의 top 에 있는 원소를 삭제한다. 비어있으면 null 을 반환하고 비어있지 않으면 top 에 있는 element 의 원소를 반환하고 top 을 1 감소시킨다.
- public T peek()
  : 스택의 top 에 있는 원소를 반환한다. 스택이 비어있으면 null 을 반환하고 비어있지 않으면 top 에 있는 element 의 원소를 반환한다.
- public void clear()

: top 과 element 를 초기화한다.

- public T elementAt(int order)
  : 원하는 위치(order)의 원소를 반환한다.

3) 종합 설명서

: 영문자는 스택에 삽입되며 다른 문자의 입력은 다음과 같이 수행된다.

- '!' : 스택의 원소를 모두 삭제하고, 프로그램을 종료한다.
- '0'~'9' : 해당 수 만큼 스택에서 삭제한다. 매번 삭제될 때 마다 삭제된 원소를 출력하며, 스택이 비어있는 상태가 되면 삭제할 원소가 없다는 메시지를 출력하고 삭제를 멈춘다.
- '-' : 스택의 top 원소를 삭제한다.
- '#' : 스택의 길이를 출력한다.
- '/' : 스택의 내용을 bottom 부터 top 까지 순서대로 출력한다.
- '\' : 스택의 내용을 top 부터 bottom 까지 순서대로 출력한다.
- '^' : 스택의 top 원소의 값을 출력한다. 스택은 변하지 않는다.
- 그 밖의 문자들 : 다음과 같이 출력하고 무시한다.

Generic 클래스는 클래스를 사용할 때 자료형에 얽매이지 않는 장점이 있으며 이로인해 최상위 클래스인 "Object"를 사용하여 객체를 생성한다. 제네릭 클래스는 구체적인 타임을 대입하여 객체를 생성할 수 있는데 이 프로그램에서는
ArrayList<Character> _arrayStack = new ArrayList<Character>();
와 같이 자료형이 정해지지 않아 생기는 문제를 구체화함으로써 해결할 수 있다.

또한 인터페이스의 개념도 사용되었는데, 이는 상속의 개념과 비슷하여 (개념은 비슷하다 다중상속이 가능하고 함수의 header 만 선언할 수 있다는 차이점이 있다.) 상속받은 클래스에서 사용될 메소드를 정리해 놓을 수 있다.

2. 프로그램 장단점 분석

: 스택의 개념을 사용해서 원소를 추가하고 삭제할 때 스택의 사이즈를 고려하지 않아도 된다. 또한 원소를 추가하고 삭제하는 과정을 스택으로 이해하니 더 이해하기 쉬웠다. 단점이 있다면 이 구조에서 접근 가능한 유일한 객체는 가장 최근에 삽입된 객체이므로 top 원소 외에는 직접적으로 접근할 수 없다.

3. 실행 결과 분석

1) 입력과 출력

```
>프로그램을 시작합니다.
[스택 사용을 시작합니다.]
- 문자를 입력하시오 : S
[Push] 삽입된 원소는 'S' 입니다.
- 문자를 입력하시오 : t
[Push] 삽입된 원소는 't' 입니다.
- 문자를 입력하시오 : a
[Push] 삽입된 원소는 'a' 입니다.
- 문자를 입력하시오 : c
[Push] 삽입된 원소는 'c' 입니다.
- 문자를 입력하시오 : k
[Push] 삽입된 원소는 'k' 입니다.
- 문자를 입력하시오 : /
[Stack] <Bottom>  S  t  a  c  k  <Top>
- 문자를 입력하시오 : \
[Stack] <Top> k  c  a  t  <Top>
- 문자를 입력하시오 : j
[Full] 스택이 꽉 차서 삽입이 불가능합니다.
- 문자를 입력하시오 : -
[Pop] 삭제된 원소는 'k'입니다.
- 문자를 입력하시오 : #
[Size] 스택에는 현재 4개의 원소가 있습니다.
- 문자를 입력하시오 : -
[Pop] 삭제된 원소는 'c'입니다.
- 문자를 입력하시오 : r
[Push] 삽입된 원소는 'r' 입니다.
- 문자를 입력하시오 : /
[Stack] <Bottom>  S  t  a  r  <Top>
- 문자를 입력하시오 : 5
[Pop] 5개 원소를 삭제하려고 합니다.
[Pop] 삭제된 원소는 'r'입니다.
[Pop] 삭제된 원소는 'a'입니다.
[Pop] 삭제된 원소는 't'입니다.
[Pop] 삭제된 원소는 'S'입니다.
[Empty] 스택에 삭제할 원소가 없습니다.
- 문자를 입력하시오 : b
[Push] 삽입된 원소는 'b' 입니다.
- 문자를 입력하시오 : !
[스택 사용을 종료합니다.]
... . . 입력된 문자는 총 15개 입니다.
... . . 정상 처리된 문자는 7개 입니다.
... . . 무시된 문자는 0개 입니다.
... . . 삽입된 문자는 7개 입니다.
>프로그램을 종료합니다.
```

```
>프로그램을 시작합니다.
[스택 사용을 시작합니다.]
- 문자를 입력하시오 : a
[Push] 삽입된 원소는 'a' 입니다.
- 문자를 입력하시오 : p
[Push] 삽입된 원소는 'p' 입니다.
- 문자를 입력하시오 : p
[Push] 삽입된 원소는 'p' 입니다.
- 문자를 입력하시오 : &
[Error] 의미 없는 문자가 입력되었습니다.
- 문자를 입력하시오 : l
[Push] 삽입된 원소는 'l' 입니다.
- 문자를 입력하시오 : %
[Error] 의미 없는 문자가 입력되었습니다.
- 문자를 입력하시오 : e
[Push] 삽입된 원소는 'e' 입니다.
- 문자를 입력하시오 : !
[스택 사용을 종료합니다.]
... . . 입력된 문자는 총 7개 입니다.
... . . 정상 처리된 문자는 5개 입니다.
... . . 무시된 문자는 2개 입니다.
... . . 삽입된 문자는 5개 입니다.
>프로그램을 종료합니다.
```

2) 결과 분석

: 알파벳이 입력되면 스택에 삽입시키고 수행에 필요한 문자는 각각에 맞는 일을 한다. 의미없는 문자가 입력되면 무시되며 이에 맞는 Error Message 를 출력한다. 스택에 저장되는 원소는 최대 5 개로 제한하며 이를 초과하면 Error Message 를 출력한다.

4. 소스코드

1) AppController

```
public class AppController<T> {
    private AppView _appView;
    private int _inputChars;
    private int _ignoredChars;
    private int _addedChars;
    private ArrayList<Character> _arrayStack = new ArrayList<Character>();;

    public AppController() {
        this._appView = new AppView();
        this._inputChars = 0;
        this._ignoredChars = 0;
        this._addedChars = 0;
    }

    private void showAllFromBottom() {

        this.showMessage(MessageID.Show_StartBottom);
        for (int i = 0; i < this._arrayStack.size(); i++)
            this._appView.outputStackElement((char) this._arrayStack
                    .elementAt(i));
        this.showMessage(MessageID.Show_EndTop);
    }

    private void showAllFromTop() {
        this.showMessage(MessageID.Show_StartTop);
        for (int i = 0; i < this._arrayStack.size() - 1; i++)
            this._appView.outputStackElement((char) this._arrayStack
                    .elementAt(this._arrayStack.size() - i - 1));
        this.showMessage(MessageID.Show_EndTop);
    }

    private void showTopElement() {
        char topElement = (char) this._arrayStack.peek();
        this._appView.outputTopElement(topElement);
    }

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

    }

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

    }
```

```
private void countIgnored() {
    this._ignoredChars++;
}

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

@SuppressWarnings("unchecked")
private void addToSteck(char inputChar) {
    if (this._arrayStack.push(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 {
        char c = (char) this._arrayStack.pop();
        this._appView.outputRemove(c);
    }

}

private void remoneN(int numOfCharsToBeRemoved) {
    this._appView.outputRemoveN(numOfCharsToBeRemoved);
    for (int i = 0; i < numOfCharsToBeRemoved; i++)
        this.removeOne();

}

private void conclusion() {
    for (int i = this._arrayStack.size() - 1; i > -1; i--)
        this._arrayStack.pop();
    this._appView.outputResult(this._inputChars, this._ignoredChars,
            this._addedChars);

}
```

```
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.addToSteck(input);
        else if (input >= '0' && input <= '9')
            this.remoneN((int) (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.countIgnored();
        }
        input = this._appView.inputCharacter();
    }
    this.showMessage(MessageID.Notice_EndMenu);
    this.conclusion();
    this.showMessage(MessageID.Notice_EndProgram);
}
```

```
private void showMessage(MessageID aMessageID) {
    switch (aMessageID) {
    case Notice_StartProgram:
        System.out.println(">프로그램을 시작합니다.");
        break;
    case Notice_EndProgram:
        System.out.println(">프로그램을 종료합니다.");
        break;
    case Notice_StartMenu:
        System.out.println("[스택 사용을 시작합니다.]");
        break;
    case Notice_EndMenu:
        System.out.println("[스택 사용을 종료합니다.]");
        break;
    case Notice_InputStack:
        System.out.print("- 문자를 입력하시오 : ");
        break;
    case Show_StartBottom:
        System.out.print("[Stack] <Bottom> ");
        break;
    case Show_StartTop:
        System.out.print("[Stack] <Top>");
        break;
    case Show_EndBottom:
        System.out.println(" <Bottom>");
        break;
    case Show_EndTop:
        System.out.println(" <Top>");
        break;
    case Error_WrongMenu:
        System.out.println("[Error] 의미 없는 문자가 입력되었습니다.");
        break;
    case Error_InputFull:
        System.out.println("[Full] 스택이 꽉 차서 삽입이 불가능합니다.");
        break;
    case Error_RemoveEmpty:
        System.out.println("[Empty] 스택에 삭제할 원소가 없습니다.");
        break;
    }
}
```

2) AppView

```
import java.util.*;

public class AppView {

    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);
    }

    public int inputInt() {
        return this._scanner.nextInt();
    }

    public String inputString() {
        return this._scanner.next();
    }

    public char inputCharacter() {
        System.out.print("- 문자를 입력하시오 : ");
        return this.inputString().charAt(0);
    }

    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 aNumOfCharsToBeRemoved) {
        System.out.println("[Pop] "+aNumOfCharsToBeRemoved+"개 원소를 삭제하려고 합니다.");

    }

    public void outputResult(int aNumOfInputChars, int aNumOfIgnoredChars,
            int aNumOfAddedChars) {
        System.out.println("... . . 입력된 문자는 총 " + aNumOfInputChars + "개 입니다.");
        System.out.println("... . . 정상 처리된 문자는 "
                + (aNumOfAddedChars - aNumOfIgnoredChars) + "개 입니다.");
        System.out.println("... . . 무시된 문자는 " + aNumOfIgnoredChars + "개 입니다.");
        System.out.println("... . . 삽입된 문자는 " + aNumOfAddedChars + "개 입니다.");
    }


}
```

//정상 처리된 문자를 이후 (aNumOfInputChars - aNumOfIgnoredChars)로 바꾸었다.

3) ArrayList

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

    private static final int DEFAULT_MAX_STACK_SIZE = 5;
    private int _maxSize;
    private int _top;
    private T[] _elements;

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

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

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

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

    @Override
    public boolean push(T anElement) {
        if (this.isFull())
            return false;
        else {
            this._top++;
            this._elements[this._top]=anElement;
            return true;

        }
    }

    @Override
    public T pop() {
        if (this.isEmpty())
            return null;
        else {
            this._top--;
            return this._elements[this._top+1];
        }

    }
```

```
    @Override
    public T peek() {
        if (!this.isEmpty())
            return null;
        else {
            return this._elements[this._top];
        }
    }

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

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

}
```

4) Stack

```
public interface Stack<T> {

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


}
```

5) MessageID

```
public enum MessageID {
    // Message IDs for Notices:
    Notice_StartProgram,
    Notice_EndProgram,
    Notice_StartMenu,
    Notice_EndMenu,
    Notice_InputStack,
    Notice_DelStack,
    Notice_ShowStack,
    // MessageIDs for Shows:
    Show_StartBottom,
    Show_StartTop,
    Show_EndBottom,
    Show_EndTop,
    // MessageIDs for Errors:
    Error_WrongMenu,
    Error_InputFull,
    Error_RemoveEmpty,

}
```

6) DS1_07_201402395_이승희

```
public class DS1_07_201402395_이승희 {

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

    }

}
```

5. 정리

1) Stack

(1) 개념

: 스택이란 후입선출 프로토콜을 구현하는 자료 구조이다. 즉, 이 구조에서 접근 가능한 유일한 객체는 가장 최근에 삽입된 객체이다.

(2) 사용법

- peek : 스택이 공백이 아니면, top 의 원소를 반환한다.
- Pop : 스택이 공백이 아니면 , top 의 원소를 삭제한 후 반환한다.
- Push : 주어진 원소를 스택의 top 에 추가한다.

위의 연산들을 따라 스택의 원소를 배열에 저장하게 된다.

2) 직접 테스트 해본 Stack 의 모습

: 3.1)의 입출력 결과를 Stack 의 모습으로 도식화하면 다음과 같다.