자료구조: 2022년 1학기 [실습]

제 8 주:

스택: 기본기능

강 지 훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

실습 목표

실습 목표

- 이론적 관점
  - 스택의 개념
  - Java 의 인터페이스 개념
- 구현적 관점
  - 스택을 배열 리스트로 구현하는 방법
  - Generic Class 를 사용해 본다.
  - 인터페이스 (Interface) "Stack" 를 사용해 본다.

과제에서 해결할 문제

문제

- 키보드에서 문자를 반복 입력 받는다.
  - 한 번에 한 개의 문자를 입력 받는다.
- 입력의 종료 조건
  - '!' 키를 치면 더 이상 입력을 받지 않는다.
- 매번 한 개의 문자를 입력 받을 때 마다, 문자에 따라 정해진 일을 한다.

입출력 [1]

- 영문자 ('A' ~ 'Z', 'a' ~ 'z'): 스택에 삽입한다.
  - 다음과 같은 메시지를 내보낸다. (입력된 영문자가 'x' 라면)
    - "[Push] 삽입된 원소는 'x' 입니다."
  - 만일 스택이 full 이면 다음과 같은 메시지를 내보낸다.
    - [Full] 스택이 꽉 차서 원소 'x' 는 삽입이 불가능합니다.

- '!': 스택의 원소를 모두 삭제하고, 프로그램을 종료한다.
  - 스택의 원소를 삭제하기 전에 먼저 다음 메시지를 내보낸다:
    - "<스택을 비우고, 사용을 종료합니다>"
  - 원소를 차례로 삭제하여 스택을 비운다.
    삭제할 때마다 다음과 같은 메시지를 내보낸다. (삭제된 원소가 'x' 라면)
    - "[Pops] 삭제된 원소는 'x' 입니다."
  - 스택 사용 통계를 출력한다.
    - 입력된 문자의 개수
    - 정상 처리된 문자의 개수
    - 무시된 문자의 개수
    - 삽입된 문자의 개수

입출력 [2]

- 숫자문자('0' ~ '9'): 해당 수 만큼 스택에서 삭제한다.
  - 매번 삭제될 때마다 다음의 메시지를 내보낸다. (삭제된 원소가 'x' 라면)
    - "[Pops] 삭제된 원소는 'x' 입니다.
  - 삭제를 하려는데 스택이 empty 이면, 다음과 같은 메시지를 내보내고 삭제를 멈춘다.
    - "[Pops.Empty] 스택에 더 이상 삭제할 원소가 없습니다."

- '-': 스택의 top 원소를 삭제한다.
  - 다음과 같은 메시지를 내보낸다. (Top 원소가 'r' 라면)
    - "[Pop] 삭제된 원소는 'r' 입니다.
  - 삭제를 하려는데 스택이 empty 이면, 다음과 같은 메시지를 내보내고 삭제를 멈춘다.
    - "[Pop.Empty] 스택에 삭제할 원소가 없습니다."

- '#': 스택의 길이를 다음과 같이 출력한다. (현재 스택에 5 개의 원소가 있다면)
  - "[Size] 스택에는 현재 5 개의 원소가 있습니다."

입출력 [3]

- '/': 스택의 내용을 Bottom 부터 Top 까지 다음과 같이 차례로 출력한다.
  - "[Stack] <Bottom> A b c d E f <Top>"
- '₩': 스택의 내용을 Top 부터 Bottom 까지 다음과 같이 차례로 출력한다.
  - "[Stack] <Top> f E d c b A <Bottom>"
- '^': 스택의 top 원소의 값을 출력한다. 스택은 변하지 않는다.
  - (스택의 top 원소가 'f' 라면) 다음과 같이 출력한다.
    - "[Top] 스택의 Top 원소는 'f' 입니다."
  - 스택이 비어 있는 상태이면 다음과 같이 출력한다:
    - "[Top.Empty] 스택이 비어서 Top 원소가 존재하지 않습니다."
- 그 밖의 문자들: 다음과 같이 출력하고 무시한다.
  - "[Ignore] 의미 없는 문자가 입력되었습니다."

출력의 예 [1]

```
<<< 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: p
(오류) 스택이 꽉 차서, 더이상 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: 5
[Pops] 삭제된 원소는 'W' 입니다.
[Pops] 삭제된 원소는 'h' 입니다.
[Pops] 삭제된 원소는 'x' 입니다.
[Pops] 삭제된 원소는 'A' 입니다.
[Pops.Empty] 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 >>>
```

출력의 예: [2]

```
<<< 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: !

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

<스택 사용 통계>
- 입력된 문자는 6 개 입니다.
- 정상 처리된 문자는 5 개 입니다.
- 무시된 문자는 1 개 입니다.
- 삽입된 문자는 2 개 입니다.

<<< 스택 기능 확인 프로그램을 종료합니다 >>>
```

의미 없는 문자의 처리

"Enter" 키를 쳤을 때에도,
다음 줄에서 입력을 받게 한다.

문자열의 앞뒤 공백 문자를 제거하고 입력 받는다.

구현할 내용

과제에서 필요한 Class / Interface

이 과제에서 필요한 Class / Interface

- Class "AppController"
- Class "AppView"
- Model
  - Interface Stack<T>
  - Class "ArrayList<T>" (implements "Stack<T>")

main()

main() 을 위한 class

```
public class _DS07_학번_이름 {
    public static void main (String[] args)
    {
        AppController  appController = new AppController() ;
            // AppController 가 실질적인 main class 이다.
        appController.run() ;
            // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
    }
}
```

Class "AppController"

AppController: 상수, 인스턴스 변수, getters/setters

```
public class AppController {
        // 상수
        private static final int STACK_CAPACITY = 5 ;

        // 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() {…}
        private void setStack (ArrayList<Character> newStack) {…}
        private int inputChars() {…}
        private void setInputChars(int newInputChars) {…}
        private int pushedChars() {…}
        private void setPushedChars(int newPushedChars) {…}
        private int ignoredChars() {…}
        private void setIgnoredChars(int newIgnoredChars) {…}

        // 생성자
        ……

        // 비공개 함수
        ……

        // 공개 함수
        ……
} // End of class "AppController"
```

응용에 맞게 설정하면 될 것임.
앞 슬라이드의 "출력 예" 는 크기가 "5" 로 설정된 것임.

AppController: 생성자

```
public class AppController {
        // 비공개 변수들
        ……

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

        // 비공개 함수
        ……

        // 공개 함수
        ……
} // End of class "AppController"
```

AppController: 비공개 함수

```
public class AppController {
    ...
    // 비공개 함수
    // 횟수 계산
    private void  countInputChar () {...}
    private void  countIgnoredChar () {...}
    private void  countPushedChar () {...}

    // 스택 수행 관련
    private void  pushToStack (char aCharForPush) {...}
    private void  popOne () {...}
    private void  popN (int numberOfCharsToBePopped) {...}
    private void  quitStackProcessing() {...}

    // 출력 관련
    private void showAllFromBottom() {...}
    private void showAllFromTop() {...}
    private void showTopElement() {...}
    private void showStackSize() {...}
    private void showStatistics() {...}

    // 입력 관련
    private char inputChar() {...}
```

AppController: showAllFromBottom(), showAllFromTop()

- private void showAllFromBottom() {

```
    // 스택의 모든 원소를 Bottom 부터 Top 까지 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()
  - this.stack() 의 elementAt() 을 이용하여 Top 부터 Bottom 까지 출력
  - showAllFromBottom() 을 참고하여 작성

AppController: showTopElement(), showStackSize()

- private void showTopElement()
  - Stack 객체의 peek() 을 이용하여 Top 원소를 출력
  - Stack 이 비어 있으면, 알림 메시지 출력
    - 문제 설명과 출력 예제를 참고할 것

- private void showStackSize()
  - Stack 객체의 size() 를 이용하여 원소의 개수를 출력

```
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.");
}
```

AppController: countInputChars(),…

- private void countInputChar()
  - this.setInputChars (this.inputChars()+1) ;
- private void countIgnoredChar()
  - this.setIgnoredChars (this.ignoredChars()+1) ;
- private void countPushedChar()
  - this.setPushedChars (this.pushedChars()+1) ;

AppController: pushToStack()

```
private void  pushToStack (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.") ;
        }
    }
}
```

문자를 문자 객체로 변환

AppController: popOne()

```
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.");
        }
    }
}
```

AppController: popN()

```
private void popN (int numberOfCharsToBePopped) {
    if (numberOfCharsToBePopped == 0) {
        AppView.outputLine("[Pops] 삭제할 원소의 개수가 0 개 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) {
            // Stack has become emtpy before we remove N elements.
            AppView.outputLine("[Pops.Empty] 스택에 더이상 삭제할 원소가 없습니다.");
        }
    }
}
```

AppController: quitStackProcessing()

```
private void quitStackProcessing() {
    AppView.outputLine("");
    AppView.outputLine("<스택을 비우고 사용을 종료합니다>");
    this.showAllFromBottom();
    this.popN(this.stack().size());
}
```

AppController: inputChar()

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

AppController: run() [1]

```
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();
        }
```

알파벳 검사

숫자 문자 검사

숫자 문자를 정수 값으로 변환

AppController: run() [2]

```
        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 >>>") ;
} // End of Run()
```

Back slash (\ ) 는 Java 문자열에서 사용하는 특수 문자. Java 가 원래의 문자로 인식하게 하려면 이렇게 두 번 표기한다.

Class "AppView"

AppView

```
public class  AppView {
  // 비공개 상수/변수들
  private static Scanner  scanner = new Scanner(System.in) ;


  // 공개함수의 구현

  // 출력을 위한 공개함수
  public static void outputLine(String aMessage) {…}
  public static void output (String aMessage) {…}
  ……

  // 입력을 위한 공개함수
  public static char inputChar() {…}
}
```

AppView: inputChar()

- 문자 한 개를 입력 받는다.
  - 문자열 한 줄을 입력 받아, 그 줄의 맨 처음 공백이 아닌 문자를 취하기로 한다.
  - 문자열의 앞뒤 공백 없애기: trim()
  - 공백을 없앤 입력된 문자열이 비어 있지 않을 때까지 입력받는다.

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

}
```

문자열의 앞뒤 공백 문자를 제거해 준다.

입력된 문자열이 비어 있다.

Interface "Stack<E>"

인터페이스 (Interface)

- Abstract Class 와 유사하다:
  - (Abstract) Method 의 정의로만 구성된다.
    - 당연히 구현 코드가 주어지지 않는다.
    - 함수의 header 만 정의된다.
- 그렇지만, class 는 아니다:
  - 상수나 인스턴스 변수를 선언할 수 없다.
  - 생성자가 존재할 수 없다.
  - 상속의 대상이 아니다:
    - "extends" 할 수 없다.
- 사용법:
  - "extends" 대신 "implements" 를 사용한다.
  - 다중 "implements" 가 가능하다: 다중 상속?
    - "extends" 는 단 하나의 class 로부터의 상속 만을 허락한다.

Interface "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();

}
```

스택을 구현하는
Class "ArrayList<E>"
(implements Stack<E>)

ArrayList:

- 이전 과제의 "UnsortedArrayList" 를 복사해서 사용한다.
- Interface "Stack" 구현을 추가한다.

ArrayList: 상수, 인스턴수 변수

```
public class ArrayList<E extends Comparable<E>>
        implements Stack<E>
{
        // 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() {...}
        private void setCapacity (int newCapacity) {...}

        @Override
        public int size() {...}
        private void setSize (int newSize) {...}

        private E[] elements() {...}
        private void setElements(E[] newElements) {...}
```

Interface "Stack" 을 구현해야 함을 선언

@Override:
Interface Stack 에 선언되어 있어, 구현해야 하는 함수.

ArrayList: 생성자

```
public class ArrayList<E extends Comparable<E>>
        implements Stack<E>
{
        // Constant
        ......
        // Private instance variables
        ......
        // Getters/Setters
        ......

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

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

Generic Type E 의 선언:
E 를 대체할 실제 class 는 반드시 Interface "Comparable<E>" 를 구현하고 있어야 한다.

구현되어 있는 다른 생성자를 사용

Generic Type E 의 선언을 볼 것
<E extends Comparable<E> >

ArrayList: Private methods

```
public class ArrayList<E extends Comparable<E>>
        implements Stack<E>
{
        ......

        // Private Methods
        private void makeRoomAt (int aPosition) {...}
        private void removeGapAt (int aPosition) {...}

        // Public Methods
        ......
```

이전 과제를 보고,
각자 구현할 것

ArrayList: Public methods

```
public class ArrayList<E extends Comparable<E>>
        implements Stack<E>
{       ……

        // Public Methods
        @Override
        public boolean isEmpty() {…}
        @Override
        public boolean isFull() {…}

        public boolean doesContain (E anElement) {…}
        public int orderOf (E anElement) {…} //
        public E elementAt (int anOrder) {…}
        public void setElementAt (int anOrder, E anElement) {…}

        public boolean addTo (int anOrder, E anElement) {…}
        public boolean addToFirst (E anElement) {…}
        public boolean addToLast (E anElement) {…}

        public E removeFrom (int anOrder) {…}
        public E removeFirst() {…}
        public E removeLast() {…}
```

강의 자료와
이전 과제를 보고,
각자 구현할 것

ArrayStack: push(), pop(), peek()

```
public class ArrayList<E extends Comparable<E>> implements Stack<E>
{       ……

        // Public Methods
        ……

        @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
              }
        }
}
```

ArrayList: 공개함수

- @Override 가 붙은 함수
  - Interface Stack<T> 에 선언되어 override 되고 있는 함수

요약

확인하자

- 주요 개념:
  - Generic Type 의 개념, 사용법
  - Interface 의 개념, 사용법
  - Stack 의 개념, 사용법
- 구현:
  - ArrayList 를 사용하여 Stack 을 구현하는 방법

생각해 볼 점

- Abstract class 와 interface 의 유사점과 차이점은?
  - 특별히, Abstract class 를 상속 받는 것과 interface 를 구현하는 것의 유사점과 차이점은?

⇒ 생각해 볼 점에 대해, 자신의 의견을 보고서에 작성하시오.

[실습 끝]