

스택 (Stack)

강 지 훈

jhkang@cnu.ac.kr



스택(Stack) 이란?



□ 스택 (Stack)

- 더미(stack): 쌓여 있는 것들
- 쌓인 순서가 있다
- 새로운 것을 쌓기 편한 곳은?
- 스택에서 하나를 빼 내기 편한 곳은?





□ 쓸모 [1]

■ 웹 브라우저: 방문 History 관리

- 링크 클릭: 현재의 주소를 “방문 History 스택”에 쌓는다
- Back 단추 클릭: 스택에서 직전에 쌓은 주소를 꺼낸다

■ 수식 계산기

■ 컴파일러(Compiler) / 인터프리터(Interpreter)

- 고급 언어를 기계어로 번역 (compile) / 실행 (interpret)
- 구문 분석 (Parsing)
 - ◆ 컴파일러는 수식 계산식을 포함하는, 보다 일반화된 복잡한 고급언어 표현을 번역

□ 쓸모 [2]


■ 재귀 함수의 처리

- Activation record: 현재 실행 중인 함수에서 다른 함수 call 이 발생하는 시점에, 스택에 쌓아서 보관하려는 현재의 함수의 실행 정보:
 - ◆ 현재의 함수의 지역변수들의 값
 - ◆ Call이 처리된 후에 return 될 때 돌아와야 할 현재 함수 내의 위치
- call: 현재의 activation record를 스택에 쌓는다
- return: 직전의 activation record를 꺼내어 이전 상태로 돌아간다


□ 스택 (Stack)

- 원소의 삽입과 삭제가 순서 리스트의 한쪽 끝에서만 발생
 - 스택의 **꼭대기(top)**: 삽입과 삭제가 발생하는 한쪽 끝
 - 사용자가 얻을 수 있는 유일한 원소는 가장 최근에 삽입된 원소

$$S = (e_0, \dots, e_{n-1})$$



bottom 원소

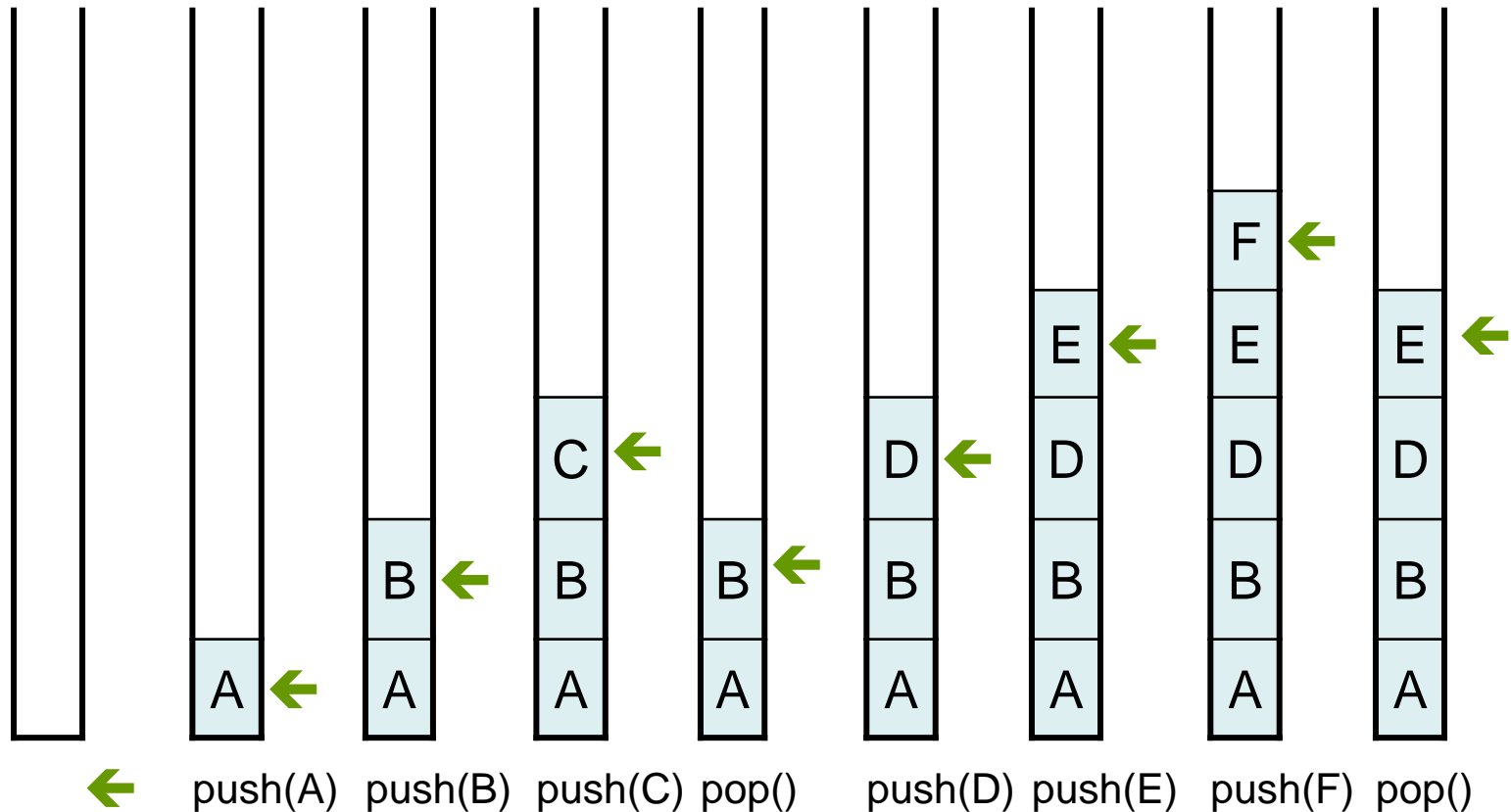


top 원소

- 후입선출 (後入先出)
 - LIFO: last-in-first-out

□ 삽입(push)과 삭제(pop)

"←" : 스택의 꼭대기(top) 원소를 가리킨다



□ 스택의 사용법 (공개함수)

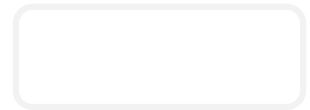
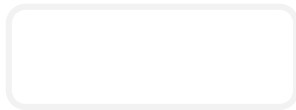
```
public class Stack<Element>
{
    public                Stack() { } // 생성자

    public boolean        isEmpty() { }
    public boolean        isFull() { }
    public int            size() { }

    public boolean        push(Element anElement) { }
    public Element        pop() { }
    public Element        peek() { }
    public void            clear() { }
}
```

- isEmpty() : 스택이 비어있는지를 알려 준다.
- isFull() : 스택의 꽉 차서 더 이상 삽입할 수 없는 상태인지를 알려준다.
- size() : 스택에 있는 원소의 수를 얻는다.
- push() : 주어진 원소를 스택의 맨 위에 올려 놓는다.
- pop() : 비어 있는 스택이 아니면, 가장 꼭대기의 원소를 빼내어 얻는다.
- peek() : 비어 있는 스택이 아니면, 가장 꼭대기의 원소를 얻는다. 스택은 변하지 않는다.
- clear() : 스택을 비운다.

Class "ArrayStack"



□ ArrayStack 의 공개함수

■ Stack 객체 사용법을 Java로 구체적으로 표현

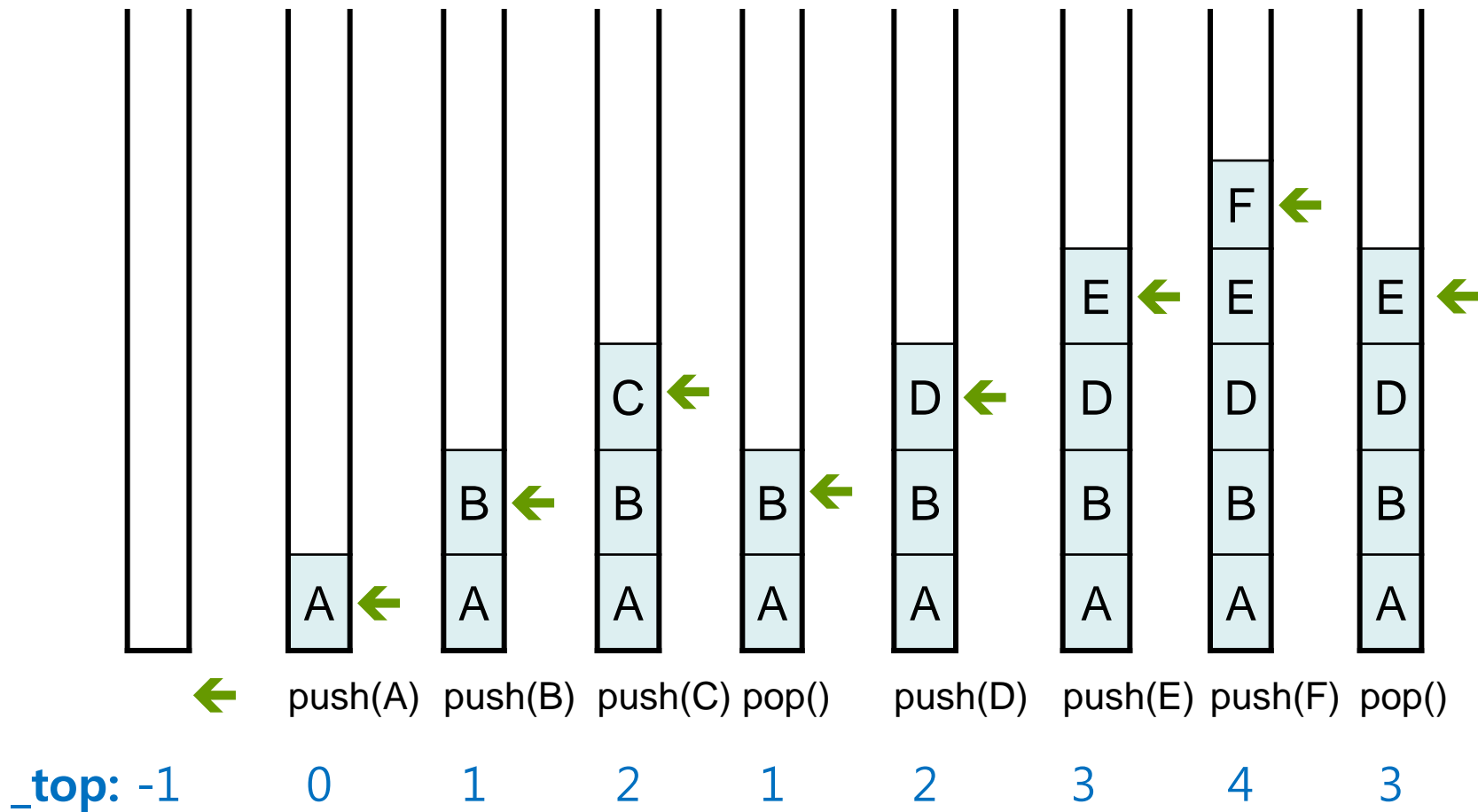
- public `ArrayStack() { }`
- public boolean `isEmpty() { }`
- public boolean `isFull() { }`
- public int `size() { }`
- public boolean `push(Element anElement) { }`
- public T `pop() { }`
- public T `peek() { }`
- public void `clear() { }`

□ Class "ArrayStack"의 구현: 멤버변수

```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    private static final int DEAFULT_MAX_STACK_SIZE = 50 ;
    private    int          _maxSize ;
    private    int          _top ;
    private    Element[]    _elements ;
}
```

- DEAFULT_MAX_STACK_SIZE: 사용자가 지정하지 않을 경우에 사용될 초기 스택 크기
- _maxSize: 현재 스택의 최대 크기
- _top : 현재 스택의 top 원소의 배열에서의 위치
- _elements : 스택의 원소들이 저장되는 배열

□ 배열로 구현된 스택에서의 삽입/삭제



□ 일반화된(generic) 클래스

```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    private static final int MAX_STACK_SIZE = 50 ;
    private int _maxSize ;
    private int _top ;
    private Element[] _elements;
```

Java.lang.Object

org.glassfish.gmbal.util.GenericConstructor<Element>

- 스택의 선언에서 원소의 자료형(클래스)을 매개변수처럼 사용
 - Element는 구체화 되지 않은 상태의 object class이다.
- ArrayStack<Element>는 일반화된 클래스로 선언되었다.
 - 사용자가 필요한 스택 객체를 선언할 때, Element가 구체적으로 주어지게 된다.
- 목적: 클래스를 정의하는 코드의 재활용
 - 동일한 기능을 하는 스택을, 원소가 달라질 때마다 따로 정의할 필요가 없다
 - 편의성, 효율성

□ 일반화된 클래스의 사용법

```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    private static final int MAX_STACK_SIZE = 50 ;
    private int _maxSize ;
    private int _top ;
    private Element[] _elements;
```

■ 일반화된 클래스 변수의 선언과 사용

```
ArrayStack<String> wordStack = new ArrayStack<String>() ;
wordStack.push("Hello");
wordStack.push("abc");
```

- 원소의 자료형이 String인 ArrayStack 객체가 생성되고, wordStack이 소유한다.

□ Class "ArrayStack"의 구현: 생성자

```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    .....

    // 생성자
    public ArrayStack ( )
    {
        this._elements =
            (Element[]) new Object[ArrayStack.DEFAULT_MAX_STACK_SIZE] ;
        this._maxSize = ArrayStack.DEFAULT_MAX_STACK_SIZE ;
        this._top = -1 ;
    }
}
```

□ ArrayStack: 상태 알아보기

```
public class ArrayStack<Element>
{
```

```
.....
```

```
// 비공개 함수
```

```
.....
```

```
// size
```

```
public int size()
```

```
{
```

```
    return (this._top + 1) ;
```

```
}
```

```
// Stack이 비어있는지 확인
```

```
public boolean isEmpty ()
```

```
{
```

```
    return (this._top < 0);
```

```
}
```

```
// Stack이 꽉차 있는지 확인
```

```
public boolean isFull ()
```

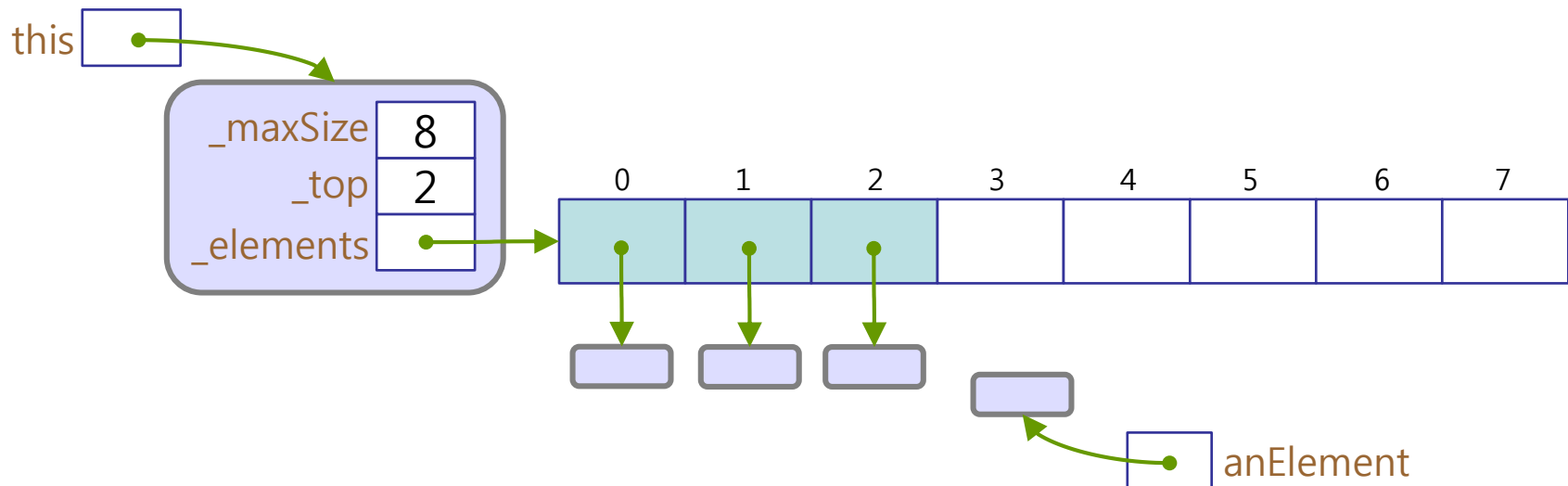
```
{
```

```
    return ( (this._top+1) == this._maxSize ) ;
```

```
}
```

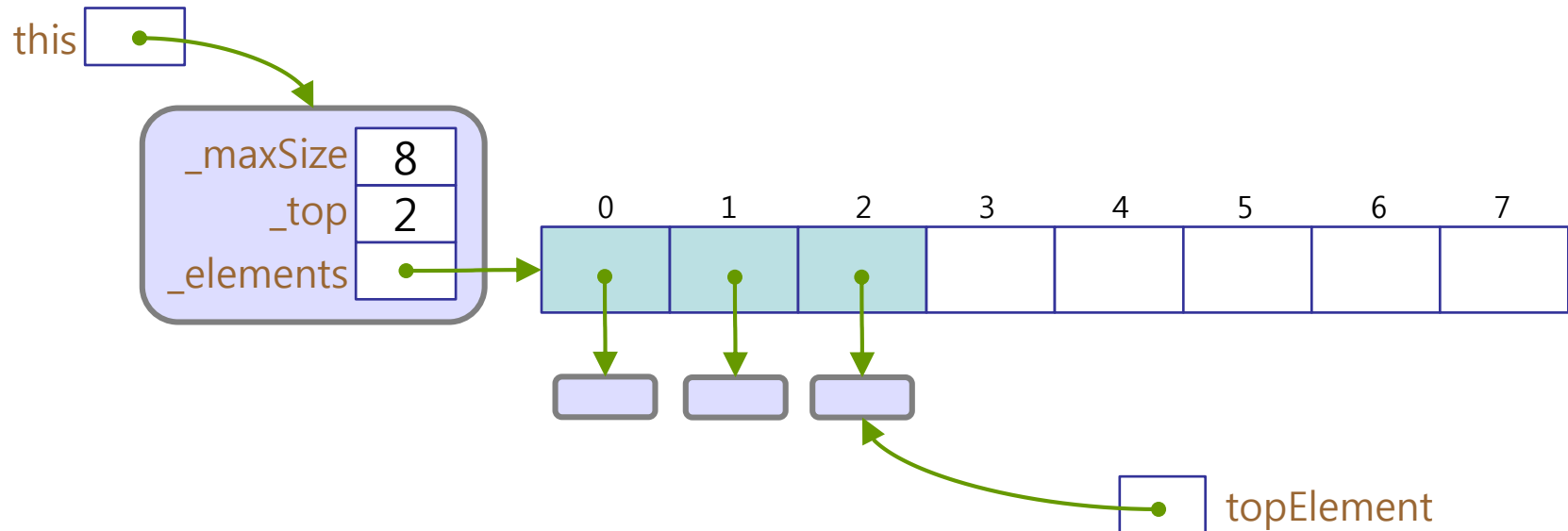
❏ ArrayStack: push()

```
// Push
public boolean push(Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        this._top++;
        this._elements[this._top] = anElement ;
        return true ;
    }
}
```



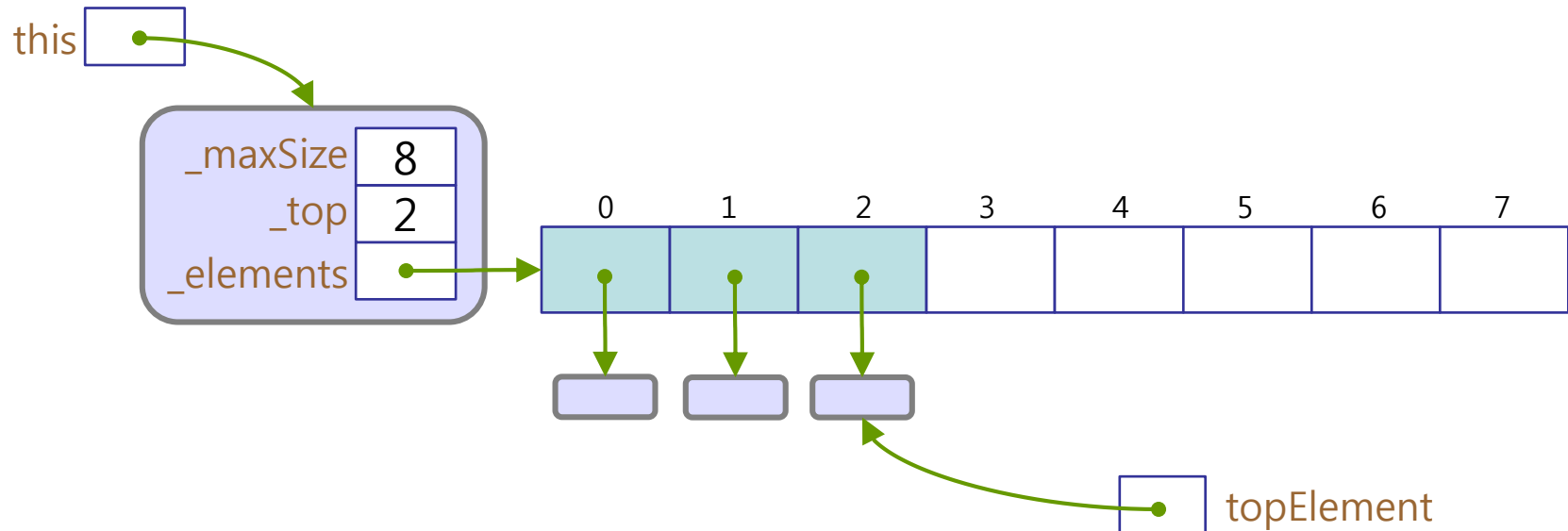
❑ ArrayStack: peek()

```
// peek
public Element peek()
{
    Element topElement = null ;
    if ( ! this.isEmpty() ) {
        topElement = this._elements[this._top] ;
    }
    return topElement ;
}
```



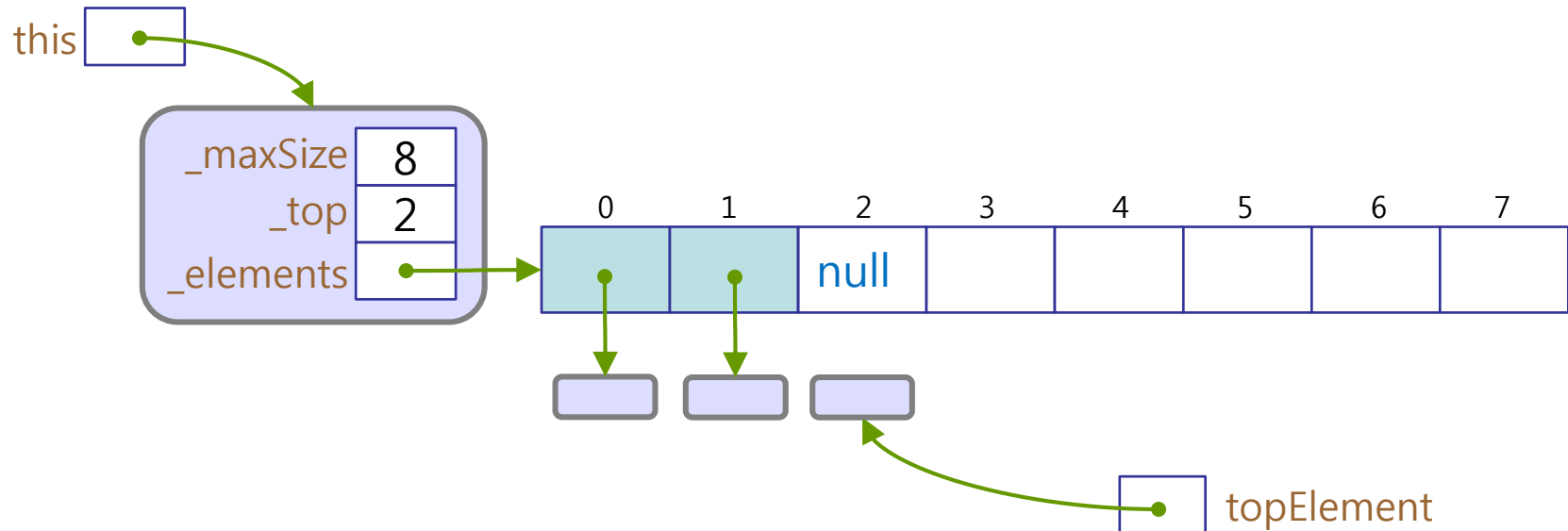
❑ ArrayStack: pop() [1]

```
// pop
public Element pop()
{
    Element topElement = null ;
    if ( ! this.isEmpty() ) {
        topElement = this._elements[this._top];
        this._elements[this._top] = null ;
        this._top--;
    }
    return topElement ;
}
```



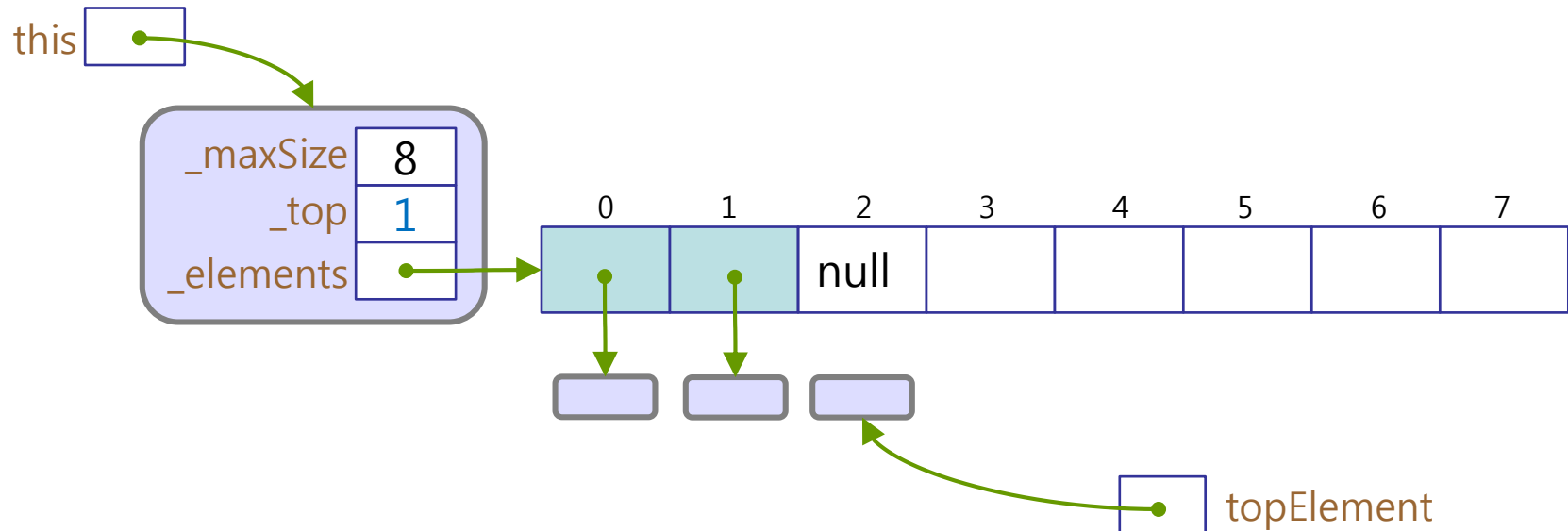
❑ ArrayStack: pop() [2]

```
// pop
public Element pop()
{
    Element topElement = null ;
    if ( ! this.isEmpty() ) {
        topElement = this._elements[this._top] ;
        this._elements[this._top] = null ;
        this._top-- ;
    }
    return topElement ;
}
```



❑ ArrayStack: pop() [3]

```
// pop
public Element pop()
{
    Element topElement = null ;
    if ( ! this.isEmpty() ) {
        topElement = this._elements[this._top] ;
        this._elements[this._top] = null ;
        this._top-- ;
    }
    return topElement ;
}
```



ArrayStack: clear()

```
// clear
public void clear()
{
    while (this._top >= 0) {
        this._elements[this._top] = null ;
        this._top-- ;
    }
}
```

□ Full 처리 방법: `resize()`

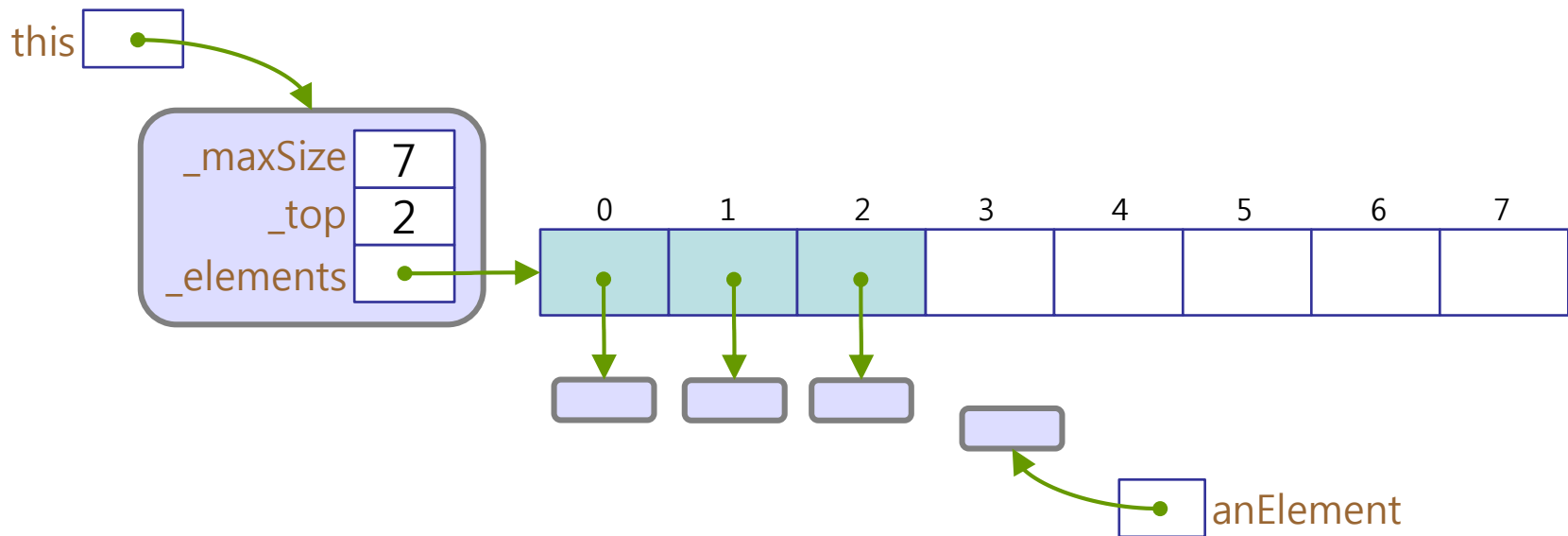
```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    .....
    // 비공개 함수
    private void resize()
    {
        // 배열의 크기를 2배로 늘려준다.
        this._maxSize *= 2 ;
        Element [] oldElements = this._elements ;
        this._elements = new Elements[this._maxSize] ;
        for (int i= 0 ; i <= this._top ; i++) {
            this._elements[i] = oldElements[i] ;
            oldElements[i] = null ;
        }
        // this._elements = Arrays.copyOf(this._elements, this._maxSize);
    }
}
```

□ Full 처리 방법: `resize()`

```
public class ArrayStack<Element>
{
    // 비공개 멤버 변수
    .....
    // 비공개 함수
    private void resize()
    {
        // 배열의 크기를 2배로 늘려준다.
        this._maxSize *= 2 ;
        // Element [] oldElements = this._elements ;
        // this._elements = new Elements[this._maxSize] ;
        // for (int i= 0 ; i <= this._top ; i++) {
        //     this._elements[i] = oldElements[i] ;
        //     oldElements[i] = null ;
        // }
        this._elements = Arrays.copyOf(this._elements, this._maxSize);
    }
}
```

❑ ArrayStack: push() // resize() 사용

```
// Push with resize()
public boolean push(Element anElement)
{
    if ( this.isFull() ) {
        this.resize() ;
    }
    this._top++;
    this._elements[this._top] = anElement ;
    return true ;
}
```



Interface in Java



□ 기존 class의 특정 기능 확장은?

■ Class "List<Element>"의 공개 함수

- `public boolean isEmpty () { }`
- `public boolean isFull () { }`
- `public boolean size () { }`

- `public Element elementAt (int aPosition) { }`
- `public Element last () { }`

- `public boolean addToLast (Element anElement) { }`
- `public Element removeLast () { }`
- `public void clear() { }`

□ 기존 class의 특정 기능 확장은?

■ Class "Stack<Element>"의 공개 함수

- `public boolean isEmpty () { }`
- `public boolean isFull () { }`
- `public boolean size () { }`

- `public Element elementAt (int aPosition) { }`
- `public Element last () { }`

- `public boolean addToLast (Element anElement) { }`
- `public Element removeLast () { }`
- `public void clear() { }`

- `public boolean push (Element anElement) { }`
- `public Element pop () { }`
- `public Element peek () { }`

□ 기존 class의 특정 기능 확장은?

- Stack은 List의 특수한 경우이다:
 - Stack 고유의 기능을 하도록 하는 공개함수가 추가되었다
 - ◆ push() / pop() / peek()
- 이런 경우에 Stack을 별도의 class로 정의하는 대신, Java Interface로 선언할 수 있다.
- 즉, Stack 고유의 공개함수를 모아서 Interface로 선언한다.
- 그리고, class "List"는 interface "Stack"을 구현(implements)한다.

□ Java interface 로서의 스택

```
public interface Stack<Element>
{
    public abstract boolean    push (Element anElement) ;
    public abstract Element    pop() ;
    public abstract Element    peek() ;
}
```

■ Interface는 단지 함수 header만 정의한다.

- 코드 구현은 없다
- 따라서, 반드시 "abstract" 가 있어야 한다.

□ Java interface 사용

```
public class ArrayList<Element> implements Stack<Element>
{
    public boolean    isEmpty () { }
    public boolean    isFull () { }
    public boolean    size () { }

    public Element    elementAt (int aPosition) { }
    public Element    last () { }

    public boolean    addToLast (Element anElement) { }
    public Element    removeLast () { }
    public void        clear() { }

    // 구현은 이곳 ArrayList 안에서
    public boolean push (Element anElement) { }
    public Element pop () { }
    public Element peek () { }
}
```

□ Java interface 구현 [1]

```

public class ArrayList<Element> implements Stack<Element>
{
    private int size;
    ..... // 필요한 변수 선언

    public boolean isEmpty () { }
    public boolean isFull () { }
    public boolean size () { }

    public Element elementAt (int aPosition) { }
    public Element last () { }

    public boolean addToLast (Element anElement) { }
    public Element removeLast () { }
    public void clear () { }

    // 구현은 이곳 ArrayList 안에서
    public boolean push (Element anElement)
    {
        if ( this.isFull() ) {
            return false ;
        }
        else {
            this_elements[this_size] = anElement ;
            this_size++;
            return true ;
        }
    }
    public Element pop ()
    {
        Element topElement = null ;
        if ( ! this.isEmpty() ) {
            this_size-- ;
            topElement = this_elements[this_top] ;
            this_elements[this_top] = null ;
        }
        return topElement ;
    }
    public Element peek ()
    {
        Element topElement = null ;
        if ( ! this.isEmpty() ) {
            topElement = this_elements[this_size-1] ;
        }
        return topElement ;
    }
}

```

□ Java interface 구현 [2]

```

public class ArrayList<Element> implements Stack<Element>
{
    private int _size;
    ..... // 필요한 변수 선언

    public boolean    isEmpty () { }
    public boolean    isFull () { }
    public boolean    size () { }

    public Element    elementAt (int aPosition) { }
    public Element    last () { }

    public boolean    addToLast (Element anElement) { }
    public Element    removeLast () { }
    public void        clear() { }

    // 구현은 이곳 ArrayList 안에서 : 또 다른 구현 방법
    public boolean push (Element anElement)
    {
        return this.addToLast(anElement) ;
    }
    public Element pop ()
    {
        return this.removeLast();
    }
    public Element peek ()
    {
        return elementAt (this.size()-1);
    }
}

```

□ LinkedList도 스택으로

```
public class LinkedList<Element> implements Stack<Element>
{
    public boolean    isEmpty () { }
    public boolean    isFull () { }
    public boolean    size () { }

    public Element    elementAt (int aPosition) { }
    public Element    last () { }

    public boolean    addToLast (Element anElement) { }
    public Element    removeLast () { }
    public void        clear() { }

    // 구현은 이곳 LinkedList 안에서
    public boolean push (Element anElement) { }
    public Element pop () { }
    public Element peek () { }
}
```

실습: 수식 계산기



□ 실습: 수식 계산기

- ArrayStack과 LinkedStack 두 가지 방법으로 구현해본다.
- 스택이 필요한 곳
 - 중위(infix) 연산식을 후위(postfix) 연산식으로 변환할 때
 - 후위 연산식을 계산할 때

“Stack” [끝]



