스택Stack

1. ADT Stack
2. Array Stack (Stack on Array)
3. Linked Stack (Stack by Link)
4. Reusing List
5. Stack 활용

1. ADT Stack

도입을 위한 예

'←' in keyboard input line

e.g., abcd←←efgh←←←ij←km←

결과: abeik

한 문자를 읽어 '←' 이 아니면 저장하고

'←' 이면 **최근에 저장된** 문자를 제거한다.

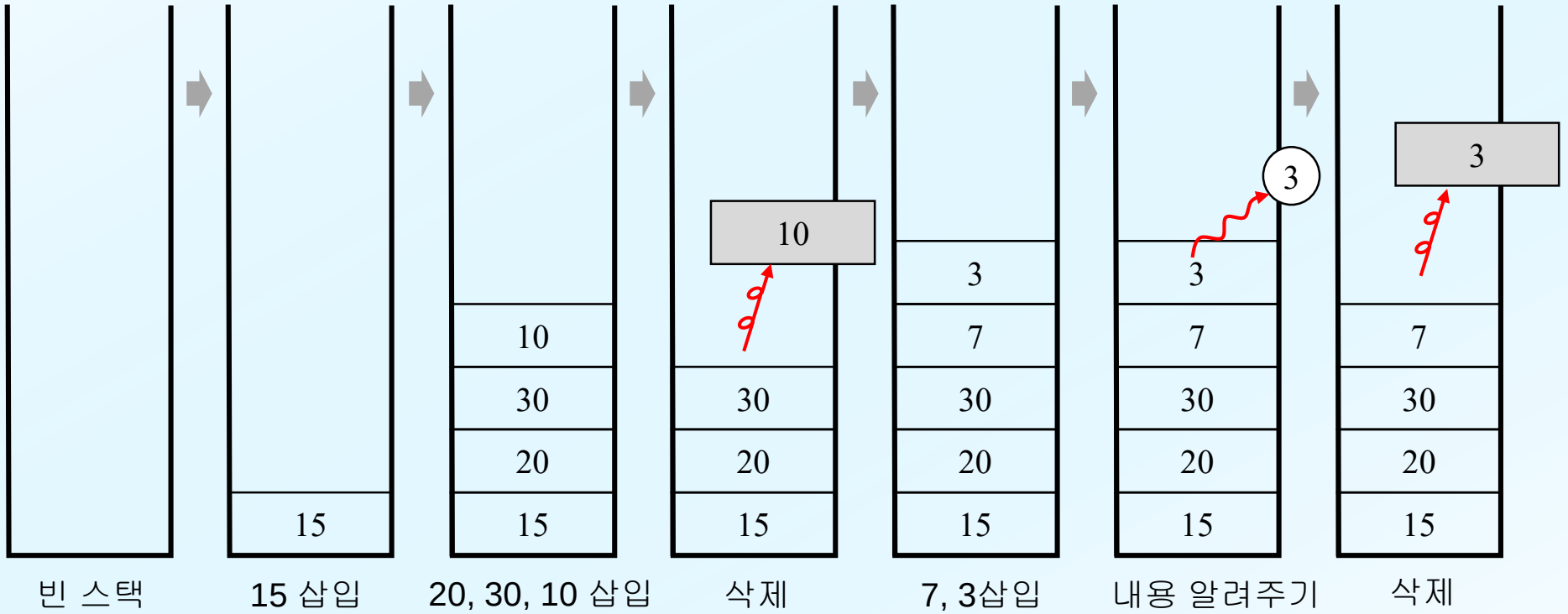
실생활에서 스택의 예

식판의 스택

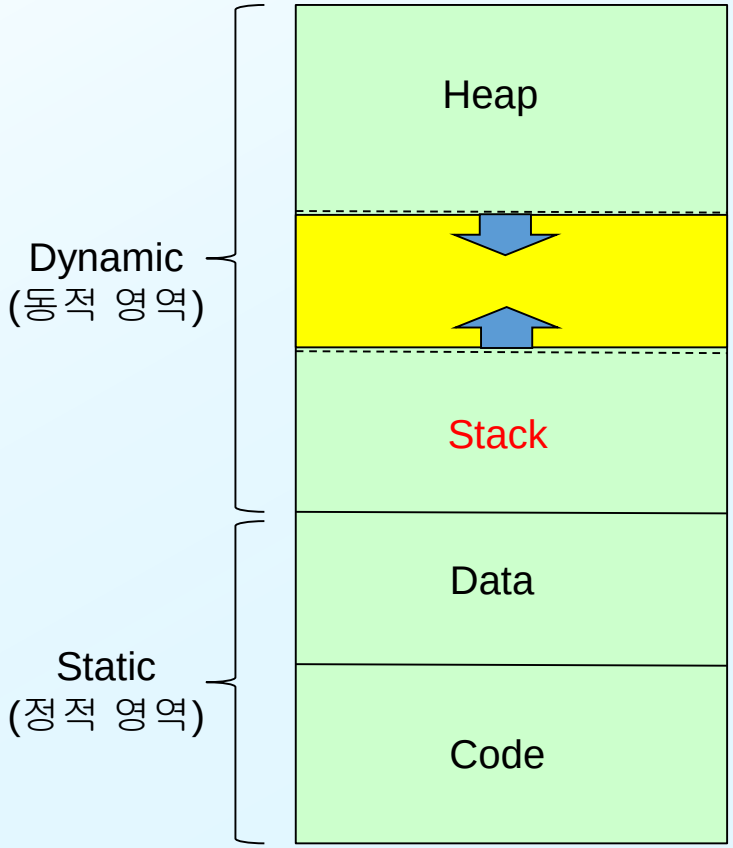
최근에 쌓은 식판을 꺼낸다

Stack

넣을 때는 위로 쌓아올리고,
뺄 때는 위에서부터 뺀다



프로그램 수행과 스택 활용



수행중에 사용자가 할당받는 메모리
예: 자바의 new, C의 malloc()

active 상태의 함수 관련 파라미터와 변수들

프로그램이 끝날 때까지 없어지지 않는 데이터.
전역 변수, 정적 변수, main() 관련 데이터(자바)

프로그램 수행 코드

일반적인 가상 메모리 구조

ADT Stack

ADT Stack
맨 위에 원소 x를 추가한다
맨 위의 원소를 알려주면서 삭제한다
맨 위의 원소를 알려준다

유일하게 접근 가능한 원소는
최근에 삽입한 원소

LIFO(Last-In-First-Out)란 별칭을 갖고 있다

2. Array Stack

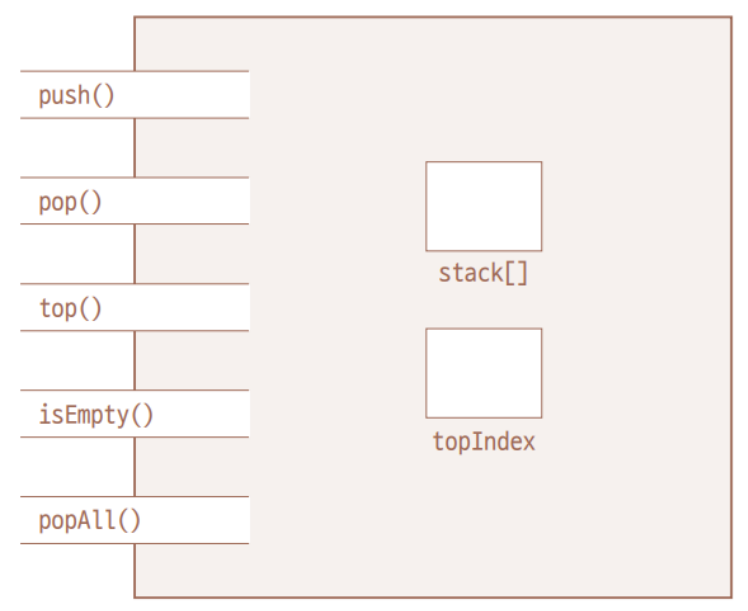
Array Stack 객체 구조

필드:

stack[]	◀ 스택 원소들이 저장되는 배열
topIndex	◀ 스택 탑 원소의 인덱스

작업:

push()	◀ 스택의 맨 위에 원소 를 삽입한다
pop()	◀ 스택의 맨 위에 있는 원소를 알려주고 삭제한다
top()	◀ 스택의 맨 위에 있는 원소를 알려준다
isEmpty()	◀ 스택이 빈 스택인지를 알려준다
popAll()	◀ 스택을 깨끗이 청소한다

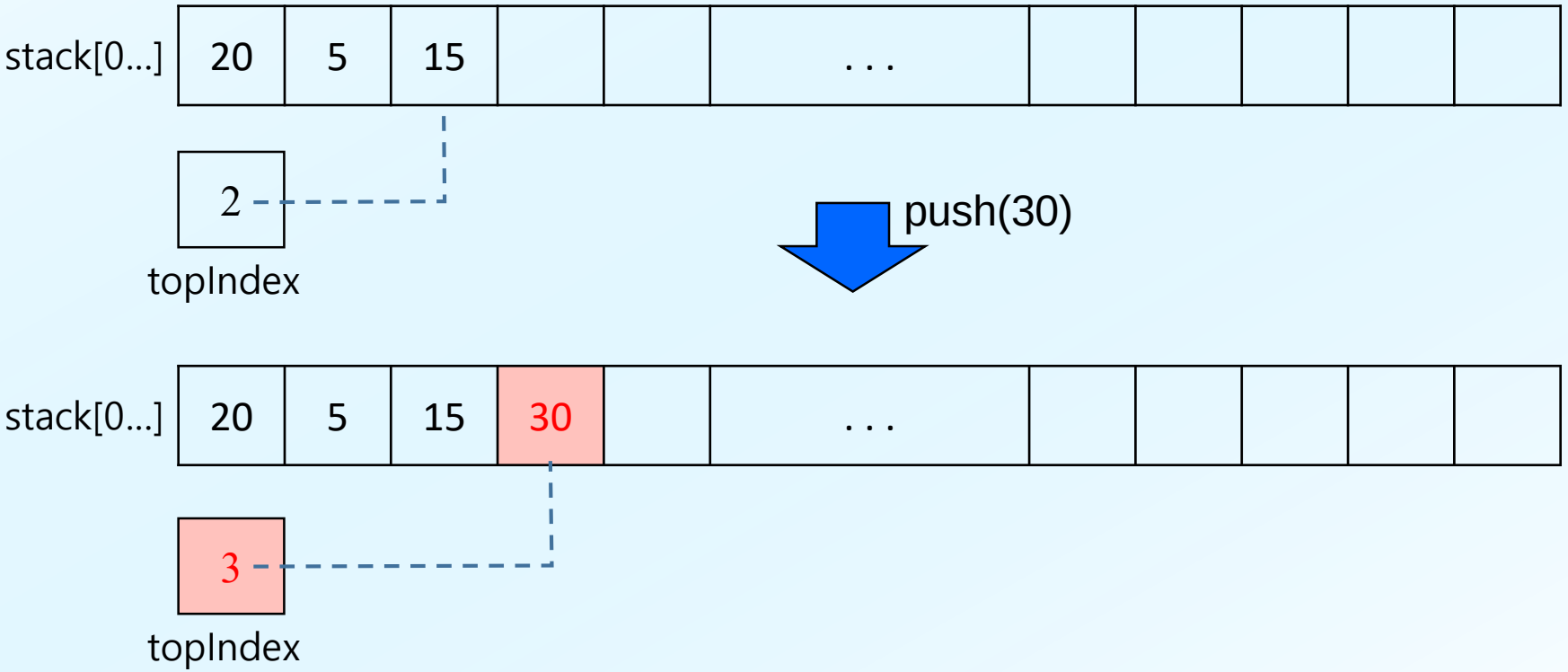


삽입 Insertion

핵심 작업

```
topIndex++
stack[topIndex] ← x
```

```
push(x):  ◀ Insert x
    if (topIndex >= stack.length-1)
        /*에러 처리*/   ◀ 꽉 찬 상태
    else
        topIndex++
        stack[topIndex] ← x
```

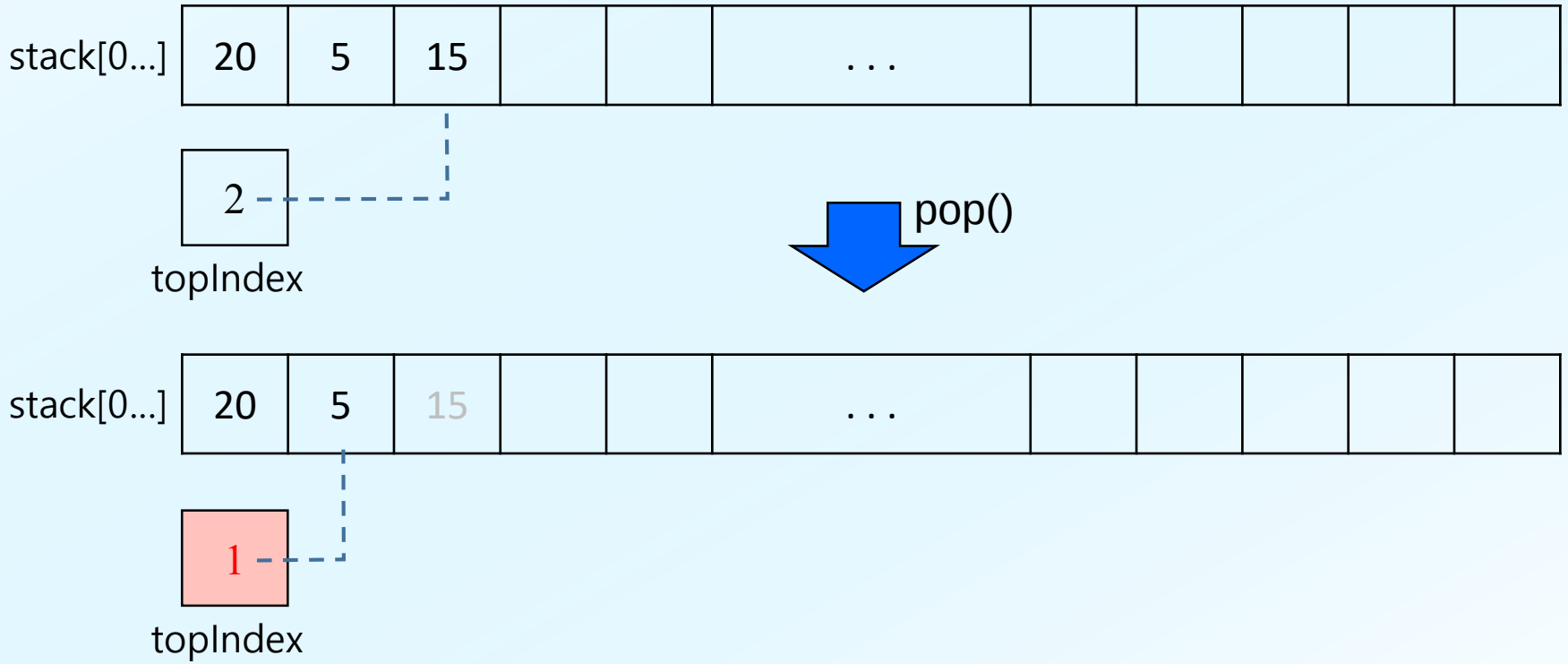


삭제 Deletion

핵심 작업

```
topItem ← stack[topIndex]
topIndex--
return topItem
```

```
pop(): ◀ Remove top item
    if (isEmpty())
        /*에러 처리*/
    else
        topItem ← stack[topIndex]
        topIndex--
        return topItem
```



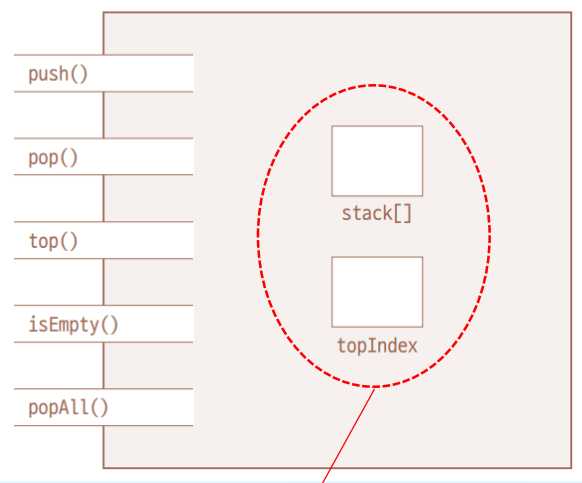
기타 작업

```
top():
        if (isEmpty())  /* 에러 처리 */
        else return stack[topIndex]
```

```
isEmpty():
        if (topIndex < 0)
                return true
        else
                return false
```

```
popAll():
        topIndex ← -1
```

Java 구현




```
public interface StackInterface<E> {
        public void push(E newItem);
        public E pop();
        public E top();
        public boolean isEmpty();
        public void popAll();
}
```

```
public class ArrayStack<E> implements StackInterface<E> {
        private E[] stack;
        private int topIndex;
        private static final int DEFAULT_CAPACITY = 64;
        public ArrayStack(int n) {  // 생성자(generator) 1
                stack = (E[]) new Object[n];
                topIndex = -1;
        }
        public ArrayStack() { // 생성자 2
                stack = (E[]) new Object[DEFAULT_CAPACITY];
                topIndex = -1;
        }
…
```

1/2 of class ArrayStack

```
        …
        public void push(E newItem) {
                if (isFull()) { /*에러 처리*/ }
                else stack[++topIndex] = newItem;
        }
        public E pop() {
                if (isEmpty()) { /*에러 처리*/ }
                else return stack[topIndex--];
        }
        public E top() {
                if (isEmpty()) { /*에러 처리*/ }
                else return stack[topIndex];
        }
        public boolean isEmpty() {
                return (topIndex < 0);
        }
        private boolean isFull() {
                return (topIndex == stack.length-1);
        }
        public void popAll() {
                stack = (E[]) new Object[stack.length];
                topIndex = -1;
        }
} // class ArrayStack<>
```

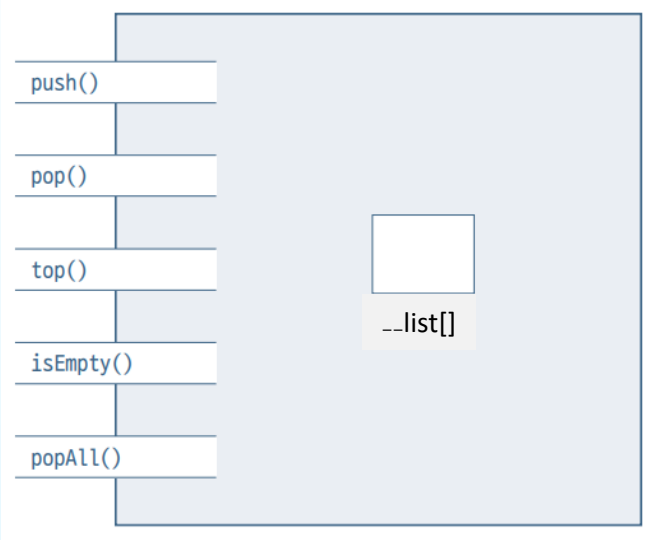
이 클래스의 객체를 생성해서 사용할 때는

```
ArrayStack<Integer> sample = new ArrayStack<>();
sample.push(300);
sample.push(100);
sample.pop();
…
```

원소가 다른 타입이면 <**E**> 부분만 변경

```
ArrayStack<String> sample = new ArrayStack<>();
sample.push(“sample string”);
sample.push(“100”);
sample.pop();
…
```

참고: 파이썬 리스트 스택 구현



리스트 스택 객체 구조

```
class ListStack:
        def __init__(self):
                self.__list = []

        def push(self, x):
                self.__ list.append(x)

        def pop(self):
                return self.__ list.pop()

        def top(self):
                if self.isEmpty():
                        return None
                else:
                        return self.__ list[-1]

        def isEmpty(self) -> bool:
                return not bool(self.__list)
                # 또는 return len(self.__list) == 0

        def popAll(self):
                self.__list.clear()
```

Java의 Array 운용 원리

<쉽게 배우는 자료구조, p.185>

앞의 Java 코드 popAll()에서

```
public void popAll() {
        stack = (E[]) new Object[stack.length];
        topIndex = -1;
}
```

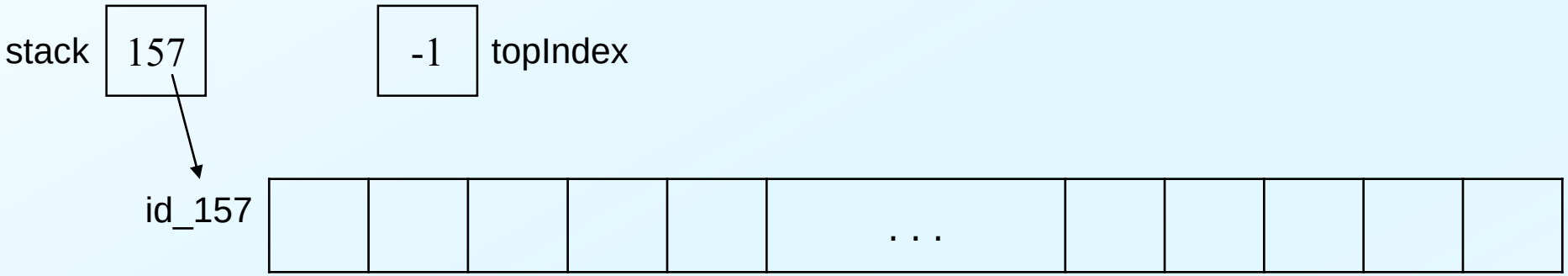
← 이게 왜 필요한가?

```
public void popAll() {
        topIndex = -1;
}
```

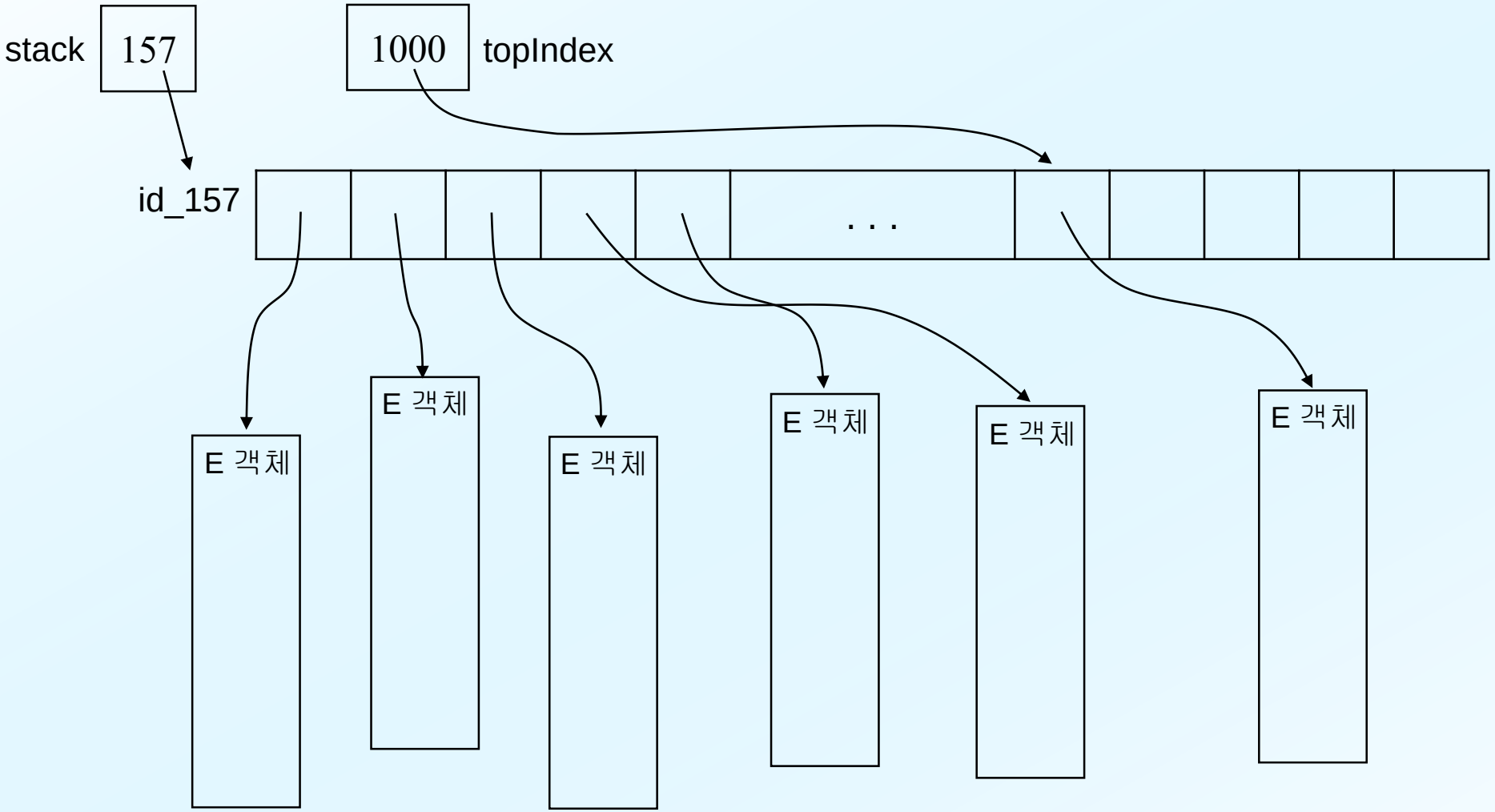
그냥 이걸로 충분하지 않은가?

맞게 작동한다. 그러나...

“stack = **new** arrayStack[*n*];” 직후

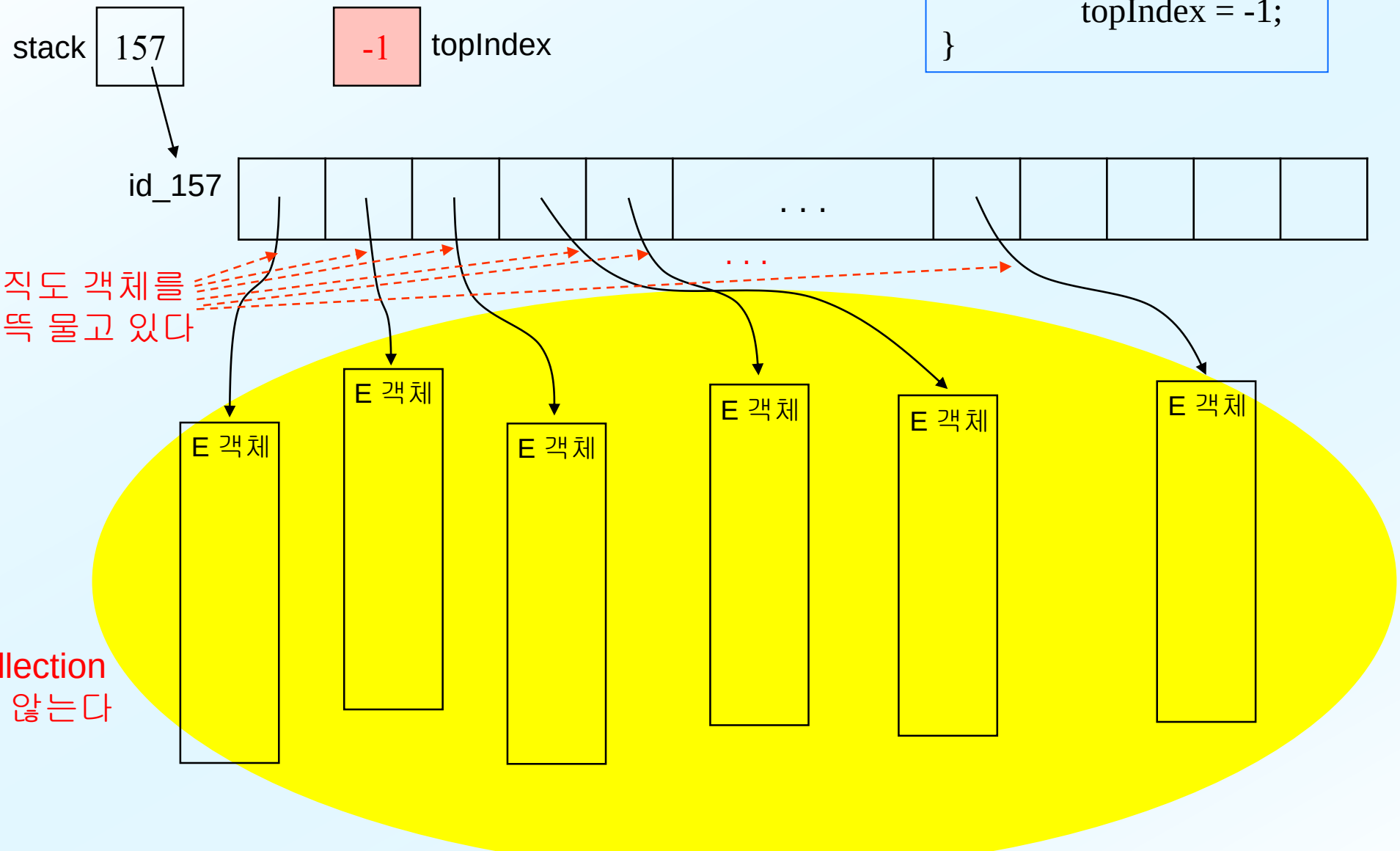


“stack = **new** …[*n*];” 이후 일련의 삽입/삭제 후의 예

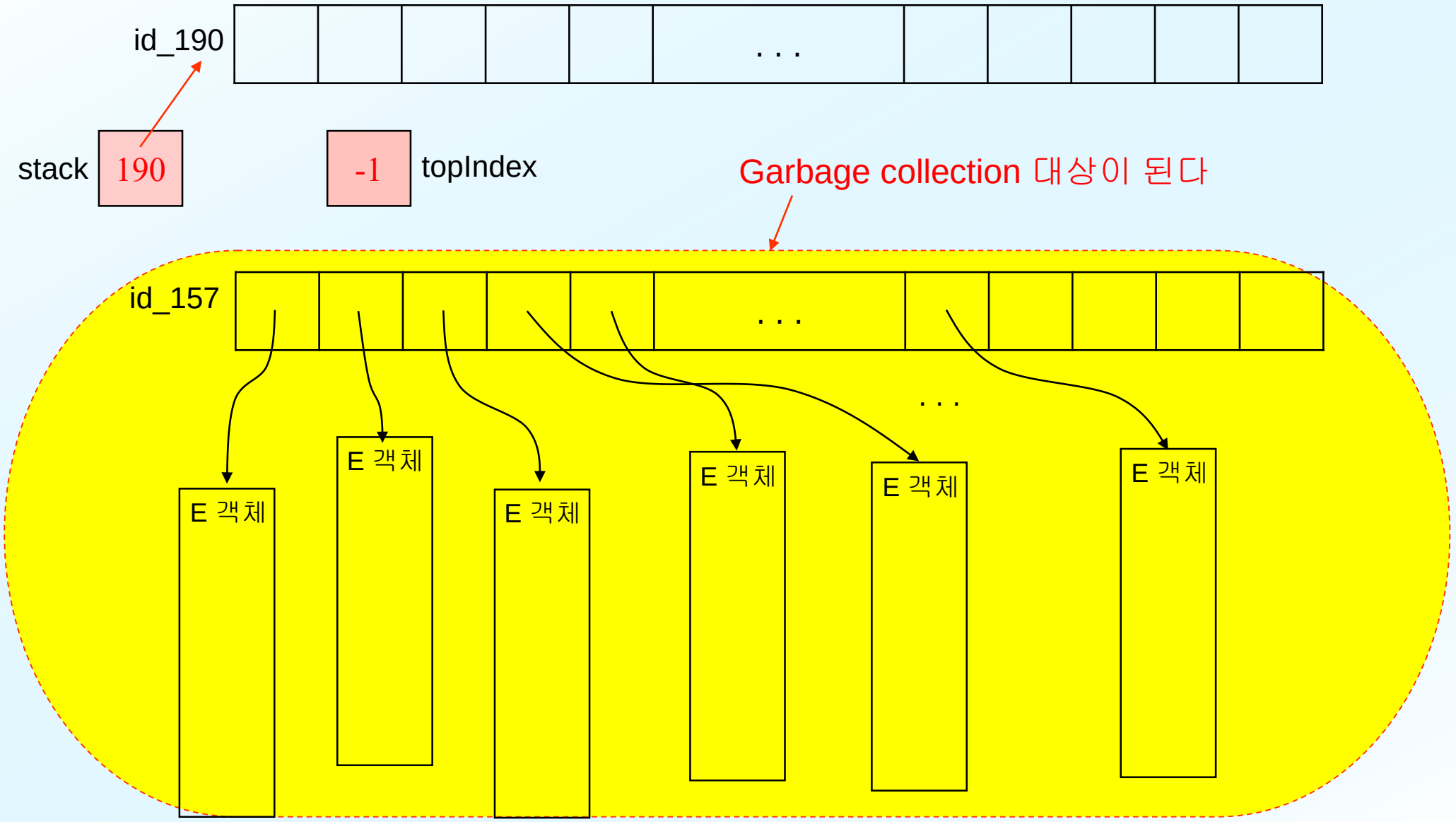


이렇게 해도 작동에는 문제 없지만..

```
public void popAll() {
        topIndex = -1;
}
```

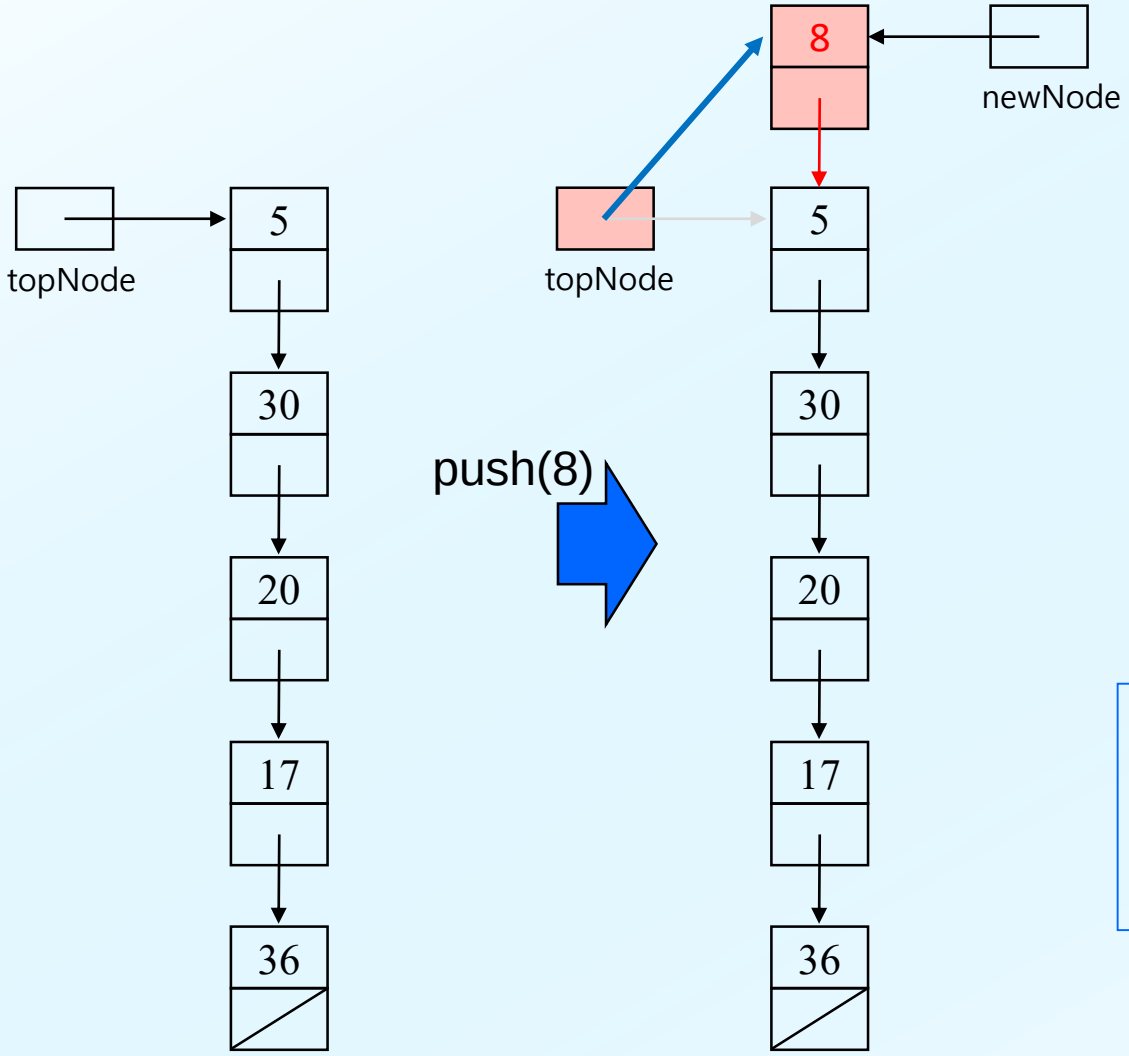


```
stack = (E[]) new Object[stack.length];
topIndex = -1;
```



3. Linked Stack

삽입 Insertion



push(x):

newNode.item ← x

newNode.next ← topNode

topNode ← newNode

```
public void push(E x) {
        topNode = new Node(x, topNode);
}
```

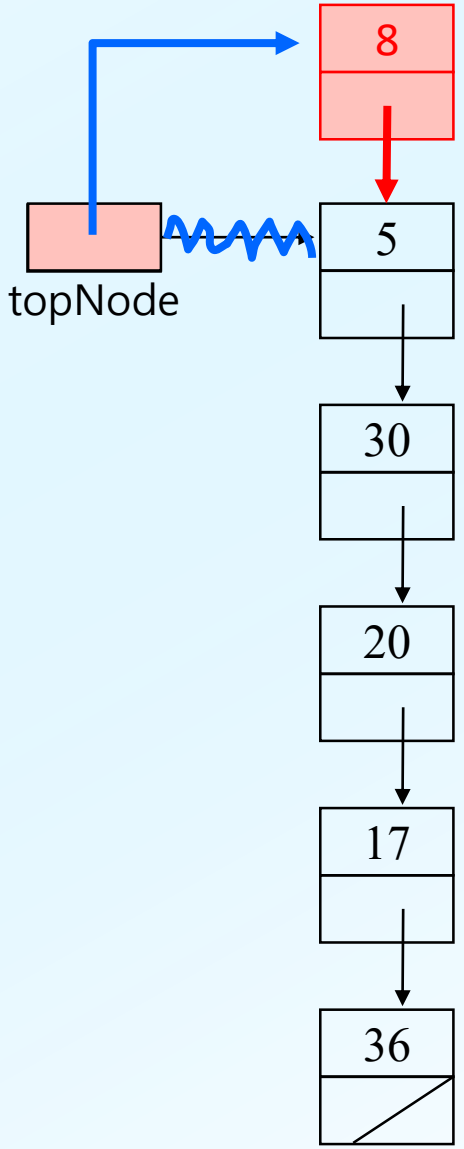
Java 코드

push(*x*):

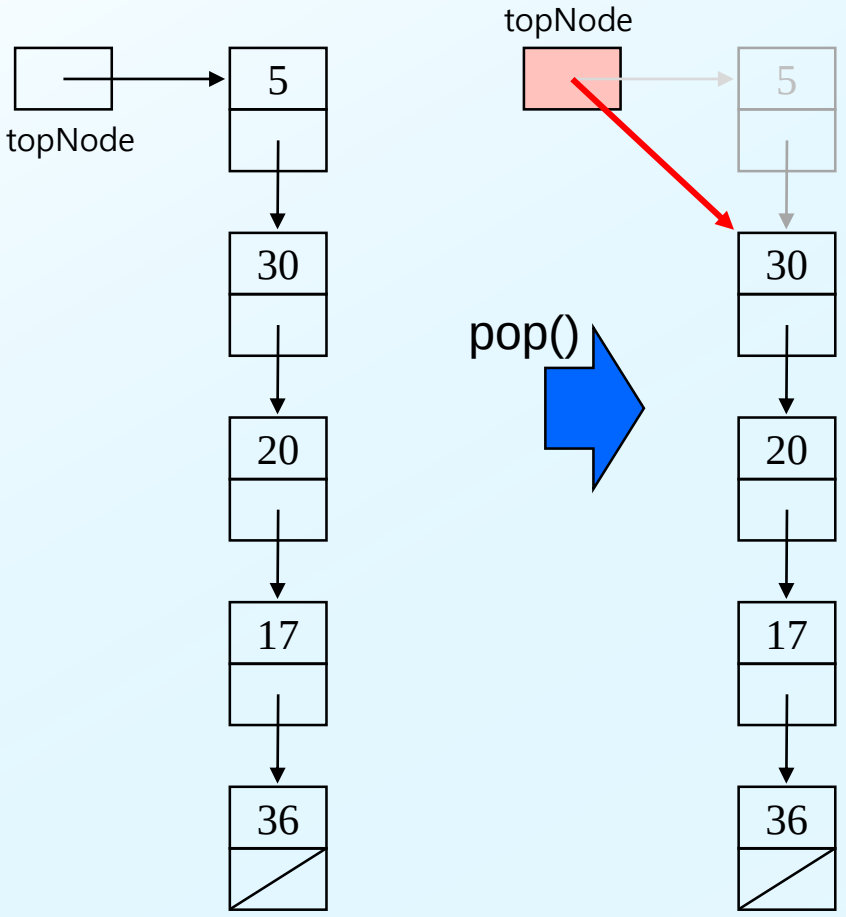
newNode.item ← *x*

newNode.next ← topNode

topNode ← newNode



삭제 Deletion



핵심 작업

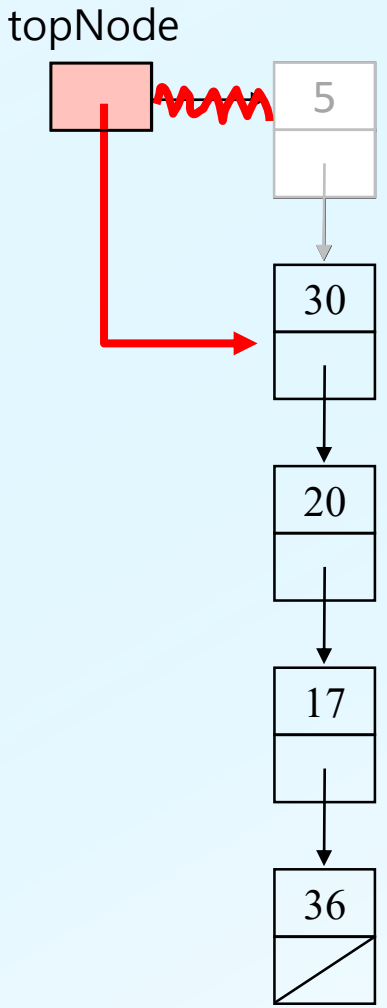
```
temp ← topNode
topNode ← topNode.next
return temp.item
```

```
pop():
        if (isEmpty())
                /*에러 처리*/
        else
                temp ← topNode
                topNode ← topNode.next
                return temp.item
```

```
public E pop( ) {
        if (isEmpty( ))
                /* 에러 처리 */
        else {
                Node temp = topNode;
                topNode = topNode.next;
                return temp.item;
        }
}
```

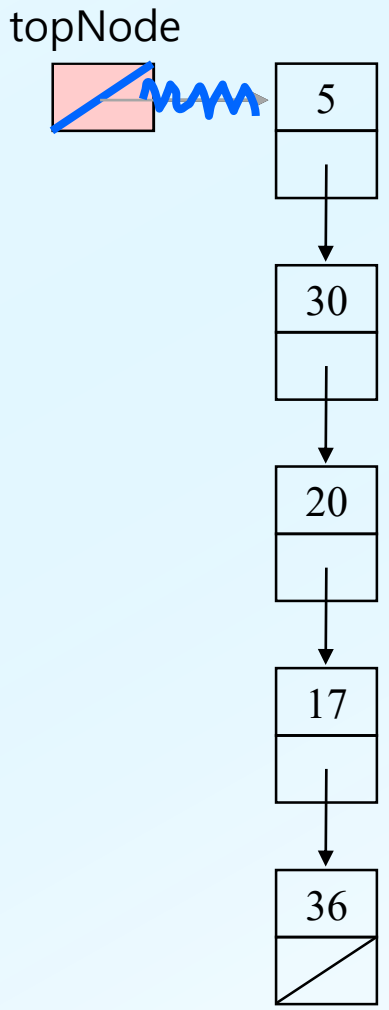
Java 코드

```
pop():
        if (isEmpty())
                /*에러 처리*/
        else
                temp ← topNode
                topNode ← topNode.next
                return temp.item
```



Cleaning

```
public void popAll() {
        topNode = null;
}
```



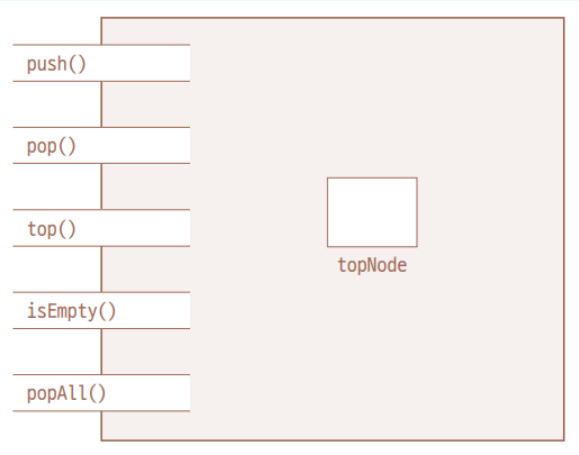
기타 작업

```
top():
        if (isEmpty())  /* 에러 처리 */
        else return topNode.item
```

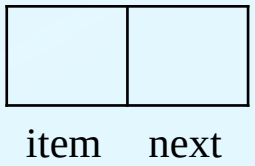
```
isEmpty():
        if (topNode = null)
                return true
        else
                return false
```

Java 구현

객체 구조




```
public interface StackInterface<E> {
        public void push(E newItem);
        public E pop();
        public E top();
        public boolean isEmpty();
        public void popAll();
}
```




```
public class Node<E> {
        public E item;
        public Node<E> next;
        public Node(E newItem) {
                item = newItem;
                next = null;
        }
        public Node(E newItem, Node<E> nextNode) {
                item = newItem;
                next = nextNode;
        }
}
```

클래스 LinkedStack

```
public class LinkedStack<E> implements StackInterface<E> {
        private Node<E> topNode;
        public LinkedStack() {  // 생성자
                topNode = null;
        }
        public void push(E newItem) {
                topNode = new Node<>(newItem, topNode);
        }
        public E pop() {
                if (isEmpty()) { /*에러 처리*/ }
                else {
                        Node<E> temp = topNode;
                        topNode = topNode.next;
                        return temp.item;
                }
        }
        public E top() {
                if (isEmpty()) { /*에러 처리*/ }
                else return topNode.item;
        }
        public boolean isEmpty() {
                return (topNode == null);
        }
        public void popAll() {
                topNode = null;
        }
} // class LinkedStack<>
```

이 클래스의 객체를 생성해서 사용할 때는

```
LinkedStack<Integer> sample = new LinkedStack<>();
sample.push(300);
sample.push(100);
sample.pop();
…
```

원소가 다른 타입이면 <**E**> 부분만 변경

```
LinkedStack<String> sample = new LinkedStack<>();
sample.push(“sample string”);
sample.push(“100”);
sample.pop();
…
```

4. Stack by Reusing

상속

```
public class InheritedStack<E> extends LinkedList<E>
                                implements StackInterface<E> {
        public InheritedStack() {
                super();
        }
        public void push(E newItem) {
                add(0, newItem);
        }
        public E pop() {
                if (!isEmpty())
                        return remove(0);
                else return null;
        }
        public E top() {
                return get(0);
        }
        public void popAll() {
                clear();
        }
} // End InheritedStack<>
```

Note: method isEmpty()는 정의되지 않았다

클래스 InheritedStack<>에서 isEmpty()를 호출하면
먼저 클래스 InheritedStack<>에서 isEmpty()가 있으면 그것을 수행하고
없으면 상위 클래스인 LinkedList<>의 isEmpty()를 수행한다.
add(), get(), remove(), clear()도 마찬가지.

또는.. super를 명시해도 된다

super: parent class

```
public class InheritedStack<E> extends LinkedList<E>
                                    implements StackInterface<E> {
        public InheritedStack() {
                super();
        }
        public void push(E newItem) {
                super.add(0, newItem);
        }
        public E pop() {
                return super.remove(0);
        }
        public E top() {
                return super.get(0);
        }
        public void popAll() {
                super.clear();
        }
} // End InheritedStack<>
```

이렇게 emptiness 체크를
안해도 결과는 앞 페이지와 동일.

앞 페이지는 호출 구조를
설명하기 위한 목적으로
isEmpty()를 포함.

비교

```
public class LinkedStack<E> implements StackInterface<E> {

        private Node<E> topNode;
        public LinkedStack(int n) {  // 생성자
                topNode = null;
        }
        public void push(E newItem) {
                topNode = new Node<>(newItem, topNode);
        }
        public E pop() {
                if (isEmpty()) { /*에러 처리*/ }
                else {
                        Node<E> temp = topNode;
                        topNode = topNode.next;
                        return temp.item;
                }
        }
        public E top() {
                if (isEmpty()) { /*에러 처리*/ }
                else return topNode.item;
        }
        public boolean isEmpty() {
                return (topNode == null);
        }
        public void popAll() {
                topNode = null;
        }
} // class LinkedStack<>
```

```
public class InheritedStack<E> extends LinkedList<E>
                                implements StackInterface<E> {

        public InheritedStack() {
                super();
        }
        public void push(E newItem) {
                add(0, newItem);
        }
        public E pop() {
                return remove(0);
        }


        public E top() {
                return get(0);
        }

        // isEmpty()는 정의하지 않았으므로 상위 클래스의 isEmpty()가 사용됨

        public void popAll() {
                clear();
        }
} // End InheritedStack<>
```

재사용

```
public class ListStack<E> implements StackInterface<E> {
        private ListInterface<E> list;
        public ListStack() {
                list = new LinkedList<>();
        }
        public void push(E newItem) {
                list.add(0, newItem);;
        }
        public E pop() {
                return list.remove(0);
        }
        public E top() {
                return list.get(0);
        }
        public boolean isEmpty( ) {
                return list.isEmpty();
        }
        public void popAll( ) {
                list.clear();
        }
} // class ListStack<>
```

비교

```
public class LinkedStack<E> implements StackInterface<E> {
        private Node<E> topNode;
        public LinkedStack(int n) {  // 생성자
                topNode = null;
        }
        public void push(E newItem) {
                topNode = new Node<>(newItem, topNode);
        }
        public E pop() {
                if (isEmpty()) { /*에러 처리*/ }
                else {
                        Node<E> temp = topNode;
                        topNode = topNode.next;
                        return temp.item;
                }
        }
        public E top() {
                if (isEmpty()) { /*에러 처리*/ }
                else return topNode.item;
        }
        public boolean isEmpty() {
                return (topNode == null);
        }
        public void popAll() {
                topNode = null;
        }
} // class LinkedStack<>
```

```
public class ListStack<E> implements StackInterface<E> {
        private ListInterface<E> list;
        public ListStack() {
                list = new LinkedList<>();
        }
        public void push(E newItem) {
                list.add(0, newItem);;
        }
        public E pop() {
                return list.remove(0);
        }
        public E top() {
                return list.get(0);
        }
        public boolean isEmpty( ) {
                return list.isEmpty();
        }
        public void popAll( ) {
                list.clear();
        }
} // class ListStack<>
```

5. Stack 활용

Postfix 계산

Java 코드

```
public class PostfixEval {
	public static void main(String[] args) {
		String postfix = "700 3 47 + 6 * - 4 /"; // 테스트 샘플 입력(후위 표현식)
		System.out.println("Input string: " + postfix);
		int answer = evaluate(postfix);
		System.out.println("Answer: " + answer);
	}
	private static int evaluate(String p) {
		int A, B;
		ArrayStack<Integer> s = new ArrayStack<>();
		boolean digitPreviously = false;
		for (int i = 0; i < p.length(); i++) {
			char ch = p.charAt(i); // postfix의 i번 문자. 번호는 0번부터.
			if (Character.isDigit(ch)) { // ch가 숫자
				if (digitPreviously == true) {
					int tmp = s.pop();
					tmp = 10 * tmp + (ch - '0');
					s.push(tmp);
				} else s.push(ch - '0');
				digitPreviously = true;
			} else if (isOperator(ch)) { // ch가 연산자
				A = s.pop();
				B = s.pop();
				int val = operation(A, B, ch);
				s.push(val);
				digitPreviously = false;
			} else digitPreviously = false; // ch가 공백
		}
		return s.pop();
	}
	private static int operation(int a, int b, char ch) { // 연산하기
		int val = 0;
		switch (ch) {
			case '+': val = b + a; break;
			case '-': val = b - a; break;
			case '*': val = b * a; break;
			case '/': val = b / a; break;
		}
		return val;
	}
	private static boolean isOperator(char ch) { // 연산자인가?
		return ch == '+' || ch == '-' || ch == '*' || ch == '/';
	}
}
```

출력

```
Input string: 700 3 47 + 6 * - 4 /
Answer: 100
```

문자열 뒤집기

Java 코드

```
public class ReverseString {
        public static void main(String arg[]) {
                String input = "Test Seq 12345"; // 테스트 입력 문자열
                System.out.println("Input string: " + input);
                String answer = reverse(input);
                System.out.println("Reversed string: " + answer);
        }
        private static String reverse(String s) {
                LinkedStack<Character> st = new LinkedStack<>();
                for (int i = 0; i < s.length(); i++)
                        st.push(s.charAt(i)); // s의 i번 문자. 번호는 0번부터.
                String output = "";
                while ( !st.isEmpty() )
                        output = output + st.pop();
                return output;
        }
}
```

출력

```
Input string: Test Seq 12345
Answer: 54321 qeS tseT
```