

A stack of books is shown on the right side of the image. The books have various colored spines, including white, red, and blue. A black rectangular overlay is positioned in the center-right, containing the word 'Stack' in white, followed by a horizontal line and the Thai text 'กองซ้อน' (Stack) in white.

# Stack

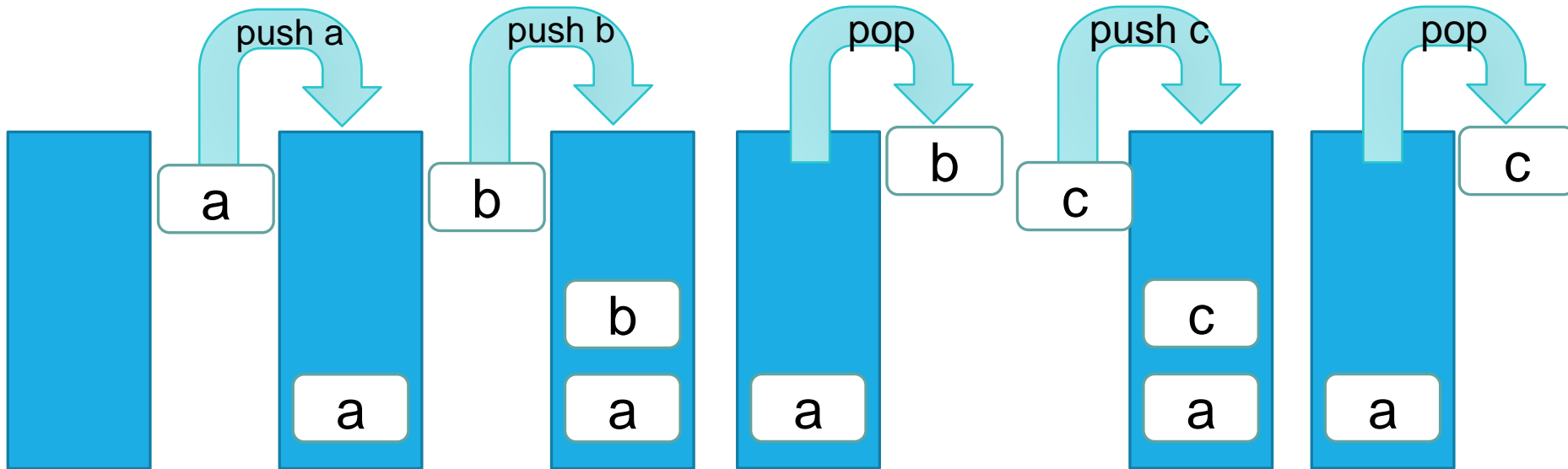
กองซ้อน

# Stack

คล้ายลิสต์

เพิ่ม / ลบ (Push / Pop) ข้อมูลที่ปลายด้านเดียว

Last-In-First-Out (LIFO) เอาของที่เก็บเข้าทีหลังออกมาก่อน



# ประโยชน์ของ Stack

# ประโยชน์ของ Stack

---

ใช้เมื่อต้องการเอาของที่เก็บเข้าที่หลังออกมาก่อน เช่น

- การจัดการ memory ที่ใช้เก็บตัวแปรในฟังก์ชัน เมื่อมีการเรียกฟังก์ชัน
- การตรวจสอบคู่ของวงเล็บที่สามารถซ้อนกันได้
- การแปลงนิพจน์แบบ infix เป็นแบบ postfix

# ประโยชน์ของ Stack : การเรียกฟังก์ชัน

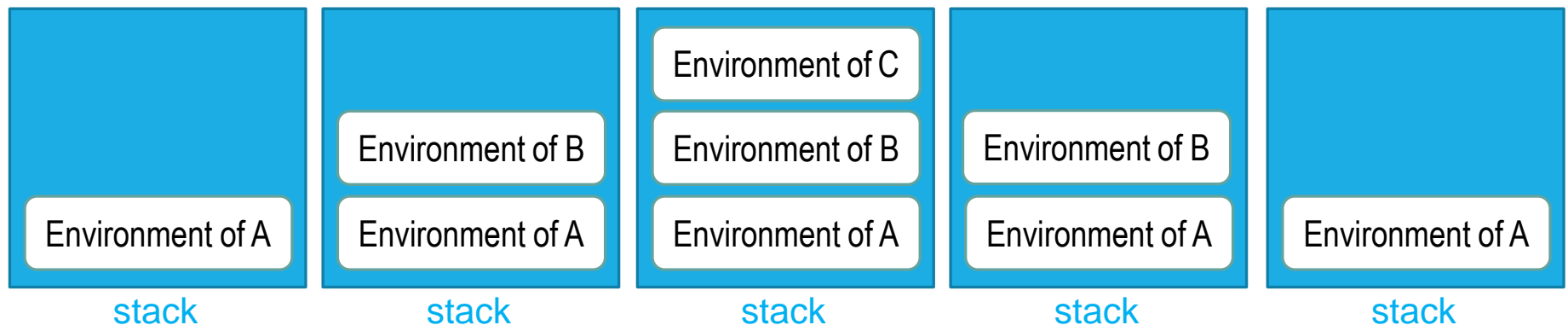
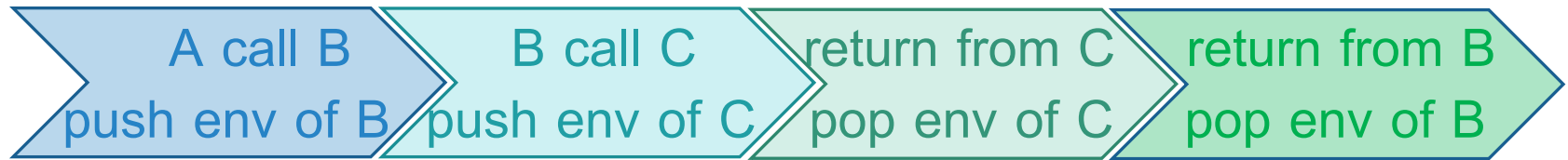
Function A เรียก Function B

Function B เรียก Function C

เมื่อ Function C ทำงานเสร็จจะกลับไปทำงาน Function B ต่อ

เมื่อ Function B ทำงานเสร็จจะกลับไปทำงาน Function A ต่อ

```
Function A
{
  ...
  call B
  ...
}
Function B
{
  ...
  call C
  ...
}
Function C
{
  ...
}
```



# ประโยชน์ของ Stack : การเรียกฟังก์ชัน

Function A เรียก Function B

Function B เรียก Function C

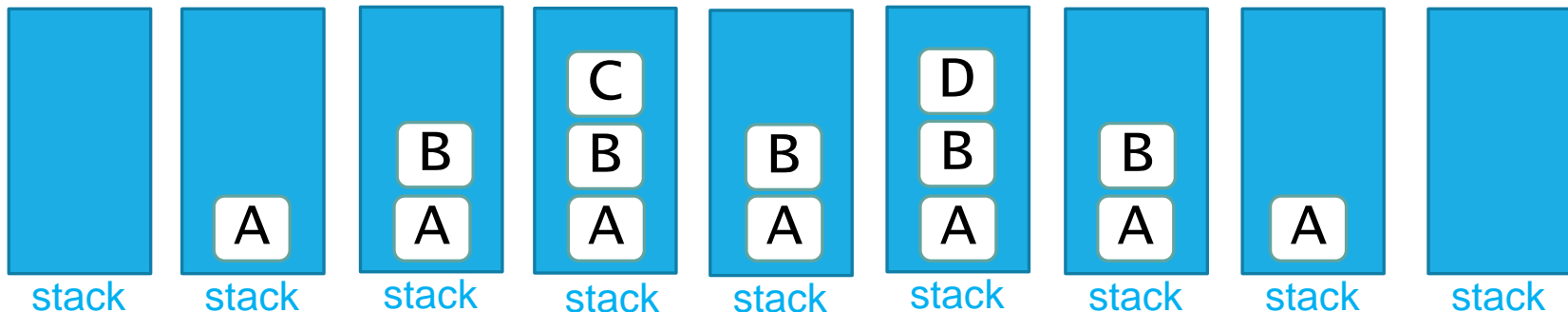
เมื่อ Function C ทำงานเสร็จจะกลับไปทำงาน Function B ต่อ

Function B เรียก Function D

เมื่อ Function D ทำงานเสร็จจะกลับไปทำงาน Function B ต่อ

เมื่อ Function B ทำงานเสร็จจะกลับไปทำงาน Function A ต่อ

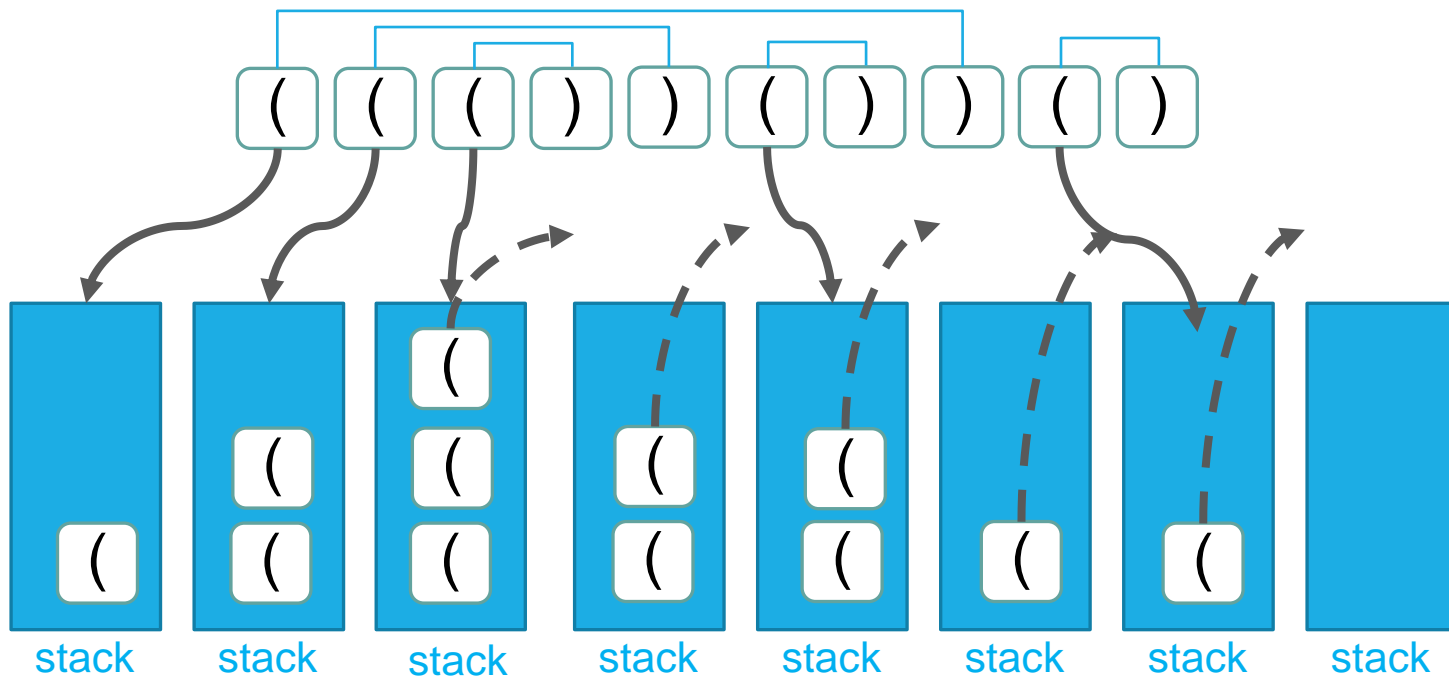
```
Function A
{
  ...
  call B
  ...
}
Function B
{
  ...
  call C
  call D
  ...
}
Function C
{
  ...
}
Function D
{
  ...
}
```



# ประโยชน์ของ Stack : การตรวจสอบวงเล็บ

อ่าน token ถัดไปจนกว่าจะหมด

- ถ้า token ที่อ่านมาเป็น ( ให้ push token ลงใน stack
- ถ้า token ที่อ่านมาเป็น ) ให้ pop )ออกจาก stack



# Interface Stack

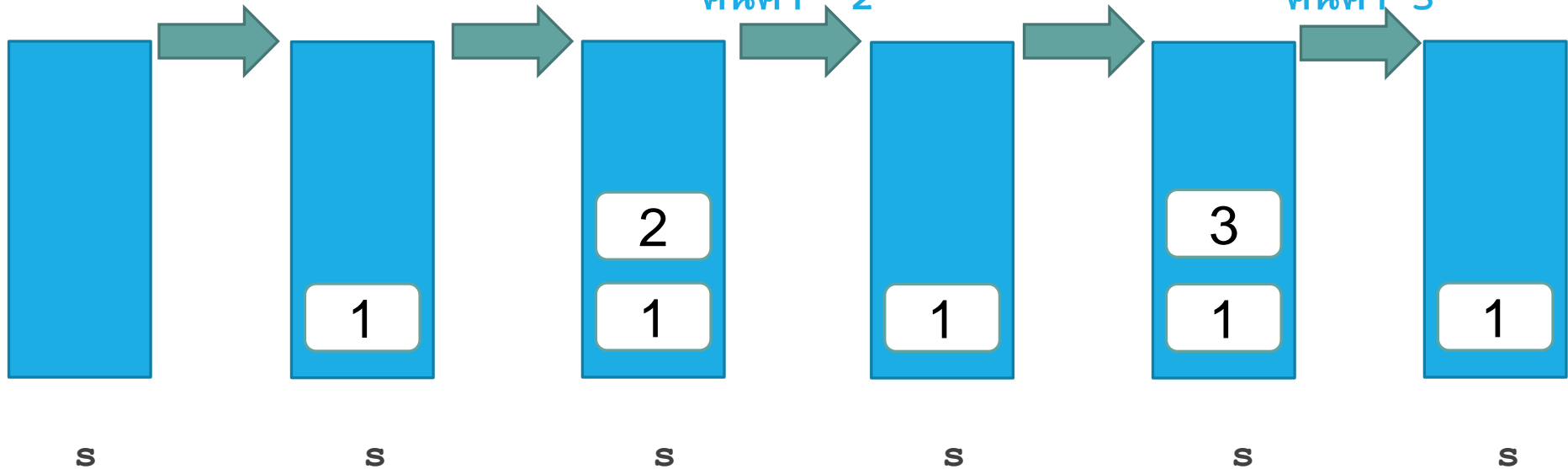
---

```
public interface Stack {  
    public boolean isEmpty();    // stack ว่างหรือไม่  
    public int size();           // มีข้อมูลกี่ตัวใน stack  
    public void push(Object e);  // ใส่ e ใน stack  
    public Object pop();         // เอาข้อมูลตัวบนสุดออกจาก stack  
    public Object peek();       // ดูข้อมูลตัวบนสุดใน stack แต่ไม่เอาออก  
}
```



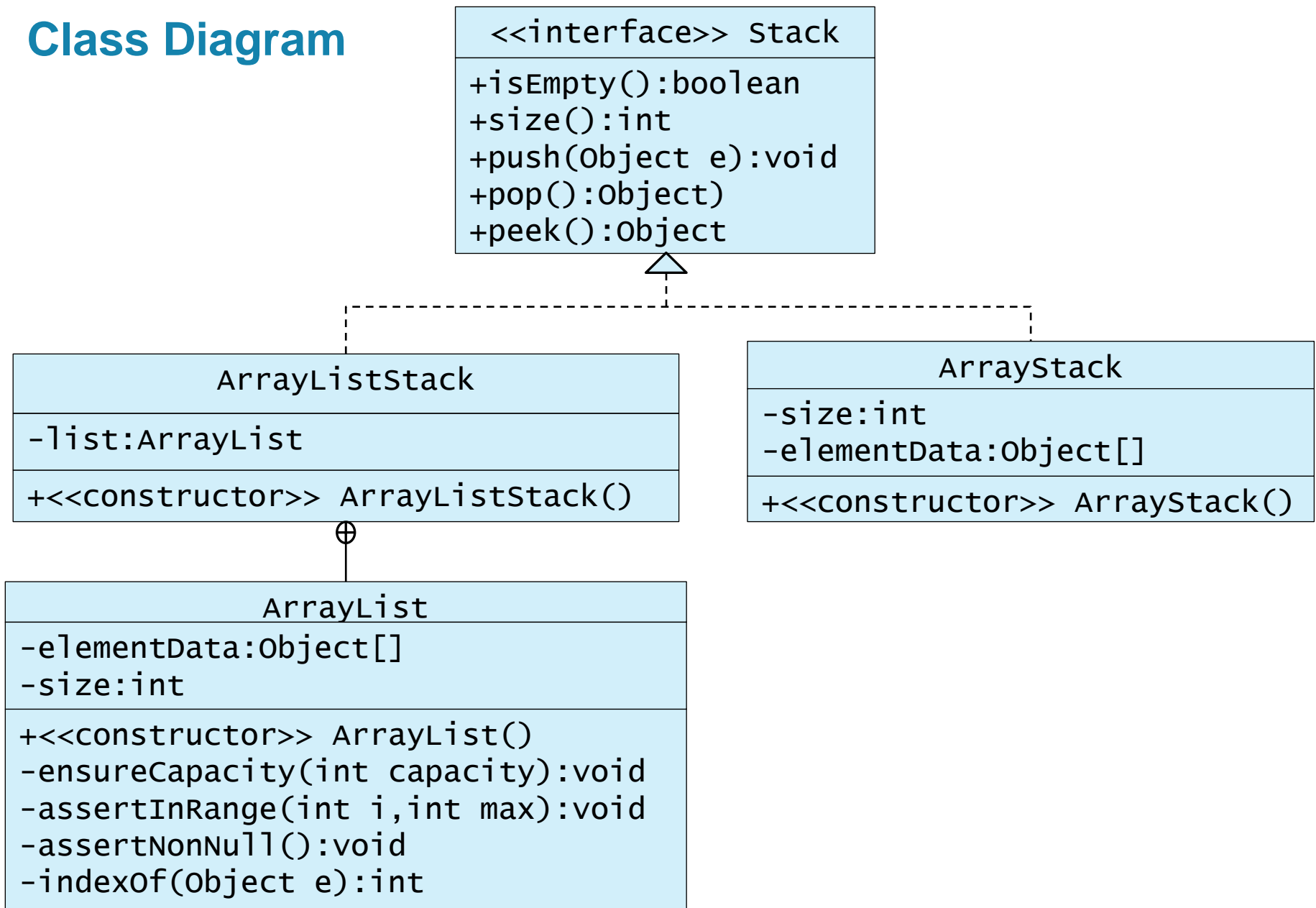
# Example

`s.push(1);`   `s.push(2);`   `s.pop();`   `s.push(3);`   `s.pop();`  
คีนค่า 2   คีนค่า 3



สร้างจาก ArrayList หรือ Array ใน Java

# Class Diagram



Class ArrayListStack

# Class ArrayListStack

---

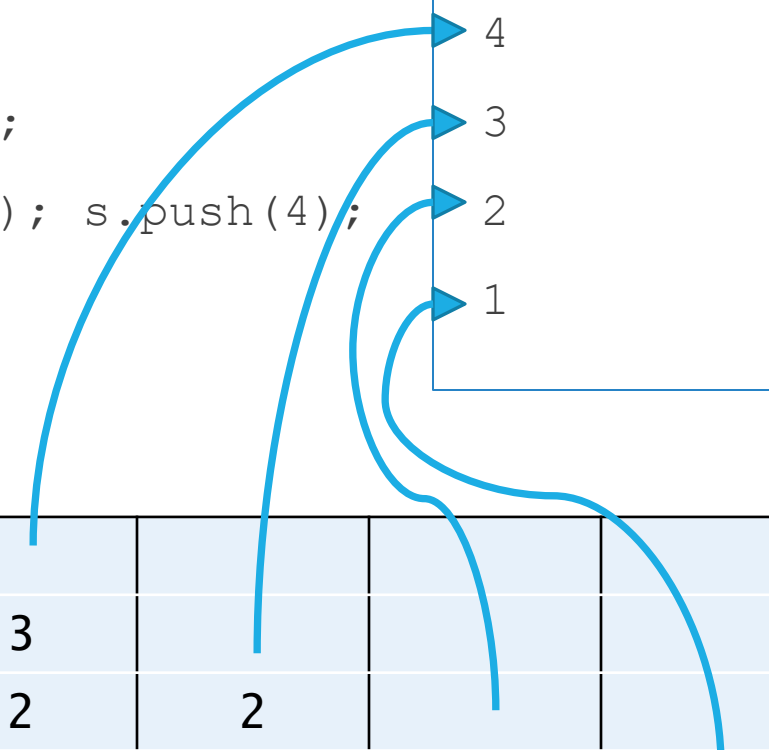
```
public class ArrayListStack implements Stack {
    private ArrayList list = new ArrayList();
    public boolean isEmpty()      { return list.isEmpty();      }
    public int size()              { return list.size();          }
    public void push(Object e)    { list.add(list.size(), e); }
    public Object peek() {
        if (isEmpty()) throw new IllegalStateException();
        return list.get(list.size()-1);
    }
    public Object pop() {
        Object e = peek();
        list.remove(list.size()-1);
        return e;
    }
}
```

# Using ArrayListStack

```
public class test {  
    public static void main() {  
        Stack s = new ArrayListStack();  
        s.push(1); s.push(2); s.push(3); s.push(4);  
        while (! s.isEmpty())  
            System.out.println(s.pop());  
    }  
}
```

Output:

4  
3  
2  
1



			4				
		3	3	3			
	2	2	2	2	2		
1	1	1	1	1	1	1	
s	s	s	s	s	s	s	s
push(1)	push(2)	push(3)	push(4)	pop()	pop()	pop()	pop()

# Using ArrayListStack

```
public class test {  
    public static void main() {  
        Stack s = new ArrayListStack();  
        s.push("A");  s.push("B");  
        System.out.println(s.pop());  
        s.push("C");  s.push("D");  
        System.out.println(s.pop());  
        System.out.println(s.pop());  
        System.out.println(s.pop());  
    }  
}
```

Output:

B  
D  
C  
A

				D			
	B		C	C	C		
A	A	A	A	A	A	A	
S	S	S	S	S	S	S	S
push("A")	push("B")	pop()	push("C")	push("D")	pop()	pop()	pop()

Class ArrayStack

# Class ArrayStack

---

```
public class ArrayStack implements Stack {
    private Object[] elementData = new Object[1];
    private int size;

    public boolean isEmpty()      { return size==0;      }
    public int size()              { return size;        }
    public void push(Object e) {
        if (size == elementData.length) {
            Object[] arr = new Object[2*elementData.length];
            for (int i=0; i<size; i++)
                arr[i] = elementData[i];
            elementData = arr;
        }
        elementData[size++] = e;
    }
    public Object peek() { ... }
    public Object pop()  { ... }
}
```



# Class ArrayStack

---

```
public class ArrayStack implements Stack {
    private Object[] elementData = new Object[1];
    private int size;

    public boolean isEmpty()      { ... }
    public int size()              { ... }
    public void push(Object e)    { ... }

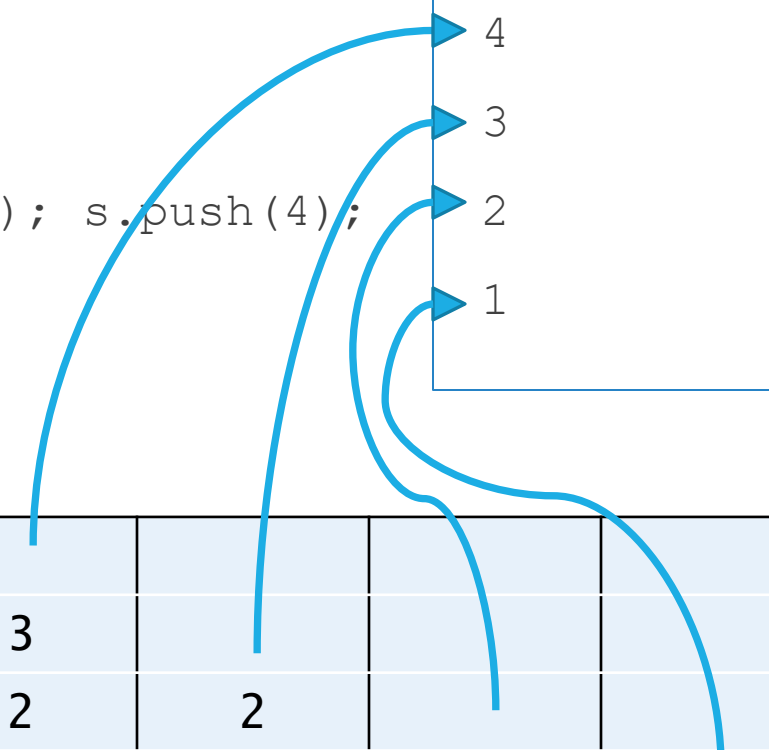
    public Object peek() {
        if (isEmpty()) throw new IllegalStateException();
        return elementData[size-1];
    }
    public Object pop() {
        Object e = peek();
        elementData[--size] = null;
        return e;
    }
}
```

# Using ArrayStack

```
public class test {  
    public static void main() {  
        Stack s = new ArrayStack();  
        s.push(1); s.push(2); s.push(3); s.push(4);  
        while (! s.isEmpty())  
            System.out.println(s.pop());  
    }  
}
```

Output:

4  
3  
2  
1



			4				
		3	3	3			
	2	2	2	2	2		
1	1	1	1	1	1	1	
s	s	s	s	s	s	s	s
push(1)	push(2)	push(3)	push(4)	pop()	pop()	pop()	pop()

# Using ArrayStack

```
public class test {  
    public static void main() {  
        Stack s = new ArrayStack();  
        s.push("A");  s.push("B");  
        System.out.println(s.pop());  
        s.push("C");  s.push("D");  
        System.out.println(s.pop());  
        System.out.println(s.pop());  
        System.out.println(s.pop());  
    }  
}
```

Output:

B  
D  
C  
A

				D			
	B		C	C	C		
A	A	A	A	A	A	A	
S	S	S	S	S	S	S	S
push("A")	push("B")	pop()	push("C")	push("D")	pop()	pop()	pop()

การนำ Stack ไปใช้

การใช้ STACK

การตรวจสอบคู่ของวงเล็บ

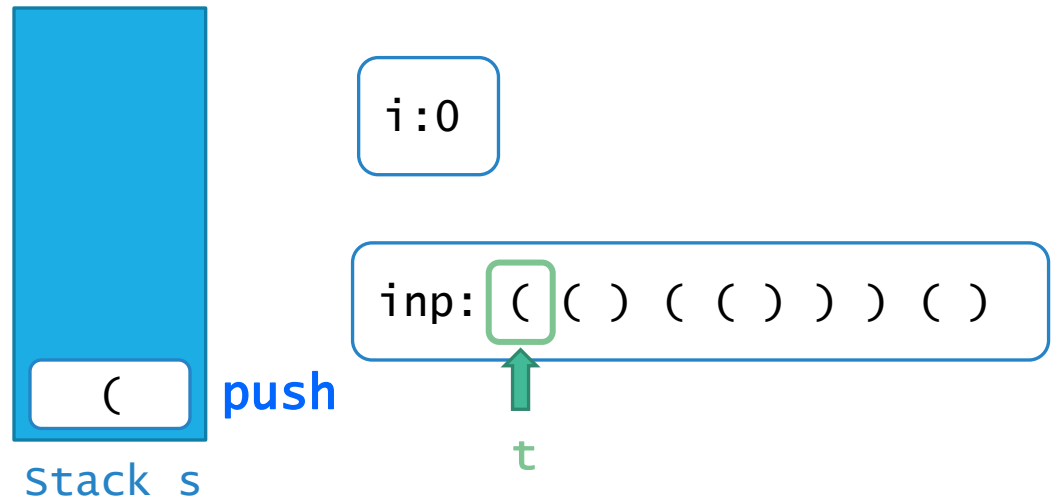
# Using Stack to Match parentheses

---

```
public static boolean checkParen(String inp) {  
    Stack s = new ArrayStack();    // เปลี่ยนเป็น ArrayListStack ได้  
    String open = "{[(", close = "}]";  
    for (int i=0; i<inp.length(); i++) {  
        String t = inp.substring(i,i+1);  
        if (open.contains(t))    // เป็นวงเล็บเปิดแบบหนึ่ง  
            s.push(t);  
        else if (close.contains(t)) { // เป็นวงเล็บปิดแบบหนึ่ง  
            int c = close.indexOf(t);    // ตรวจสอบว่าคู่วงเล็บปิดเป็นแบบเดียวกัน  
            if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))  
                return false;  
        }  
    }  
    return s.isEmpty(); // ถ้า input หมดแล้วจับคู่ได้ครบ คือ ถูกต้อง  
}
```

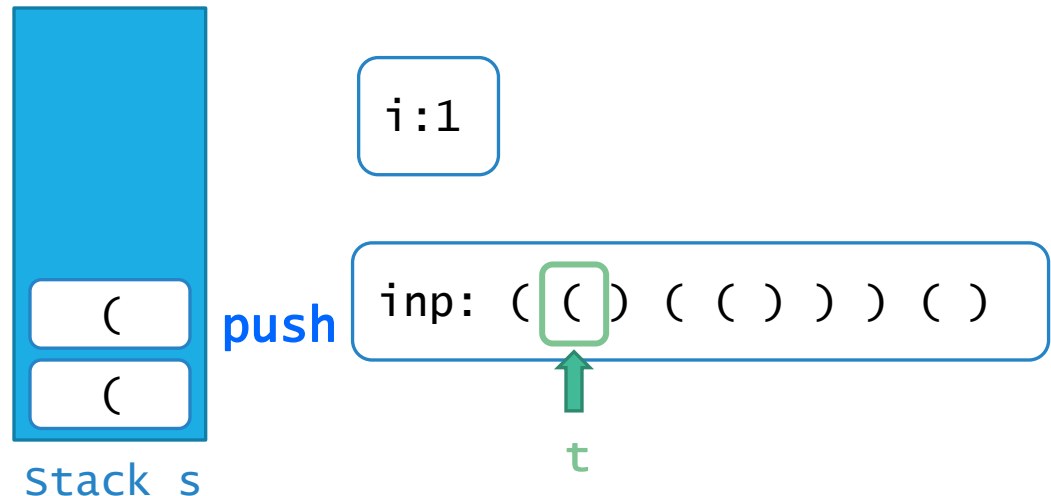
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



# Using Stack to Match parentheses : Execution

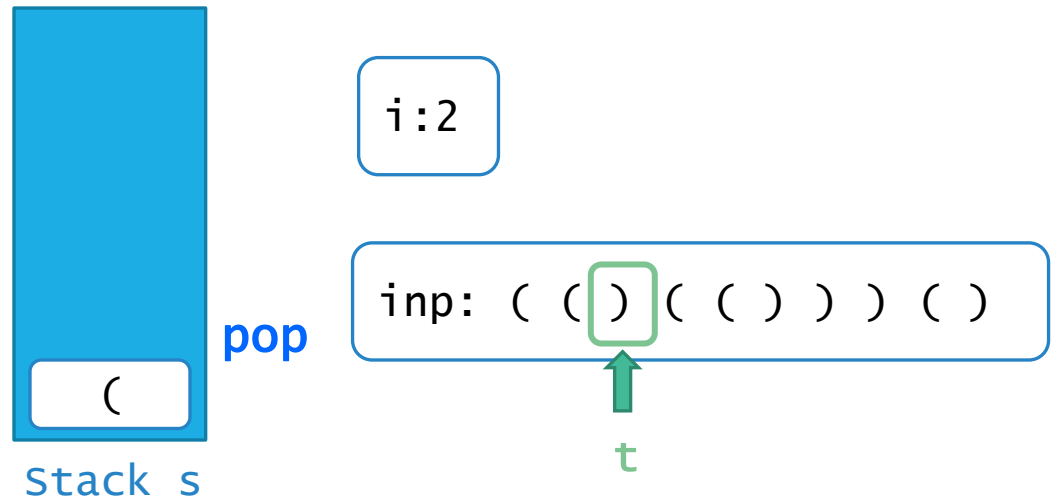
```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```





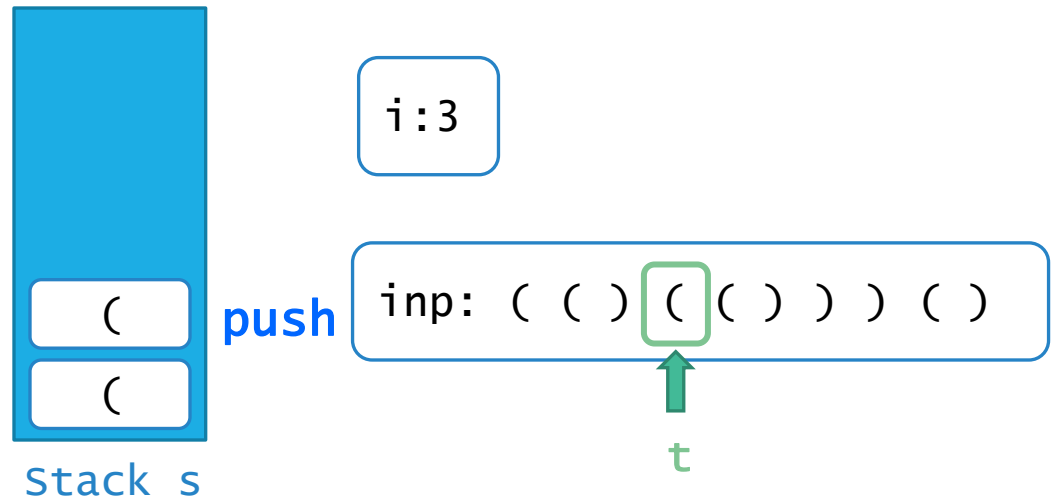
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



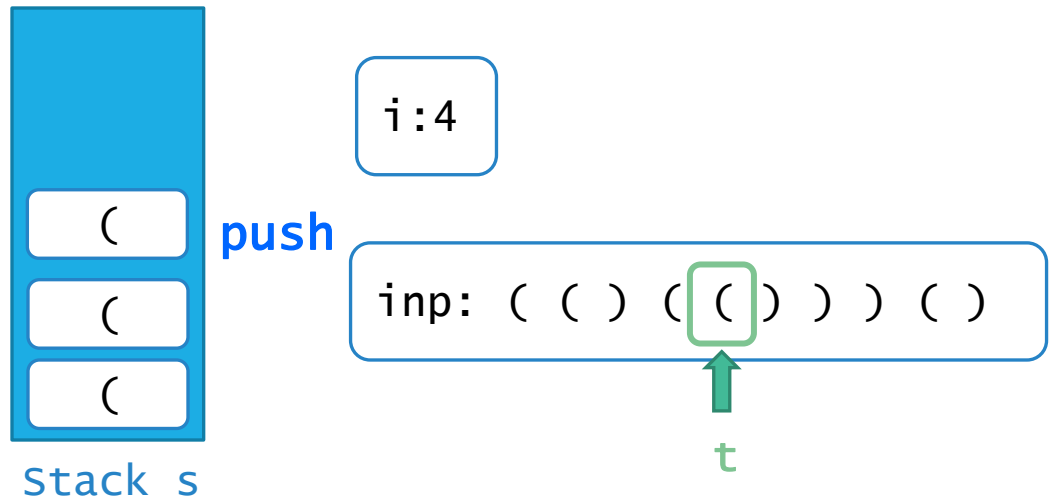
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



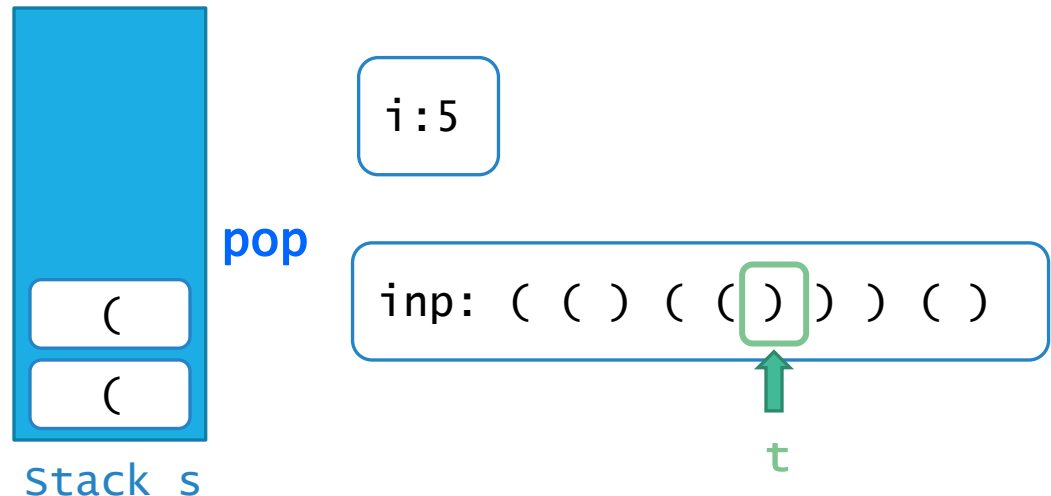
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



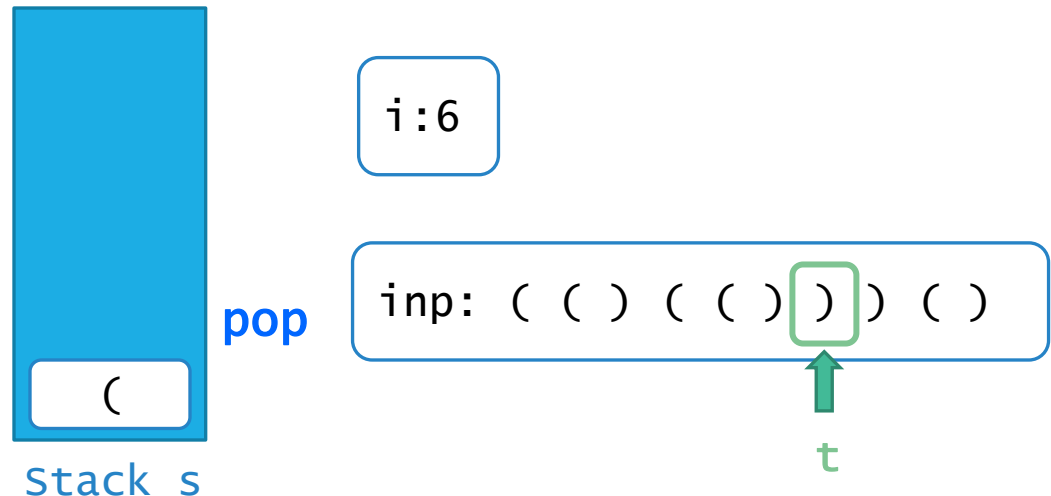
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



Stack s

pop

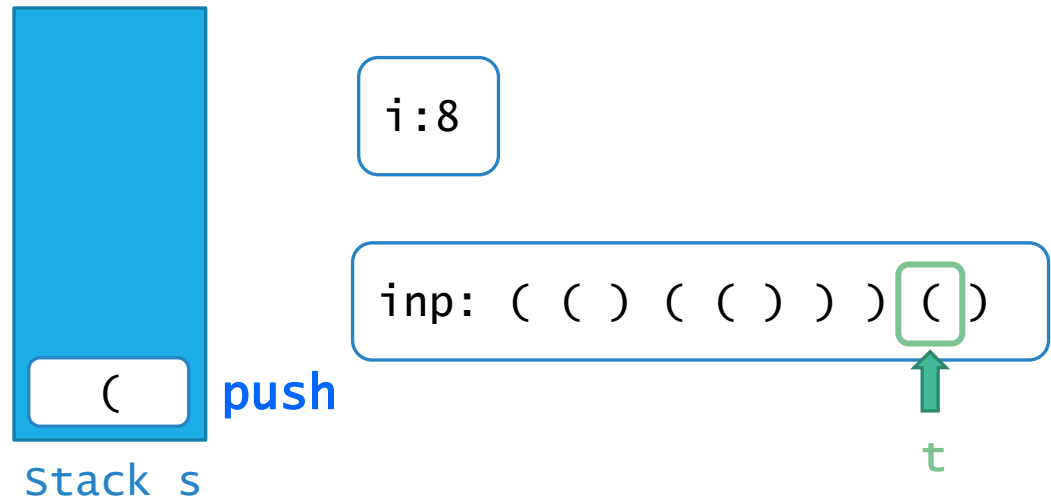
i:7

inp: ( ( ) ( ( ) ) ) ( )

t

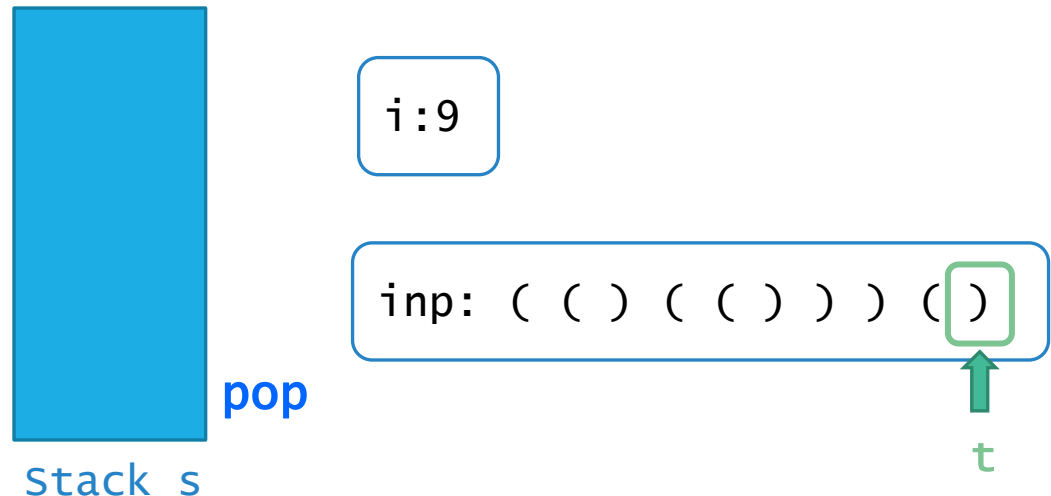
# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";
for (int i=0; i<inp.length(); i++) {
    String t = inp.substring(i,i+1);
    if (open.contains(t))
        s.push(t);
    else if (close.contains(t)) {
        int c = close.indexOf(t);
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))
            return false;
    }
}
return s.isEmpty();
```



# Using Stack to Match parentheses : Execution

```
String open = "{[(", close = "}]";  
for (int i=0; i<inp.length(); i++) {  
    String t = inp.substring(i,i+1);  
    if (open.contains(t))  
        s.push(t);  
    else if (close.contains(t)) {  
        int c = close.indexOf(t);  
        if (s.isEmpty() || !open.substring(c,c+1).equals(s.pop()))  
            return false;  
    }  
}  
return s.isEmpty();
```





# การแปลงนิพจน์แบบ infix เป็นแบบ postfix

---

การใช้ STACK

# Expressions

---

## Infix Expression:

operand **operator** operand

3 + 4 \* ( 5 - 6 ) + 7

## Postfix Expression:

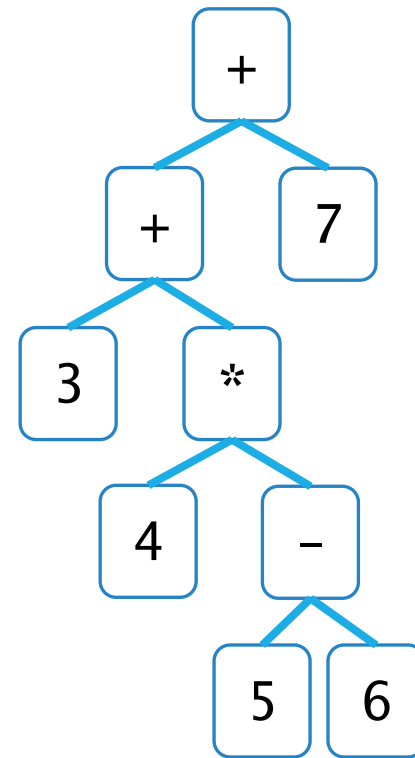
◦ operand operand **operator**

◦ 3 4 5 6 - \* + 7 +

## Prefix Expression:

◦ **operator** operand operand

◦ + + 3 \* 4 - 5 6 7



# Use stack to convert infix expression to postfix expression

```
public class Expression {  
    public static List infix2Postfix(List infix) { ... }  
    static String operators = "+-*/^()";  
    static int inPriority[] = {3,3,5,5,7,0};  
    static int outPriority[] = {3,3,5,5,7,9,1};  
    private static int inPriority(String x) {  
        return inPriority[operatorIndex(x)];  
    }  
    private static int outPriority(String x) {  
        return outPriority[operatorIndex(x)];  
    }  
    private static boolean isOperator(String x) {  
        return operatorIndex(x) >= 0;  
    }  
    private static int operatorIndex(String x) {  
        return operators.indexOf(x);  
    }  
}
```

operator	inPriority	outPriority
+ -	3	3
* /	5	5
^	7	7
(	0	9
)	-	1

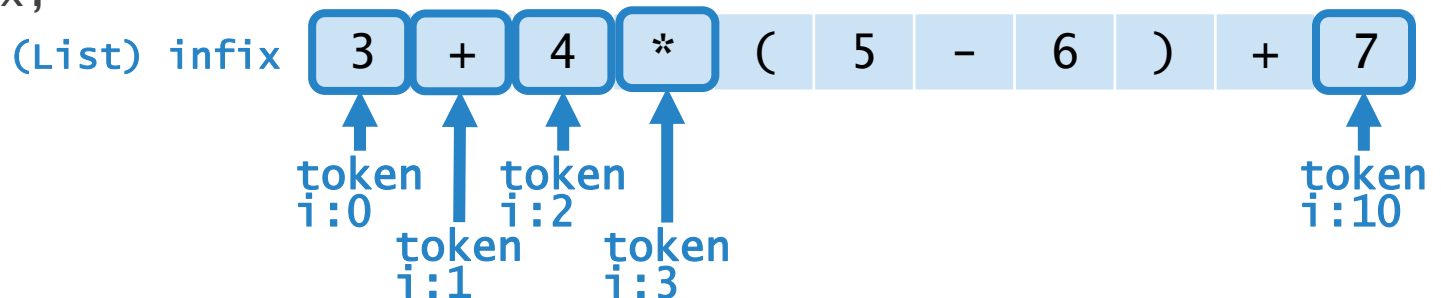
# Use stack to convert infix expression to postfix expression

```
public static List infix2Postfix(List infix) {
    ArrayList postfix = new ArrayList();
    Stack s = new ArrayStack();
    for (int i = 0; i<infix.size(); i++) {
        String token = (String) infix.get(i);
        if (!isOperator(token))    postfix.add(token); // output operand
        else { // for operator
            int p = outPriority(token); // compare to token's outPriority
            while (!s.isEmpty() && inPriority((String) s.peek())>=p)
                postfix.add(s.pop()); // pop op when inPriority >= outPriority
            if (token.equals("(")) s.pop();
            else                    s.push(token);
        }
    }
    while (! s.isEmpty()) postfix.add(s.pop());
    return postfix;
}
```

operator	inPriority	outPriority
+ -	3	3
* /	5	5
^	7	7
(	0	9
)	-	1

# Convert infix expression to postfix expression : Execution

```
for (int i = 0; i<infix.size(); i++) {  
    String token = (String) infix.get(i);  
    if (!isOperator(token))    postfix.add(token);  
    else {  
        int p = outPriority(token);  
        while (!s.isEmpty() && inPriority((String) s.peek())>=p)  
            postfix.add(s.pop());  
        if (token.equals("(")) s.pop();  
        else  
            s.push(token);  
    }  
}  
while (! s.isEmpty()) postfix.add(s.pop());  
return postfix;
```



# Convert infix expression to postfix expression : Execution

```
for (int i = 0; i<infix.size(); i++) {  
    String token = (String) infix.get(i);  
    if (!isOperator(token))    postfix.add(token); // push operand  
    else {                      // for operator  
        int p = outPriority(token); // compare to token's outPriority  
        while (!s.isEmpty() && inPriority((String) s.peek())>=p)  
            postfix.add(s.pop());  
        if (token.equals("("))  
            s.pop();  
        else  
            s.push(token);  
    }  
}  
while (! s.isEmpty()) postfix.add(s.pop());  
return postfix;
```

i	token	postfix	s (=>)
0	3	3	
1	+	3	+
2	4	3 4	+
3	*	3 4	+ *
4	(	3 4	+ * (
5	5	3 4 5	+ * (
6	-	3 4 5	+ * ( -
7	6	3 4 5 6	+ * ( -
8	)	3 4 5 6 -	+ *
9	+	3 4 5 6 - * +	+
10	7	3 4 5 6 - * + 7	+

# Convert infix expression to postfix expression : Execution

```
for (int i = 0; i<infix.size(); i++) {
    String token = (String) infix.get(i);
    if (!isOperator(token))    postfix.add(token); // push operand
    else {                    // for operator
        int p = outPriority(token); // compare to token's outPriority
        while (!s.isEmpty() && inPriority((String) s.peek())>=p)
            postfix.add(s.pop());
        if (token.equals("("))
            s.pop();
        else
            s.push(token);
    }
}
while (! s.isEmpty()) postfix.add(s.pop());
return postfix;
```

i	token	postfix	s (=>)
10	7	3 4 5 6 - * + 7	+
while loop		3 4 5 6 - * + 7 +	