กองซ้อน

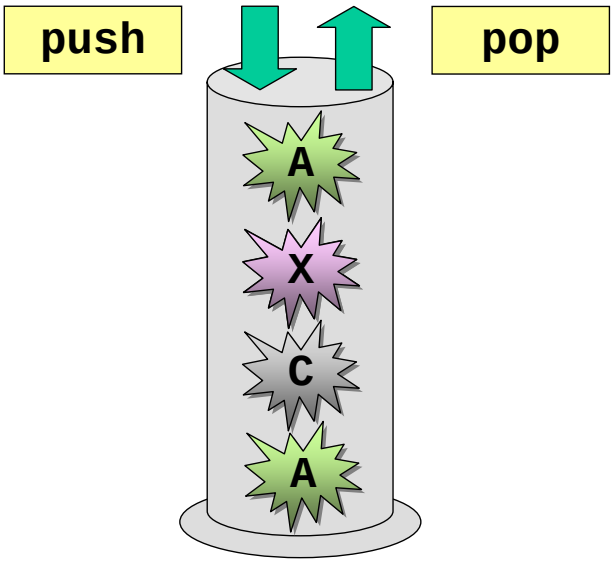
(Stack)

หัวข้อ

- นิยามกองซ้อน และอินเตอร์เฟส Stack
- การสร้างกองซ้อนด้วยอาเรย์
- ตัวอย่างการใช้งานกองซ้อน
  - การตรวจการเขียนวงเล็บเปิดปิด
  - การใช้กองซ้อนใน java virtual machine
  - การคำนวณค่าของนิพจน์เติมหลัง (postfix)
  - การเปลี่ยนนิพจน์เติมกลาง (infix) เป็นเติมหลัง

การเพิ่ม/ลบข้อมูลในกองซ้อน

- ข้อมูล เข้าหลัง ออกก่อน (Last-In First-Out)



กองซ้อน : stack

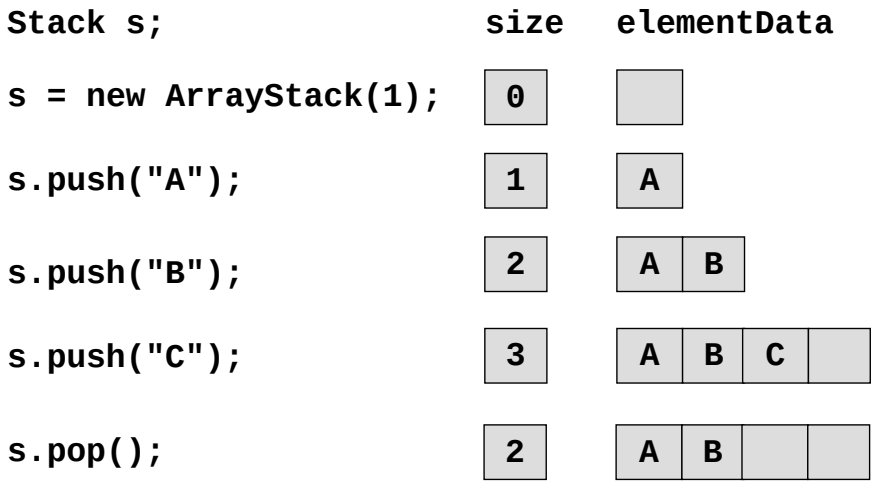
```
public interface Stack {
  public boolean isEmpty();
  public int size();
  public void push(Object e);
  public Object peek();
  public Object pop();
}
```

A B C * * D E * * *

C B E D A

ArrayStack : สร้าง Stack ด้วยอาเรย์

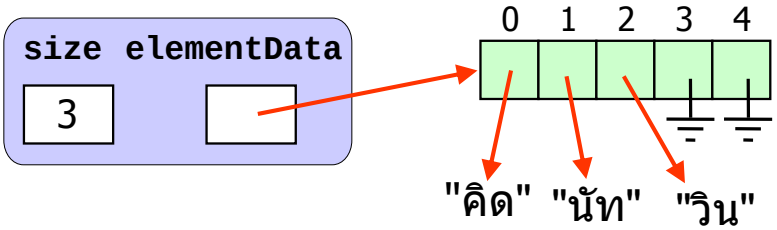
- คล้าย ArrayCollection
- เพิ่มกับลบเฉพาะที่ด้านท้าย
- ขยายขนาดของอาเรย์ได้ เมื่อเต็ม



ArrayStack : โครงสร้าง

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

  public ArrayStack(int cap) {
    elementData = new Object[cap];
  }
  public boolean isEmpty() {
    return size == 0;
  }
  public int size() {
    return size;
  }
```



ArrayStack : push

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

ข้อมูลเต็มอาเรย์ ก็ขยายขนาดอาเรย์เป็นสองเท่า

ถ้าไม่ต้องขยายขนาด : $\Theta(1)$

ถ้าต้องขยายขนาด : $O(n)$

ArrayStack : peek, pop

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

$\Theta(1)$

ตัวอย่างการใช้งาน Stack

- การตรวจสอบโครงสร้างแบบซ้อนกัน
  เช่น การใส่วงเล็บ ( ) { } [ ] ...
- การจัดเก็บตัวแปรและการเรียกเมท็อดของ jvm
- การประมวลผลนิพจน์ทางคณิตศาสตร์
- การทำ undo/redo
- การค้นคำตอบแบบ depth-first search
- ...

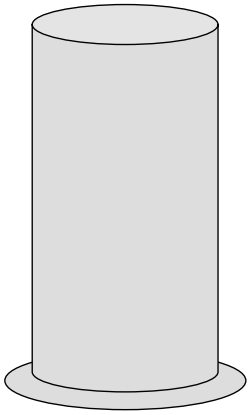
การตรวจสอบการใส่วงเล็บ

- ถูก : **({ ( ) [{ }] })**
- ผิด : เปิดปิดไม่ตรงกัน (**{]**)
- ผิด : มีปิดมากไป ({ ( ) }) **) }**
- ผิด : มีเปิดมากไป **(** { ( ) }
- วิธีทำ
  - อ่านมาทีละตัว
  - ถ้าเป็นวงเล็บเปิด ให้ push ลง stack
  - ถ้าเป็นวงเล็บปิด ให้ pop จาก stack มาตรวจสอบว่าเป็นวงเล็บเปิดที่ตรงกันวงเล็บปิดที่พบหรือไม่
  - เมื่อใดอยาก pop ถ้า isEmpty แสดงว่า ปิดมีมากไป
  - เมื่ออ่านเสร็จหมด stack ยังมีข้อมูล แสดงว่า เปิดมีมากไป

ตัวอย่าง ๑

- ({ ( ) [{ }] })

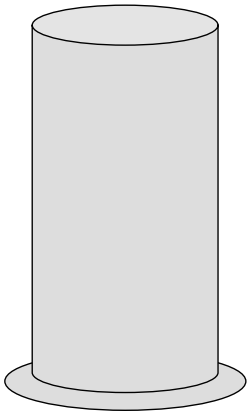
ถูกต้อง



ตัวอย่าง ๒

- ({ ( ) [{)] })

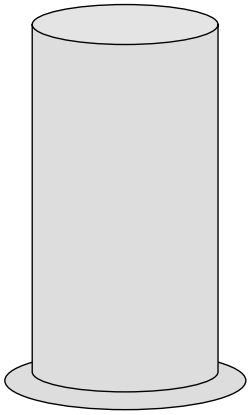
วงเล็บปิด
ไม่ตรงกับเปิด
ผิด !!



ตัวอย่าง ๓

- ({ ( ) })]

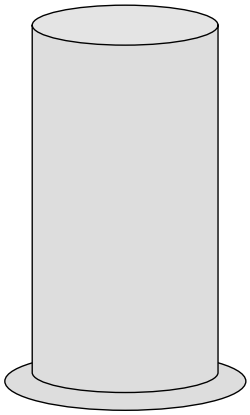
ยังไม่หมด
แต่ stack ว่าง
ผิด !!



ตัวอย่าง ๔

- ({ ( ) [{ }

หมดแล้ว
แต่ stack ไม่ว่าง
ผิด !!



โปรแกรมตรวจสอบการใส่วงเล็บ

```
public static boolean checkParentheses(String t) {
  String open = "{([", close = "})]";
  Stack s = new ArrayStack(100);
  for (int i = 0; i < t.length(); i++) {
    String token = t.substring(i, i + 1);
    if (open.indexOf(token) >= 0) {
      s.push(token);
    } else {
      int k = close.indexOf(token);
      if (k >= 0)
        if (s.isEmpty() ||
            !open.substring(k, k + 1).equals(s.pop()))
          return false;
    }
  }
  return s.isEmpty();
}
```

a.indexOf(b) คืนตำแหน่งในสตริง a ที่มี b ปรากฏอยู่ ถ้าหาไม่พบคืน –1

ผิด ถ้ากองซ้อนว่าง หรือวงเล็บปิดไม่ตรงกับเปิด

ผิด ถ้ากองซ้อนไม่ว่าง

การใช้กองซ้อนภายใน java virtual machine

- ตัว jvm ใช้กองซ้อน (java stack) ในการเก็บ
  - สถานะของการเรียกเมท็อด
  - พารามิเตอร์ และ local variables
  - ที่เก็บชั่วคราวเพื่อการคำนวณ (operand stack)

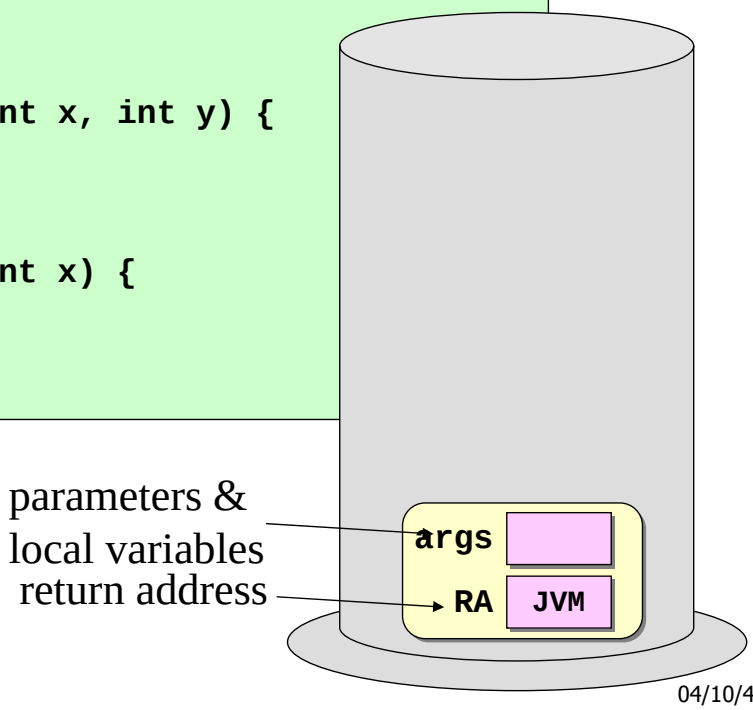
```
public class Jeng3 {
  public static void main(String[] a) {
    main(args);
  }
}
```

```
Exception in thread "main" java.lang.StackOverflowError
        at Jeng3.main(Jeng3.java:3)
```

Java Stack

```
01:public class StackFrame {
02:  public static void main(String[] args) {
03:    a(3, 2);
04:    b(5);
05:  }
06:  static void a(int x, int y) {
07:    int z = x/y;
08:    b(z);
09:  }
10:  static void b(int x) {
11:    ++x;
12:  }
13:}
```

กรอบกองซ้อน (Stack Frame)



นิพจน์ Infix และ Postfix

- infix (เติมกลาง)
  - a + b * c / d – 2, (a + b) * c / (d – 2)
  - ต้องกำหนดลำดับการทำก่อนหลังของ operators
  - ใช้วงเล็บช่วย
- postfix (เติมหลัง)
  - a b c * d / + 2 – , a b + c * d 2 - /
  - ลำดับการทำงานของ operators คือจาก ซ้ายไปขวา
  - ไม่จำเป็นต้องมีวงเล็บ

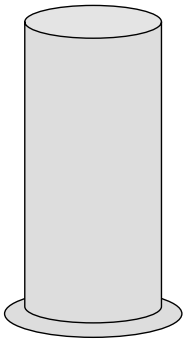
```
1   2   +   3   *   4   1   -   /
        3   3   *   4   1   -   /
                9   4   1   -   /
                9           3   /
                                3
```

การใช้กองซ้อนคำนวณค่าของนิพจน์เติมหลัง

- 2 3 + 4 5 – 6 * + มีค่าเท่าไร ?
- สามารถใช้ stack ช่วยหาค่าของนิพจน์ postfix
- วิธีทำ
  - ดูทีละตัวใน postfix จากซ้ายไปขวา
  - ถ้าเป็น operand ให้ push ของ stack
  - ถ้าเป็น operator ให้ pop operands จาก stack ตามที่ operator ต้องการมาประมวลผล แล้ว push ผลลัพธ์
  - ทำเสร็จ คำตอบจะอยู่ที่ top of stack

ตัวอย่าง

2	3	+	4	-



ผลลัพธ์อยู่บนกองซ้อน

การใช้กองซ้อนช่วยแปลง infix เป็น postfix

- input : infix expression
- output : postfix expression
- ขั้นตอนการทำงาน
  - ดูแต่ละตัวใน infix
  - ถ้าเป็น operand นำไปต่อท้าย output
  - ถ้าเป็น operator
    - อาจ pop operator ออกไปต่อท้าย output
    - push operator ลงกองซ้อน
  - เมื่อดู infix ครบตัวแล้ว
    - ให้ pop operators ทุกตัวออกไปต่อท้าย output

เก็บใน list

โครงของโปรแกรม

```
public static List infix2Postfix(List infix) {
  List postfix = new ArrayList(infix.size());
  Stack s = new ArrayStack(infix.size());
  for (int i = 0; i < infix.size(); ++i) {
    String token = (String) infix.get(i);
    if (!isOperator(token)) {
      postfix.add(token);
    } else {
      while (  ???  ) {
        postfix.add(s.pop());
      }
      s.push(token);
    }
  }
  while (!s.isEmpty()) postfix.add(s.pop());
  return postfix;
}
```

ดูทีละตัว

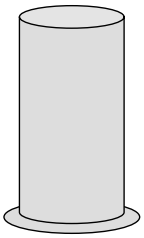
ถ้าเป็น operand, เพิ่มต่อในผลลัพธ์

ถ้าเป็น operator
อาจ pop operators ใน stack ออกเป็นผลลัพธ์
ตามด้วยการ push operator ตัวใหม่

ความสำคัญของ operators

output

input | 2 | * | 3 | + | 4 |



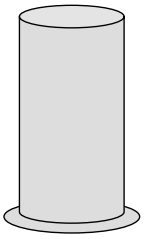
* มากก่อน และสำคัญกว่า +

pop ออก ถ้า operator บนกองซ้อนสำคัญกว่า operator ใหม่

ความสำคัญของ operators

output

input | 2 | + | 3 | - | 4 |



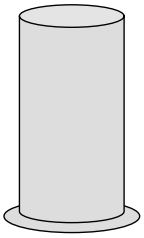
+ มากก่อน
และสำคัญเท่ากับ -

pop ออก ถ้า operator บนกองซ้อน
สำคัญไม่น้อยกว่า operator ใหม่

ความสำคัญของ operators

output

input | 2 | + | 3 | * | 4 |



+ มาก่อน แต่ * สำคัญกว่า +

push ทับ ถ้า operator ใหม่สำคัญกว่า operator บนกองซ้อน

การเปรียบเทียบความสำคัญของ operators

```
public static List infix2Postfix(List infix) {
  List postfix = new ArrayList(infix.size());
  Stack s = new ArrayStack(infix.size());
  for (int i = 0; i < infix.size(); ++i) {
    String token = (String) infix.get(i);
    if (!isOperator(token)) {
      postfix.add(token);
    } else {
      int p = priority(token);
      while (!s.isEmpty() && priority((String)s.peek())>= p) {
        postfix.add(s.pop());
      }
      s.push(token);
    }
  }
  while (!s.isEmpty()) postfix.add(s.pop());
  return postfix;
}
```

หาค่าความสำคัญของ operator ตัวใหม่ที่พบ

pop ออก ถ้า operator บนกองซ้อน สำคัญไม่น้อยกว่า operator ใหม่

การหาความสำคัญของ operators

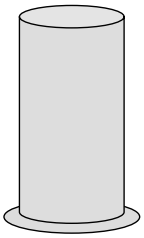
- ให้ ^ แทนการยกกำลัง 7
- ^ ทำก่อน + – * /
- * กับ / ทำก่อน + กับ –
- * มีความสำคัญเท่ากับ / 5
- + มีความสำคัญเท่ากับ – 3

```
private static String operators = "+-*/^";
private static int[] priority =  {3,3,5,5,7};
private static boolean isOperator(String x) {
  return operators.indexOf(x) >= 0;
}
private static int priority(String x) {
  return priority[operators.indexOf(x)];
}
```


ซับซ้อนขึ้นเมื่อมีวงเล็บใน infix

output

input 2 * (3 + 4) ...



- ภายในวงเล็บเสมือนเป็นนิพจน์ย่อย
- พบวงเล็บเปิด push เสมอ (มีความสำคัญมาก)
- วงเล็บเปิดในกองซ้อน มีความสำคัญน้อยมาก
- พบวงเล็บปิด ลุย pop จนกว่าจะพบวงเล็บเปิด

ความสำคัญของ operators กรณีมีวงเล็บ

- ขณะพบวงเล็บเปิด
  - มีความสำคัญมาก, จึง push (เสมอ
- แต่พอวงเล็บเปิดอยู่ในกองซ้อน
  - มีความสำคัญน้อยสุด, เพื่อให้ตัวอื่น push ทับ, ยกเว้น)

```
int p = priority(token);
while (!s.isEmpty() && priority((String)s.peek())>= p) {
  postfix.add(s.pop());
}
s.push(token);
```

```
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);
```


ความสำคัญของ operators กรณีมีวงเล็บ

	+	-	*	/	^	(	)
ขณะที่พบในนิพจน์ (อยู่นอกกองซ้อน)	3	3	5	5	7	9	1
ขณะอยู่ในกองซ้อน	3	3	5	5	7	0	

```
  private static String operators = "+-*/^()";
  private static int[] outPriority =  {3,3,5,5,7,9,1};
  private static int[] inPriority  =  {3,3,5,5,7,0};
  private static int outPriority(String x) {
    return outPriority[operators.indexOf(x)];
  }
  private static int inPriority(String x) {
    return inPriority[operators.indexOf(x)];
  }
```

operator ที่ทำจากซ้ายไปขวา

	+	-	*	/	^	(	)
ขณะที่พบในนิพจน์ (อยู่นอกกองซ้อน)	2	2	4	4	6	9	1
ขณะอยู่ในกองซ้อน	3	3	5	5	7	0	

2 + 3 + 4 + 5

2 3 + 4 + 5 +

```
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);
```


operator ที่ทำจากขวาไปซ้าย

	+	-	*	/	^	(	)
ขณะที่พบในนิพจน์ (อยู่นอกกองซ้อน)	3	3	5	5	8	9	1
ขณะอยู่ในกองซ้อน	3	3	5	5	7	0	

2 ^ 3 ^ 4 ^ 5

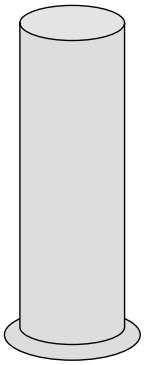
2 3 4 5 ^ ^ ^

```
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);
```

ตัวอย่าง

output

input | 2 | + | 3 | - | 4 | ^ | 5 | ^ | 6 |



สรุป

- ประยุกต์กองซ้อนในการแก้ปัญหาหลากหลาย
- การดำเนินการหลัก : push / pop / peek
- สร้างกองซ้อนได้ง่ายด้วยอาเรย์
- ถ้าจองขนาดให้เพียงพอ การทำงานทุกครั้งเป็น $\Theta(1)$