Recursion (การเรียกซ้ำ)

ตอนที่ 1 นิยาม

วัตถุจะมีการเรียกตัวเองซ้ำ หรือมีการทำงานแบบ Recursive เมื่อมีองค์ประกอบส่วนหนึ่งเป็นตัวเอง หรือ มีการอ้างอิงถึงตัวเองตัวอย่างเช่น นิยามของจำนวนธรรมชาติ (natural number - N) ดังต่อไปนี้

จำนวนธรรมชาติ

ก. 1 เป็นจำนวนธรรมชาติ

ข. ตัวเลขที่มีตำแหน่งอยู่ด้านหลัง (successor) ของจำนวนธรรมชาติ เป็นจำนวนธรรมชาติ

หรือ

ก. 1 เป็นจำนวนธรรมชาติ

ข. หาก '*n*' เป็นจำนวนธรรมชาติแล้ว '*n* + 1' เป็นจำนวนธรรมชาติ

ศักยภาพของการเรียกตัวเองซ้ำอยู่ที่สามารถนิยามเซตไม่จำกัดของวัตถุที่สนใจได้ โดยใช้ถ้อยคำหรือคำสั่งที่จำกัดได้ โดยทั่วไปนิยามแบบเรียกตัวเองมีองค์ประกอบสองอย่างคือ base case ซึ่งเป็นขั้นตอนที่สามารถหาคำตอบได้โดยตรง และ general case ซึ่งได้แก่ขั้นตอนส่วนที่เหลือซึ่งยังไม่สามารถแก้ปัญหาโดยตรงได้ ต้องทำการลดขนาดปัญหาและเรียกตัวเองซ้ำ ดังตัวอย่างต่อไปนี้

นิยามของ Factorial

$$n! = \begin{cases} 1 & \text{ถ้า } n = 0 \text{ หรือ } n = 1 \quad \text{<base-case>} \\ n * (n-1)! & \text{ถ้า } n > 1 \quad \text{<general-case>} \end{cases}$$

นิยามของอนุกรม Fibonacci

$$f(n) = \begin{cases} 1 & \text{ถ้า } n = 0 \text{ หรือ } n = 1 \quad \text{<base-case>} \\ f(n-1) + f(n-2) & \text{ถ้า } n > 1 \quad \text{<general-case>} \end{cases}$$

แบบฝึกหัด จากนิยามของอนุกรม Fibonacci จงหาสมาชิก 10 ลำดับแรกของอนุกรม

... ...

ตอนที่ 2 ตัวอย่างการประยุกต์ Factorial และ Fibonacci

เมื่อต้องการนำนิยามแบบเรียกตัวเองซ้ำมาประยุกต์เป็นโปรแกรม ต้องเขียนในรูปของ method จึงจะสามารถเรียกตัวเองซ้ำได้ และใช้โครงสร้างแบบมีการเลือกสำหรับ base case และ general case เช่น นิยามของ Factorial เป็น

$$n! = \begin{cases} 1 & \text{ถ้า } n = 0 \text{ หรือ } n = 1 \quad \text{<base-case>} \\ n * (n-1)! & \text{ถ้า } n > 1 \quad \text{<general-case>} \end{cases}$$

จัดให้เป็นโครงสร้างแบบมีการเลือกแบบ if … else … ตามโครงสร้างทางคณิตศาสตร์ได้เป็น

```
if  n = 0 or n = 1 then
   return 1
else
   return n * (n-1)!
```

ทางเลือกในส่วนของ else มีการเรียกตัวเองซ้ำ (นั่นคือ ส่วนที่ขีดเส้นใต้ เรียก factorial อีกครั้ง) ดังนั้นจึงต้องกำหนดให้การทำงานในส่วนนี้เป็น method เพื่อให้สามารถตั้งชื่อได้ และใช้ชื่อนั้นในการเรียกตัวเองซ้ำ เช่นหากกำหนดให้เป็น method สำหรับหาค่า factorial เป็น method ชื่อ fact จะเขียนโปรแกรมเป็นภาษา Java ได้ดังนี้

ตัวอย่างที่ 1 method สำหรับหาค่า factorial ของ n ชนิดเรียกตัวเองซ้ำ

```
public static void   fact ( int   n ) {
      if  ( n == 0 || n == 1 )            // นิยามพื้นฐาน
          return 1;
      else
          return n * fact (n-1);          // เรียกตัวเองซ้ำ
}
```

เมื่อเขียน method ได้แล้วจึงเขียนโปรแกรมหลัก (method main ในภาษา Java) เพื่อเรียก method fact ให้ทำงาน พร้อมด้วยจำนวนเต็มที่ต้องการหา factorial (ข้อมูลที่ต้องการหาคำตอบนี้เรียกว่าค่าเป้าหมาย หรือ goal) จากนั้น method fact จะทำการลดขนาดข้อมูลและเรียกตัวเองซ้ำจนกว่าข้อมูลนั้นจะเล็กพอที่จะสามารถหาคำตอบได้โดยตรง ขั้นตอนส่วนนี้เรียกว่า Winding phase และเมื่อได้คำตอบแล้วจะนำคำตอบที่ได้คำนวณกลับจนได้คำตอบของเป้าหมายที่ต้องการ ขั้นตอนส่วนนี้เรียกว่า Unwinding phase ตัวอย่างเช่นเมื่อโปรแกรมหลักมีการเรียกใช้ fact(5) มีการทำงานดังนี้

Winding phase

```
fact(5)
    5 * fact(4)                                 ปัญหายังมีขนาดใหญ่
          4 * fact(3)                           ยังไม่สามารถ หาคำตอบได้
                3 * fact(2)                     เรียกตัวเองซ้ำ จนกระทั่งถึง
                      2 * fact(1)               1! = 1 โดยนิยาม
-------------------------------------------------------------------------------------------
                      2 * 1                     ได้คำตอบแล้วเริ่มคำนวณกลับ
                3 *   2                       เพื่อหาคำตอบเป้าหมายตามที่
          4 *     6                             ผู้เรียกต้องการ
    5 *     24
120
```

Unwinding phase

<u>ตัวอย่างที่ 2</u> method สำหรับคำนวณค่าสมาชิกลำดับที่ n ของอนุกรม Fibonacci ชนิดเรียกตัวเองซ้ำ

<u>นิยาม</u>

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \quad \text{<base-case>} \\ f(n-1) + f(n-2) & \text{if } n > 1 \quad \text{<general-case>} \end{cases}$$

<u>method ภาษา Java</u>

```
public static int   fibo ( int   n ) {
      if ( n == 0 || n == 1 )                   // base-case
          return 1;
      else
          return fibo (n-1) + fibo (n-2);       // เรียกตัวเองซ้ำ
}
```

วิธีการแก้ปัญหาโดยการลดขนาดของปัญหาเป้าหมายหรือข้อมูลเป้าหมายลงจนมีขนาดเล็กพอที่จะหาคำตอบได้ จากนั้นจึงคำนวณย้อนกลับขึ้นไปหาข้อมูลเป้าหมายจนได้คำตอบที่ต้องการ ตามตัวอย่างที่กล่าวมาแล้วนี้มีเรียกเฉพาะว่าแก้ปัญหาแบบ "แบ่งแยกแล้วเอาชนะ" (Divide and conquer) ซึ่งทุกครั้งที่มีการลดขนาดของเป้าหมายลง จะเรียกใช้วิธีการเดิม (เรียกตัวเองซ้ำ) เพื่อแก้ปัญหาทุกครั้ง วิธีการนี้เป็นวิธีการที่สำคัญในการแก้ปัญหาด้านโครงสร้างข้อมูลและขั้นตอนวิธีซึ่งจะได้ศึกษาต่อไป

ตอนที่ 3 ประสิทธิภาพและความเหมาะสมในการใช้ขั้นตอนวิธีแบบเรียกตัวเองซ้ำ

ขั้นตอนวิธีแบบเรียกตัวเองซ้ำ (Recursive) เหมาะสมที่จะใช้แก้ปัญหาที่มีนิยามอ้างอิงถึงตัวเอง หรือใช้ดำเนินการกับข้อมูลที่มีลักษณะอ้างอิงเข้าหาตัวเอง แต่ไม่ได้หมายความว่าขั้นตอนวิธีแบบเรียกตัวเองซ้ำจะเป็นวิธีการที่ดีที่สุดหรือเป็นวิธีการที่มีประสิทธิภาพมากที่สุดในการแก้ปัญหา หากพิจารณาขั้นตอนวิธีในการสร้างตัวเลขในอนุกรม Fibonacci แบบเรียกตัวเองซ้ำ จะเห็นว่าเป็นวิธีการที่มีจุดอ่อนในการทำงาน กล่าวคือหากต้องการหาค่าของสมาชิกของอนุกรมลำดับที่ n หรือ S_n ต้องทำการคำนวณหาค่าของสมาชิกที่อยู่ในลำดับก่อนหน้านั้นทุกตัว คือ S_1, ... , S_{n-1} ทำให้ต้องมีการคำนวณวนเป็นจำนวนมาก โดยเฉพาะเมื่อค่าของ n สูงขึ้น จึงมีความพยายามหาสูตรสำหรับใช้คำนวณโดยตรงเพื่อช่วยให้โปรแกรมทำงานได้เร็วขึ้น

ตัวอย่างที่ 3 นิยามแบบเรียกตัวเองซ้ำของอนุกรมหนึ่งเป็นดังนี้

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 * g(n-1) & \text{if } n > 0 \end{cases}$$

เมื่อนำมาเขียนเป็นโปรแกรมแบบเรียกตัวเองซ้ำในภาษา Java ได้ดังนี้

```
//  method ลดค่าของ n ลงและ เรียกตัวเองซ้ำ จนถึงระดับที่หาคำตอบได้โดยตรง
public static int   g ( int   n )
{
    if ( n == 0 )
        return 1;
    else
        return 2 * g(n-1);
}

// method หลัก
public static void main (String[] args) {
    int       n;

    // พิมพ์ค่าของ g(n) เมื่อ 0 ≤ n < 10
    for ( n = 0; n < 10; ++n )  {
        System.out.print(g(n) + ", ");
    }
    System.out.println();
}
```

เมื่อทำการ run โปรแกรม ผลลัพธ์ที่ได้เป็นดังนี้

1, 2, 4, 8, 16, 32, 64, 128, 256, 512,

ตัวอย่างที่ 4 การหาผลบวกของอนุกรม 1 + 2 + ... + n สามารถทำได้ 3 แบบคือ

- การใช้รูปแบบปิด (closed form คือการหาสูตร), f(n) = (n(n + 1))/2
- การใช้ method แบบเรียกตัวเองซ้ำ
- การใช้โครงสร้างทำซ้ำ

```
//  method หาผลบวกแบบใช้โครงสร้างทำซ้ำ (iterative หรือ loop)
public static int sumi ( int   n ) {
    int       acc = 0, i;

    for ( i = 1; i <= n; ++i )
        acc += i;
    return acc;
}
```

```
//  method หาผลบวกแบบเรียกตัวเองซ้ำ (recursive)
public static int sumr ( int   acc, int  n ) {
    if ( n == 0 )
        return acc;
    else
        return n + sumr(acc, n - 1);
}

// โปรแกรมหลัก
public static void   main (String[] args) {
    int       n = 10;

    System.out.println("sum = " + (n * (n + 1))/2);       // รูปแบบปิด
    System.out.println("sum = " + sumr(0, 10));           //  method แบบเรียกตัวเองซ้ำ
    System.out.println("sum = " + sumi(10));              //  method แบบใช้โครงสร้างทำซ้ำ
}
```

<u>คำถาม</u> จากรูปแบบปิด, f(n) = (n(n + 1))/2 หากกำหนดให้ m และ n เป็นตัวแปรชนิด **int** แล้ว นิพจน์ m = n * (n + 1)/2 จะให้ผลลัพธ์เท่ากับ m = n * ((n + 1)/ 2) และ m = (n * (n + 1))/ 2 หรือไม่ หากไม่เท่ากัน สามารถอธิบายสาเหตุของความแตกต่างนี้ได้หรือไม่ (คำแนะนำ พิจารณาโดยใช้ลำดับความสำคัญของเครื่องหมาย และการหารจำนวนเต็ม (integer division – div))

จากตัวอย่างที่ 4 จะเห็นได้ว่าการทำงานซ้ำโดยใช้ method แบบเรียกตัวเองซ้ำ สามารถแทนได้ด้วยโครงสร้างชนิดทำซ้ำ ซึ่งอาจใช้โครงสร้าง while, do .. while หรือ for ก็ได้แล้วแต่ลักษณะของงาน หรืออาจกล่าวได้อีกนัยหนึ่งว่า การทำงานซ้ำโดยใช้โครงสร้างแบบทำซ้ำ และการทำซ้ำโดยใช้ method ชนิดเรียกตัวเองเป็นการดำเนินการที่ให้ผลลัพธ์เหมือนกันและสามารถใช้ทดแทนกันได้

<u>แบบฝึกหัด</u>

๑. จงแปลง method factorial จาก method ชนิดเรียกตัวเองซ้ำเป็นโครงสร้างแบบทำซ้ำ

๒. จงแปลง method fibonacci จาก method ชนิดเรียกตัวเองซ้ำเป็นโครงสร้างแบบทำซ้ำ โดยให้พยายามลองคิดวิธีการเองดูก่อน หากไม่สามารถทำได้จึงศึกษาจากแนวทางการแก้ปัญหาต่อไปนี้

<u>คำแนะนำ</u> วิธีการแก้ปัญหาแบบหนึ่งทำได้โดยการใช้ตัวแปรสามตัว เช่น a, b และ c แทนสมาชิกใดๆ ที่เรียงลำดับกันสามตัว เมื่อทำการเลื่อนกรอบที่ประกอบด้วยตัวแปรทั้งสามนี้ไปตามลำดับของสมาชิกอนุกรมจะได้ค่าของสมาชิกทั้งสามที่ตำแหน่งนั้น หากเริ่มต้นที่สมาชิกตัวแรก และเลื่อนกรอบไปครั้งละตำแหน่งจะได้ผลดังนี้

```
a  b  c
1  1  2  3  5  8  13  21  34 ...      a = 1, b = 1, c = a + b = 2

   a  b  c
1  1  2  3  5  8  13  21  34 ...      a = 1, b = 2, c = a + b = 3

      a  b  c
1  1  2  3  5  8  13  21  34 ...      a = 2, b = 3, c = a + b = 5
```

การแทนโครงสร้างทำซ้ำด้วย method แบบเรียกตัวเองซ้ำ สามารถทำได้ทั้งในในกรณีที่เป็นโครงสร้างทำซ้ำชนิดชั้นเดียวและโครงสร้างทำซ้ำชนิดหลายชั้น ขอให้ศึกษาจากโปรแกรมต่อไปนี้

ตัวอย่างที่ 5 โปรแกรมที่ใช้โครงสร้างทำซ้ำชนิดสองชั้นซ้อนกัน

```
// พิมพ์เครื่องหมายดอกจันจำนวน n แถว แต่ละแถวพิมพ์ดอกจันเป็นจำนวนเท่ากับหมายเลขบรรทัด
// บรรทัดที่ 1 พิมพ์ดอกจัน 1 ดอก, บรรทัดที่ 2 พิมพ์ดอกจัน 1 ดอก, ... …
public static void stari ( int   n ) {
    int row, col;

    for ( row = 1; row <= n; ++row ) {
        for ( col = 1; col <= row; ++col )
            System.out.print("*");
        System.out.println();
    }
}

// โปรแกรมหลัก
public static void  main (String[] args) {
    stari(10);
}
```

ตัวอย่างที่ 6 โปรแกรมที่ใช้ method ชนิดเรียกตัวเองซ้ำสองโปรแกรมที่มีการทำงานเทียบเท่ากับโปรแกรมในตัวอย่างที่ 5

```
// col - พิมพ์เครื่องหมาย '*' ในแต่ละแถวด้วยการเรียกตัวเองซ้ำ
public static void col ( int   star ) {
    if ( star == 0 ) {
        System.out.println();
    } else {
        System.out.print("*");
        col(star - 1);
    }
}

// row - พิมพ์บรรทัดแต่ละบรรทัดด้วยการเรียกตัวเองซ้ำ
public static void    row ( int   start, int   stop ) {
    if ( start <= stop ) {
        col(start);
        row(start + 1, stop);
    }
}

// โปรแกรมหลัก
public static void  main (String[] args) {
    row(1, 10);
}
```

<u>แบบฝึกหัด</u>

1. กำหนด method g ดังนี้

```
public static int    g ( int   x, int   y ) {
    if ( x < y )
        return -3;
    else
        return  g(x - y, y + 3) + y;
}
```

จงหาค่าที่ method g คืนให้แก่ผู้เรียก เมื่อถูกเรียกใช้ดังนี้คือ

g(2, 7) = _____

g(5, 3) = _____

g(15, 3) = _____

2. กำหนด method h ดังนี้

```
public static int    h ( int   x, int   y ) {
    if ( x > y )
        return -1;
    else if ( x == y )
        return 1;
    else
        return x * h(x + 1, y);
}
```

จงหาค่าที่ method h คืนให้แก่ผู้เรียก เมื่อถูกเรียกใช้ดังนี้คือ

h(10, 4) = _____ h(4, 3) = _____

h(4, 7) = _____ h(0, 0) = _____

3. กำหนดให้ $f(x) = \begin{cases} 3 * x & \text{ถ้า } x < 5 \\ 2 * f(x - 5) + 7 & \text{ถ้า } x \geq 5 \end{cases}$

3.1 จงหาค่าของ f(4), f(10) และ f(12)

f(4) = _____ f(10) = _____ f(12) = _____

3.2 จงเขียน method f ตามนิยามข้างต้น