自己呼叫自己...一直一直呼叫...快瘋了!
找不到遞迴的真諦....遞迴真的不是人寫的!

遞迴 Recursion

簡介

方法或函式自己呼叫自己，稱之為遞迴(recursion)。

一定要定義返回條件，否則程式無法結束。

遞迴程式通常程式碼很少很簡潔，寫一個複雜問題的遞迴，需要很清晰的頭腦，邏輯能力要很強。

寫遞迴程式須注意：

呼叫的需要使用堆疊 stack 紀錄返回位址(就是紀錄是誰呼叫，堆疊是後進先出)。stack 存入取出需花時間，因此，執行速度比不上使用迴圈方式的程式。

因為使用堆疊 stack，recursive 次數太大，超出可用範圍，會產生堆疊溢位 stack overflow 的問題(程式會掛掉)。

遞迴結構 - 階乘運算(Factorial)

所謂的階乘 n!指的是 n! = 1 × 2 × 3 × ... × (n-1) × n
例如：4! = 1 × 2 × 3 × 4 = 24
計算時不將 0 考慮進去，因為任意數乘以 0 答案皆為 0。
也因此在階乘運算上，0! = 1，因此階乘運算需要條件敘述。

使用靜態方法寫一個遞迴的方法

```
package recursion;
public class MyMath {


    public static long factorial(int n) {
        if (n <= 1) {
            return 1;
        } else {
            //System.out.println(n + "*" + (n - 1));
            return n * factorial(n - 1);
```

```
        }
    }

    public static long factorialFor(int n) {
        long result = 1;
        for (int i = n; i >= 1; i--) {
            result *= i;
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println(MyMath.factorial(3));
        //System.out.println(MyMath.factorialFor(5));
    }
}
```

利用 if…else 敘述式來設定返回 return 條件，如果整數 n<=1 則回傳 1，否則回傳以下數值(呼叫自己):

return n * factorial(n - 1)

為何要 n<=1 則回傳 1

if (n<=1) return 1?

if (n ==1) return 1 可以嗎?

or

if (n==0) return 1 可以嗎?

or

if (n<=0) return 1 可以嗎?

定義:

1! = 1

0! = 1 為甚麼?

n!=n*(n-1)!

遞迴結構 – 計算次方(power)

```
public class RecursionPow {
	public static void main(String[] args) {
		int x=3;
		int y=4;
		System.out.println( pow(x,y));
	}
	public static int pow(int x, int y)
	{
		if(y==0)
		{
			return 1;
		} else if(y<=0)
		{
			return 0;
		}
		return x*pow(x, y-1);
	}
}
```

Fabinacci 數列 費氏數列

0 1 1 2 3 5 8 13 21 …

```
public class Fibonacci {
	public static void main(String[] args) {
		//System.out.println(fibonacci(8));
		int index = 0;
		int count = 50; //
		while (index <= count) {
			System.out.printf("%d:%d\n", index, fibonacci(index));
			index++;
		}
```

```
    }
    public static long fibonacci(int n) {
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return fibonacci(n - 1) + fibonacci(n - 2);
        }
    }
    public static long fibonacciFor(int n) {
        long n0 = 0;
        long n1 = 1;
        long n2=1;
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        }
        for (int i = 2; i <= n; i++) {
            n2 = n0 + n1;
            n0 = n1;
            n1 = n2;
        }
        return n2;
    }
}
```

最大公因數

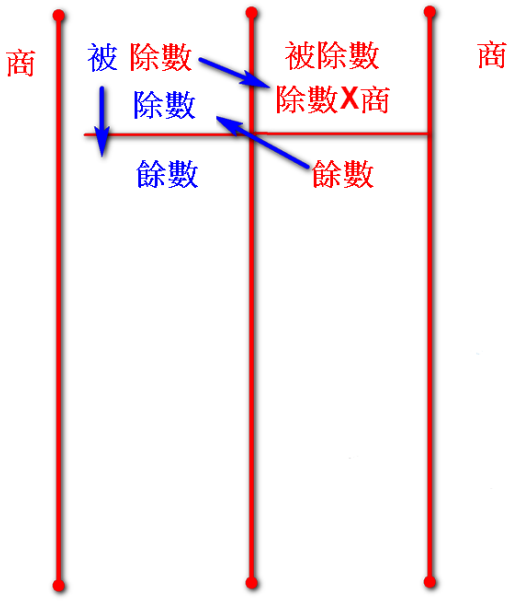
就是輾轉相除(求餘數)，直到餘 0 為止。

國小高年級數學：

[教學影片 1]
[教學影片 2]

實例：

1	123	321	2
	75	246	
1	48	75	1
	27	48	
3	21	27	1
	18	21	
	3	6	2
		6	
		0	



連結：[Scratch 網站]

另例：

x=42,y=75 求 x,y 最大公因數

1. 以較大的數(75)為被除數，較小的數(42)為除數，75 / 42 = 1 餘 33
2. 前一步驟的除數 42 為被除數，餘數 33 為除數，42 / 33 = 1 餘 9
3. 33 / 9 = 3 餘 6
4. 9 / 6 = 1 餘 3
5. 6 / 3 = 2 餘 0，除數 3 即可為最大公因數

程式碼

```
public static int getGcd(int a, int b) {
        if (a % b != 0) {
            return getGcd(b, a % b);
        } else {
            return b;
        }
```

Towers of Hanoi(河內塔)

河內塔是一個數學遊戲：有三個塔柱 A、B、C，將 A 塔柱的圓盤全數移到 C 塔柱。河內 Hanoi 是越南首都。最早發明這個問題的人是法國數學家愛德華·盧卡斯(參看維基百科)。

遊戲規則：

1. 一次只能搬移一個盤
2. 小盤可壓大盤，大盤不可壓小盤

當圓盤總數為 1 時， 移動 1 次
當圓盤總數為 2 時， 移動 3 次
當圓盤總數為 3 時， 移動 7 次
當圓盤總數為 4 時， 移動 15 次

有沒有發現規則？
若有 n 個盤子，需移動 $2^n - 1$ 次

盤子的編號是: 最大的盤子編號最大: n，最小的盤子編號最小: 1。

- 當圓盤總數為 1 時， 移動 1 次

盤子 1 從 A 到 C

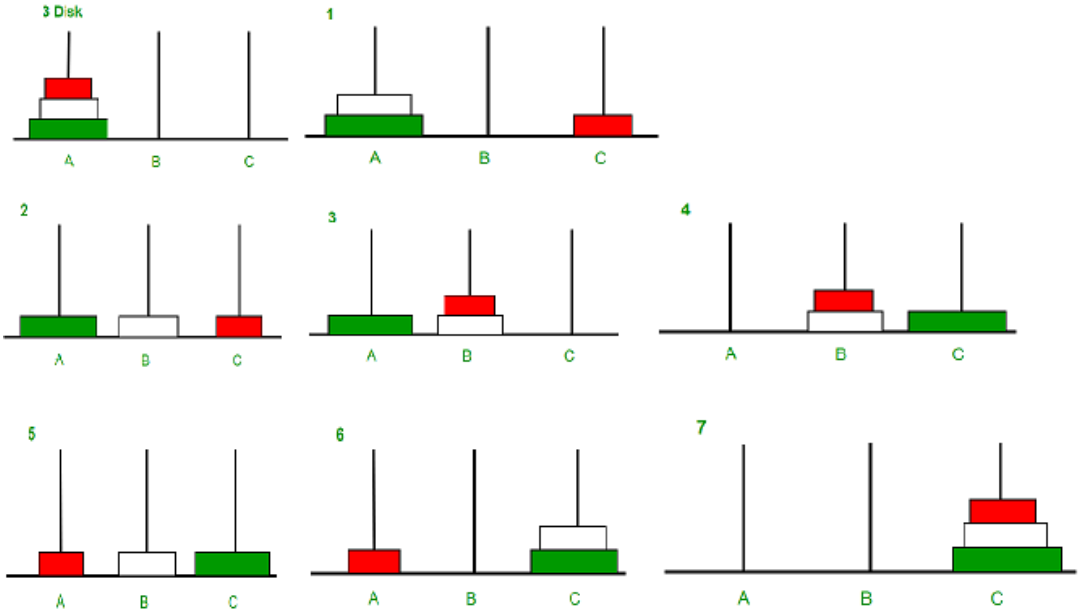
- 當圓盤總數為 2 時， 移動 3 次

盤子 1 從 A 到 B
盤子 2 從 A 到 C
盤子 1 從 B 到 C

- 當圓盤總數為 3 時， 移動 7 次

盤子 1 從 A 到 C
盤子 2 從 A 到 B
盤子 1 從 C 到 B
盤子 3 從 A 到 C

盤子 1 從 B 到 A
盤子 2 從 B 到 C
盤子 1 從 A 到 C



來源

- 當圓盤總數為 4 時， 移動 15 次

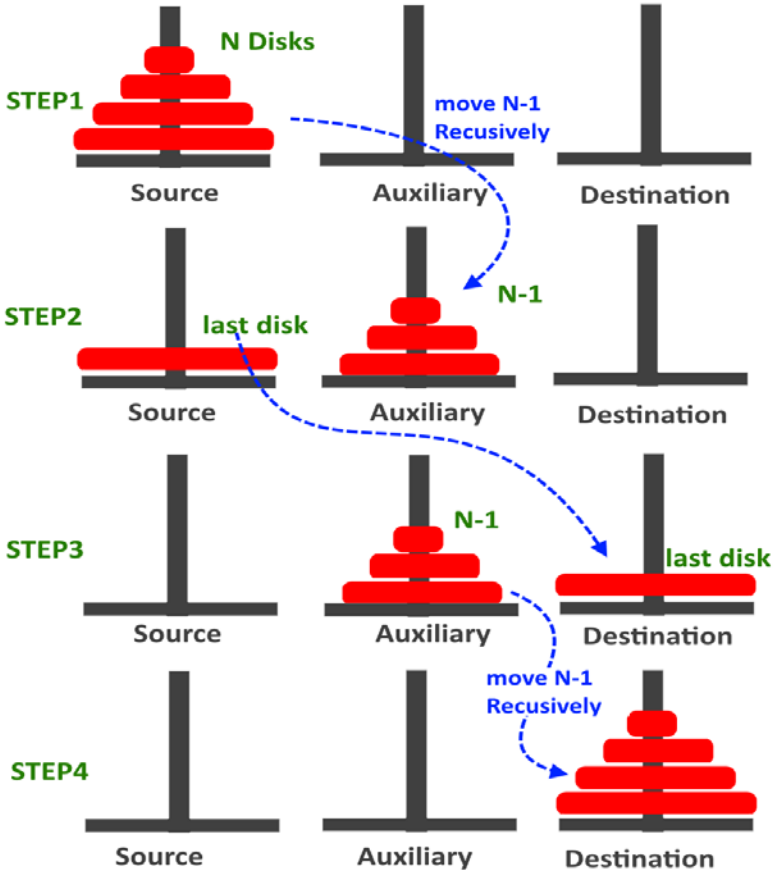
盤子 1 從 A 到 B
盤子 2 從 A 到 C
盤子 1 從 B 到 C
盤子 3 從 A 到 B
盤子 1 從 C 到 A
盤子 2 從 C 到 B
盤子 1 從 A 到 B
盤子 4 從 A 到 C
盤子 1 從 B 到 C
盤子 2 從 B 到 A
盤子 1 從 C 到 A
盤子 3 從 B 到 C
盤子 1 從 A 到 B
盤子 2 從 A 到 C
盤子 1 從 B 到 C

你有沒有發現規律的對稱現象?

當圓盤總數為 1 時，移動 1 次	當圓盤總數為 2 時，移動 3 次	當圓盤總數為 3 時，移動 7 次	當圓盤總數為 4 時，移動 15 次
盤子 1 從 A 到 C	盤子 1 從 A 到 B 盤子 2 從 A 到 C 盤子 1 從 B 到 C	盤子 1 從 A 到 C 盤子 2 從 A 到 B 盤子 1 從 C 到 B 盤子 3 從 A 到 C 盤子 1 從 B 到 A 盤子 2 從 B 到 C 盤子 1 從 A 到 C	盤子 1 從 A 到 B 盤子 2 從 A 到 C 盤子 1 從 B 到 C 盤子 3 從 A 到 B 盤子 1 從 C 到 A 盤子 2 從 C 到 B 盤子 1 從 A 到 B 盤子 4 從 A 到 C 盤子 1 從 B 到 C 盤子 2 從 B 到 A 盤子 1 從 C 到 A 盤子 3 從 B 到 C 盤子 1 從 A 到 B 盤子 2 從 A 到 C 盤子 1 從 B 到 C

1. //步驟 1: 將 n-1 個盤子 移到 暫存區
2. //步驟 2: 將 第 n 個(最大的)盤子 移到 目的區
3. //步驟 3: 將 暫存區的 n-1 個盤子 移到 目的區

不管有多少個盤子，可以簡化為 3 個主要步驟：



來源

教學影片(小孩講解)

from, temp, to

參考程式碼

```
public class Hanoi {


    public static void main(String args[]) {
        hanoi(4, 'A', 'B', 'C');
    }


    public static void hanoi(int n, char from, char temp, char to) {
        if (n == 1)
        {
            System.out.println("盤子" + n + "從" + from + "到" + to);
        }
```

```
        else {
            //步驟 1: 將 n-1 個盤子 移到 暫存區
            hanoi(n - 1, from, to, temp);

            //步驟 2: 將 第 n 個(最大的)盤子 移到 目的區
            System.out.println("盤子" + n + "從" + from + "到" + to);

            //步驟 3: 將 暫存區的 n-1 個盤子 移到 目的區
            hanoi(n - 1, temp, from, to);
        }
    }
```

另一種寫法:

```
package recursion;
public class HanoiTower
{
    public static void main(String args[]) {
        hanoi(2, 'A', 'B', 'C');
    }

    public static void hanoi(int n, char start, char temp, char end) {
        if(n > 0){
            //將所有盤子移到暫存區
            hanoi(n - 1, start, end, temp);
            //將最後一個盤子移到目的區
            display(n, start, end);
            //把暫存區的盤子移回目的區
            hanoi(n - 1, temp, start, end);
        }
    }
    public static void display(int n, char start, char end) {
        System.out.println("盤子" + n + "從" + start + "到" + end);
    }
}
```

習題:請修改階乘的遞迴程式得到以下的輸出

n=5，呼叫 factorial(4)
n=4，呼叫 factorial(3)
n=3，呼叫 factorial(2)
n=2，呼叫 factorial(1)
n=1，呼叫 factorial(0)
n:0，到底了!返回值:1
n=1，返回 factorial(0)的值:1
n=2，返回 factorial(1)的值:1
n=3，返回 factorial(2)的值:2
n=4，返回 factorial(3)的值:6
n=5，返回 factorial(4)的值:24
最終結果:120

以下幾行指令，放置於何處?

```
//觸底返回
System.out.printf("n:%d，到底了!返回值:1\n", n);

//呼叫自己過程
System.out.printf("n=%d，呼叫 factorial(%d)\n", n, n-1);

//返回過程
System.out.printf("n=%d，返回 factorial(%d)的值:%d\n", n, n-1, result);
```

請填空:

```
public static long factorial2(int n) {
    if (n <= 0) {

        return 1;
    } else {
```

```
            long result = factorial2(n - 1);

            return n * result;
        }
    }
```

習題:請修改費氏數列的遞迴程式，展示顯示呼叫與回傳的過程。

習題:

爬樓梯的方式:可以一步踏一階、二階、最長跨三階。因此有很多種爬階方法。
請問爬到第 20 階，有多少種爬法?

想法:

爬到第 1 階，有 1 種爬法
爬到第 2 階，有 2 種爬法
爬到第 3 階，有 4 種爬法:
(1)一次一階。
(2)先爬二階，再爬一階
(3)先爬一階，再爬二階
(4)一次三階。
爬到第 4 階，有 7 種爬法: 4+2+1
爬到第 5 階，有 13 種爬法: 7+4+2
…

說明:

爬到第 4 階，有 7 種爬法: 4+2+1

只看最後一步步伐大小:

1. 最後爬 1 步 前提:站上第 3 階 (不管前面是怎麼走的，只要站上第 3 階)(有 4 種)

2. 最後爬 2 步 前提:站上第 2 階(有 2 種)
3. 最後爬 3 步 前提:站上第 1 階(有 1 種)

只要前面三種前提情況之爬法相加，就是爬到第 4 階的所有可能爬法了。

通則:
一次可跨 1~n 階時(n>=1)，從第 n+1 階開始，其全部可能爬法為前 n 個台階之方法數總和
n>=4 時 f(n) = f(n-1) + f(n-2) + f(n-3)

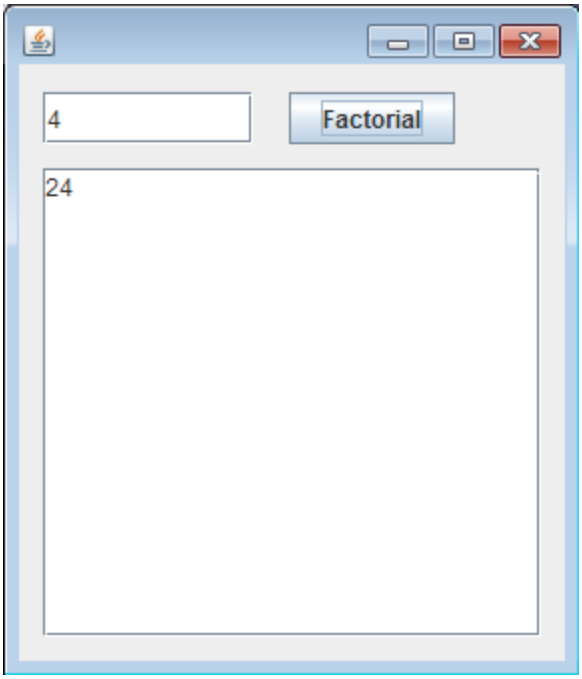
遞迴程式如何思考?
若 n==1 return 1
若 n==2 return 2
若 n==3 return 4
n>=4 return f(n-1) + f(n-2) + f(n-3)

參考程式碼

```
```

遞迴結構 - 階乘運算結合簡易 GUI 介面

搭配簡易版的 GUI 介面，可以讓執行結果更直覺化、更容易操作。



當使用者於 TextField 輸入欲計算階乘之數字後，按下「Factorial」按鈕，下方 TextArea 即會立刻計算出該數字的階乘結果。