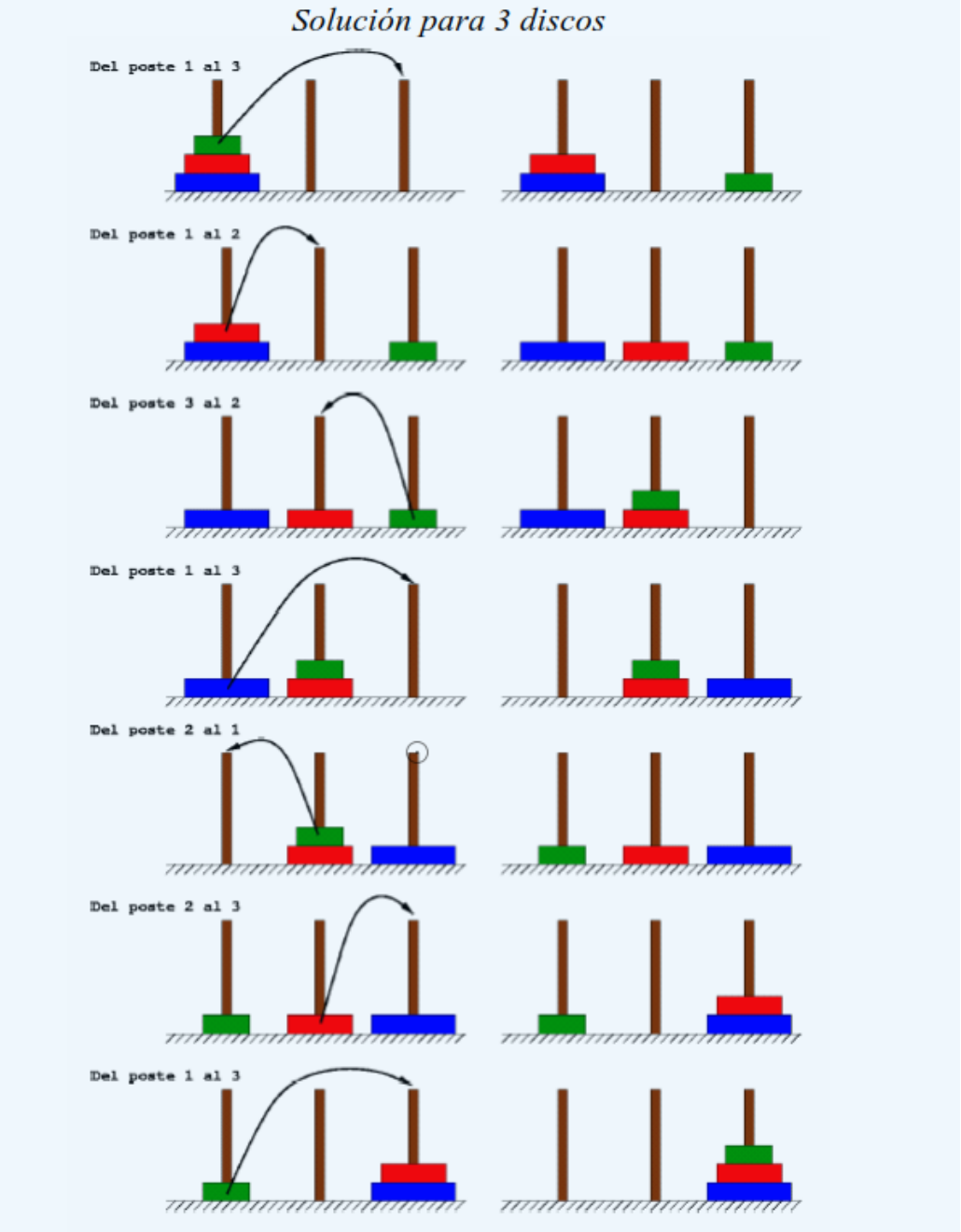
**UN PROBLEMA RECURSIVO DE PURA RAZA:TORRES DE HANOI**

OBJETIVO: no es saber jugar a la perfección a las torres de hanoi,si no entender el problema lo suficiente para captar que es mucho más fácil resolverlo con recursividad que con bucles y que por lo tanto, hay un lugar importante para la recursividad como técnica de programación.

Según la leyenda, los monjes de un templo tenían que mover una pila de 64 discos sagrados de un sitio a otro. Sólo podían mover un disco al día y, en el templo, sólo había otro sitio en el que podían dejarlos, siempre ordenados de forma que los mayores quedasen en la base. El día en que los monjes realizasen el último movimiento, el final del mundo habría llegado… ¿en cuántos días?

Fijate en el siguiente ejemplo que pasa 3 discos de la aguja 1 a la 3 usando la 2 como temporal cumpliendo las dos normas básicas:

- sólo se puede mover un disco de cada vez
- un disco pequeño no puede estar nunca debajo de uno grande



Puedes reproducir el ejemplo anterior en la siguiente página online http://www.uterra.com/juegos/torre_hanoi.php, si el enlace está roto puedes buscar algo similar en google con "torres hanoi online" o similar, por ejemplo https://www.ajedrezeureka.com/torres-de-hanoi/

*Un algoritmo sistemático para resolver el problema*
El siguiente algoritmo es óptimo, es decir, es el que menor número de movimientos requiere, para mover una torre de n discos de la primera varilla a la tercera usando la segunda como temporal.

*Si n = 0, entonces no hacer nada.*
*Si n = 1, entonces mover el único disco de la primera varilla a la tercera.*
*Si n > 1, entonces:*
*1. Mover la torre de n − 1 discos de la primera varilla a la segunda, aplicando el mismo algoritmo considerando la segunda varilla como la tercera y viceversa.*
*2. Mover el n-ésimo disco de la primera varilla a la tercera.*
*3. Mover la torre de n − 1 discos de la segunda varilla a la tercera, aplicando el mismo algoritmo considerando la primera varilla como la segunda y viceversa.*

Además, este algoritmo es el único algoritmo óptimo posible, en el sentido de que cualquier otro algoritmo óptimo ejecuta los mismos pasos.

Para hacer menos abstracto el algoritmo anterior voy a pensar en un n concreto, por ejemplo n=3
mover una torre de 3 discos de la aguja 1 a la 3 usando la 2 como temporal, consiste en:

1. Mover una torre de 2 discos de la aguja 1 a la aguja 2, usando la aguja 3 como un área de almacenamiento temporal.
2. Mover el último disco (el más grande) de la aguja 1 a la aguja 3.
3. Mover la torre de 2 discos de la aguja 2 a la aguja 3, usando la aguja 1 como área de almacenamiento temporal.

esto escrito en java siendo *hanoi(numDiscos, inicio,temp,fin)* la función recursiva:

```
class Unidad3 {

    static void hanoi(int n, int inic, int tmp, int fin) {
        if (n > 0) {

            // Mover n-1 discos de "inic" a "tmp".
            // ahora el temporal es "fin".
            hanoi(n - 1, inic, fin, tmp);

            // Mover el que queda en "inic" a "fin"
            System.out.println(inic + "->" + fin);

            // Mover n-1 discos de "tmp" a "fin".
            // ahora el temporal es "inic".
```

```
            hanoi(n - 1, tmp, inic, fin);
        }
    }

    public static void main(String[] args) {
        hanoi(3, 1, 2, 3);
    }
}
```

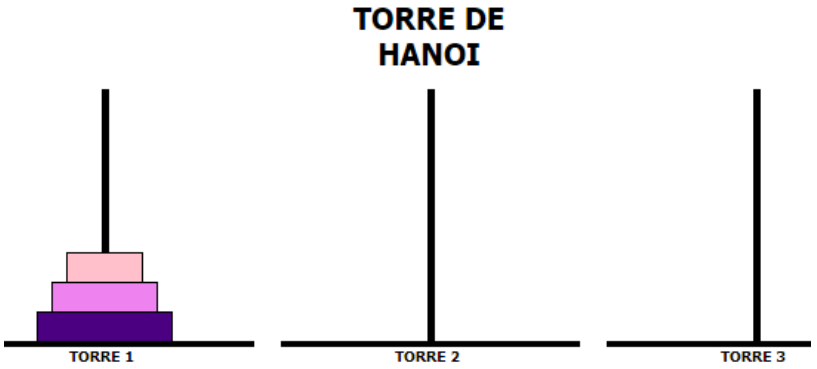
si queremos ver el caso base más claro podemos hacer lo siguiente que es totalmente equivalente

```
class Unidad3 {

    static void hanoi(int n, int inic, int tmp, int fin) {
        if(n==1){
            System.out.println(inic + "->" + fin);
        }else{
                hanoi(n - 1, inic, fin, tmp);
                System.out.println(inic + "->" + fin);
                hanoi(n - 1, tmp, inic, fin);
        }
    }

    public static void main(String[] args) {
        hanoi(3, 1, 2, 3);
    }
}
```

El main lanza el algoritmo para una torre de 3 discos (n=3) con inicio en aguja 1, destino la 3 y por tanto temporal la 2

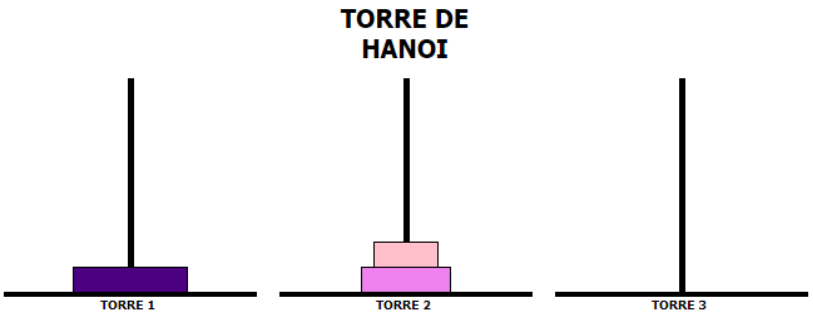



- la primera instrucción es una llamada recursiva *hanoi(n - 1, inic, fin, tmp)*

resuelve un subproblema moviendo una torre de 2 discos a la aguja 2 usando la 3 de temporal
los valores concretos de la llamada serán,
hanoi(2,1,3,2);
observa que cruzamos los valores de los parámetros para redefinir los papeles de las agujas

Ahora me abstraigo, y no pienso como funciona internamente simplemente se que finalmente genera en la aguja 2 una torre de hanoi con 2 discos. Lo que sí es importante observar es que para generar la torre en la aguja 2 tuve que cambiar el papel de las agujas de forma que ahora la aguja 2 es el destino y la 3 la auxiliar.

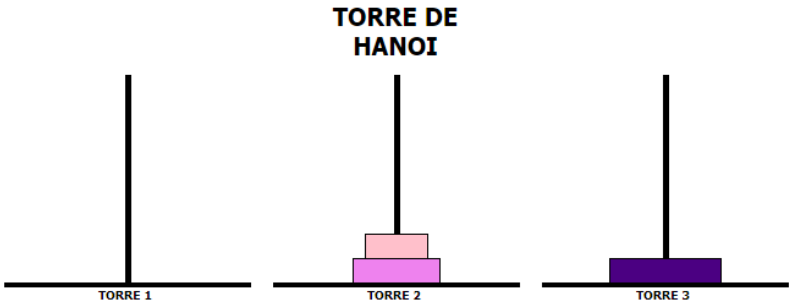
es decir




lógicamente este apilamiento de 2 discos en la aguja 2 a su vez se generó por recursividad pero insisto en que "me abstraigo" y me creo que hace eso

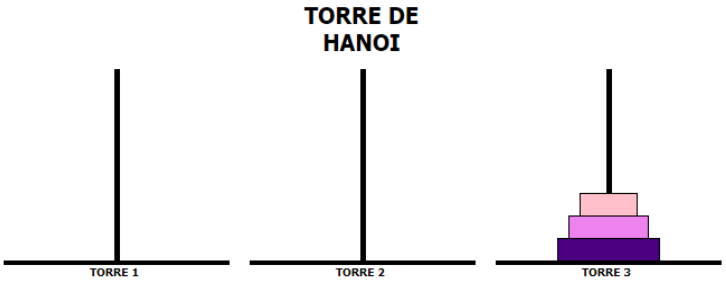
la segunda instrucción es *System.out.println(inic + "->" + fin);*

que mueve disco grande a la aguja 3




la tercera instrucción *hanoi(n - 1, tmp, inic, fin);*

pasa la torre de hanoi de la aguja 2 a la aguja 3 y para eso cambiamos los significados de inicio,fin, auxiliar ahora el inicio es la aguja 2 el auxiliar es la aguja 1 y el destino es la aguja3 y eso lo hace




es importante observar que en todas las agujas van cambiando de papel inicio/fin/temp y esto se consigue cambiando las agujas en las llamadas recursivas

El objetivo, de este boletín no es entender a la perfección el problema y la solución, si no darse cuenta de que hay problemas de naturaleza recursiva que son difíciles de hacer con bucles, en la siguiente dirección viene resuelto en C y en otros lenguajes incluido

java (busca pestaña Java) con bucles: es una solución con 150 líneas y además igual de dificil (o más de entender) que la solución recursiva, si hubiera un error en el programa sería dificil de encontrarlo en una lógica tan espesa:

http://www.geeksforgeeks.org/iterative-tower-of-hanoi/