

Vamos a ver uno de los conceptos más difíciles de la programación imperativa (la que vemos Programación 1), recursividad, por suerte esto no entra en el examen, pero puede caer en Proys y Labs, la recursividad consiste en realizar bucles por medio de funciones y condicionales, es decir, en vez de crear un bucle *for* o *while*, se crea una función que sólo contiene *ifs* y *elses*, vamos a ver algún ejemplo de recursividad:

```

3 public class Recursividad {
4
5     public static void main (String[] args) {
6
7         for (int i = 0; i < 5; i++) {
8
9             System.out.println("Hola");
10
11         }
12
13     }
14
15 }

```

```

3 public class Recursividad {
4
5     public static void funcion (int i) {
6
7         if (i < 5) {
8
9             System.out.println("Hola");
10             funcion(i+1);
11
12         }
13
14     }
15
16     public static void main (String[] args) {
17
18         funcion(0);
19
20     }
21
22 }
23

```

>>>

```

Problems  Javadoc  Declaration  Console X
<terminated> Recursividad [Java Application] C:\Program Files\
Hola
Hola
Hola
Hola
Hola

```

Ambos códigos hacen lo mismo, el primero supongo que lo entendéis, así que explicaré el segundo, llamo a *funcion* y le paso el *int* 0, dentro de *funcion* tenemos un único condicional que si el parámetro que le hemos pasado, en un principio el 0, es menor que 5, se ejecuta, nos imprime “Hola” en pantalla y **luego llama a la misma función pasándole el parámetro i+1**, de tal forma que al volver a ejecutarse éste parámetro valdrá 1, luego 2... y al llegar a 5 parará.

Veamos otro ejemplo que sea la adaptación de un ejercicio anterior, he elegido *cancionElefantes* de *BuclesFor1.java*, que encontrareis en la parte de ejercicios de mi Drive, os pongo la solución tanto en forma iterativa (bucles *for* y *while*), como en recursividad:

```

/*
 * cancionElefantes (int n) : String
 * POST: Devuelve la cancion de los elefantes hasta n veces
 *
 * EJEMPLO: n = 0 ->
 *          n = 3 -> 1 elefantes se balanceaba sobre la tela de una araña.
 *                  2 elefantes se balanceaba sobre la tela de una araña.
 *                  3 elefantes se balanceaba sobre la tela de una araña.
 *
 * Lo que se muestra, es lo que debería de salir en pantalla, si ponemos la sentencia
 * System.out.println(cancionElefantes(n));
 */
public static String cancionElefantes (int n) {

    String res = "";

    for (int i = 1; i <= n; i++) {

        res += i + " elefantes se balanceaban sobre la tela de una araña.\n";

    }

    return res;

}

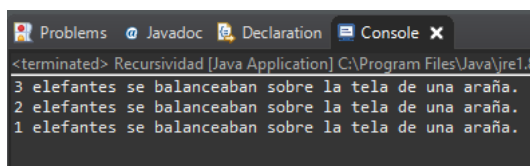
```

```

3 public class Recursividad {
4
5     public static String cancionElefantesRec (int n) {
6
7         if (n == 0) {
8             return "";
9         } else {
10
11             return cancionElefantesRec(n-1) + n + " elefantes se balanceaban sobre la tela de una araña.\n";
12
13         }
14     }
15
16     public static void main (String[] args) {
17
18         System.out.println(cancionElefantesRec(3));
19     }
20 }

```

Explicaré también este segundo paso a paso, cuando $n = 0$, devolvemos un String “vacío”, diferenciad entre un String vacío ("") y uno que no contiene nada (null), en cualquier otro caso, devolvemos la misma función pasándole $n-1$, y luego lo que sería nuestra frase, si lo hacemos al revés, primero la frase y luego la función, no nos la devolverá en el orden que queremos, lo hará así:



```

<terminated> Recursividad [Java Application] C:\Program Files\Java\jre1.8
3 elefantes se balanceaban sobre la tela de una araña.
2 elefantes se balanceaban sobre la tela de una araña.
1 elefantes se balanceaban sobre la tela de una araña.

```

Pero esto es fácil de ver, ya que al iniciar el programa vemos que nos lo imprime al revés, por lo que se deben sumar al revés, la recursividad no es solo una forma *difícil* de hacer bucles, a mí me ha pasado que hay problemas que solo se pueden resolver por recursividad, o por códigos mucho menos eficientes, no comentaré el caso ya que puede resultar bastante complicado, pero os recomiendo que, aunque no entre para el examen, como programadores aprendáis a utilizar esto de la recursividad, dicho esto, os diré que con un poco de práctica todos los bucles *for* y *while* los podréis convertir directamente a recursividad, pero los *do while*, requieren de más imaginación, no son tan directos, y pueden resultar costosos, en el ejercicio que pondré sobre recursividad tendréis que adaptar un *do while*, y ya os digo que os va a costar verlo, pero no miréis directamente la solución, dicho esto, vamos a ver como simplificar *ifs* y códigos en general.

Hay una sentencia que tiene ésta forma:

(condición)? //hacer esto : //hacer lo otro;

Qué es lo mismo que poner:

if (condición) { //hacer esto } else { //hacer lo otro }

Y lo bueno de todo esto es que nos permite hacer recursividad **en una sola línea de código**, veamos la canción de los elefantes de ésta forma:

```

3 public class Recursividad {
4
5     public static String cancionElefantesRec (int n) {
6
7         return (n == 0)? "" : cancionElefantesRec(n-1) + n + " elefantes se balanceaban sobre la tela de una araña.\n";
8     }
9
10    public static void main (String[] args) {
11
12        System.out.println(cancionElefantesRec(3));
13    }
14 }

```

La única diferencia es que el *return* va fuera de la condición, de tal forma que sería algo como:

```
return if (n == 0) "" } else { cancionElefantesRec(n-1) + ... }
```

Pero escritos de una forma más limpia, ponemos poner (condición)? dentro de otros, como si fuesen *ifs* anidados, el problema es que siempre tiene que haber un *else*, es decir, siempre tenemos que poner esos :, pero esto no solo sirve para recursividad, podemos también sumárselo a variables, basémonos en éste programa que te dice si un año es bisiesto o no:

```
public static boolean esBisiesto (int anio) {  
    return (anio % 4 == 0 && anio % 100 != 0) || anio % 400 == 0;  
}
```

Ahora hagamos un programa que nos diga los días de febrero sin utilizar la sentencia anterior:

```
public static int diasFebrero (int anio) {  
    if (esBisiesto(anio)) {  
        return 29;  
    } else {  
        return 28;  
    }  
}
```

Seguro que muchos ya veis como pasarlo a recursividad →

```
public static int diasFebrero (int anio) {  
    return esBisiesto(anio)? 29 : 28;  
}
```

Pero vamos a hacerlo de otra forma, una que algún día os ahorrará varias líneas de código como a mí me ha pasado:

```
public static int diasFebrero (int anio) {  
    return 28 + ((esBisiesto(anio)) ? 1 : 0);  
}
```

Toda la sentencia tiene que estar entre paréntesis, ya que, sino el compilador tratará de sumar un *boolean* en vez del 1 o el 0, y aunque parezca una tontería hacer esto, debido a que en la imagen de arriba es mucho más sencillo, hay casos en los que no podemos simplificar tanto, o que la simplificación cuesta más verla, solo tened en mente que si alguna vez tenéis que sumar, o hacer alguna operación como ésta, y esa operación va a depender de cierta variable, no hace falta crear una función que te lo diga, o hacer un *if* que decida qué valor tiene en cada caso, y ya imaginad si se basa en 2 variables o más, os ahorrarías **muchísimos** condicionales, y esto es todo sobre recursividad, todo ejercicio que tenga que ver con bucles se puede resolver con recursividad, por ello solo os dejaré un ejercicio específico dentro de bucles.

Cabe mencionar que a veces no vamos a poder realizar recursividad solamente con los parámetros de la función, vamos a necesitar crear una **función auxiliar**, ésta función auxiliar se llamará igual que la principal + *Aux*, y recibirá tantos parámetros como queramos, recordad que nuestro único límite es que podemos usar tantas ilimitadas funciones e *ifs*, así que cuando digáis “esto es imposible hacerlo con la función que tengo, necesito otro parámetro”, pues os creáis esa función con ese parámetro, simple.