

## **ESCRIBIR SOLUCIONES RECURSIVAS**

### **ESTRUCTURA DE UN MÉTODO RECURSIVO**

Formalizando lo visto anteriormente, un método recursivo debe contener:

- Uno o más casos base: casos para los que existe una solución directa. También se les llama "paradas" ya que no hacen llamadas recursivas y en ellos "para" la recursividad.
- Una o más llamadas recursivas: casos en los que de alguna forma se incluye una llamada recursiva.

**Caso base:** Siempre ha de existir uno o más casos para los que el método devuelve un valor directo, es decir, sin hacer uso de recursión. Ese valor directo puede ser simplemente no devolver nada, pero siempre es necesario ya que el caso base es donde el método recursivo "se detiene" o "cesa de llamarse".

En el ejemplo anterior el caso base está encubierto, ya que se da cuando  $x \leq 0$  y se corresponde con un `else` inexistente igual a un `else` "no hacer nada"

```
class Recursividad {  
    void imprimir(int x) {  
        if(x>0){  
            imprimir(x-1);  
            System.out.println(x);  
        }  
    }  
}
```

Para entender mejor cuál es el caso base reescribimos el método

```
class Recursividad {  
    void imprimir(int x) {  
        if(x==0){//caso base  
            System.out.print("");  
        }else{//llamadas recursivas  
            System.out.println(x);  
            imprimir(x-1);//no hay infinitas llamadas por que x evoluciona  
        }  
    }  
}
```

**Llamada recursiva:** Si los valores de los parámetros de entrada no cumplen la condición del caso base se llama recursivamente al método. En las llamadas recursivas el valor del parámetro en la llamada se ha de modificar de forma que se aproxime cada vez más hasta alcanzar el valor del caso base. En el ejemplo el caso recursivo sería el bloque

```
{//llamadas recursivas  
    System.out.println(x);  
    imprimir(x-1);// x evoluciona!!!  
}
```

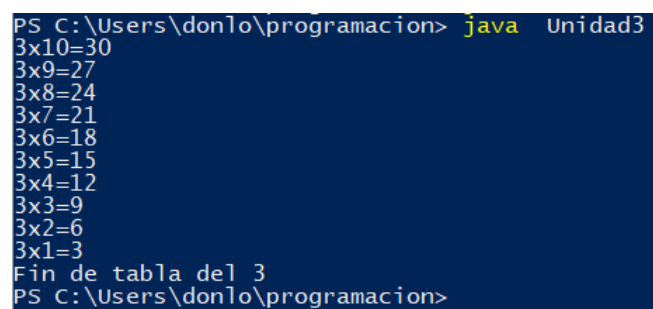
que incluye la llamada recursiva

recuerda que si al trabajar con bucles las variables no evolucionan para llegar a un fin de bucle tenemos un bucle infinito, en recursividad si la evolución de las variables no nos convergen a un caso base también hay infinitas llamadas teóricas que por

agotamiento de espacio en la pila no llegan a ocurrir y la ejecución termina con Stack overflow.

**Ejercicio U3\_B7\_E1:** Escribe recursivamente el método para imprimir una tabla de multiplicar.

```
class Unidad3 {  
    static void tablaMultiplicar(int tabla, int i){  
  
    }  
    public static void main(String[] args) {  
        //por ejemplo la tabla del 3 imprimiendo multiplicaciones desde 10 a 1 inclusive  
        tablaMultiplicar(3,10);  
    }  
}
```



```
PS C:\Users\donlo\programacion> java Unidad3  
3x10=30  
3x9=27  
3x8=24  
3x7=21  
3x6=18  
3x5=15  
3x4=12  
3x3=9  
3x2=6  
3x1=3  
Fin de tabla del 3  
PS C:\Users\donlo\programacion>
```

## LA FRASECITA

Los ejemplos de recursividad anteriores que van imprimiendo cosas en cada paso ilustran bien el funcionamiento interno de la recursividad basada en el funcionamiento de la pila. Pero realmente, para resolver un problema con recursividad el funcionamiento de la pila es un detalle de bajo nivel que no debe *venir a la cabeza* para pensar con libertad a un nivel mayor de abstracción. El camino correcto es pensar en una técnica inductiva matemática, que nosotros resumimos teniendo en cuenta que esencialmente sabemos que tendremos que hacer una serie de iteraciones y pensamos en un paso cualquiera intermedio que definimos genéricamente, y para definirlo, tenemos que ser capaces de hacer una “frasecita” que incluya una definición recursiva. Luego por supuesto tenemos que revisar los casos base para que la recursividad tenga parada y comprobar que la recursión avanza hacia la parada.

## Ejemplo: Multiplicación con recursividad

Hay un operador java, el “\*”, que multiplica números enteros así que no vemos este ejemplo porque sea útil o práctico si no por motivos didácticos ya que la multiplicación se puede ver como un proceso recursivo fácil de entender. Vamos a multiplicar basándonos en una operación más sencilla, la suma.

*Pensando en modo bucle:*

multiplicar(a,b)=>es sumar  $a$ ,  $b$  veces, esto es pensar de forma "bucle" ya que  $b$  veces me "huele" a  $b$  iteraciones  
por ejemplo:  $5*4=5+5+5+5$

*Pensando ahora recursivamente:*

Tengo que esforzarme en hacer una "frasecita recursiva", es decir, una definición recursiva de lo que es multiplicar. La clave es que para que la definición sea recursiva tengo que incluir la palabra "multiplicar/multiplicación", por ejemplo,

*multiplicacion a por b: es sumar a con la **multiplicación** de a por  $b-1$ .*

por ejemplo  $5*4=5+ 5*3$

(observa que en *pensando en modo bucle* no uso la palabra multiplicar o multiplicación en la descripción de la definición)

Una vez que tengo la esencia de mi definición, la enunció con detalle, es decir, con caso base y caso recursivo:

caso base  $b=1$ : devuelve  $a$  (es decir  $a*1=a$ )

caso recursivo  $b>1$ : devuelve  $a + multiplicar(a,b-1)$

En java ....

```
class Unidad3 {  
    public static void main(String[] args) {  
        Multiplicacion m = new Multiplicacion();  
        System.out.println(m.multiplicar(2,10));  
        System.out.println(m.multiplicar(1,3));  
        System.out.println(m.multiplicar(0,9));  
        //System.out.println(m.multiplicar(2,0));  
    }  
}
```

```
class Multiplicacion{  
    int multiplicar(int a, int b){  
        if(b==1)  
            return a;  
        else  
            return a + multiplicar(a,b-1);  
    }  
}
```

La última multiplicación está comentada porque provoca stack overflow ya que cuando  $b$  vale 0 no hay un caso base que pare la recursión. Debes de entender porqué ocurre esto.

La multiplicación se comporta de forma diferente cuando  $b$  es 1 o 0, necesito dos casos base, observa como ahora no hay stack overflow con multiplicar(2,0)

```
class Unidad3 {  
    public static void main(String[] args) {
```

```

        Multiplicacion m = new Multiplicacion();
        System.out.println(m.multiplicar(2,0));
    }
}

class Multiplicacion{
    int multiplicar(int a, int b){
        if(b==1)
            return a;
        else if (b==0)
            return 0;
        else
            return a + multiplicar(a,b-1);
    }
}

```

En realidad el caso base  $b==1$  se puede simplificar ya que si ahora tenemos “la parada”  $b==0$ , el caso  $b==1$  está cubierto por el último else queda pues

```

class Multiplicacion{
    int multiplicar(int a, int b){
        if (b==0)
            return 0;
        else
            return a + multiplicar(a,b-1);
    }
}

```

**Ejercicio U3\_B7\_E2:** Escribir multiplicar() de forma que también se puedan multiplicar números negativos:

```

class Unidad3 {
    public static void main(String[] args) {
        Multiplicacion m = new Multiplicacion();
        System.out.println(m.multiplicar(-2,-5));
        System.out.println(m.multiplicar(2,-5));
        System.out.println(m.multiplicar(-2,5));
        System.out.println(m.multiplicar(-2,0));
        System.out.println(m.multiplicar(2,0));
        System.out.println(m.multiplicar(0,-2));
    }
}

```

## Recursividad y static

El ejemplo de multiplicación lo pudimos haber escrito

```

class Unidad3 {
    static int multiplicar(int a, int b){
        if(b==1)
            return a;
        else
            return a + multiplicar(a,b-1);
    }
    public static void main(String[] args) {

```

```

        System.out.println(multiplicar(2,10));
        System.out.println(multiplicar(1,3));
        System.out.println(multiplicar(0,9));
    }
}

```

En principio para ejemplos tan cortitos no está mal usar static y “nos ahorramos” crear una clase.

## Variables y métodos recursivos

Al escribir un método recursivo, dicho método no debe usar atributos ni static ni de instancia. Usarlas redunda en una solución de peor calidad.

Un método recursivo debe funcionar de forma independiente del estado de variables externas y por lo tanto sólo debe usar variables locales para no corromper el concepto de función recursiva que se base en el concepto matemático de función e inducción en los que no se contempla “variables externas para almacenar resultados temporales”.

¡Esta solución es horrible! Es una solución recursiva pero la variable no local compartida por sucesivas llamadas no cumple al autonomía funcional de cada invocación

```

class Unidad3 {
    static int multiplicacion=0;
    static void multiplicar(int a, int b){
        if(b==1){
            multiplicacion=multiplicacion+a;//FEO Y CONFUSO
        }
        else{
            multiplicacion=multiplicacion + a;//FEO Y CONFUSO
            multiplicar(a,b-1);
        }
    }

    public static void main(String[] args) {

        multiplicar(2,10);
        System.out.println(multiplicacion);
    }
}

```

Por lo tanto: Un método recursivo bien escrito sólo debe manejar variables locales nunca variables static o de instancia.

## Ejercicio U3\_B7\_E3. Calcular el factorial de un número con bucle.

Recuerda la definición de factorial para un número n:

Si  $n = 0$  entonces

$0! = 1$

si  $n > 0$  entonces

$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$

Tu programa tendrá el siguiente aspecto

Su factorial es: 24

$n! = n \cdot (n - 1)!$ , si  $n > 0$  pensando en modo recursivo ...

```
E:\Programacion>javac Unidad3.java  
E:\Programacion>java Unidad3  
#  
##  
###  
####  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

```
public static void main(String[] args) {
```

```

String invertido=reverse("acasohubobuhosaca");
System.out.println(invertido)
System.out.println(reverse("roman"));
}

```

que produce

```

PS C:\Users\donlo\programacion> java Unidad3
acasohubobuhosaca
namor
PS C:\Users\donlo\programacion>

```

## Recursividad versus iteración con bucles

La recursividad es más ineficiente en términos de tiempo y uso de memoria, ya que para cada llamada al método hay que efectuar una serie de operaciones en la pila, no obstante, hay problemas que por su naturaleza recursiva son mucho más fáciles de resolver con recursividad que con intrincados bucles. Por lo tanto, si la eficiencia no es un problema y el problema a resolver tiene una naturaleza recursiva, mejor recursividad ya que se produce un programa más fácil de entender y de depurar. Con todo, este tipo de situaciones, son las menos y lo que abunda es la utilización de bucles más que llamadas recursivas.

## UN CUENTO RECURSIVO

Cuento anónimo: Ocho elefantes blancos

El discípulo quería elaborarlo todo a través del entendimiento intelectual. Solo confiaba en la razón y estaba encerrado en la propia jaula de su lógica. Visitó al mentor espiritual y le preguntó:

–Señor, ¿quién sostiene el mundo?

El mentor repuso:

–Ocho elefantes blancos.

–¿Y quién sostiene a los ocho elefantes blancos? –preguntó intrigado el discípulo.

–Otros ocho elefantes blancos.

Nota del profesor: Aplicando este hermoso cuento a una frecuente oposición del alumno a entender la recursividad, “la jaula lógica” la podemos ver cómo pensar y razonar siempre y sólo en los bucles tradicionales lo que genera resistencia a querer usar y entender otra forma de pensar y razonar como es la recursividad.