Ejercicios de Recursividad

La recursividad consiste en definir una entidad *en función de si misma*

En programación la recursividad nos da la posibilidad de definir un *tipo de datos* en función de si mismo, o bien nos permite definir un *problema* en función de si mismo.

Recursividad de Definición : aplicada a Estructuras de Datos

Posibilidad de definir un tipo de datos en términos de si mismo.

```
public class  Nodo {
        protected Object  elemento;
        protected Nodo  siguiente;

        public  Nodo()
        /* Crea un nuevo objeto nodo */
        { }
        ...
}
```

Recursividad de Ejecución : aplicada a los problemas

Posibilidad de definir un problema en función del propio problema.

Recursividad de Ejecución o Recursividad Funcional

Es aquella que se aplicada a la solución de los problemas y *define el problema en función del propio problema*, lo cual consiste en que *método puede llamarse así mismo* una o varias veces.

En la recursividad de ejecución se distingue:

a) Recursividad Directa: *Consiste en que un método se llama a si mismo desde uno* ***(****recursividad* ***simple)*** *ó varios puntos del código* ***(****recursividad* ***múltiple).***

b) Recursividad Indirecta o Mutúa: *Consiste en que dos métodos se llaman entre si, es decir, mutuamente.*

Para poder implementar un método de forma recursiva, es necesario que se cumplan las siguientes condiciones:

a) Que pueda definirse en términos de si mismo.

b) Que exista un criterio de finalización, llamado ***Caso Base,*** *llegado el cual no se aplique de nuevo la llamada recursiva.*

c) Que en cada llamada recursiva se este más cerca de que se cumpla el Caso Base.

d) Que se resuelva el problema en un tiempo limitado o finito.

Un ejemplo claro de método recursivo es ***el calculo del factorial*** *de un numero entero* ***N,*** *que puede definirse de* ***forma recursiva*** *o de* ***forma iterativa.***

Solución Iterativa ➜ Si X >= 0 X! = 1 * 2 * 3 * 4 * ... * X

Solución Recursiva ➜ Si X = 0 X! = 1
Si X > 0 X! = X * (X-1)!

Solución Iterativa

```
public static int factorial(int x)
{
        int fact = 1;
        for (int i=1; i<= x; i++)
        {
             fact = fact * i;
1));
        }
        return  fact;
}
```

Solución Recursiva

```
public static int factorial(int x)
{
        if (x == 0)
            return 1;
        else
            return  (x * factorial(x –

}
```

EJERCICIOS

1.- Escribir un programa que calcule todos los factoriales del 1 hasta el valor entero N que se introduce por teclado, el valor de N es mayor de cero.

Ejemplo de una ejecución:

```
Introduce un numero: -4
Introduce un numero: 0
Introduce un numero: 4

1! = 1
2! = 1 * 2 = 2
3! = 1 * 2 * 3 = 6
4! = 1 * 2 * 3 * 4 = 24
```

2.-Diseñar un método que calcule la potencia de un numero real elevado a un entero.

3.- Escribir un método que calcule la suma de los N primeros números naturales.

N = **5** → **5 + 4 + 3 + 2 + 1 = 15**

Suma (N) → N + Suma(N-1)

4.- Escribir un método que visualice los N primeros números naturales del 1 al N.

N = **4** → visualiza los numeros **1**
2
3
4

5.- Escribir un método que visualice los N primeros números naturales del N al 1.

N = **4** → visualiza los numeros **4**
3

2 **1**

6.- Escribir un método que visualice los dígitos de un número natural N al revés

N = **7815** ➜ visualizar el numero **5187**

7.- Escribir un método que visualice los dígitos de un número natural N, uno por línea.

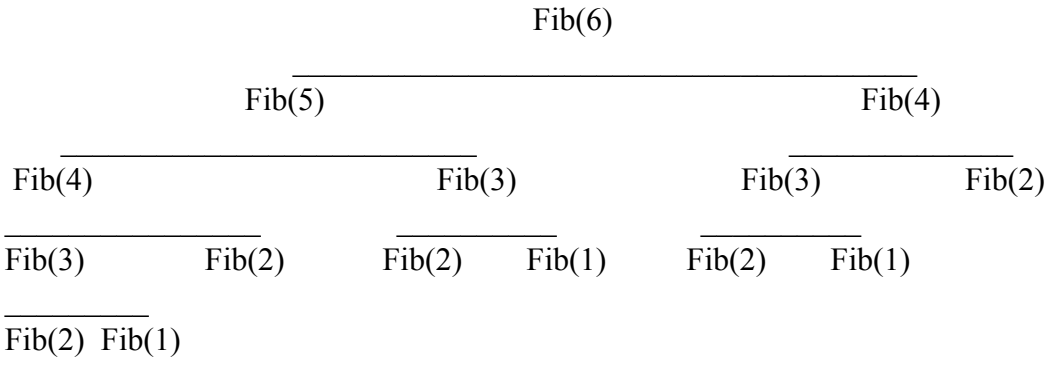
N = **7815** ➜ visualizar el numero **7** **8** **1** **5**

8.- Realizar una función recursiva que calcule el término N de la serie de Fibonacci:

1, 1. 2, 3, 5, 8, 13, 21, ... ➜ Fibonacci(N) = Fibonacc (N-1) + Fibonacc (N-2)

caso base: (N < 2) ➜ Fibonacci = n
caso recursivo: (N >= 2) ➜ Fibonacci(N)= Fibonacci(N-1) + Fibonacci(N-2)

La secuencia de llamadas para calcular el termino 6 de la serie de Fibonacci será:



Fib(5) ➜ Se llama desde el propio método 1 vez
Fib(4)➜ Se llama desde el propio método 2 veces
Fib(3)➜ Se llama desde el propio método 3 veces
Fib(2)➜ Se llama desde el propio método 5 veces
Fib(1)➜ Se llama desde el propio método 3 veces

9.- Programar un algoritmo recursivo que permita hacer la división por restas sucesivas.

10.- Programar un algoritmo recursivo que permita invertir un número. Ejemplo: Entrada: 123 Salida: 321

11.- Programar un algoritmo recursivo que permita sumar los dígitos de un número. Ejemplo: Entrada: 123 Resultado:6

12.- Programar un algoritmo recursivo que calcule el Maximo común divisor de dos números.

13.- Diseñar un método para calcular el número combinatorio de “m” elementos tomados de “n” en “n”.

Este problema deberéis resolverlo aplicando diferentes métodos:

a.- Aplicando la formula que hace uso del calculo del factorial

Cm,n = m! / n!(m-n)!

b.- Aplicando la *formula iterativa*: para i ➔ de 1 a n

Cm,n = 1 * (m-1+1) div 1 * (m-2+1) div 2* ... *(m-i+1)div I

c.- Aplicando la ley de recurrencia utilizando la *recursividad Directa Simple:*

caso base:	**si n = 0**	**Cm,n = 1**
caso general:	**si n > 0**	**Cm,n = Cm,n-1 * (m-n+1) div n**

d.- Aplicando la ley de recurrencia utilizando la *recursividad Directa Múltiple*

casos base:	**si n = 0**	**Cm,n = 1**
	si n = m	**Cm,n = 1**
	si n > m	**Cm,n = 0**
caso general:	**si m > n > 0**	**Cm,n = Cm-1,n + Cm-1,n-1**

La llamada al subprograma NumeroCombinatorio para todos los casos podría ser:

```
public static void main(String[] args)
{
        System.out.println("calculo del numero combinatorio");
        System.out.println("Introduce el valor de M");
        int m = Utilidades.LeerEntero();
        System.out.println("Introduce el valor de N");
        int n = Utilidades.LeerEntero();
        int combinaciones = numeroCombinatorio(m, n);
        System.out.print("El numero combinatorio"+m+"sobre"+n);
        System.out.println(" es igual a "+combinaciones);
}
```