**Decimal a binario**

import java.util.\*;

/\*\*

  \* Método recursivo que dado un número entero en base decimal

 \* muestre su equivalente en binario

 \*/

public class Recursividad {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n;

        do {

            System.out.print("Introduzca numero >0: ");

            n = sc.nextInt();

        } while (n < 0);

        System.out.println();

        System.out.print("Binario: ");

        decBin(n);

        System.out.println();

    }

    public static void decBin(int n) {

        if (n < 2) {

            System.out.print(n);

            return;

        } else {

            decBin(n / 2);

            System.out.print(n % 2);

            return;

        }

    }

}

**Factorial**

|  |  |
| --- | --- |
| 2  3 | int factorial(int n) {        return (n == 0) ? 1 : n \* factorial(n - 1);  } |

**Cociente**

public static int cociente(int a, int b) {  
        if (a < b)  
            return 0;  
        else  
            return 1 + cociente(a - b, b);  
    }

**Ordenar arreglo**

void ordenar (int [] v, int cant) {

if (cant > 1) {

for (int f = 0 ; f < cant - 1 ; f++)

if (v [f] > v [f + 1]) {

int aux = v [f];

v [f] = v [f + 1];

v [f + 1] = aux;

}

ordenar (v, cant - 1);

}

}

**Planteamiento**:  
  
**Ejercicio 2**. Programar un algoritmo recursivo que calcule un número de la serie fibonacci.  
  
**Solución**:

Código

1. **int** fibonaci(**int** n){
2. **if**(n==1 || n==2) **return** 1;
3. **else** **return** fibonaci(n-1)+fibonaci(n-2);
4. }

**Planteamiento**:  
  
**Ejercicio 3**. Programar un algoritmo recursivo que permita hacer la división por restas sucesivas.  
  
**Solución**:

Código

1. **int** division (**int** a, **int** b)
2. {
3. **if**(b > a) **return** 0;
4. **else**
5. **return** division(a-b, b) + 1;
6. }

**Planteamiento**:  
  
**Ejercicio 4**. Programar un algoritmo recursivo que permita invertir un número. **Ejemplo**: Entrada: *123*Salida: *321*  
  
**Solución**:

Código

1. **int** invertir (**int** n)
2. {
3. **if** (n < 10)         *//caso base*
4. **return** n;
5. **else**
6. **return** (n % 10) + invertir (n / 10) \* 10;
7. }

**Planteamiento**:  
  
**Ejercicio 5**. Programar un algoritmo recursivo que permita sumar los dígitos de un número. **Ejemplo**: Entrada: *123*Resultado:*6*  
  
**Solución**:

Código

1. **int** sumar\_dig (**int** n)
2. {
3. **if** (n == 0)      *//caso base*
4. **return** n;
5. **else**
6. **return** sumar\_dig (n / 10) + (n % 10);
7. }

**Planteamiento**:  
  
**Ejercicio 6**. Programar un algoritmo recursivo que permita hacer una multiplicación, utilizando el método Ruso. Para mas informacion: [aqui](http://es.wikipedia.org/wiki/Multiplicaci%C3%B3n_por_duplicaci%C3%B3n" \t "_blank).  
  
**Solución**:

Código

1. **int** mult\_rusa(**int** A, **int** B)
2. {
3. **if**(A==1){
4. **return** (B);
5. }
6. **if**(A%2!=0){
7. **return**(B+mult\_rusa( A/2 , B\*2));
8. }
9. **else**{
10. **return**(mult\_rusa( A/2 , B\*2));
11. }
12. }

**Planteamiento**:  
  
**Ejercicio 7**. Programar un algoritmo recursivo que permita sumar los elementos de un vector.  
  
**Solución**:

Código

1. **int** suma\_vec(**int** v [], **int** n)
2. {
3. **if** (n == 0)
4. **return** v [n];
5. **else**
6. **return** suma\_vec(v, n - 1) + v [n];
7. }

**Planteamiento**:  
  
**Ejercicio 8**. Programar un algoritmo recursivo que permita multiplicar los elementos de un vector.  
  
**Solución**:

Código

1. **int** multiplicar (**int** vec [], **int** tam)
2. {
3. **if** (tam == 0)
4. **return** (vec [0]);
5. **return** (vec [tam] \* multiplicar (vec, tam - 1));
6. }

**Planteamiento**:  
  
**Ejercicio 9**. Programar un algoritmo recursivo que calcule el Maximo comun divisor de dos números.  
  
**Solución**:

Código

1. **int** sacar\_mcd(**int** a, **int** b) {
2. **if**(b==0)
3. **return** a;
4. **else**
5. **return** sacar\_mcd(b, a % b);
6. }

**Planteamiento**:  
  
**Ejercicio 10**. Programar un algoritmo recursivo que determine si un número es positivo.  
  
**Solución**:

Código

1. **public** **boolean** positivo(**int** n){
2. **if**(n>0) **return** **true**;
3. **else** **return** negativo(n);
4. }
6. **public** **boolean** negativo(**int** n){
7. **if**(n<0) **return** **false**;
8. **else** **return**  positivo(n);
9. }

**Planteamiento**:  
  
**Ejercicio 11**. Programar un algoritmo recursivo que determine si un número es impar utilizando recursividad cruzada.  
  
**Solución**:

Código

1. **public** **boolean** par(**int** n){
2. **if**(n==0) **return** **true**;
3. **else** **return** impar(n-1);
4. }
6. **public** **boolean** impar(**int** n){
7. **if**(n==0) **return** **false**;
8. **else** **return** par(n-1);
9. }

**Planteamiento**:  
  
**Ejercicio 12**. Programar un algoritmo recursivo que permita sumar los elementos de una matriz.  
  
**Solución**:

Código

1. **int** suma (**int** fila, **int** col, **int** orden, **int** mat [] [])
2. {
3. **if** (fila == 0 && col == 0)
4. **return** mat [0] [0];
5. **else**
6. **if** (col < 0)
7. **return** suma (fila - 1, orden, orden, mat);
8. **else**
9. **return** mat [fila] [col] + suma (fila, col - 1, orden, mat);
10. }

public static int SumaPares2(int n) {  
if (n == 0) //Caso base n=0  
return 0;  
else  
if (n % 2 == 0) //si el residuo de n/2 es 0, el numero el par  
return n + SumaPares(n - 1); //Cuando es par acumula n  
else  
return SumaPares(n - 1); //Cuando es impar no acumula  
}

