

# Programación

## UD 5: Programación modular

---

1.- ¿Qué es la programación estructurada?  
¿Qué es la programación modular?

# 1.- ¿Qué es la programación estructurada?

---

“La **programación estructurada** es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas:

secuencia,  
selección (if y switch) e  
iteración (bucles for y while);

asimismo, se considera innecesario y contraproducente el uso de la instrucción de transferencia incondicional (GOTO), que podría conducir a código espagueti, mucho más difícil de seguir y de mantener, y fuente de numerosos errores de programación.”

# 1.- ¿Qué es la programación modular?

---

“La **programación modular** es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Se presenta históricamente como una **evolución de la programación estructurada** para solucionar problemas de programación más grandes y complejos de lo que esta puede resolver.

Al aplicar la programación modular, **un problema complejo debe ser dividido en varios subproblemas más simples**, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Esta técnica se llama refinamiento sucesivo, **divide y vencerás** o análisis descendente (Top-Down).”

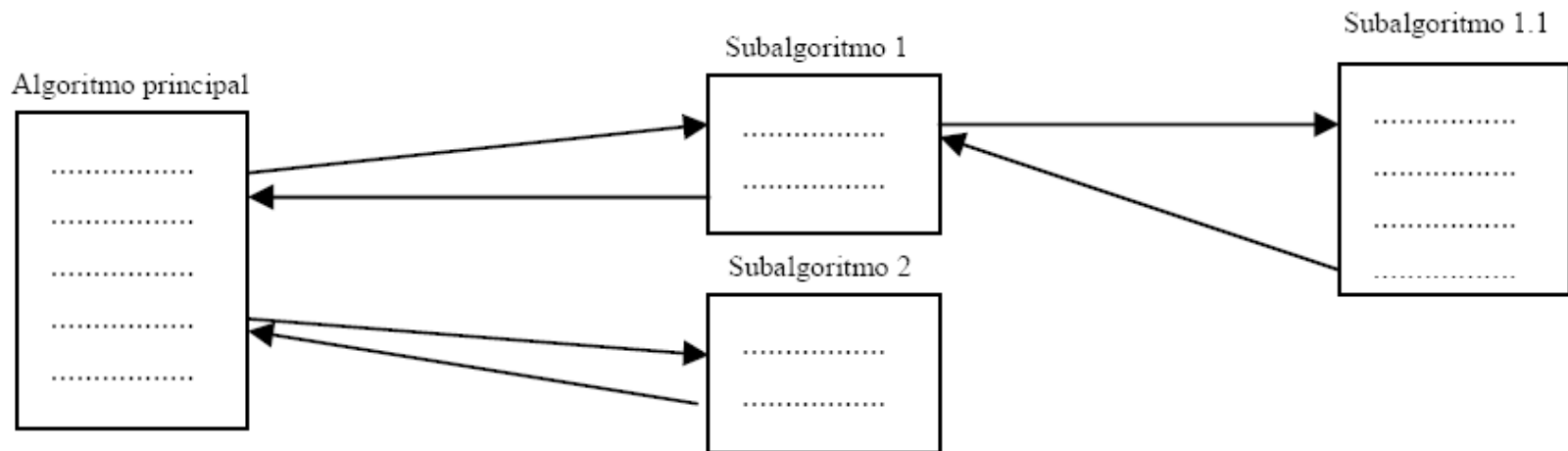
---

## 2.- Funciones (métodos)

## 2.- Funciones

Cuando el algoritmo principal hace una llamada al subalgoritmo (es decir, lo invoca), se empiezan a ejecutar las instrucciones del subalgoritmo. Cuando éste termina, devuelve los datos de salida al algoritmo principal, y la ejecución continúa por la instrucción siguiente a la de invocación.

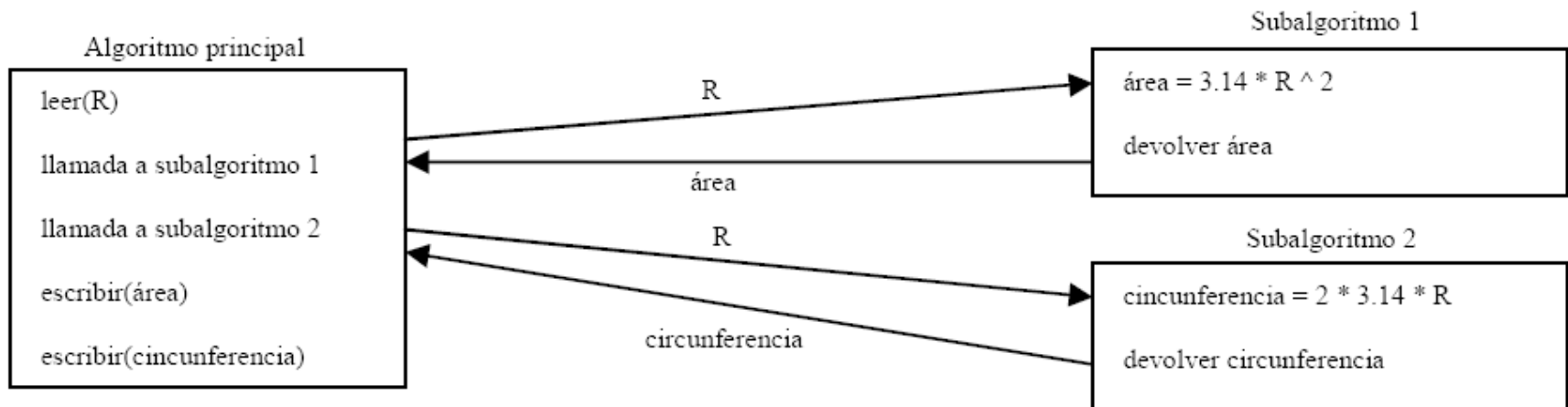
También se dice que el subalgoritmo devuelve el control al algoritmo principal, ya que éste toma de nuevo el control del flujo de instrucciones después de habérselo cedido temporalmente al subalgoritmo.



## 2.- Funciones

### Ejemplo:

Algoritmo que pide el radio de una circunferencia al usuario y devuelve su área y perímetro.



## 2.- Funciones

---

La estructura general de una **función Java** es la siguiente:

```
tipoDevuelto nombreMetodo([lista parámetros])
{
    /*
    * Bloque de instrucciones
    */
    [return valor;]
}
```

Los elementos que aparecen entre corchetes son opcionales.

**tipoDevuelto**: indica el tipo del valor que devuelve el método. En Java es imprescindible que en la declaración de un método, se indique el tipo de dato que ha de devolver. El dato se devuelve mediante la instrucción `return`. Si el método no devuelve ningún valor este tipo será `void` (procedimiento).

**nombreMetodo**: es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

**lista de parámetros** (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas. Estos parámetros son los datos de entrada que recibe el método para operar con ellos. Un método puede recibir cero o más argumentos. Se debe especificar para cada argumento su tipo. Los paréntesis son obligatorios aunque estén vacíos.



## 2.- Funciones

---

Programa principal:

```
public static void main(String[] args) {  
    Scanner s= new Scanner(System.in);  
    int a, b;  
    System.out.println("Introduzca un numero:");  
    a = s.nextInt();  
    System.out.println("Introduzca otro numero:");  
    b = s.nextInt();  
    sumar(a,b);  
}
```

Función que hace la suma y no devuelve ningún dato (tipo void):

```
public static void sumar(int a, int b) {  
    int resultado_local=0;  
    resultado_local = a + b;  
    System.out.println("El resultado de la suma es: " + resultado_local);  
}
```


## 2.- Funciones

---

Programa principal:

```
public static void main(String[] args) {  
    Scanner s= new Scanner(System.in);  
    int a, b;  
    int resultado=0;  
    System.out.println("Introduzca un numero:");  
    a = s.nextInt();  
    System.out.println("Introduzca otro numero:");  
    b = s.nextInt();  
    resultado = sumar(a,b);  
    System.out.println("El resultado de la suma es: " + resultado);  
}
```

Función que hace la suma y el resultado lo devuelve al programa principal (devuelve int):



```
public static int sumar(int a, int b) {  
    int resultado_local=0;  
    resultado_local = a + b;  
    return resultado_local;  
}
```

---

## 3.- Paso de parámetros

# 3.- Paso de parámetros

---

En Java, los parámetros se pasan siempre **por copia**, es decir, las variables del programa principal se acaban copiando en la lista de parámetros de la función. Eso significa que el valor de los parámetros se modifica EN LA FUNCION y NO EN EL PROGRAMA PRINCIPAL.

**Repetimos: SÍ QUE SE MODIFICA EN LA FUNCIÓN PERO NO EN EL PRINCIPAL**

```
public class ejer_2 {  
  
    public static void main(String[] args) {  
  
        int a = 5;  
        incrementa(a);  
        System.out.println(a); //¿Qué imprimirá?  
  
    }  
  
    public static void incrementa(int x) {  
        //Cuando comienza la función, x vale 5  
        x = x + 1;  
        //Cuando finaliza la función,. x vale 6  
    }  
}
```

# 3.- Paso de parámetros

Si queremos que se modifique esa variable en el principal, hemos de DEVOLVER ese valor con la clausula RETURN seguido del nombre de la variable.

**La variable sobre la que devolvemos el valor en el principal (a) tendrá que ser del mismo tipo que la variable que devuelve la función (x).**

```
public class ejer_2 {  
    public static void main(String[] args) {  
        int a = 5;  
        → a = incrementa(a);  
        System.out.println(a); //¿Qué imprimirá?  
    }  
    public static int incrementa(int x) {  
        //Cuando comienza la función, x vale 5  
        x = x + 1;  
        //Cuando finaliza la función,. x vale 6  
        return x; ←  
    }  
}
```

# 3.- Paso de parámetros

---

Como hemos dicho, TODAS las variables se copian a la lista de argumentos de la función, por lo que si queremos que se modifique en el programa principal, hemos de hacer uso de la cláusula RETURN dentro de la función.

Esto NO SUCEDE con los arrays o vectores, ya que cuando pasamos el dato a la función, la variable **SE MODIFICA DIRECTAMENTE Y NO NECESITAMOS HACER UN RETURN**. Por ejemplo:

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

    multiplicapor5(vector);

    //¿Qué imprimirá el siguiente bucle?
    for (int i=0; i < vector.length; i++) {
        System.out.print(vector[i] + " ");
    }
}

public static void multiplicapor5(int[] v) {
    //Dentro de "v" ya tengo el valor de "vector".
    for (int i=0; i < v.length; i++) {
        v[i] = v[i] * 5;
    }
}
```

# 3.- Paso de parámetros

¿Y porqué sucede esto?...

Básicamente porque cuando pasamos un **tipo de dato simple** (un entero, un char..) copiamos la variable, por lo que cualquier modificación NO afecta a la variable original.

Sin embargo, cuando pasamos un **tipo de dato complejo** (un vector), no estamos copiando la variable, sino la dirección de la misma, por lo que cualquier modificación sí que afecta a la variable original.

```
public static void main(String[] args) {  
    int[] vector = {1,2,3};  
  
    multiplicar5(vector); le paso 0x1AB  
  
    //¿Qué imprimirá el siguiente bucle?  
    for (int i=0; i < vector.length; i++) {  
        System.out.print(vector[i] + " ");  
    }  
}
```

0x1AB

vector = {1,2,3}

```
public static void multiplicar5(int[] v) { Modifico lo que tenga 0x1AB  
    //Dentro de "v" ya tengo el valor de "vector".  
    for (int i=0; i < v.length; i++) {  
        v[i] = v[i] * 5;  
    }  
}
```

---

## 4.- Sobrecarga de funciones



# 4.- Sobrecarga de funciones

---

En java, una clase puede contener dos métodos con el mismo nombre si:

- ✓ Tienen diferente número de argumentos.
- ✓ Si el tipo de los argumentos es distinto, aunque tenga el mismo número de ellos.

# 4.- Sobrecarga de funciones

---

Para entender esta sencilla propiedad de java, lo mejor es contemplar el siguiente ejemplo de dos métodos sobrecargados.

```
public void unMetodo() {  
    //Código del método  
}  
  
public void unMetodo(int numero) {  
    //Código del método  
}
```

En este ejemplo, ambos métodos tienen el mismo nombre y número distinto de argumentos. Cuando se llame al método `unMetodo()`, se ejecutará el primero, y cuando se llame de esta manera `unMetodo(5)`, se ejecutará el segundo.

# 4.- Sobrecarga de funciones

---

Veamos otro ejemplo:

```
public void otroMetodo(int numeroEntero) {  
    //Código del método  
}  
  
public void otroMetodo(float numeroReal) {  
    //Código del método  
}
```

Como vemos, ambos métodos tienen el mismo nombre y el mismo número de argumentos, en cambio el argumento que reciben es de distinto tipo de modo que cuando se llame al método `otroMetodo(15)`, se ejecutará el primero, y cuando se llame de esta manera `otroMetodo(15.5)`, se ejecutará el segundo.

# Programación

## UD 5: Programación modular Recursividad

# 1.- Definición de un método recursivo

---

Una función recursiva es aquella que se llama a sí misma. Se considera una alternativa a la versión iterativa (con bucles).

La característica principal de la recursividad es que siempre existe un medio de salir de la definición (**caso base o salida**), y la segunda condición (**caso recursivo**) es propiamente donde se llama a sí misma.

Una función recursiva simple en Java es:

```
public void infinito()  
{  
    infinito();  
}
```

## 2.- Resolver un problema recursivo

---

### Caso base

Es el caso más simple de una función recursiva, y **simplemente devuelve un resultado** (el caso base se puede considerar como una salida no recursiva).

### Caso general

Relaciona el resultado del algoritmo con resultados de casos más simples. Dado que cada caso de problema aparenta o se ve similar al problema original, **la función llama una copia nueva de si misma**, para que empiece a trabajar sobre el problema más pequeño y esto se conoce como una llamada recursiva y también se llama el paso de recursión.

## 2.- Resolver un problema recursivo

Imaginemos que quisiéramos emular el funcionamiento de la cuenta atrás de una bomba. El método recursivo sería el siguiente:

```
public class recur1 {  
    public static void main(String[] args) {  
        cuenta_atras(5);  
    }  
  
    public static void cuenta_atras(int n) {  
        System.out.println(n);  
        if (n == 0) {  
            System.out.println("BOOM!");  
        }else {  
            cuenta_atras(n - 1);  
        }  
    }  
}
```



CASO BASE



CASO RECURSIVO

## 2.- Resolver un problema recursivo

Imaginemos que ahora queremos calcular el factorial de un número de manera recursiva. Por ejemplo si se quiere calcular el factorial de 5, se tendría:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

$$5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$$

$$5! = 5 \cdot 4!$$

```
public static void main(String[] args) {  
    int res = fact(5);  
    System.out.println(res);  
}
```

```
public static int fact(int n) {  
    if (n > 0) {  
        return n * fact(n - 1);  
    }  
    else {  
        return 1;  
    }  
}
```

CASO RECURSIVO

CASO BASE



## 2.- Resolver un problema recursivo

### Factorial de un número

