Métodos, funciones y procedimientos en Java.

INDICE

- Concepto.
- Funciones, métodos o procedimientos.
- Ejemplo
- Cohesión y acoplamiento. Niveles.
- Ejercicios

CONCEPTO

- Los métodos son una herramienta indispensable para programar.
- Ya que nos permiten automatizar tareas que requiramos con frecuencia, uso de parámetros y hacer uso de otras librerías (funciones matemáticas, aritméticas, de archivos, de fechas, etc).
- Aprender a crear métodos y usarlos correctamente es de gran importancia, separar nuestro código en módulos (según su utilidad o tarea que realiza, ejemplo: abrir, cerrar, cargar, ...).
- Lo más adecuado es tener un método asociado a una tarea o funcionalidad básica, es decir, descomponer un problema en subproblemas y solucionarlos en pequeñas funciones.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Funciones**:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, *usualmente reciben parámetros*, cuyos valores utilizan para efectuar operaciones y adicionalmente *devuelven/retornan* un valor. En otras palabras, una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y devuelve un valor usando la instrucción *return*. Si no devuelve algo, entonces le llamamos **procedimiento**.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Métodos**:

Los métodos realizan las mismas tareas que las funciones, es decir, son funcionalmente idénticos. Su diferencia radica en la manera en que hacemos uso de uno u otro, es decir, un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo un método está **siempre** asociado a un **objeto**.

¡ojO! En Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función/procedimiento, esta debe ser **static**, para que no requiera de un objeto para ser llamada.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Procedimientos**:

Los procedimientos son un conjunto de instrucciones que se ejecutan sin retornar ningún valor, pueden recibir o no argumentos. En Java, un procedimiento es un método cuyo tipo de retorno es void, que no nos obliga a utilizar una sentencia return.

EJEMPLO

[**acceso**] [**modificador**] **tipo** nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento],...)

```
{
        /*
                Bloque de instrucciones
        */

        return valor;
}
```

1) **Modificador de acceso** => Puede ser public, private o protected. En caso de no indicarlo, se asume el modificador de acceso por defecto.
2) **Modificador** => Puede ser final o static (o ambas), también es opcional.
3) Un método o función siempre **retorna** algo, por lo tanto es obligatorio declararle un **tipo** (int, boolean,...). Si no es función, pondremos **void**.
4) **Nombre** de dicha función, para poder identificarla y llamarla (invocarla) durante la ejecución.
5) En el interior del paréntesis, indicaremos el conjunto de parámetros.

Hasta este punto es lo que denominaremos **cabecera o prototipo** del método.

A continuación, entre llaves, definiremos el **cuerpo** del método, conjunto de sentencias a ejecuta.

COHESIÓN Y ACOPLAMIENTO

- **Cohesión**:
  1) Se puede definir como la relación entre los componentes de un ente/conjunto/unidad.
  2) Se refiere a que los módulos tienen una sola responsabilidad/función

- **Acoplamiento**:
  1) Es la manera en la que se relacionan los componentes entre ellos.
  2) Relación y grado de dependencia de los módulos entre ellos.

COHESIÓN NIVELES : FUERTE

- **Funcional**
  - Un módulo realiza una única acción.
- **Secuencial**
  - Las acciones dentro de un módulo han de realizarse en un orden concreto, debido a los datos que requiere.
- **De comunicación**
  - El módulo contiene un conjunto de operaciones que son realizadas sobre los mismos datos.
- **Temporal**
  - Las operaciones dentro del módulo tienen que realizarse al mismo tiempo, por ejemplo, inicialización.

COHESIÓN NIVELES : DÉBIL

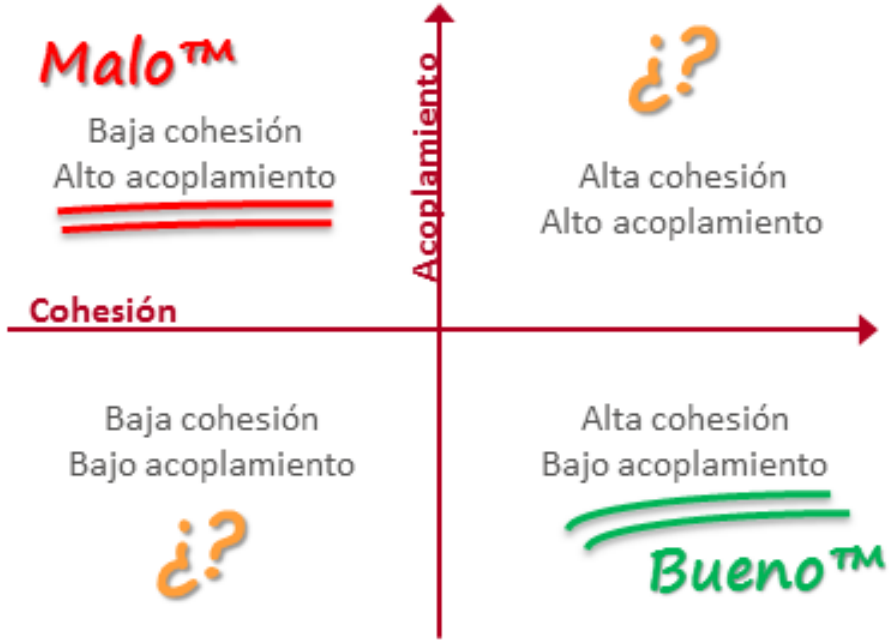
- **Procedural**
  - ✓ Las operaciones del módulo se realizan en un orden concreto aunque sean independientes.
- **Lógica**
  - ✓ El módulo depende un parámetro para la ejecución de las operaciones, es decir, el nexo de unión entre es las operaciones del módulo es el flujo de ejecución.
- **Coincidental**
  - ✓ No existe relación observable entre las operaciones realizadas dentro de un módulo.

ACOPLAMIENTO NIVELES : DE MEJOR A PEOR

- **De datos (normal)**
  - Todo lo que se comparte entre módulos es especificado en una lista de parámetros por el módulo invocado.
- **De control**
  - Un módulo le pasa datos a otro para indicarle que hacer, el primer módulo conoce el funcionamiento interno del segundo.
- **Externo**
  - Cuando dos o más módulos utilizan los mismo datos globales.
  - Generalmente, este tipo de acoplamiento, no es recomendable.
- **Patológico o de contenido**
  - Cuando un módulo utiliza el código de otro o altera sus datos locales.

COHESIÓN Y ACOPLAMIENTO

- Objetivo
  :



- Beneficios:

Al obtener una alta cohesión y bajo acoplamiento, obtenemos, los siguientes beneficios:

1. Aumenta la **legibilidad** del código.
2. **Mantenibilidad**, los cambios a realizar serán mínimos (módulo específico).
3. **Reusabilidad**, si los módulos están bien definidos, podrán ser reutilizados en otros “programas”.
4. La realización de **pruebas** estará bien acotado y serán mínimas por módulo utilizado.

EJERCICIOS

1) Crea una función que reciba dos números reales y devuelva su suma.
2) Crea una aplicación que nos pida un número y, mediante un procedimiento, nos indique si es este es primo o no.
3) Crea una función que nos calcule el factorial de un número.
4) Crea una procedimiento que indique la posición de la primera ocurrencia de un dígito dentro de un número. Si no se encuentra, devuelve -1.
5) Crea una función que le de la vuelta a un número.
6) Crea una función que dado un número decimal, devuelva su equivalente en binario.
7) Crea una función que dado un número decimal, devuelva su equivalente en octal.
8) Crea una función que dado un número decimal, devuelva su equivalente en hexadecimal.
9) Crea un procedimiento que nos dado un número N, nos devuelva los primos anteriores a este.
10) Crea un procedimiento que lea un número N y calcule el factorial de los números desde 0 hasta N.
11) Crea una aplicación que nos permita calcular las operaciones básicas entre dos números mediante un método.
12) Crea una aplicación que nos permita convertir un número en base decimal a binario, octal o hexadecimal. Esto lo realizará mediante un método al que le pasaremos el número como parámetro, devolverá un String con el número convertido al formato específico, es decir, posteriormente habrá que indicar el formato requerido.