Repaso de Los Métodos en JAVA.

Un método es una estructura del lenguaje Java que nos sirve para encapsular cierta funcionalidad, la cual podamos llamar desde diferentes sitios y así no tener que repetir el código.

Los métodos definen las acciones/comportamientos de los objetos de una clase dada.
Para la creación de un método en Java debemos conocer la estructura del mismo:

```
tipo_acceso tipo_dinamico_o_no tipo_dato nombre_metodo(tipo_parametro parametro)
```

Un método generalmente usa toda esa estructura solo exceptuando la declaración de si es dinámico u estático. La primera parte de creación de un método se refiere a el tipo de acceso que puede ser:

- **protected**, acceso protegido de datos
- **private**, acceso solo de modo interno de la clase
- **public**, acceso desde una instancia externa de la clase

La segunda parte se refiere a el uso del método Java, si es estático lo cual significa que el método sería accesible desde fuera de la clase sin necesidad de instanciar la clase.

- **static**, el acceso al método es estático.

El tipo de dato es dependiente de lo que se desea como resultado del método como puede ser por ejemplo **void** si nuestro método no tiene salida alguna, o un tipo de dato específico como puede ser **double** o **int** si es una salida de tipo numérico.

El nombre de método de preferencia debe ser escrito en *notación camelCase* por ejm: (la notación camel case detalla que se debe usar en los métodos con nombres compuestos siempre la primera letra de la segunda palabra en mayúscula) Siempre comienza en minúscula.

Para la creación del método no en todos los casos es necesario argumentos pero si deseamos usar algún argumento, cada argumento deberá tener su tipo de dato y nombre de argumento.

```
1. public void miMetodo(int argumento1){
2.  //funcionamiento debe ser escrito aqui....
3. //Este es el cuerpo del metodo!!!!
4.
5. }
```

El cuerpo del método va delimitado por {}

Bueno ahora solo nos queda ver un ejemplo de cómo crear el método con Java. Para ello vamos a definir un método que nos sume dos números con Java. De esta forma, cada vez que queramos sumar dos números nos bastará con llamar a este método.

```
1. public static int sumarNumeros (int numero1, int numero2) {
2.  return numero1 + numero2;
3. }
```

En este método podemos ver que *el tipo de acceso es público*, cabe detallar que este tipo de método también es de acceso estático por tanto no necesitamos instanciar un objeto de la clase a la cual pertenece este método. También tomando en cuenta el tipo de dato a devolver del método se puede decir que trabaja con entradas de tipo entero tanto como salidas de tipo entero tal como detalla su signatura.

Para poder ver como este ejemplo funcionaria en código lo probamos:

```
1. System.out.println("Programa de Suma de números iniciando");
2. //iniciamos sumando
3. int sumando1=4234;
4. System.out.println("Sumando 1: "+sumando1);
5. //iniciamos sumando 2
```

```
6. int sumando2=64782;
7. System.out.println("Sumando 2: "+sumando1);
8. // obtenemos el resultado de la suma de los dos sumandos
9. int resultado= sumarNumeros(sumando1, sumando2);
10.       System.out.println("Resultado: "+resultado);
11.       //fin de ejecucion
12.       System.out.println("Programa de Suma de números finalizando");
13.
```

Sobrecarga de Métodos:

Lenguajes como Java permiten que existan distintos métodos con el mismo nombre siempre y cuando su signatura no sea idéntica (algo que se conoce con el nombre de *sobrecarga*)

Ejemplo

`System.out.prinln(…);`

- `System.out.println()`
- `System.out.println(boolean)`
- `System.out.println(char)`
- `System.out.println(char[])`
- `System.out.println(double)`
- `System.out.println(float)`
- `System.out.println(int)`
- `System.out.println(long)`
- `System.out.println(Object)`
- `System.out.println(String)`

No es válido definir dos métodos con el mismo nombre que difieran únicamente por el tipo del valor que devuelven.

Como se usan los métodos?

Para enviarle un mensaje a un objeto,
invocamos (llamamos a) uno de sus métodos:

> La llamada a un método de un objeto
> le indica al objeto que delegamos en él
> para que realice una operación de la que es reponsable.

- A partir de una referencia a un objeto,
  podermos llamar a uno de sus métodos con el **operador .**
- Tras el nombre del método, entre paréntesis,
  se han de indicar sus parámetros (si es que los tiene).
- El método podemos usarlo cuantas veces queramos.

  MUY IMPORTANTE: De esta forma, evitamos la existencia de código duplicado en nuestros programas.

Ejemplo

```
Cuenta cuenta = new Cuenta();

cuenta.mostrarMovimientos();
```

Obviamente, el objeto debe existir antes de que podamos invocar uno de sus métodos. Si no fuese así, en la ejecución del programa obtendríamos el siguiente error:

```
java.lang.NullPointerException
    at ...
```

al no apuntar la referencia a ningún objeto (`null` en Java).

Ejemplo de ejecución paso a paso

Cuando se invoca un método, el ordenador pasa a ejecutar las sentencias definidas en el cuerpo del método:

```
public class Mensajes
{
  public static void main (String[] args)
  {
    mostrarMensaje("Bienvenida");
    // ...
    mostrarMensaje("Despedida");
  }

  private static void mostrarMensaje (String mensaje)
  {
    System.out.println("*** " + mensaje + " ***");
  }
}
```

Al ejecutar el programa (con `java Mensajes`):

1. Comienza la ejecución de la aplicación, con la primera sentencia especificada en el cuerpo del método `main`.
2. Desde `main`, se invoca `mostrarMensaje` con *"Bienvenida"* como parámetro.
3. El método `mostrarMensaje` muestra el mensaje de bienvenida decorado y termina su ejecución.
4. Se vuelve al punto donde estábamos en `main` y se continúa la ejecución de este método.
5. Justo antes de terminar, volvemos a llamar a `mostrarMensaje` para mostrar un mensaje de despedida.
6. `mostrarMensaje` se vuelve a ejecutar, esta vez con *"Despedida"* como parámetro, por lo que esta vez se muestra en pantalla un mensaje decorado de despedida.
7. Se termina la ejecución de `mostrarMensaje` y se vuelve al método desde donde se hizo la llamada (`main`).
8. Se termina la ejecución del método `main` y finaliza la ejecución de nuestra aplicación.

Los Parámetros:

Un método puede tener parámetros:

- A través de los parámetros se especifican los datos de entrada que requiere el método para realizar su tarea.
- Los parámetros definidos en la cabecera del método se denominan **parámetros formales**.
- Para cada parámetro, hemos de especificar tanto su tipo como un identificador que nos permita acceder a su valor actual en la implementación del método.
- Cuando un método tiene varios parámetros, los distintos parámetros se separan por comas en la cabecera del método.

Cuando se efectúa la llamada a un método, los valores indicados como parámetros actuales se asignan a sus parámetros formales.

- En la implementación del método, podemos utilizar entonces los parámetros del método como si fuesen variables normales (y de esta forma acceder a los valores concretos con los que se realiza cada llamada al método).
- Obviamente, el número y tipo de los parámetros indicados en la llamada al método ha de coincidir con el número y tipo de los parámetros especificados en la definición del método.

Cuando un método devuelve un resultado, la implementación del método debe terminar con una sentencia `return`:

```
return expresión;
```

Como es lógico, el tipo de la *expresión* debe coincidir con el tipo del valor devuelto por el método, tal como éste se haya definido en la cabecera del método.

Ejemplo

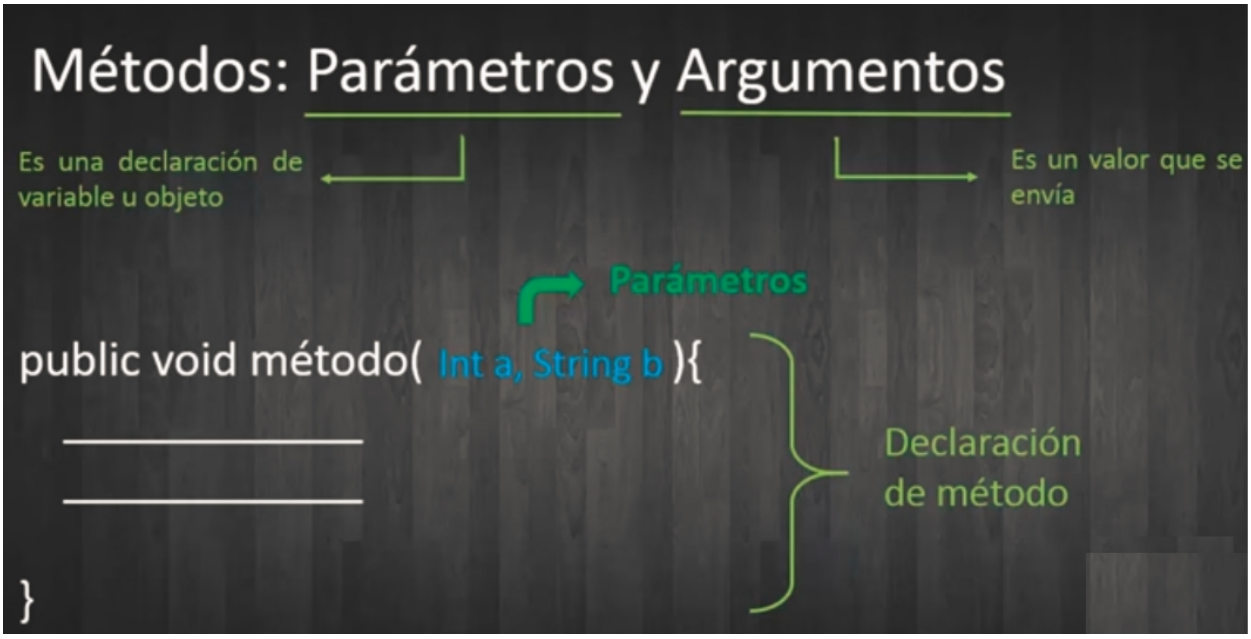
```
public static float media (float n1, float n2)
{
  return (n1+n2)/2;
}

public static void main (String[] args)
{
  float resultado = media (1,2);

  System.out.println("Media = " + resultado);
}
```

- El compilador de Java comprueba que exista una sentencia `return` al final de un método que deba devolver un valor.
  Si no es así, nos dará el error

  ```
  Missing return statement
  ```

- El compilador también detecta si hay algo después de la sentencia `return` (un error porque la sentencia return finaliza la ejecución de un método y nunca se ejecuta lo que haya después):

  ```
  Unreachable statement
  ```



Constructores:

Los constructores son métodos especiales que sirven para inicializar el estado de un objeto cuando lo creamos con el operador `new`

- Su nombre ha de coincidir coincide con el nombre de la clase.
- Por definición, no devuelven nada.