MÉTODOS EN JAVA

En Java, las "funciones" se denominan más comúnmente **métodos** cuando están definidas dentro de una clase. Son bloques de código que realizan una tarea específica y pueden ser llamados (invocados) desde otras partes del programa.

Aquí tienes una guía completa sobre cómo funcionan los métodos (funciones) en Java, cubriendo todos los puntos que mencionaste.

En Java, las "funciones" se denominan más comúnmente **métodos** cuando están definidas dentro de una clase. Son bloques de código que realizan una tarea específica y pueden ser llamados (invocados) desde otras partes del programa.

---

1. Anatomía Básica: Parámetros y Retornos

La firma de un método define qué acepta y qué devuelve.

Parámetros (Lo que entra)

Los parámetros son los valores que un método **recibe** para poder trabajar. Se declaran en los paréntesis `( )` después del nombre del método, especificando su tipo y nombre.

- **Sin parámetros:** Si el método no necesita ninguna información, los paréntesis se dejan vacíos.
- **Con parámetros:** Se listan los tipos y nombres de las variables.

Retornos (Lo que sale)

El tipo de retorno se declara justo antes del nombre del método. Indica el tipo de dato que el método **devuelve** al código que lo llamó.

- **`void` (Sin retorno):** Si el método realiza una acción (como imprimir en pantalla) pero no devuelve ningún valor, se usa la palabra clave `void`.
- **Tipo específico (p.ej., `int`, `String`):** Si el método calcula o busca un valor, debe declarar ese tipo y usar la palabra clave `return` para devolver el resultado.

Ejemplo combinado:

```
class Calculadora {

    // 1. Método SIN parámetros y SIN retorno (void)
    public void mostrarBienvenida() {
        System.out.println("Bienvenido a la calculadora");
```

```
    }

    // 2. Método CON parámetros y CON retorno (int)
    public int sumar(int a, int b) {
        // 'a' y 'b' son parámetros
        int resultado = a + b;
        return resultado; // 'return' devuelve el valor
    }

    // 3. Método CON parámetros y SIN retorno (void)
    public void imprimirSuma(int a, int b) {
        int resultado = a + b;
        System.out.println("El resultado es: " + resultado);
    }
}
```

2. Static vs. Non-Static (El Contexto)

Esta es la diferencia más importante para entender el uso de métodos.

Métodos de Instancia (Non-Static)

- **¿Qué son? Son métodos que pertenecen a un objeto o instancia de la clase.**
- **¿Cómo funcionan? Solo pueden ser llamados *después* de crear un objeto de esa clase usando la palabra clave `new`.**
- **¿Para qué sirven? Generalmente, trabajan con los atributos (variables) específicos de ese objeto.**
- **Analogía: Piensa en la clase `Coche` como el plano. Un método no estático como `arrancar()` solo funciona en un coche *específico* (un objeto). No puedes "arrancar" el plano, necesitas un coche real.**

Ejemplo (Non-Static):

```
// Definición en la clase Coche
class Coche {
    String marca; // Atributo de instancia

    // Método de instancia (non-static)
    public void arrancar() {
        // 'this.marca' se refiere a la marca de ESTE coche en
particular
        System.out.println("Arrancando el " + this.marca);
    }
```

```
}

// Uso en OTRA clase
Coche miCoche = new Coche(); // 1. CREAR LA INSTANCIA (Objeto)
miCoche.marca = "Toyota";
miCoche.arrancar(); // 2. LLAMAR AL MÉTODO SOBRE EL OBJETO

// Esto daría ERROR:
// Coche.arrancar(); // Error: no se puede llamar a un método no
estático
```

Métodos de Clase (Static)

- **¿Qué son? Son métodos que pertenecen a la clase en sí misma, no a un objeto individual.**
- **¿Cómo funcionan? Se pueden llamar directamente usando el nombre de la clase, sin necesidad de crear una instancia (new).**
- **¿Para qué sirven? Se usan para tareas de utilidad que no dependen del estado de un objeto. `Math.random()` es un ejemplo clásico.**
- **Analogía: Un método estático es como una herramienta de utilidad pública. `Math.random()` es una calculadora que cualquiera puede usar sin necesidad de "comprar" (instanciar) su propia calculadora.**

Ejemplo (Static):

```
// Definición en la clase Utilidades
class Utilidades {

    // Método de clase (static)
    public static double convertirAPulgadas(double cm) {
        return cm / 2.54;
    }
}

// Uso en OTRA clase
// NO necesitamos 'new Utilidades()'
double pulgadas = Utilidades.convertirAPulgadas(10.0);
System.out.println("10 cm son " + pulgadas + " pulgadas.");
```

3. Uso Dentro de una Misma Clase

Aquí es donde se combinan `static` y `non-static`. Las reglas son simples:

Regla 1: Un método `static` NO PUEDE llamar a un método `non-static` directamente.

- **Razón: El método estático existe "en general" (en la clase), pero el método no estático necesita un objeto *específico* para funcionar. El método estático no sabe en qué objeto (instancia) debe ejecutar el método no estático.**

Regla 2: Un método `non-static` SÍ PUEDE llamar a un método `static` directamente.

- **Razón: El método estático siempre está disponible para la clase, por lo que cualquier método de instancia puede usarlo.**

Ejemplo de reglas en una clase:

```
class MiAplicacion {

    String mensajeInstancia = "Hola desde un objeto"; // Atributo
non-static

    // --- Métodos de Instancia (Non-Static) ---

    // Método non-static (principal)
    public void iniciar() {
        System.out.println("Iniciando aplicación...");

        // Regla 2: non-static puede llamar a static
        String info = getInfoEstatica();
        System.out.println(info);

        // non-static puede llamar a non-static
        this.imprimirMensaje();
    }

    // Otro método non-static
    public void imprimirMensaje() {
        System.out.println(this.mensajeInstancia);
    }

    // --- Métodos de Clase (Static) ---

    // Método static
    public static String getInfoEstatica() {
        return "Información de la clase";
    }

    // El método main es siempre static
    public static void main(String[] args) {
```

```
        // Un método static (main) puede llamar a otro static
        System.out.println(getInfoEstatica());

        // Regla 1: Un método static (main) NO puede llamar a
'iniciar()'
        // iniciar(); // ¡ERROR! ¿En qué objeto lo ejecuto?

        // Para solucionar esto, debemos crear una instancia:
        MiAplicacion app = new MiAplicacion();
        app.iniciar(); // Ahora sí, llamamos a 'iniciar()' sobre el
objeto 'app'
    }
}
```

4. "Parámetros por Defecto" (Mediante Sobrecarga)

Java **no tiene** parámetros por defecto de la misma manera que Python o JavaScript (donde puedes escribir `function(nombre = "Mundo")`).

En Java, esto se logra mediante la **sobrecarga de métodos** (Method Overloading).

La sobrecarga consiste en crear múltiples métodos con el **mismo nombre** pero con **diferentes parámetros** (diferente número de parámetros o diferentes tipos).

Ejemplo de sobrecarga para simular valores por defecto:

```
class Saludo {

    // 1. El método "principal" con todos los parámetros
    public void saludar(String nombre, String saludo) {
        System.out.println(saludo + ", " + nombre + "!");
    }

    // 2. Sobrecarga: Si solo me dan el nombre, uso un saludo "por
defecto"
    // Este método actúa como si 'saludo' tuviera un valor por defecto.
    public void saludar(String nombre) {
        // Llama al método principal usando un valor por defecto
        saludar(nombre, "Hola");
    }

    // 3. Sobrecarga: Si no me dan nada, uso valores por defecto para
todo
```

```
    public void saludar() {
        // Llama al método de un parámetro, que a su vez llama al
principal
        saludar("Mundo");
    }
}

// --- Cómo se usa ---
Saludo s = new Saludo();

s.saludar("Ana", "Buenos días"); // Llama al (1) -> "Buenos días, Ana!"
s.saludar("Juan");               // Llama al (2) -> "Hola, Juan!"
s.saludar();                     // Llama al (3) -> "Hola, Mundo!"
```