



Métodos en Java

Laura Daniela Capera Vásquez

ingeniería de Sistemas y Computación

Programación I

William Alexander Matallana Porras

2025-2

Contenido

Introducción	4
Objetivos	5
Objetivo General	5
Objetivos Específicos	5
Desarrollo	6
¿Qué es un método en Java?	6
Estructura	6
Tipos de métodos en Java	8
Código de menú utilizando método static y los diferentes métodos	18
Referencias	19

Lista de Ilustraciones

Ilustración 1- Fuente propia	7
Ilustración 2- Fuente propia	9
Ilustración	
3- https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=65&codigo=143&inicio=60	10
Ilustración 4- Fuente propia	10
Ilustración 5-	
https://www.reddit.com/r/learnprogramming/comments/rn43f9/what_are_instances_in_java_are_they_the_same_as/?t=es-419	11
Ilustración 6-	
https://www.arquitecturajava.com/static-method-vs-instance-method-y-su-uso-correcto/#Java_Metodo_de_instancia	12
Ilustración 7- https://youtu.be/vEG2NXFMaGQ?si=iivvrHOqwSyx86Ve	13
Ilustración 8- https://www.datacamp.com/es/doc/java/return	14
Ilustración 9- https://www.datacamp.com/es/doc/java/return	14
Ilustración 10- https://www.datacamp.com/es/doc/java/constructors	15
Ilustración 11- https://www.datacamp.com/es/doc/java/constructors	15
Ilustración 12- https://es.javascript.info/property-accessors 16	
Ilustración 13- https://dat-science.com/metodos-getter-y-setter/	17
Ilustración 14- https://javamagician.com/java-sobrecarga-sobrescritura-metodos/	18
Ilustración 15- https://javamagician.com/java-sobrecarga-sobrescritura-metodos/	18
Ilustración 16- https://www.tutorialspoint.com/java/java_overriding.htm	19
Ilustración 17- https://www.tutorialspoint.com/java/java_overriding.htm	20
Ilustración 18- https://openwebinars.net/blog/introduccion-a-java-metodos-parametros-y-argumentos/ .	20
Ilustración 19- Fuente propia	21
Ilustración 20- https://www.datacamp.com/es/doc/java/abstract 22	

Ilustración 21-Fuente propia.....	22
Ilustración 22-Fuente propia.....	23
Ilustración 23- https://medium.com/@vinodkumarbheel61/factory-design-pattern-in-java-with-example-ab896d5b75fd	23

Introducción

En el desarrollo de códigos en Java, los métodos son herramientas fundamentales para organizar y estructurar el código. Un método es un bloque de instrucciones con un nombre específico que realiza una tarea en concreto. Gracias a esto se mejora la legibilidad y el mantenimiento, ya que en lugar de escribir el mismo código una y otra vez, podemos agruparlo en un método y llamarlo/invocarlo cada vez que necesitemos que la tarea se ejecute. Estos se clasifican en los que retornan valores, los que no retornan, los que reciben parámetros y los que no lo hacen.

Además, analizaremos la importancia del modificador static y como se realizará un código que contenga los demás métodos.

En este documento se presentarán ejemplos de los distintos métodos de Java, mostrando dos casos de cada uno. También se incluirán ejemplos específicos del uso de métodos static. Todo el código será trabajado en la multiplataforma IntelliJ-IDEA Community y posteriormente se almacenará en el repositorio de GitHub.

Objetivos

Objetivo General

- ★ Comprender y diferenciar los distintos métodos en Java, incluyendo su implementación con la palabra `static`, a través de ejemplos prácticos desarrollados en IntelliJ-IDEA Community y almacenarlos en el repositorio de GitHub.

Objetivos Específicos

- ★ Identificar los diferentes métodos de Java y su función.
- ★ Implementar dos ejemplos de cada tipo de método para reforzar la comprensión.
- ★ Aplicar la palabra clave `static` en la creación de métodos de clase utilizando diferentes tipos de retorno.
- ★ Utilizar la multiplataforma IntelliJ-IDEA Community para escribir, copiar y ejecutar el código.

Desarrollo

¿Qué es un método en Java?

Un método en Java es un bloque de código que realiza una tarea en específico, ya que en lugar de escribir el mismo conjunto de instrucciones una y otra vez, podemos juntar esas acciones en un método y luego simplemente llamarlo cada vez que lo necesitemos. (Branding, L. (s. f.)). Los métodos también son conocidos como funciones ya que se utilizan para realizar acciones específicas. De igual manera se tiene el método `main()` también sirven como punto de entrada para la máquina virtual Java inicie la ejecución del programa Java. (TheKnowledgeAcademy (s. f.)).

```
public static void main(String[] args) {  @ LauraC
```

Ilustración 1- Fuente propia

Estructura

1. Especificador de acceso: también conocido como modificador, determina el tipo de acceso y la visibilidad del método. Existen 4 tipos:

- ★ Público: Habilita el acceso del método desde todas las clases de la aplicación.

(TheKnowledgeAcademy (s. f.)).

- ★ Privado: Limita la accesibilidad del método únicamente a las clases dentro de las que esta definido. (TheKnowledgeAcademy (s. f.)).

- ★ Protegido: Permite accesibilidad del método dentro del mismo paquete o subclases en otros paquetes. (TheKnowledgeAcademy (s. f.)).

- ★ Predeterminado: Si no se utiliza alguno de los anteriores mencionados Java utiliza el especificador de acceso “predeterminado” por defecto. Significa que la visibilidad del método se limita al mismo paquete. (TheKnowledgeAcademy (s. f.)).
- 2. Tipo de retorno: Es el tipo de dato que devuelve el método, devolverá un valor de método o, si no, void. Es obligatorio declarar el tipo de retorno en la sintaxis del método. Siendo estos int, void, double, string o bool.
- 3. Nombre del método: Es un identificador que se utiliza para referirse a un método específico. Este nombre suele ser único y se utiliza para definir la funcionalidad del método. (TheKnowledgeAcademy (s. f.)).
- 4. Lista de parámetros: Esta es separada por comas y entre paréntesis, también incluye el nombre de la variable precediendo por el tipo de dato. Si el método no tiene parámetros los paréntesis pueden dejarse vacíos. (TheKnowledgeAcademy (s. f.)).
- 5. Lista de excepciones: El método podría generar excepciones, en ese caso debe especificarse como una lista, para esto Java utiliza dos palabras claves: “throw” y “throws” . La palabra clave “throw” se utiliza para generar explícitamente una excepción dentro del método o de un bloque de código. Por otro lado, la palabra clave “throws” se usa normalmente para especificar las excepciones que pueden ocurrir durante la ejecución del programa, sin embargo, un método que declara una expresión usando la palabra no es necesario para gestionar la excepción en sí. Además, las excepciones que se comprueban durante la ejecución deben lanzarse usando la palabra clave

"throws", mientras que las excepciones no comprobadas no necesitan lanzarse explícitamente en el código. (TheKnowledgeAcademy (s. f.)).

6. Cuerpo del método: Este componente del encabezado del metodo contiene las sentencias de código que se utilizan para ejecutar tareas, esta suele ir entre llaves {}

```
public static void main(String[] args) {  @LauraC *  
  
//Ejercicio 1: Genera 100 numeros entre 1 a 200 de manera aleatoria  
    Random ale = new Random();  
    int n3, i, j = 0;  
    for (i = 1; i <= 100; i++) {  
        n3 = ale.nextInt( bound: (200 - 1) + 1) + 1;  
        System.out.println(n3);  
        j = n3 + j;  
    }  
    System.out.println("La suma de esos 100 números es:" + j);  
}
```

Ilustración 2- Fuente propia

Tipos de métodos en Java

1. Estáticos: Son aquellos que pertenecen a la clase en lugar de una instancia de la clase, su nombre incluye la palabra clave "static" antes del nombre del método. Pueden ser invocados directamente desde la clase sin la necesidad de crear un objeto. (Branding, L. (s. f.)). Ejemplos:


```
public class Operacion {

    public static int sumar(int x1, int x2) {
        int s = x1 + x2;
        return s;
    }

    public static int restar(int x1, int x2) {
        int r = x1 - x2;
        return r;
    }

}
```

Ilustración 3-<https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=65&codigo=143&inicio=60>

```
public class Main {  LauraC *
    public static void main(String[] args) {  LauraC *

        // Generar 100 numero aleatorios entre 1-200 y cuanto es la suma de los pares e impares
        Random al=new Random ();
        int n1, m, pares=0, impares=0;
        for (m=1; m<=100; m++){
            n3=al.nextInt( bound: (200-1)+1)+1;
            System.out.println(n3);
            if (n3%2==0){
                pares=n3+pares;
            }else if (n3%2!=0){
                impares=n3+impares;
            }
        }
        System.out.println("La suma de los pares es:" + pares);
        System.out.println("La suma de los impares es:" + impares);
    }
}
```

Ilustración 4- Fuente propia

2. Instancia: Estos operan en instancias de una clase y pueden acceder a los campos y otros métodos de esa instancia. Son invocados a través de un objeto creado a partir de la clase. (Branding, L. (s. f.)).

Puede acceder a variables estáticas y no estáticas, así como a métodos de la clase, puede utilizar la

palabra clave “this” o “super” . Para llamar a un método de instancia, usamos

“objectName.methodName()”. (TheKnowledgeAcademy (s. f.)). Ejemplos:

```
public class Car extends Object {
    private String name;
    public Car(String name) {
        this.name = name; // asigna el valor del argumento “name”
                           // a la propiedad “name” de esta instancia.
    }
    public String name() {
        return this.name;
    }

    public static void main(String[] args) {

        Car myCar = new Car("Mine");
        Car yourCar = new Car("Yours");

        System.out.println(myCar.name()); // Mine
        System.out.println(yourCar.name()); // Yours

        System.out.println(myCar.equals(yourCar)); // false. No es la misma instancia.

        System.out.println(myCar); // imprime la dirección de la instancia en memoria
        System.out.println(yourCar); // dirección de memoria diferente
    }
}
```

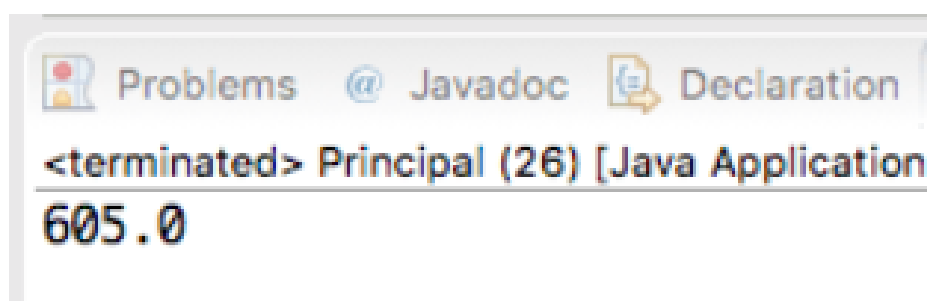
Ilustración 5-

https://www.reddit.com/r/learnprogramming/comments/m43f9/what_are_instances_in_java_are_they_the_same_as/?t=es-419

Se trata de un método de instancia (instance method) . Un método de instancia esta relacionado **con el estado que almacena un objeto que creamos a partir de una clase**. En este caso queremos acceder al importe y calcular su IVA. Por lo tanto nosotros en nuestro programa main construimos una factura y calculamos su importe con el IVA usando el método de instancia.

```
1. package com.arquitecturajava;
2.
3. public class Principal {
4.
5.     public static void main(String[] args) {
6.
7.         Factura f= new Factura(1, "ordenador", 500);
8.
9.         System.out.println(f.getImporteIVA());
10.
11.     }
12.
13. }
```

El resultado le veremos aparecer en la consola :



Problems @ Javadoc Declaration
<terminated> Principal (26) [Java Application]
605.0

Ilustración 6-

https://www.arquitecturajava.com/static-method-vs-instance-method-y-su-uso-correcto/#Java_Metodo_de_instancia

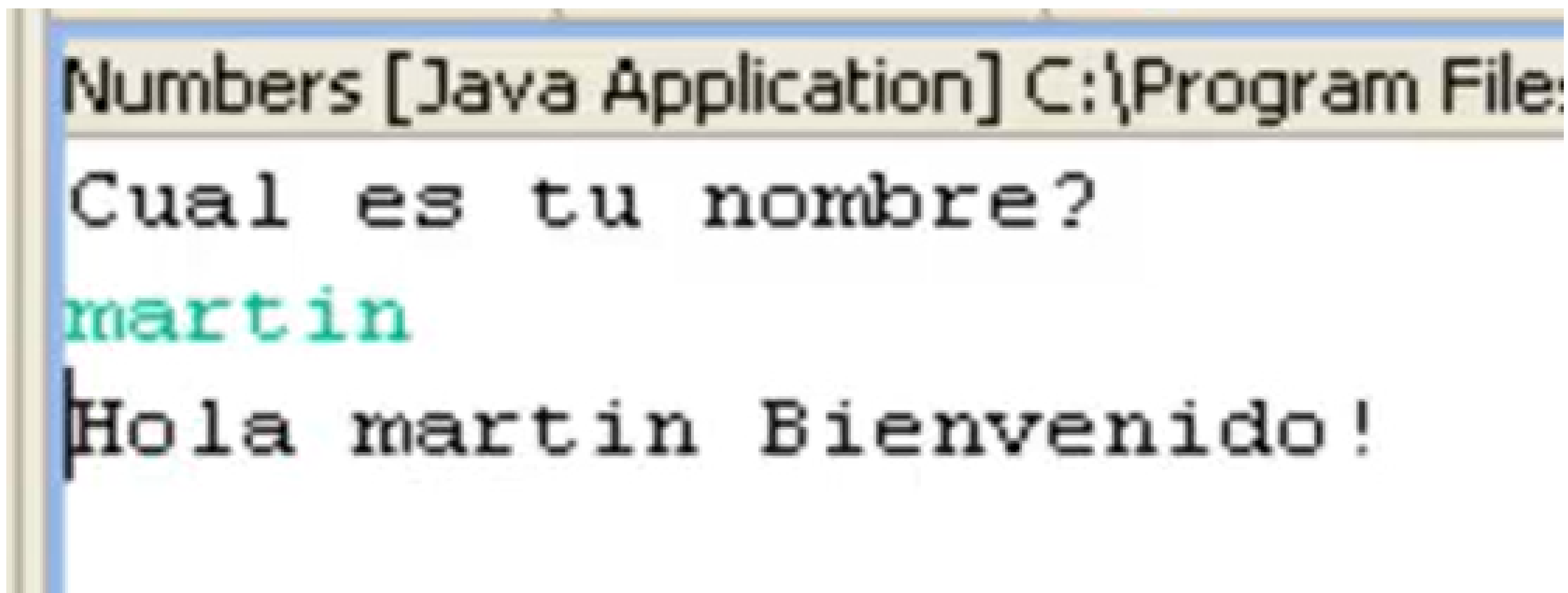
3. Parámetros: Los métodos pueden aceptar uno a más parámetros, que son los valores que se pasan al método cuando se invoca, estos pueden influir en el comportamiento del método. (Branding, L. (s. f.)). Ejemplos:

```
import java.util.Scanner;

class Numbers{

    public static void main(String[] args){
        Scanner entrada = new Scanner(System.in);
        String nombre;

        System.out.println("Cual es tu nombre? ");
        nombre = entrada.nextLine();
        mensaje(nombre);
    }
    public static void mensaje(String nombre){
        System.out.println("Hola " + nombre + " Bienvenido!");
    }
}
```



Numbers [Java Application] C:\Program Files:
Cual es tu nombre?
martin
Hola martin Bienvenido!

Ilustración 7- <https://youtu.be/vEG2NXFMaGQ?si=iivrHOqwSyx86Ve>

4. Con valor de retorno: Algunos métodos devuelven un valor después de completar su tarea, el tipo de dato del valor de retorno debe coincidir con el tipo específico en la firma del método. (Branding, L. (s. f.)). Ejemplos:

```
public class ReturnVoidExample {  
    public static void main(String[] args) {  
        checkNumber(5);  
        checkNumber(-1);  
    }  
  
    public static void checkNumber(int number) {  
        if (number < 0) {  
            System.out.println("Negative number");  
            return; // Exit the method early  
        }  
        System.out.println("Positive number");  
    }  
}
```

Ilustración 8- <https://www.datacamp.com/es/doc/java/return>

```
public class ReturnExample {  
    public static void main(String[] args) {  
        int result = add(5, 3);  
        System.out.println("Result: " + result);  
    }  
  
    public static int add(int a, int b) {  
        return a + b;  
    }  
}
```

Ilustración 9- <https://www.datacamp.com/es/doc/java/return>

5. Constructores: Estos en sí no son métodos, los constructores son bloques de código especiales utilizados para inicializar objetos cuando se crean instancias de una clase. (Branding, L. (s. f.)).

Ejemplos:

```
public class Car {
    String model;
    int year;

    // Default Constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    public static void main(String[] args) {
        Car car = new Car();
        System.out.println("Model: " + car.model + ", Year: " + car.year);
    }
}
```

Ilustración 10- <https://www.datacamp.com/es/doc/java/constructors>

```
public class Car {
    String model;
    int year;

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public static void main(String[] args) {
        Car car = new Car("Toyota", 2021);
        System.out.println("Model: " + car.model + ", Year: " + car.year);
    }
}
```

Ilustración 11- <https://www.datacamp.com/es/doc/java/constructors>

6. Getter y Setter: Esta es más utilizada más que todo en la orientación de objetos y se utiliza para obtener (get) y establecer (set) los valores de los campos privados de una clase. (Branding, L. (s. f.)).

Ejemplos:

```
1  let user = {
2    name: "John",
3    surname: "Smith",
4
5    get fullName() {
6      return `${this.name} ${this.surname}`;
7    },
8
9    set fullName(value) {
10     [this.name, this.surname] = value.split(" ");
11   }
12 };
13
14 // set fullName se ejecuta con el valor dado.
15 user.fullName = "Alice Cooper";
16
17 alert(user.name); // Alice
18 alert(user.surname); // Cooper
```

Ilustración 12- <https://es.javascript.info/property-accessors>

```
1 public class Coche{
2     int CV;
3     int numeroPuertas;
4     int añoFabricacion;
5     String propietario;
6     String color;
7     String combustible;
8     boolean encendido;
9     private int porcentajeTanqueLleno;
10    public Coche(int np, String pro){
11        this.numeroPuertas = np;
12        this.propietario=pro;
13    }
14    public void arranque(){
15        if (porcentajeTanqueLleno>5){
16            encendido = true;
17        }
18    }
19    public void parado(){
20        encendido = false;
21    }
22    public int llenadoTanque(){
23        if (porcentajeTanqueLleno<80){
24            porcentajeTanqueLleno = 100;
25        }
26        return porcentajeTanqueLleno;
27    }
28    //metodos getter and setter
29    public int getporcentajeTanqueLleno(){
30        return this.porcentajeTanqueLleno;
31    }
32    public void setporcentajeTanqueLleno(int val){
33        this.porcentajeTanqueLleno=val;
34    }
35 }
```

Ilustración 13- <https://dat-science.com/metodos-getter-y-setter/>

7. Sobrecargados: En este implica definir varios métodos en la misma clase con el mismo nombre, pero diferentes tipos o cantidad de parámetros. (Branding, L. (s. f.)). Ejemplos:


```
public class Multiplicador {  
  
    public int multiplicar(int a, int b) {  
        return a * b;  
    }  
  
    public int multiplicar(int a, int b, int c) {  
        return a * b * c;  
    }  
}
```

Ilustración 14- <https://javamagician.com/java-sobrecarga-sobrescritura-metodos/>

```
public class Multiplicador {  
  
    public int multiplicar(int a, int b) {  
        return a * b;  
    }  
  
    public int multiplicar(int a, int b, int c) {  
        return a * b * c;  
    }  
  
    public double multiplicar(double a, double b) {  
        return a * b;  
    }  
}
```

Ilustración 15- <https://javamagician.com/java-sobrecarga-sobrescritura-metodos/>

8. Anulados: “En la herencia, una subclase puede proporcionar una implementación diferente para un método que ya está definido en su clase padre. Estos métodos anulados permiten lograr el polimorfismo.” (Branding, L. (s. f.)). Ejemplos:

```
class Animal {  
    public void move() {  
        System.out.println("Animals can move");  
    }  
}  
  
class Dog extends Animal {  
    public void move() {  
        System.out.println("Dogs can walk and run");  
    }  
}  
  
public class TestDog {  
  
    public static void main(String args[]) {  
        Animal a = new Animal();    // Animal reference and object  
        Animal b = new Dog();       // Animal reference but Dog object  
  
        a.move();    // runs the method in Animal class  
        b.move();    // runs the method in Dog class  
    }  
}
```

```
Animals can move  
Dogs can walk and run
```

```
class Animal {  
    public void move() {  
        System.out.println("Animals can move");  
    }  
}  
  
class Dog extends Animal {  
    public void move() {  
        super.move(); // invokes the super class method  
        System.out.println("Dogs can walk and run");  
    }  
}  
  
public class TestDog {
```

```
    public static void main(String args[]) {  
        Animal b = new Dog(); // Animal reference but Dog object  
        b.move(); // runs the method in Dog class  
    }  
}
```

```
Animals can move  
Dogs can walk and run
```

Ilustración 17- https://www.tutorialspoint.com/java/java_overriding.htm

9. Sin retorno: “Estos métodos son aquellos que no devuelven ningún valor. Se utilizan principalmente para ejecutar un conjunto de instrucciones o modificar el estado de un objeto.” (Branding, L. (s. f.)).

Ejemplos:

```
public void saludar() {  
    System.out.println("¡Hola, mundo!");  
}
```

Ilustración 18-<https://openwebinars.net/blog/introducción-a-java-metodos-parametros-y-argumentos/>

```
public static void edad(Scanner teclado){ 1 usage

    System.out.println("método sin retorno");
    System.out.println("mayor o menor de edad?");

    System.out.println("Digite su edad:");
    int edad=teclado.nextInt();
    if (edad>=18){
        System.out.println("Eres mayor de edad!");
    }else {
        System.out.println("Eres menor de edad!");
    }
}
```

Ilustración 19-Fuente propia

10. Abstracto: Según DataCamp. (n.d.) un método abstracto es un método que se declara sin una implementación. Las subclases de la clase abstracta deben proporcionar una implementación de estos métodos.

```
abstract class Animal {
    // Abstract method
    abstract void makeSound();

    // Concrete method
    void sleep() {
        System.out.println("Sleeping...");
    }
}

class Dog extends Animal {
    // Implementing the abstract method
    void makeSound() {
        System.out.println("Bark");
    }
}

public class AbstractExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); // Outputs: Bark
        dog.sleep();     // Outputs: Sleeping...
    }
}
```

Ilustración 20- <https://www.datacamp.com/es/doc/java/abstract>

```
static abstract class Animal { 2 usages 1 inheritor
    public abstract void habla (); 1 usage 1 implementation
}

static class bubu extends Animal{ 1 usage
    public void habla(){ 1 usage
        System.out.println("Metodo abstracto");
        System.out.println("Holly!!");
    }
}
```

Ilustración 21-Fuente propia

11. Fabrica: Según Refactoring.Guru. (2025, 1 enero) el método de fábrica define un método que debe usarse para crear objetos en lugar de una llamada directa al constructor (newoperador). Las subclases pueden sobrescribir este método para cambiar la clase de los objetos que se crearán.

```
public static class conexion { 10 usages
    private String tipo; 3 usages
    private conexion() { 2 usages

    }
    public static conexion conexionL() { 1 usage
        conexion c= new conexion();
        c.tipo= "Local";
        return c;
    }
    public static conexion conexionR() { 1 usage
        conexion c= new conexion();
        c.tipo= "Remota";
        return c;
    }
    public void conectar() { 2 usages
        System.out.println("Espere, conectando a la base "+ tipo);
    }
}
```

Ilustración 22-Fuente propia

```
clase pública Cliente {  
    public static void main (String[] args) {  
        ShapeFactory circleFactory = new CircleFactory ();  
        Forma círculo = circleFactory.createShape();  
        círculo.draw();  
  
        ShapeFactory squareFactory = new SquareFactory ();  
        Forma cuadrado = squareFactory.createShape();  
        cuadrado.draw();  
  
        ShapeFactory rectánguloFactory = new RectangleFactory ();  
        Forma rectángulo = rectangleFactory.createShape();  
        rectángulo.draw();  
    }  
}
```

Ilustración 23- <https://medium.com/@vinodkumarbheel61/factory-design-pattern-in-java-with-example-ab896d5b75fd>

Código de menú utilizando los diferentes métodos

<https://github.com/LauraD-19/Men-y-m-todos-de-Java.git>

Conclusiones

- ★ En este trabajo conocimos algunos de los muchos métodos de Java y podemos concluir que estos métodos son una manera que de a la hora de escribir nuestro código sea de manera ordenada y que agilizar el proceso de desarrollo.

Referencias

Barragán, A. (2023a, octubre 24). Introducción a Java: Métodos, parámetros y argumentos.

OpenWebinars.net. <https://openwebinars.net/blog/introduccion-a-java-metodos-parametros-y-argumentos/>

Barragán, A. (2023b, octubre 24). Introducción a Java: Métodos, parámetros y argumentos.

OpenWebinars.net. <https://openwebinars.net/blog/introduccion-a-java-metodos-parametros-y-argumentos/>

Branding, L. (s. f.). *INTRODUCCIÓN a LOS MÉTODOS EN JAVA: TIPOS y FUNCIONAMIENTO*.

<https://cursosnascor.com/blog-detalle/introduccion-los-metodos-en-java-tipos-y-funcionamiento>

DataCamp. (n.d.). palabra clave abstracta en Java: Uso y ejemplos.

<https://www.datacamp.com/es/doc/java/abstract>

De TheKnowledgeAcademy). (s. f.). Introduction To Methods In Java: A Detailed Guide.

<https://www.theknowledgeacademy.com/blog/methods-in-java/#:~:text=%C2%BFCu%C3%A1les%20son%20las%20partes%20del,definen%20la%20visibilidad%20del%20m%C3%A9todo.>

Métodos en Java. (s. f.). <http://www.edu4java.com/es/java/metodos-en-java.html>

Métodos estáticos de una clase. (s. f.).

<https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=65&codigo=143&inicio=60>

Refactoring.Guru. (2025, 1 enero). *Factory Method in Java*.

[https://refactoring.guru/design-patterns/factory-method/java/example#:~:text=Factory%20method%20is%20a%20creational,constructor%20call%20\(%20new%20operator\).](https://refactoring.guru/design-patterns/factory-method/java/example#:~:text=Factory%20method%20is%20a%20creational,constructor%20call%20(%20new%20operator).)



Selawsky, J. (2024, 5 febrero). *Métodos en Java*. CodeGym.

<https://codegym.cc/es/groups/posts/es.34.metodos-en-java>