

Línea horizontal

**Lic. Ignacio G Fontaine Digier**

ITU

Programación Orientada a Objetos

**19 de agosto de 2024**

[**Capítulo 1: Introducción a la Programación Orientada a Objetos y Java 3**](#_dxsx05lplqkx)

[1.1. Historia y Evolución de la Programación Orientada a Objetos 3](#_4vbspbob3x6o)

[1.1.1. Orígenes de la Programación Orientada a Objetos 3](#_wianjmk1no1o)

[1.1.2. Evolución y Popularización 3](#_ays1r5oam0nv)

[1.1.3. POO en el Contexto Moderno 3](#_tcpko7uj4u17)

[1.2. Principios Fundamentales de la POO 3](#_ls4uo6dlqpaj)

[1.2.1. Abstracción 3](#_9o4hny9ago25)

[1.2.2. Encapsulamiento 4](#_y2ooy0zdqcds)

[1.2.3. Herencia 4](#_t93rnexrmdrh)

[1.2.4. Polimorfismo 4](#_3t4xtz1tmrfr)

[1.3. Introducción a Java como Lenguaje Orientado a Objetos 5](#_rpliqvn9vom3)

[1.3.1. Historia de Java 5](#_6av5w9aaidsz)

[1.3.2. Características de Java 5](#_7g96kmejgat2)

[1.3.3. Sintaxis Básica de Java 5](#_6r53v2x8o4x9)

[1.3.4. La JVM y el Entorno de Ejecución de Java 6](#_qutp2yl76yjt)

[1.4. Beneficios de la POO en Java 6](#_7tc5h1eluxyp)

[1.4.1. Reutilización de Código 6](#_e5l5ursf2pa3)

[1.4.2. Modularidad 6](#_fdsjrklge6wy)

[1.4.3. Mantenibilidad 6](#_u2pkoo6t8i4s)

[1.4.4. Escalabilidad 7](#_dlwti7gpmfmc)

[1.5. Conclusión 7](#_vbsxf32s1rue)

[**Capítulo 2: Clases y Objetos en Java 7**](#_bbluooad2hqb)

[2.1. Introducción a las Clases y Objetos 7](#_p79wfuf9adur)

[2.1.1. Definición de Clases y Objetos 7](#_p7nj5leun9tw)

[2.1.2. Atributos y Métodos 8](#_zqpnckzv1pg)

[2.1.3. Creación de Objetos 8](#_csagjphijdsx)

[2.1.4. Diferencia entre Clases y Objetos 9](#_z1ugoxl7eqh3)

[2.2. Constructores en Java 9](#_2ngof7m7z19o)

[2.2.1. Introducción a los Constructores 9](#_67w7631df4au)

[2.2.2. Sobrecarga de Constructores 10](#_48nfgtx13v3i)

[2.2.3. Constructores por Defecto 11](#_7owo20qpyhks)

[2.3. Modificadores de Acceso 11](#_931y769p8h6i)

[2.3.1. Tipos de Modificadores de Acceso 11](#_1vpcd91dis8t)

[2.3.2. Encapsulamiento mediante Modificadores de Acceso 12](#_pc4woy2jmdyg)

[2.3.3. Acceso a Atributos y Métodos 12](#_b762dm73kz79)

[2.4. Métodos en Java 13](#_p4m11yvxscjc)

[2.4.1. Definición de Métodos 13](#_zgf8fsiljq9g)

[2.4.2. Sobrecarga de Métodos 13](#_9y1edajdfyr0)

[2.4.3. Métodos Estáticos 14](#_9plhpndq1pht)

[2.4.4. Métodos get y set 14](#_ubx0sy9djqn9)

[2.5. this en Java 15](#_lqk8p6o7open)

[2.5.1. Uso de this 15](#_u7g1881emnxm)

[2.5.2. Referencia al Objeto Actual 16](#_z4b83oi93ci5)

[2.6. Ejemplo Práctico: Clase Coche Completa 16](#_1po8p1ehihg3)

[2.7. Conclusión 18](#_shi8argpc2xz)

[2.8. Ejercitación 19](#_qtwz3j1908yd)

# Capítulo 1: Introducción a la Programación Orientada a Objetos y Java

## 1.1. Historia y Evolución de la Programación Orientada a Objetos

### 1.1.1. Orígenes de la Programación Orientada a Objetos

La ***Programación Orientada a Objetos*** ***(POO)*** tiene sus raíces en la década de 1960, cuando los primeros lenguajes de programación comenzaron a desarrollarse. Uno de los primeros lenguajes orientados a objetos fue Simula, creado por Ole-Johan Dahl y Kristen Nygaard en 1967. Simula fue diseñado inicialmente para simulaciones, pero sus conceptos de clases y objetos sentaron las bases de lo que hoy conocemos como POO.

### 1.1.2. Evolución y Popularización

A lo largo de los años, la POO evolucionó y se adoptó en muchos lenguajes de programación, como Smalltalk, C++, y más tarde Java. La popularidad de la POO creció gracias a su capacidad para modelar el mundo real de manera más natural, facilitando la creación de sistemas complejos y reutilizables.

### 1.1.3. POO en el Contexto Moderno

Hoy en día, la POO es una de las metodologías de programación más utilizadas en el desarrollo de software. La mayoría de los lenguajes modernos, como Java, C#, Python, y Ruby, soportan la POO, lo que la convierte en una habilidad esencial para los desarrolladores de software.

## 1.2. Principios Fundamentales de la POO

La Programación Orientada a Objetos se basa en cuatro principios fundamentales que permiten la creación de software modular, reutilizable y fácil de mantener. Estos principios son:

### 1.2.1. Abstracción

La abstracción es el proceso de simplificar un sistema complejo mediante la identificación de sus características esenciales, ignorando los detalles no relevantes. En POO, la abstracción se logra mediante el uso de clases, que definen un modelo simplificado del objeto real.

* Ejemplo: Una clase **Coche** podría abstraer los atributos y comportamientos esenciales de un coche, como **marca**, **modelo**, y **conducir()**, sin preocuparse por los detalles internos del motor.

### 1.2.2. Encapsulamiento

El ***encapsulamiento*** consiste en agrupar datos y métodos que operan sobre esos datos dentro de una misma unidad, conocida como clase, y restringir el acceso directo a ciertos componentes de un objeto. Esto permite proteger los datos y asegurar que solo se acceda a ellos de manera controlada.

* Ejemplo: Una clase **CuentaBancaria** podría encapsular atributos como **saldo** y métodos como **depositar()** y **retirar()**, protegiendo el saldo mediante el uso de modificadores de acceso como private.

### 1.2.3. Herencia

La ***herencia*** permite crear una nueva clase basada en una clase existente, reutilizando el código y extendiendo o modificando su comportamiento. La clase nueva se denomina "subclase" o "clase derivada", mientras que la clase existente se denomina "superclase" o "clase base".

* Ejemplo: Una clase **CocheDeportivo** podría heredar de la clase **Coche**, añadiendo características adicionales como **turbo** y un método **activarTurbo()**.

### 1.2.4. Polimorfismo

El ***polimorfismo*** permite que un objeto tome muchas formas, permitiendo que una operación o método se comporte de manera diferente según el objeto que la invoque. Existen dos tipos de polimorfismo: en tiempo de compilación (sobrecarga de métodos) y en tiempo de ejecución (sobrescritura de métodos).

* Ejemplo: Un método **arrancar()** podría comportarse de manera diferente dependiendo de si es llamado por un objeto de la clase **Coche** o **Moto**.

## 1.3. Introducción a Java como Lenguaje Orientado a Objetos

### 1.3.1. Historia de Java

Java fue desarrollado por Sun Microsystems en 1995 bajo el liderazgo de James Gosling. Desde su creación, Java ha sido un lenguaje orientado a objetos diseñado para ser portátil, robusto, y fácil de usar. Su promesa de "write once, run anywhere" (escribir una vez, ejecutar en cualquier lugar) ha hecho de Java un lenguaje popular para aplicaciones empresariales, móviles, y web.

### 1.3.2. Características de Java

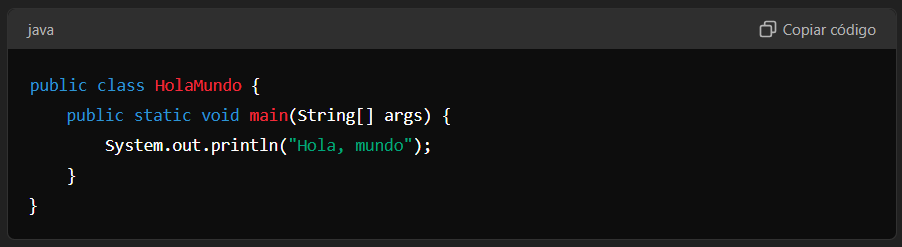
Java es un lenguaje de programación robusto con características clave que lo hacen ideal para la POO:

* Orientado a Objetos: Todo en Java se basa en objetos, lo que facilita la implementación de los principios de la POO.
* Portabilidad: Java se ejecuta en la Java Virtual Machine (JVM), lo que permite que los programas sean ejecutados en cualquier plataforma sin modificación.
* Seguridad: Java incluye características de seguridad como la gestión de memoria automática y la verificación de bytecode.
* Bibliotecas Ricas: Java ofrece un vasto conjunto de bibliotecas que simplifican el desarrollo de aplicaciones complejas.

### 1.3.3. Sintaxis Básica de Java

Java tiene una sintaxis clara y directa que se deriva en gran parte de C y C++. Esto hace que sea fácil de aprender para programadores con experiencia en otros lenguajes.

* Ejemplo: Un simple programa en Java que imprime ***"Hola, mundo"*** en la consola:



### 1.3.4. La JVM y el Entorno de Ejecución de Java

La ***Java Virtual Machine (JVM)*** es el entorno de ejecución que permite que el código Java se ejecute en cualquier plataforma que tenga una JVM instalada. El código Java se compila en bytecode, que es interpretado por la JVM, proporcionando la portabilidad característica de Java.

## 1.4. Beneficios de la POO en Java

### 1.4.1. Reutilización de Código

La POO permite reutilizar código mediante la herencia y la composición, reduciendo la duplicación y facilitando el mantenimiento.

### 1.4.2. Modularidad

La POO promueve la creación de aplicaciones modulares donde diferentes partes del sistema se encapsulan en objetos, facilitando la gestión y evolución del software.

### 1.4.3. Mantenibilidad

El uso de clases y objetos bien definidos facilita la localización y corrección de errores, así como la adición de nuevas características sin afectar otras partes del sistema.

### 1.4.4. Escalabilidad

Los sistemas orientados a objetos son más fáciles de escalar porque permiten añadir nuevas funcionalidades mediante la extensión de clases existentes y la creación de nuevas clases sin alterar la estructura existente.

## 1.5. Conclusión

La Programación Orientada a Objetos es una metodología poderosa que, cuando se combina con un lenguaje como Java, proporciona un marco sólido para el desarrollo de software moderno. En los siguientes capítulos, explicaremos en detalle cómo implementar los principios de la POO en Java, comenzando con la creación de clases y objetos, y avanzando hacia conceptos más avanzados como la herencia, el polimorfismo, y las interfaces.

# Capítulo 2: Clases y Objetos en Java

## 2.1. Introducción a las Clases y Objetos

### 2.1.1. Definición de Clases y Objetos

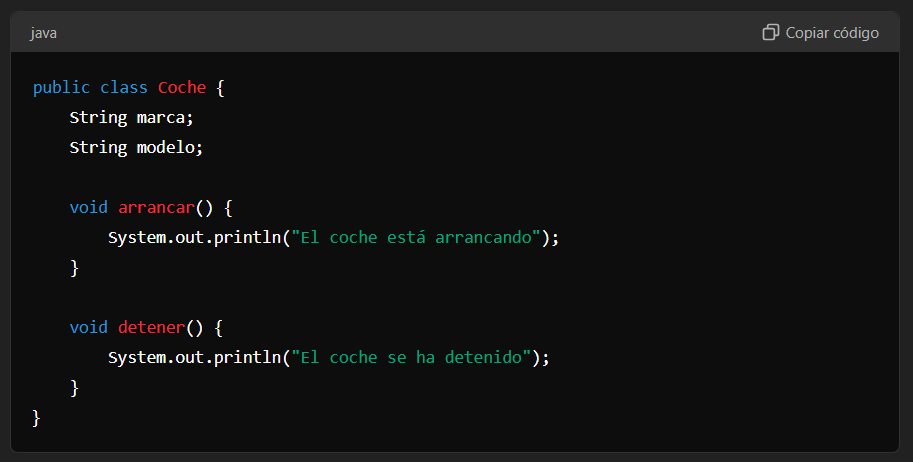
En la Programación Orientada a Objetos, una ***clase*** es una plantilla o un plano que define las ***propiedades*** (atributos) y ***comportamientos*** (métodos) de los ***objetos*** que se crearán a partir de ella. Un objeto es una instancia de una clase, representando una entidad concreta que sigue las definiciones de la clase.

* Ejemplo: Si tienes una clase **Coche**, los objetos **coche1** y **coche2** podrían ser instancias de esa clase, representando diferentes coches con atributos como **marca**, **modelo**, y **color**.

## 2.1.2. Atributos y Métodos

Los ***atributos*** son variables que almacenan el estado de un objeto, mientras que los ***métodos*** son funciones que definen el comportamiento del objeto. En Java, los atributos y métodos se declaran dentro de la clase.

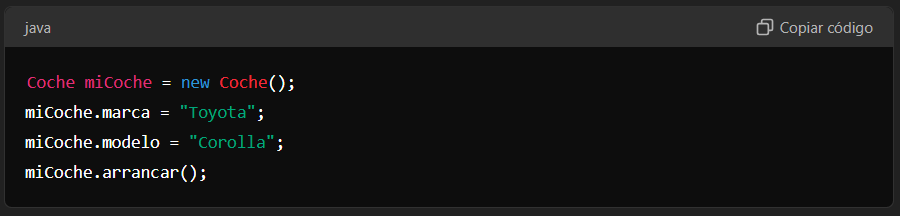
* Ejemplo: En la clase **Coche**, **marca** y **modelo** serían atributos, mientras que **arrancar()** y **detener()** serían métodos.



## 2.1.3. Creación de Objetos

Para crear un objeto en Java, se utiliza el operador ***new*** junto con el ***constructor*** de la clase. El constructor es un método especial que se llama automáticamente cuando se crea un objeto.

* Ejemplo: Crear un **objeto** de la clase **Coche**.



### 2.1.4. Diferencia entre Clases y Objetos

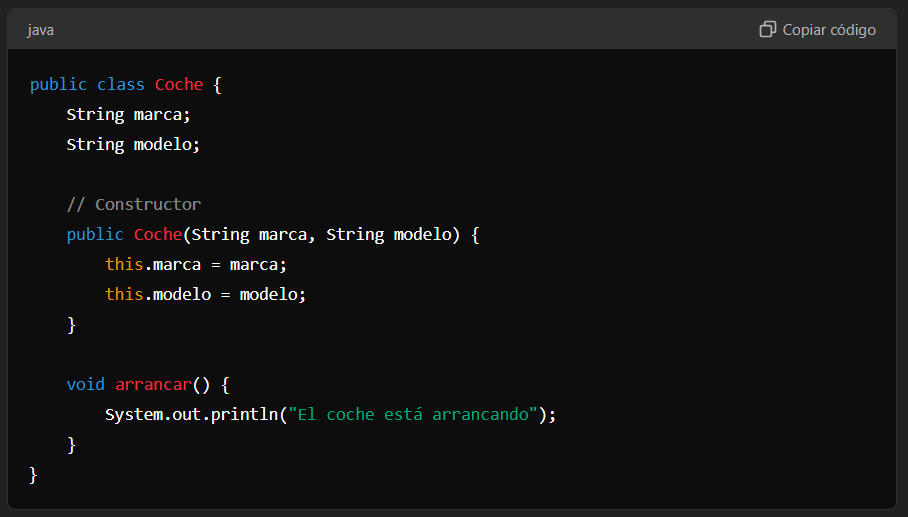
La principal diferencia entre una clase y un objeto es que la ***clase*** es una definición, mientras que un ***objeto*** es una instancia concreta de esa clase. La clase actúa como un molde, mientras que el objeto es el producto tangible que se crea a partir de ese molde.

## 2.2. Constructores en Java

### 2.2.1. Introducción a los Constructores

Un ***constructor*** es un método especial utilizado para inicializar objetos. Se llama automáticamente cuando se crea un nuevo objeto y se utiliza para establecer los valores iniciales de los atributos del objeto.

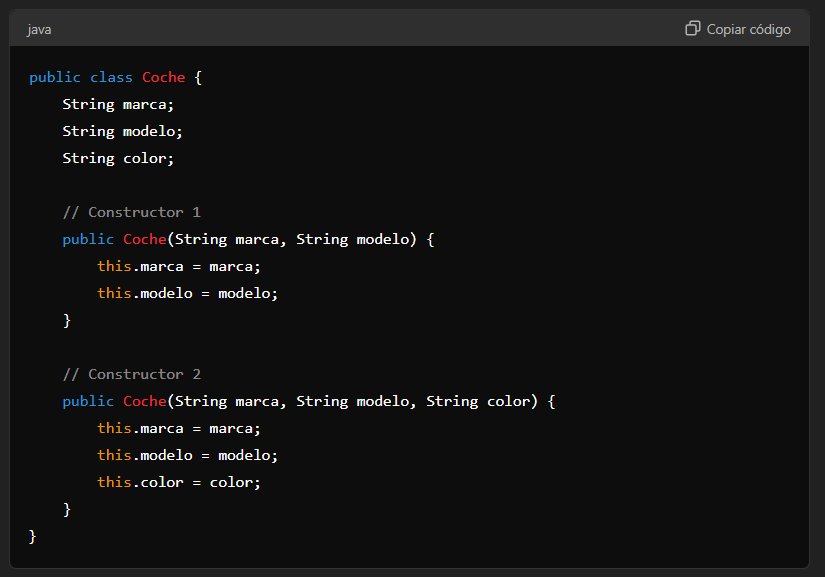
* Ejemplo: Constructor básico de la clase **Coche**.



### 2.2.2. Sobrecarga de Constructores

En Java, es posible tener múltiples constructores en una clase, siempre que difieran en el número o tipo de parámetros. Esto se conoce como ***sobrecarga de constructores*** y permite crear objetos de diferentes maneras.

* Ejemplo: **Sobrecarga de constructores** en la clase **Coche**.



### 2.2.3. Constructores por Defecto

Si no se define un constructor en una clase, Java proporciona un constructor por defecto sin parámetros que inicializa los atributos con sus valores predeterminados. Sin embargo, si se define cualquier constructor, el constructor por defecto no se generará automáticamente.

## 2.3. Modificadores de Acceso

### 2.3.1. Tipos de Modificadores de Acceso

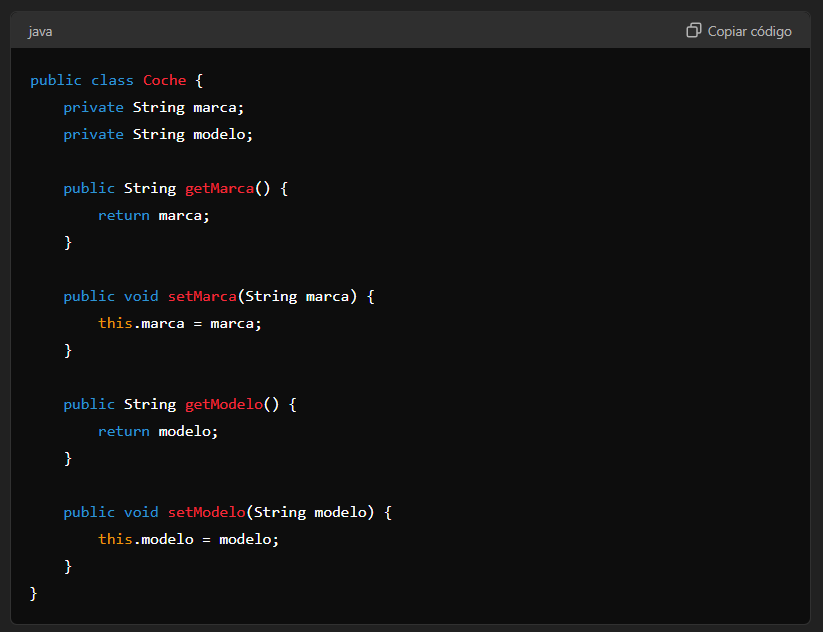
Java proporciona varios ***modificadores de acceso*** que controlan la visibilidad de los atributos y métodos dentro y fuera de la clase:

* **public**: El miembro es accesible desde cualquier otra clase.
* **private**: El miembro es accesible solo dentro de la clase en la que se define.
* **protected**: El miembro es accesible dentro del mismo paquete y en subclases.
* **default** (sin modificador): El miembro es accesible solo dentro del mismo paquete.

### 2.3.2. Encapsulamiento mediante Modificadores de Acceso

El ***encapsulamiento*** se logra utilizando modificadores de acceso para restringir el acceso directo a los atributos de una clase. Los atributos se declaran como private y se accede a ellos mediante métodos públicos get y set.

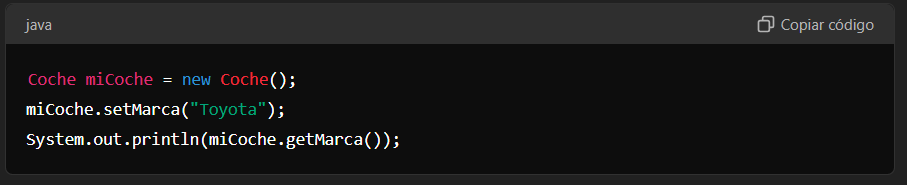
* Ejemplo: **Encapsulación** en la clase **Coche**.



### 2.3.3. Acceso a Atributos y Métodos

Dependiendo del modificador de acceso utilizado, los atributos y métodos pueden ser accedidos directamente desde otras clases o deben ser accedidos a través de métodos públicos.

* **Ejemplo**: Acceso a atributos privados a través de métodos públicos.

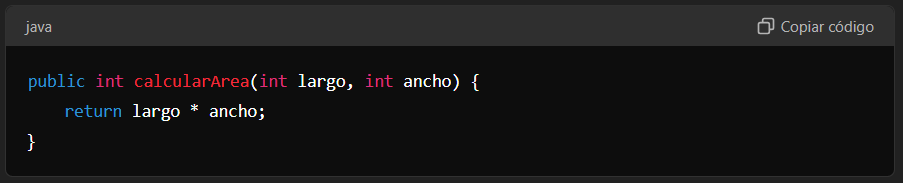


## 2.4. Métodos en Java

### 2.4.1. Definición de Métodos

Un ***método*** es una función definida dentro de una clase que realiza una tarea específica. Los métodos pueden aceptar parámetros y devolver valores. La firma de un método incluye su nombre, tipo de retorno, y lista de parámetros.

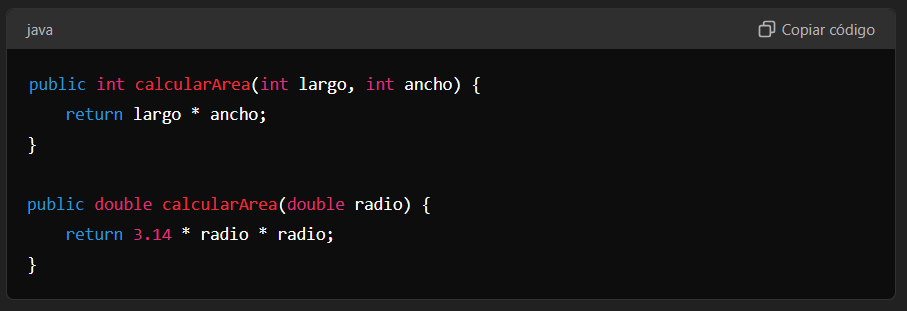
* **Ejemplo**: Método para calcular el área de un rectángulo.



### 2.4.2. Sobrecarga de Métodos

La ***sobrecarga de métodos*** permite definir múltiples métodos con el mismo nombre en la misma clase, siempre que tengan diferentes listas de parámetros. Esto proporciona flexibilidad para realizar la misma operación de diferentes maneras.

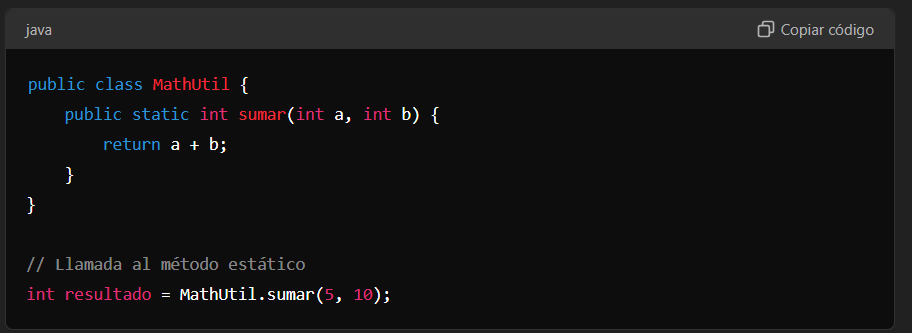
* Ejemplo: **Sobrecarga** del método **calcularArea**.



### 2.4.3. Métodos Estáticos

Los ***métodos estáticos*** pertenecen a la clase en lugar de a una instancia específica de la clase. Se pueden llamar sin crear un objeto de la clase. Los métodos estáticos se definen utilizando la palabra clave static.

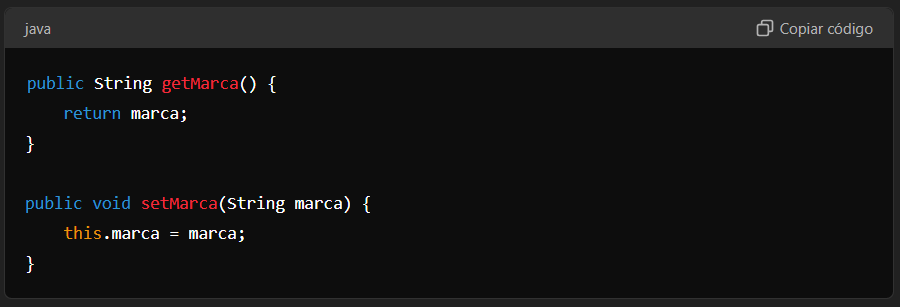
* Ejemplo: **Método estático** en la clase **Math**.



### 2.4.4. Métodos get y set

Los ***métodos get*** y ***set*** son métodos convencionales en Java que se utilizan para acceder y modificar los valores de los atributos privados de una clase. Siguen la convención de nombrar los métodos get con el prefijo "get" seguido del nombre del atributo, y los métodos set con el prefijo "set".

* Ejemplo: **Métodos get** y **set** en la clase **Coche**.

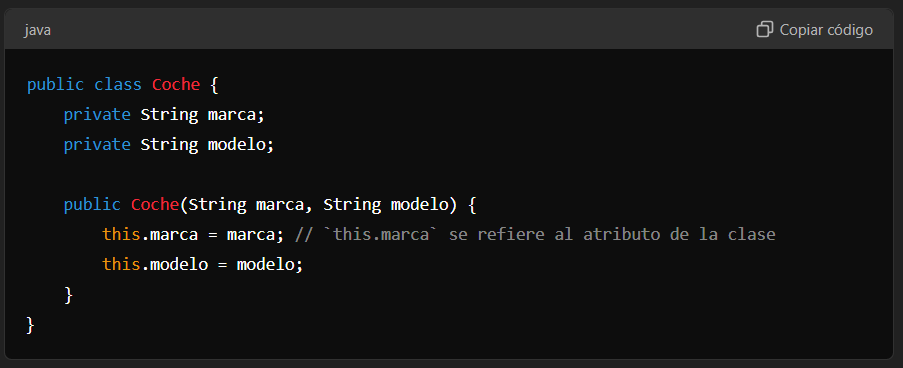


## 2.5. this en Java

### 2.5.1. Uso de this

La palabra clave ***this*** se utiliza dentro de un método o constructor para referirse al objeto actual. Es útil cuando los nombres de los parámetros y los atributos de la clase son los mismos, para diferenciarlos.

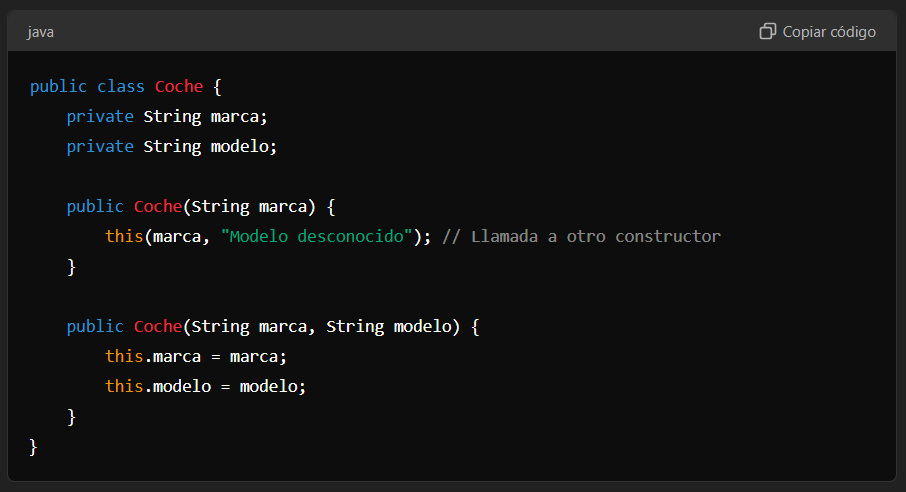
* Ejemplo: Uso de **this** en un constructor.



### 2.5.2. Referencia al Objeto Actual

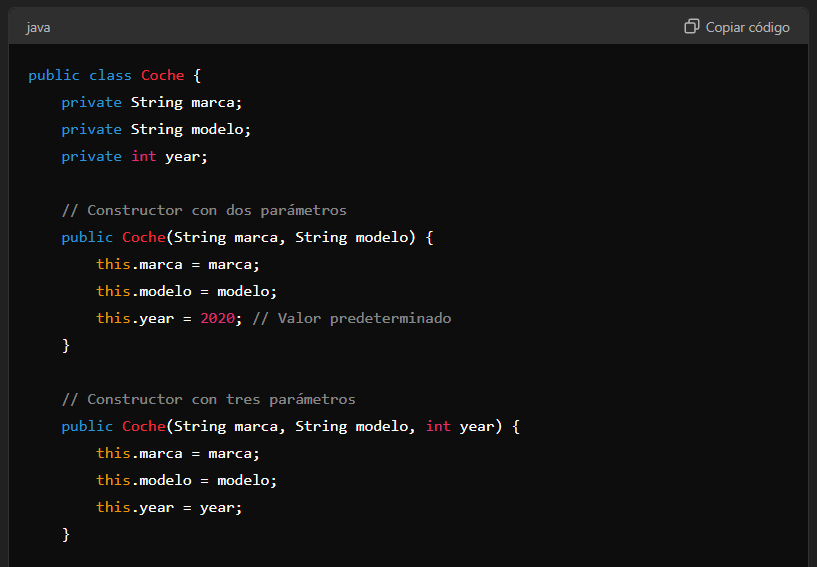
***this*** también se puede utilizar para llamar a otros constructores de la misma clase, facilitando la reutilización de código.

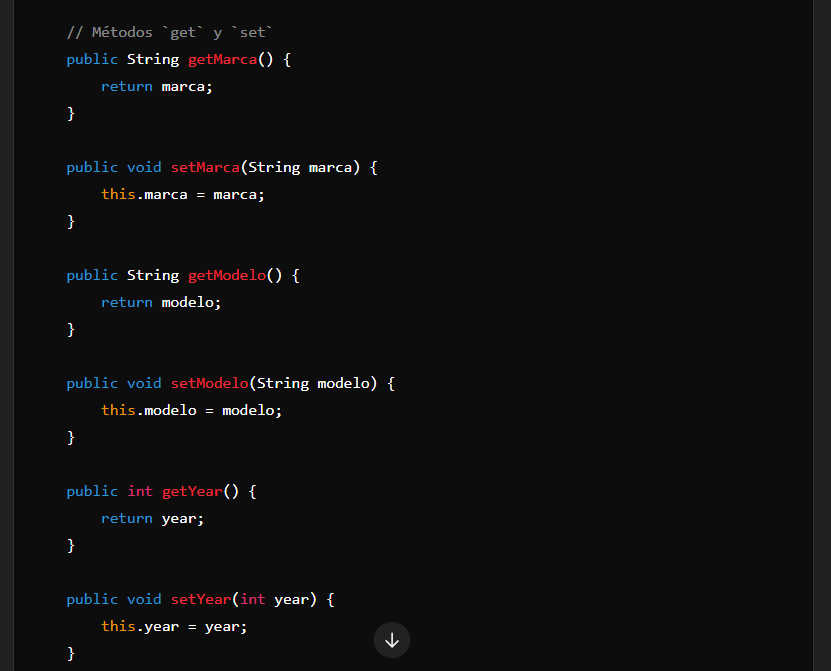
* Ejemplo: Uso de **this** para llamar a otro constructor.

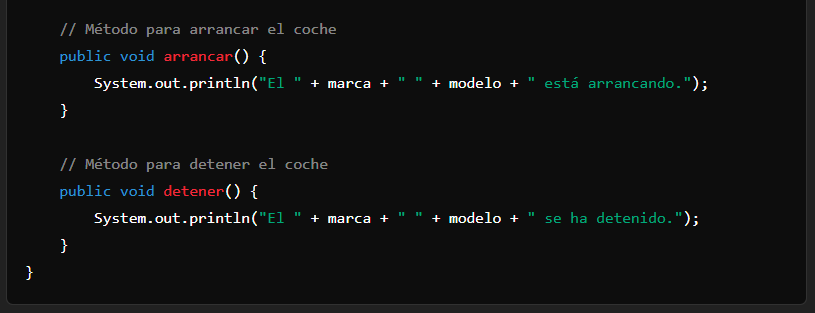


## 2.6. Ejemplo Práctico: Clase Coche Completa

Para consolidar lo aprendido, a continuación, se muestra un ejemplo práctico de una clase Coche completa que incorpora atributos, métodos, constructores, y el uso de this.







## 2.7. Conclusión

En este capítulo, hemos explorado los conceptos fundamentales de clases y objetos en Java, incluyendo la creación de clases, el uso de constructores, y la importancia de los modificadores de acceso para encapsular los datos. También hemos aprendido sobre métodos, sobrecarga de métodos, y el uso de la palabra clave this. En el siguiente capítulo, profundizaremos en la herencia y cómo se implementa en Java para crear relaciones jerárquicas entre clases.

## 2.8. Ejercitación

Ejercitación: [Ejercicios POO - Java](https://docs.google.com/document/d/1wDc4fLgC6CKWfYTqRkLHt7UKVX1ZARWjER-CqmcmA88/edit?usp=sharing)