Programación Orientada a Objetos

¡Bienvenidos al fascinante mundo de la Programación Orientada a Objetos! En esta presentación, exploraremos los fundamentos, pilares y conceptos clave que te permitirán dominar este paradigma esencial. Prepárate para construir aplicaciones más modulares, reutilizables y fáciles de mantener. Descubre cómo la POO transforma la forma en que abordamos el desarrollo de software, permitiéndonos crear soluciones más eficientes y escalables.

por Cinthia Rigoni

¿Qué es la POO?

La Programación Orientada a Objetos (POO) es un paradigma que se centra en la creación y manipulación de "objetos". Estos objetos son unidades autocontenidas que combinan datos (atributos) y comportamientos (métodos) en una sola entidad.

En la POO, los objetos se construyen a partir de clases, que actúan como plantillas o planos. Una clase define la estructura y el comportamiento que compartirán todos los objetos creados a partir de ella. Cada objeto es una instancia única de su clase, con sus propios valores de atributos.

Los Pilares de la POO

1 Abstracción

Simplifica la realidad, enfocándose en lo esencial e ignorando detalles irrelevantes. Permite crear modelos generales sin especificar la implementación.

2 Encapsulamiento

Oculta los detalles internos de un objeto, protegiendo su estado y asegurando la integridad de los datos. Controla el acceso a los atributos y métodos.

3 Herencia

Permite que una clase herede características y comportamientos de otra, promoviendo la reutilización de código y la creación de jerarquías.

4 Polimorfismo

Permite que un objeto se comporte de diferentes maneras según el contexto, facilitando la flexibilidad y la adaptabilidad del código.

Abstracción en Detalle

Simplificación del Mundo Real

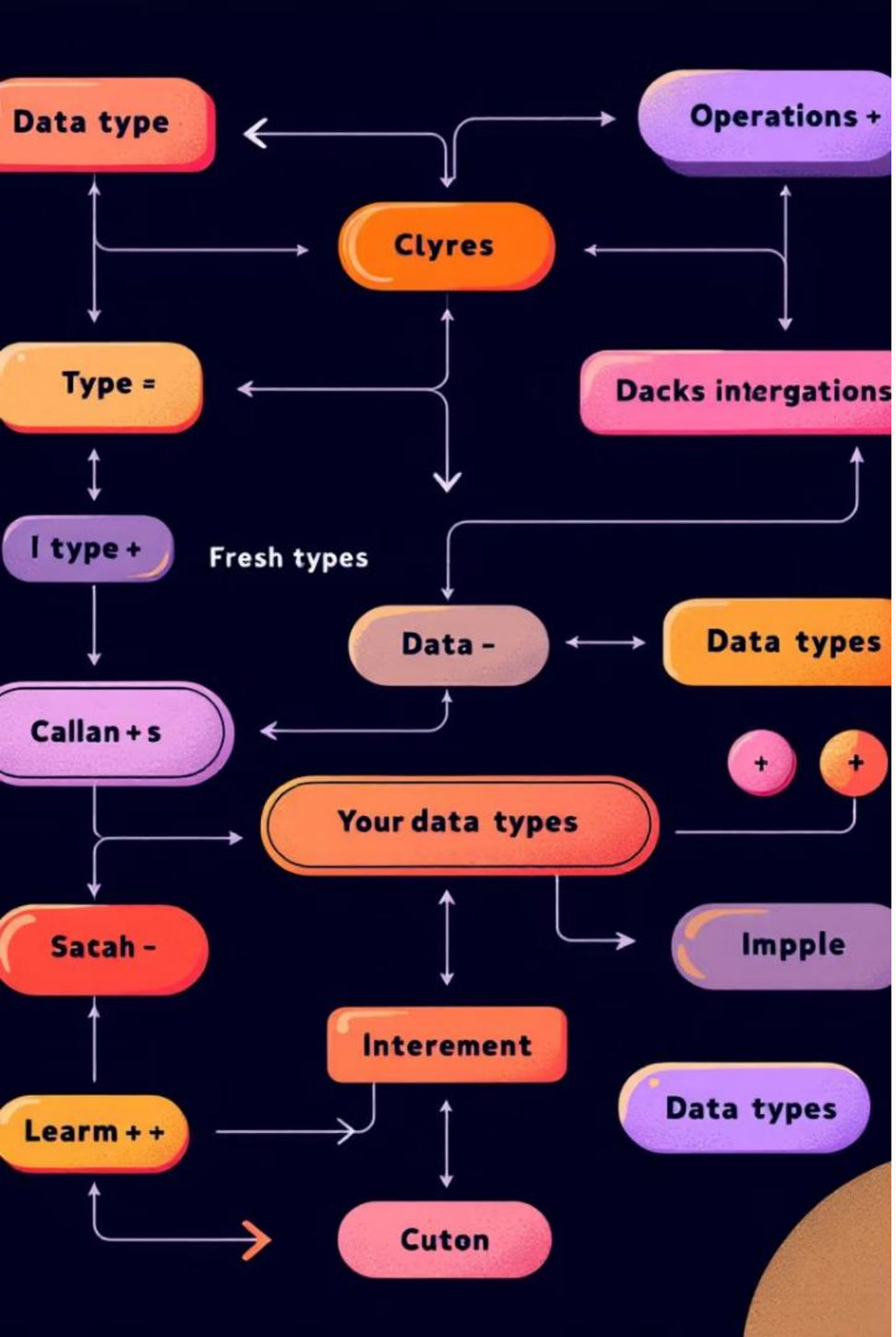
Convierte la complejidad del mundo en modelos manejables, representando solo la información esencial.

Modelado de Conceptos

Representa conceptos del mundo real como objetos, definiendo interfaces y comportamientos comunes.

Clases Abstractas

Define plantillas con métodos y propiedades comunes, permitiendo la creación de clases concretas con implementaciones específicas.



Tipos de Datos Abstractos (TAD)

1

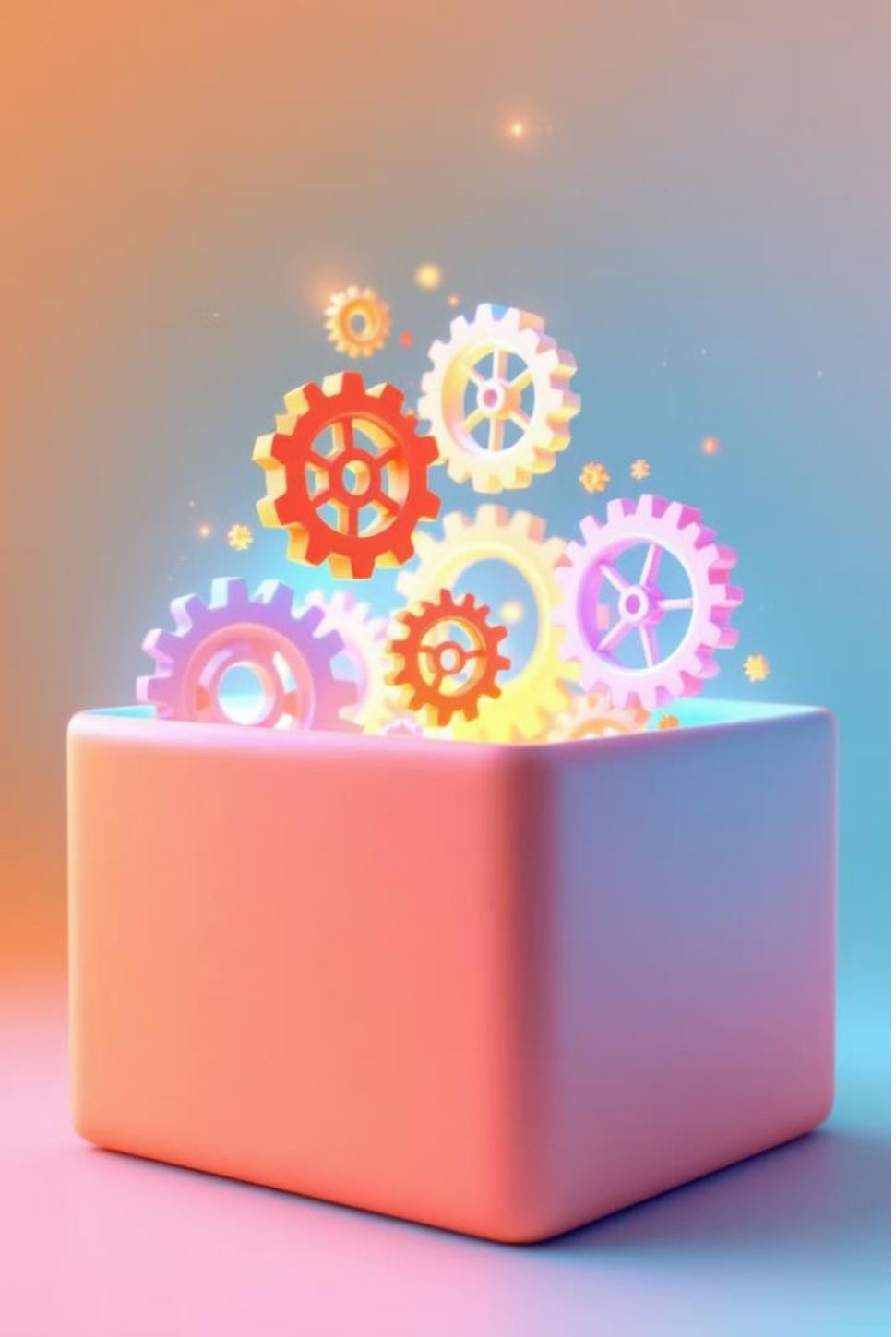
Un Tipo de Dato Abstracto (TDA) es un modelo que define los valores y las operaciones que se pueden realizar sobre ellos. La clave es que el usuario no necesita conocer los detalles internos de la representación o implementación.

2

El TDA provee un grado de abstracción que permite desacoplar el código que lo usa del código que lo implementa. Esto significa que puedes cambiar la implementación del TDA sin afectar el código que lo utiliza.

3

Los TDA son fundamentales para la modularidad y la reutilización de código. Al definir interfaces claras y bien definidas, los TDA facilitan la construcción de sistemas complejos a partir de componentes independientes.



Encapsulamiento y Ocultamiento

1

Separación del Qué y el Cómo

Distingue entre lo que representa un objeto (su interfaz) y cómo actúa internamente (su implementación).

2

Interfaz vs. Implementación

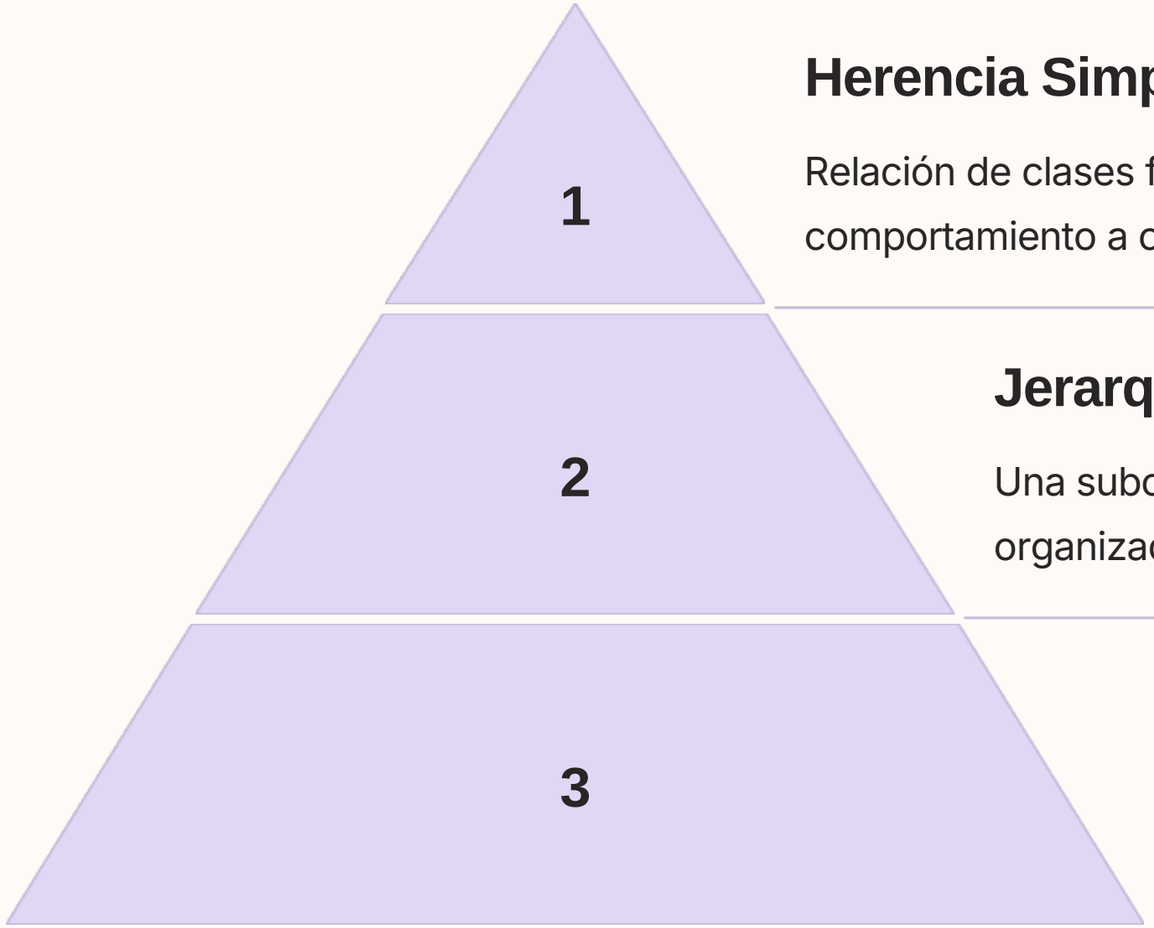
Almacena en una misma sección los elementos de una abstracción que constituyen su estructura y su comportamiento.

3

Niveles de Acceso

- Público: Acceso irrestricto.
- Protegido: Acceso dentro de la clase y subclases.
- Privado: Acceso solo dentro de la clase.

Herencia: Construyendo Jerarquías



Herencia Simple

Relación de clases fundamental en la POO. Una clase brinda estructura y comportamiento a otra.

Jerarquía de Abstracciones

Una subclase hereda de una o más superclases, creando una jerarquía organizada.

Especialización

La subclase especializa el comportamiento o la estructura más general de sus superclases.

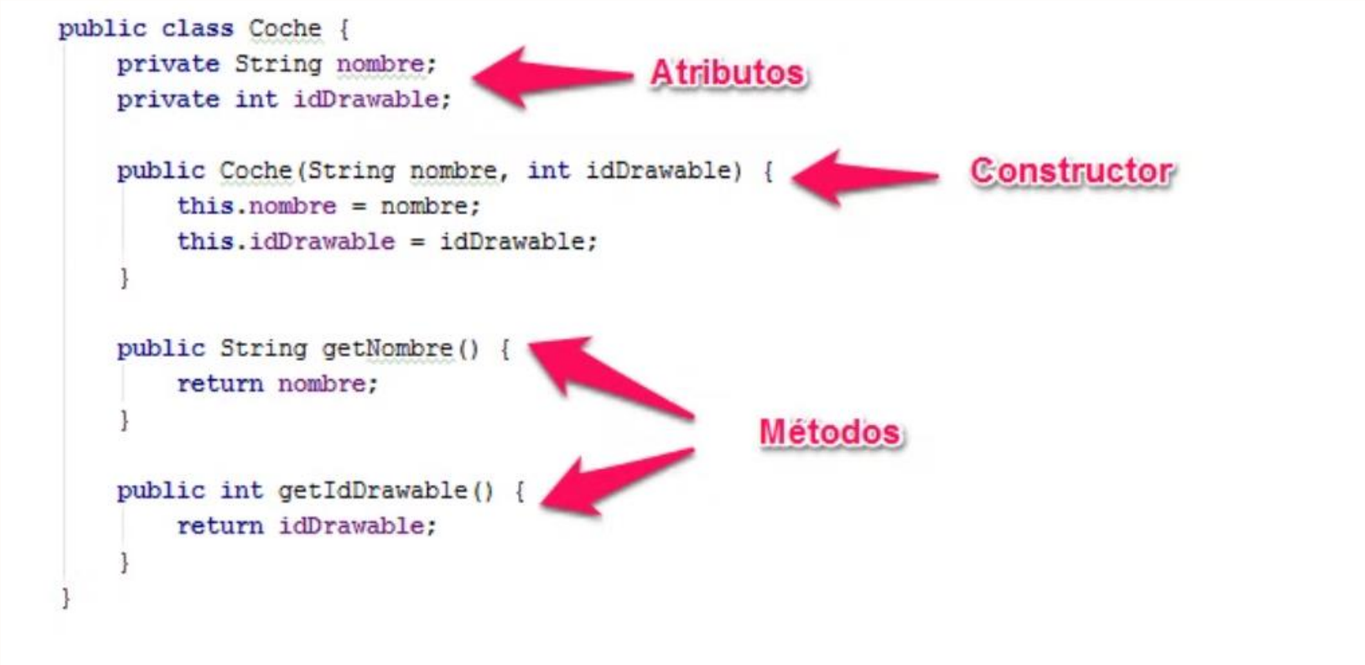
Clases en Java: El Plano de los Objetos

Definición

En Java, las clases son una parte fundamental de la POO y son la base para la creación de objetos. Una clase es un plano o modelo que define la estructura y el comportamiento de los objetos que se pueden crear a partir de ella.

Analogía

Piensa en una clase como un plano arquitectónico para una casa. Define los planos, las dimensiones y las características de la casa, pero no es la casa en sí. Los objetos son las casas reales construidas a partir de ese plano.



Conceptos Fundamentales de las Clases

Atributos

Las clases tienen atributos (variables de instancia) que representan las características o propiedades del objeto, como el nombre, la edad o el color.

Métodos

Las clases contienen métodos que representan el comportamiento del objeto. Los métodos son funciones que realizan acciones y manipulan los atributos de la clase.

Constructores

Las clases pueden tener constructores, que son métodos especiales utilizados para inicializar objetos cuando se crean.

Además, las clases implementan conceptos como encapsulación, herencia e instanciación, que son pilares de la programación orientada a objetos.

Calificadores de Acceso de Clases

1 Public

Una clase **public** es accesible desde cualquier parte del programa. Es decir, cualquier otra clase puede acceder a ella y utilizarla.

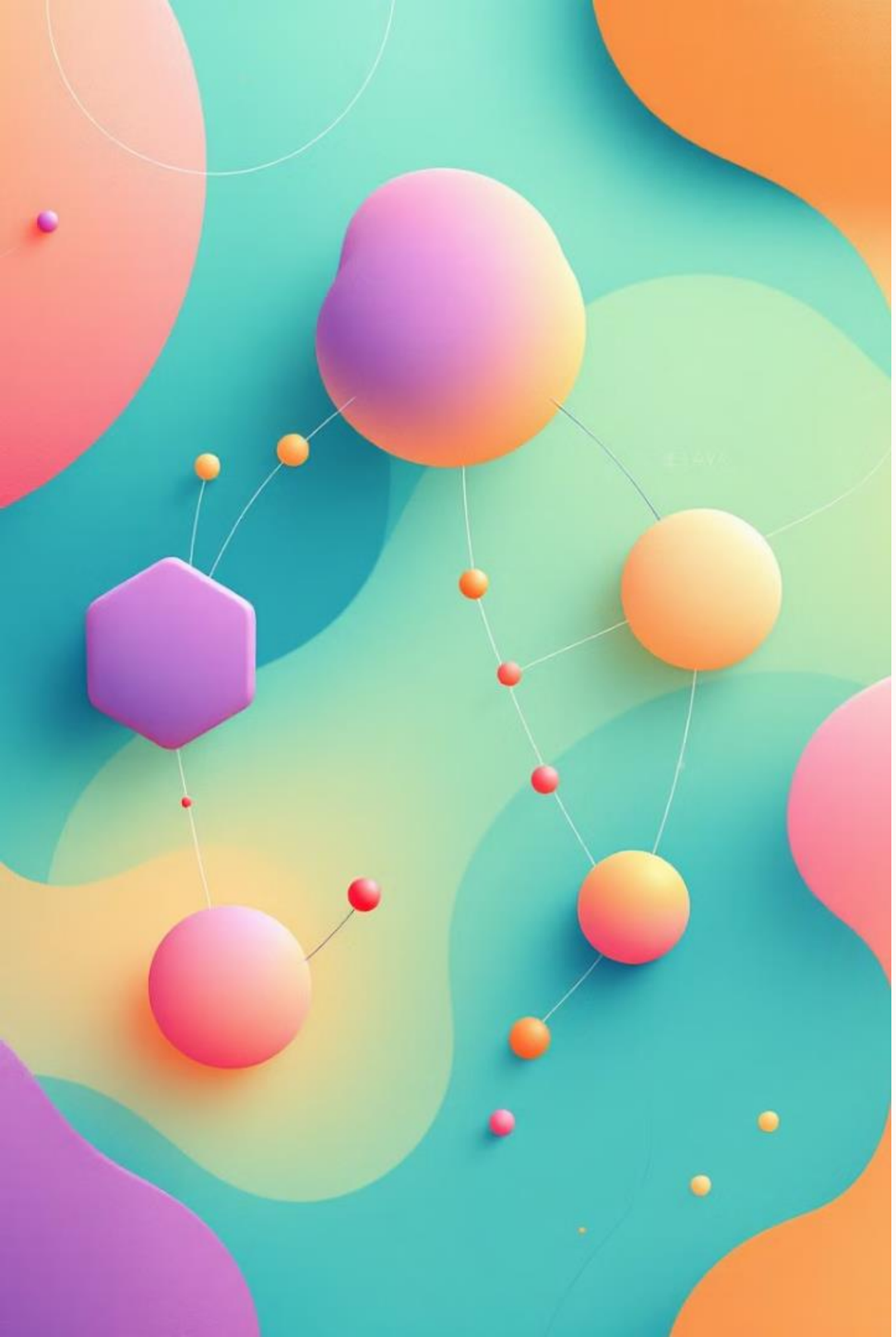
2 Abstract

Una clase **abstract** se considera incompleta y no se pueden crear instancias de ella directamente. Sirve como base para otras clases que la extienden.

3 Final

Una clase **final** no admite subclases. Esto significa que no se puede heredar de ella ni extender su funcionalidad.

Es importante destacar que una clase no puede ser a la vez **final** y **abstract**, ya que estas propiedades son mutuamente excluyentes.



Objeto: Instancia de una Clase

1

Identidad

Cada objeto tiene una identidad única que lo distingue de otros objetos, incluso si tienen el mismo estado.

2

Estado

El estado de un objeto se define por los valores de sus atributos en un momento dado. Puede cambiar a lo largo del tiempo.

3

Comportamiento

El comportamiento de un objeto se define por sus métodos, que determinan cómo interactúa con otros objetos y cómo modifica su propio estado.

Recuerda: no puedes tener un objeto sin nombre, ni un comportamiento sin un estado definido. Ambos son componentes esenciales de un objeto Java.

UML – Diagrama de Clases

1

Representación

El diagrama de clases UML es una representación gráfica de las clases en un sistema y sus relaciones entre ellas.

2

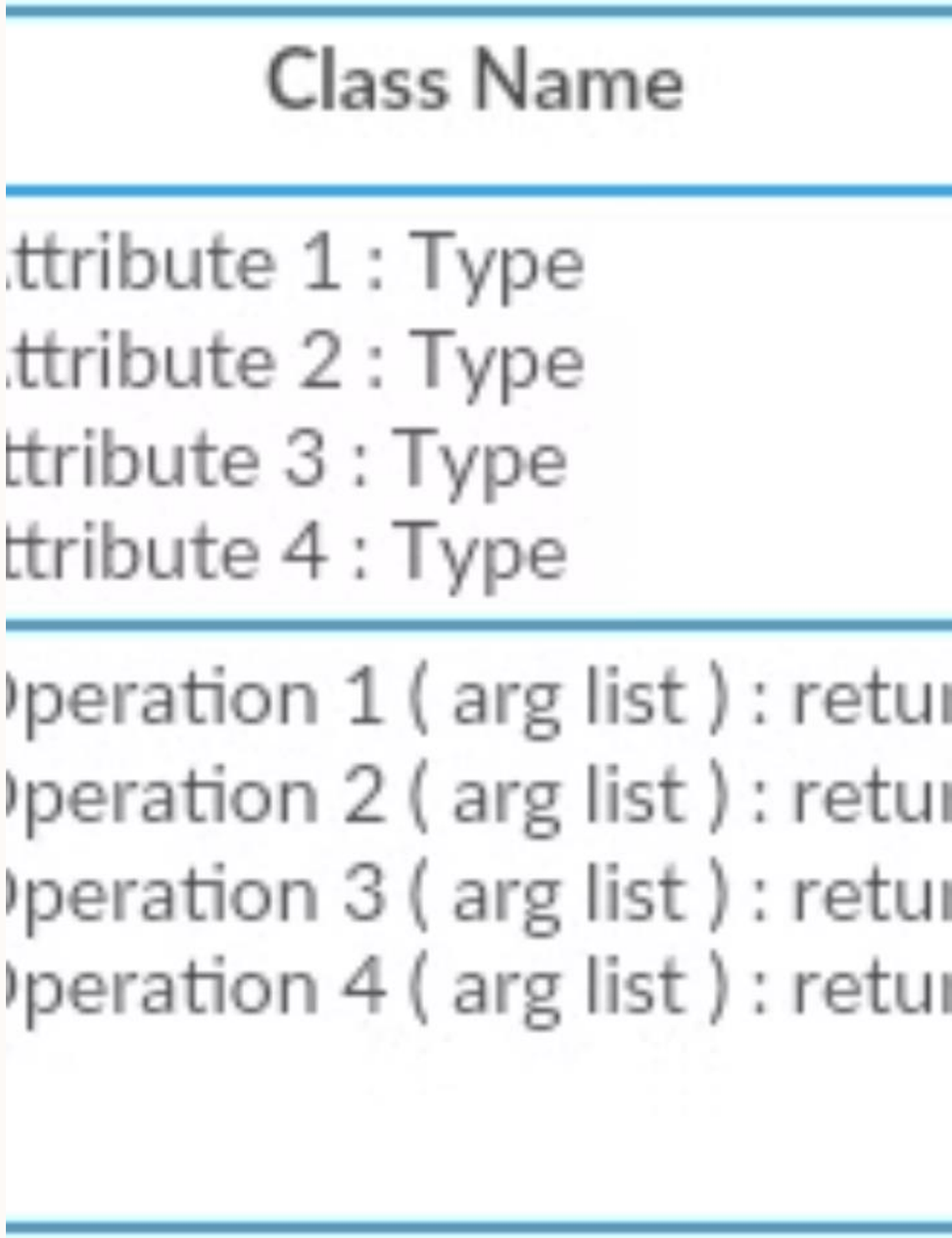
Estructura

Muestra los atributos, métodos y relaciones de herencia, asociación, agregación y composición entre las clases.

3

Beneficios

Facilita la comunicación y la comprensión del diseño del sistema, permitiendo una mejor colaboración entre los desarrolladores.



Entendiendo Static en Java

Pertenencia al Tipo

En Java, el uso de **static** indica que un miembro pertenece al tipo en sí mismo, no a una instancia del tipo. Solo se crea una instancia compartida por todas las instancias de la clase.

Uso de Memoria

El uso de **static** hace uso de la memoria una vez en el área de la clase al cargarla. No importa cuántas veces se inicialice la clase, siempre habrá una sola copia del campo definido con **static**.

La palabra clave **static** es fundamental en Java para definir miembros que pertenecen a la clase en lugar de a las instancias individuales. Esto significa que solo existe una copia de la variable o método **static**, y se comparte entre todos los objetos de esa clase. Es esencial comprender este concepto para gestionar la memoria y el comportamiento de las aplicaciones Java de manera eficiente.

Dominando Final en Java

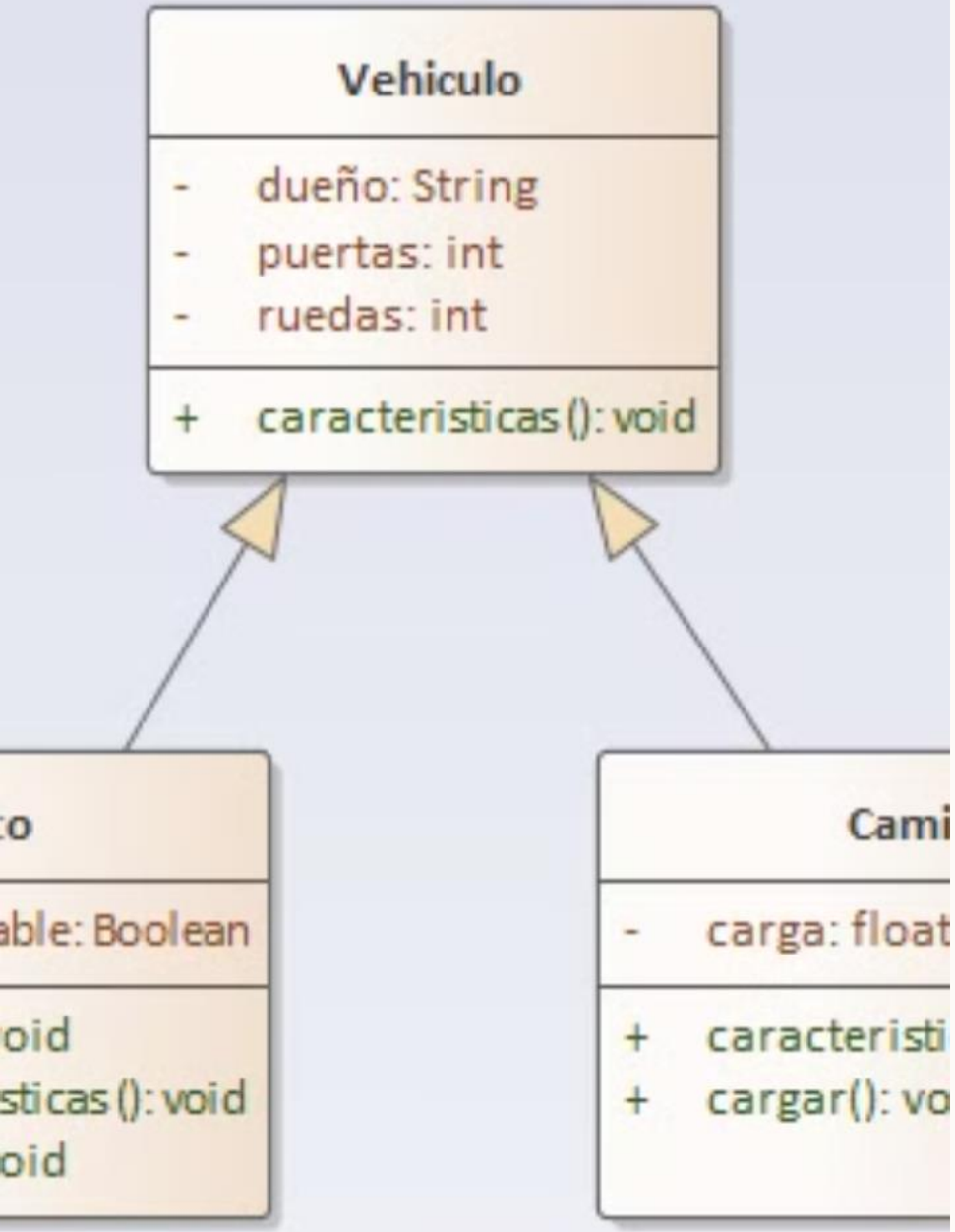
1 Variables Finales

La palabra clave **final** define constantes. Solo se puede asignar un valor una vez. Después, no se puede cambiar de ninguna forma. Esto garantiza la inmutabilidad del dato.

2 Métodos Finales

Un método **final** no puede ser redefinido por una clase hija. Si una clase hija hereda un método **final**, no puede cambiar su definición.

La palabra clave **final** es crucial para controlar la mutabilidad en Java. Al declarar una variable como **final**, se impide que su valor cambie después de la inicialización. En el caso de los métodos, **final** evita que las clases hijas los redefinan, lo que garantiza un comportamiento consistente y predecible en la jerarquía de clases.



Ejercicio Práctico

1

Objetivo

Analiza el código proporcionado e identifica el uso de **static** y **final**.

2

Desafío

Modifica el código para experimentar con los conceptos aprendidos. Intenta cambiar el valor de una variable **final** o redefinir un método **final**.

3

Reflexión

¿Qué errores o advertencias obtuviste? ¿Cómo se comportó el programa? Comprende las restricciones impuestas por **static** y **final**.

Para consolidar tu comprensión de **static** y **final**, te propongo un ejercicio práctico. Observa el código proporcionado, identifica dónde se utilizan estas palabras clave y experimenta modificando el código. Intenta cambiar el valor de una variable declarada como **final** o redefinir un método **final** en una subclase. Analiza los resultados y reflexiona sobre las restricciones impuestas por estas palabras clave.