
PROMETEO
PROGRAMACIÓN
Introducción a la orientación a objetos

Índice

01

Clases

¿Qué es una clase?

En Java, una clase es una plantilla o modelo a partir de la cual se crean objetos. Define los atributos (características) y métodos (comportamientos) que tendrán los objetos basados en esa clase.

```
public class NombreClase {

    // Atributos

    // Constructor

    // Getters y Setters

    // Métodos

}
```

01

Clases

Atributos

Los **atributos** son las variables que almacenan el estado de un objeto. Se definen dentro de una clase y pueden ser de diferentes tipos: primitivos (int, double, boolean) o referencias (como objetos de otras clases).

Características de los atributos:

- **Alcance**: Determinado por el nivel de visibilidad (public, private, etc.).
- **Tipo**: Puede ser primitivo o referenciado.
- **Valor inicial**: Puede asignarse al declararlo o mediante el constructor.

01

Clases

Constructores

Un **constructor** es un método especial que se ejecuta automáticamente cuando se crea un objeto. Su propósito principal es inicializar los atributos de la clase.

En Java, un constructor:

- Tiene el mismo nombre que la clase.
- No tiene un tipo de retorno.
- Puede sobrecargarse para aceptar diferentes conjuntos de parámetros.

Sobrecarga de constructores

Puedes definir múltiples constructores con diferentes parámetros para inicializar objetos de varias formas.

01

Clases

Getters y Setters

Los getters y setters son métodos especiales que permiten acceder y modificar los atributos de una clase mientras mantienen la encapsulación. Esto protege los datos internos del objeto y asegura que los cambios se realicen de forma controlada.

Métodos

Los **métodos** son funciones que definen el comportamiento de los objetos. Pueden realizar tareas específicas, modificar atributos o devolver información.

Tipos de métodos:

1. **Métodos de instancia**: Se ejecutan sobre un objeto en particular.
2. **Métodos estáticos**: Pertenecen a la clase y no requieren un objeto para ejecutarse.
3. **Constructores**: Métodos especiales que inicializan los objetos.

01

Clases

Métodos de Instancia

Los **métodos de instancia** se ejecutan sobre objetos específicos, mientras que los **métodos estáticos** pertenecen a la clase y no requieren un objeto.

Métodos de instancia

- Se utilizan cuando el método depende de los atributos del objeto.

Métodos estáticos

- Se utilizan para operaciones que no dependen de atributos específicos de objetos.

02

Objetos

¿Qué es un Objeto?

Un objeto es una **entidad concreta** que combina datos y comportamientos. Es una representación de algo del mundo real o abstracto, modelado mediante una clase. En Java, un objeto es una instancia de una clase.

- **Ejemplo del mundo real**: Un coche tiene:
  - **Estado**: Su marca, modelo, color, nivel de combustible.
  - **Comportamiento**: Puede acelerar, frenar, girar.
  - **Identidad**: Cada coche tiene un número de serie único.

02

Objetos

Estado

El estado de un objeto se define mediante sus atributos. Es el conjunto de valores que describen al objeto en un momento dado. El estado puede cambiar durante la ejecución del programa mediante la interacción con sus métodos.

Los métodos mutadores (como encender o apagar) son responsables de cambiar el estado del objeto. Es una buena práctica usar métodos en lugar de acceder directamente a los atributos.

Comportamiento

El comportamiento de un objeto define lo que puede hacer o cómo puede interactuar con otros objetos. En Java, el comportamiento se implementa mediante métodos.

02

Objetos

Identidad

Dos objetos pueden tener el mismo estado y comportamiento, pero son **diferentes** debido a su identidad. En Java, la identidad está asociada con la dirección de memoria.

```
Persona persona1 = new Persona("Carlos", 30);

Persona persona2 = new Persona("Carlos", 30);
```

Explicación

- Aunque persona1 y persona2 tienen el mismo estado (nombre y edad), son **objetos diferentes** porque ocupan direcciones distintas en memoria.

03

Encapsulado

¿Qué es el Encapsulado?

El encapsulado permite:

1. **Ocultar detalles internos** de la implementación de una clase.
2. Proveer una **interfaz controlada** para acceder y modificar sus datos.
3. **Proteger la integridad** de los atributos, evitando que sean manipulados directamente de manera no controlada.

03

Encapsulado

Visibilidad

La **visibilidad** define quién puede acceder a los atributos y métodos de una clase. Esto se controla mediante modificadores como public, private, protected y el modificador por defecto (**default**).

Modificadores de visibilidad en Java:

1. **public**: Accesible desde cualquier clase.
2. **private**: Accesible sólo dentro de la clase en la que se define.
3. **protected**: Accesible desde clases del mismo paquete y subclases.
4. **Paquete (default)**: Accesible sólo dentro del mismo paquete.

03

Encapsulado

Encapsulación con Atributos Privados y Métodos Públicos

1. La solución es **hacer privados** los atributos y proporcionar métodos públicos para interactuar con ellos.

Ventajas del Encapsulado

1. **Protección de datos**:
   - Los datos sensibles se ocultan y no pueden ser manipulados directamente.
2. **Control de acceso**:
   - Los métodos permiten validar los datos antes de asignarlos.
3. **Mantenimiento y flexibilidad**:
   - Cambiar la implementación interna no afecta el resto del código si la interfaz pública se mantiene.

04

Relaciones entre clases

En Java, las relaciones entre clases permiten modelar cómo interactúan las entidades dentro de un programa. Estas relaciones son fundamentales en la programación orientada a objetos (POO) y se utilizan para construir sistemas complejos de manera estructurada.

Tipos de Relaciones entre Clases

1. **Asociación**: Representa una relación lógica entre dos clases.
    - Puede ser **unidireccional** o **bidireccional**.
2. **Agregación**: Una relación más fuerte donde una clase contiene a otra como parte de su estructura, pero los objetos tienen ciclos de vida independientes.
3. **Composición**: Una relación de dependencia más fuerte donde una clase contiene otra, y ambas tienen el mismo ciclo de vida.
4. **Herencia**: Una clase (subclase) hereda atributos y métodos de otra clase (superclase).

5. **Dependencia**: Una clase utiliza otra para realizar una acción específica.

05

Principios básicos POO

La programación orientada a objetos (POO) se basa en cuatro principios fundamentales que permiten estructurar programas de forma lógica, reutilizable y escalable. Estos principios son:

1. **Abstracción**
2. **Encapsulación**
3. **Herencia**
4. **Polimorfismo**

Cada uno de estos conceptos se enfoca en diferentes aspectos del diseño y desarrollo de software, y juntos constituyen el núcleo de la POO.

05

Principios básicos POO

Abstracción

La **abstracción** consiste en modelar entidades del mundo real seleccionando únicamente las características esenciales que son relevantes para el programa, ignorando los detalles irrelevantes.

Ejemplo práctico: Vehículos

1. **Clase abstracta**: Define los métodos y atributos generales que todas las subclases deben implementar o heredar.
2. **Subclases**: Implementan o personalizar la funcionalidad según el tipo específico de objeto.

05

Principios básicos POO

Encapsulación

El **encapsulamiento** se refiere a proteger los datos internos de una clase restringiendo su acceso directo desde fuera de la clase, como se explicó anteriormente. Esto permite validar y controlar cómo se interactúa con los atributos.

Ventajas:

1. **Oculta la implementación interna**.
2. **Proporciona seguridad** al proteger los datos de manipulación directa.
3. Facilita el **mantenimiento** del código.

05

Principios básicos POO

Herencia

1. La **herencia** permite que una clase (subclase) reutilice atributos y métodos de otra clase (superclase). Esto promueve la reutilización de código y facilita la extensión de funcionalidades.

Polimorfismo

El **polimorfismo** permite que un objeto tome diferentes formas, es decir, un objeto de una subclase puede ser tratado como un objeto de su superclase. Esto se logra mediante:

1. **Sobrecarga de métodos**: Métodos con el mismo nombre pero diferentes parámetros.
2. **Sobreescritura de métodos**: Una subclase redefine un método de la superclase.


PROMETEO