

# Patrón de Comportamiento: Visitor

- Alfonso Jiménez Rivas
- Sebastián Soto Ángel



# Definición del Patrón Visitor

## Propósito

Patrón de comportamiento que permite separar una operación de la estructura de datos sobre la cual opera. De este modo, se pueden añadir nuevas operaciones a objetos ya existentes sin modificar sus clases.

## En Términos Sencillos

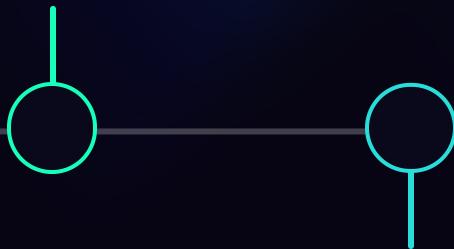
1. Cada objeto “acepta” (accept) un visitante.
2. El visitante “visita” al objeto y realiza la operación deseada.



# Analogía: Museo

## Colección de Objetos (Museo)

- Usuario, Producto y Pedido.



## Distintas Operaciones sobre Ellos

- Generar un reporte, calcular impuestos, exportarlos a JSON, XML.

## Guía ("Visitante")

- Hace un recorrido por cada exhibición y explica algo diferente según el tipo de exhibición que ve (Método).

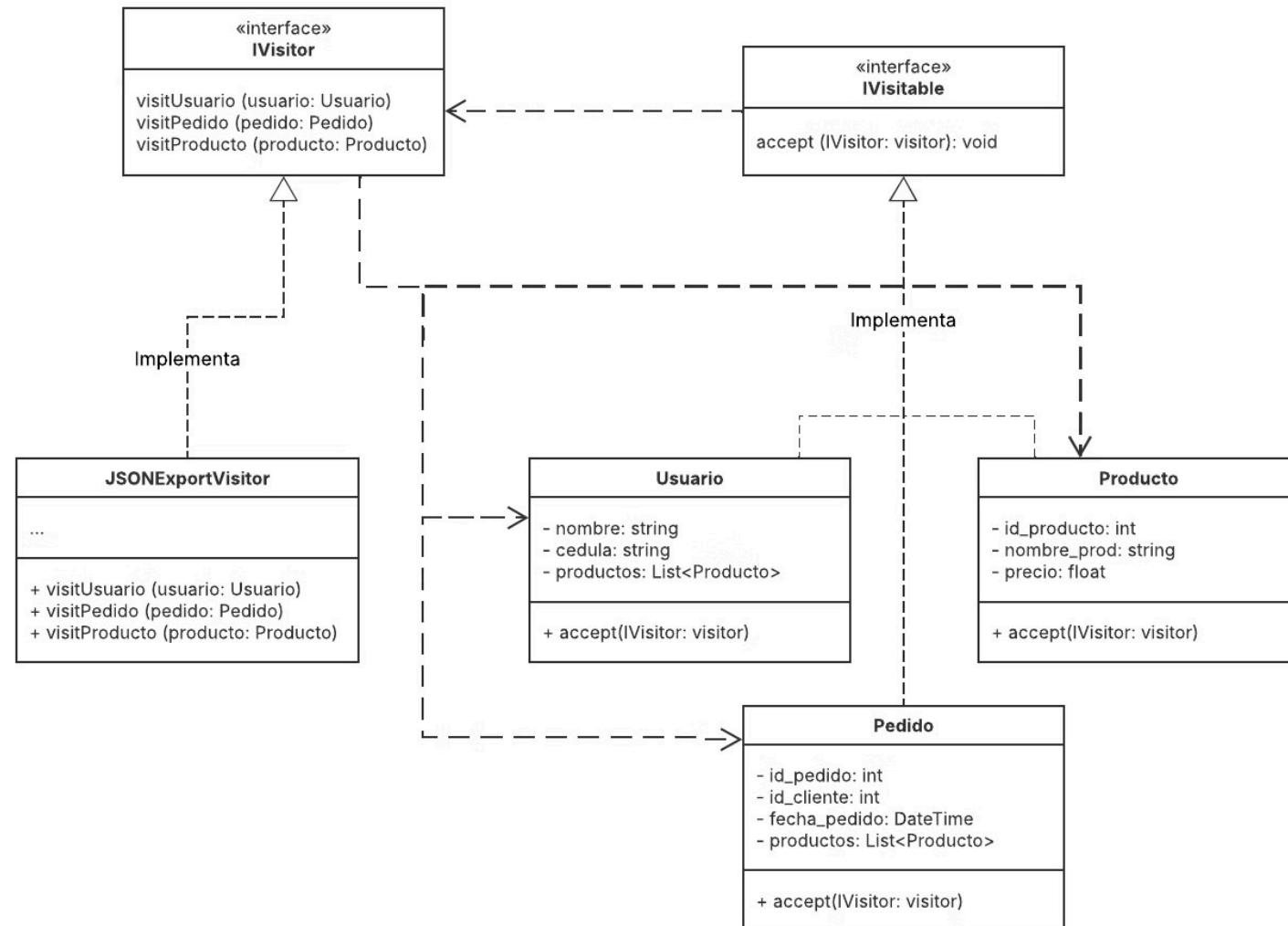
De este modo, si mañana traes otro guía (otro visitante) que hace un recorrido distinto (otra operación), las exhibiciones siguen igual, no tienes que tocarlas.



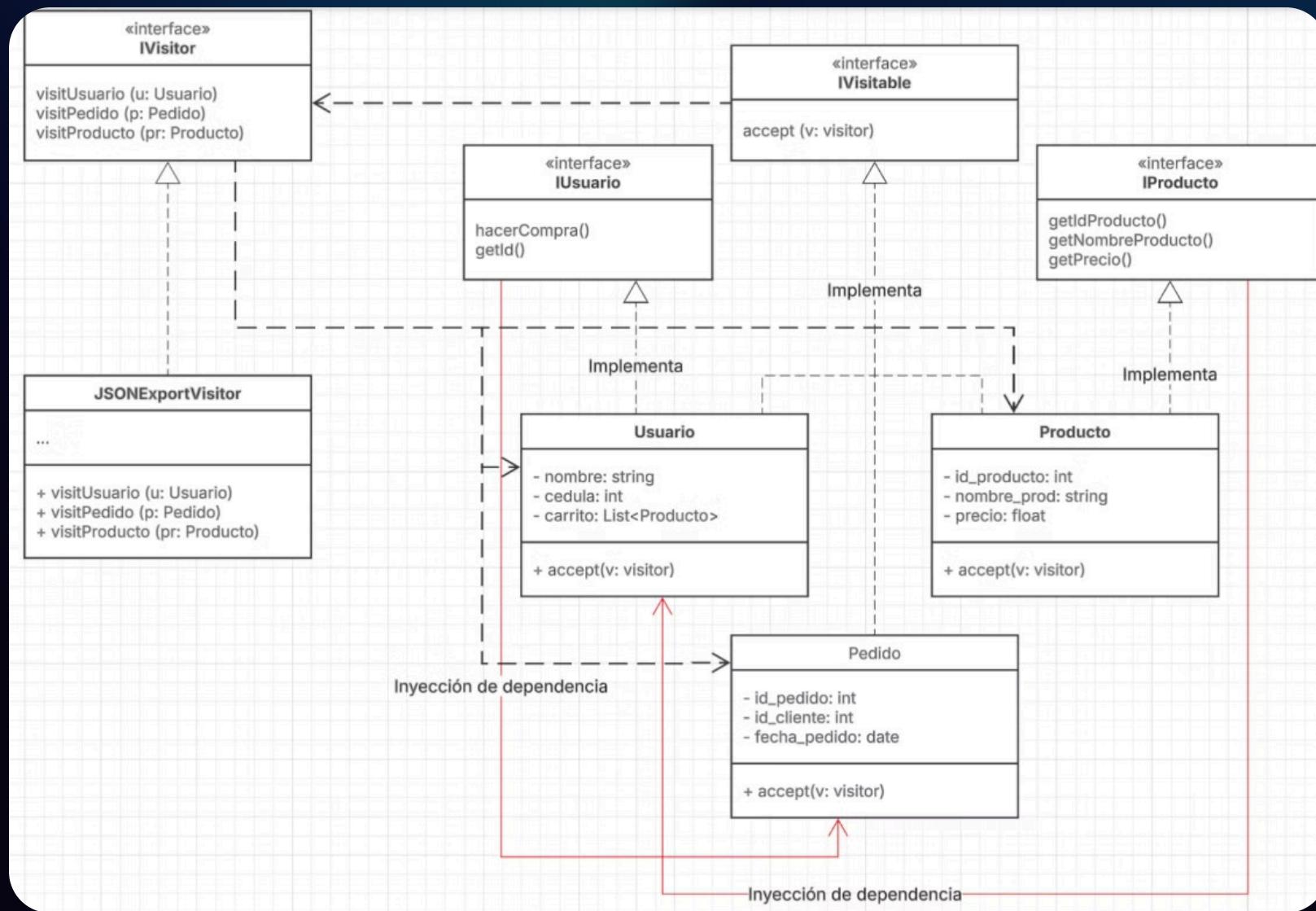
## Recibir al Guía Accept(Visitor)

- Cada exhibición sabe cómo "recibir" al guía, pero la **explicación concreta** (la operación que se hace) está en el guía, no en la exhibición

# Diagrama UML de Visitor



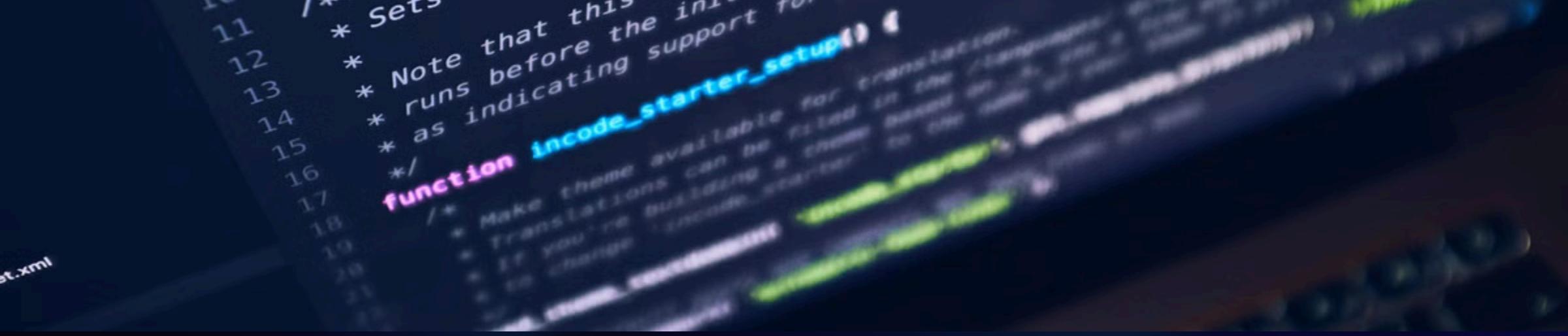
# Diagrama UML de Visitor



## Estructura General de Clases

- **IVisitor**: Declara las operaciones de visita para cada tipo de objeto "visitable".
- **IVisitable**: Declara el método `accept(IVisitor visitor)` para permitir que un visitante acceda al objeto.
- **ConcreteElementX**: Implementa el método `accept` para llamar al método de visita correspondiente.
- **ConcreteVisitor**: Implementa las operaciones específicas que se quieren realizar sobre los distintos tipos de elementos.

# Implementación en Código de Visitor



# Ventajas Clave del Patrón Visitor

## Principio de Abierto/Cerrado

Introducir un nuevo comportamiento que puede funcionar con objetos de clases diferentes sin cambiar esas clases.

## Principio de Responsabilidad única

Se pueden tomar varias versiones del mismo comportamiento y ponerlas en la misma clase.

## Concentración de Lógica

Se evitan “ifs” o “switches” repetidos en cada clase para cada operación.

## Acumulación de Información

Un objeto visitante puede acumular cierta información útil mientras trabaja con varios objetos.

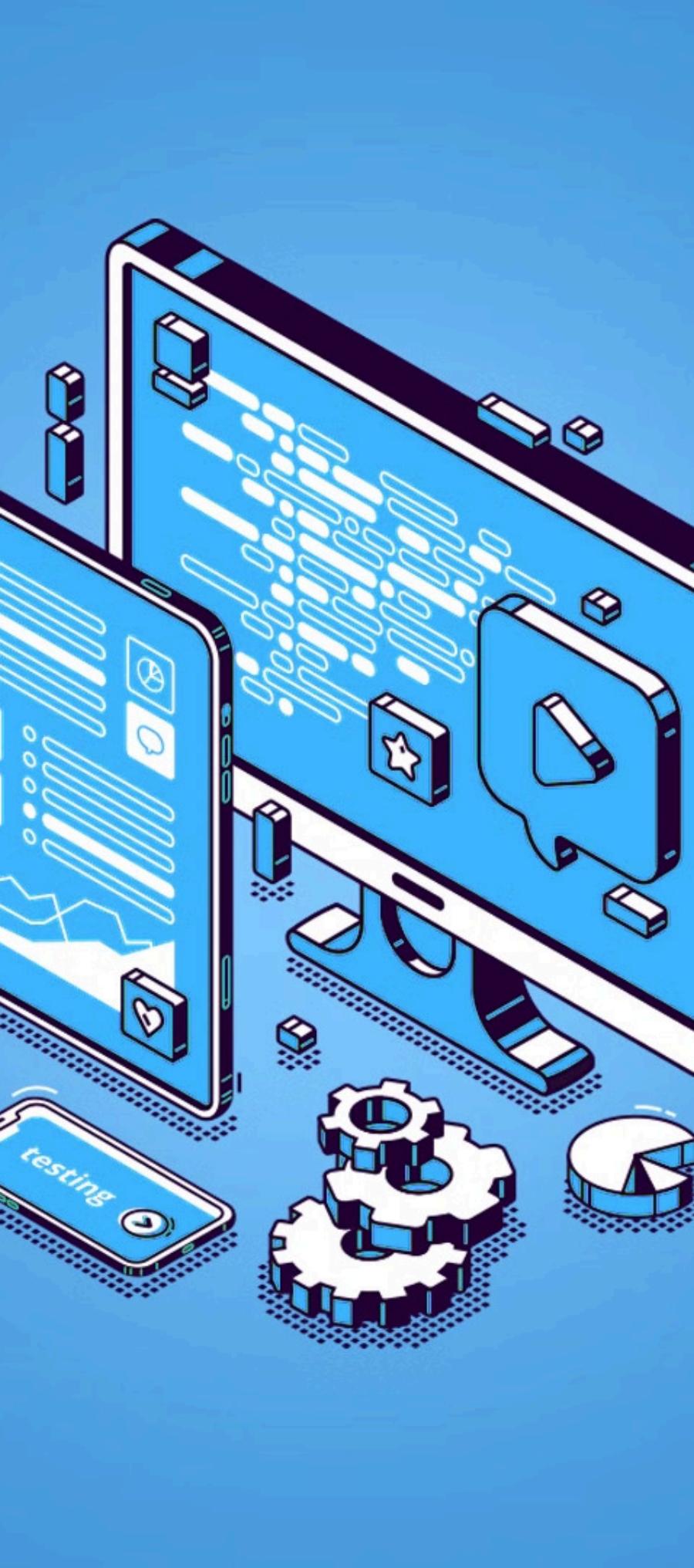
# Desventajas del Patrón Visitor

## Dificultad al Añadir Nuevas Clases de Elementos

Si se agregan nuevos tipos de elementos (nuevas clases "visitables"), hay que actualizar la interfaz `IVisitor` y todos los visitantes concretos.

## Puede Romper la Encapsulación

El visitante, para operar, necesita a menudo acceso a detalles internos de las clases visitadas.



# ¿Cuándo un Arquitecto de Software Debería de usar Visitor?

- 1 Cuando se necesita realizar una operación sobre todos los elementos de una compleja estructura de objetos.
- 2 Cuando necesita limpiar la lógica de negocio de comportamientos auxiliares. (Single Responsibility)
- 3 Cuando un comportamiento solo tenga sentido en algunas clases de una jerarquía de clases pero no en otras.
- 4 Cuando se necesite agregar operaciones (métodos) a una jerarquía de clases sin modificarlas directamente.
- 5 Tienes múltiples clases con estructuras internas diferentes, pero requieres aplicar un comportamiento homogéneo o similar en cada una (por ejemplo, para reportes, estadísticas, exportaciones, etc.).