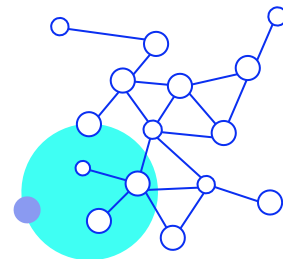


Relaciones entre objetos

Objetos conectados

Los objetos no operan solos, **colaboran activamente, entre sí**, para cumplir con sus responsabilidades y completar tareas complejas de manera eficiente.

Por ejemplo: en un videojuego, un objeto **Jugador** puede interactuar con un objeto **Enemigo** para realizar acciones como atacar, esquivar y recibir daño durante el juego.



Tipos de relaciones

Asociación

Relación básica donde **un objeto utiliza los servicios de otro** para realizar una tarea específica.



Agregación

Caso especial de asociación donde **un objeto “tiene un” otro objeto**, pero este último puede existir de manera independiente.



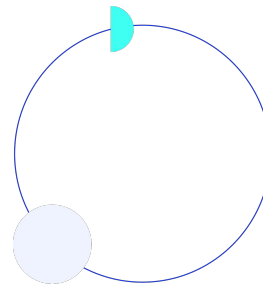
Composición

Caso especial de asociación donde **un objeto “es parte de” otro objeto** y no puede existir independientemente de él.



Dependencia (Relación de uso)

Relación donde **un objeto utiliza temporalmente los servicios de otro** objeto para llevar a cabo una tarea específica.



Asociación

Dos clases u objetos están conectados para realizar una tarea o compartir información.

La asociación permite **modelar relaciones flexibles donde los objetos colaboran entre sí para realizar tareas específicas, sin que ninguno de ellos sea parte integral o propiedad del otro.**



Por ejemplo, se considera la relación entre:

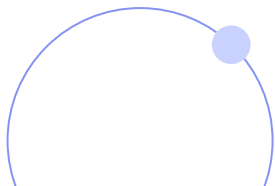
- Un **C**liente.
- Una **C**uenta.

Características clave de la asociación:

- El **C**liente utiliza una **C**uenta bancaria para acceder a los servicios financieros que proporciona.
- **No hay una dependencia** estructural fuerte; el cliente y la cuenta bancaria pueden existir de manera independiente.
- No son parte intrínseca el uno del otro.

```
public class Cliente {  
    private String nombre;  
  
    public Cliente(String nombre) {  
        this.nombre = nombre;  
    }  
  
    // Métodos y atributos ...  
}
```

```
// Clase Cuenta que tiene un cliente asociado  
public class Cuenta {  
    private Cliente cliente;  
    private double saldo;  
  
    public Cuenta(Cliente cliente) {  
        this.cliente = cliente;  
    }  
  
    public void realizarTransaccion(double monto) {  
        saldo += monto;  
    }  
}
```



Agregación

Un objeto "tiene un" conjunto de objetos como parte de su estructura, pero estos objetos pueden existir de manera independiente fuera de la relación.

La agregación permite **modelar relaciones flexibles donde los objetos pueden asociarse temporalmente sin una dependencia estructural fuerte.**

Por ejemplo, la relación entre:

- Clase **Equipo**.
- Clase **Jugador**.

Características clave de la agregación:

- Los objetos Jugador pueden existir fuera del contexto del equipo.
- **Un equipo puede tener varios jugadores**, y los jugadores pueden cambiar de equipo **sin que se destruya su existencia**.
- No hay una dependencia fuerte entre el equipo y los jugadores; son **entidades independientes** que pueden relacionarse dinámicamente.



```
// Clase Equipo que tiene una lista de Jugadores
//(agregación)
public class Equipo {
    private String nombre;
    private List<Jugador> jugadores;

    public Equipo(String nombre) {
        this.nombre = nombre;
        this.jugadores = new ArrayList<>();
    }

    public void agregarJugador(Jugador jugador) {
        jugadores.add(jugador);
    }

    // Otros métodos según sea necesario
}
```

```
// Clase Jugador
public class Jugador {
    private String nombre;
    private int numeroCamiseta;

    public Jugador(String nombre, int
numeroCamiseta) {
        this.nombre = nombre;
        this.numeroCamiseta = numeroCamiseta;
    }

    // Otros métodos según sea necesario
}
```

Composición

Un objeto "es parte de" otro objeto y no puede existir de manera independiente fuera de esa relación.

La composición permite **modelar relaciones fuertes donde un objeto es esencial para el funcionamiento o la existencia de otro objeto**, encapsulando su comportamiento y datos de manera integral.



Se considera la relación entre:

- Un **Automóvil**.
- Su **Motor**.

Características clave de la composición:

- El objeto Motor es creado y destruido junto con el objeto Automóvil.
- Un **automóvil no puede funcionar sin un motor**; están intrínsecamente ligados.
- La vida útil del motor está directamente vinculada a la vida útil del automóvil; **no puede ser compartido**.


```
// Clase Automóvil que tiene un objeto Motor  
//(composición)
```

```
public class Automovil {  
    private String marca;  
    private Motor motor;  
  
    public Automovil(String marca) {  
        this.marca = marca;  
        // Creación del objeto Motor  
        //como parte del Automóvil  
        this.motor = new Motor();  
    }  
  
    // Métodos para operar el automóvil  
}
```

```
// Clase Motor (parte del Automóvil)
```

```
public class Motor {  
    private int cilindrada;  
    private String tipo;  
  
    public Motor() {  
        this.cilindrada = 1600;  
        this.tipo = "Diesel";  
    }  
  
    // Métodos relacionados con el motor  
}
```

Dependencia (Relación de uso)

La dependencia de uso es un tipo de relación donde **un objeto utiliza temporalmente los servicios de otro objeto para llevar a cabo una tarea específica.**



Se considera la relación entre:

- Un **Cliente**.
- Un **ServicioCorreoElectronico**.

El **Cliente** utiliza el servicio **ServicioCorreoElectronico** para enviar correos electrónicos cuando sea necesario.

Características clave de la dependencia de uso:

- El objeto **Cliente** utiliza al otro objeto para una tarea específica.
- No hay una relación estructural permanente; **la dependencia es temporal** y basada en la necesidad de la funcionalidad.
- Los objetos **pueden existir de manera independiente** fuera de la relación de uso.

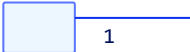
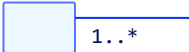

Cardinalidad

La cardinalidad de una relación describe el **número de instancias de una clase que pueden estar asociadas con una instancia de otra clase** en una relación específica:

- Uno a uno (1:1).
- Uno a muchos (1:n).
- Muchos a muchos (n).

Veamos la tabla de la próxima pantalla.



	Uno a uno (1:1)	Uno a muchos (1:n)	Muchos a muchos (n)
Descripción	En esta relación, una instancia de una clase (A) está asociada con exactamente una instancia de otra clase (B) .	Una instancia de una clase (A) puede estar asociada con muchas instancias de otra clase (B), pero cada instancia de B está asociada con solo una instancia de A .	Varios objetos de una clase (A) pueden estar asociados con varios objetos de otra clase (B) , y viceversa.
Representación en los diagramas			
Ejemplo	<p>Un Estudiante tiene exactamente un DNI y viceversa.</p> <p>En código se puede utilizar un atributo, en cada clase, que referencie directamente a la otra clase.</p> <p><i>Por ejemplo: una clase Persona puede tener un atributo DNI, que es una instancia única de la clase Documento.</i></p>	<p>Un Curso tiene varios Estudiantes, pero cada Estudiante está inscrito en un solo Curso.</p> <p>En código: Se puede utilizar una colección en una de las clases para mantener referencias a múltiples instancias de la otra clase.</p> <p><i>Por ejemplo: una clase DepartamentoAcademico puede tener una lista de Profesores que trabajan en ese área.</i></p>	<p>Autores pueden escribir varios Libros, y cada Libro puede tener varios Autores.</p> <p>En código: Se pueden usar colecciones en ambas clases para mantener referencias a múltiples instancias de la otra clase.</p> <p><i>Por ejemplo: en un sistema de gestión de biblioteca, la clase Libro podría tener una lista de autores, y la clase Autor podría tener una lista de libros que ha escrito.</i></p>

Navegabilidad



Se refiere a la **capacidad de un objeto de acceder a los objetos relacionados**, a través de la relación definida.

En los diagramas UML y otros modelos visuales, la navegabilidad se representa con **flechas entre las clases relacionadas**.

En código se implementa mediante **referencias o punteros entre las clases** que están relacionadas.



Tipos de navegabilidad

	Unidireccional	Bidireccional
Descripción	Una clase puede acceder a otra clase, pero no necesariamente al revés.	Ambas clases pueden acceder y referenciar a la otra clase.
Representación en los diagramas		
Ejemplo	En un sistema de ventas, un Pedido puede tener una referencia al Cliente, pero el Cliente no necesita una referencia directa al Pedido.	En un sistema de redes sociales, una Publicación puede tener una referencia al Usuario que la creó, y el Usuario puede tener referencias a múltiples Publicaciones.

Aplicación de IA

La IA junto con los *Large Language Models* (LLM), pueden proporcionar diversas **herramientas y técnicas para optimizar el diseño y la gestión de relaciones en programación orientada a objetos.**

- **Generación automatizada de Diagramas**

UML: Los LLM pueden generar automáticamente diagramas UML que representan relaciones entre clases basadas en descripciones textuales de los requerimientos o especificaciones del sistema.

- **Análisis de complejidad y diseño:** La IA puede analizar la complejidad de las relaciones propuestas y ofrecer recomendaciones para mejorar el diseño y la estructura de las clases.
- **Explorar alternativas a partir de lenguaje natural:** Se puede modificar rápidamente código existente y explorar alternativas de forma sencilla con lenguaje natural.

