PROMETEO

Unidad 6:

Modelado UML – Diagramas de clases

Entornos de desarrollo

Técnico Superior de DAM / DAW

Sesión 22: Introducción y elementos de los diagramas de clases

Comprender la arquitectura del sistema antes de programar

Cuando trabajas en desarrollo de software, uno de los mayores retos es visualizar la estructura del sistema antes de escribir una sola línea de código. El Lenguaje Unificado de Modelado (UML) surge precisamente para resolver ese problema: te permite representar de forma gráfica aquello que normalmente sería abstracto o difícil de comunicar solo con palabras.

Dentro de los diferentes tipos de diagramas que ofrece UML, el diagrama de clases es el más utilizado en diseño orientado a objetos. Su función es mostrar cómo se organizan las entidades principales del sistema, qué atributos poseen, qué métodos las definen y, sobre todo, cómo se relacionan entre ellas. Podrías pensar en él como el mapa estructural de tu aplicación.

El diagrama de clases te ayuda a responder preguntas clave antes de comenzar a programar:

- ¿Qué objetos existen en mi sistema?
- ¿Qué información guarda cada uno?
- ¿Qué responsabilidades tiene?
- ¿Cómo se comunican entre sí?

Visualizar estas respuestas permite anticipar errores habituales, como duplicar lógica, crear dependencias innecesarias o diseñar clases demasiado grandes. Cuando trabajas en equipo, este diagrama también sirve como lenguaje común: todos ven el mismo mapa y todos entienden cómo debe estar organizado el sistema.

Para construir correctamente un diagrama de clases es imprescindible dominar sus cuatro pilares básicos:

1

La Clase

Es la unidad mínima del modelo. Representa una entidad con características (atributos) y comportamientos (métodos). En UML se dibuja como un rectángulo dividido en tres secciones: nombre, atributos y operaciones.

2

El Atributo

Describe las propiedades de una clase. Son variables que almacenan información relevante del objeto, como titulo o ISBN dentro de una clase Libro.

3

El Método

Define las acciones que el objeto puede realizar, como reservar(), registrar(), validar(). Representan su comportamiento dentro del sistema.

4

Las Relaciones

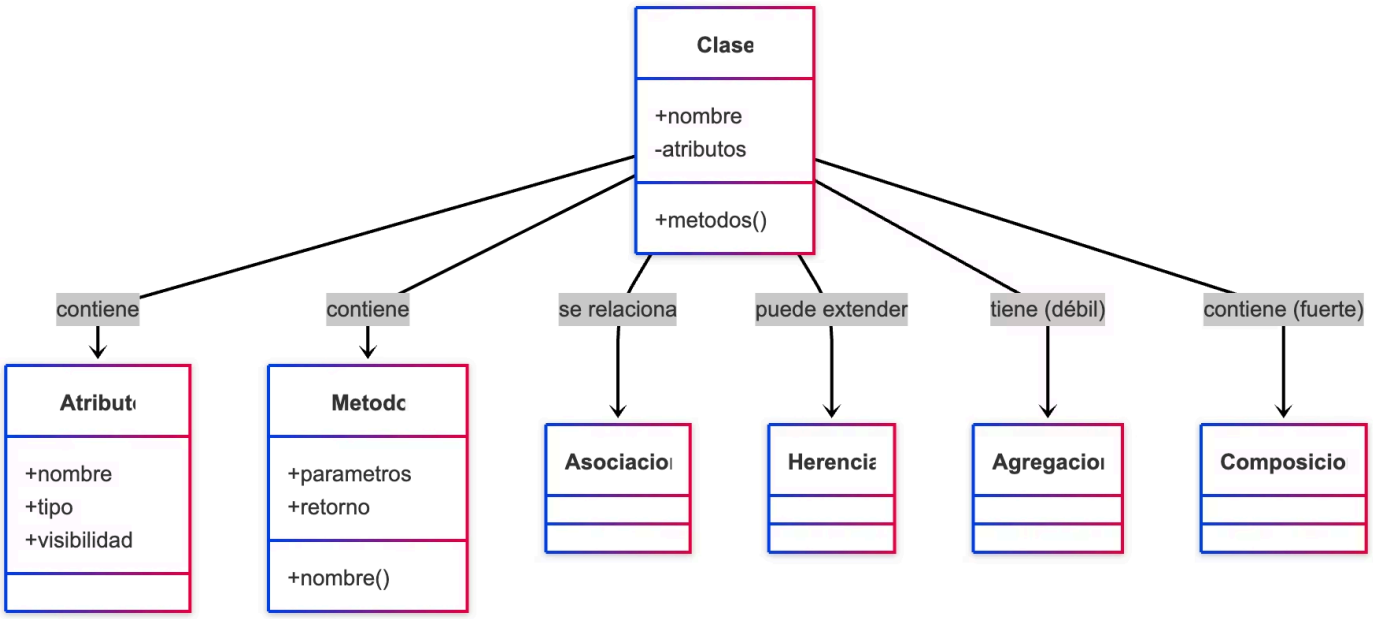
Son la clave del diseño orientado a objetos. Dan sentido al modelo porque explican cómo colaboran las clases. Las más habituales son:

- **Asociación**: vínculo general entre clases.
- **Herencia**: una clase hija "es un" subtipo de otra.
- **Agregación**: relación débil "tiene un".
- **Composición**: relación fuerte "contiene un"; si muere la clase padre, mueren sus componentes.

Cuando unes estos elementos, el diagrama de clases te permite ver la "anatomía" completa del sistema. No necesitas imaginar cómo interactúan los componentes: lo ves claramente, igual que un arquitecto observa los planos antes de construir un edificio.

En resumen, dominar este tipo de diagramas no es solo una habilidad técnica, sino una herramienta estratégica que mejora la calidad del software y reduce errores durante el desarrollo.

Esquema Visual: Anatomía de un diagrama de clases UML



A continuación tienes una representación detallada en formato Mermaid. Este diagrama refleja los elementos esenciales explicados en los fundamentos:

Explicación visual paso a paso

01

Clase es el nodo central

Representa cualquier entidad del sistema.

02

Atributo y Método

Cuelgan de la clase porque forman parte de ella.

03

Asociación

Muestra una relación flexible: dos clases se conocen y colaboran.

04

Herencia

Indica jerarquía: una clase especializada extiende a otra.

05

Agregación

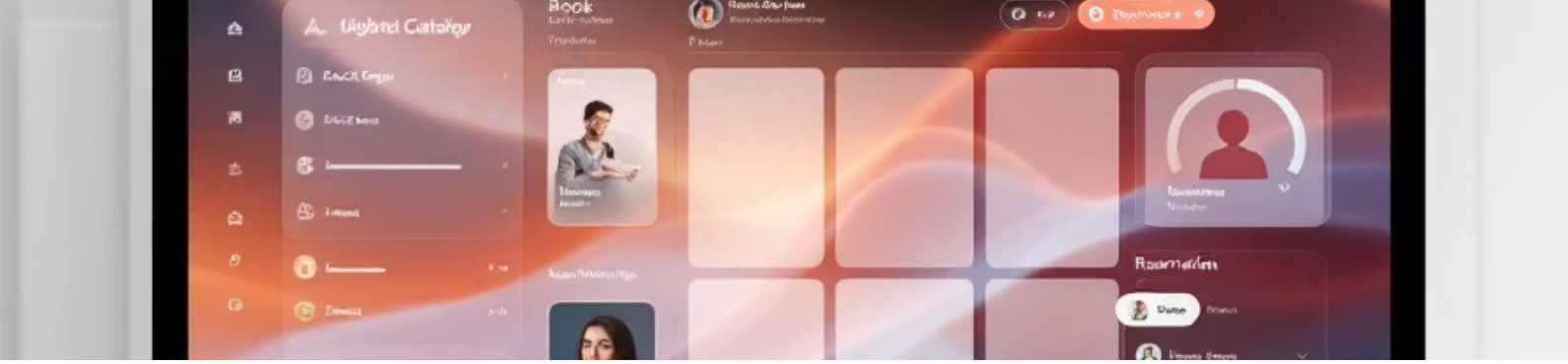
Indica que una clase tiene componentes, pero no depende totalmente de ellos.

06

Composición

Muestra dependencia absoluta: si la clase principal desaparece, sus partes también.

Con este esquema puedes recrear fácilmente un diagrama de clases básico, incluso si estás empezando con UML.



Caso de Estudio: Biblioteca Digital – Planificación del modelo antes de programar

Para comprender realmente el valor del diagrama de clases, observemos un caso realista basado en el diseño inicial de una aplicación de biblioteca digital.

Contexto

El equipo de desarrollo necesitaba crear una plataforma donde los usuarios pudieran consultar libros disponibles, reservarlos temporalmente y gestionar su actividad como lectores. Antes de escribir código, se elaboró un diagrama de clases para visualizar la estructura.

Estrategia

Se definieron tres clases principales:

Clase Libro

- **Atributos:** titulo, autor, ISBN, anioPublicacion.
- **Métodos:** ninguno por el momento, ya que la clase actúa como entidad de datos.

Clase Usuario

- **Atributos:** nombre, email, idUsuario.
- **Métodos:**
  - registrar()
  - reservar(libro)
  - cancelarReserva()

Relación entre ambas

Se decidió usar una asociación múltiple:

- Un usuario puede reservar varios libros.
- Un libro solo puede estar reservado por un usuario a la vez.

Esto se representó con una asociación 1...* desde Usuario hacia Libro.

Resultado

Gracias al diagrama de clases:

1. El equipo validó rápidamente que no se necesitaban relaciones complejas como herencias.
2. Se identificó un potencial problema: ¿qué ocurre si dos usuarios intentan reservar el mismo libro simultáneamente? El modelo reveló la necesidad de una clase extra: **Reserva**.
3. Se definieron claramente las entidades para implementar la base de datos.

Este caso demuestra que un diagrama bien hecho evita retrabajo, inconsistencias y ambigüedades durante la fase de desarrollo.

Herramientas y Consejos para tu Futuro Profesional

Diseñar diagramas de clases es una habilidad que usarás constantemente en análisis, diseño, documentación y comunicación entre equipos. Aquí tienes recomendaciones prácticas:

1

Usa nombres claros y significativos

Evita abreviaturas innecesarias. Ejemplo: ClientePremium es mejor que CliPrem.

2

Aplica herencia solo si existe una relación "es un"

Una regla práctica: Si "A es un tipo de B" funciona, entonces herencia es adecuada. Si no funciona, probablemente necesites composición.

3

Limita el tamaño del diagrama

No mezcles demasiadas clases. Máximo recomendado: 10–12 clases por diagrama conceptual.

4

Herramientas recomendadas

- **Lucidchart:** ideal para equipos y colaboración.
- **Draw.io:** gratuito, rápido y accesible.
- **StarUML:** orientado a ingeniería directa y generación de código.
- **Visual Paradigm:** muy completo para proyectos grandes.

5

Mantén consistencia en el estilo

Usa la misma notación para atributos y métodos, por ejemplo:

- + público
- - privado
- # protegido

6

Apóyate en el diagrama para validar requisitos

Antes de diseñar la base de datos o escribir código, revisa si el diagrama cubre todas las reglas de negocio.

Mitos y Realidades

❌ Mito 1: "UML solo se usa en proyectos grandes."

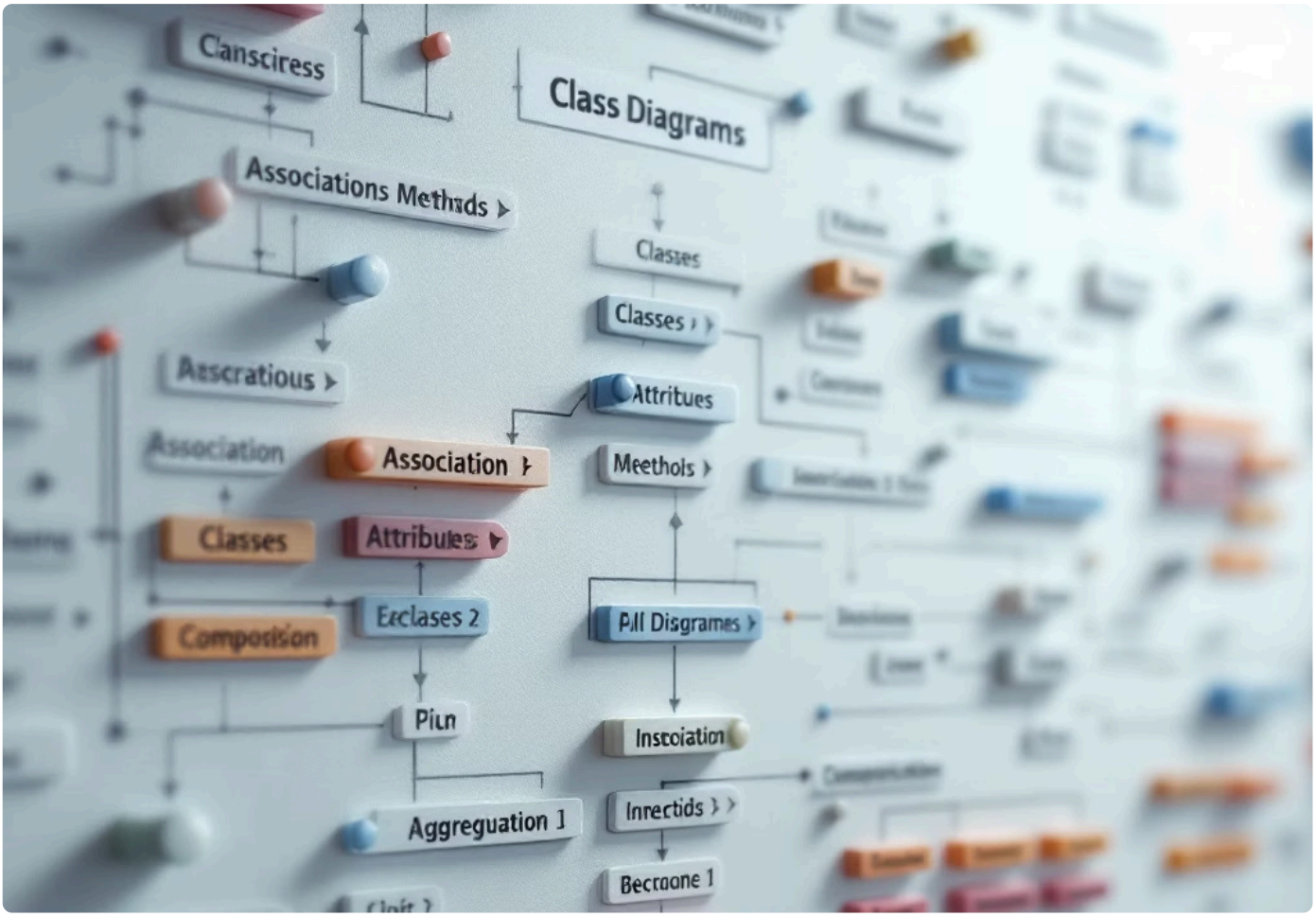
➡️ **FALSO.** Incluso en proyectos pequeños te permite aclarar entidades, validar relaciones y evitar reescrituras costosas. No importa el tamaño: el diagrama de clases ayuda a pensar antes de programar.

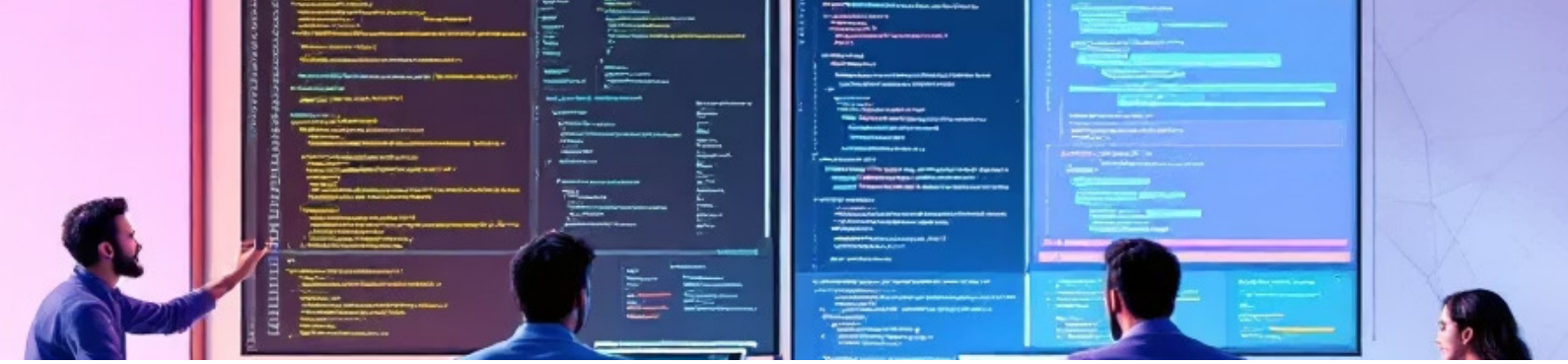
❌ Mito 2: "Los diagramas de clases son solo documentación."

➡️ **FALSO.** Un buen diagrama sirve para diseñar, comunicar, descubrir fallos y alinear al equipo. En herramientas como StarUML o Visual Paradigm incluso puedes generar código base desde el propio diagrama.

Resumen Final

- UML es el estándar para modelar software.
- El diagrama de clases representa la estructura del sistema.
- Elementos clave: clase, atributo, método.
- Relaciones: asociación, herencia, agregación, composición.
- Útil para planificar, comunicar y evitar errores antes de programar.





Sesión 23: Caso práctico y herramientas UML

Aplicar el modelado UML para validar la arquitectura real

Una vez que dominas los elementos esenciales del diagrama de clases —clases, atributos, métodos y relaciones— el siguiente paso es utilizarlos en un caso práctico que refleje cómo se diseña realmente un sistema antes de programarlo.

En esta sesión profundizas en la aplicación profesional del modelado UML, donde la teoría se traduce en decisiones de arquitectura y en la alineación entre analistas, programadores y responsables de producto.

El modelado visual tiene un objetivo clave: garantizar que todo el equipo tenga la misma imagen del sistema antes de escribir código, minimizando riesgos de diseño y permitiendo identificar problemas estructurales antes de que se conviertan en errores costosos durante la implementación.

UML actúa como un lenguaje común entre perfiles técnicos y no técnicos, facilitando debates, revisiones y mejoras.

Uno de los grandes avances de las herramientas UML modernas es que han dejado de ser simples plataformas de dibujo. Ahora se integran profundamente en el ciclo de desarrollo, permitiendo trabajar en dos direcciones:

Ingeniería directa

A partir del diagrama, la herramienta genera automáticamente:

- clases vacías o esqueleto de código,
- estructuras iniciales de atributos,
- métodos declarados para ser implementados,
- paquetes organizados según el diseño.

Esto acelera el desarrollo, reduce inconsistencias y evita que cada programador interprete el diseño a su manera.

Ingeniería inversa

El proceso contrario: subir código ya existente y generar automáticamente el diagrama de clases. Este enfoque es muy útil cuando:

- te incorporas a un proyecto ya avanzado,
- quieres analizar la arquitectura de un sistema legado,
- necesitas documentar un proyecto que nunca tuvo diagramas.

Hoy en día, herramientas como StarUML, Visual Paradigm, Enterprise Architect o Lucidchart permiten trabajar de forma colaborativa, sincronizar cambios y validar reglas del modelo (por ejemplo, herencias incorrectas o relaciones sin sentido).

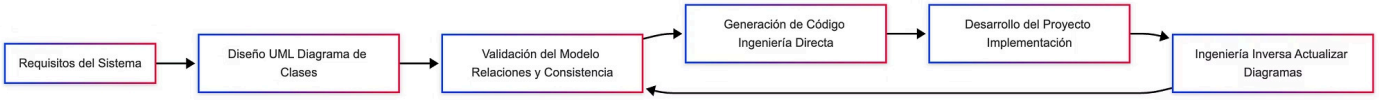
El diseño con UML no pretende sustituir la programación, sino garantizar que el código nazca bien estructurado. Un modelo sólido evita clases duplicadas, relaciones equivocadas y estructuras de datos deficientes.

En entornos ágiles como Scrum, UML se utiliza de forma estratégica: ligera, funcional y orientada a resolver dudas críticas.

En esta sesión, comprenderás cómo aplicar UML en un caso práctico realista y qué herramientas profesionales te permiten integrar el modelado en tu flujo de trabajo.

Esquema Visual: Flujo profesional de un proyecto UML

Este esquema Mermaid representa el ciclo de uso de UML en proyectos reales: diseño → validación → generación → mantenimiento.



Explicación detallada

A. Requisitos del Sistema

punto de partida donde se definen las funcionalidades y reglas de negocio.

B. Diseño UML

se modelan entidades, atributos, métodos y relaciones.

C. Validación del Modelo

se revisa que la estructura tenga coherencia y no existan contradicciones.

D. Generación de Código

herramientas como Visual Paradigm o StarUML producen la estructura inicial del código.

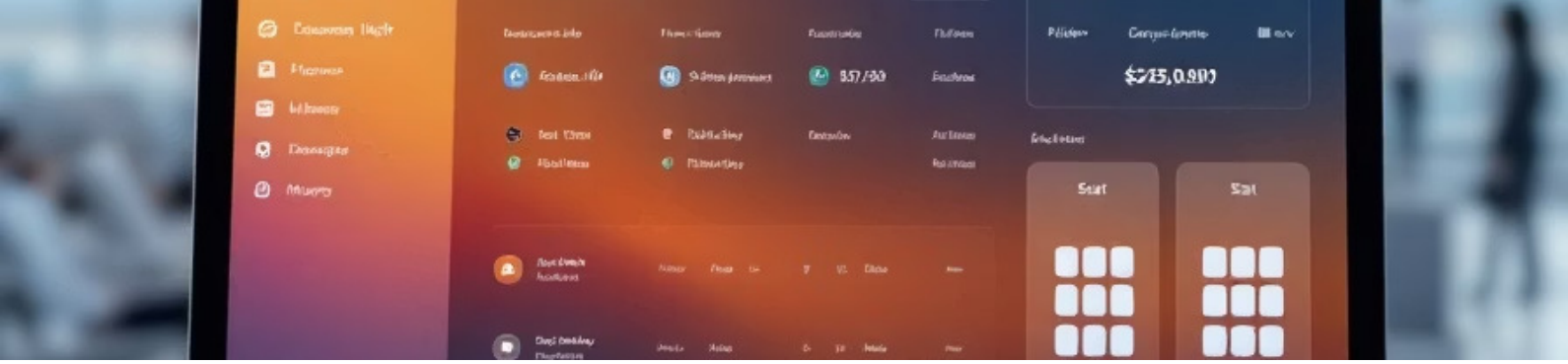
E. Desarrollo

los programadores implementan la lógica real.

F. Ingeniería Inversa

se actualiza el diagrama en función del código real para mantener la documentación al día.

Este ciclo continuo garantiza que el diseño y el código evolucionen de manera alineada.



Caso de Estudio: Sistema de reserva de vuelos con Visual Paradigm

Contexto

Un equipo de desarrollo debía crear un sistema de reservas de vuelos para una aerolínea. El objetivo era permitir a los pasajeros consultar vuelos, registrarse como clientes, realizar reservas y gestionar asientos. Antes de iniciar la programación, decidieron trabajar con Visual Paradigm, que permite modelar, validar el diagrama y generar código Java automáticamente.

Estrategia

El equipo identificó tres clases esenciales:

Clase Vuelo

- **Atributos:** codigo, origen, destino, fechaSalida, hora, capacidad.
- **Métodos:** consultarDisponibilidad(), actualizarPlazas().

Clase Pasajero

- **Atributos:** nombre, dni, email, telefono.
- **Métodos:** registrar(), actualizarDatos(), realizarReserva().

Clase Reserva

- **Atributos:** numeroReserva, fechaReserva, asiento, estado.
- **Métodos:** confirmar(), cancelar(), cambiarAsiento().

Relaciones

- Un pasajero puede tener varias reservas → asociación 1…n.
- Una reserva pertenece a un único vuelo → relación de composición (sin vuelo no existe la reserva).
- Un vuelo tiene varias reservas, pero si el vuelo se elimina, todas sus reservas también → composición legítima.

Resultado

Con el diagrama validado, el equipo utilizó la función de generación automática de código de Visual Paradigm:

1. Se generaron clases Java con atributos y métodos vacíos.
2. Se creó la estructura de paquetes (model, service, controller).
3. Se redujo el tiempo de desarrollo inicial.
4. El diagrama se mantuvo sincronizado con el código mediante ingeniería inversa semanal.

Este caso demuestra cómo UML se convierte en un aliado práctico y no solo en una herramienta de documentación. El diseño facilitó la discusión con el equipo de negocio, permitiendo detectar inconsistencias antes de desarrollar funcionalidades críticas.

Herramientas y Consejos para tu Futuro Profesional

Aquí tienes recomendaciones basadas en buenas prácticas de ingenieros de software y equipos reales:

Usa colores, iconos o estereotipos

Ayudan a identificar:

- clases de modelo,
- controladores,
- repositorios,
- servicios,
- entidades persistentes.

Visual Paradigm y StarUML permiten asignar estereotipos visuales como <<Entity>>, <<Controller>>.

Valida el modelo antes de generar código

Hazte estas preguntas:

- ¿La herencia está bien justificada?
- ¿Las relaciones tienen sentido desde el punto de vista del negocio?
- ¿Hay atributos duplicados en varias clases?
- ¿Hay clases con demasiadas responsabilidades?

Un buen diseño se revisa igual que un documento.

Elige la herramienta adecuada según tu objetivo

- **Draw.io:** perfecto para estudiantes o proyectos rápidos.
- **Lucidchart:** ideal para colaboración en tiempo real.
- **StarUML:** muy potente para ingeniería directa y plugins.
- **Visual Paradigm:** completa integración con bases de datos, BPMN y generación de código.
- **Enterprise Architect:** la preferida para grandes empresas y sistemas críticos.

Sincroniza diagrama y código

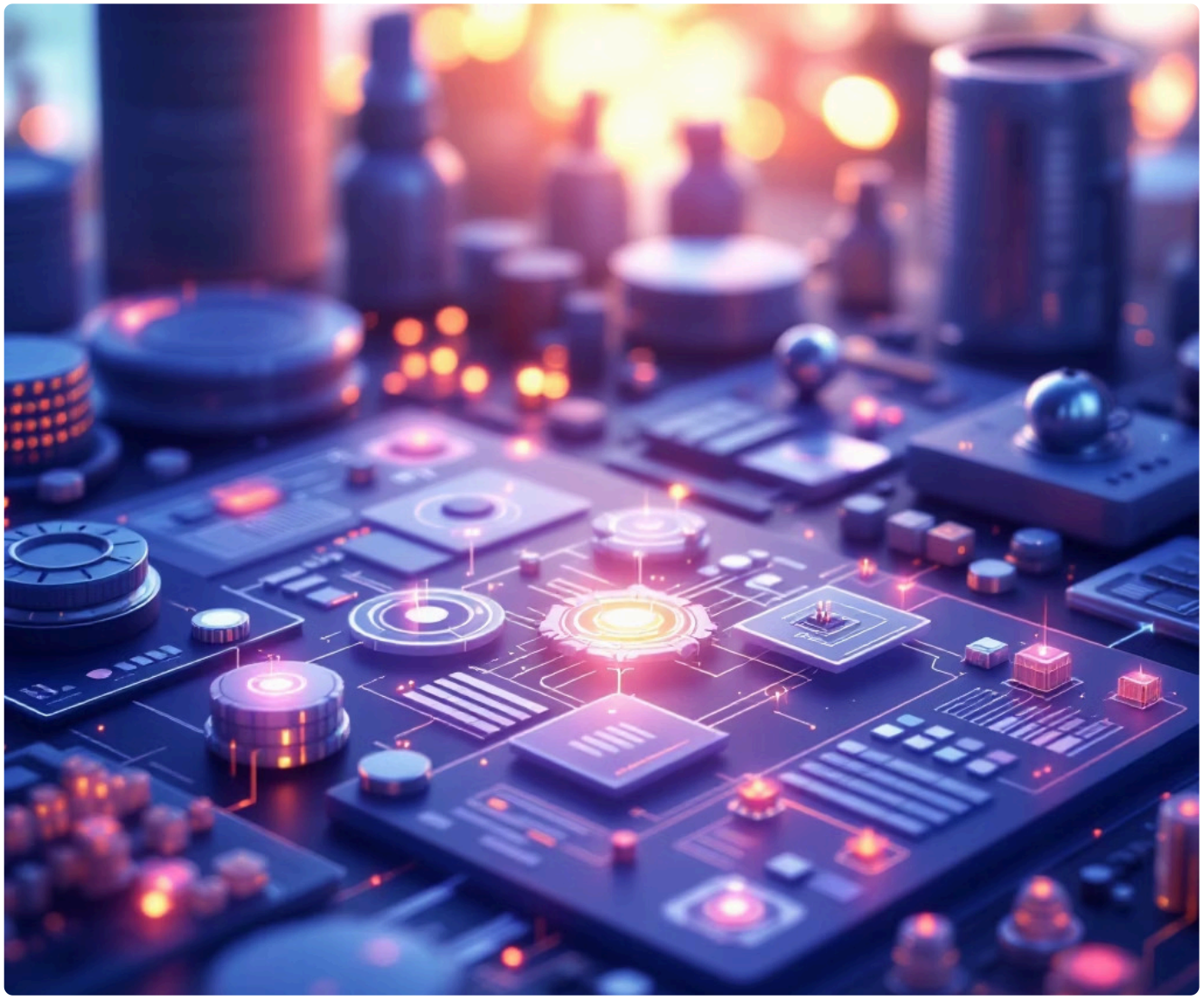
Haz ingeniería inversa periódicamente para evitar que el diagrama se quede desactualizado.

Alinea el modelo con los requisitos funcionales

Cada clase debe responder a un concepto real del sistema:

- ¿Existe este elemento en los requisitos?
- ¿Tiene sentido que otras clases dependan de él?
- ¿Falta alguna entidad relevante que el negocio sí necesita?

Un diagrama bien alineado previene malentendidos entre analistas y programadores.



Mitos y Realidades

❌ Mito 1: "Los diagramas UML solo sirven como documentación."

➡️ **FALSO.** Las herramientas modernas permiten generar código, validar consistencia, crear bases de datos y sincronizar automáticamente los cambios. UML es parte del proceso, no solo documentación final.

❌ Mito 2: "Modelar con UML retrasa el desarrollo."

➡️ **FALSO.** Modelar ahorra tiempo: evita errores estructurales, acelera el desarrollo cuando se genera código automáticamente y facilita la comunicación entre perfiles técnicos y no técnicos.

Resumen Final (para examen)

- UML permite modelar sistemas antes de programar.
- Etapas: diseño → validación → generación de código.
- Herramientas clave: StarUML, Visual Paradigm, Lucidchart, Enterprise Architect.
- Ingeniería directa genera clases automáticamente.
- Ingeniería inversa crea diagramas desde código.
- Caso práctico: sistema de vuelos con clases Vuelo, Pasajero y Reserva