Guía práctica para elaborar Diagramas UML: Diagrama de Clases y Diagrama de Secuencia

Enero 2026

Enfoque para proyectos de desarrollo de software (con ejemplos en PlantUML)
Autoría recomendada: Alfredo Weitzenfeld (UML orientado a objetos)

---

1. ¿Para qué sirven estos diagramas?

UML (Unified Modeling Language) es un estándar para **visualizar, especificar, construir y documentar** artefactos de sistemas de software.

En práctica, dos de los diagramas más usados para diseño y comunicación en equipos son:

- **Diagrama de clases:** describe la **estructura estática** (clases, atributos, operaciones y relaciones).
- **Diagrama de secuencia:** describe el **comportamiento** como **interacción temporal** (mensajes) entre objetos/participantes.

Usados en conjunto, ayudan a pasar de requisitos (casos de uso/historias) a un diseño comprensible y trazable hacia el código.

2. Flujo recomendado (de requisitos a diseño)

- **1)** Especifica requisitos con **casos de uso** (o historias) y **escenarios** (principal y alternos).
- **2)** Para cada escenario clave, elabora un **diagrama de secuencia** (o SSD + secuencia de diseño).
- **3)** A partir de responsabilidades y mensajes, deriva o ajusta el **diagrama de clases**.
- **4)** Itera: cambios en secuencia suelen revelar métodos, clases o asociaciones faltantes.

3. Diagrama de clases: lo esencial

3.1 Elementos y notación mínima

- **Clase:** nombre (sustantivo), atributos (estado) y operaciones/métodos (comportamiento).

- **Visibilidad (opcional):** + público, - privado, # protegido.
- **Relaciones:** asociación (`---`), agregación (`o---`), composición (`*---`), generalización (`<|---`), dependencia (`..>`).
- **Multiplicidad:** `1`, `0..1`, `0..*`, `1..*`, `n..m`.
- **Interfaces (opcional):** útil para contratos (p. ej., repositorios/servicios).

3.2 Pasos para construirlo (heurística práctica)

- Identifica **clases del dominio** (sustantivos en requisitos): entidades (Estudiante, Pedido, Libro) y objetos de valor (Dirección, Dinero).
- Define **responsabilidades** y evita la clase "Dios" (God Object).
- Atributos: guarda solo datos propios de la clase (evita duplicidad) y usa tipos coherentes.
- Relaciones: modela **"conoce a"** (asociación), **"parte de"** (composición), **"es un"** (herencia) cuando aplique.
- Agrega operaciones después de ver interacciones (secuencia): **los mensajes suelen convertirse en métodos**.

3.3 Checklist rápido de calidad

Diagrama de Casos de Uso:

Revisar	Cumple	Qué debe cumplirse	Observacion
Nombres		Clases en singular y con significado del dominio.	Los nombres de los casos de uso deben ser coherentes (a los de las HU)

Diagrama de Clases:

Revisar	Cumple	Qué debe cumplirse	Error común
Nombres		Clases en singular y con significado del dominio.	Nombres genéricos (Datos, Manager, Utils).
Multiplicidad		Cada asociación relevante tiene cardinalidad.	Describir dependencias entre las clases
Cohesión		Responsabilidades claras y relacionadas.	Coherencia entre codigo y
Acoplamiento		Dependencias mínimas y justificadas.	Asociaciones por conveniencia sin razón del dominio.
Trazabilidad		Métodos justificados por escenarios (secuencias).	Inventar métodos sin escenario real.

Diagrama de Secuencia:

Revisar	Cumple	Qué debe cumplirse	Onservaciones
Nombres		Clases en singular y con significado del dominio.	Nombres genéricos (Datos, Manager, Utils).
Multiplicidad		Cada asociación relevante tiene cardinalidad.	Describir dependencias entre las clases
Cohesión		Responsabilidades claras y relacionadas.	Coherencia entre codigo y
Acoplamiento		Dependencias mínimas y justificadas.	Asociaciones por conveniencia sin razón del dominio.
Trazabilidad		Métodos justificados por escenarios (secuencias).	Inventar métodos sin escenario real.

4. Diagrama de secuencia: lo esencial

4.1 Elementos clave

- **Participantes / lifelines:** actor, UI, controlador, servicio, entidad, repositorio, etc.
- **Mensajes:** síncronos (llamada), asíncronos, retornos (opcional).
- **Activación:** periodo de ejecución en un participante.
- **Fragmentos combinados:** `alt` (alternativas), `opt` (opcional), `loop` (repetición), `par` (paralelo).
- **Creación / destrucción:** `create`, `destroy` (cuando aplica).

4.2 Cómo construirlo (paso a paso)

- Elige un **escenario concreto** (p. ej., "Registrar préstamo" o "Checkout").
- Lista eventos/mensajes en orden temporal desde la perspectiva del actor.
- Introduce un **controlador** (Controller) que orqueste el flujo y evita lógica en UI.
- Modela validaciones y alternativas con `alt/opt`.
- Verifica que cada mensaje tenga un dueño razonable (responsabilidad) y luego convierte mensajes en métodos.

5. Ejemplos (PlantUML) para proyectos

5.1 Ejemplo A: Sistema de Biblioteca (préstamos)

Escenario: un bibliotecario registra el préstamo de un ejemplar a un usuario.

Diagrama de clases (simplificado)

@startuml

```
title Diagrama de clases (simplificado)

class Usuario {
  -id: String
  -nombre: String
  +puedePrestar(): boolean
}

class Libro {
  -isbn: String
  -titulo: String
}

class Ejemplar {
  -codigo: String
  -estado: EstadoEjemplar
}

class Prestamo {
  -id: String
  -fechaInicio: Date
  -fechaFin: Date
  +cerrar(): void
}

Usuario "1" -- "0..*" Prestamo
Prestamo "1" -- "1" Ejemplar
Libro "1" o-- "1..*" Ejemplar : contiene

@enduml
```

Diagrama de secuencia (Registrar préstamo)

```
@startuml
actor Bibliotecario
boundary UI as "UI Prestamos"
control PrestamoController
entity Usuario
entity Ejemplar
entity Prestamo
database Repo as "Repositorio"

Bibliotecario -> UI: solicitarPrestamo(idUsuario, codEjemplar)
UI -> PrestamoController: registrarPrestamo(idUsuario, codEjemplar)
PrestamoController -> Repo: buscarUsuario(idUsuario)
Repo --> PrestamoController: Usuario
PrestamoController -> Repo: buscarEjemplar(codEjemplar)
Repo --> PrestamoController: Ejemplar
```

```
alt usuario no habilitado
  PrestamoController -> UI: mostrarError("Usuario no puede prestar")
else ejemplar no disponible
  PrestamoController -> UI: mostrarError("Ejemplar no disponible")
else OK
  PrestamoController -> Prestamo: create(fechaInicio, fechaFin)
  PrestamoController -> Repo: guardarPrestamo(Prestamo)
  PrestamoController -> UI: confirmarPrestamo(idPrestamo)
end
@enduml
```

Nota: en un diseño más detallado, `Usuario` y `Ejemplar` podrían exponer métodos como `puedePrestar()` o `estaDisponible()`, y el controlador delega validaciones a dichas entidades.

6. Herramientas y consejos de entrega

- Define un estándar de diagramación para el equipo (nombres, nivel de detalle, convenciones).
- Mantén los diagramas "lo suficientemente detallados": útiles para discutir decisiones, no para duplicar el código.
- Versiona diagramas junto al repositorio (p. ej., PlantUML en `docs/uml`).
- Para revisión académica: exige que cada diagrama tenga **un escenario asociado** (secuencia) y reglas de negocio (multiplicidades).

7. Referencias (APA 7)

1. Object Management Group. (2017). *Unified Modeling Language (UML) Version 2.5.1*. https://www.omg.org/spec/UML/2.5.1/
2. Weitzenfeld, A. (2005). *Ingeniería de software orientada a objetos con UML, Java e Internet*. International Thomson Editores.
3. Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley.
4. Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development*. Prentice Hall.
5. IBM Developer. (2023, October 9). *Explore the UML sequence diagram*. https://developer.ibm.com/articles/the-sequence-diagram/