



Resumen Completo de UML

1. ¿Qué es UML?

- **UML (Unified Modeling Language)** es un lenguaje gráfico para modelar sistemas de software.
- Permite visualizar, construir y documentar aplicaciones mediante diagramas.
- Fue desarrollado en los años 90 por **Grady Booch, James Rumbaugh e Ivar Jacobson** (conocidos como "Los Tres Amigos").
- En 1997 fue adoptado por **Object Management Group (OMG)** y ha evolucionado en varias revisiones.

¿Por qué UML es un "lenguaje gráfico"?

- Utiliza símbolos y diagramas para representar:
 - **Estructura** del sistema (clases, objetos).
 - **Comportamiento** del sistema (casos de uso, secuencia de eventos).
 - **Relaciones** entre los elementos.
-

2. Modelos de Desarrollo y UML

♦ Modelo Waterfall (Cascada)

- Fases rígidas y secuenciales (Análisis → Diseño → Implementación → Pruebas → Mantenimiento).
- Se usa cuando los requisitos están muy bien definidos desde el principio.
- UML se emplea para crear planos detallados que no deben cambiar.
- Ideal para **aplicaciones críticas** (militares, médicas, de seguridad).

♦ Modelo Agile

- Enfoque iterativo, con cambios frecuentes según necesidades del cliente.
 - UML se usa como herramienta flexible de comunicación entre equipo y cliente.
 - **4 principios del "Agile Manifesto":**
 1. Individuos e interacciones sobre procesos y herramientas.
 2. Software funcional sobre documentación extensiva.
 3. Colaboración con el cliente sobre negociación contractual.
 4. Respuesta al cambio sobre seguir un plan rígido.
-

3. Principios de Programación Orientada a Objetos (POO) y UML

UML se usa principalmente en sistemas **orientados a objetos**, por lo que es clave entender estos conceptos:

♦ Abstracción

- Definir **qué propiedades y métodos** deben tener los objetos.
- Ejemplo: Diferentes modelos de Renault comparten un mismo chasis → **Clase base "Vehículo"**.

♦ Modularización

- Dividir el sistema en **módulos reutilizables**.
- Ejemplo: Casas prefabricadas donde cada módulo encaja en el diseño general.

♦ Encapsulamiento

- Ocultar detalles internos de un objeto para evitar accesos no controlados.
- Se usa con modificadores de acceso (**private**, **protected**, **public**).
- **Ejemplo:** El motor de un coche es inaccesible en modelos modernos.

♦ Polimorfismo



- Un objeto puede comportarse de diferentes maneras según el contexto.

Ejemplo en Java:

```
class Empleado {  
    void trabajar() { System.out.println("Trabajo general."); }  
}  
class Director extends Empleado {  
    void trabajar() { System.out.println("Trabajo de dirección."); }  
}
```

- Un **Empleado** puede actuar como **Director** en tiempo de ejecución.

♦ Herencia

- Permite reutilizar código estableciendo relaciones jerárquicas.
 - Se usa la regla **"ES UN"**:
 - ¿Un **director** es un **empleado**? →  Sí → **Director** hereda de **Empleado**.
 - ¿Un **empleado** es un **director**? →  No.
-

4. Diagramas UML más utilizados

1. Diagrama de Casos de Uso

- Representa la interacción de los usuarios con el sistema.
- **Elementos:**
 - **Actores** (Usuarios, Administrador, Aplicaciones externas).
 - **Casos de Uso** (Acciones que realiza el sistema).
 - **Relaciones:**
 - **Include** → **Obligatoria** (Ej: Un **login** siempre incluye la verificación de credenciales).
 - **Extend** → **Opcional** (Ej: Mostrar recomendaciones solo si el usuario lo solicita).

2. Diagrama de Clases

- Muestra la estructura de clases y relaciones entre ellas.
- **Tipos de relaciones:**
 - **Asociación** (↔) → Conexión simple entre clases.
 - **Agregación** (◻ →) → Una clase usa objetos de otra, pero pueden existir por separado (Ej: una biblioteca tiene libros, pero los libros pueden existir sin la biblioteca).
 - **Composición** (◼ →) → Una clase depende totalmente de otra (Ej: Un motor pertenece a un coche y no existe sin él).
 - **Herencia** (→) → Relación "ES UN" (Ej: **Director** es un **Empleado**).

3. Diagrama de Objetos

- Representa instancias concretas de clases en un momento dado.
- Se usa para visualizar la estructura en tiempo de ejecución.

4. Diagrama de Implementación o Despliegue

- Representa hardware, servidores y software involucrado en la ejecución de una aplicación.

5. Buenas prácticas al usar UML

- ✓ No abusar de dependencias en los diagramas para evitar confusión.
- ✓ No confundir **extends** con **if-else**, ya que **extends** representa herencia.
- ✓ Distinguir entre **dependencia** y **caso de uso independiente**.
- ✓ Usar **generalización** cuando se pueda decir "ES UN TIPO DE".
- ✓ **UML no es un documento fijo**, debe adaptarse conforme avanza el desarrollo del software.

★ 6. Puntos clave para el examen

- ✓ UML es un lenguaje gráfico usado para documentar, analizar y diseñar sistemas.
- ✓ Waterfall = UML como **plano obligatorio**, Agile = UML como **herramienta flexible**.
- ✓ Principios de POO en UML: **abstracción, encapsulamiento, modularización, herencia y polimorfismo**.
- ✓ Diagramas UML más importantes:
 - Casos de Uso → Representa acciones y actores.
 - Clases → Representa la estructura del sistema.
 - Objetos → Representa instancias en tiempo de ejecución.
 - Implementación → Representa la infraestructura del software.
 - ✓ **Include** = dependencia obligatoria, **Extend** = dependencia opcional.
 - ✓ Herencia ("ES UN") permite reutilizar código (**Director** hereda de **Empleado**).
 - ✓ Composición vs. Agregación:
 - Composición: Si el "todo" desaparece, las partes también (Ej: un motor dentro de un coche).
 - Agregación: Las partes pueden existir sin el "todo" (Ej: clientes en una tienda).
 - ✓ UML es flexible y puede cambiar conforme avanza el desarrollo del sistema.