ChatGPT

Diagramas de Clases en UML

Un **diagrama de clases** en UML es una representación gráfica de la estructura estática de un sistema orientado a objetos [1] [2]. Cada clase se dibuja como un rectángulo dividido en tres partes: nombre, atributos y métodos [2] [3]. Las líneas y flechas que conectan los rectángulos muestran las **relaciones** entre clases, como herencia, asociación, agregación o composición [2] [4]. Por ejemplo, los diagramas de clases se usan para visualizar y documentar la arquitectura del software, detallando qué clases existen en el sistema y cómo se relacionan [1] [5].

Los diagramas de clases sirven principalmente para planificar y comunicar la estructura de un sistema antes de implementarlo. Permiten entender la visión global del modelo de datos y su organización en objetos, lo que facilita la comunicación entre desarrolladores y el diseño del código [5] [6]. También ayudan a detectar requisitos y dependencias clave: al identificar las clases y sus relaciones, los equipos pueden diseñar la arquitectura de forma consistente y prepararse para la implementación en un lenguaje de programación (por ejemplo, traducir el diagrama a código Python) [6] [7].

Notación de un Diagrama de Clases

En UML se usan convenciones específicas para dibujar diagramas de clases:
- **Clase:** se dibuja como un rectángulo dividido en tres partes. La parte superior tiene el *nombre de la clase*, la parte media lista sus *atributos* y la parte inferior enumera sus *operaciones o métodos* [8] [3].
- **Atributos:** representan las propiedades de la clase. Se anotan en la segunda sección del rectángulo. Por convención se puede indicar su visibilidad: por ejemplo, `+` (público), `-` (privado), `#` (protegido), `~` (paquete) [9].
- **Métodos:** representan el comportamiento o acciones de la clase. Se listan en la tercera sección del rectángulo [8] [3]. También se puede indicar visibilidad y si son métodos de clase (estáticos) subrayando su nombre [9].

Por ejemplo, en un diagrama típico se leerá algo así: `+ nombre: String` indica un atributo público `nombre` de tipo cadena, y `+ calcular(): void` indica un método público `calcular`.

Atributos y Métodos en el Diagrama de Clases

Los atributos y métodos definen la estructura interna de cada clase. Los **atributos** representan datos o características (como por ejemplo `nombre`, `precio`, `fechaNacimiento`), mientras que los **métodos** representan operaciones o acciones que la clase puede realizar (por ejemplo, `calcularTotal()`, `obtenerNombre()`) [10] [11]. Cada atributo o método se anota en su propia línea dentro de la clase. En UML se diferencia el alcance de cada miembro: atributos o métodos estáticos se subrayan; públicos se marcan con `+`, privados con `-`, etc. [12] [10].

En resumen, al mirar un rectángulo de clase: la fila del medio lista las propiedades o atributos, y la fila inferior lista los métodos. Estos elementos describen cómo será la clase al implementarla en código (por ejemplo, en Python serían sus variables de instancia y funciones miembro) [10] [13].

Colaboración (Asociación) y Composición

Las **relaciones** entre clases se dibujan con líneas entre los rectángulos:

- **Asociación (colaboración):** una línea simple entre dos clases indica que las instancias de una clase están relacionadas con instancias de otra. Por defecto es bidireccional, pero se puede usar una flecha para indicar dependencia unidireccional [14] [15]. La multiplicidad se anota cerca de los extremos de la línea (por ejemplo `1`, `0..*`, `*`) para indicar cuántas instancias pueden relacionarse [16] [17]. Por ejemplo, una asociación unidireccional se dibuja con una línea con una flecha abierta, indicando qué clase "conoce" a cuál [18] [14].

- **Herencia (generalización):** se dibuja con una línea con punta de flecha cerrada en sentido de la superclase [19] [20]. Indica relación "es-un" entre subclase y superclase. Por ejemplo, si `Student` hereda de `Person`, en el diagrama aparece una flecha sólida apuntando de `Student` hacia `Person` [19] [20].

- **Agregación:** es una relación de "parte-todo" débil. Se representa con un rombo hueco en el extremo de la clase *contenedora* apuntando hacia la clase *contenida* [21] [22]. Indica que la clase contenedora tiene una relación de "tiene a" con la contenida, pero las instancias contenidas pueden existir independientemente.

- **Composición:** es un tipo fuerte de agregación. Se dibuja igual, pero con el rombo relleno de negro en el extremo de la clase contenedora [23] [24]. Significa que las instancias de la clase parte dependen totalmente del todo: si el objeto contenedor se destruye, sus partes también. Por ejemplo, un `Order` compuesto por `OrderItem`s se modela con composición, porque los ítems viven dentro del pedido [24] [23].

En cualquiera de estos casos, el diagrama muestra visualmente las "colaboraciones" entre clases. Al leer el diagrama, se distinguen estos símbolos: flecha de herencia, rombo vacío o lleno, etc., lo cual indica exactamente qué tipo de relación existe [21] [23].

El diagrama de clases siguiente ejemplifica un sistema de biblioteca. Cada rectángulo representa una clase (por ejemplo, `Book`, `Librarian`, `User`, etc.) con sus atributos y métodos, y las líneas con rombos indican relaciones de agregación o asociación entre ellas. Este tipo de diagrama ilustra cómo cada objeto (clase) se estructura y colabora, de acuerdo a la definición de UML [25] [7].

Leyendo un Diagrama de Clases

Para **interpretar** un diagrama de clases, se analiza cada componente gráfico:

- **Clases:** se leen los nombres de las clases en la parte superior de cada rectángulo. Bajo cada nombre, los atributos y métodos listados nos indican qué información guarda el objeto y qué acciones puede realizar.
- **Herencia:** una flecha cerrada "vacía" hacia arriba significa que la clase origen hereda de la clase destino [20]. Esto implica que la subclase incluye los atributos/métodos de la superclase.
- **Asociaciones:** una línea simple entre clases indica que los objetos se conocen o relacionan. Si junto a la línea aparece un número o `*`, indica multiplicidad (por ejemplo, `*` para "muchos") [16] [17]. Una flecha en la línea indica la dirección de la dependencia.

- **Agregación/Composición:** un rombo al final de la línea señala las relaciones de parte-todo. Un rombo vacío (no relleno) indica agregación [21], mientras que un rombo relleno indica composición [23]. Al leer el diagrama, esto indica si las instancias de la clase parte dependen fuertemente del contenedor o no.

En conjunto, el diagrama describe la estructura del sistema. Por ejemplo, si vemos `Order —<> OrderItem` con un rombo relleno junto a `Order`, entendemos que un pedido **está compuesto de** ítems y que los ítems no tienen sentido fuera de ese pedido [23] [24]. De este modo se "lee" cómo las clases colaboran y cómo se debe implementar esa lógica en el código.

Escribiendo un Diagrama de Clases

Para **crear** (dibujar) un diagrama de clases a partir de un problema, conviene seguir estos pasos guía, basados en buenas prácticas de UML [26] [27]:

1. **Identificar las clases del dominio:** encuentre los posibles *sustantivos* en la descripción del sistema; estos suelen corresponder a las clases principales [26]. Por ejemplo, en un problema de biblioteca podríamos identificar `Libro`, `Usuario`, `Biblioteca`.
2. **Determinar atributos y métodos:** por cada clase, defina sus propiedades (atributos) y comportamientos (métodos) relevantes al problema [28] [10]. Las clases se representan con su nombre y luego sus atributos y métodos listados en filas separadas.
3. **Establecer relaciones:** determine cómo se vinculan esas clases entre sí. Por ejemplo, ¿hay herencia o "es un tipo de"? ¿Alguna clase contiene o usa a otra? Identifique asociaciones (relaciones simples), agregaciones o composiciones entre ellas [27] [21].
4. **Dibujar el diagrama:** dibuje cada clase como un rectángulo de tres partes y conecte con líneas y flechas según las relaciones identificadas. Asegúrese de etiquetar multiplicidades (1, * , etc.) si aplica. El resultado será un diagrama de clases UML que refleja la estructura del sistema y sirve de guía para la implementación [26] [27].

Por ejemplo, la plantilla UML de una clase contiene tres secciones (nombre, atributos, métodos) y puede completarse paso a paso con la información obtenida del análisis del problema [8] [28]. Este proceso garantiza que el diagrama refleje fielmente las necesidades del sistema antes de escribir código.

Ejemplos en Python

A continuación se muestran dos ejemplos prácticos. Cada ejemplo parte de un diagrama conceptual de clases y presenta su implementación en Python.

- **Ejemplo 1: Sistema de Pedidos.** Supongamos un sistema de comercio electrónico con clases `Customer`, `Order`, `OrderItem` y `Product`. En el diagrama, *un* cliente puede tener varios pedidos (asociación uno-a-muchos), y cada `Order` está compuesto de varios `OrderItem` (composición). Un `OrderItem` referencia a un `Product`. El código Python podría ser:

```
class Customer:
    def __init__(self, name, email):
        self.name = name
        self.email = email
        self.orders = []  # asociación: un cliente tiene varios pedidos
```

```
    def place_order(self, order):
        self.orders.append(order)

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class OrderItem:
    def __init__(self, product, quantity):
        self.product = product  # asociación con Product
        self.quantity = quantity

    def subtotal(self):
        return self.product.price * self.quantity

class Order:
    def __init__(self, order_id, date):
        self.order_id = order_id
        self.date = date
        self.items = []  # composición: un pedido contiene varios ítems

    def add_item(self, item):
        self.items.append(item)

    def total(self):
        return sum(item.subtotal() for item in self.items)

# Uso de las clases:
p1 = Product("Laptop", 1200.0)
p2 = Product("Mouse", 50.0)
order1 = Order(order_id=1, date="2025-11-29")
order1.add_item(OrderItem(p1, 1))
order1.add_item(OrderItem(p2, 2))
customer = Customer("Alice", "alice@example.com")
customer.place_order(order1)
print(f"Total del pedido de {customer.name}: {customer.orders[0].total():.
2f}")
```

En este código se refleja el diagrama: `Customer` tiene una lista de `Order` (asociación), cada `Order` tiene una lista de `OrderItem` (composición fuerte), y cada `OrderItem` conoce a qué `Product` corresponde (asociación). Los atributos (`name`, `email`, `order_id`, etc.) y métodos (`place_order`, `add_item`, `total`) de cada clase se derivan directamente del diagrama conceptual.

- **Ejemplo 2: Sistema Académico.** Consideremos un modelo de universidad con `Person`, `Student` y `Teacher` (donde `Student` y `Teacher` heredan de `Person`), más la clase `Course`. En el diagrama: un *estudiante* se inscribe en varios cursos (asociación muchos-a-muchos mediada por listas) y un *profesor* imparte cursos (asociación uno-a-muchos). Ejemplo en Python:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id
        self.courses = []  # lista de cursos en los que está inscrito

    def enroll(self, course):
        self.courses.append(course)
        course.students.append(self)  # agregamos estudiante al curso

class Teacher(Person):
    def __init__(self, name, age, employee_id):
        super().__init__(name, age)
        self.employee_id = employee_id
        self.courses = []  # lista de cursos que imparte

    def assign_course(self, course):
        self.courses.append(course)
        course.teacher = self  # asignamos profesor al curso

class Course:
    def __init__(self, title):
        self.title = title
        self.students = []  # estudiantes inscritos
        self.teacher = None  # profesor asignado

# Uso de las clases:
teacher = Teacher("Dr. Smith", 40, 1001)
student1 = Student("Alice", 20, 2001)
student2 = Student("Bob", 22, 2002)
math = Course("Math 101")
teacher.assign_course(math)    # el profesor imparte el curso
student1.enroll(math)         # estudiantes se inscriben
student2.enroll(math)
print(f"Curso: {math.title}")
print(f"Profesor: {math.teacher.name}")
print("Estudiantes inscritos:", [s.name for s in math.students])
```

Aquí se muestra la **herencia** (`Student` y `Teacher` extienden `Person`) y las asociaciones con `Course`. El diagrama UML correspondiente tendría flechas de herencia apuntando de `Student`/`Teacher` hacia `Person`, y líneas que conectan `Course` con sus estudiantes y profesor (con multiplicidad 0..* para estudiantes). Los atributos (como `name`, `student_id`, `title`) y métodos (`enroll`, `assign_course`) siguen lo definido en el diagrama.

Estos ejemplos ilustran cómo un diagrama de clases bien diseñado se traduce en una implementación orientada a objetos en Python: cada clase del diagrama corresponde a una clase en el código, y las relaciones (composición, asociación, herencia) se implementan usando listas de objetos, herencia de clases, etc. De esta forma, el diagrama sirve como plano para el desarrollo del software.

Fuentes: El contenido se basa en definiciones y guías UML de fuentes educativas y técnicas [1] [29] [26] [30]. Los ejemplos de código ilustran implementaciones típicas de las relaciones conceptuales mostradas en los diagramas.

---

[1] [24] Diagrama de clases - Wikipedia, la enciclopedia libre
https://es.wikipedia.org/wiki/Diagrama_de_clases

[2] [5] [8] [10] [14] [20] [21] [23] [26] [27] [28] [29] [30] Guía completa para entender el diagrama de clases UML básico | Boardmix
https://boardmix.com/es/knowledge/class-diagram/

[3] [4] [6] [13] [15] [17] [22] [25] Diagramas de clases :: IES San Clemente
https://manuais.pages.iessanclemente.net/plantillas/DUAL/cd/ud05/3.diagramas_clases/index.print.html

[7] [16] Class Diagram for Library Management System - GeeksforGeeks
https://www.geeksforgeeks.org/software-engineering/class-diagram-for-library-management-system/

[9] [11] [12] [18] [19] Tutorial de diagrama de clases UML | Lucidchart
https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml