

Unified Modeling Language (UML)

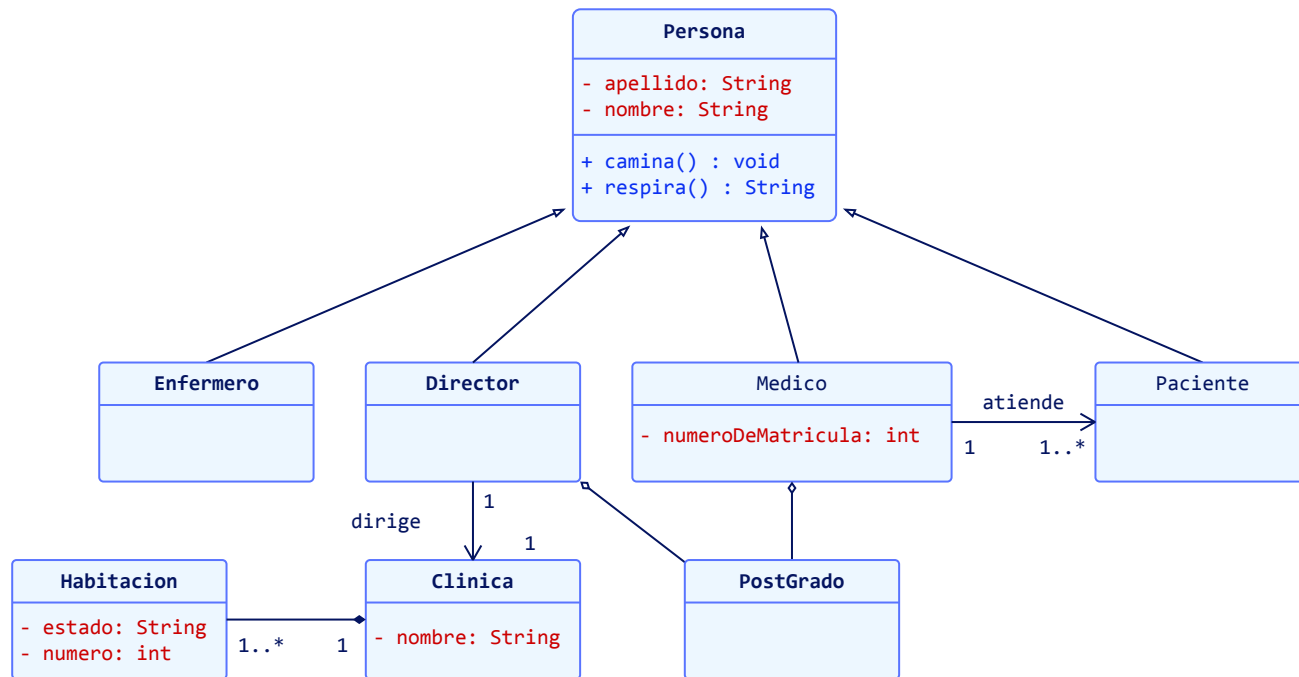
Unified Modeling Language

UML es un lenguaje visual estándar para la especificación, construcción y documentación del *software*.

Es ampliamente utilizado en el desarrollo de *software* orientado a objetos y desempeña un papel crucial a la hora de **diseñar y visualizar sistemas complejos**.

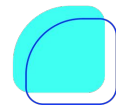


Ejemplo:



Características

- Proporciona una **notación gráfica** para representar diferentes aspectos de un sistema de *software*.
- Facilita la **comunicación** entre los miembros del equipo.
- Permite **capturar y documentar** los requisitos y el diseño del sistema de manera clara y concisa.
- Ayuda a **comprender y visualizar** la estructura, el comportamiento y las interacciones del sistema antes de su implementación.
- Es **independiente** del lenguaje de programación y la tecnología que se utilicen.



La unificación del modelado de objetos

En la década de 1990, existían varios métodos de modelado de objetos diferentes y propietarios. **La falta de un lenguaje de modelado estándar dificultaba la comunicación.**

En **1997**, **Grady Booch** (metodologista de *software*), **James Rumbaugh** (ingeniero de *software*) e **Ivar Jacobson** (ingeniero en Ciencias de la Computación) se unieron a *Rational Software Corporation* para **crear un lenguaje de modelado unificado.**

El [Object Management Group \(OMG\)](#) lanzó **UML 1.0** oficialmente, en 1997. UML **se convirtió rápidamente en un estándar** de facto para el modelado de *software* orientado a objetos.



El ***Object Management Group (OMG)*** es un consorcio internacional, sin fines de lucro, dedicado al establecimiento de estándares de *software*. Su objetivo es promover el uso de tecnologías orientadas a objetos a través del desarrollo de estándares abiertos.

Diagramas UML

UML proporciona una variedad de diagramas para modelar diferentes aspectos del *software*.

Diagramas de estructura	Diagramas de comportamiento	Diagramas de interacción
<ul style="list-style-type: none">▪ Diagrama de clases.▪ Diagrama de objetos.▪ Diagrama de componentes.▪ Diagrama de paquetes.	<ul style="list-style-type: none">▪ Diagrama de casos de uso.▪ Diagrama de actividades.▪ Diagrama de estado.▪ Diagrama de secuencia.	<ul style="list-style-type: none">▪ Diagrama de comunicación.▪ Diagrama de tiempos.

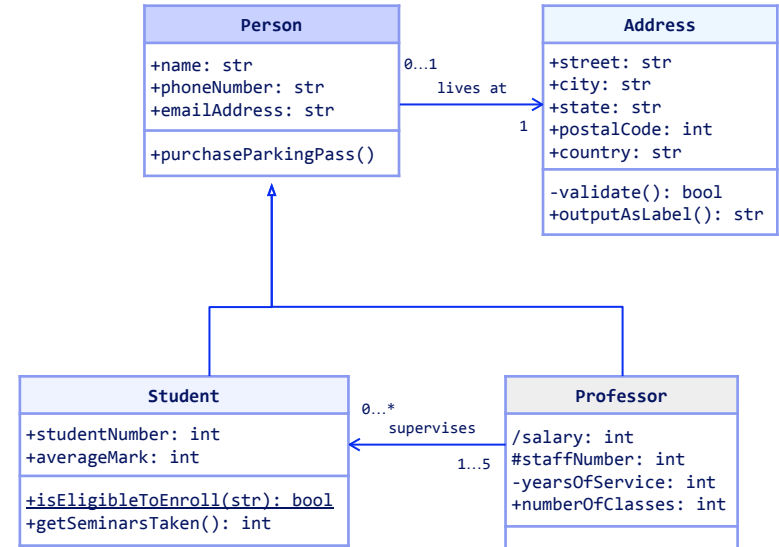
En este curso, nos centraremos en el **Diagrama de clases** que representa la estructura estática de un sistema, muestra clases, atributos, operaciones y relaciones entre clases.



Diagrama de clases

Es uno de los **diagramas más importantes** y ampliamente utilizados en UML.

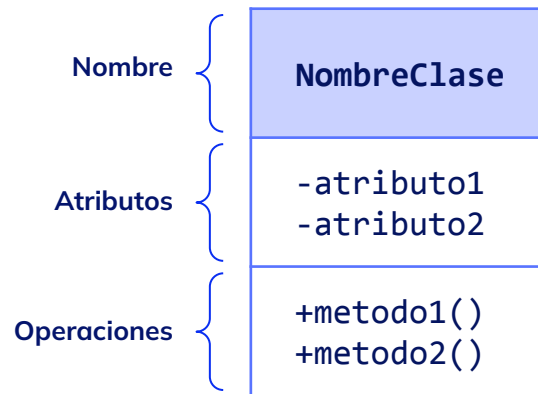
- Representa la **estructura estática** de un sistema.
- **Muestra clases**, atributos, operaciones y las relaciones entre ellas.
- Ayuda a **visualizar y explorar** soluciones antes de la implementación.
- A partir de este diagrama, es posible **generar automáticamente el código** fuente en un lenguaje orientado a objetos.



Clase

Como vimos anteriormente una clase es una **plantilla que nos permitirá crear objetos** de un tipo determinado.

Las clases se representan como **rectángulos con tres secciones**:



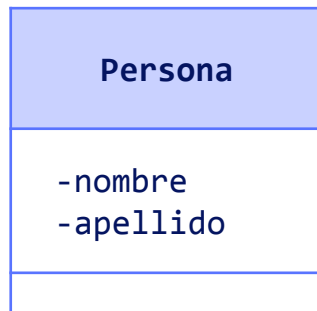
Salvo el nombre, los rectángulos de **atributos y operaciones son opcionales** y dependen del grado de detalle que se le quiera dar al diagrama.

Atributos

Características o propiedades que poseen los objetos de dicha clase.

Sintaxis:

```
[visibilidad] nombre [: tipo] [= valor_inicial]
```



- **Visibilidad (opcional):** Representada por un carácter.

+ Público.

- Privado.

Protegido.

~ Paquete.

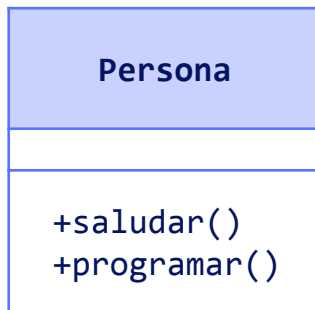
- **Nombre (obligatorio):** Identificador del atributo.
- **Tipo de dato del atributo (opcional):** Depende de él la implementación o el lenguaje elegido.
- **Valor inicial del atributo (opcional).**

Operaciones

Acciones o métodos que los objetos de dicha clase pueden realizar.

Sintaxis:

```
[visibilidad]nombre(parametros) [:tipo_retorno]
```

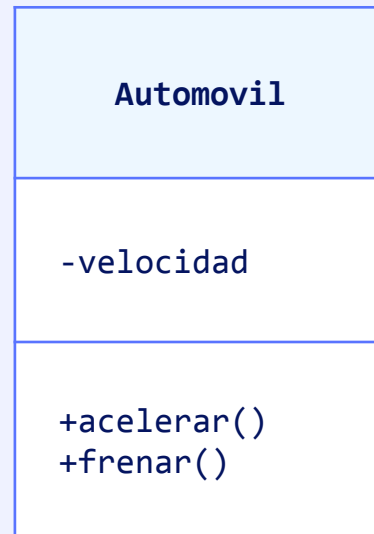


- **Visibilidad (opcional):** Representada por un carácter al igual que los atributos.
- **Nombre de la operación (obligatorio).**
- **Parámetros (opcional):** Lista de parámetros de la operación separado con comas, con sus tipos.
- **Tipo de retorno (opcional):** Tipo de dato que retorna la operación.

Ejemplo

El diagrama muestra una clase llamada **Automovil**. A simple vista el diagrama representa una **abstracción de la vida real**.

- Tiene **un atributo**:
 - **velocidad**: Aquí se guarda la velocidad actual del automóvil.
- Tiene **dos métodos**:
 - **acelerar()**: Este método simboliza la acción de aumentar la velocidad del automóvil.
 - **frenar()**: Este método representa la acción de reducir la velocidad del automóvil.



Relaciones

Conexiones entre clases que definen **cómo se relacionan entre sí**.

Existen distintos tipos:

Asociación

Relación general entre clases.

- Agregación.
- Composición.

Herencia

Mecanismo para compartir estructura y comportamiento entre clases.

Dependencia

Relación en la que un cambio en una clase puede afectar a otra.

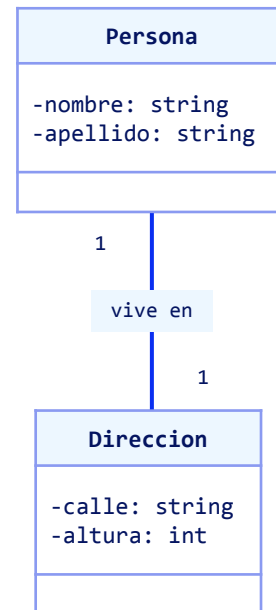
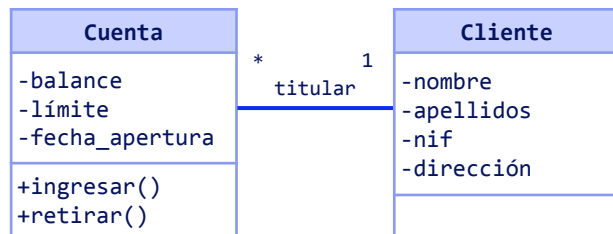
Las relaciones se muestran mediante **líneas y símbolos específicos** (flechas, rombos y otros).

Las relaciones en UML pueden tener:

- **Nombre:** Define un identificador único para la relación.
- **Cardinalidad:** Especifica el número de instancias de una clase que pueden estar relacionadas con una instancia de la otra clase.
- **Navegabilidad:** Indica si es posible acceder a los objetos de una clase desde la otra a través de la relación.
- **Un rol en cada extremo:** Define el papel o función que desempeña una clase en el contexto de la relación.
- **Nivel de visibilidad:** Determina el alcance o visibilidad de la relación: pública, privada, protegida o de paquete.



Ejemplos:



Modelado con LLM y lenguaje natural

Mermaid

Es un lenguaje y una herramienta para crear **diagramas y visualizaciones a partir de texto simple** utilizando una sintaxis fácil de escribir y fácil de leer.



Es un proyecto de código abierto que permite generar:

- Diagramas de flujo.
- Diagramas de secuencia.
- Diagramas de clases.
- Diagramas de estado.
- Diagramas de Gantt.
- ... y otros.

Para generar un primer diagrama, ingresar a [Mermaid.live](https://mermaid.live).

Características

- **Portabilidad:** Los diagramas se representan como texto, lo que facilita su inclusión en otros documentos.
- **Simplicidad:** Sintaxis es sencilla y fácil de aprender.
- **Integración:** Mermaid se puede integrar en diferentes entornos y herramientas.
- **Flexibilidad:** Admite una amplia variedad de diagramas.
- **Colaboración:** Se pueden compartir y editar fácilmente.
- **Uso en diagramas de clases UML:** Mermaid ofrece una sintaxis específica para crear diagramas de clases UML.
- **Generación desde ChatGPT:** Al ser texto se puede utilizar ChatGPT tanto para crear o modificar diagramas de clases.



Modelado con LLM y lenguaje natural

Podemos utilizar ChatGPT o cualquier LLM para **generar diagramas de clases a partir del lenguaje natural**.

Al proporcionar los requisitos del sistema, ChatGPT analiza el texto, identifica clases, atributos, operaciones y relaciones, y **genera un diagrama de clases correspondiente en notación UML o sintaxis de Mermaid**.

Esta funcionalidad facilita la **transición de requisitos a representaciones visuales** estructuradas en el diseño de sistemas orientados a objetos.



Ventajas utilizar IA a la hora de codificar

- **Comprensión de lenguaje natural:** Capacidad para entender descripciones y requisitos expresados en lenguaje natural.
- **Identificación de elementos clave:** Detección e identificación de clases, atributos, operaciones y relaciones a partir de texto.
- **Generación de diagramas:** Capacidad para generar diagramas de clases en notación UML o sintaxis de Mermaid.
- **Manejo de detalles implícitos:** Interpretación y procesamiento de detalles y contexto implícitos en las descripciones.
- **Adaptabilidad a diferentes dominios:** Capacidad para comprender y generar diagramas de clases en diversos dominios y campos de aplicación.
- **Iteración y refinamiento:** Permiten realizar iteraciones y refinar los diagramas generados mediante interacciones y retroalimentación adicional.
- **Integración con herramientas:** Posibilidad de integrar los LLM con herramientas de desarrollo para facilitar la generación de diagramas de clases.

Utilizar ChatGPT para generar un diagrama de clases a partir de un enunciado

Copia y pega el siguiente *prompt*:

Generar un diagrama de clases en Mermaid para un sistema de gestión de bibliotecas, que incluya clases para libros, usuarios, préstamos y multas. Los libros tienen título, autor y año de publicación. Los usuarios tienen nombre y dirección. Los préstamos registran el libro prestado, el usuario y las fechas de préstamo y devolución. Las multas se calculan en función de los días de retraso en la devolución.



ChatGPT generará el diagrama Mermaid solicitado.

- Se pueden **solicitar modificaciones** y experimentar realizando cambios.
- Cuando se esté conforme con el resultado, **copiar el código** y pegarlo en la web [Mermaid.live](https://mermaid.live).

Nota: Los resultados que brinda el LLM, en base a un mismo *prompt*, no siempre son iguales. El código que obtengas podría mostrar algunas diferencias con el de esta imagen.

```
mermaid Copiar código

classDiagram
    class Libro {
        - String título
        - String autor
        - int añoPublicación
    }

    class Usuario {
        - String nombre
        - String dirección
    }

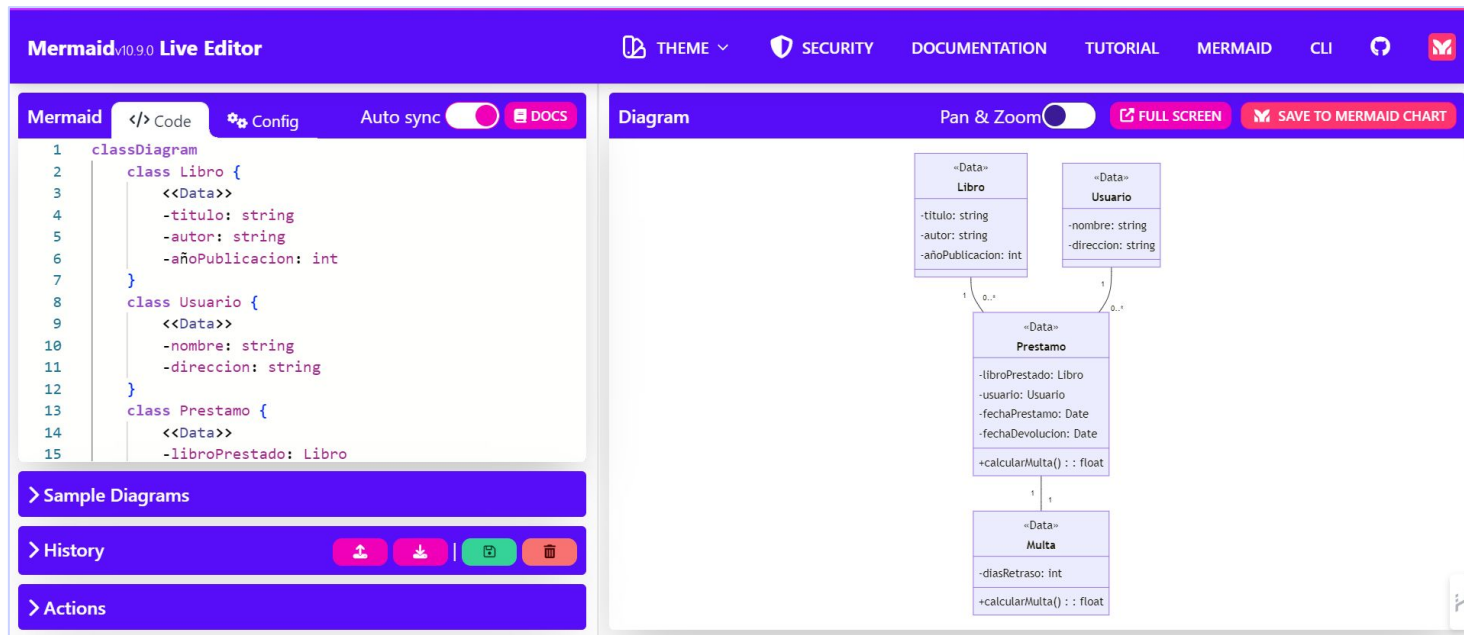
    class Prestamo {
        - Date fechaPrestamo
        - Date fechaDevolucion
        + calcularMulta() : double
    }

    class Multa {
        - int diasRetraso
        - double monto
        + calcularMonto(int diasRetraso) : double
    }

    Libro "1" --> "0..*" Prestamo : prestado
    Usuario "1" --> "0..*" Prestamo : realiza
    Prestamo "1" --> "0..1" Multa : genera
```

Ahora puedes **visualizar el diagrama Mermaid** con la herramienta.

ChatGPT puede cometer errores, en ese caso, se deberá modificar el diagrama a mano.



Ahora que se conoce cómo se ve el modelo de negocios, se puede utilizar **ChatGPT** para que lo **convierta en código del lenguaje que se elija**.



Tú

¿Puedes convertir el diagrama en clases de Java?



ChatGPT

Claro, aquí tienes una representación de las clases en Java basadas en el diagrama de clases que proporcioné anteriormente:



Ahora resta llevar el código al IDE que se prefiera.



java

 Copiar código

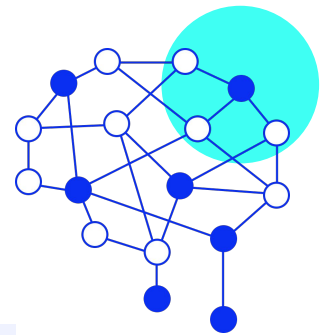
```
import java.time.LocalDate;

public class Libro {
    // Atributos
    private String titulo;
    private String autor;
    private LocalDate fecha;

    // Constructor
    public Libro(String titulo, String autor, LocalDate fecha) {
        this.titulo = titulo;
        this.autor = autor;
        this.fecha = fecha;
    }
}
```


La **inteligencia artificial**, a través de los LLM como ChatGPT, está **revolucionando la forma en que se diseñan y programan sistemas de software**.

La capacidad de estos modelos para comprender requisitos en lenguaje natural y generar diagramas de clases de manera automática, abre nuevas oportunidades para **agilizar el proceso de diseño y mejorar la calidad de los sistemas** orientados a objetos.



Conclusiones

Esta tecnología no solo ahorra tiempo y esfuerzo, sino que también **facilita la comunicación y la colaboración** entre los miembros del equipo, al permitir la exploración de soluciones de diseño de manera más intuitiva y accesible.

Sin embargo, es importante recordar que la inteligencia artificial es una herramienta poderosa, pero **no reemplaza el conocimiento y la experiencia de los desarrolladores**.

Es nuestro papel como profesionales aprovechar al máximo estas tecnologías y **combinarlas con nuestras habilidades** y criterio para crear sistemas de *software* de alta calidad y valor.

Explora las capacidades de ChatGPT y los Modelos de Lenguaje Grandes, y descubre cómo pueden revolucionar tu enfoque al diseño y desarrollo de sistemas orientados a objetos.