

Modelando el mundo real con clases: UML + Python






👋 Bienvenida y contexto:

¿Te ha pasado que quieres comenzar un proyecto y no sabes cómo estructurarlo?

Hoy aprenderás a pensar como un arquitecto del software usando diagramas de clases UML.

Objetivos de Aprendizaje

 Al finalizar esta clase serás capaz de:

-  Comprender qué es un diagrama de clases UML y para qué sirve.
-  Leer y crear diagramas de clases.
-  Identificar correctamente relaciones entre clases: asociación, herencia, composición, agregación.
-  Representar un modelo orientado a objetos en Python.
-  Aplicar estos diagramas para diseñar mejores sistemas antes de programar.

¿Qué es un Diagrama de Clases?

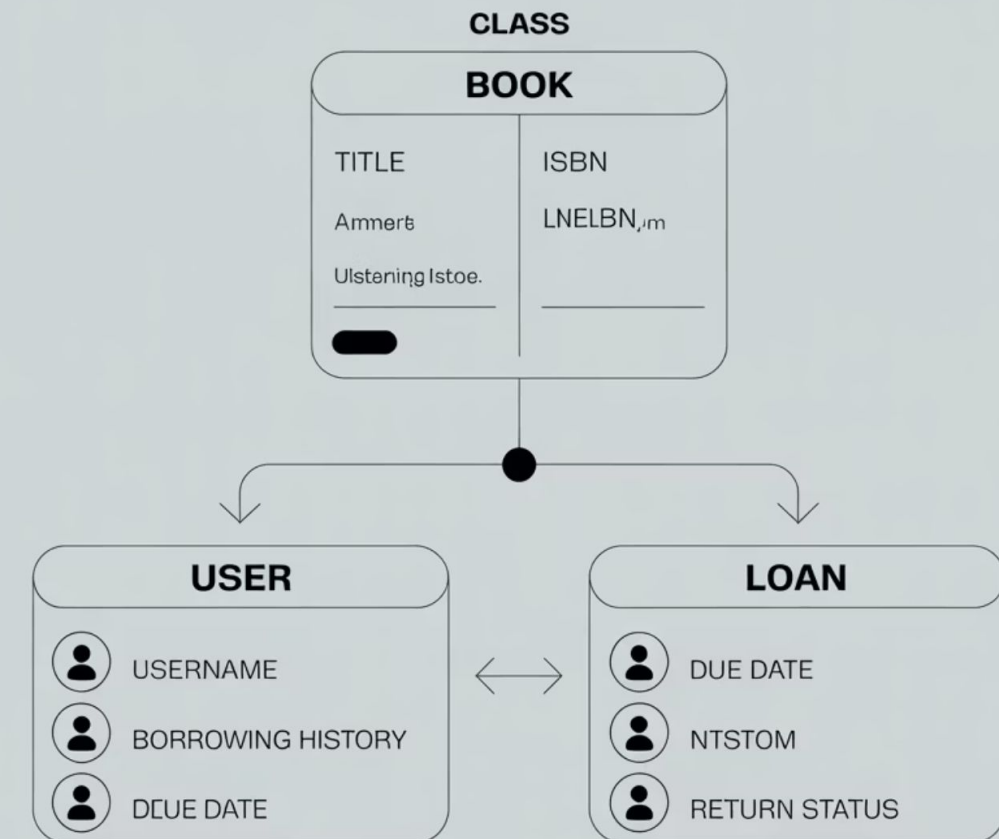
 Es un dibujo estructurado que representa:

- Las clases de un sistema.
- Sus atributos y métodos.
- Las relaciones entre ellas.

 Ejemplo del mundo real: En una biblioteca:

Clase Libro, Clase Usuario, Clase Préstamo.

Podemos mostrar visualmente cómo se relacionan.



¿Para qué sirve?



Planificación

Planificar la estructura del software antes de escribir código.



Comunicación

Comunicar ideas complejas de forma visual.



Prevención

Identificar problemas de diseño antes de que existan.



Colaboración

Trabajar colaborativamente en equipos técnicos y no técnicos.

🎓 Analogía: Es como el plano de una casa antes de construir.

Estructura Básica de una Clase UML

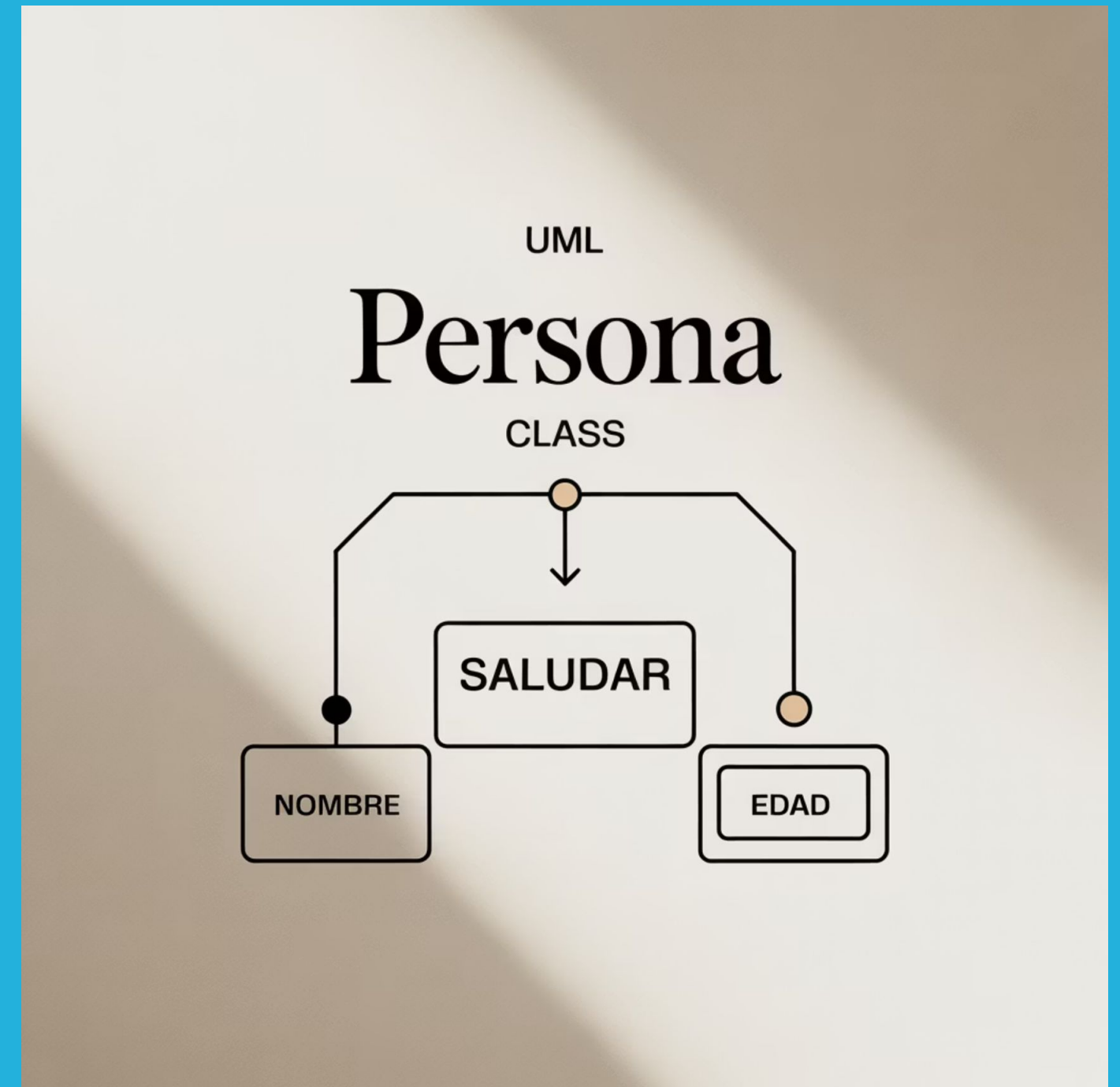
Una clase se representa con un rectángulo dividido en 3 partes:

1. Nombre de la clase (en la parte superior)
2. Atributos (al medio)
3. Métodos (abajo)

Notación:

- + público
- - privado
- # protegido

```
+-----+
|      Persona      |
+-----+
| - nombre: str     |
| - edad: int       |
+-----+
| + saludar(): str  |
+-----+
```



Atributos y Métodos

🧩 Los atributos son los datos que posee la clase (variables).

🔧 Los métodos son las acciones o comportamientos (funciones).

📖 UML:

```
- nombre: str  
+ saludar(): str
```

📌 Ejemplo:

```
class Persona:  
    def __init__(self, nombre):  
        self.nombre = nombre  
    def saludar(self):  
        return f"Hola, soy {self.nombre}"
```


Visibilidad en UML

 ¿Qué significan los símbolos?

Símbolo	Visibilidad	¿Dónde se puede usar?
+	Público	En cualquier parte del programa
-	Privado	Solo dentro de la misma clase
#	Protegido	Clase y sus subclases

 En Python se usa por convención:

`self.__atributo` (privado)

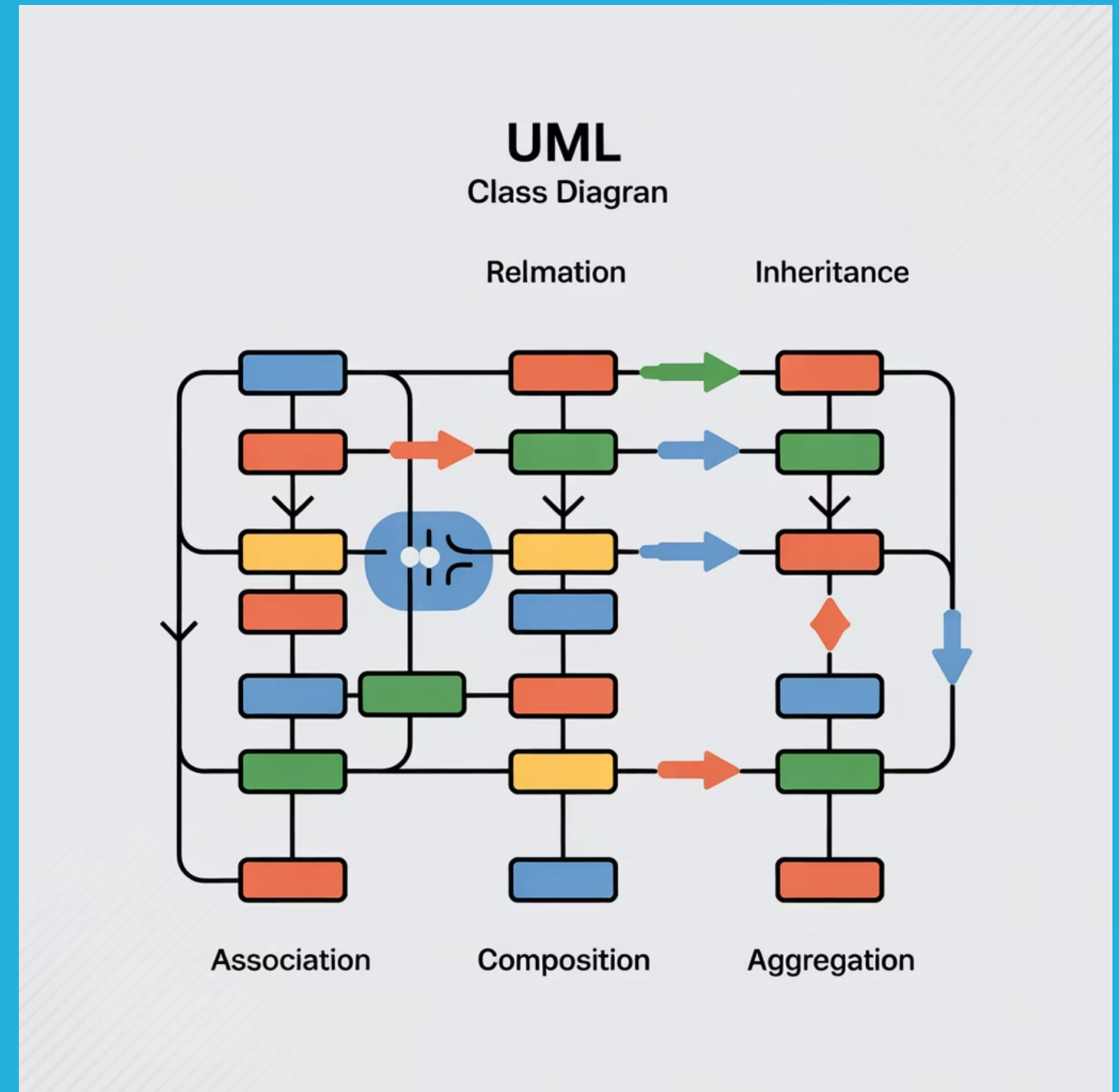
`self._atributo` (protegido)

Relaciones entre Clases – Panorama General

🔗 Las clases no están solas, se relacionan. Las relaciones indican cómo interactúan entre sí.

📌 Tipos:

- Asociación: "Trabajan juntas".
- Herencia: "Una clase es una versión especializada de otra".
- Composición: "Una clase contiene otra de forma vital".
- Agregación: "Una clase contiene otra de forma débil".



Asociación

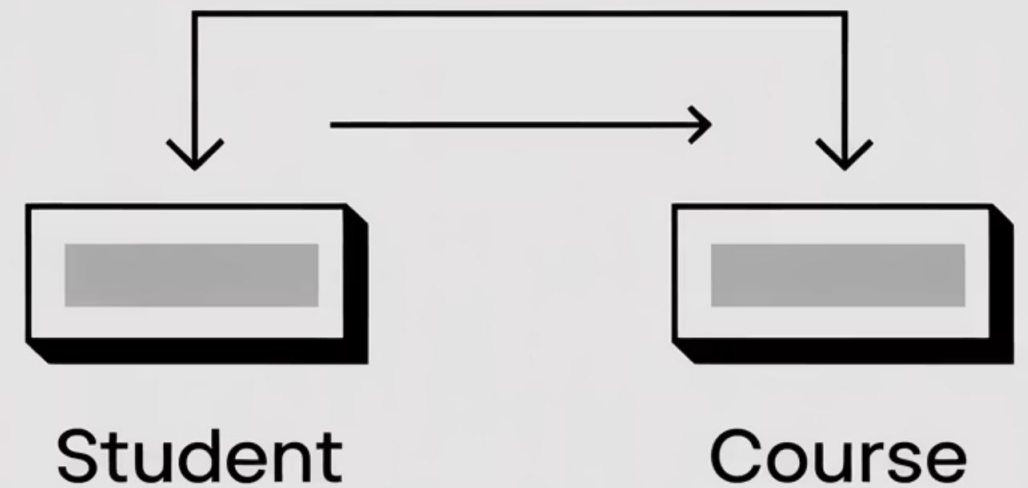
📖 Relación básica. Ej: Un Estudiante se inscribe en un Curso.

🖼️ UML:


Estudiante ----- Curso

📌 Código:

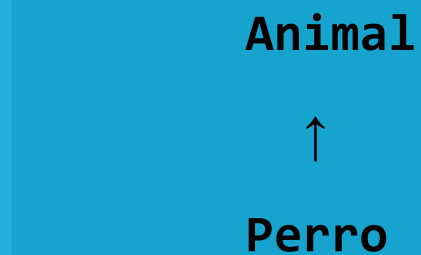
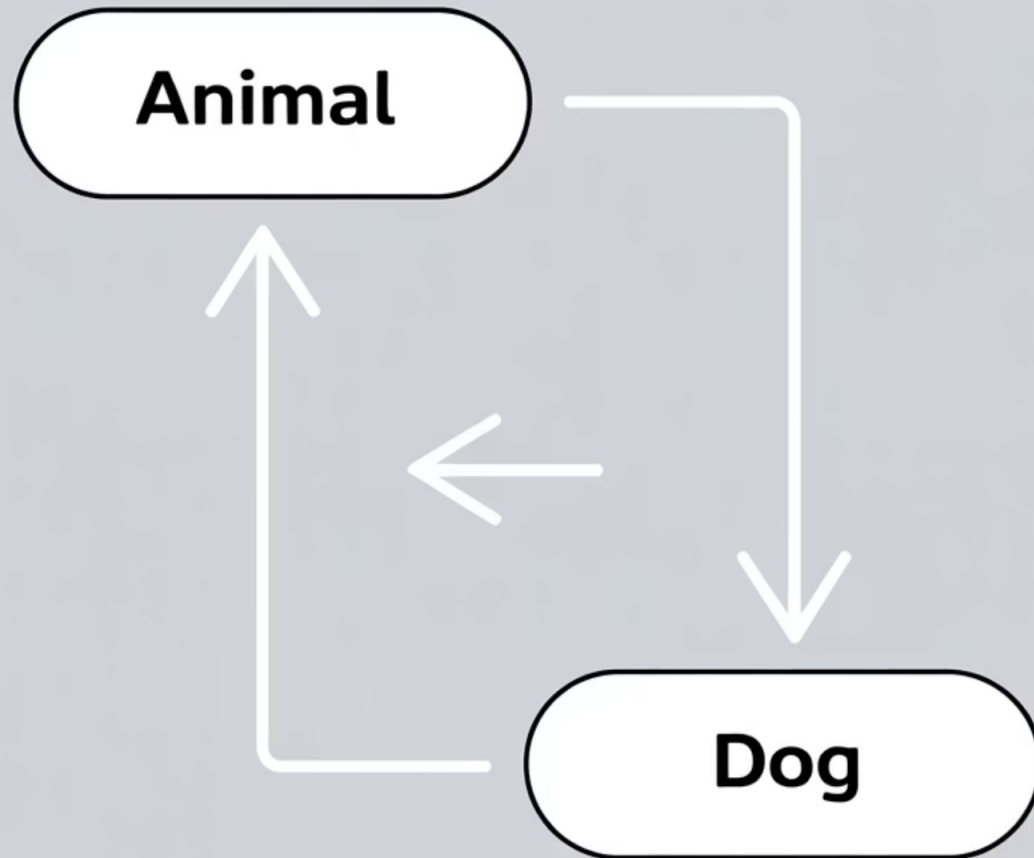
```
class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre
class Curso:
    def __init__(self, nombre):
        self.nombre = nombre
```



Herencia (Generalización)

 Una clase hereda atributos y métodos de otra.

 UML:



```
graph BT; Perro --> Animal;
```

A simplified UML diagram showing inheritance. It consists of two text labels: 'Animal' at the top and 'Perro' at the bottom. A small upward-pointing arrow is positioned between them, indicating that 'Perro' inherits from 'Animal'.

 Código:

```
class Animal:
    def dormir(self):
        pass

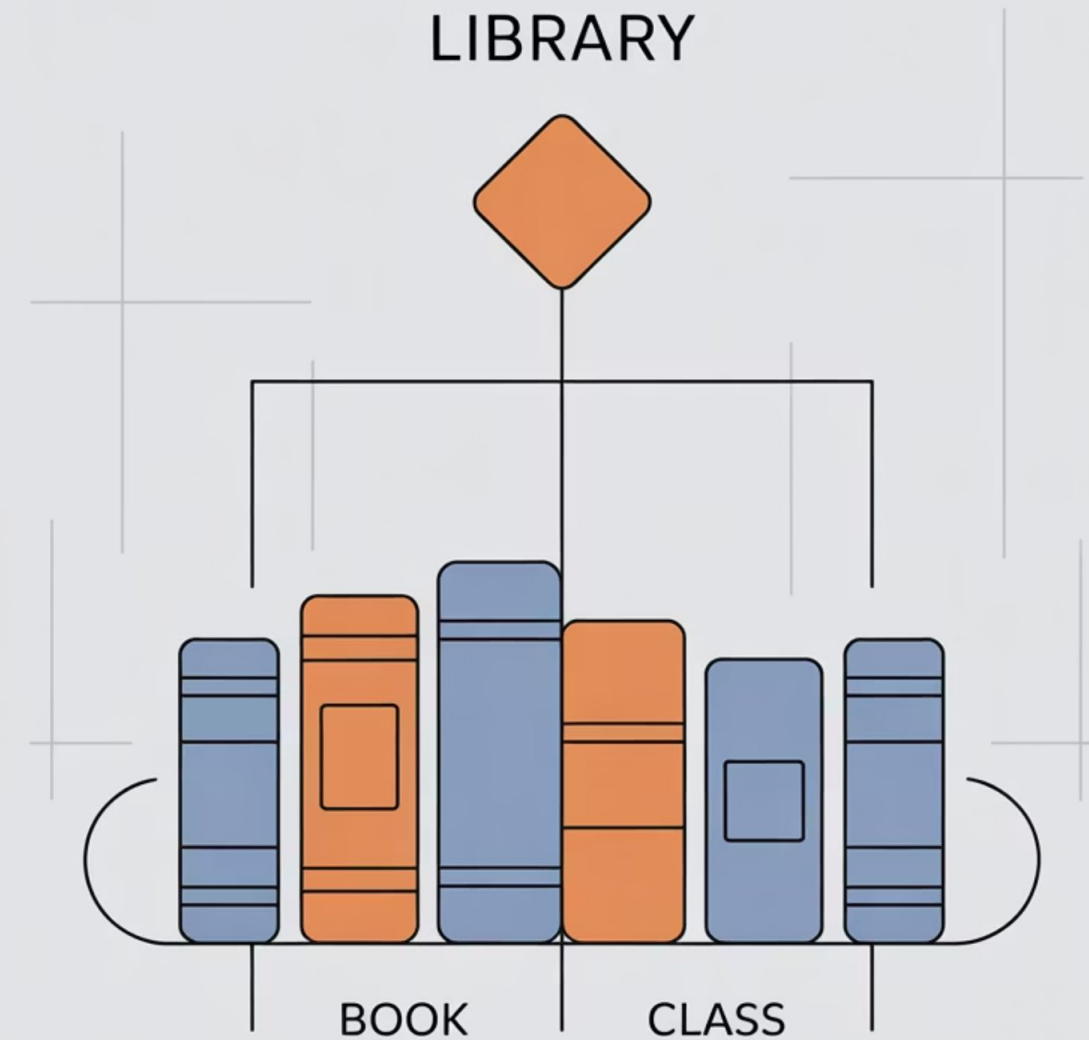
class Perro(Animal):
    def ladrar(self):
        pass
```

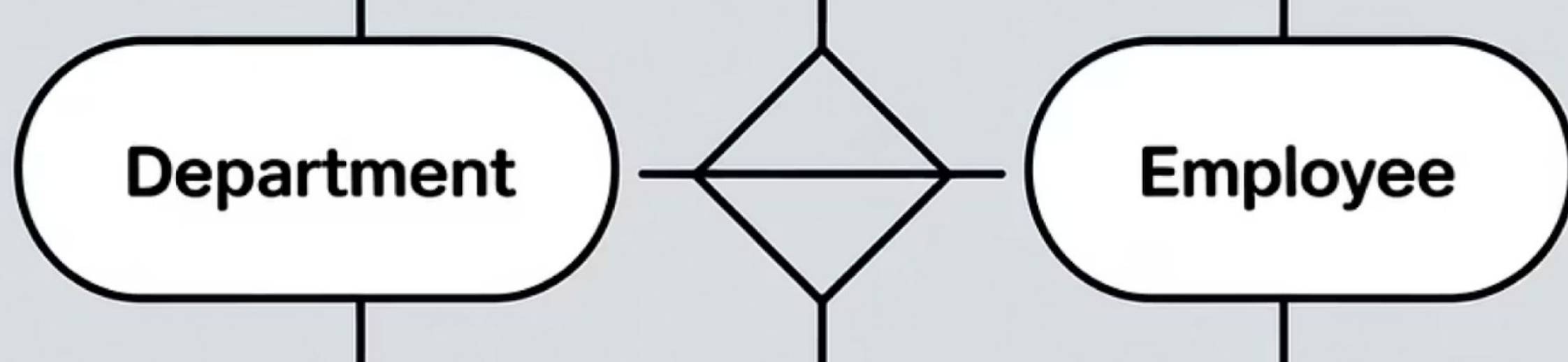
Composición

🧱 Una clase contiene otra y depende de ella. Si se elimina la clase contenedora, también desaparece la contenida.

🖼️ UML:

Biblioteca ← Libro





Agregación

🔄 Relación parecida a la composición, pero más flexible. El objeto contenido puede vivir por sí solo.

🖼️ UML:

Departamento ○— Empleado

📌 Código:

```
class Empleado:
    def __init__(self, nombre):
        self.nombre = nombre

class Departamento:
    def __init__(self):
        self.empleados = []
```

Cómo Crear un Diagrama de Clases (Paso a Paso)

Identificar Entidades

Identifica las entidades clave.

Definir Características

Define sus atributos y métodos.

Establecer Relaciones

Establece las relaciones.

Diagramar

Usa una herramienta como draw.io para diagramar.

 Consejo: Empieza con lápiz y papel si es tu primera vez.

Diseño a partir de una Situación Real

Caso: Sistema de Biblioteca

Clases: Usuario, Libro, Prestamo

Relación:

- Un Usuario puede tener muchos Préstamos (asociación).
- Cada Préstamo contiene un Libro (composición).



De UML a Python

 Mostrar cómo convertir un diagrama a código Python.

```
class Usuario:  
    def __init__(self, nombre):  
        self.nombre = nombre  
        self.prestamos = []
```

```
class Prestamo:  
    def __init__(self, libro):  
        self.libro = libro
```

```
class Libro:  
    def __init__(self, titulo):  
        self.titulo = titulo
```

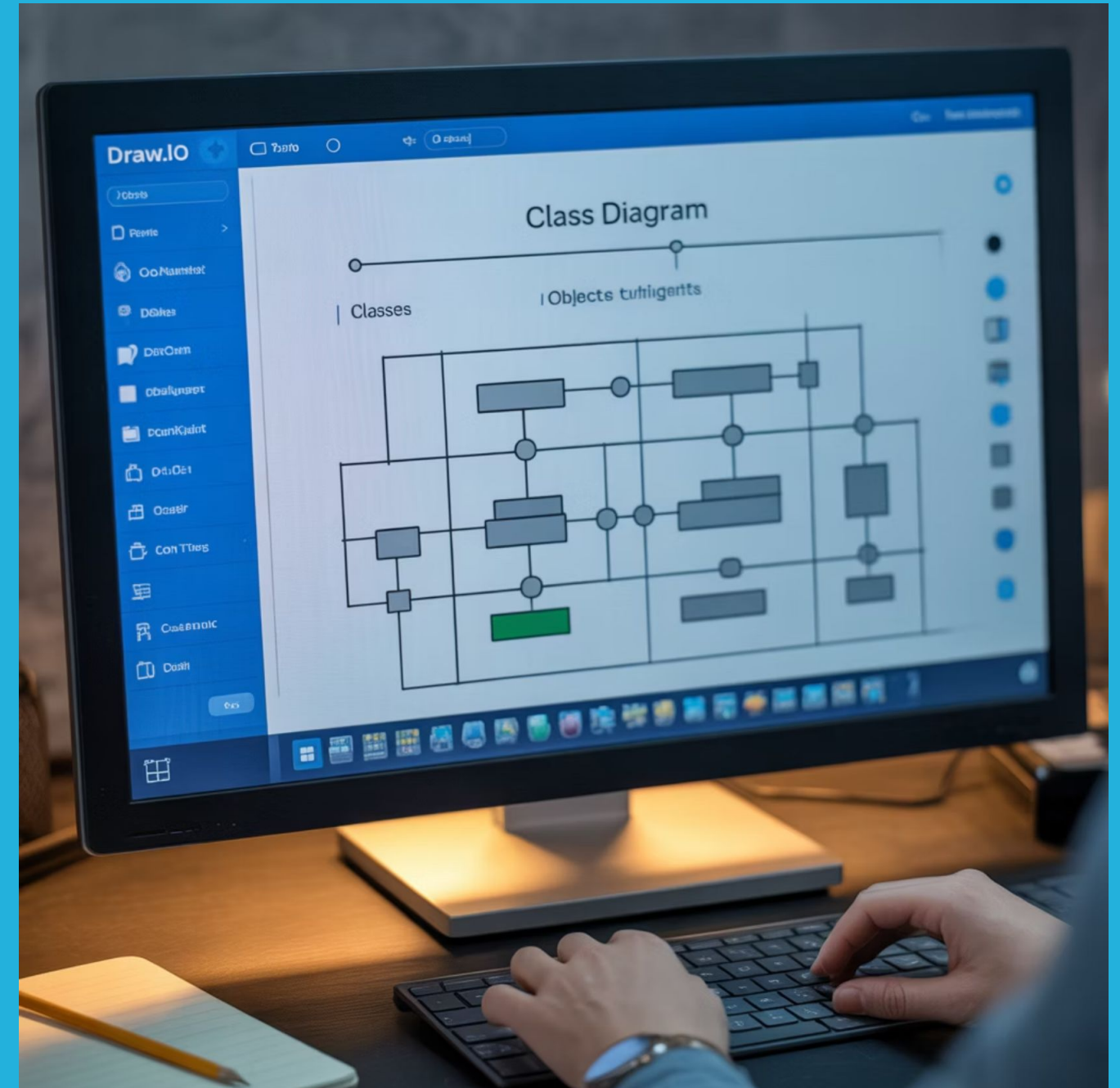


Diagrama de Clases + Draw.io en Vivo

 Mostrar cómo crear un diagrama desde cero en draw.io.

Elementos:

1. Crear 3 clases.
2. Agregar atributos y métodos.
3. Establecer relaciones.
4. Guardar y exportar.



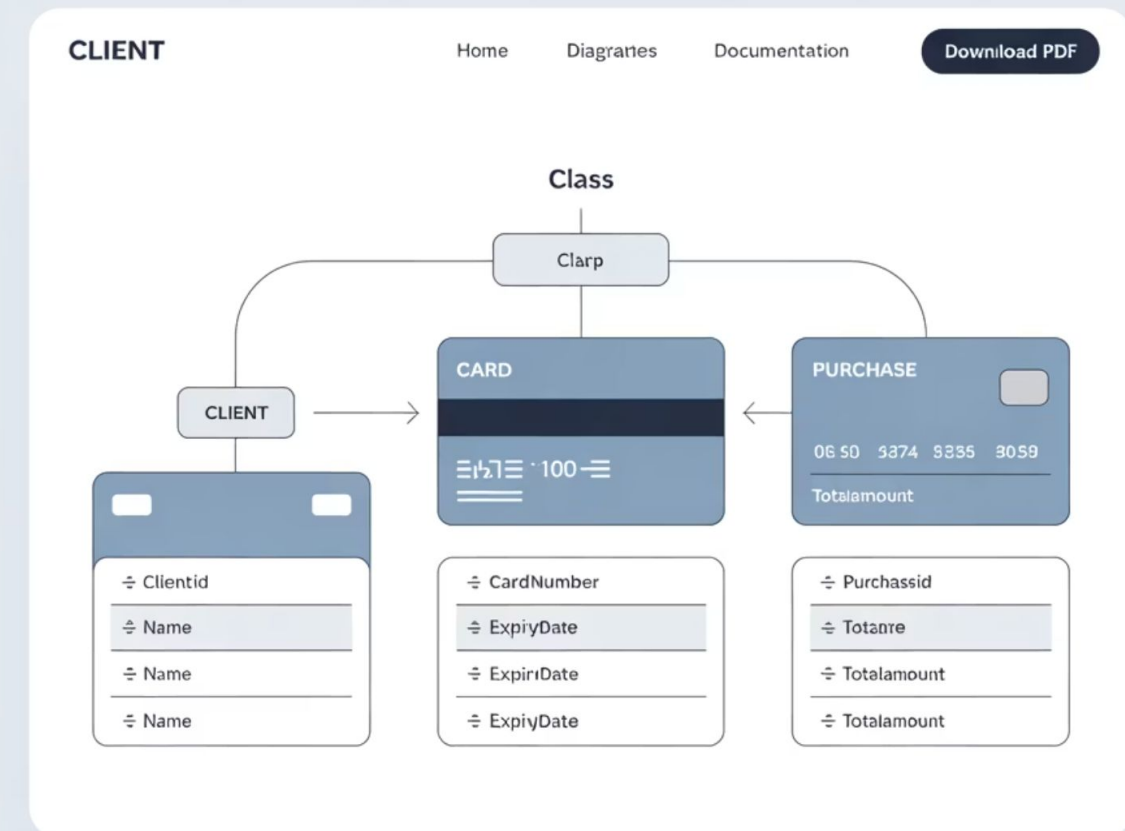
Actividad Práctica (Individual o Parejas)

📌 Enunciado: Modela un sistema simple con:

- Cliente (nombre, email)
- Tarjeta (número, banco)
- Compra (monto, fecha)

Relaciones:

- Cliente tiene 1 tarjeta (composición)
- Cliente realiza muchas compras (asociación)



Revisión en Vivo del Ejercicio

✓ Compartir soluciones

¿Están bien representadas las relaciones?

Verificar si la composición y asociación están correctamente dibujadas.

¿Faltan atributos o métodos?

Revisar si se han incluido todos los elementos necesarios.

¿Podría mejorarse algo del diseño?

Analizar posibles optimizaciones en la estructura.

Recursos Complementarios

Libros:

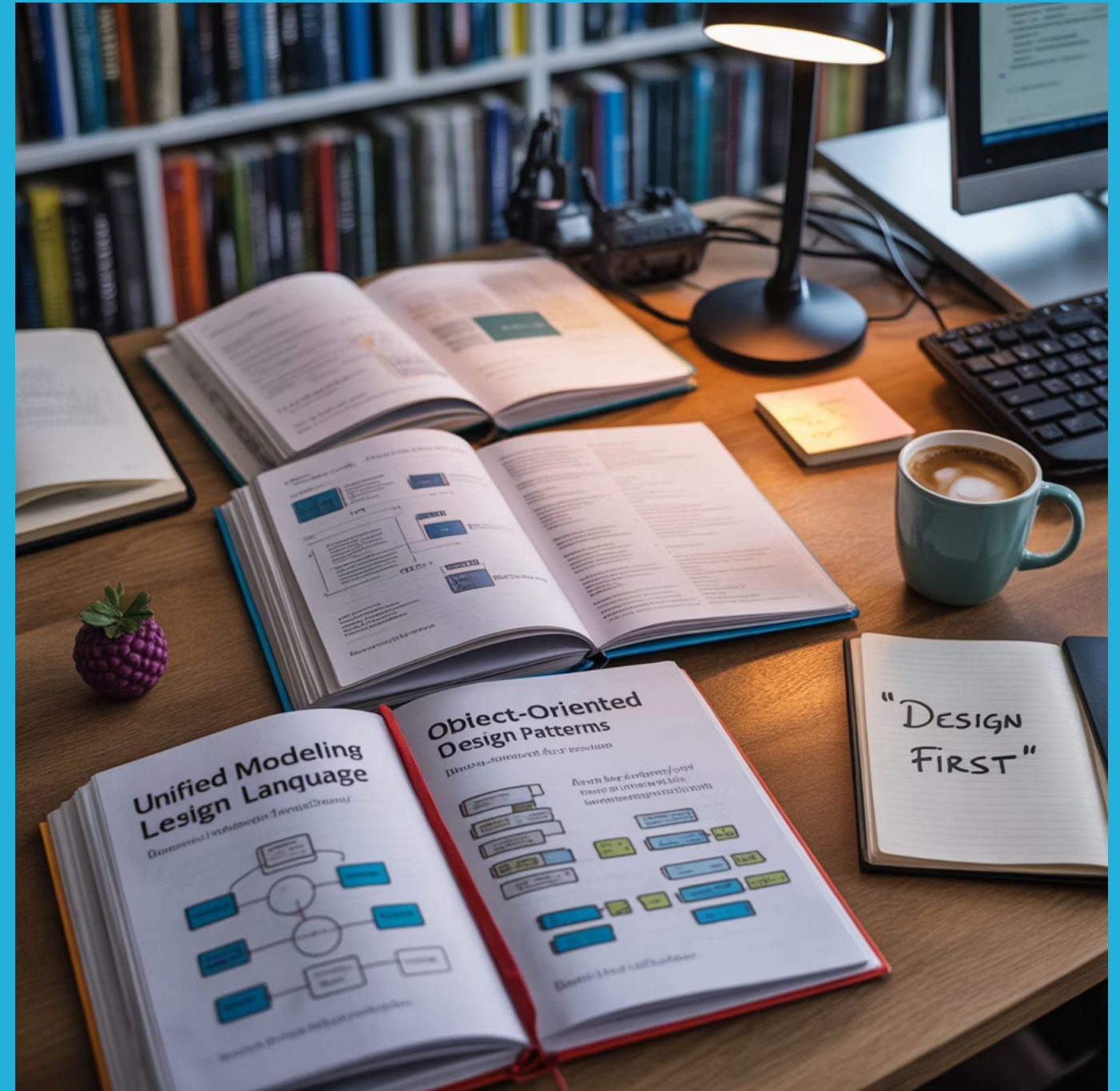
- UML Distilled – Martin Fowler
- Head First OOAD – McLaughlin

Herramientas:

- draw.io
- PlantUML
- Pyreverse

Sitio Bootcamp:

[Plataforma SkillNest](#)



Cierre y Reflexión

 ¿Qué te llevas de esta clase?

Claridad

Modelar antes de programar = claridad.

Puente

Diagramas son el "puente" entre la idea y el código.

Realidad

¡Es como pensar en objetos del mundo real!

 **Pregunta final:**

¿Qué sistema podrías modelar con clases? ¿Tu emprendimiento? ¿Tu app favorita?