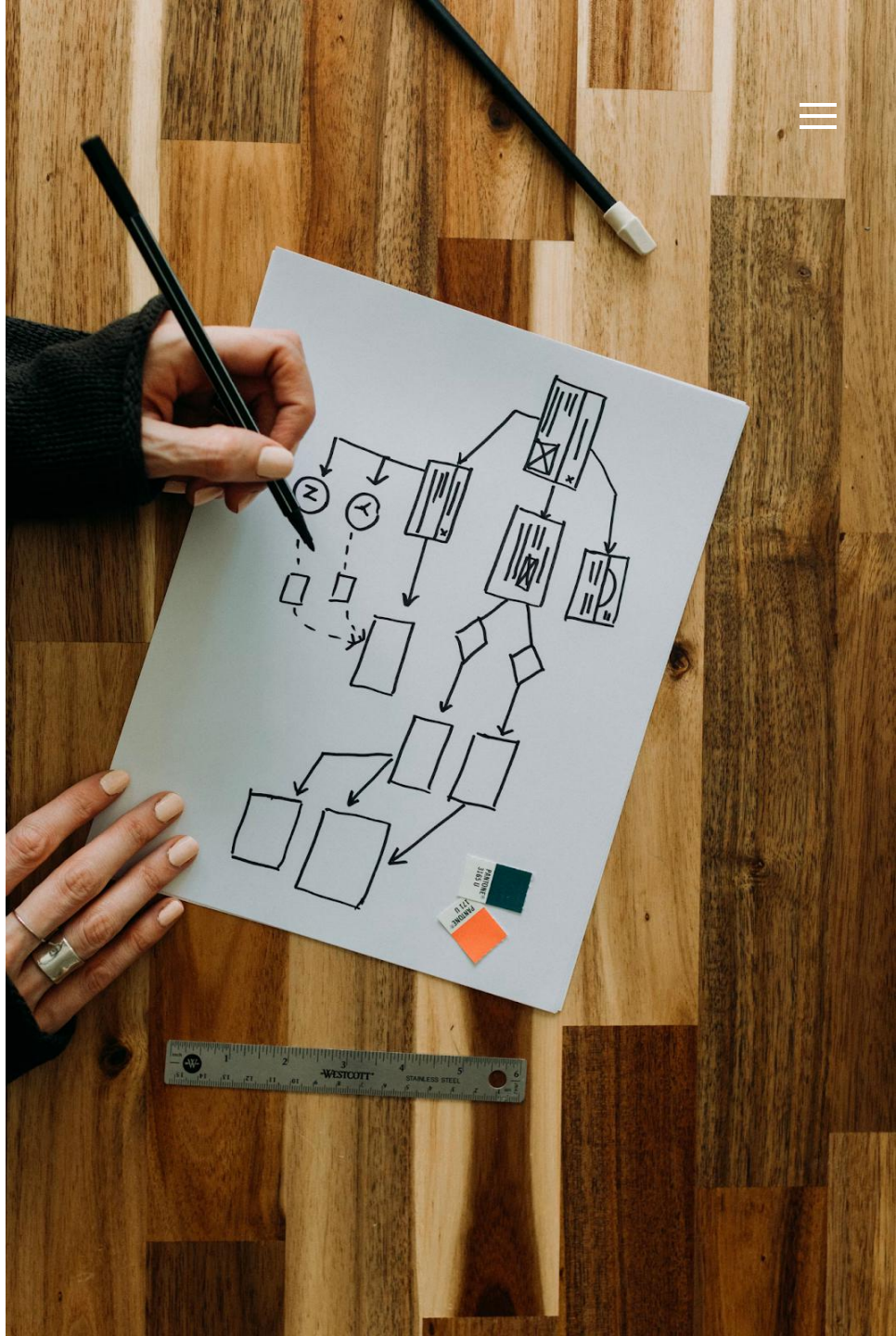
U1. Modelado

Programación Orientada a Objetos

Ingenieria en Tecnologias de la Información e Innovación Digital

Universidad Politecnica del Estado de Morelos



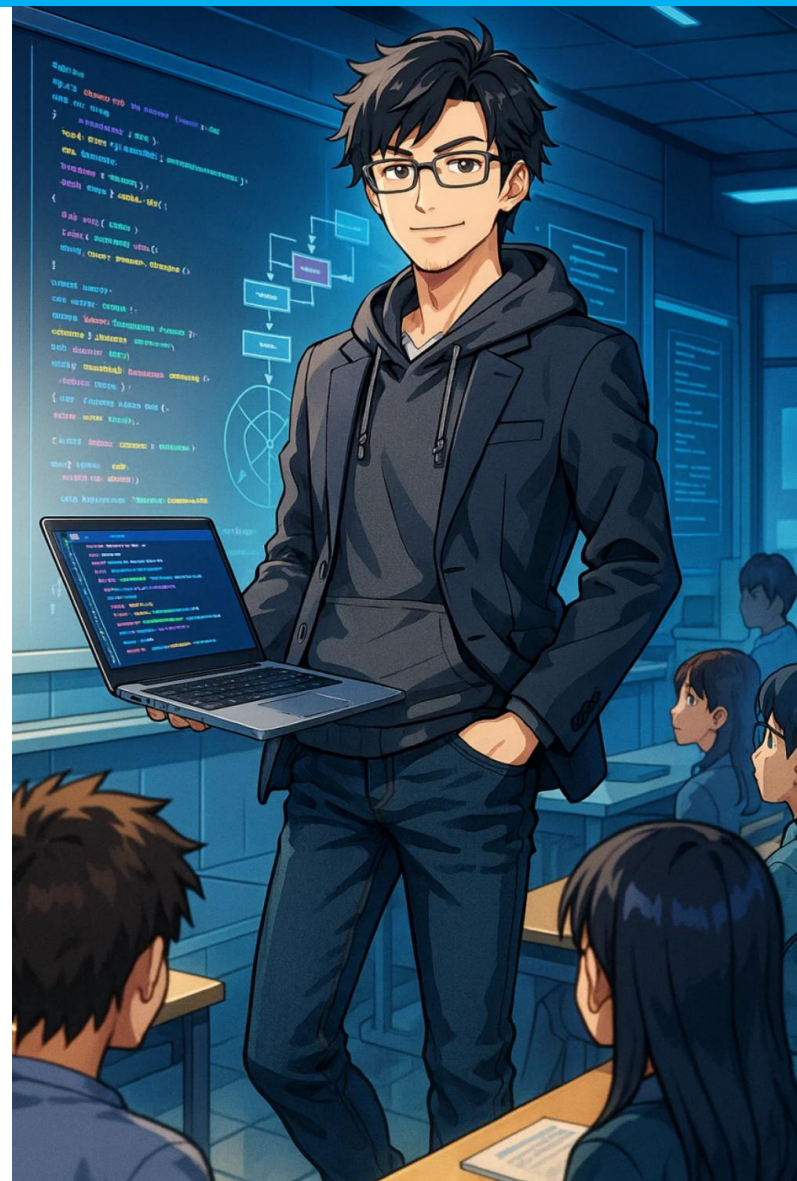
Los Patrones de diseño


def quicksort(arr):
pivot = arr;
pivot = crr;carr);
tr for i in
range(n); 1;
return n greater);
}
for i in range(n):
def initialiizze(a);
arr b = arro{
pivot = pross(a. b
pivot = a t b)
if (nr < i a b:) (
break = a:
return (b. blotk
}
fetурn grecter --;
START
INPUT
Yes
No
AB

Definición

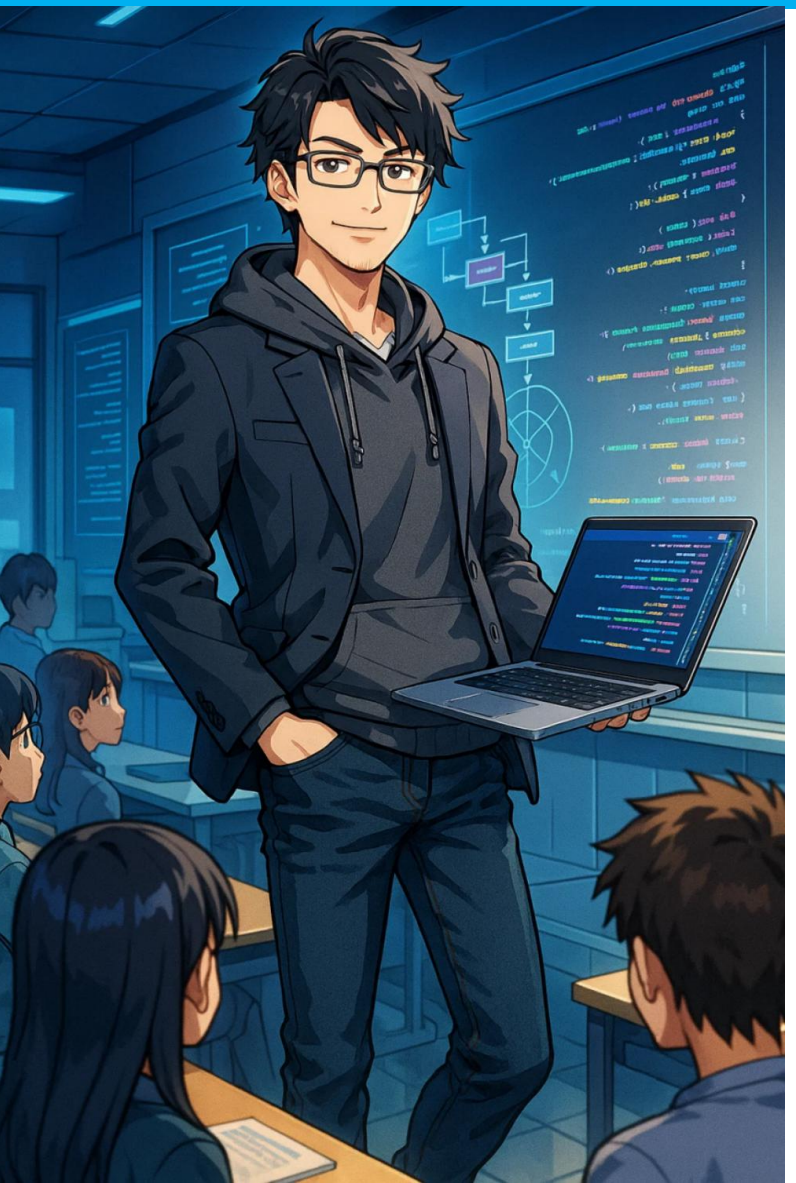
Definición

- Los patrones de diseño son soluciones habituales a problemas que ocurren con frecuencia en el diseño de software.
- Son como planos prefabricados que se pueden personalizar para resolver un problema de diseño recurrente en tu código.
- El patrón no es una porción específica de código, sino un concepto general para resolver un problema particular.
- A menudo los patrones se confunden con algoritmos porque ambos conceptos describen soluciones típicas a problemas conocidos.
- Un patrón de diseño es una forma recurrente y comprobada de resolver un problema de diseño común en sistemas orientados a objetos. Es una idea de solución, no una receta rígida.



Los principios de SOLID

Definición

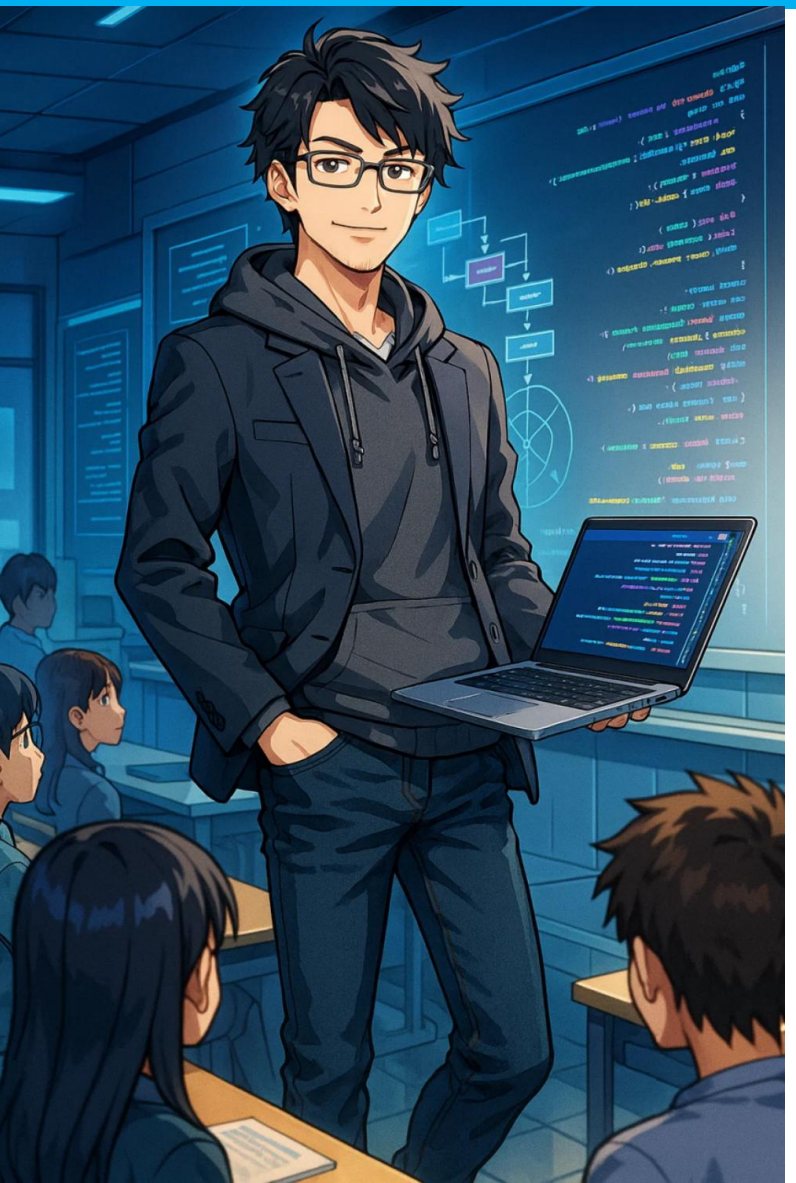


SOLID se compone de una serie de principios y de buenas prácticas que se deberían tener como base antes de proponer una arquitectura de software para el desarrollo de nuestras aplicaciones.

SOLID es un conjunto de 5 principios básicos del diseño orientado a objetos que ayudan a construir software:

- Más fácil de entender.
- Más fácil de mantener.
- Más fácil de escalar.
- Más resistente a errores cuando cambias cosas.

Definición

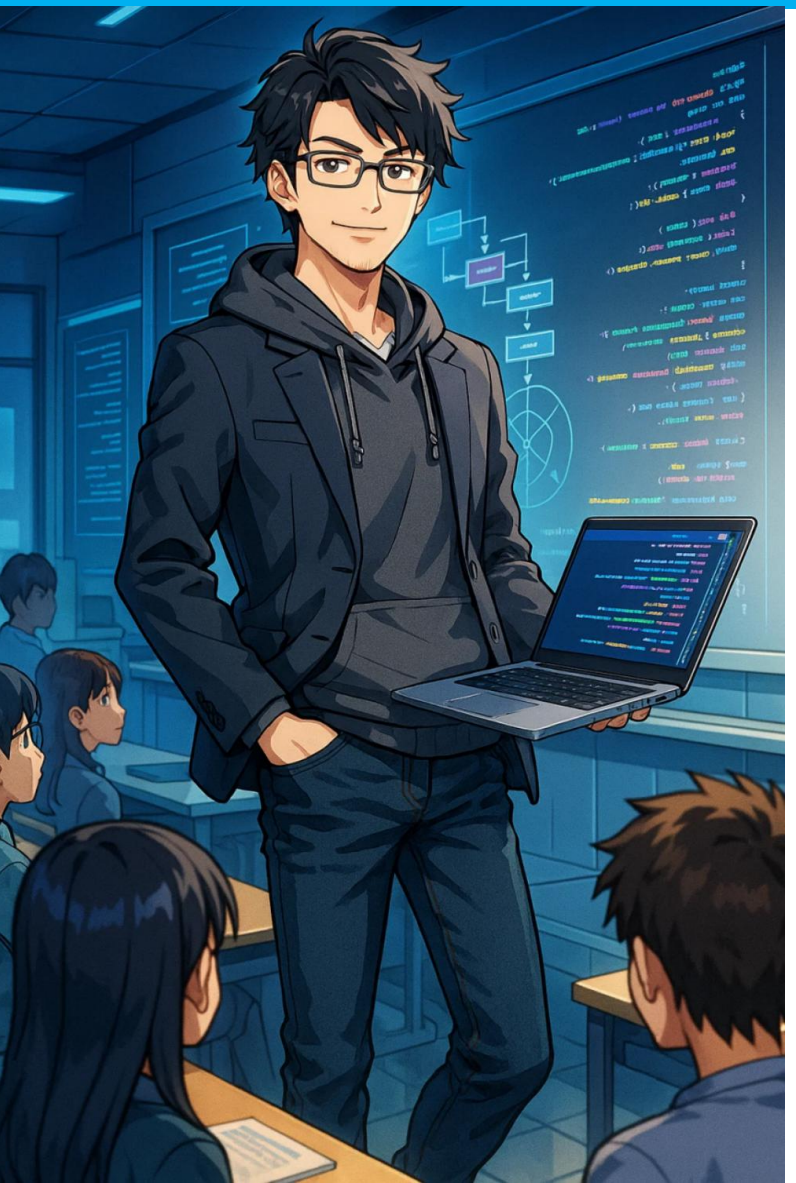


SOLID es un acrónimo, y cada una de las letras que lo componen tiene un significado:

- *S (Principio de Responsabilidad Única | Single Responsibility Principle)*
  - Una clase debe tener una sola razón para cambiar.
  - Cada clase debe hacer solo una cosa. Si haces que una clase haga muchas cosas, se vuelve difícil de mantener.

- *O (Abierto/Cerrado | Open/Closed Principle)*
  - El código debe estar abierto a la extensión, pero cerrado a la modificación.
  - No deberías tener que cambiar código existente para agregar nuevas funciones.

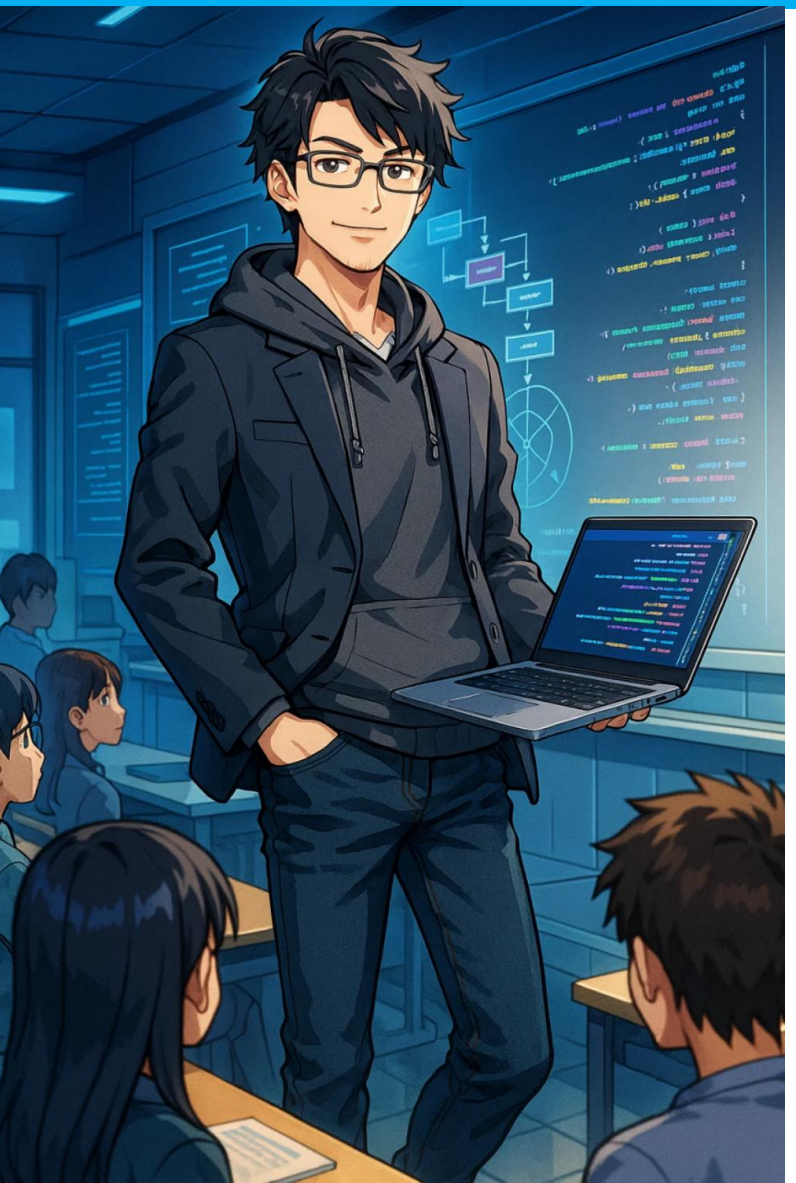
Definición



- *L (Principio de Sustitución de Liskov | Liskov Substitution Principle)*
  - Una subclase debe poder usarse en lugar de su superclase sin romper el código.
  - Las clases hijas deben respetar el comportamiento esperado por la clase padre.

- *I (Principio de Segregación de Interfaces | Interface Segregation Principle*
  - Una clase no debería verse obligada a implementar métodos que no necesita.
  - Las interfaces deben ser pequeñas y específicas. Es mejor tener varias interfaces chicas que una gigante.

Definición



- *D (Principio de Inversión de la Tendencia | Dependency Inversion Principle)*
  - Las clases deben depender de abstracciones, no de clases concretas.
  - Si una clase depende directamente de otra, el sistema se vuelve frágil. Es mejor trabajar con interfaces o abstracciones.

En resumen podemos decir que:

- SOLID te ayuda a escribir mejor código: más limpio, claro y flexible.
- No se trata de memorizar, sino de pensar bien cómo separar responsabilidades y cómo hacer que tu código soporte el cambio sin romperse.
- Al principio parece mucho, pero con la práctica se vuelve natural.

Los patrones de diseño clasicos

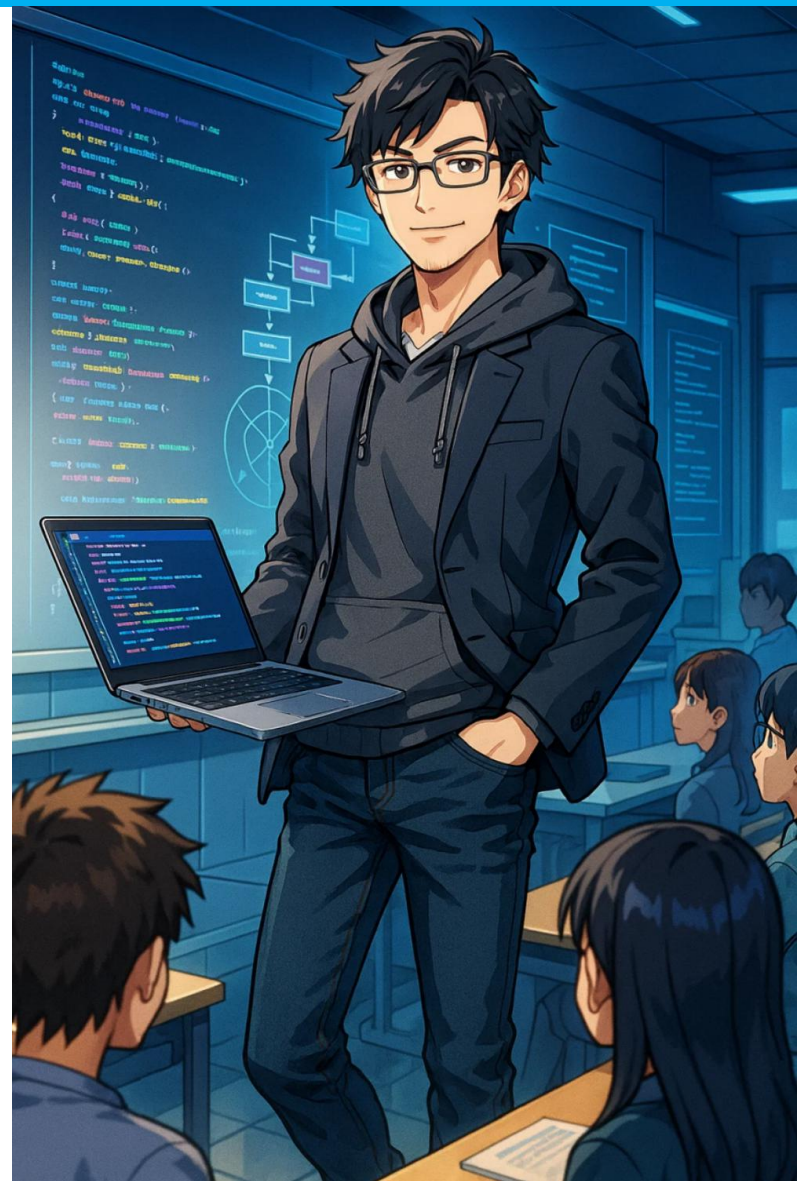
Los patrones de diseño clasicos

Son formas recurrentes de organizar clases y objetos que los programadores han identificado por experiencia.

Su objetivo es resolver problemas típicos de diseño en aplicaciones de software orientadas a objetos.

Algunos patrones importantes desde una visión clásica son:

- Controlador.
- Entidad.
- Fachada.
- Delegado.
- Plantilla.
- Modelo-Vista-Controlador (MVC).
- Observador.
- Pool de Objetos.
- Proxy.
- Registro.



Los patrones de diseño clasicos

Controlador

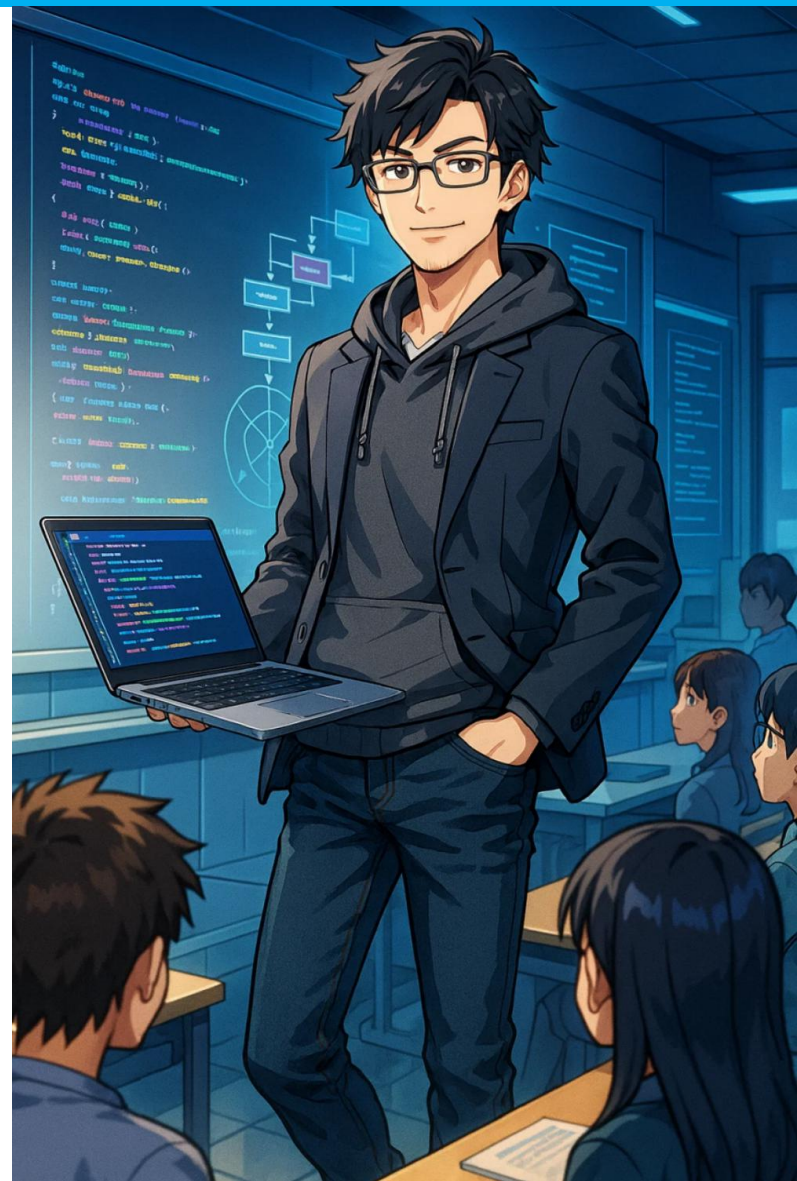
- Centraliza la lógica para manejar eventos, acciones del usuario o flujo de procesos.
- Útil para separar la lógica de presentación de la lógica de negocio.

Entidad (o Modelo)

- Representa un objeto del mundo real con identidad y datos *persistentes*.
- Muy común en sistemas de gestión (usuarios, productos, clientes).

Fachada (Facade)

- Ofrece una interfaz simple a un sistema o conjunto de clases más complejo.
- Útil para ocultar la complejidad de varios subsistemas.



Los patrones de diseño clasicos

Delegación

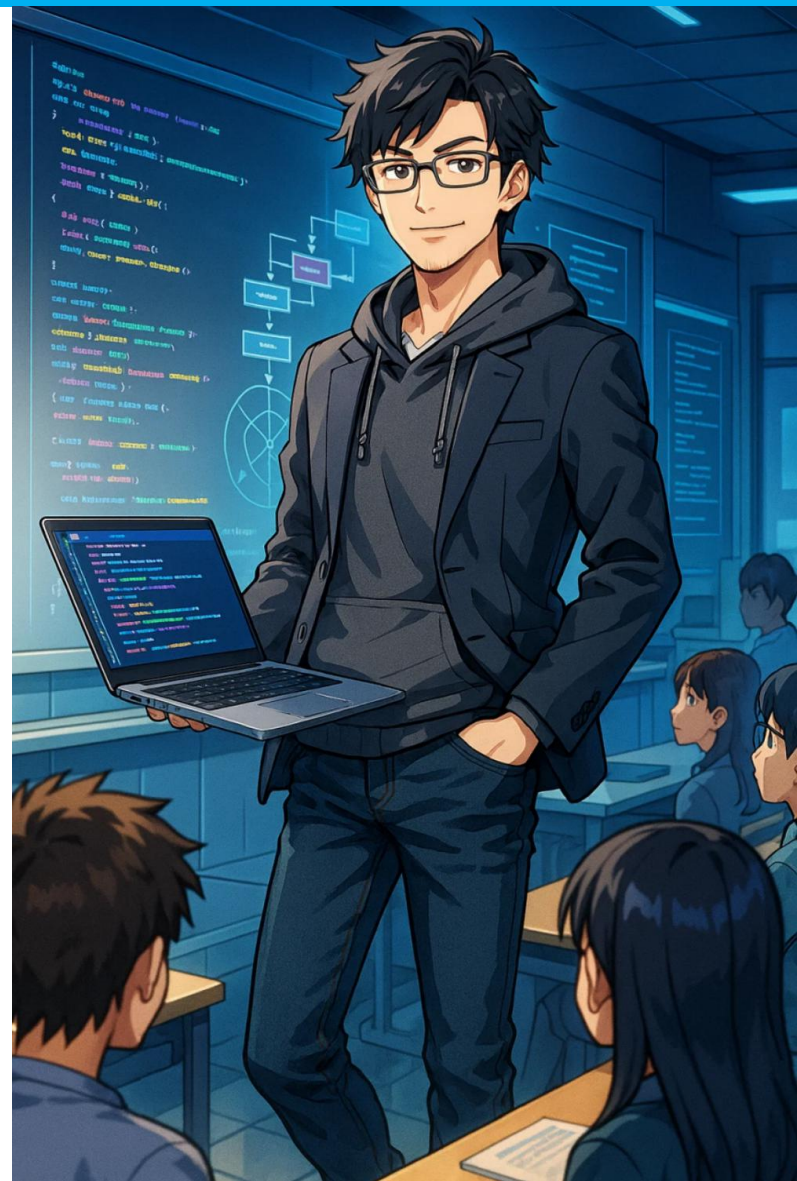
- Un objeto delegado realiza parte del trabajo por otro.
- Se usa cuando una clase asigna responsabilidad a otra clase especializada.

Observador (Observer)

- Permite que varios objetos "se suscriban" a otro para recibir actualizaciones.
- Ideal para notificaciones, actualizaciones en tiempo real.

Repository

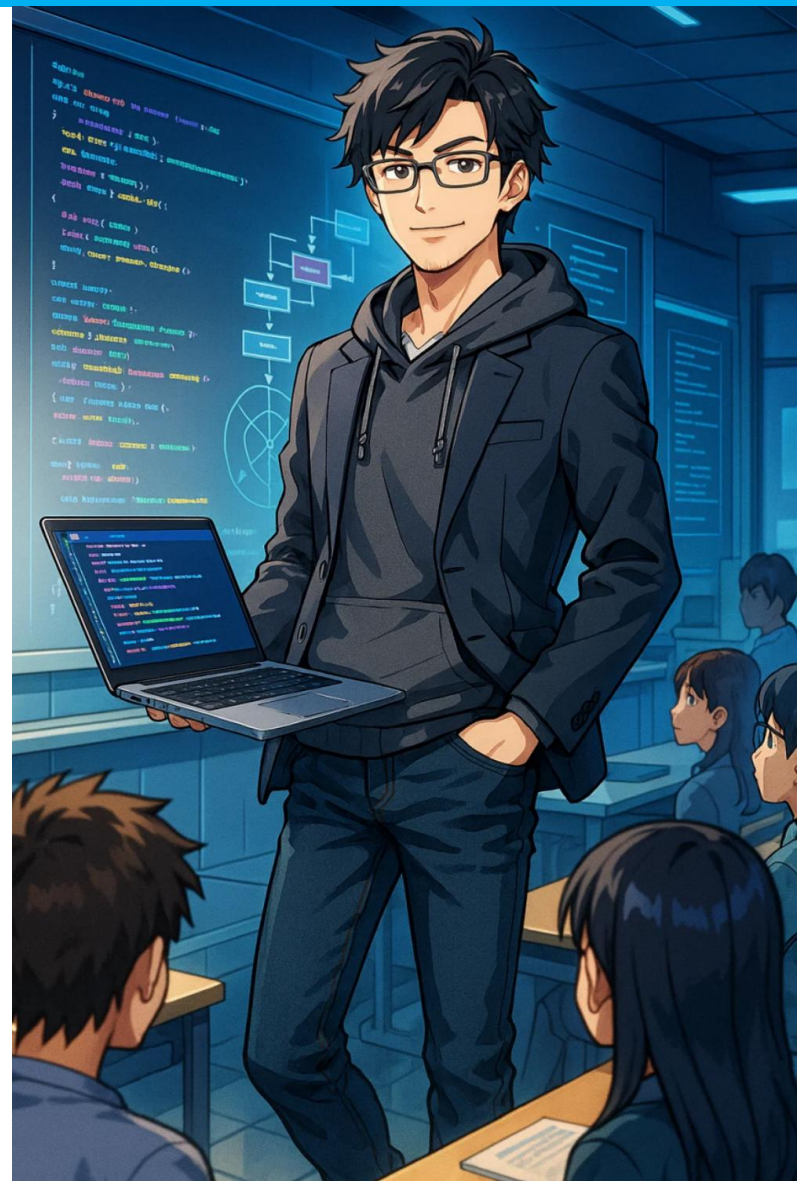
- Separar la lógica del programa de la lógica para guardar/leer datos.
- Hacer el código más limpio, reutilizable y fácil de probar.
- Facilitar el cambio de base de datos sin tocar toda la lógica del sistema.



Los patrones de diseño clasicos

Singleton

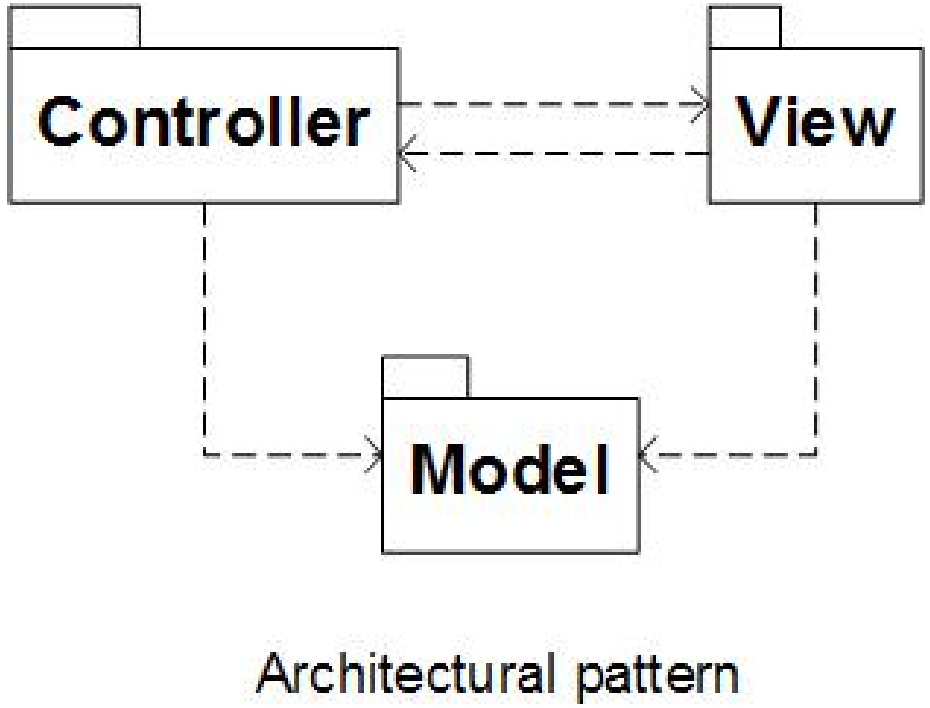
- Es un patrón de diseño que asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella.



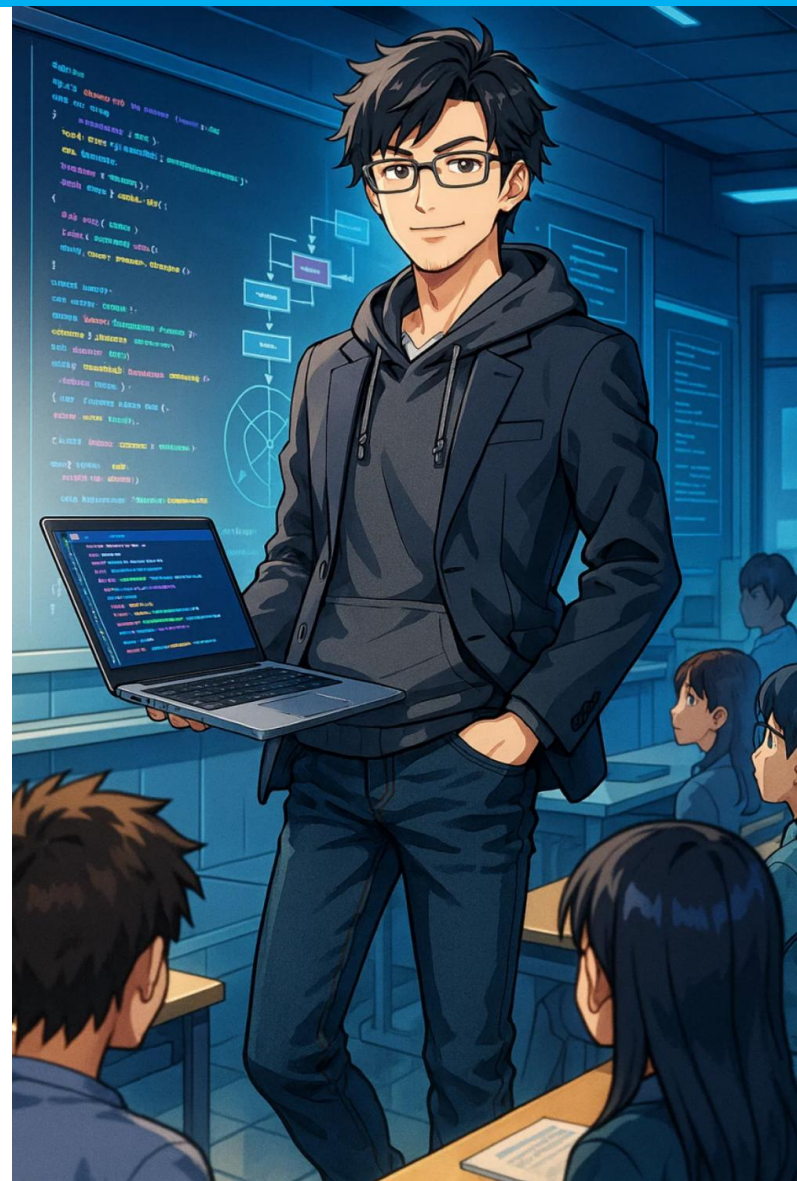
Los patrones de diseño clasicos

Modelo-Vista-Controlador (MVC)

- Es un patrón de diseño muy usado en aplicaciones con interfaz de usuario, como sitios web, apps móviles o sistemas de escritorio.
- La idea es separar responsabilidades: cada parte del sistema hace solo su trabajo, evitando que una sola clase lo haga todo.

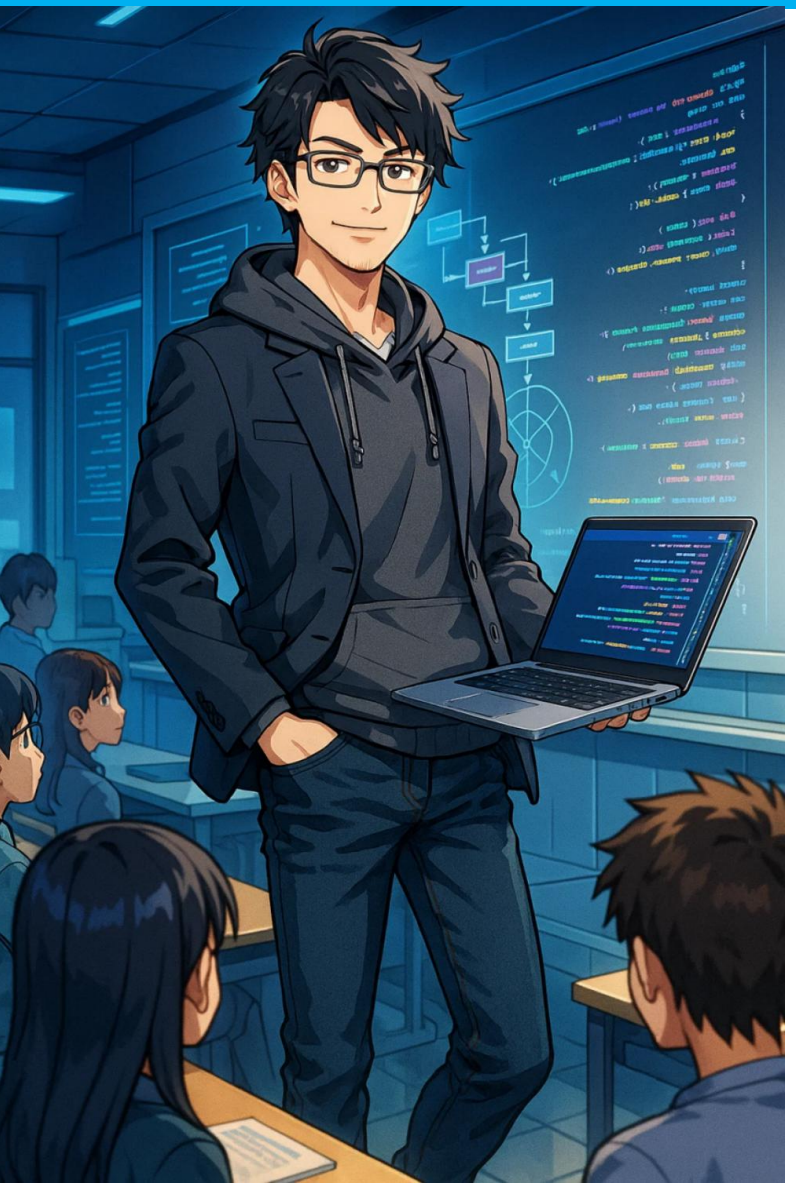


Architectural pattern

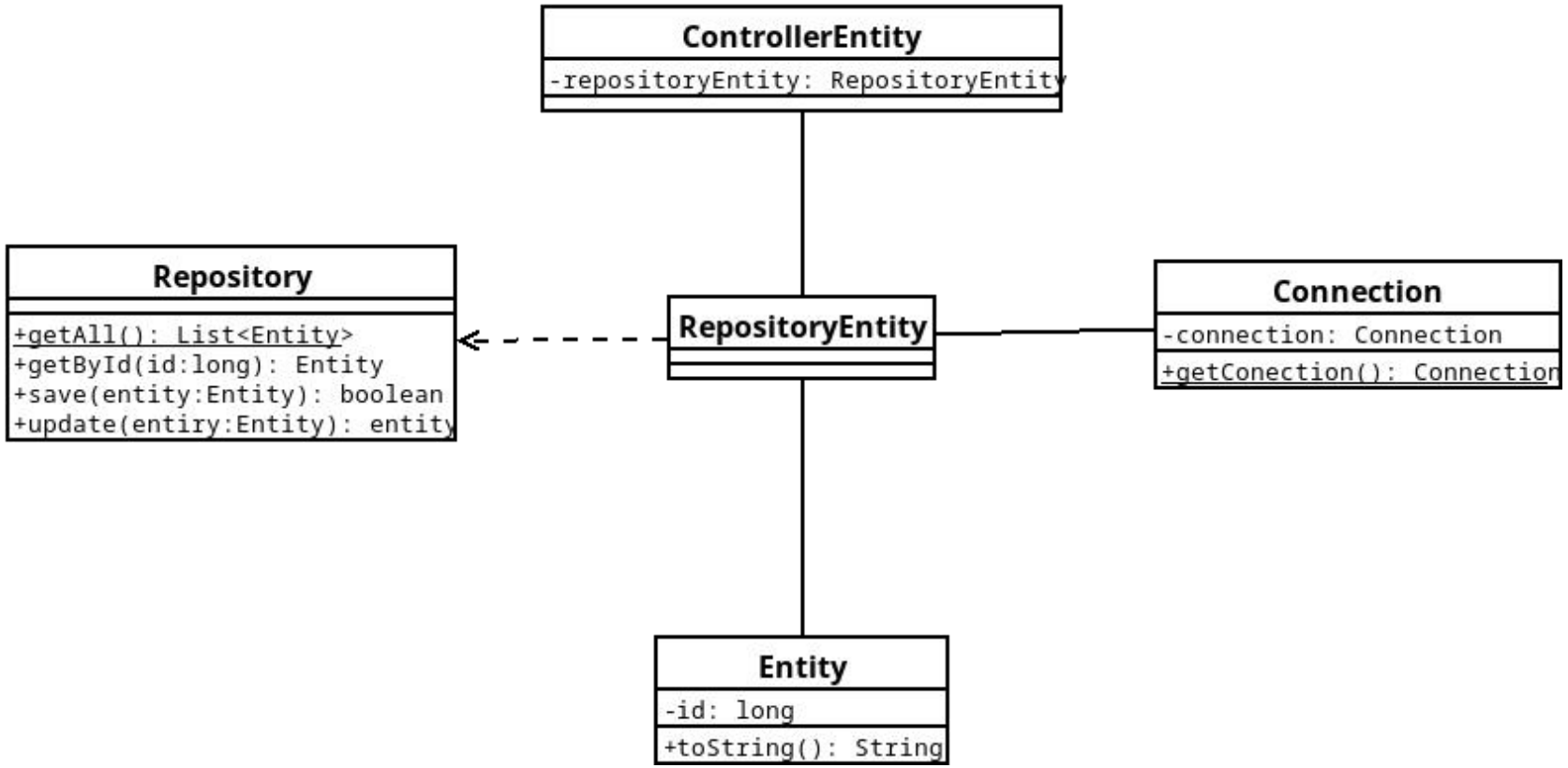


Arquitecturas limpias

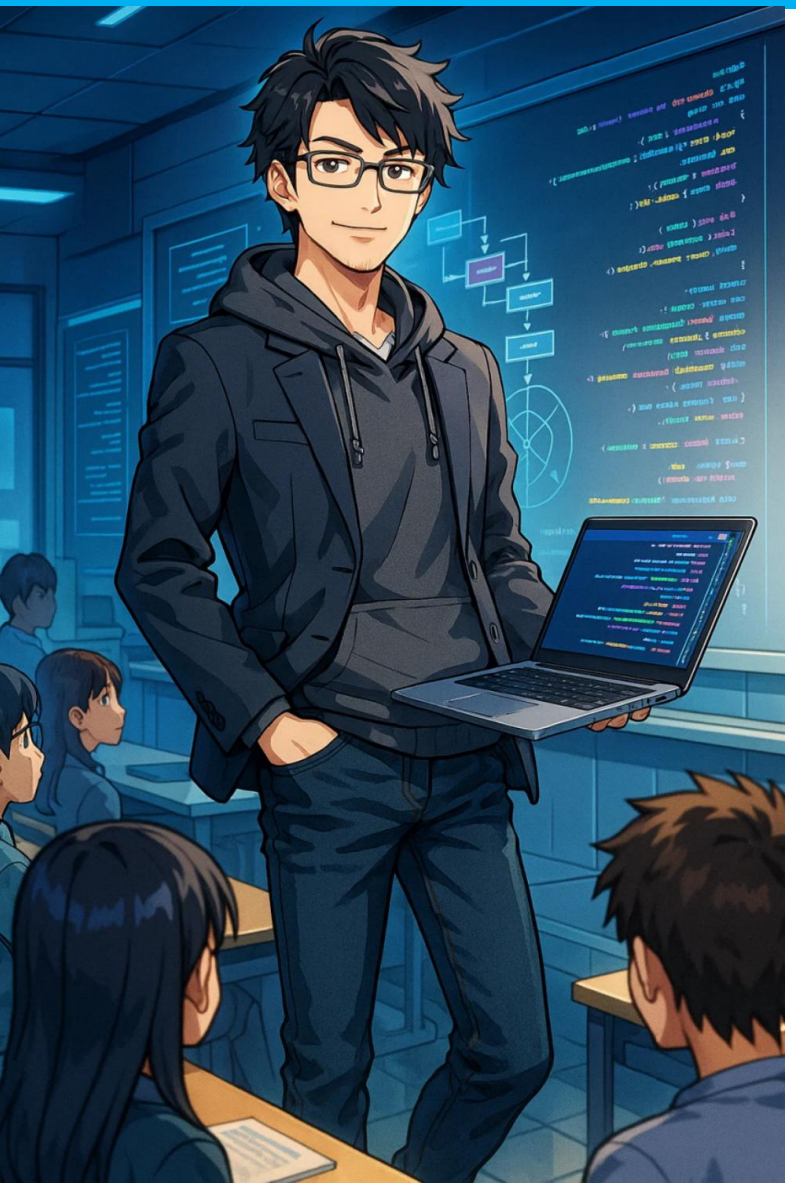
Arquitecturas limpias



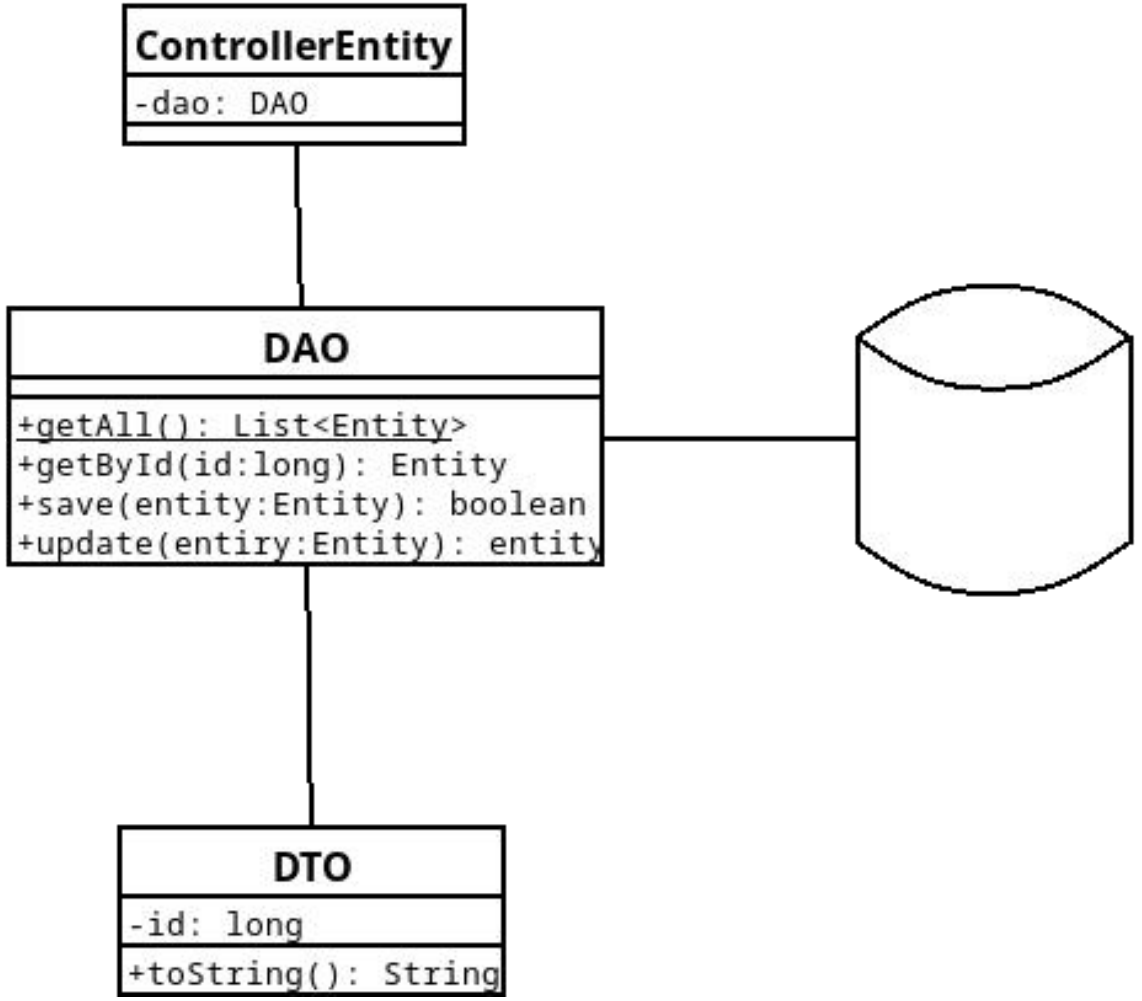
Entity - Repository



Arquitecturas limpias



DAO - DTO - Facade



Los Patrones de diseño

U1. Modelado

Programación Orientada a Objetos

Ingenieria en Tecnologias de la Información e Innovación Digital

Universidad Politecnica del Estado de Morelos