Diagrama

El contenido generado por IA puede ser incorrecto.

# Explicación del proyecto de Gestión de libros.

Estructura General del Proyecto:

**MVC** divide el proyecto en tres capas principales:

**Modelo (/Modelo):** Contiene la lógica del negocio y los datos, incluyendo:

* Libro.java y Categoria.java: Representan las entidades básicas.
* PersistenciaJSON.java: Se encarga de guardar y cargar datos en formato JSON.
* Observable.java y Observer.java: Implementan un sistema de notificación de cambios.
* EntityFactory.java: Crea objetos Libro y Categoria dinámicamente.

**Vista (/Vista):** Interfaz gráfica que interactúa con el usuario:

* VentanaPrincipal.java: Ventana principal de la aplicación.
* DialogoLibro.java y DialogoCategoria.java: Ventanas emergentes para añadir o editar libros y categorías.

**Controlador (/Controlador):** Gestiona la comunicación entre la Vista y el Modelo:

* Controlador.java: Coordina los eventos principales.
* Categorias.java: Maneja las acciones específicas relacionadas con categorías.

**Otros recursos**

* libros.json y categorias.json: Archivos de almacenamiento persistente.
* . settings,. classpath, etc.: Archivos de configuración del entorno (Eclipse).

Patrones de Diseño Implementados

*Se aplicaron* ***tres patrones de diseño importantes****:*

***1. Factory Method – EntityFactory.java***

*Permite crear instancias de Libro o Categoria* ***de forma dinámica*** *según el tipo:*

* *Útil para encapsular la creación de objetos.*
* *Hace el código más escalable y flexible.*

***2. Strategy – PersistenciaStrategy y PersistenciaJSON***

*Define un* ***comportamiento de persistencia*** *intercambiable:*

* *Puedes guardar datos en distintos formatos (por ahora JSON, pero podría ser XML, DB, etc.).*
* *Aplica el principio de* ***abierto/cerrado****: puedes agregar nuevas estrategias sin modificar las clases existentes.*

***3. Observer – Observable.java y Observer.java***

*Permite notificar automáticamente a la vista cuando cambian los datos del modelo.*

* *Mejora la sincronización entre capas.*
* *Permite que la interfaz gráfica se actualice cuando hay cambios sin acoplarla fuertemente al modelo.*

**3. Definición dinámica de atributos en la arquitectura MVC**

El uso de EntityFactory permite crear instancias sin depender del tipo exacto. Esto habilita una especie de "definición dinámica" de atributos, ya que no es necesario saber de antemano si se trabajará con Libro o Categoria.

* En la **Vista**, los diálogos pueden trabajar con instancias genéricas.
* El **Controlador** puede enviar entidades genéricas al Modelo.
* Esta abstracción ayuda a reducir el acoplamiento entre clases.

**4. Consideraciones de diseño y limitaciones**

**Aspectos positivos**

* Buena separación de responsabilidades (MVC bien aplicado).
* El uso de Strategy, Factory, y Observer mejora la extensibilidad del código.
* Persistencia en JSON permite trabajar sin base de datos.

**Limitaciones**

* La definición de atributos dinámicos no es completamente flexible (no se usan reflejos ni mapas genéricos).
* Las clases están fuertemente acopladas al formato JSON, lo que dificulta cambiar la persistencia si no se modifica PersistenciaJSON.
* Falta de validaciones más robustas al agregar entidades.
* No hay un sistema de logs o manejo de errores consistente (excepciones a veces impresas directamente).
* Solo se considera un tipo de usuario; no hay control de permisos o múltiples roles.