**💡 Aplicación y Consideración de Patrones de Diseño**

En el desarrollo del sistema, se han aplicado y considerado distintos patrones de diseño orientados a mejorar la organización, escalabilidad y mantenibilidad del código.

**🔹 Patrón DAO (Data Access Object)**

Actualmente el sistema implementa el **patrón DAO**, encargado de encapsular la lógica de acceso a datos y separar esta capa del resto de la aplicación.  
Cada entidad del sistema (por ejemplo, Propiedad, Contrato, Pago, Persona, etc.) posee su propio DAO, responsable de ejecutar las operaciones sobre la base de datos mediante consultas SQL a través de la librería **Sql2o**.

Este patrón permite desacoplar la lógica de persistencia del resto de la aplicación, facilitando futuras modificaciones en la base de datos sin afectar las capas superiores.

**Ejemplo de aplicación:**

* ContratoDAO → se encarga de registrar, modificar y consultar contratos.
* PagoDAO → permite registrar pagos y generar reportes de historial según filtros de búsqueda (por período, inquilino o propietario).

**🔹 Patrón de Inyección de Dependencias**

A través del uso de la anotación @Autowired de **Spring**, se aplicó el patrón de **Inversión de Control / Inyección de Dependencias**, lo cual permite inyectar las instancias de los DAO dentro de los servicios correspondientes.  
De esta forma, las clases no crean directamente sus dependencias, lo que facilita el mantenimiento, las pruebas unitarias y el reemplazo de componentes en el futuro.

**Ejemplo:**  
El PropiedadService utiliza las dependencias PropiedadDAO, PropietarioDAO, ComercialDAO, y FamiliarDAO inyectadas automáticamente por el framework.

**🔹 Posible uso futuro del patrón *Factory Method***

En el caso de que el sistema evolucione para incorporar distintos tipos de **pagos** o **contratos**, podría aplicarse el patrón **Factory Method**.  
Este permitiría crear los objetos de pago o contrato de forma flexible, según el tipo requerido, sin modificar la lógica existente del sistema.

Por ejemplo, se podría implementar una fábrica que genere diferentes tipos de pago (PagoEfectivo, PagoTransferencia, PagoTarjeta) o contratos con condiciones específicas (ContratoResidencial, ContratoComercial).

*Actualmente no se considera necesario implementar este patrón, pero la arquitectura permite su incorporación futura si el sistema se amplía.*

**🔹 Posible uso futuro del patrón *Façade***

Los servicios del sistema funcionan de manera similar a una **fachada**, ya que encapsulan la lógica de negocio y coordinan las operaciones entre varios DAOs.  
En versiones futuras, si el sistema incorpora procesos más complejos (por ejemplo, registrar contrato + generar pago inicial + enviar notificación), puede formalizarse la aplicación del patrón **Façade** para ofrecer una única interfaz de alto nivel hacia el módulo de negocio.

*Esto simplificaría la comunicación con la capa de presentación y reduciría el acoplamiento entre los controladores y la lógica interna.*

**🔹 Conclusión**

En la versión actual, el sistema aplica efectivamente los patrones **DAO** e **Inyección de Dependencias**, lo que asegura una estructura modular, mantenible y extensible.  
Si el desarrollo continúa creciendo, será factible incorporar patrones como **Factory Method** y **Façade** para mejorar la flexibilidad y escalabilidad del sistema.