## Liskov

**1. ¿Cumple la clase Configuracion el Principio OCP? Justifica la respuesta.**

No, la clase Configuracion no cumple completamente el **Principio Abierto-Cerrado (OCP)**. Si se añade una nueva clase que implemente RecursoPersistente, como una nueva configuración, el código dentro de la clase Configuracion necesitaría ser modificado en el método cargarConfiguracion() para agregar esa nueva configuración. Esto viola el principio, que dice que una clase debe estar abierta para la extensión (añadir nuevas configuraciones) pero cerrada para la modificación.

Podemos mejorar esto eliminando la responsabilidad de agregar nuevas configuraciones directamente en la clase Configuracion y moviéndola fuera para que no se modifique la clase cada vez que se quiera agregar una nueva configuración.

**2. ¿Cumple la clase Configuracion el Principio de Liskov (LSP)? Justifica la respuesta.**

No, la clase Configuracion no cumple con el **Principio de Sustitución de Liskov (LSP)**. El principio de Liskov dice que las subclases o implementaciones deben poder sustituir a su clase base sin afectar el comportamiento esperado. En este caso, la clase ConfiguracionHoraria no cumple con el contrato de la interfaz RecursoPersistente porque su método save() lanza un error al intentar guardar, ya que es de solo lectura.

La interfaz establece que cualquier implementación debería ser capaz de guardar los datos, pero ConfiguracionHoraria viola esta expectativa. Esto rompe el comportamiento esperado en la clase Configuracion, ya que al intentar guardar todas las configuraciones, se encuentra con una excepción no esperada.

**3. Explicación general del problema y la solución**

El problema original es que la implementación de la clase ConfiguracionHoraria rompe el contrato de la interfaz RecursoPersistente porque no puede cumplir con todas las funcionalidades esperadas, en particular, la capacidad de guardar. Esto viola el Principio de Sustitución de Liskov, que establece que las clases derivadas o implementaciones deben poder sustituir a su clase base sin alterar el comportamiento correcto del sistema.

La solución consiste en usar **herencia por interfaz**, separando las responsabilidades de carga y guardado en diferentes interfaces. De esta manera, ConfiguracionHoraria solo implementa la funcionalidad que puede soportar (lectura), y otras clases que pueden hacer ambas (cargar y guardar) implementan una interfaz más rica.

Esto asegura que el código siga siendo consistente y que el comportamiento esperado se mantenga, mejorando la flexibilidad y el mantenimiento del sistema.