Buenas prácticas

Procedimientos recomendados para la creación y mantenimiento de clases

- Construir objetos consistentes con el Constructor.
- Mantener integridad con *getters*, *setters* y encapsulamiento.
- Imaginar que la Clase la usa otra persona.
- Pruebas Unitarias.
- Gestionar errores para asegurar la integridad de los objetos.
- Aplicar el Principio de responsabilidad única (SRP).

Construir objetos consistentes con el Constructor

Utilizar el constructor para asegurar que los objetos se **inicializan correctamente desde el principio**.

Prácticas recomendadas para lograr esto:

- No permitir la creación de objetos en un **estado inconsistente o incompleto**.
- Utilizar **excepciones**.
- Ofrecer **múltiples constructores** para facilitar la creación de objetos con diferentes estados iniciales.

¡Advertencia!

En la práctica, algunas librerías y *frameworks* pueden no cumplir estrictamente con esta buena práctica, lo que puede llevar a la creación de objetos con estados no válidos o incompletos.

Mantener integridad con *getters*, *setters* y encapsulamiento

Una vez que se crea un objeto consistente, es crucial garantizar que mantenga su **coherencia durante todo su ciclo de vida** mediante el uso adecuado de:

- *Getters.*
- *Setters.*
- Validaciones.

Aplicar correctamente el **principio de encapsulamiento** para ocultar la implementación interna y los atributos detrás de *getters* y *setters* para validar y **facilitar futuros cambios** sin afectar otras partes del código.

Imaginar que la Clase la usa otra persona

Imaginar que otros desarrolladores utilizarán el código en el que se está trabajando, fomenta prácticas de programación más limpias y comprensibles y conduce a un **código mantenible y menos propenso a errores**.

Es importante:

- **Documentar y nombrar** claramente las clases y métodos para que sean comprensibles y fáciles de usar por otros.
- Considerar el **diseño de interfaces** claras y simples.

Esta buena práctica ayuda muchísimo a concentrarse, pensar mejor la interfaz de las clases.

Pruebas Unitarias

Escribir pruebas unitarias para verificar el **comportamiento esperado** de las clases y métodos, incluyendo las validaciones realizadas en los *setters*.

Las pruebas unitarias no solo aseguran que el código funcione correctamente, también **facilitan futuras modificaciones y refactorizaciones** al proporcionar una red de seguridad contra cambios inadvertidos.

Las **pruebas unitarias** son además una excelente herramienta para enfrentarnos con nuestro código, entender su funcionamiento y, llegado el caso, mejorarlo.

Gestionar errores para asegurar la integridad de los objetos

Emplear excepciones para **manejar errores** y asegurar la consistencia del estado de los objetos.

- Definir excepciones para **situaciones específicas** que puedan causar un estado inconsistente en los objetos.
- Lanzar excepciones **donde el estado del objeto podría ser comprometido**, como en *setters* y métodos críticos.
- **Capturar y manejar excepciones** de manera adecuada para garantizar que el sistema siga funcionando.

El uso adecuado de **excepciones** no solo **mejora la legibilidad y el mantenimiento** del código, sino que también **fortalece la robustez y la fiabilidad** de tus aplicaciones al asegurar que los objetos mantengan su consistencia interna frente a condiciones excepcionales.

Aplicar el Principio de responsabilidad única (SRP)

Asegurarse de que cada clase y método tenga **una única responsabilidad** clara y bien definida.

- Identificar y separar las **responsabilidades específicas** de cada clase y método.
- Evitar que una **clase o método asuma múltiples tareas no relacionadas**. Esto facilita la comprensión y la modificación del código.
- Promover la **modularidad, la reutilización** del código y reducir la complejidad general del sistema.

Utilizar el **SRP** como guía al diseñar nuevas clases y al refactorizar código existente para garantizar una **estructura clara y mantenible.**

Buenas prácticas en generación de clases con IA

¿Por qué son importantes las buenas prácticas de POO?	¿Qué se debe hacer?	¿Cómo integrarlas en los *prompts*?
Aseguran la consistencia, mantenibilidad y fiabilidad del código. **Riesgo con IA**: La inteligencia artificial, a veces, **no aplica** automáticamente estas prácticas.	**Estar atentos:** Supervisar y corregir cuando la IA no sigue principios como el **constructor, *getters/setters*, y manejo de excepciones**. **Solicitar cumplimiento:** Es crucial pedir explícitamente a la IA que respete estas prácticas.	Agregar directrices sobre buenas prácticas en los prompts de generación de clases obtiene resultados mejorados. Esto asegura **respuestas más coherentes y alineadas** con estándares de desarrollo.