**ADS**

**Assignment:** Lecture 1

**By:** Cesar Mamani

Diseñar un software OO es complicado, pero más aún hacer uno reutilizable, por ejemplo encontrar un objeto en específico que tenga un nivel de detalle correcto, luego estructurar la interfaz, las relaciones, etc. Sin que pierda la generalización para futuros problemas o extensiones. La diferencia entre los diseñadores experimentados de OO y los nuevos nuevos, es la reutilización, encontrar patrones que son recurrentes en las clases, con la finalidad de que luego se pueda agregar funciones y quitarlas fácilmente.

Como en la anterior lectura el libro comenta la importancia de que un patrón de diseño cuente con un nombre, explicación y evaluación. Las cuales facilitarán la reutilización de código y arquitectura.

En conclusión la primera parte explica cómo los patrones de diseño son útiles para los diseñadores de OO.

Patrón de diseño cuenta con 4 elementos esenciales, **Nombre** el cual será el identificador del patrón, El **problema** que explica el contexto y cuando se debe aplicar, La **solución** que es como una plantilla que da una descripción reducida, además que describe relaciones, colaboraciones y las responsabilidades que tiene, y por último las **consecuencias** de usar el patrón, como ventajas y desventajas, como también el impacto, portabilidad y extensibilidad del patrón.

El libro también nos habla del patrón de diseño MVC que consta del Modelo que son los objetos, Vista que es la interfaz gráfica y el controlador que define como la interfaz gráfica reacciona a eventos del usuario. Por ejemplo si nosotros damos click a un botón para hacer un logout este llama al controlador que eliminará la data del modelo y notifica los cambios del modelo a la vista. Este controlador hace uso de los patrones de diseño observable, compuesto, y estrategia.

El capítulo también nos da un resumen de los 23 patrones de diseño más comunes. Cada patrón varía en su nivel de detalle y abstracción y como se tienen muchos, estos se identifican por su propósito y alcance.

Por su propósito tenemos estructural que explican cómo ensamblar objetos de clases en estructuras más grandes, manteniendo la flexibilidad y eficiencia, mientras que la creacional proporciona mecanismos para crear objetos que incrementen flexibilidad y ayudan a reusar código, conductual que se encargan de comunicar estados entre objetos.

Por su scope tenemos los patrones de clase que explican u ocupan de las relaciones entre padre e hijo, y los de objetos que son por la relación entre objetos.

De la intersección del propósito y scope se tiene:

Patrones de clase creacional, Patrones de objeto creaciones, Patrones de clase estructural usan herencia, Patrones de objeto estructura ensamblan objetos, Patrones de clase de comportamiento usan herencia para describir flujos o algoritmos, Patrones de objeto de comportamiento explica cómo trabajan juntos un conjunto de objetos para un objetivo.

Determinar la granularidad de un objeto por tamaño, por ejemplo caminos para descomponer objetos en objetos más pequeños tenemos a Facade, Flyweight, Abstract Factory, Builder, Visitor, Command.

Por la especificación de la interfaz de un objeto. describe que es un conjunto de todas las firmas de las operaciones de un objeto, llamadas también como una interfaz.

Las interfaces son fundamentales en la programación orientada a objetos, además un tipo es un nombre usado para denotar una interfaz. El capítulo también explica que puedes tener 2 objetos con las mismas interfaces pero con diferente implementación.

Además en el capítulo vemos sobre los enlaces dinámicos los cuales son como un late binding, donde la implementación es descubierta en tiempo de ejecución, por lo que puedes escribir programas que esperan una interfaz y en tiempo de ejecución pueden llegar hijos o subTypes a la interfaz con diferentes implementaciones, Lo cual también lleva al Polimorfismo.

—------------

Diseñar un software orientado a objetos (OO) puede ser complicado, especialmente cuando se busca hacerlo reutilizable. Encontrar el nivel adecuado de detalle en los objetos y estructurar la interfaz y las relaciones sin perder la generalización para futuros problemas o extensiones es un desafío. Los diseñadores experimentados de OO se distinguen por su habilidad para encontrar patrones recurrentes en las clases, lo que facilita la adición y eliminación de funciones de manera sencilla.

En este sentido, los patrones de diseño son herramientas útiles para los diseñadores de OO. Un patrón de diseño consta de cuatro elementos esenciales: el nombre, que es un identificador para el patrón; el problema, que describe el contexto en el que se aplica el patrón; la solución, que proporciona una plantilla con una descripción reducida, incluyendo relaciones, colaboraciones y responsabilidades; y las consecuencias, que incluyen ventajas, desventajas, impacto, portabilidad y extensibilidad del patrón.

El libro también menciona el patrón de diseño MVC (Modelo-Vista-Controlador), que consta de tres componentes: el Modelo, que representa los objetos; la Vista, que es la interfaz gráfica; y el Controlador, que define cómo la interfaz gráfica reacciona a los eventos del usuario. El patrón MVC hace uso de otros patrones de diseño, como Observable, Compuesto y Estrategia.

El capítulo ofrece un resumen de los 23 patrones de diseño más comunes, cada uno con diferentes niveles de detalle y abstracción. Estos patrones se identifican por su propósito y alcance. En cuanto al propósito, existen patrones estructurales, que se enfocan en ensamblar objetos de clases en estructuras más grandes manteniendo la flexibilidad y eficiencia; patrones creacionales, que proporcionan mecanismos para crear objetos y aumentar la flexibilidad y reutilización de código; y patrones conductuales, que se encargan de la comunicación de estados entre objetos.

En cuanto al alcance, hay patrones de clase, que se ocupan de las relaciones entre clases padre e hijo; y patrones de objeto, que se centran en la relación entre objetos individuales.

La granularidad de un objeto se puede determinar considerando su tamaño. Para descomponer objetos en unidades más pequeñas, se pueden utilizar patrones como Facade, Flyweight, Abstract Factory, Builder, Visitor y Command.

El libro también resalta la importancia de la especificación de interfaces en la programación orientada a objetos. Una interfaz define un conjunto de firmas de operaciones que un objeto debe implementar. Múltiples objetos pueden tener la misma interfaz pero con diferentes implementaciones.

Además, el capítulo explora los enlaces dinámicos, que permiten un enlace tardío (late binding) de la implementación en tiempo de ejecución. Esto significa que un programa puede esperar una interfaz y, en tiempo de ejecución, objetos hijos o subtipos pueden cumplir esa interfaz con diferentes implementaciones, lo que conduce al polimorfismo.

Un objeto puede tener muchos tipos y objetos de diferentes clases pueden tener el mismo tipo. La herencia de clases y la herencia de interfaces difieren en que la herencia de clases extiende los comportamientos del padre y las modificaciones se basan en él, mientras que la herencia

Cuando diseñamos software orientado a objetos, es importante comprender varios conceptos clave. Uno de ellos es que un objeto puede tener múltiples tipos y que objetos de diferentes clases pueden tener el mismo tipo. Esto se debe a la capacidad de un objeto de implementar interfaces, lo que le permite ser utilizado en diferentes contextos y tratado como diferentes tipos de objetos.

Otra distinción fundamental es la diferencia entre herencia de clases y herencia de interfaces. Al heredar de una clase, se extienden los comportamientos del padre y las modificaciones se basan en la implementación del padre. Por otro lado, la herencia de interfaces describe la capacidad de un objeto de ser utilizado en lugar de otro objeto, siempre que cumpla con la misma interfaz. El uso de interfaces proporciona una visualización clara de los métodos del código, oculta la implementación y los tipos de objetos que lo utilizan.

En muchos casos, es preferible utilizar la composición en lugar de la herencia. La composición implica construir objetos complejos combinando objetos más simples. Esta relación más débil y flexible entre objetos permite una mayor modularidad, reutilización de componentes y facilita el cambio de instancias en tiempo de ejecución. En contraste, la herencia crea una relación más fuerte y estática que puede limitar la flexibilidad y la reutilización.

La delegación es un concepto estrechamente relacionado con la composición. Implica transferir la responsabilidad de realizar acciones a otros objetos. La delegación facilita el cambio de instancias en tiempo de ejecución y brinda flexibilidad en la asignación de responsabilidades. Sin embargo, la delegación puede ser más estática en términos de comprensión y puede requerir un mayor esfuerzo para seguir el flujo de control.

Además de la herencia y la composición, también existen los genéricos. Los genéricos permiten modificar el tipo que una clase utiliza en tiempo de compilación, lo que brinda mayor flexibilidad y reutilización de código. Sin embargo, al igual que con la herencia, los genéricos están sujetos a restricciones y deben ser definidos y verificados durante la etapa de compilación del software.

—----------

Uno de los puntos importantes para mi fue que un objeto puede tener muchos tipos y un objeto de diferentes clases puede tener el mismo tipo. También sobre la diferencia entre herencia de clases y de interfaz, la cual cuando heredas de una clase, extiendes comportamientos del padre y las modificaciones realizadas están basadas en el padre. Mientras que la herencia de interfaz describe un objeto que puede ser usado en lugar de otro, Entonces podemos decir que el beneficio de usar interfaces es que podemos tener una visualización clara de los métodos del código, además que escondemos la implementación y los tipos de objetos que lo usan.

Otro punto interesante es que muchas veces es mejor usar composición(En tiempo de ejecución) que herencia(Ejecuta en tiempo de compilación), ya que la herencia genera una relación muy fuerte, lo cual limita la flexibilidad y en última instancia la reutilización.

El libro también habla sobre la delegación y como ayuda fuertemente al concepto de composición, donde una de las ventajas de la delegación es facilitar el cambio de instancias en tiempo de ejecución, pero no todo es bueno ya que también tiene una desventaja la cual es el entendimiento, puesto que es más estático.

Además de la herencia, composición también existe los genéricos que permiten modificar el tipo que una clase usa, pero esto se ve en tiempo de compilación lo cual es lo mismo para la herencia.