Diseño de sistemas año 2022

Luciano Villa

Patrones

Introducción - DEFINICION:

Los Patrones de diseño son TECNICAS QUE YA HAN SIDO PROBADAS.

Los patrones de diseño representan soluciones a problemas específicos de diseño de software orientado a objetos, **que han sido desarrolladas y han ido evolucionando a lo largo del tiempo**. Estos patrones resuelven problemas concretos de diseño, y hacen que los diseños orientados a objetos sean más flexibles, elegantes y reutilizables. Los patrones ayudan a los diseñadores a reutilizar buenos diseños al basar los nuevos diseños en la experiencia previa.

Cada patrón nomina, explica y evalúa un diseño importante y recurrente en los sistemas orientados a objetos. Los patrones de diseño hacen que sea más fácil reutilizar buenos diseños y arquitecturas. Un patrón de diseño nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. Cada patrón de diseño se centra en un problema concreto, describiendo cuándo aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias y las ventajas e inconvenientes de su uso.

Los patrones de diseño nos ayudan a elegir las alternativas de diseño que hacen que un sistema sea reutilizable, y a evitar aquellas que dificultan dicha reutilización. Pueden incluso mejorar la documentación y el mantenimiento de los sistemas existentes al proporcionar una especificación explicita de las INTERACCIONES entre clases y objetos y de cuál es su intención.

Definición de otro pensador: “Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces”

Un PATRON ES MAS BIEN COMO UNA PLANTILLA QUE PUEDE APLICARSE EN MUCHAS SITUACIONES DIFERENTES. El patrón proporciona una descripción abstracta de un problema de diseño y cómo lo resuelve una disposición general de elementos (Clases y objetos).

Por otro lado, puesto que la reutilización suele ser uno de los factores de los diseños orientados a objetos, las consecuencias de un patrón incluyen su impacto sobre la FLEXIBILIDAD, EXTENSIBILIDAD Y PORTABILIDAD de un sistema.

LOS PATRONES DE DISEÑO SON DESCRIPCIONES DE CLASES Y OBJETOS RELACIONADOS QUE ESTÁN PARTICULARIZADOS PARA RESOLVER UN PROBLEMA DE DISEÑO GENERAL EN UN DETERMINADO CONTEXTO.

Los patrones reflejan todo el rediseño y la recodificación (tiene que ser mínimo) que los desarrolladores han ido haciendo a medida que luchaban por **conseguir mayor reutilización y flexibilidad en su producto software final**. Los patrones de diseño expresan las soluciones de manera breve, concisa, fácilmente aplicable.

Los patrones de diseño hacen que sea más fácil reutilizar buenos diseños y arquitecturas. Al expresar como patrones de diseño técnicas que ya han sido probadas, las estamos haciendo más accesibles para los desarrolladores de nuevos sistemas.

El uso de patrones generalmente diferencia a un buen diseño de otro que, en el mejor de los casos, resuelve un problema concreto pero que se adapta muy mal, o no lo hace en absoluto, a nuevos requisitos, funcionalidades, requerimientos o cambios en los ya existentes. Por lo que decimos que, al usar patrones de diseño, estamos favoreciendo la escalabilidad de nuestro producto software.

TIPOS DE PATRONES

Los patrones de diseño varían en su granularidad y nivel de abstracción. Clasificamos los patrones de diseño siguiendo dos criterios. El primero de ellos, denominado **Propósito**, refleja qué hace un patrón. Los patrones pueden tener un propósito de *creación*, *estructural* o *de comportamiento*. Los patrones de creación tienen que ver con el proceso de creación de objetos. Los patrones estructurales tratan con la composición de clases u objetos. Los de comportamiento caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad.

El segundo criterio, denominado **ámbito**, especifica si el patrón se aplica principalmente a clases o a objetos. Los patrones de clases se ocupan de las relaciones entre las clases y sus subclases. Estas relaciones se establecen a través de la herencia, **los patrones de comportamiento basados en clases usan la herencia para distribuir el comportamiento entre clases y para describir algoritmos y flujos de control,** de modo que son relaciones estáticas (fijadas en tiempo de compilación).

Mientras que, los patrones de objetos, tratan con las relaciones entre objetos, que pueden cambiarse en tiempo de ejecución y son más dinámicas.

Los patrones de comportamiento tienen que ver con algoritmos y con la asignación de responsabilidades a objetos. Los patrones de comportamiento describen no sólo patrones de clases y objetos, sino también patrones de comunicación entre ellos. Estos patrones describen el flujo de control complejo que es difícil de seguir en tiempo de ejecución, lo que nos permite olvidarnos del flujo de control para concentrarnos simplemente en el modo en que se interconectan los objetos.

Patrón Template Method

Propósito: *Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos de un algoritmo* ***sin cambiar su estructura****.* Cuando digo "sin cambiar su estructura", hago referencia a que una subclase no puede cambiar el orden de llamada de un método redefinido. Es la clase padre quien me va a definir el orden de llamada.

Al definir algunos de los pasos de un algoritmo usando operaciones abstractas, el método plantilla fija su ordenación, pero permite que las subclases modifiquen dichos pasos para adecuarse a sus necesidades.

El objetivo de este patrón es que podamos definir en una operación el esqueleto de un algoritmo o de cierta lógica que luego iremos a ejecutar, y DEJAR algunos detalles DE IMPLEMENTACION, para que los hijos los puedan sobrescribir mediante HERENCIA, pero solo algunos detalles concretos, ya que, el cuerpo principal del algoritmo estará en el padre.

Un método plantilla define un algoritmo en términos de operaciones abstractas que las subclases deber redefinir para proporcionar un determinado comportamiento.

APLICABILIDAD:

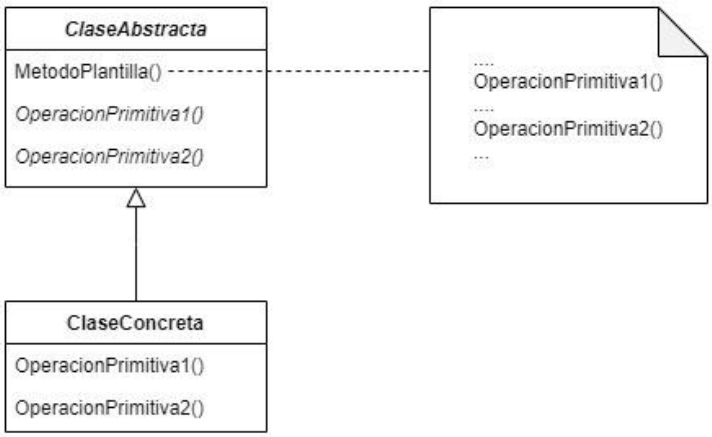
Este patrón debería usarse:

- Para implementar las partes de un algoritmo que no cambian y dejar que sean las subclases quienes implementen el comportamiento que puede variar. O sea que en este patrón tenemos una clase abstracta que tiene definido el esqueleto, un conjunto de operaciones abstractas que van a ser parte de la plantilla. Estas operaciones van a ser implementadas y redefinidas en una clase concreta que le DARÁ EL COMPORTAMIENTO ESPECÍFICO.

- Cuando el comportamiento repetido de varias subclases debería factorizarse y ser localizado en una clase común para evitar el código duplicado.

- Para controlar las extensiones de las subclases.
  El método plantilla utiliza métodos especiales (métodos de enganche) en ciertos puntos, siendo los únicos puntos que pueden ser redefinidos y, por lo tanto, los únicos puntos donde es posible la extensión. Podemos definir un método plantilla que llame a operaciones “de enganche” en determinados puntos, permitiendo así las extensiones sólo en esos puntos. Las operaciones de enganche me proporcionan el comportamiento predeterminado que puede ser modificado por las subclases si es necesario.

Estructura ESTATICA del patrón:



Explicación de estructura: En este patrón tenemos una clase abstracta que tiene definido el esqueleto, un conjunto de operaciones primitivas abstractas que van a ser parte de la plantilla. Estas operaciones van a ser implementadas y redefinidas en una clase concreta que le DARÁ EL COMPORTAMIENTO ESPECÍFICO. Además, se encuentra una operación concreta, que es justamente la operación MetodoPlantilla () que es la que va a permitir realizar el conjunto de operaciones y la secuencia que corresponda, a partir de las implementaciones CONCRETAS. Tenemos una clase Abstracta que me define el esqueleto del algoritmo, también tenemos un método público que es el MetodoPlantilla() y después tenemos dos métodos que son protected y serán abstractos también. Cuando llamemos al método MetodoPlantilla(), lo que hará esta clase, será llamar al método OperacionPrimitiva1() que es protected, por lo tanto los hijos lo tendrán que sobrescribir, ya que es abstracto, luego el método OperacionPrimitiva1() hará lo que tenga que hacer así luego se llama a OperacionPrimitiva2().

Cada hijo tendrá que implementar los detalles de implementación.

La claseConcreta es un hijo que está implementando la clase abstracta. Este hijo lo que hará es sobrescribir esos dos métodos y aplicarán los detalles de cada operación.

Una clase abstracta delega parte o toda su implementación en las operaciones definidas en sus subclases. De ahí que no se pueda crear una instancia de una clase abstracta. Las operaciones que una clase abstracta

define, pero no implementa se denominan OPERACIONES ABSTRACTAS. Las clases que no son abstractas, se denominan clases concretas.

La clase abstracta define operaciones primitivas abstractas que son definidas por las subclases para implementar los pasos de un algoritmo.

Una clase concreta implementa las operaciones primitivas para realizar los pasos del algoritmo específicos de las subclases.

Consecuencias:

Dentro de consecuencias aparece el ¿Cómo consigue el patrón sus objetivos? ¿Cuáles son las ventajas e inconvenientes y los resultados de usar el patrón?

- El método plantilla es una técnica fundamental de reutilización de código. Son particularmente importantes en las bibliotecas de clases, ya que son el modo de factorizar y extraer el comportamiento común de las clases de la biblioteca.
- El método plantilla llevan a una estructura de control invertido. Esto se refiere a cómo una clase padre llama a las operaciones de una subclase y no al revés.
- El método plantilla llama a los siguientes tipos de operaciones:
    - Operaciones concretas (ya sea de la ClaseConcreta o de las clases cliente)
    - Operaciones concretas de ClaseAbstracta (o sea, operaciones que suelen ser útiles para las subclases)
    - Operaciones primitivas (operaciones abstractas)
    - Métodos de fabricación (patrón Factory method).
    - Operaciones de enganche.

Es importante que el método plantilla especifiquen qué operaciones son enganches (que pueden ser redefinidas) y cuales son operaciones abstractas (que deben ser redefinidas). Para reutilizar una clase abstracta apropiadamente, los escritores de las subclases deben saber qué operaciones están diseñadas para ser redefinidas.

Una subclase puede Extender el comportamiento de una operación de una clase padre redefiniendo la operación y llamando explícitamente a la operación del padre.

La idea es llamar a una operación de enganche desde un método plantilla en la clase padre. Las subclases pueden entonces redefinir esta operación de enganche. La operación de enganche no hace nada en la clase padre. Las subclases redefinen a la operación de enganche para extender su comportamiento.

IMPLEMENTACION

Nos interesan tres detalles de implementación:

- Se recomienda declarar las operaciones primitivas de tal forma que sólo puedan ser llamadas por el método plantilla (Protected, miembros protegidos).
- Minimizar las operaciones primitivas: Debe reducirse el número de operaciones primitivas que van a ser invocadas desde el método Plantilla. De esta forma, se reducirá la complejidad de las subclases y resultará menos tediosa su implementación.
- Convenios de nominación: se pueden identificar las operaciones que deberían ser redefinidas añadiendo un prefijo a sus nombres. Por ejemplo, añadir a los nombres de métodos plantilla el prefijo “Do-”. Por ejemplo: “DoCreateDocument”, “DoRead”.

Ejemplo para que se entienda la estructura: Un Windows Forms de un juego, que puede ser tanto ajedrez como las damas. La persona elige a qué quiere jugar y, dependiendo del juego, se inicializará, se jugará y se finalizará.

Código: Ejemplo de cuando una persona quiere sacar un préstamo en un banco. Préstamo hipotecario o personal.

El objetivo de este patrón es definir en una operación el esqueleto de un algoritmo o de cierta lógica que luego iremos a ejecutar, y dejar algunos detalles de implementación para que sean los hijos que mediante la herencia los puedas sobrescribir, pero solo algunos detalles ya que el cuerpo principal del algoritmo va a estar en la clase padre.