Patrones de Diseño

Yober Nain Catari Cabrera

October 9, 2020

Abstract

Resumen

Por más específico que un problema sea en el desarrollo de software, existe una gran probabilidad de que alguien haya enfrentado a un problema muy similar en el pasado, de cual en el momento se pueda modelar de la misma forma.
La estructura de las clases se refieren al modelado, por lo que esta estructura conforma la solución de problema , que probablemente puede estar inventada. Si la manera de solucionar ese problema tiene los siguientes factores: extracción, explicación y reutilización en los múltiples ámbitos, se puede analizar que es un patrón de diseño de software.

Abstract

No matter how specific a problem is in software development, there is a high probability that someone has faced a very similar problem in the past, which at the moment can be modeled in the same way. The structure of the classes refers to the modeling, so this structure makes up the problem solution, which can probably be invented. If the way to solve that problem has the following factors: extraction, explanation and reuse in the multiple domains, it can be analyzed that it is a software design pattern.

I. Introduccion

Los patrones de diseño han demostrado ser un medio útil de capturando soluciones de diseño probadas en el tiempo y facilitando su reutilización. Los patrones apuntan a representar explícitamente el conocimiento de diseño que es entendido implícitamente por profesionales calificados. Quizás en ninguna parte el enfoque del patrón ha sido más efectivo que en la ingeniería de software. Gamma y col. [1] describe el software diseñar patrones como "descripciones de objetos comunicantes y clases personalizadas para resolver problemas de diseño dentro de un contexto particular ". Las descripciones en la literatura de investigación a menudo colocan más énfasis en características novedosas que en patrones de diseño recurrentes. Como un resultado, puede ser difícil identificar, evaluar y volver a aplicar el diseño en soluciones implementadas dentro de los marcos existentes

II. Marco Teórico

i. Historia

ii. Patrones de Diseño de Software

Los patrones de diseño son soluciones típicas a problemas comunes en el diseño de software. Son como planos prefabricados que puede personalizar para resolver un problema de diseño recurrente en su código. Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular.[**?**]

iii. Elementos de un patrón

- **Patrones de diseño creacionales:** nombre estándar del patrón por el cual será reconocido en la comunidad [**?**]
- **Clasificación del patrón:** creacional, estructural o de comportamiento.[**?**]

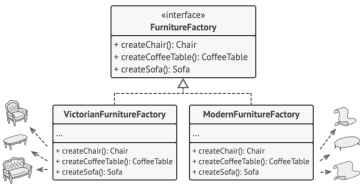
- **Intención:** ¿qué problema pretende resolver el patrón?[**?**]
- **Motivación:** Escenario de ejemplo para la aplicación del patrón.[**?**]
- **Consecuencias:** consecuencias positivas y negativas en el diseño derivadas de la aplicación del patrón[**?**]

iv. Clasificación de los patrones

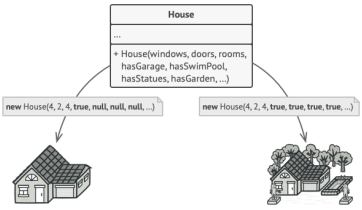
- **Patrones de diseño creacionales**

  Son patrones de diseño que se ocupan de los mecanismos de creación de objetos, tratando de crear objetos de manera adecuada a la situación. La forma básica de creación de objetos podría ocasionar problemas de diseño o agregar complejidad al diseño. Los patrones de diseño creacionales resuelven este problema controlando de alguna manera la creación de este objeto.[**?**]

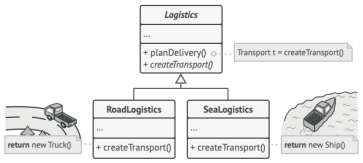
  – **Fábrica abstacta:** crea una instancia de varias familias de clases.



  – **El constructor:** bstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.



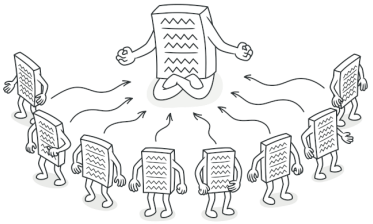
  – **Método de fabricación:** crea una instancia de varias clases derivadas.



  – **Grupo de objetos:** evita la adquisición costosa y la liberación de recursos al reciclar objetos que ya no están en uso.

  – **Prototipo:** una instancia completamente inicializada para ser copiada o clonada.

  – **Instancia única:** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Restringe la instanciación de una clase o valor de un tipo a un solo objeto.



- **Patrones de diseño estructural**

  Estos patrones de diseño tienen que ver con la composición de clases y objetos. Los patrones estructurales de creación de clases utilizan la herencia para componer interfaces. Los patrones de objetos estructurales definen formas de componer objetos para obtener una nueva funcionalidad.[**?**]

  - **Adaptador:** crea una instancia de varias familias de clases.
  - **Puente:** separa la interfaz de un objeto de su implementación.
  - **Objeto Compuesto:** permite tratar objetos compuestos como si de uno simple se tratase.
  - **Decorador:** permite tratar objetos compuestos como si de uno simple se tratase.
  - **Fachada:** una sola clase que representa un subsistema completo.
  - **Peso ligero:** reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
  - **Proxy:** proporciona un intermediario de un objeto para controlar su acceso.
  - **Módulo:** agrupa varios elementos relacionados, como clases, singletons, y métodos, utilizados globalmente, en una entidad única.

- **Patrones de diseño de comportamiento**

  Estos patrones de diseño tienen que ver con la comunicación de objetos de Class. Los patrones de comportamiento son aquellos que se ocupan más específicamente de la comunicación entre objetos.[**?**]

  - **Cadena de responsabilidad:** permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
  - **Orden:** encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
  - **Intérprete:** dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
  - **Iterador:** permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
  - **Mediador:** define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
  - **Recuerdo:** permite volver a estados anteriores del sistema.
  - **Observador:** define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
  - **Estado:** permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
  - **Estrategia:** permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
  - **Método plantilla:** define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
  - **Visitante:** permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

v. Ventajas del uso de patrones

- Conforman un amplio catálogo de problemas y soluciones.
- Estandarizan la resolución de determinados problemas.
- Condensan y simplifican el aprendizaje de las buenas prácticas.
- Proporcionan un vocabulario común entre desarrolladores.

III. Objetivos

- Los patrones son soluciones típicas a problemas comunes en el diseño orientado a objetos. Cuando una solución se repite una y otra vez en varios proyectos, alguien finalmente le pone un nombre y describe la solución en detalle. .

- el concepto de patrones de diseño e la programación patrones que resuelven varios problemas de diseño orientado a objetos

IV. Ejemplo de Codigos

V. Conclusiones

- Los patrones de diseño ayudan al arquitecto a resolver problemas frecuentes en términos de composiciones utilizadas como esquemas de trabajo, cuyo punto de partida es un conjunto predefinido de componentes y conexiones válidas.

VI. Recomendaciones

References

https://refactoring.guru/design-patterns http://siul02.si.ehu.es/alfredo/iso/06Patrones.pdf