UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

LICENCIATURA EN DESARROLLO DE SOFTWARE

DESARROLLO DE SOFTWARE IX

PATRONES DE DISEÑO

Prof. Elida E. González Jaén

Integrantes:

Pérez, Erick

Rodríguez, German

Vásquez, José

Grupo: 1LS241

10 DE ABRIL DEL 2024

Contenido

Patrones de diseño de software

Los patrones de diseño de software son técnicas para resolver problemas recurrentes de diseño de software. ¿Qué persona los definió o creó? La idea fue orientada a la informática por cuatro autores: John Vlissides, Erich Gamma, Ralph Johnson y Richard Helm. En 1995 publicaron Patrones de diseño, donde aplicaron el concepto de patrones de diseño a la programación, el libro presentaba 23 patrones que resolvían problemas del diseño orientado a objetos, y desde entonces surgieron nuevos patrones en otros campos de la programación, y hoy existen muchos otros relacionados.

Generalmente los patrones contienen ciertos elementos a tener en cuenta a la hora de utilizarlos o de decidir cual utilizar: Nombre del patrón, problema, solución, consecuencias.

También se puede decir que son de utilidad ya que nos aportan los aspectos:

Reusabilidad: Los patrones de diseño permiten reutilizar soluciones probadas en diferentes contextos. Esto ahorra tiempo y esfuerzo al no tener que reinventar la rueda cada vez que se enfrenta a un problema similar.

Consistencia: Al seguir patrones de diseño, se establece una estructura coherente en el código. Esto facilita la comprensión y el mantenimiento del software, especialmente en equipos de desarrollo.

Escalabilidad: Los patrones de diseño ayudan a diseñar sistemas escalables. Pueden adaptarse a cambios y crecimiento sin afectar la funcionalidad principal.

Comunicación: Los patrones de diseño proporcionan un lenguaje común para los desarrolladores. Al referirse a patrones conocidos, se facilita la comunicación y la colaboración entre miembros del equipo.

Mejora de la calidad: Al aplicar patrones de diseño, se reducen los errores y se mejora la calidad del software. Estas soluciones han sido probadas y optimizadas en múltiples proyectos.

Podemos clasificarlos dentro de tres grupos según su uso y solución a problemas que ofrecen dentro de 3 grupos: Patrones creacionales, patrones estructurales, patrones de comportamiento.

Patrones creacionales

Dentro de los patrones de diseño los creacionales nos proporcionan distintos mecanismos de creación de objetos que nos facilitan la reutilización del código existente como también nos ofrece una mayor flexibilidad de creación; Dentro de los

patrones de diseño creacionales encontramos los siguientes: Factory method, Abstract factory, builder, Prototype, singleton.

Factory Method es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Abstract Factory este patrón nos permite producir familias de objetos relacionados sin especificar sus clases concretas.

Builder es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

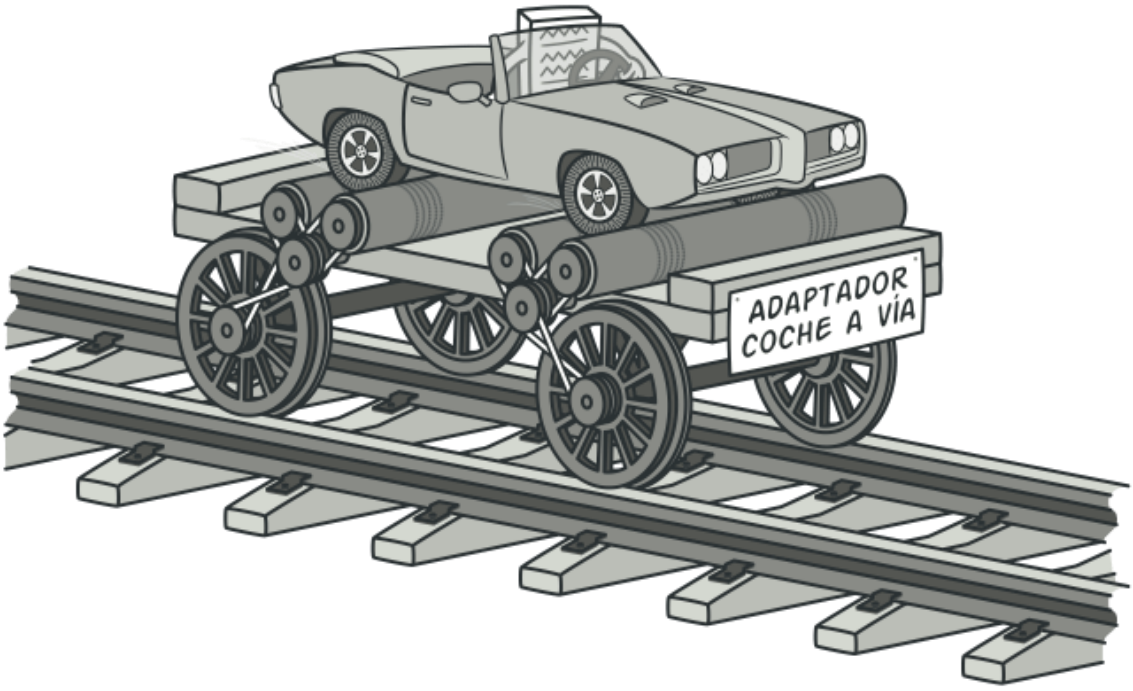
Prototype es un patrón de diseño creacional que nos permite copiar objetos existentes sin que el código dependa de sus clases.

Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Patrones de diseño Estructurales

Un Patrón Estructural en diseño de software es una solución probada y estandarizada para resolver problemas relacionados con la organización y composición de clases y objetos dentro de un sistema. Estos patrones se centran en cómo las clases y objetos se relacionan entre sí para formar estructuras mas complejas, facilitando la reutilización, la flexibilidad y la comprensión del código.

Una analogía sobre el patrón de diseño estructural Adapter podría ser como la siguiente imagen.



Así como este, existen otros patrones de diseños estructurales, entre ellos se pueden mencionar

- Adapter (adaptador)
- Bridge (puente)
- Composite (compuesto)
- Decorator (decorador)
- Facade (fachada)
- Flyweight (peso ligero)
- Proxy (proxy)

Ventajas de los patrones estructurales.

- Orden en la Construcción: Los patrones estructurales nos ayudan a organizar y construir nuestros programas de manera mas ordenada, como cuando armamos un rompecabezas y cada pieza tiene su lugar específico.
- Mantenimiento más fácil: Al dividir nuestro programa en partes mas pequeñas y ordenadas, es mas fácil arreglar errores o hacer cambios sin que todo el programa se vuelva un lío.
- Poder usar piezas diferentes: Los patrones estructurales nos permiten usar diferentes partes de programas ya existentes y hacer que funcionen juntas.
- Añadir nuevas funciones: Podemos agregar nuevas funciones a nuestro programa sin tener que cambiar todo lo que ya hemos hecho.
- Mejorar la funcionalidad y velocidad: Al usar patrones estructurales, nuestro programa puede funcionar de manera más eficiente y rápida.

Desventajas de los patrones estructurales.

- Complejidad Adicional: Al introducir patrones estructurales en un diseño, puede aumentar la complejidad del código y hacerlo más difícil de entender para los desarrolladores menos experimentados.
- Uso inapropiado: Aplicar patrones estructurales de manera incorrecta o inapropiada puede llevar a una mayor complejidad y dificultades innecesarias en el código, así como a un diseño poco claro y difícil de mantener.

Usos prácticos de los patrones estructurales:

Los Patrones Estructurales son herramientas versátiles que encuentran aplicación en diversas situaciones del desarrollo de software. Aquí se detallan algunos casos prácticos donde estos patrones demuestran su eficacia:

Gestión de Transacciones en Bases de Datos: El Patrón Proxy es útil en la gestión de transacciones en bases de datos. Puede actuar como un representante que controla el acceso a las operaciones de la base de datos, garantizando la consistencia y la integridad de las transacciones.

Optimización de Recursos en Aplicaciones Móviles: En el desarrollo de aplicaciones móviles, donde los recursos son limitados, el Patrón Flyweight puede utilizarse para conservar memoria al compartir objetos comunes entre diferentes partes de la aplicación.

Migración de Versiones de Aplicaciones: Al actualizar o migrar versiones de una aplicación, el Patrón Adapter puede ser de gran utilidad. Permite que la nueva versión de la aplicación interactúe con componentes o servicios existentes sin reescribir el código existente.

Patrones de diseño de Comportamiento

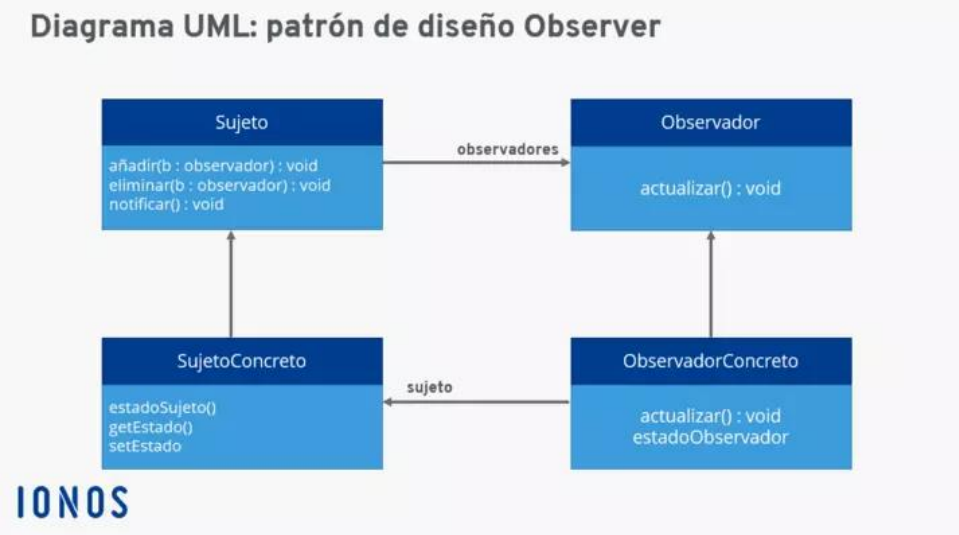
Los patrones de diseño de comportamiento son un conjunto de soluciones reutilizables para problemas comunes en el diseño de software relacionados con la interacción y comunicación entre objetos. Estos patrones se centran en cómo los objetos interactúan entre sí y cómo delegan responsabilidades. Algunos de los patrones de diseño de comportamiento más comunes incluyen:

Patrón Observer

El patrón de diseño Observer, Observer Pattern o patrón observador es uno de los patrones de diseño de software más populares. Esta herramienta ofrece la posibilidad de definir una dependencia uno a uno entre dos o más objetos para transmitir todos los cambios de un objeto concreto de la forma más sencilla y rápida posible. Para conseguirlo, puede registrarse en un objeto (observado) cualquier otro objeto, que funcionará como observador. El primer objeto, también llamado sujeto, informa a los observadores registrados cada vez que es modificado.

¿Como es la representación UML de un patrón Observer?

El modo de funcionamiento y uso de los patrones de diseño, como el patrón Observer, suele ser difícil de entender para quienes no están familiarizados con el tema. Para facilitar la comprensión, puede ser útil observar una representación gráfica del patrón de diseño



¿Qué ventajas y desventajas presenta el patrón observer?

El patrón Observer puede ser la solución adecuada a numerosos problemas de diseño. Su mayor ventaja es el alto grado de independencia entre el objeto observado (sujeto) y los objetos observadores interesados en el estado actual del sujeto. El objeto observado no requiere información sobre los observadores, ya que la interacción se realiza independiente a través de la interfaz de los observadores, que reciben actualizaciones automáticamente. De esta manera, ya no se realizan solicitudes en vano (cuando el sujeto no se ha modificado desde la última solicitud).

Sin embargo, las actualizaciones automáticas por parte del sujeto hacia todos los observadores registrados no siempre es una ventaja, ya que las informaciones de cambios transmitidas pueden ser irrelevantes para ciertos observadores. Este proceso supone un inconveniente especialmente cuando el número de observadores registrados es muy alto, puesto que se malgasta mucho tiempo de computación. Otra desventaja del patrón Observer es que el código fuente del sujeto a menudo no revela qué observadores están siendo informados.

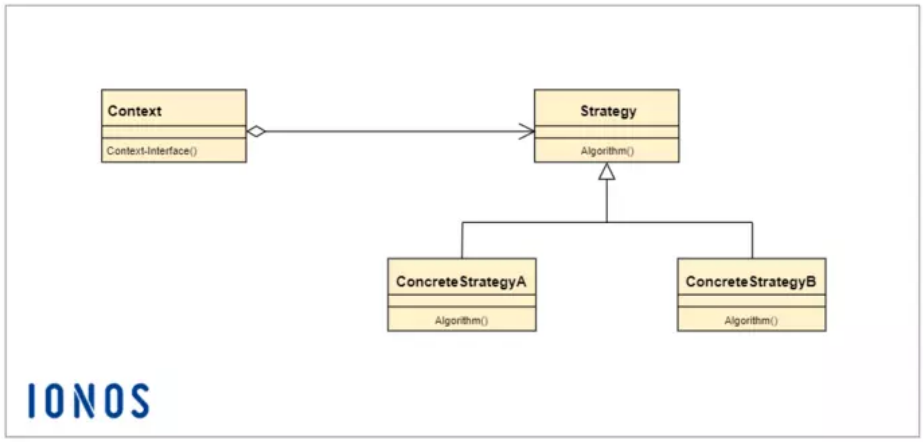
Patrón Strategy

Es un patrón de comportamiento que permite definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. Esto significa que los algoritmos pueden variar independientemente de los clientes que los utilicen. Por ejemplo, los patrones strategy describen cómo construir u organizar un grupo de clases y crear objetos. Una característica especial del patrón strategy es que un comportamiento variable de programas y objetos también se puede realizar en el tiempo de ejecución de un software.

¿Como es la representación UML de un patrón Strategy?

Los patrones strategy se diseñan generalmente con el lenguaje de modelado gráfico UML (Unified Modeling Language). El UML establece distintos tipos de diagramas para la programación orientada a objetos. Para representar un patrón strategy, se suelen utilizar los llamados diagramas de clase con al menos tres componentes básicos:

- Context (contexto o clase de contexto)
- Strategy (estrategia o clase de estrategia)
- ConcreteStrategy (estrategia concreta)



Con el patron strategy, los componentes básicos asumen funciones especiales. Los patrones de comportamiento de la clase Context se almacenan en diferentes clases Strategy. Estas clases separadas contienen los algoritmos llamados ConcreteStrategies.

Utilizando una referencia interna, el contexto puede acceder a las variables de computación externalizadas (ConcreteStrategyA, ConcreteStrategyB, etc.) si lo necesita. No interactúa directamente con los algoritmos, sino con una interfaz.

La interfaz Strategy encapsula las variantes de cálculo y puede ser implementada simultáneamente por todos los algoritmos. Para la interacción con Context, la interfaz genérica proporciona solo un método para activar los algoritmos de ConcreteStrategy. La interacción con Context incluye, además del acceso a Strategy, intercambio de datos. La interfaz Strategy también participa en los cambios de estrategia, que pueden tener lugar, además, en el tiempo de ejecución del programa.

Patrón Command

Command es un patrón de diseño de comportamiento que convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

Los elementos del patrón Command se describen a continuación:

- ICommand: Interfaz que describe la estructura de los comandos, la cual define el método de ejecución genérico para todos los comandos.
- Concrete Command: Representan los comandos concretos, éstos deberán heredar de ICommand. Cada una de estas clases representa un comando que podrá ser ejecutado de forma independiente.
- Command Manager: Este componente nos servirá para administrar los comandos que tenemos disponibles en tiempo de ejecución, desde aquí podemos solicitar los comandos o registrar nuevos.
- Invoker: El invoker representa la acción que dispara alguno de los comandos.

¿Como es la representación UML de un patrón Command?

1. El invoker obtiene un Comando del CommandManager.

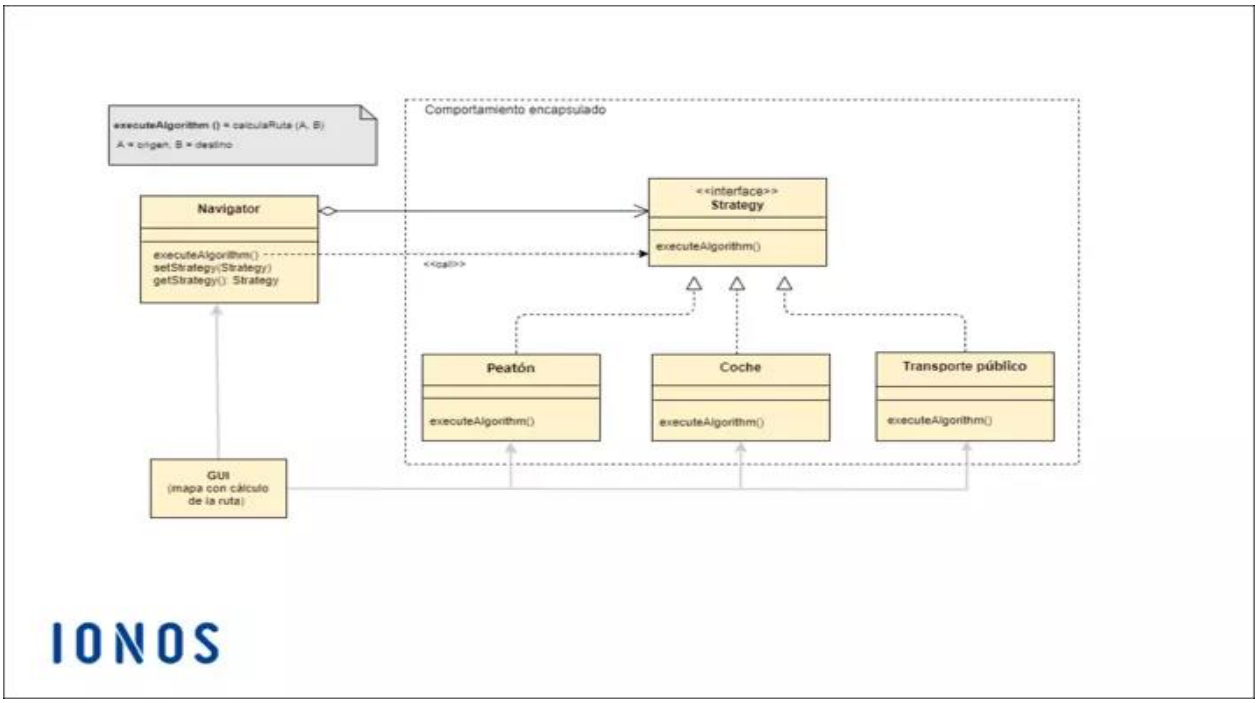
2. El invoker ejecuta el comando.
3. El invoker obtiene otro Comando del CommandManager.
4. El invoker ejecuta el comando.

Aplicación de patrones en proyectos reales

En nuestro ejemplo nos orientamos según el proyecto de estudio sobre strategy patterns de Philipp Hauer. En el mismo vamos a crear una aplicación de navegación que hace uso de un patron strategy. La aplicación debe calcular una ruta basada en los medios de transporte habituales. El usuario puede elegir entre tres opciones:

- Peatón (ConcreteStrategyA)
- Coche (ConcreteStrategyB)
- Transporte público (ConcreteStrategyC)

Si se transfieren estas especificaciones a un gráfico UML, se puede observar la estructura y la función del patron strategy.



En nuestro ejemplo, el cliente o Client es la Interfaz Gráfica de Usuario (GUI) de una aplicación de navegación con botones para el cálculo de la ruta. Si el usuario hace una selección y pulsa un botón, se calcula una ruta concreta. El Context (clase Navigator) tiene la tarea de calcular y mostrar una serie de puntos de control en el mapa. La clase Navigator tiene un método para cambiar la estrategia de enrutamiento activa. Los botones permiten cambiar fácilmente entre los medios de transporte.

Si, por ejemplo, se activa un comando correspondiente con el botón peatonal del Client, se solicita el servicio “Calcular la ruta peatonal” (ConcreteStrategyA). El método executeAlgorithm() (en nuestro ejemplo, el método: calculaRuta(A, B)) acepta un origen y un destino y devuelve un conjunto de los puntos de control de la ruta. Context acepta el comando del Client y decide la estrategia apropiada basándose en las directivas previamente definidas (policy) (setStrategy: peatonal). Mediante Call, delega la solicitud al objeto Strategy y a su interfaz.

Con getStrategy(), la estrategia actualmente seleccionada se almacena en Context (clase Navigator). Los resultados de los cálculos de ConcreteStrategy se utilizan para el procesamiento posterior y en la visualización gráfica de la ruta en la aplicación de navegación. Si el usuario elige una ruta diferente, por ejemplo, haciendo clic en el botón “Coche”, Context cambia a la estrategia solicitada

(ConcreteStrategyB) e inicia un nuevo cálculo a través de otra llamada. Al final del procedimiento, se devuelve una descripción de ruta modificada para el coche.

Conclusiones

José Vásquez

Son herramientas poderosas para abordar problemas comunes relacionados con la interacción y el comportamiento de objetos en el diseño de software. Estos patrones ofrecen una forma flexible de encapsular comportamientos específicos y permiten que estos comportamientos varíen independientemente de quien los utilice.

Es importante comprender los principios y las aplicaciones de cada patrón de diseño de comportamiento para poder seleccionar y aplicar el más apropiado para una situación dada.

German Rodríguez

Con el desarrollo de cada proyecto se pueden dar diferentes tipos de dificultades, estas dificultades pueden ser solucionadas con los patrones de diseño de software ya que están pensadas para la mayoría de los problemas que comúnmente se suelen encontrar durante el desarrollo de un proyecto. Sin embargo también hay que mencionar que aunque suelen ser soluciones, la mala utilización de estos patrones puede llegar a ser también perjudicial para el software.

Erick Pérez

Los patrones de diseño de software son soluciones generales, que me aportan factores útiles como son la reusabilidad, consistencia, escalabilidad, comunicación y mejora de la calidad a diferentes problemas en el diseño de software. y para mí que aspiro ser un buen desarrollador de software me ofrecen una guía a seguir para la creación de código limpio y bien documentado.

Bibliografía

Equipo editorial de IONOS. (2020, 19 octubre). *Strategy pattern: un patrón de diseño de software para estrategias variables de comportamiento*. IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/strategy-pattern/

Equipo editorial de IONOS. (2020a, octubre 19). *Patrón Observer: ¿en qué consiste este patrón de diseño?* IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-patron-observer/

Command. (s. f.). https://reactiveprogramming.io/blog/es/patrones-de-diseno/command

- 2014-2024 Refactoring.Guru. https://refactoring.guru/es/design-patterns/proxy

Ilustraciones por Dmitry Zhart

- Carlos Batista, publicado el 9 de noviembre de 20023 https://es.linkedin.com/pulse/patrones-de-dise%C3%B1o-descubriendo-los-estructurales-carlos-batista-tgf4e