Patrones de Diseño

Bridge and Composite

JEFFRY ARAYA CHAVES 2022437443
MÉLANIE WONG HERRERA 2023042240
NORMAN SÁENZ APEZTEGUÍA 2019003307

Índice

Patrón Bridge

El patrón bridge es un patrón de diseño estructural que permite separar una abstracción de su implementación, de modo que ambas puedan variar independientemente. Este patrón se utiliza cuando se quiere evitar un vínculo permanente entre la abstracción y su implementación, por ejemplo, cuando la implementación debe ser seleccionada o cambiada en tiempo de ejecución.

El patrón bridge implica crear una interfaz que actúa como un puente entre la abstracción y la implementación, y usar la composición para delegar la responsabilidad a uno de los objetos. De esta forma, se reduce el acoplamiento y se aumenta la cohesión de las clases.

1. Propósito*:*

El propósito del patrón bridge es separar la abstracción de la implementación, de modo que ambas puedan variar independientemente. Esto permite que el código sea más flexible y adaptable a diferentes contextos. El patrón bridge se compone de cuatro elementos: una abstracción, una implementación, una refinación de la abstracción y una concreción de la implementación.

La abstracción define la interfaz de alto nivel que se usa para interactuar con el objeto. La implementación define la interfaz de bajo nivel que se usa para acceder a los recursos del sistema. La refinación de la abstracción extiende o modifica la funcionalidad de la abstracción. La concreción de la implementación provee una instancia específica de la implementación.

2. Componentes Clave:

Abstracción: Define la interfaz de abstracción y mantiene una referencia a un objeto del tipo Implementador. La abstracción puede contener métodos que delegarán la operación al objeto Implementador.

Implementación: Define la interfaz para las clases de implementación. Esta interfaz no tiene que corresponder exactamente con la interfaz de abstracción y puede

ser muy diferente. La abstracción puede utilizar cualquier función de la implementación que necesite para realizar su trabajo.

Abstracción Refinada: Extiende o refina la interfaz definida por la Abstracción.

Implementador Concreto: Es una subclase de Implementador e implementa sus operaciones concretas.

3. Cómo Funciona:

En el patrón Bridge, no extiendes las abstracciones y sus implementaciones como una jerarquía heredada. En su lugar, defines una interfaz (la Abstracción) y realizas que las abstracciones concretas contengan un objeto de tipo Implementador. Las abstracciones delegan las llamadas a los métodos correspondientes en el Implementador.

4. Ventajas:

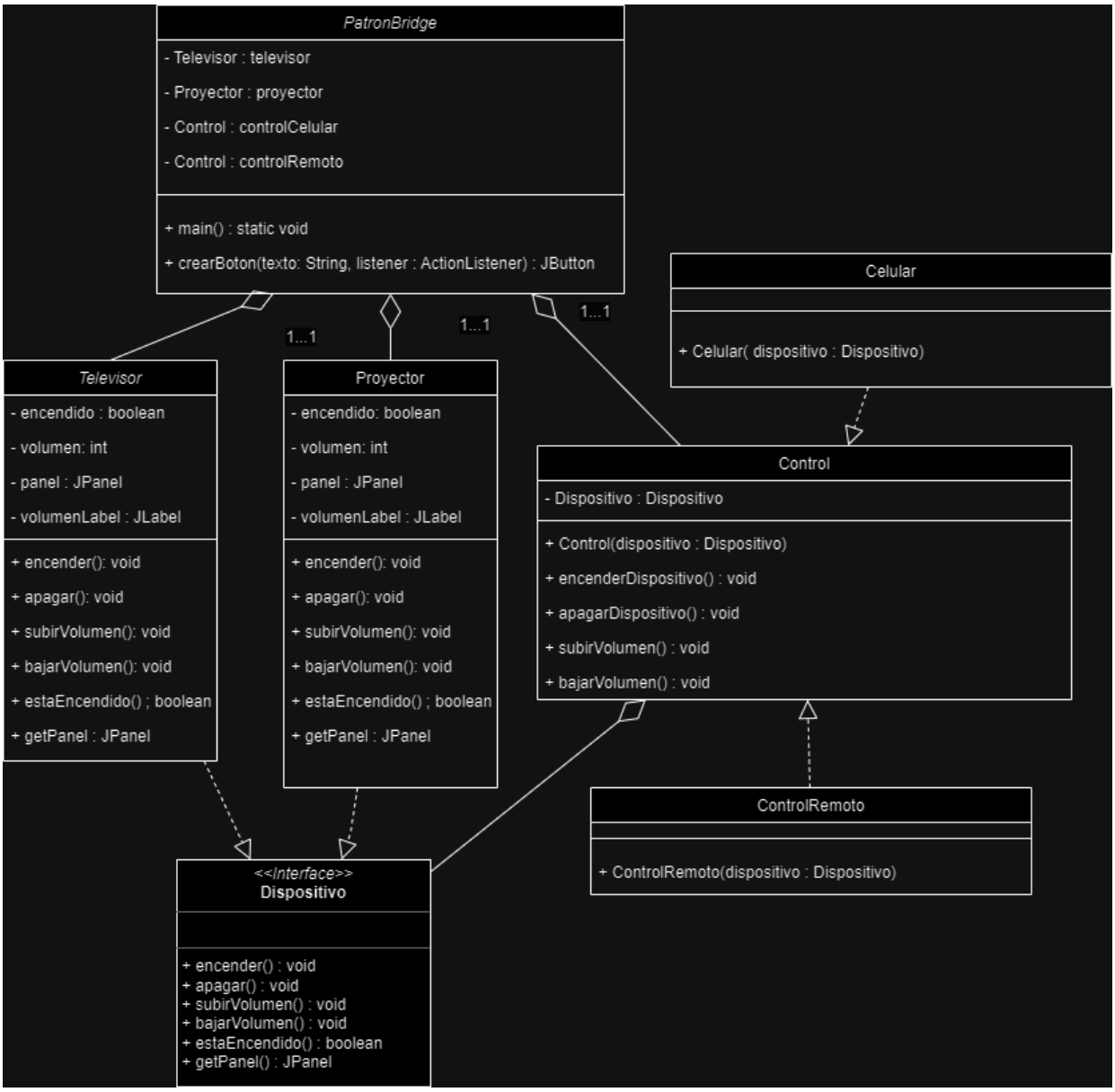
- Permite crear jerarquías de clases paralelas, evitando el problema de la explosión de clases.
- Facilita la extensibilidad y la reutilización del código, ya que se pueden añadir nuevas abstracciones e implementaciones sin modificar las existentes.
- Mejora el desacoplamiento entre las capas de abstracción e implementación, favoreciendo la cohesión y el principio de responsabilidad única.
- Permite cambiar la implementación en tiempo de ejecución, adaptándose a las necesidades del contexto.

5. Uso Común:

Se utiliza en situaciones donde es necesario evitar un vínculo permanente entre una abstracción y su implementación, especialmente en casos donde los cambios en la implementación no deben afectar a los clientes que usan la abstracción. Es común en el desarrollo de bibliotecas y frameworks, donde se requiere flexibilidad y extensibilidad.

El patrón Bridge es útil cuando se anticipan cambios frecuentes tanto en las abstracciones como en las implementaciones, o cuando las abstracciones e implementaciones pueden desarrollarse independientemente.

6. Ejemplo UML



https://drive.google.com/file/d/1DEUjfBo082afYqhApN8hJtrlfe4vjOaj/view?usp=sharing

Patrón Composite

El “Patrón Composite” es un patrón de diseño estructural que permite componer objetos en estructuras de árbol para representar jerarquías de parte-todo. El patrón composite define una clase abstracta que representa tanto a las hojas como a los contenedores de la jerarquía, y permite tratarlos de manera uniforme. El patrón composite facilita la manipulación de objetos compuestos, ya que se puede aplicar el mismo método a una hoja o a un subárbol sin distinguir entre ellos.

1. Propósito:

El propósito del Patrón Composite en la Programación Orientada a Objetos (POO) es organizar objetos en estructuras de árbol para representar jerarquías parte-todo, permitiendo que los clientes traten de manera uniforme tanto a objetos individuales como a grupos de objetos.

2. Componentes clave:

Componente: Es la interfaz o clase abstracta que define las operaciones comunes para los objetos en la composición. Actúa como el ancestro común para los componentes concretos.

Leaf (Hoja): Representa los elementos finales de la estructura que no tienen subelementos. Implementan las operaciones de la interfaz Componente.

Composite: Es un contenedor que puede tener elementos Leaf y otros contenedores Composite. Implementa las operaciones de la interfaz Componente y añade operaciones para agregar o eliminar subcomponentes.

3. Como funciona

En el Patrón Composite define una interfaz común para todos los objetos, tanto simples como compuestos, de manera que se pueden tratar de forma uniforme. Esto facilita la manipulación de los objetos y reduce la complejidad del código. Un ejemplo de uso del patrón composite es el de las aplicaciones gráficas que pueden agrupar varias figuras en una sola entidad.

4. Ventajas:

- Permite crear jerarquías complejas de objetos de forma sencilla y uniforme.
- Facilita la manipulación y el recorrido de las estructuras compuestas mediante la implementación de una interfaz común para todos los componentes.
- Favorece el principio de responsabilidad única, ya que cada componente se encarga de su propia funcionalidad y delega la de sus hijos en ellos.
- Promueve el principio de abierto/cerrado, ya que se pueden añadir nuevos tipos de componentes sin modificar el código existente.

5. Uso Común:

Este patrón se usa cuando se quiere representar jerarquías de objetos que pueden tener una o varias ramas. Un ejemplo de uso común del patrón composite es el sistema de archivos de un ordenador, donde cada carpeta puede contener archivos o subcarpetas, y cada archivo o subcarpeta puede tener diferentes propiedades y comportamientos. El patrón composite permite tratar a cada carpeta o archivo como un objeto único, independientemente de su nivel de profundidad en la jerarquía.

6. Ejemplo UML

https://drive.google.com/file/d/14lteNdSdPEhQuDolhLHFUfYTByj54X3g/view?usp=sharing

Bibliografía

Bruns, M. (15 de Marzo de 2023). *Medium*. Obtenido de https://medium.com/@MTrax/golang-composite-pattern-b611f30fb2f#:~:text=In%20object%2Doriented%20programming%2C%20the%20Composite%20pattern%20is%20a%20structural,Gang%20of%20Four%E2%80%9D%20design%20patterns.

geeksforgeeks. (7 de Noviembre de 2023). *GeeksForGeeks*. Obtenido de https://www.geeksforgeeks.org/bridge-design-pattern/

Gulati, V. (24 de Abril de 2022). *ScalerTopics*. Obtenido de https://www.scaler.com/topics/design-patterns/bridge-design-pattern/

IONOS. (9 de Noviembre de 2020). *DIgital Guide IONOS*. Obtenido de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-composite/

JKHENZA. (4 de Julio de 2023). *DEV*. Obtenido de https://dev.to/jkenzai/programacion-orientado-a-objetos-patrones-de-diseno-codigo-limpio-arquitectura-limpia-4d22

Kushwaha, N. (s.f.). *LEARNCSDESING*. Obtenido de https://www.learncsdesign.com/learn-the-composite-design-pattern/

Martin, E., & baeldung. (8 de Enero de 2024). *Baeldung*. Obtenido de https://www.baeldung.com/java-bridge-pattern

Merced, A. (26 de Noviembre de 2023). *DEV.to*. Obtenido de https://dev.to/alexmercedcoder/oop-design-patterns-in-javascript-3i98

Mundada, M. (27 de Diciembre de 2023). *Scaler Topics*. Obtenido de https://www.scaler.com/topics/design-patterns/composite-design-pattern/

Patros, P., & CPEng. (18 de Agosto de 2023). *RAYGUN*. Obtenido de https://raygun.com/blog/object-oriented-software-patterns-part-two/#:~:text=The%20Bridge%20pattern%20separates%20an,codebase%20adaptable%20to%20future%20changes.

Saxena, B. (20 de Octubre de 2020). *DZone*. Obtenido de https://dzone.com/articles/composite-design-pattern-in-java-1

The Code Team. (4 de Octubre de 2023). *Medium*. Obtenido de https://medium.com/@thecodebean/bridge-design-pattern-implementation-in-java-f71c853979fe

UMLBoard. (17 de Enero de 2023). *UMLBoard*. Obtenido de https://www.umlboard.com/design-patterns/bridge.html