POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

7. Bridge

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

- *Cierta persona debe ser responsable de interpretar ciertos datos que guarda en cierto soporte (p.e. en una libreta o en una hoja de cálculo o de memoria o …). Esta persona debe saber escribir, utilizar el ordenador, técnicas de memorización, … lo cual es complejo, tiende a crecer (p.e. curricula vitae difíciles de encontrar) e inflexible (p.e. formación continua por nuevos soportes)*
- *Una alternativa sería que esta persona sólo sea responsable de interpretar dichos datos y que otras personas por separado sean responsables de guardar en diversos soportes específicos dependiendo de su especialidad. Así, cada uno está especializado en un única responsabilidad, lo cual no es complejo ni tiende a crecer y es flexible escogiendo una nueva pareja para el nuevo soporte*

- ***Desacopla una abstracción de su implementación de tal forma que las dos pueden variar independientemente***

1. Problema

- Cuando una abstracción puede tener varias implementaciones posibles, la forma más habitual de darles cabida es mediante la herencia. Una clase abstracta define la interfaz de la abstracción, y las subclases concretas la implementan de distintas formas. Pero este enfoque no siempre es lo bastante flexible. La herencia liga una implementación a la abstracción de forma permanente, lo que dificulta modificar, extender y reutilizar abstracciones e implementaciones de forma independiente.
  - Produciendo jerarquías de herencia paralelas
  - Produciendo dependencia de la plataforma

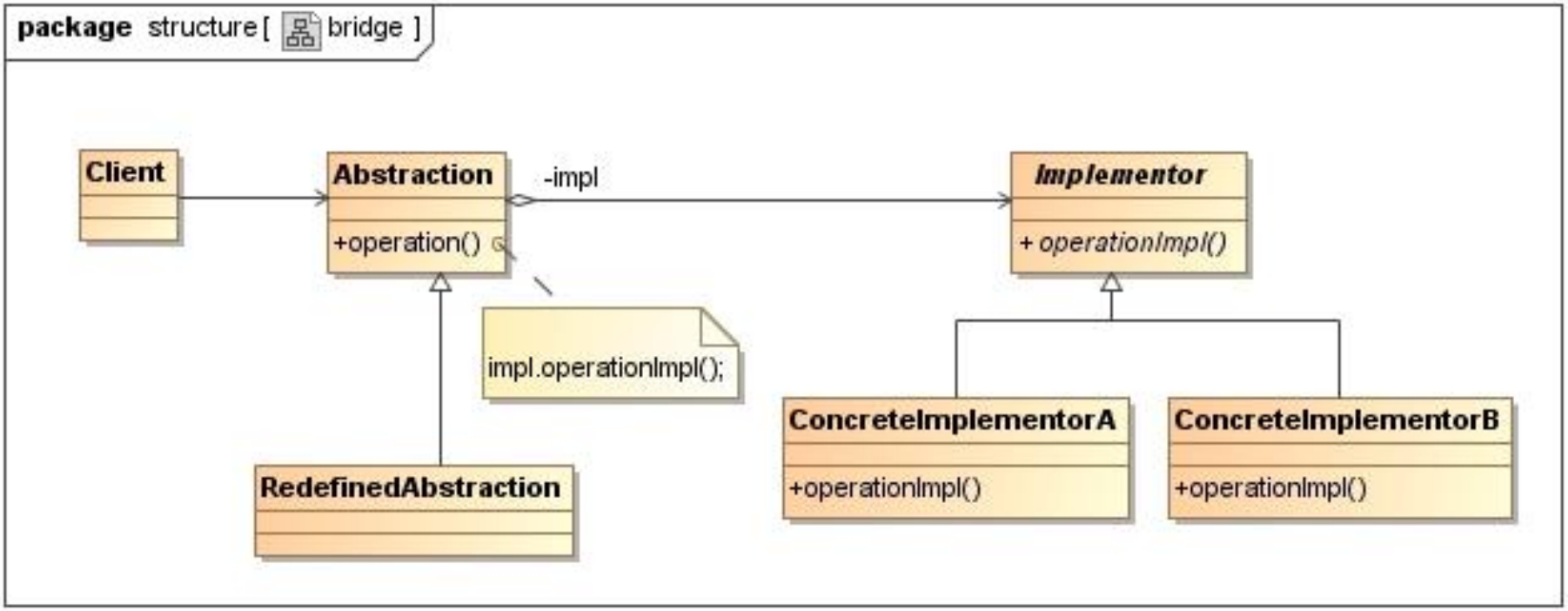
1. Problema

- Úsese cuando:
  - Quiera evitar un enlace permanente entre una abstracción y su implementación. Por ejemplo, cuando debe seleccionarse o cambiarse la implementación en tiempo de ejecución
  - Tanto las abstracciones como sus implementaciones deberían ser extensibles mediante subclases. En este caso, el patrón *Bridge* permite combinar las diferentes abstracciones y sus implementaciones, y extenderlas independientemente.
  - Los cambios en la implementación de una abstracción no deberían tener un impacto en los clientes; es decir, su código no tendría que ser recompilado.

1. Problema

- Quiera en C++ ocultar completamente a los clientes la implementación de una abstracción. En C++ la representación de una clase es visible en la interfaz de la misma
- Tenga una proliferación de clases en una jerarquía de herencia paralela
- Quiera compartir una implementación entre varios objetos (tal vez usando un contador de referencias) y este hecho deba permanecer oculto al cliente.

2. Solución


package structure [bridge]
Client
Abstraction
+operation()
-impl
Implementor
+operationImpl()
impl.operationImpl();
RedefinedAbstraction
ConcreteImplementorA
+operationImpl()
ConcreteImplementorB
+operationImpl()

2. Solución

- Todas las operaciones de las subclases de la abstracción se implementan en términos de operaciones abstractas de la interfaz del implementador. Esto desacopla las abstracciones de la diferentes implementaciones específicas de cada plataforma. Nos referimos a la relación entre la abstracción y la implementación como un puente, porque una abstracción con su implementación, permitiendo que ambas varíen de forma independiente.
- En situaciones en las que sólo hay una implementación, no es necesario crear una clase abstracta Implementador. Éste es un caso degenerado del patrón *Bridge*, cuando hay una relación uno-a-uno entra Abstracción e Implementador. Sin embargo, esta separación sigue siendo útil cuando un cambio en la implementación de una clase no debe afectar a sus clientes existentes, éstos no debería ser recompilados, sólo enlazados.

2. Solución

- Si Abstracción conoce a todas las clases ImplementadorConcreto, puede crear una instancia de ellas en su constructor; puede decidir de cuál basándose en los parámetros pasados a su constructor (por ejemplo: lista enlazada vs *hash*)
- También es posible delegar totalmente la decisión en otro objeto: *Abstract Factory* sabe qué tipo de objetos implementadores crear para la plataforma en uso. Una ventaja es que Abstracción no está acoplada directamente a ninguna de las clases Implementador.

3. Consecuencias

- No une permanentemente una implementación a una interfaz, sino que la implementación puede configurarse en tiempo de ejecución. Incluso es posible que un objeto cambie su implementación en tiempo de ejecución.
  - Desacoplar *Abstracción* e *Implementador* también elimina de la implementación dependencias de tiempo de compilación. Ahora, cambiar una clase ya no requiere recompilar la clase Abstracta y sus clientes. Esta propiedad es esencial cuando debemos asegurar la compatibilidad binaria entre distintas versiones de una biblioteca de clases
- Podemos extender las jerarquía de Abstracción y de Implementador de forma independiente
- Podemos aislar a los clientes de los detalles de implementación, como el comportamiento de objetos implementadores y el correspondiente mecanismo de conteo de referencias (si es que hay alguno)

4. Relaciones

- ***Abstract Factory*** para crear y configurar el *Bridge*

5. Comparativa

- *Bridge vs Adapter*
  - Ver ***Adapter vs Bridge***