Instituto Tecnológico de Costa Rica

Curso: Diseño de software

Profesor: Ing. Cristian Campos.


| Estudiante: | Carnet: |
| --- | --- |
| Jordano Escalante López | 2018161994 |


Primer Semestre

Año: 2024

# Patrones de Diseño: Elementos de Software Orientado a objetos reutilizables.

## Patrón Adapter y Bridge.

# Patrón Adapter.

## Definición.

El patrón de diseño adapter consiste en la implementación de un intermediario que permita la comunicación y/o cooperación de componentes que no se pueden comunicarse entre ellos, resulta más eficiente implementar un adaptador que modificar los componentes existentes para que logren una comunicación entre ellos.
Cuando se implementa un adaptador o intermediario, la clase desde la que recibe los datos se llama “Target”, en español objetivo, y el componente, clase u objeto que se beneficia de la adaptación se llama adaptee.
Es posible implementar adaptadores de dos vías que permitan al Target comunicarse con el adaptee, y a la vez permitan al adaptee comunicarse con el Target.

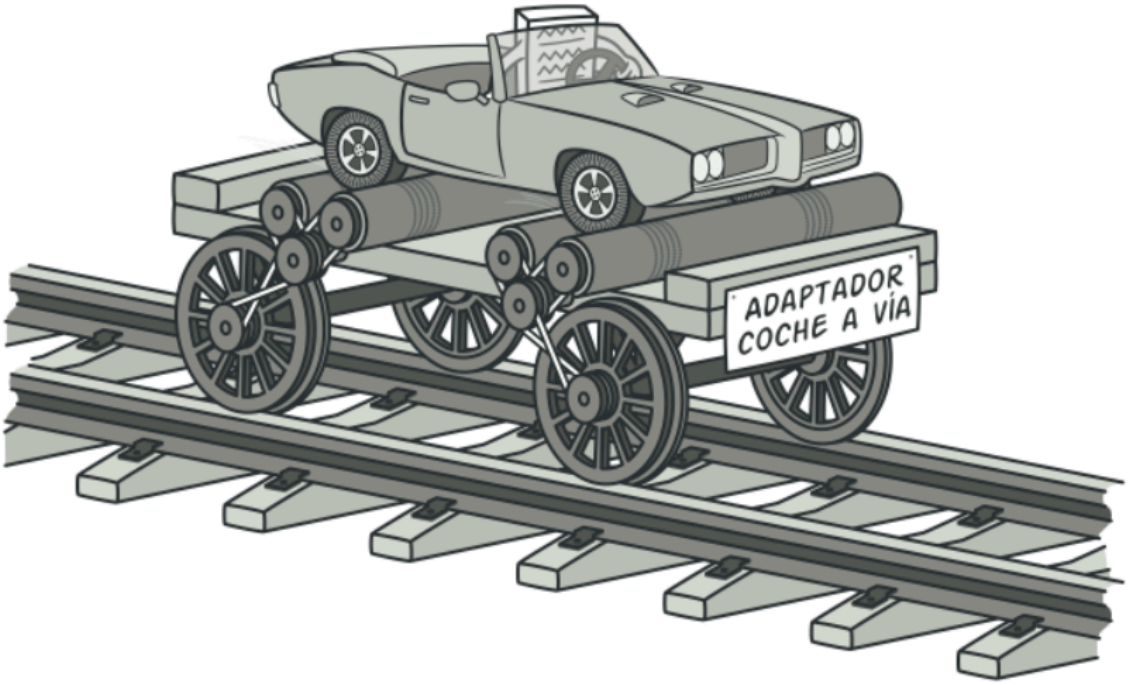


Imagen 1: Ilustración de ejemplo del patrón adapter

Existen dos distintas implementaciones del patrón adapter, el caso del adaptador de clases y el caso del adaptador de objetos.

## Adaptador de clase:

La principal diferencia respecto al adaptador de objetos, es que este emplea la herencia, ya sea sencilla o múltiple (en lenguajes en los que se permita esta), para heredar la clase Target e implementar la interface que necesitamos, de esta manera la nueva clase va a implementar métodos que permitan el uso de la superclase e implementen la interface que necesitamos.

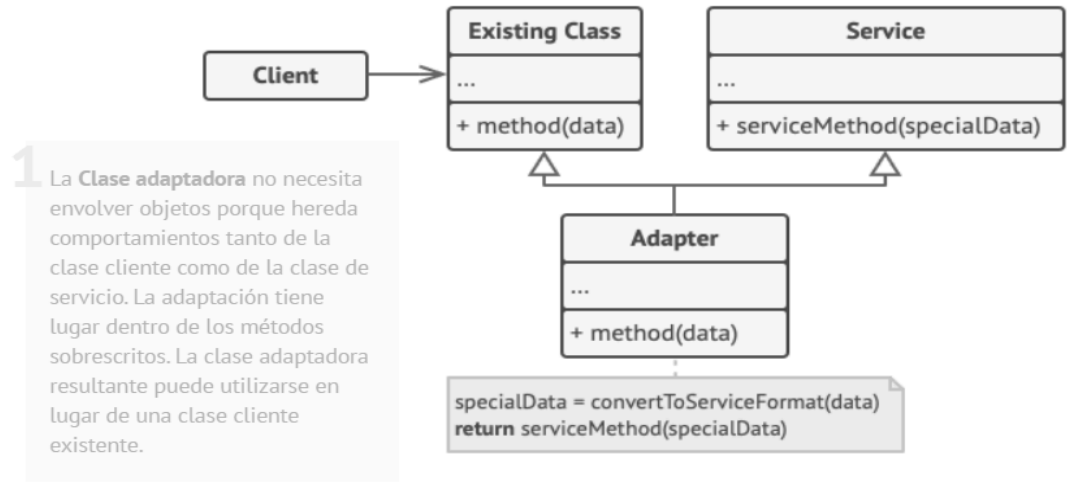


Imagen 2: Diagrama de clases UML que ejemplifica un caso de adaptador de clases.

## Adaptador de objetos:

Este caso no emplea la herencia, en este sencillamente se implementa una interfaz o clase aparte que implemente las funcionalidades necesarias para la comunicación entre objetos que necesitan comunicarse.

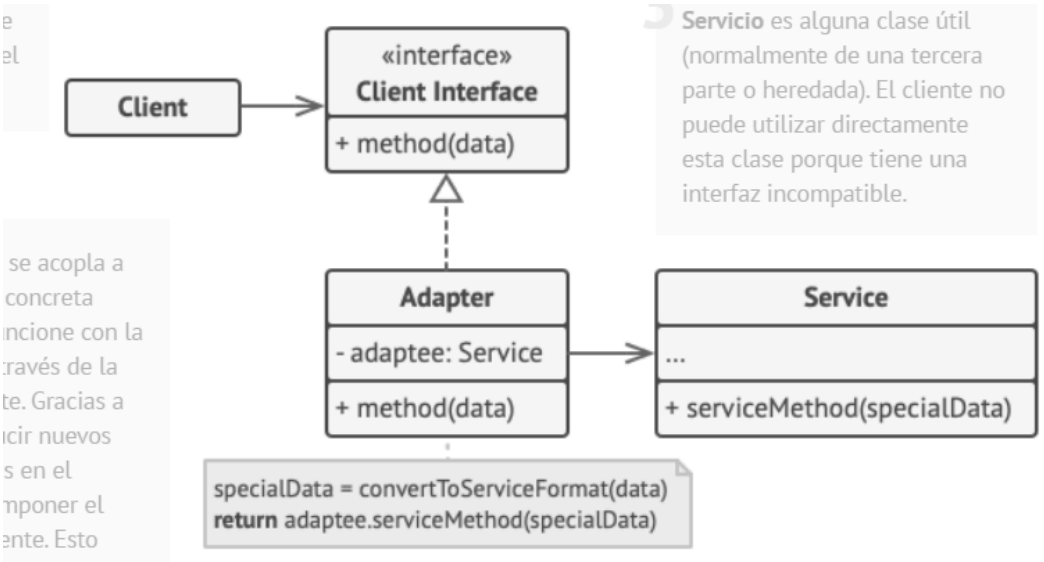


Imagen 3: Diagrama UML ejemplificando un caso de adaptador de tipo objeto.

## Casos donde se recomienda emplear.

Este tipo de soluciones son recomendables de emplear en sistemas en los que se desea comunicar componentes no compatibles pero que no deseamos modificar, como sistemas heredados, comunicación entre entidades u organizaciones distintas, comunicación entre componentes muy robustos cuyas modificaciones implican una inversión significativa.

## Ejemplo.

Imaginemos que tenemos un sistema de facturación en un comercio, nuestro sistema almacena los datos en una base de datos que nos retorna la información en archivos con formato JSon, ahora imaginemos que nuestro sistema debe enviar reportes a hacienda para realizar la declaración de impuestos, para ello con cada compra debe enviarse los detalles la empresa, datos del comprador y la información de los productos que ha comprado, el problema es que hacienda solamente trabaja con formato XML, a pesar de ser un formato relativamente similar al formato JSon debemos realizar algunas modificaciones para que pueda realizarse una comunicación efectiva entre el comercio y hacienda.
No podemos realizar cambios al sistema de hacienda dado que es una entidad externa y no tenemos control de la misma, además realizar cambios en esta no es una opción viable ya que deberíamos cambiar un sistema muy extenso del que, además, dependen muchas otras entidades, cambiar nuestro sistema de facturación puede ser una opción, pero nos podría implicar una gran inversión de tiempo y recursos, por lo que lo mejor es optar por crear un intermediario, un adaptador de las comunicaciones entre nuestro sistema y hacienda.
El sistema podría llamarse “AdaptadorComunicaciones”, este puede ser una interface o una clase, e implementar un método para envío de información que transforme los datos de JSon a XML, y otro método para recepción de la información, que transforme los datos de XML a JSon, siendo este un adaptador de doble vía que permite comunicación entre el Target “Hacienda” y el “adaptee”, nuestro sistema de facturación.

Otro ejemplo sería un sistema que recibe información de la bolsa de valores, los datos se reciben en formato XML, y para analizarlos nuestro sistema envía los datos a un sistema externo, el problema es que este sistema solamente entiende datos en formato JSon.

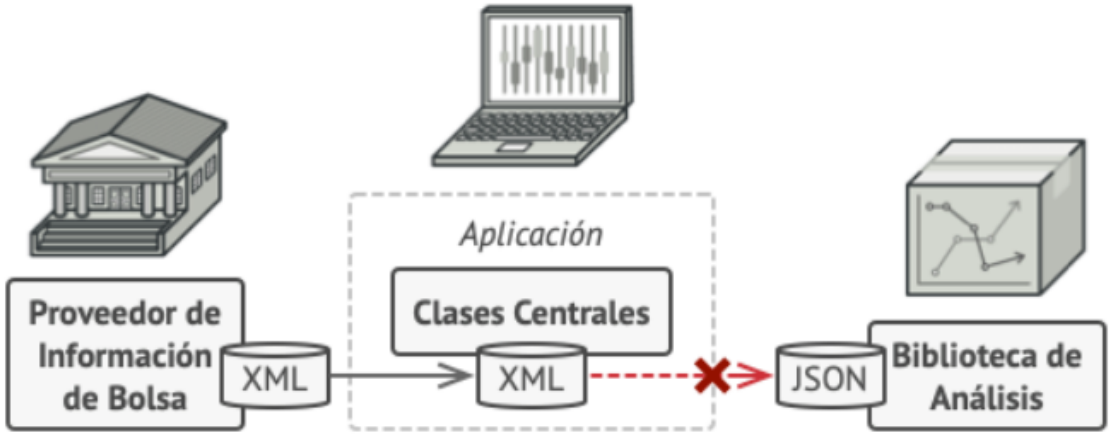

Imagen 4: Ilustración del caso de comunicación fallida entre dos sistemas que manejan datos en formatos distintos.

Para solucionar esto, el sistema debe emplear un adapter que transforme los datos de XML a JSon, de manera que la información de la bolsa recibida en formato XML sea transformada a formato JSon para ser enviada al sistema de análisis..

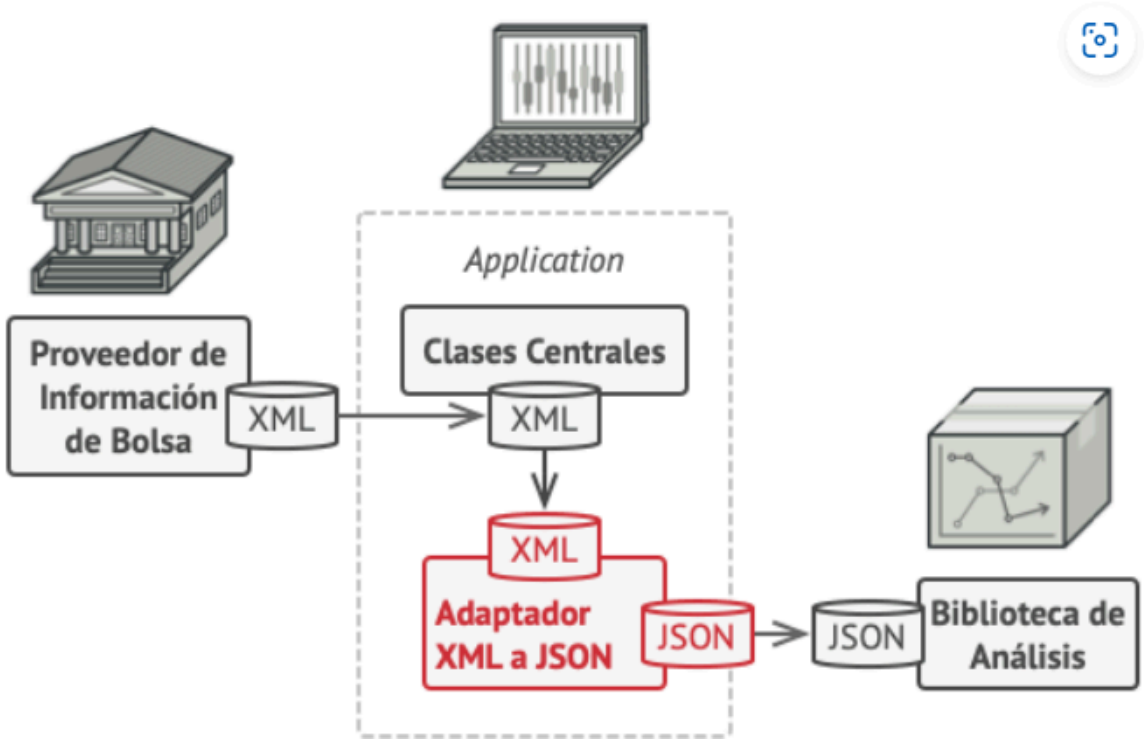


Imagen 5: Ilustración de la implementación de un adaptador que permita el intercambio de información entre sistemas con datos en distintos formatos.

# Patrón Bridge.

## Definición.

Bridge es un patrón que permite separar la implementación de un programa de su abstracción, normalmente se utiliza para separar clases o grupos de ellas muy grandes en grupos más pequeños y reducir el tamaño de los componentes.
Usualmente lo que se hace es uso de la composición, de manera que alguna característica o grupo de ellas se agrupen en una clase especializada para ese tipo de características, y por medio de la composición forme parte de los atributos o métodos de la clase que deseamos desarrollar sin modificarla a ella misma.

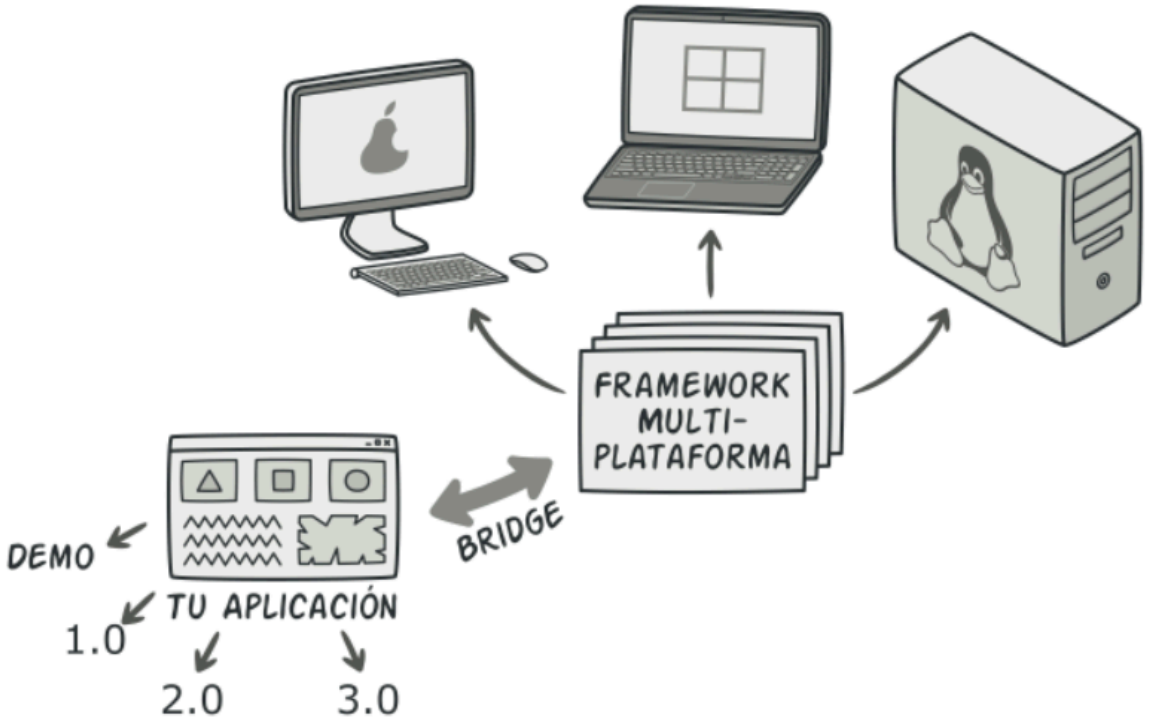


Imagen 6: Ilustración que muestra un escenario en el que nos conviene implementar un bridge.

## Casos donde se recomienda emplear.

Se recomienda hacer uso de ésta en casos en los que se va a desarrollar una aplicación grande en la que podemos contar con gran cantidad de especializaciones para una interfaz o aplicación, el puente sería la composición que se logra al establecer una interfaz o clase como atributo para crear variar interfaces o atributos especializados a partir de ella y especializar estos de manera que estos se encarguen de los detalles de implementación para cada tipo de sub-clase/interfaz en que pueda derivarse la composición.

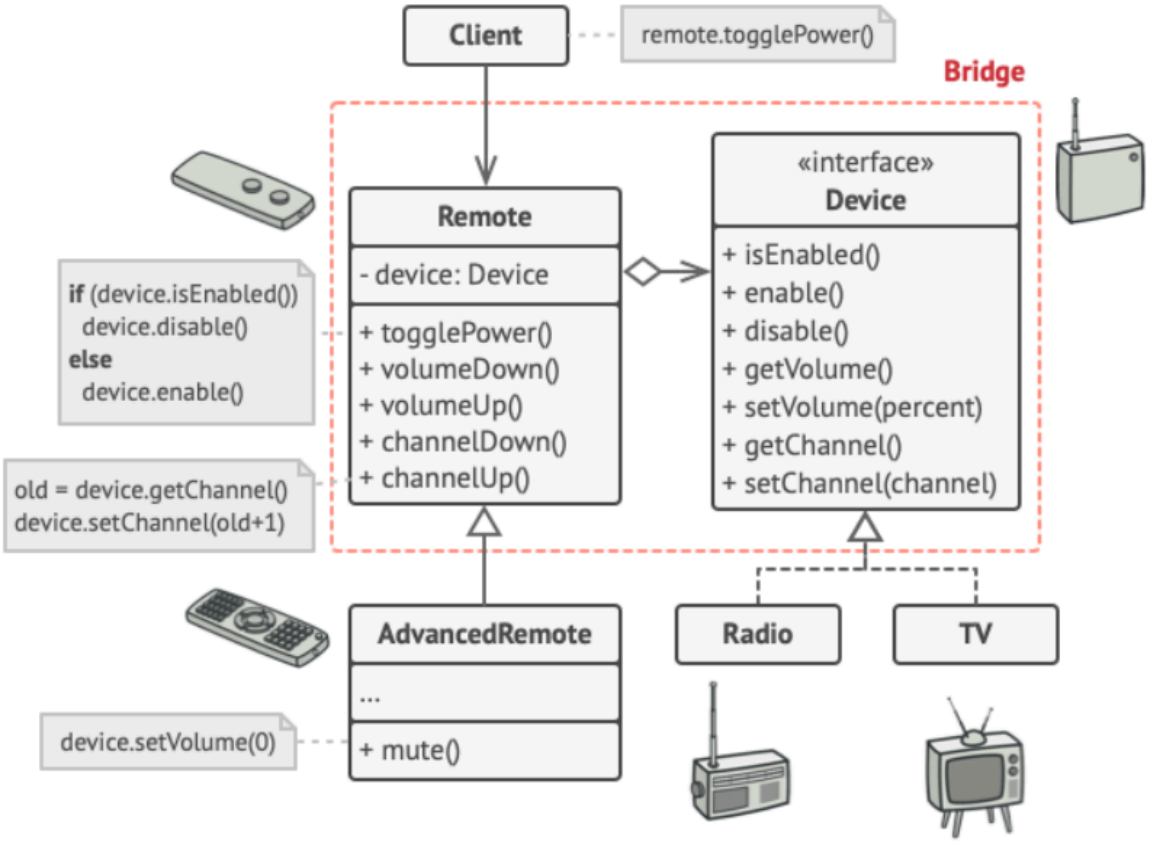


Imagen 7: Representación UML de un caso donde se implementa una interfaz “Device” usando el patrón bridge.

## Ejemplo.

Siguiendo el ejemplo de la ilustración anterior, Imaginemos que tenemos un caso en el que debemos desarrollar un programa embebido para un control remoto universal, casi todos los controles remotos van a tener los mismo métodos para cambiar de canal, emisora, aplicación dependiendo del dispositivo que estemos controlando, también vamos a necesitar los métodos para subir y bajar el volumen.
Dependiendo del dispositivo que controlamos tendríamos que implementar un método de control de volumen para un televisor y otro para una radio ya que ambos funcionan de forma distinta, a mayor medida de dispositivos que controlemos, mayor cantidad de métodos.
Para solucionar esto implementamos una interfaz que implementa estos métodos abstractos, la agregamos por medio de una composición a nuestra clase de control remoto y posteriormente realizaríamos las especializaciones de interfaz para incluir los detalles de implementación para controlar un radio o televisor desde nuestro control remoto.

# Referencias:

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education.

Refactoring.Guru. (s.f.). Adapter en Java / Patrones de diseño. https://refactoring.guru/es/design-patterns/adapter

ProgramacionYMas.com. (s.f.). Patrón de Diseño Adapter. https://programacionymas.com/blog/patron-adapter

Refactoring Guru. (s. f.). Patrón de diseño Bridge. Recuperado de https://refactoring.guru/es/design-patterns/bridge