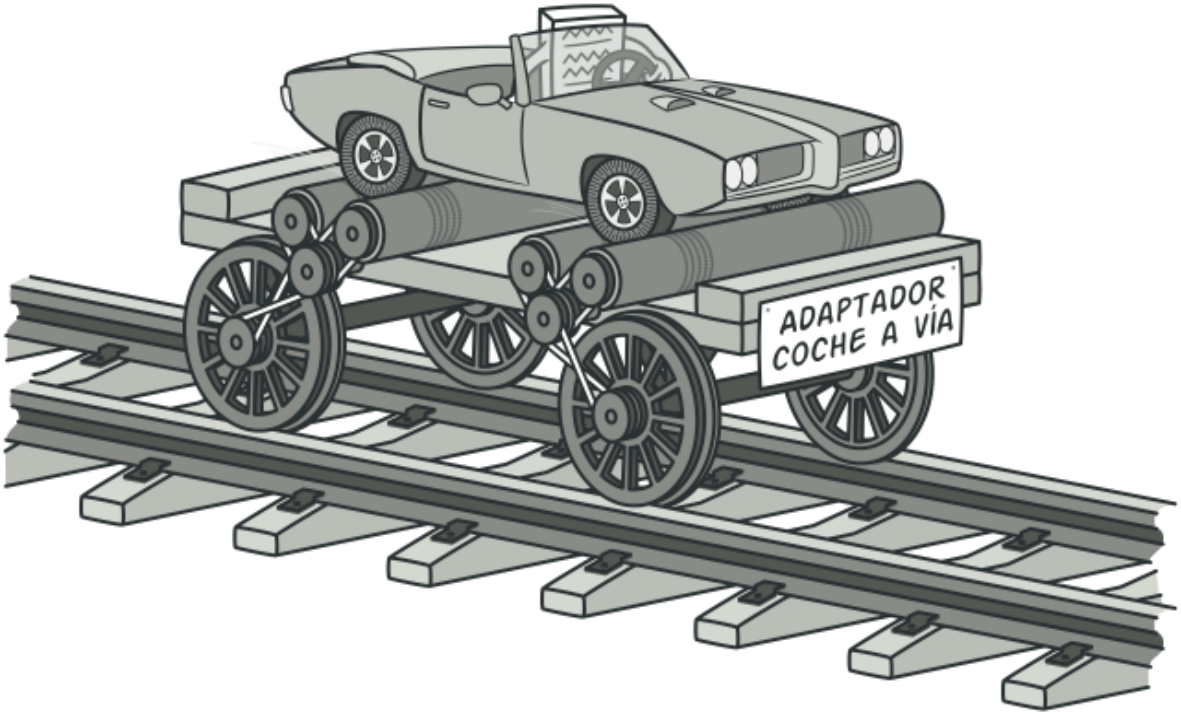
Adapter

También llamado: Adaptador, Envoltorio, Wrapper

Propósito

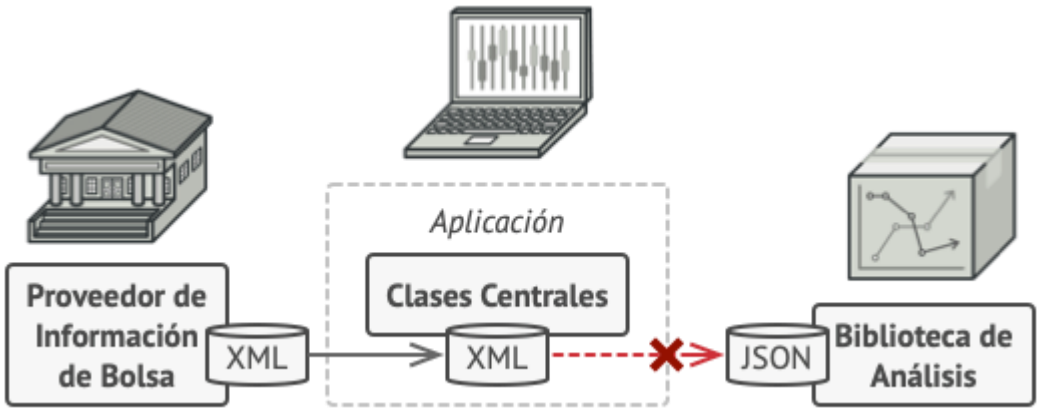
Adapter es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.



Problema

Imagina que estás creando una aplicación de monitoreo del mercado de valores. La aplicación descarga la información de bolsa desde varias fuentes en formato XML para presentarla al usuario con bonitos gráficos y diagramas.

En cierto momento, decides mejorar la aplicación integrando una inteligente biblioteca de análisis de una tercera persona. Pero hay una trampa: la biblioteca de análisis solo funciona con datos en formato JSON.



No puedes utilizar la biblioteca de análisis “tal cual” porque ésta espera los datos en un formato que es incompatible con tu aplicación.

Podrías cambiar la biblioteca para que funcione con XML. Sin embargo, esto podría descomponer parte del código existente que depende de la biblioteca. Y, lo que es peor, podrías no tener siquiera acceso al código fuente de la biblioteca, lo que hace imposible esta solución.

Solución

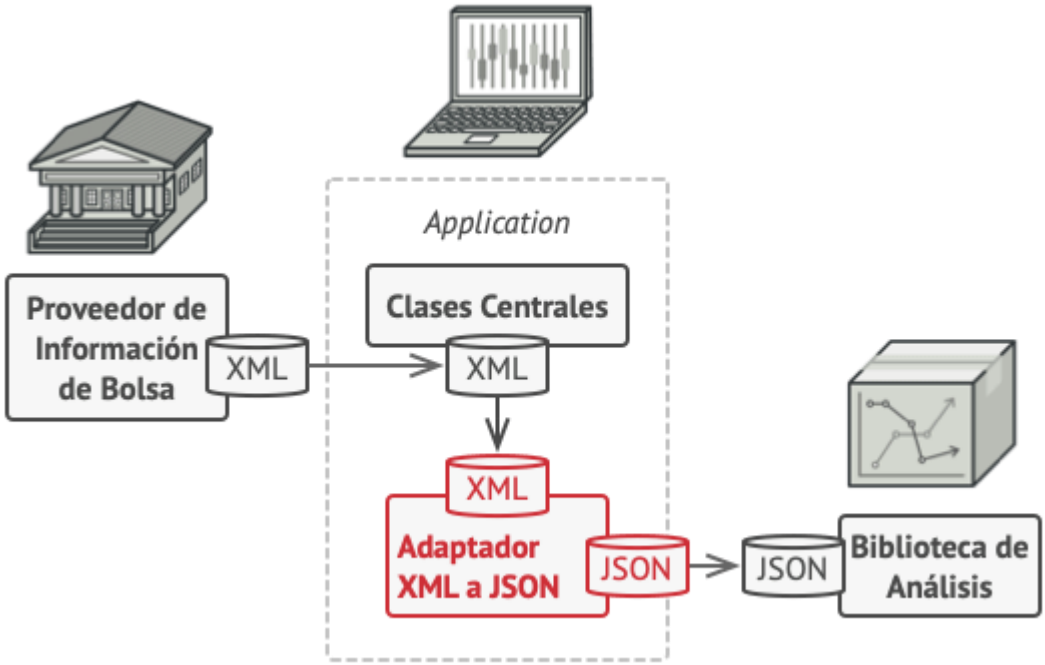
Puedes crear un *adaptador*. Se trata de un objeto especial que convierte la interfaz de un objeto, de forma que otro objeto pueda comprenderla.

Un adaptador envuelve uno de los objetos para esconder la complejidad de la conversión que tiene lugar tras bambalinas. El objeto envuelto ni siquiera es consciente de la existencia del adaptador. Por ejemplo, puedes envolver un objeto que opera con metros y kilómetros con un adaptador que convierte todos los datos al sistema anglosajón, es decir, pies y millas.

Los adaptadores no solo convierten datos a varios formatos, sino que también ayudan a objetos con distintas interfaces a colaborar. Funciona así:

1. El adaptador obtiene una interfaz compatible con uno de los objetos existentes.
2. Utilizando esta interfaz, el objeto existente puede invocar con seguridad los métodos del adaptador.
3. Al recibir una llamada, el adaptador pasa la solicitud al segundo objeto, pero en un formato y orden que ese segundo objeto espera.

En ocasiones se puede incluso crear un adaptador de dos direcciones que pueda convertir las llamadas en ambos sentidos.



Regresemos a nuestra aplicación del mercado de valores. Para resolver el dilema de los formatos incompatibles, puedes crear adaptadores de XML a JSON para cada clase de la biblioteca de análisis con la que trabaje tu código directamente. Después ajustas tu código para que se comunique con la biblioteca únicamente a través de estos adaptadores. Cuando un adaptador recibe una llamada, traduce los datos XML entrantes a una estructura JSON y pasa la llamada a los métodos adecuados de un objeto de análisis envuelto.

Analogía en el mundo real

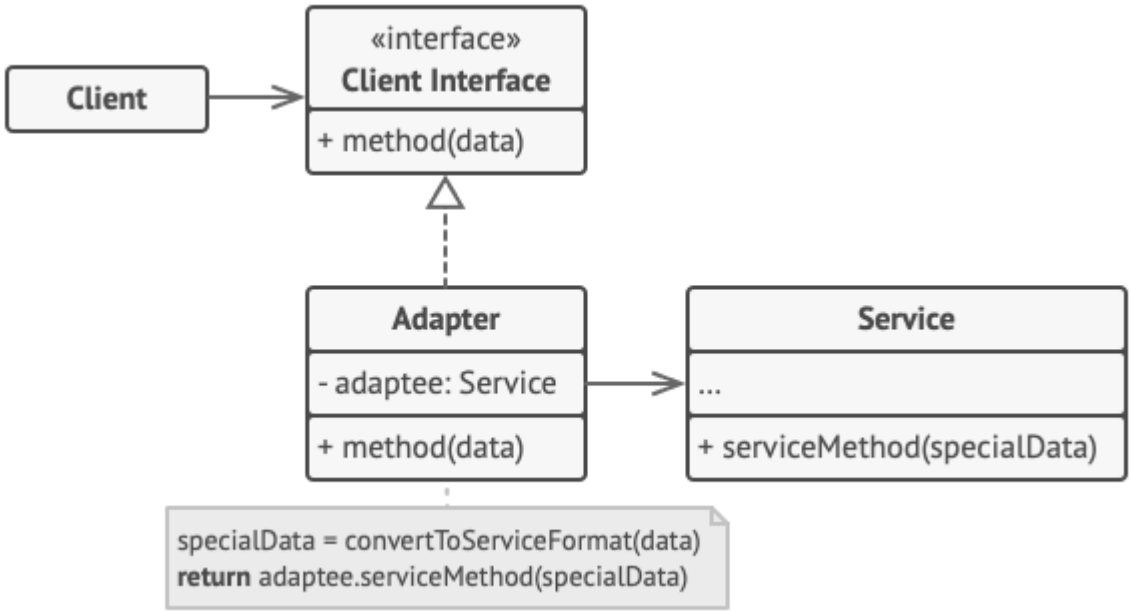
Una maleta antes y después de un viaje al extranjero.

Cuando viajas de Europa a Estados Unidos por primera vez, puede ser que te lleves una sorpresa cuanto intentes cargar tu computadora portátil. Los tipos de enchufe son diferentes en cada país, por lo que un enchufe español no sirve en Estados Unidos. El problema puede solucionarse utilizando un adaptador que incluya el enchufe americano y el europeo.

Estructura

Adaptador de objetos

Esta implementación utiliza el principio de composición de objetos: el adaptador implementa la interfaz de un objeto y envuelve el otro. Puede implementarse en todos los lenguajes de programación populares.

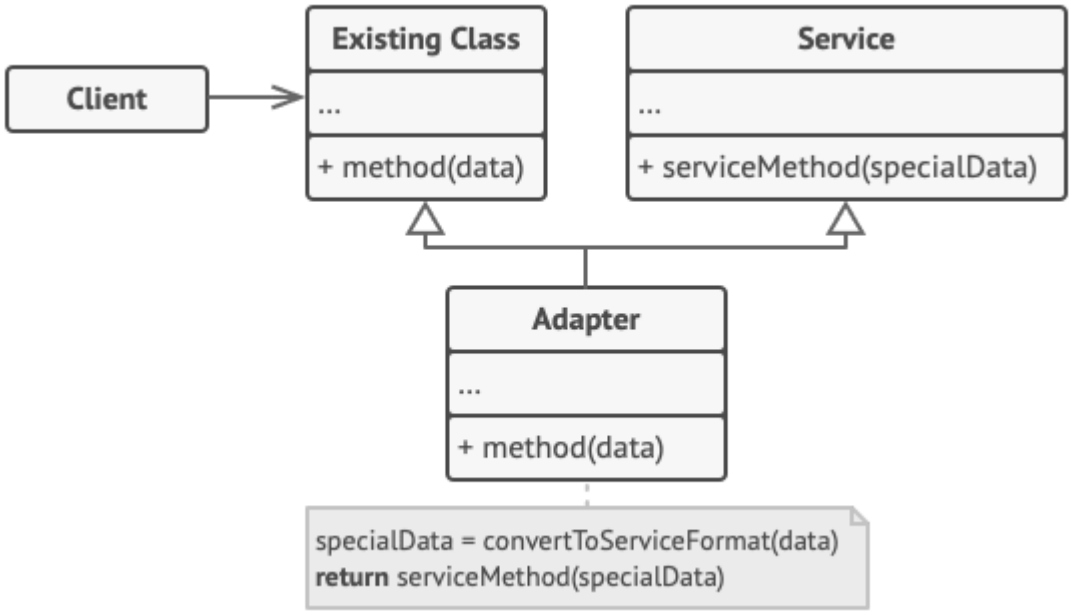


1. La clase **Cliente** contiene la lógica de negocio existente del programa.
2. La **Interfaz con el Cliente** describe un protocolo que otras clases deben seguir para poder colaborar con el código cliente.
3. **Servicio** es alguna clase útil (normalmente de una tercera parte o heredada). El cliente no puede utilizar directamente esta clase porque tiene una interfaz incompatible.
4. La clase **Adaptadora** es capaz de trabajar tanto con la clase cliente como con la clase de servicio: implementa la interfaz con el cliente, mientras envuelve el objeto de la clase de servicio. La clase adaptadora recibe llamadas del cliente a través de la interfaz de cliente y las traduce en llamadas al objeto envuelto de la clase de servicio, pero en un formato que pueda comprender.

5. El código cliente no se acopla a la clase adaptadora concreta siempre y cuando funcione con la clase adaptadora a través de la interfaz con el cliente. Gracias a esto, puedes introducir nuevos tipos de adaptadores en el programa sin descomponer el código cliente existente. Esto puede resultar útil cuando la interfaz de la clase de servicio se cambia o sustituye, ya que puedes crear una nueva clase adaptadora sin cambiar el código cliente.

Clase adaptadora

Esta implementación utiliza la herencia, porque la clase adaptadora hereda interfaces de ambos objetos al mismo tiempo. Ten en cuenta que esta opción sólo puede implementarse en lenguajes de programación que soporten la herencia múltiple, como C++.



1. La **Clase adaptadora** no necesita envolver objetos porque hereda comportamientos tanto de la clase cliente como de la clase de servicio. La adaptación tiene lugar dentro de los métodos sobrescritos. La clase adaptadora resultante puede utilizarse en lugar de una clase cliente existente.