# Trabajo Práctico N°2
# Patrones de Diseño - Adapter

Universidad de la Cuenca del Plata.
Facultad de Ingeniería y Tecnología.
Ingeniería de Software II.

Boattini Nicolás.
2021.

## Tabla de Contenidos

# Adapter

## El Patrón

### Propósito

Adapter es un patrón de diseño estructural que permite colaborar a objetos incompatibles. Este también es conocido como Wrapper o Adaptador. Dicho de otra manera, este patrón permite que la interfaz de una clase se convierta en una que pueda ser consumida por el cliente.

Puede lograr la reutilización del código de una biblioteca, por ejemplo, creando una clase de Adaptador. Esta clase adaptador se ubica entre el código cliente, y el que código que está en la librería. Es uno de los patrones más comunes y más útiles disponibles para nosotros como desarrolladores de software.

### Utilización

Se utiliza muy a menudo en sistemas basados en algún código heredado. En estos casos, los adaptadores crean código heredado con clases modernas.

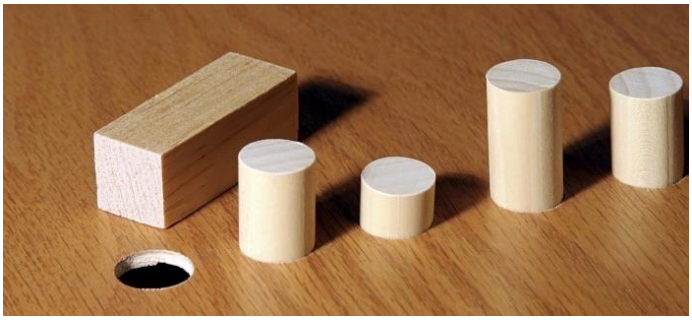
### Adapter en el mundo real


Al viajar al exterior por primera vez nos encontramos con el problema de que a dónde vayamos las tomas de corriente son distintas a las de nuestro lugar de residencia. Esto se soluciona llevando con nosotros un adaptador que incluya múltiples tipos de enchufe.

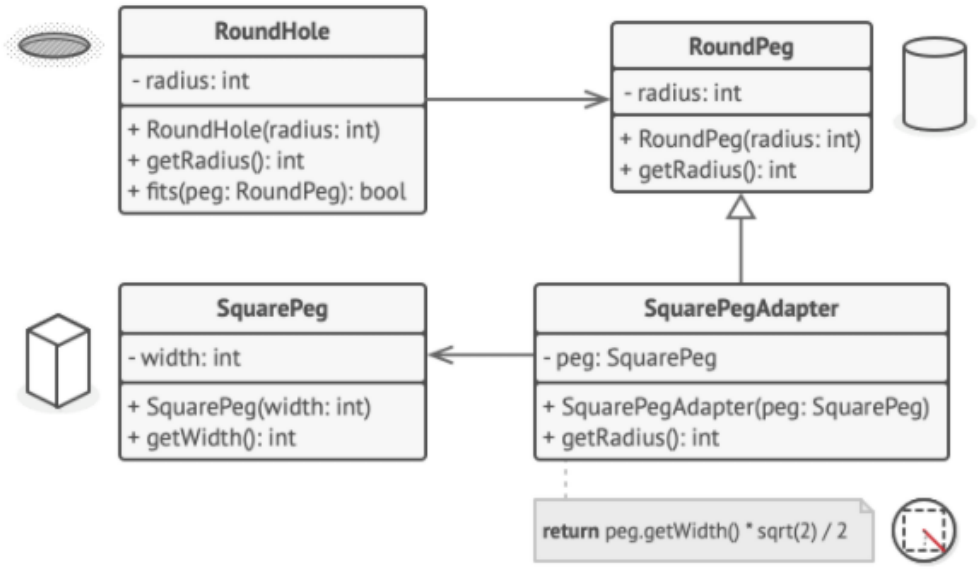
# Aplicaciones de Adapter

## Caso: Figuras y Huecos

Supongamos, tenemos un hueco redondo de radio r, luego tenemos figuras cilíndricas de radio r, distintos al hueco. Y tenemos 1 figura cuadrada. A simple vista, y para conocimientos entre objetos, esta última es incompatible con el hueco. Aquí entra en juego el Adapter, quien permite que puedan ser compatibles, ya que, por ejemplo, un cuadrado tiene una circunferencia circunscrita, y por tanto un radio r.



## Diagrama de Clase

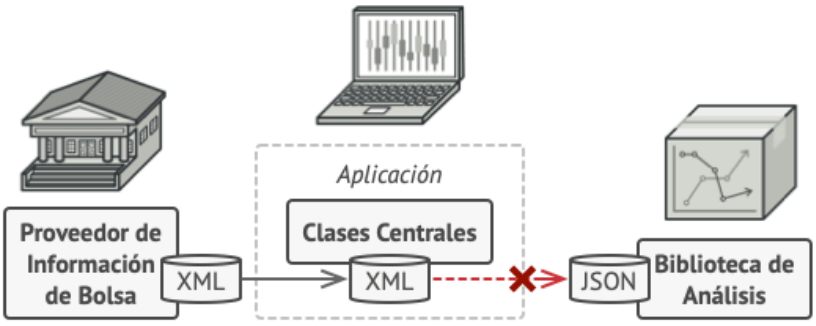



*Adaptando piezas cuadradas a agujeros redondos.*

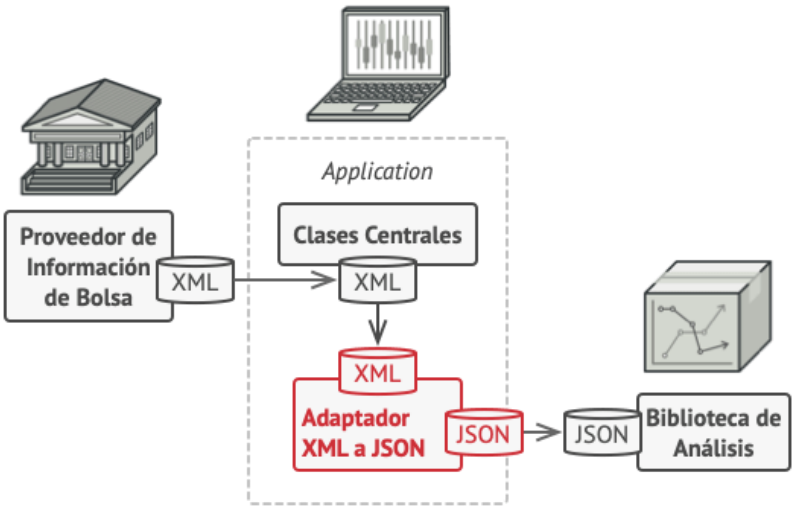
**Caso: Monitoreo Mercado de Valores**

Tenemos 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.

En cierto momento, se decide mejorar la aplicación integrando una inteligente biblioteca de análisis de un tercero. Pero hay un problema: la biblioteca de análisis solo funciona con datos en formato JSON.




Se crea el adapter de XML a JSON. Después ajustamos el código para que se comunique con la biblioteca únicamente a través de este adapter. Cuando el adapter 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.



**Funcionamiento.**

El adaptador obtiene una interfaz compatible con uno de los objetos existentes.

Utilizando esta interfaz, el objeto existente puede invocar con seguridad los métodos del adaptador.

Al recibir una llamada, el adaptador pasa la solicitud al segundo objeto, pero en un formato y orden que ese segundo objeto espera.

## Estructura

### Participantes:

Los participantes son:

- Target: la interfaz consumida por el cliente.
- Adapter: adapta la interfaz incompatible a la interfaz de Target.
- Adaptee: la interfaz incompatible.
- Cliente: el consumidor final.

### Interfaz Adaptadora:

Esta implementación utiliza el principio de composición de objetos: el adaptador implementa la interfaz de un objeto y envuelve el otro.

La Interfaz Target describe un protocolo que otras clases deben seguir para poder colaborar con el código cliente.

La clase Adapter es capaz de trabajar tanto con la clase Cliente como con la clase Adaptee. Implementa la interfaz con el cliente, mientras envuelve el objeto de la clase Adaptee. La clase adaptadora recibe llamadas del cliente a través de la interfaz y las traduce en llamadas al objeto envuelto de la clase de servicio, pero en un formato que pueda comprender. adaptador obtiene una interfaz compatible con uno de los objetos existentes. Utilizando esta interfaz, el objeto existente puede invocar con seguridad los métodos del adaptador.

Adaptee es alguna clase útil (normalmente de una tercera parte o heredada). El cliente no puede utilizar directamente esta clase porque tiene una interfaz incompatible.

La clase Cliente contiene la lógica de negocio existente del programa.

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.

**Clase Adaptadora:**

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

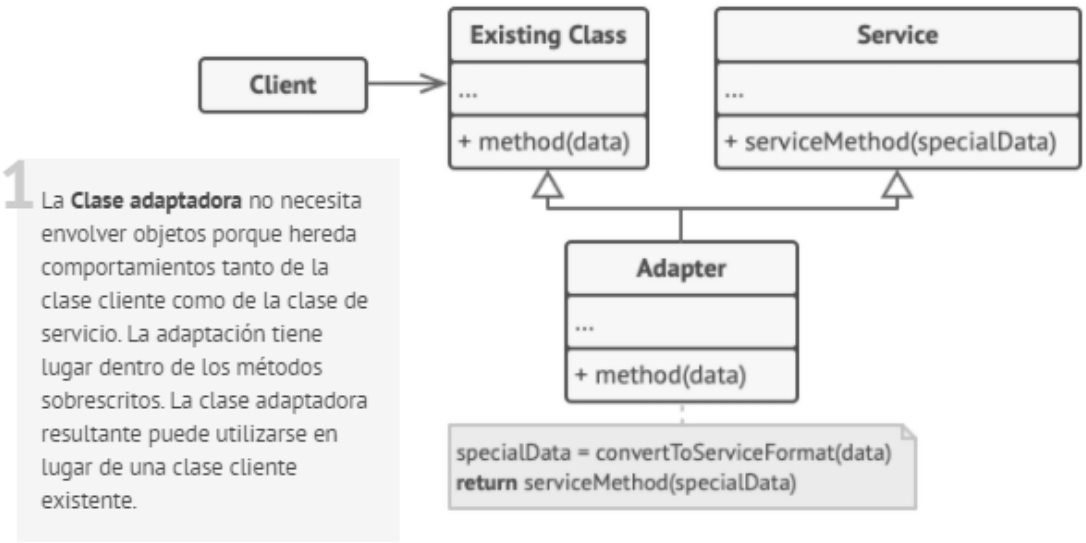
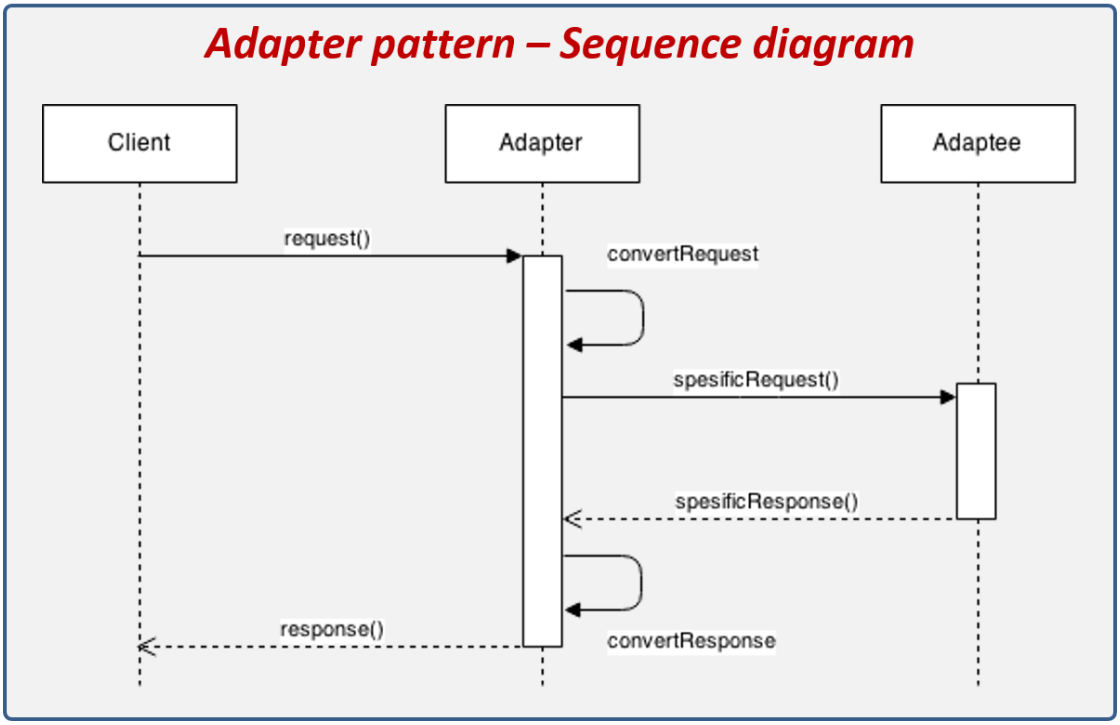


**Diagrama de Secuencia**




El Cliente invoca al Adapter con parámetros genéricos.

El Adapter convierte los parámetros genéricos en parámetros específicos del Adaptee.

El Adapter invoca al Adaptee.

El Adaptee responde.

El Adapter convierte la respuesta del Adaptee a una respuesta genérica para el Client.

El Adapter responde al Client con una respuesta genérica.

**Implementación:**

Para este implementar este patrón debemos definir una clase Adaptador, que extienda del componente existente e implemente la interfaz obligatoria. De este modo se tiene la funcionalidad que se quería y se cumple la condición de implementar la interfaz.

Por ejemplo, para el caso de Figuras y Huecos:

```
public class FiguraCuadradaAdapter: FiguraRedonda
{
    public FiguraCuadrada cuad;
    2 references | nicoboattini, 1 day ago | 1 author, 1 change
    public FiguraCuadradaAdapter(FiguraCuadrada fig)
    {
        this.cuad = fig;
    }
    2 references | nicoboattini, 1 day ago | 1 author, 1 change
    public override double GetRadio()
    {
        return (cuad.GetAncho() * ((Math.Sqrt(2)) / (2)));
    }
}
```

## Conclusiones Finales

### Consecuencias

**Adaptador de Clase:**

Adapta Adaptee a Target encargando a una clase Adaptee concreta. Como consecuencia, una clase adaptadora no funcionará cuando se desea adaptar una clase y todas sus subclases.

Permite a los Adapter sobrescribir algo de comportamiento de Adaptee, ya que Adapter es una subclase de Adaptee.

**Adaptador de Objeto:**

Permite que un único Adapter trabaje con muchos Adaptees. El Adapter también puede agregar funcionalidad a todos los Adaptees de una sola vez.

hace difícil sobrescribir el comportamiento de Adaptee. Esto requerirá derivar Adaptee y hacer que Adapter se refiera a la subclase en lugar que al Adaptee por sí mismo.

**¿Cuánta adaptación hace el Adapter?:**

Adapter varía en la cantidad de trabajo que hace para adaptar Adaptee a la interfaz Target. Hay un espectro de trabajo posible, desde una simple conversión hasta soportando un conjunto de operaciones enteramente diferentes. La cantidad de trabajo que Adapter hace depende de cuan similar es la interfaz Target con Adaptee.

**Adaptadores enganchables**

Una clase es más reutilizable cuando se deja a un lado la suposición de que otras clases deben utilizarla, creando código genérico. Convirtiendo la adaptación de una interfaz en una clase, se elimina la suposición de que otras clases ven la misma interfaz.

Dicho de otra manera, la adaptación de la interfaz permite incorporar a la clase en sistemas existentes que pueden esperar diferentes interfaces de la misma.

**Identificación**

Adapter es reconocible por un constructor que toma una instancia de distinto tipo de clase abstracta/interfaz. Cuando el adaptador recibe una llamada a uno de sus métodos, convierte los parámetros al formato adecuado y después dirige la llamada a uno o varios métodos del objeto envuelto.

**Aplicación**

Adapter se aplica cuando se quiera usar la funcionalidad de una clase existente, pero su interfaz no es compatible.

Si creamos código reutilizable y no se desea vincularlo a una implementación en particular, se debe usar una interfaz de Adaptader sobre la cual dependa el código, para que a futuro los clientes pueden implementar su propia versión.

Permite que este siga mejor el *Principio Abierto/Cerrado*, que establece que los módulos deben estar abiertos a la extensión, pero cerrados a la modificación.

**Opinión Personal:**

El Patrón de Adapter es uno muy útil a la hora de realizar código genérico, permitiendo a este funcionar o comunicarse con cualquier objeto, la idea es programar sin pensar en quién o qué va a usar/implementar lo que estamos codeando, si no que se pueda adaptar a quien lo necesite. También es una muy buena opción a la hora de implementar funcionalidades nuevas a un sistema, porque nos permite agregar extensiones, sin modificar drásticamente nuestro código base o el de un tercero.

## Lista de referencias

https://endjin.com/blog/2020/10/design-patterns-in-csharp-the-adapter-pattern
https://refactoring.guru/es/design-patterns/adapter/csharp/example
https://codewithshadman.com/adapter-design-pattern-in-csharp/
https://the5coders.net/2020/10/18/el-patron-adapter/
https://www.dofactory.com/net/adapter-design-pattern
https://refactoring.guru/es/design-patterns/adapter
https://reactiveprogramming.io/blog/es/patrones-de-diseno/adapter
https://informaticapc.com/patrones-de-diseno/adapter.php