

Capítulo 10

El patrón de diseño Adapter

1. Descripción

El objetivo del patrón de diseño **Adapter** es convertir la interfaz de una clase existente en la interfaz esperada por los clientes para que puedan colaborar.

2. Ejemplo

El servidor web del sistema de venta de vehículos crea y administra los documentos destinados a los clientes. La interfaz **Documento** se ha definido para realizar esta gestión. La figura 10.1 muestra su representación UML así como los tres métodos **setContenido**, **dibujar** e **imprimir**. Se ha realizado un primer ejemplo de implementación de esta interfaz: la clase **DocumentoHtml** que concretiza estos tres métodos.

Por otro lado, la agregación de documentos PDF supone un problema, pues se trata de documentos más complejos de construir y de administrar que los documentos HTML. Para ello se ha escogido otro componente de terceros, aunque su interfaz no se corresponde con la interfaz **Documento**. Por lo tanto, va a haber que adaptar la una a la otra. La figura 10.1 muestra el componente **ComponentePdf** cuya interfaz incluye más métodos y la nomenclatura es bien diferente (con el prefijo pdf).

El patrón de diseño **Adapter** proporciona una solución que consiste en crear la clase **DocumentoPdf** que implemente la interfaz **Documento** y posea una asociación de tipo composición con una instancia de la clase **ComponentePdf**. La implementación de los tres métodos de la interfaz **Documento** consiste en delegar correctamente las llamadas a **ComponentePdf**. Esta solución se muestra en la figura 10.1, el código de los métodos se detalla con ayuda de notas.

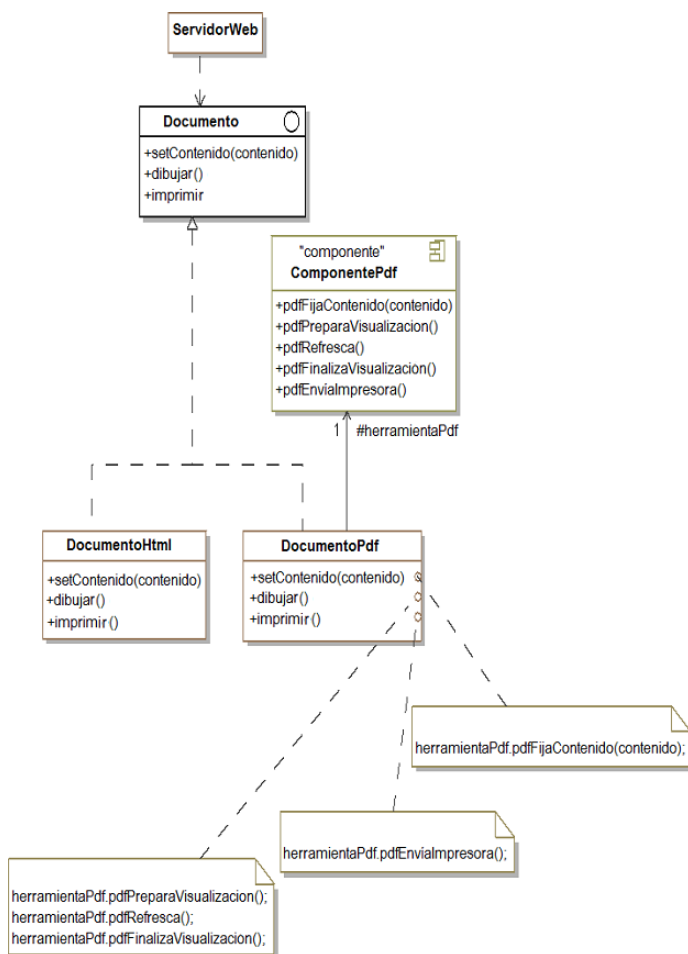


Figura 10.1 - El patrón de diseño **Adapter** aplicado a un componente de documentos PDF

3. Estructura

3.1 Diagrama de clases

La figura 10.2 detalla la estructura genérica del patrón de diseño **Adapter**.

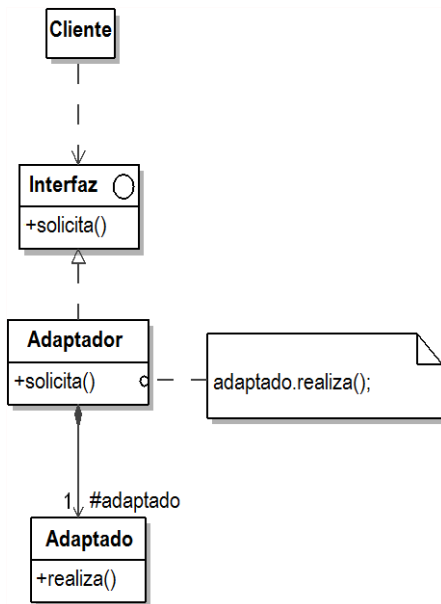


Figura 10.2 - Estructura del patrón de diseño **Adapter**

3.2 Participantes

Los participantes del patrón de diseño **Adapter** son los siguientes:

- **Interfaz** (**DocumentoInterfaz**) expone la firma de los métodos indispensables en todo documento.
- **Cliente** interactúa con los objetos que se adaptan a **Interfaz**.
- **Adaptador** (**DocumentoPdf**) implementa los métodos de **Interfaz** invocando por delegación los métodos del objeto adaptado.

- **Adaptado** (**ComponentePdf**) es la clase cuya interfaz ha sido adaptada para corresponder a lo que exige **Interfaz**.

3.3 Colaboraciones

El cliente invoca el método **solicitud** del adaptador que, en consecuencia, interactúa con el objeto adaptado invocando al método **realiza**. La figura 10.3 ilustra estas colaboraciones.

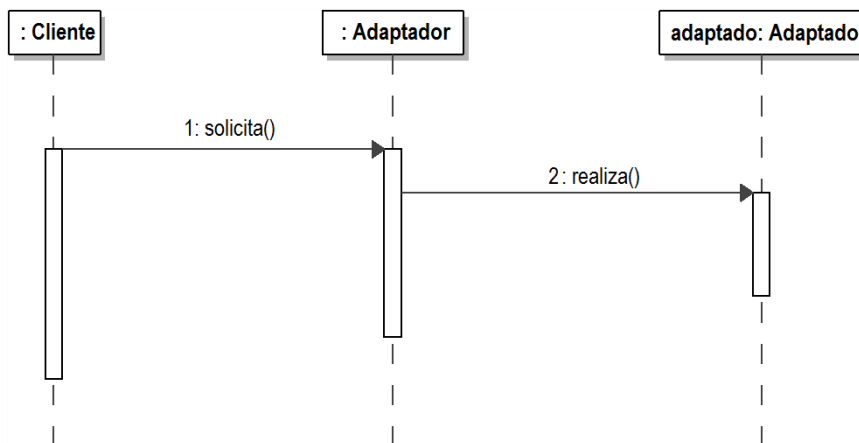


Figura 10.3 - Diagrama de secuencia del patrón de diseño **Adapter**

4. Dominios de aplicación

El patrón de diseño **Adapter** se utiliza en los siguientes casos:

- Para integrar en el sistema un objeto cuya interfaz no se corresponde con la interfaz requerida en el interior de este sistema.
- Para proveer interfaces múltiples a un objeto en su etapa de diseño.

5. Ejemplo en PHP

A continuación presentamos el código del ejemplo escrito en PHP. Empezamos por la interfaz **DocumentoInterfaz**:

```
<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\Adapter;

interface DocumentoInterfaz
{
    public function setContenido(string $contenido): void;

    public function dibuja(): void;

    public function imprime(): void;
}
```

La clase **DocumentoHtml** es el ejemplo de clase que implementa la interfaz **DocumentoInterfaz**.

```
<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\Adapter;

class DocumentoHtml implements DocumentoInterfaz
{
    private readonly string $contenido;

    public function setContenido(string $contenido): void
    {
        $this->contenido = $contenido;
    }

    public function dibuja(): void
    {
        echo "Dibuja documento HTML: $this->contenido" . PHP_EOL;
    }

    public function imprimir(): void
    {

```

```
        echo "Imprime documento HTML : $this->contenido" . PHP_EOL;
    }
}
```

La clase **ComponentePdf** representa el componente de terceros integrado en la aplicación. Su diseño es independiente de la aplicación y, en particular, de la interfaz **DocumentoInterfaz**. Esta clase tendrá que adaptarse a continuación.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Adapter;

class ComponentePdf
{
    private readonly string $contenido;

    public function pdfFijaContenido(string $contenido): void
    {
        $this->contenido = $contenido;
    }

    public function preparaVisualizacion(): void
    {
        echo 'Visualiza PDF: Comienzo' . PHP_EOL;
    }

    public function pdfRefresca(): void
    {
        echo "Visualiza contenido PDF: $this->contenido" . PHP_EOL;
    }

    public function pdfTerminaVisualizacion(): void
    {
        echo 'Visualiza PDF: Fin' . PHP_EOL;
    }

    public function pdfEnviaImpresora(): void
    {
        echo "Impresión PDF: $this->contenido" . PHP_EOL;
    }
}
```

La clase **DocumentoPdf** representa el adaptador. Está asociada a la clase **ComponentePdf** mediante el atributo **herramientaPdf** que contiene una referencia hacia el objeto adaptado.

Implementa la interfaz **DocumentoInterfaz** y cada uno de sus métodos invoca a los métodos necesarios del objeto adaptado para realizar la adaptación entre ambas interfaces.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Adapter;

class DocumentoPdf implements DocumentoInterfaz
{
    public function __construct(
        private readonly ComponentePdf $herramientaPdf = new
        ComponentePdf()
    ) {
    }

    public function __construct()
    {
        $this->herramientaPdf = new ComponentePdf();
    }

    public function setContenido(string $contenido): void
    {
        $this->herramientaPdf->pdfFijaContenido($contenido);
    }

    public function dibujar(): void
    {
        $this->herramientaPdf->preparaVisualizacion();
        $this->herramientaPdf->pdfRefresca();
        $this->herramientaPdf->pdfTerminaVisualizacion();
    }

    public function imprimir(): void
    {
    }
}
```

```
        $this->herramientaPdf->pdfEnviaImpresora();  
    }  
}
```

Ahora, pasamos al programa cliente; primero crea un documento HTML, fija el contenido y a continuación lo dibuja.

A continuación, realiza las mismas acciones con un documento PDF.

```
<?php  
  
declare(strict_types=1);  
  
namespace ENI\DesignPatterns\Adapter;  
  
require __DIR__ . '/../../../../../autoload.php';  
  
$documentoHtml = new DocumentoHtml();  
$documentoHtml->setContenido('Hello');  
$documentoHtml->dibujar();  
  
$documentoPdf = new DocumentoPdf();  
$documentoPdf->setContenido('Hola');  
$documentoPdf->dibujar();  
?>
```

La ejecución de este código cliente muestra en pantalla el resultado siguiente.

```
Dibuja documento HTML: Hello  
Visualiza PDF: Comienzo  
Visualiza contenido PDF: Hola  
Visualiza PDF: Fin
```