



PRÁCTICA 02 - PATRONES

PATRONES DE ESTRUCTURACIÓN

DESARROLLO RÁPIDO DE APLICACIONES

Por Juan José Camacho Hidalgo
UNIVERSIDAD DE ALMERÍA

Contenido

1.	Introducción	2
2.	Patrones de estructuración	2
2.1.	Adapter	2
2.1.1.	Funcionamiento (base: AdapterConsoleApp)	2
2.1.2.	Ejecución	3
2.1.3.	Ampliación	3
2.2.	Bridge	4
2.2.1.	Funcionamiento (base: BridgeConsoleApp)	4
2.2.2.	Ejecución	5
2.2.3.	Ampliación	5
2.3.	Composite	6
2.3.1.	Funcionamiento (base: CompositeConsoleApp)	6
2.3.2.	Ejecución	7
2.3.3.	Ampliación	7
2.4.	Decorator	8
2.4.1.	Funcionamiento (DecoratorConsoleApp)	8
2.4.2.	Ejecución	8
2.4.3.	Ampliación	8
2.5.	Facade	9
2.5.1.	Funcionamiento (FacadeConsoleApp)	9
2.5.2.	Ejecución	9
2.5.3.	Ampliación	10
2.6.	Flyweight	10
2.6.1.	Funcionamiento	10
2.6.2.	Ejecución	11
2.6.3.	Ampliación	11
2.7.	Proxy	12
2.7.1.	Funcionamiento	12
2.7.2.	Ejecución	12

1. Introducción

Este documento tiene como objetivo la explicación y justificación del uso de patrones de estructuración en la programación, mediante ejemplos claros escritos en C#.

2. Patrones de estructuración

Los patrones estructurales se enfocan en como las clases y objetos se componen para formar estructuras mayores, los patrones estructurales describen como las estructuras compuestas por clases crecen para crear nuevas funcionalidades de manera de agregar a la estructura flexibilidad y que la misma pueda cambiar en tiempo de ejecución lo cual es imposible con una composición de clases estáticas.

2.1. Adapter

2.1.1. Funcionamiento (base: AdapterConsoleApp)

Este patrón se encarga de convertir la interfaz Documento, en la esperada por los clientes existentes. En este caso, ServidorWeb actúa como cliente. Todos los tipos de documento implementan la interfaz Documento, y se encargan de “adaptar” los métodos definidos en la interfaz, a sus necesidades locales. Todos incluyen los métodos de dibujar y de imprimir. Pero cada clase se encarga de imprimir y de dibujar de una forma particular. Por ejemplo, los documentos pdf contienen ComponentePdf que se encarga de redefinir la forma de dibujar.

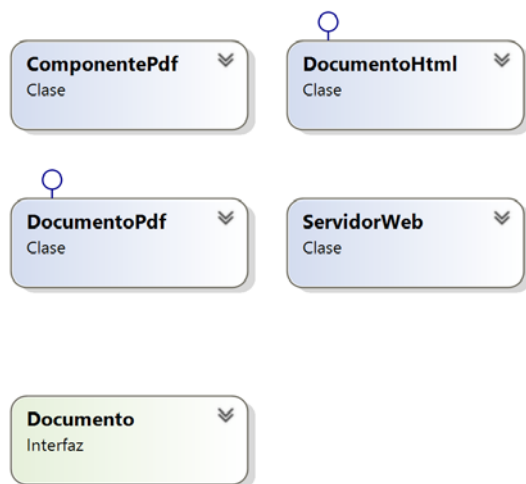


Diagrama de clases de AdapterConsoleApp.

2.1.2. Ejecución

Como hemos comentado anteriormente, el PDF dibuja de manera distinta que el HTML, pero ambos implementan los mismos métodos, solo que de diferente forma.

C:\WINDOWS\system32\cmd.exe

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

2.1.3. Ampliación

He añadido otro tipo de documento, en este caso, RTF, con un método de dibujar distinto, pero sin incorporar un componente nuevo (que podría hacerlo). En este caso, firma con una sonrisa al final. Mismo esquema de documento, distinta forma de dibujar.

C:\WINDOWS\system32\cmd.exe

```
Dibuja el documento HTML: Hola  
Imprime el documento HTML: Hola  
  
Visualiza PDF: Comienzo  
Visualiza contenido PDF: Hola  
Visualiza PDF: Fin  
Impresión PDF: Hola  
Dibuja el documento RTF: Hola  
Dibuja una sonrisa: :)  
Imprime el documento RTF: Hola
```



Diagrama de clases de *AdaptarConsoleApp* ampliado.

2.2. Bridge

2.2.1. Funcionamiento (base: BridgeConsoleApp)

Este patrón tiene como objetivo separar la implementación de un objeto de su representación e interfaz. Para ello, este ejemplo muestra una clase **Usuario** que controla el programa. El objetivo es rellenar un formulario, según la nacionalidad. Los formularios tienen una interfaz común, que define su estructura, pero pueden ser de distintos formatos. Los formatos (**FormAppletImpl** y **FormHtmlImpl**) evolucionan independientemente de los tipos de formularios adaptados a cada país. Entonces, cualquier nacionalidad puede disponer de cualquier tipo de formato, pero siempre con la restricción de límite de número de matrícula propia de ese país. Por tanto un objeto cualquiera puede cambiar su implementación en tiempo de ejecución, algo que puede ser útil si quieres cambiar el formato en un momento concreto, o incluso si se cambia la nacionalidad, pero con igual formato.

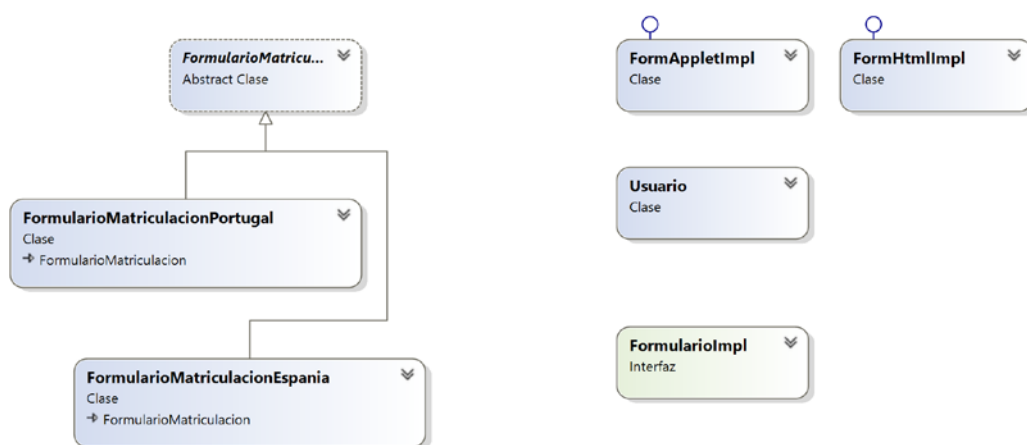



Diagrama de clases de *BridgeConsoleApp*.

2.2.2. Ejecución

Como vemos, se hace uso de una longitud 6 y después 7 de número de matrícula, correspondientes a España y Portugal. Además se hace uso de formato HTML y formato Applet.


 C:\WINDOWS\system32\cmd.exe

```
HTML: número de matrícula existente:
766312
HTML: Solicitud de matriculación
HTML: número de matrícula: 766312

Applet: número de matrícula existente:
1452345
Applet: Solicitud de matriculación
Applet: número de matrícula: 1452345
```

2.2.3. Ampliación

He realizado una ampliación creando un formato RTF, y una nacionalidad nueva, propia de Alemania. En esta ampliación, muestro como se han desarrollado de forma independiente, pero luego se ha creado una nacionalidad Alemana en el formulario, haciendo uso del formato nuevo, sin conflictos. En este caso el número de matrícula llega hasta 8.

 C:\WINDOWS\system32\cmd.exe

```
HTML: número de matrícula existente:
341242
HTML: Solicitud de matriculación
HTML: número de matrícula: 341242

Applet: número de matrícula existente:
1523534
Applet: Solicitud de matriculación
Applet: número de matrícula: 1523534
RTF: número de matrícula existente:
12525423
RTF: Solicitud de matriculación
RTF: número de matrícula: 12525423
```

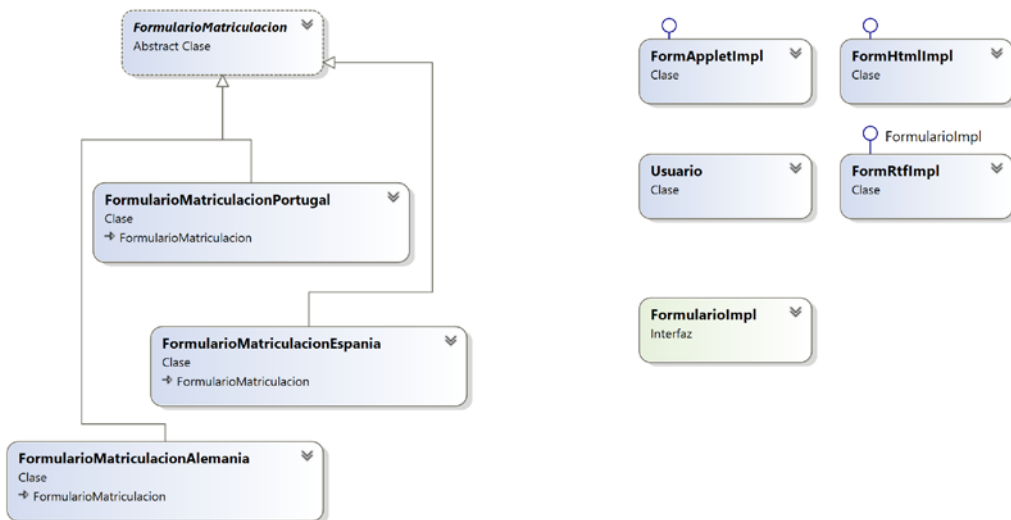


Diagrama de clases BridgeConsoleApp ampliado.

2.3. Composite

2.3.1. Funcionamiento (base: CompositeConsoleApp)

Este patrón se basa en componer objetos en estructuras jerárquicas, basado en un árbol. En este sentido vemos como tenemos dos niveles: EmpresaMadre, y por debajo, EmpresaSinFilial. Empresa es la clase abstracta que nos define los métodos y atributos comunes de todas las empresas. Esto nos permite realizar métodos de calculo obteniendo el conjunto de resultados obtenidos de todas las clases hijas en niveles a lo largo del árbol.

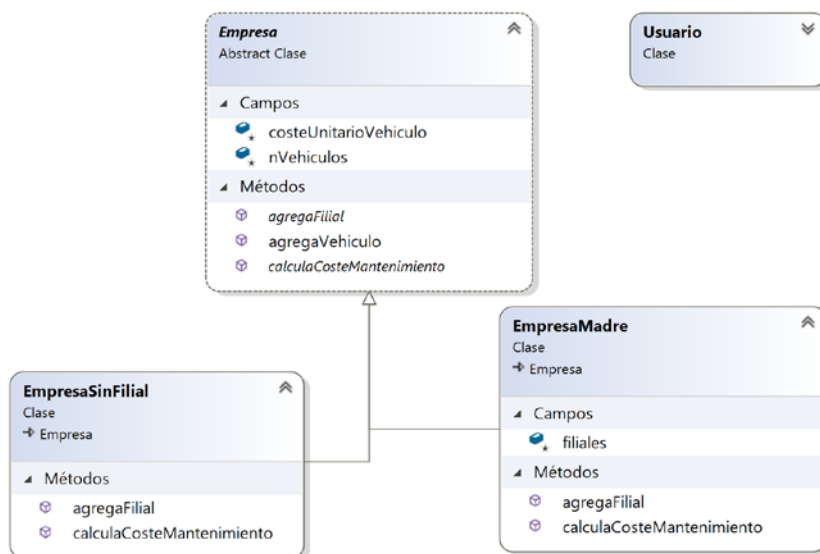


Diagrama de clases CompositeConsoleApp.

2.3.2. Ejecución

En nuestra ejecución, se calcula el mantenimiento del grupo de la EmpresaMadre, compuesta por vehículos y empresas filiales.

```
C:\WINDOWS\system32\cmd.exe
```

```
Coste de mantenimiento total del grupo: 20
```

2.3.3. Ampliación

Para comprobar como se usa este patrón, hemos ampliado el programa, creando un tipo más de empresa, EmpresaStartup, que se ubicaría en el mismo nivel que EmpresaSinFiliar. La EmpresaMadre hace uso de los métodos de todas las empresas startup y sin filial de las que son hijas lógicamente hablando. Se visualiza el coste aumentado, por la creación de una empresa startup.

```
C:\WINDOWS\system32\cmd.exe
```

```
Coste de mantenimiento total del grupo: 32
```

Aquí visualizamos el esquema nuevo resultante:

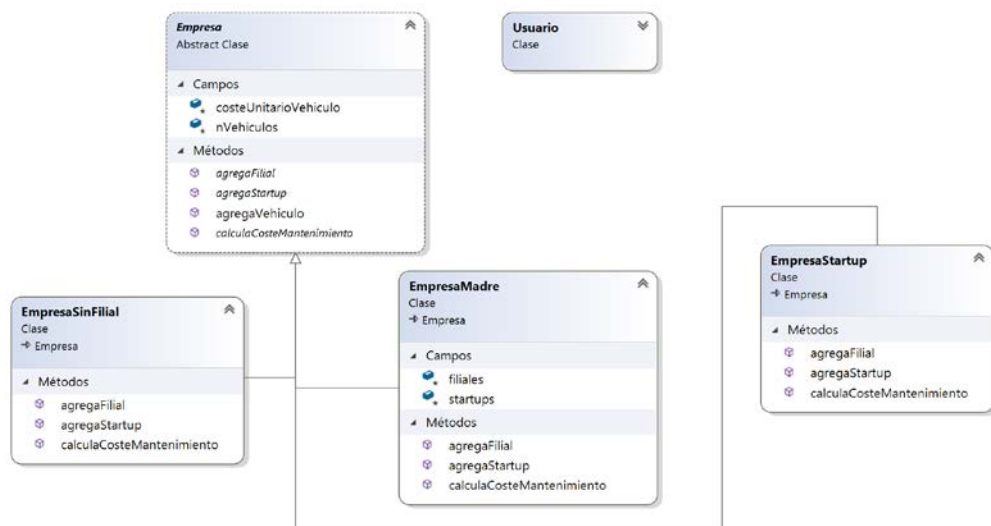
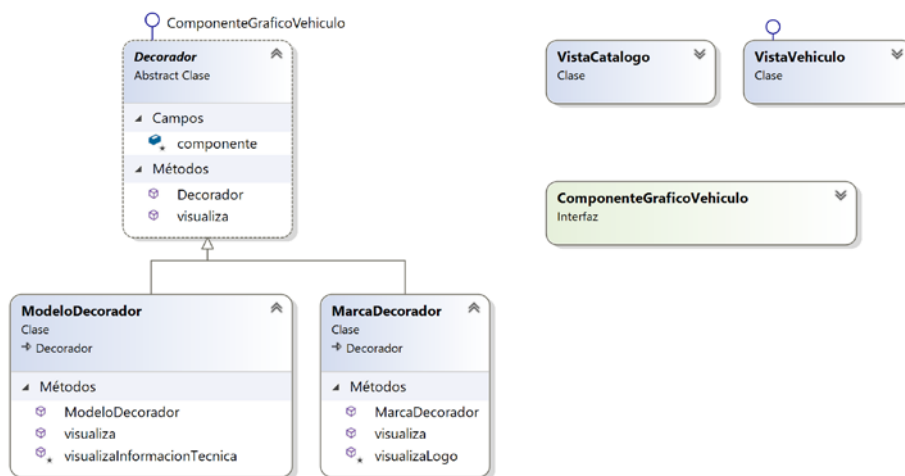


Diagrama de clases CompositeConsoleApp ampliado.

2.4. Decorator

2.4.1. Funcionamiento (DecoratorConsoleApp)

Disponemos de una interfaz, `ComponenteGraficoVehiculo`, que presenta el método `visualiza()`, que tienen que implementar todas las clases. También lo incorpora la `VistaVehículo`, que nos mostraría la visualización del vehículo. La `VistaCatalogo`, nos muestra el vehículo haciendo uso de esta clase, y mediante `Decorator`, nos muestra todos los datos del vehículo, por los que está compuesto. Así se asignan a este objeto dinámicamente todas las propiedades que debe contener, o responsabilidades. Es decir, el patrón se encarga de añadir funcionalidad a un objeto existente, en este caso, sobre un vehículo, a partir de la `VistaVehiculo`.



2.4.2. Ejecución

En esta captura vemos como la ejecución del programa nos muestra la vista del vehículo, el modelo y la marca. Unos a partir de otros.

`C:\WINDOWS\system32\cmd.exe`

```
Visualización del vehículo
Información técnica del modelo
Logotipo de la marca
```

2.4.3. Ampliación

Tendríamos que añadir otra clase, de la misma forma que `ModeloDecorador` o `MarcaDecorador`, podría ser tipos de neumáticos, y el `visualiza` incluiría información técnica de los neumáticos del fabricante del vehículo. A partir de la marca obtendríamos la información de los neumáticos.

2.5. Facade

2.5.1. Funcionamiento (FacadeConsoleApp)

Esta solución muestra tres interfaces. Una de ellas engloba las otras dos, proporcionando una única interfaz unificada. Disponemos de una interfaz `WebServiceAuto` que incorpora los métodos de las interfaces `Catalogo` y `GestionDocumento`, las cuales son implementadas por `ComponenteCatalogo` y `ComponenteGestionDocumento`, respectivamente. La interfaz `WebServiceAuto` es utilizada (implementada) por `WebServiceAutoImpl`, que es la clase que hace uso de la interfaz unificada y así crear una implementación que incluya toda la funcionalidad necesaria para los requisitos del usuario, que este caso, se trata de la clase `UsuarioWebService`.

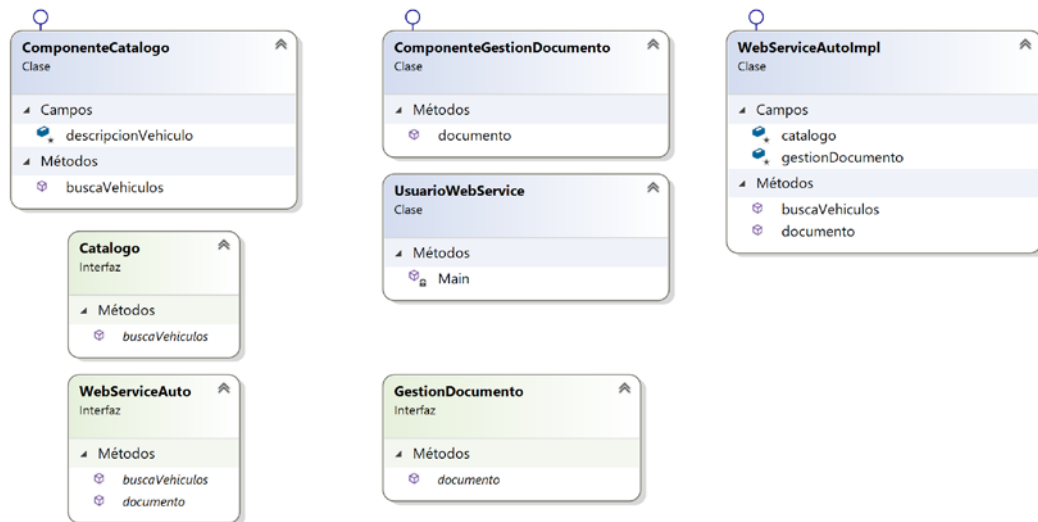


Diagrama de clases de FacadeConsoleApp.

2.5.2. Ejecución

Como vemos en la ejecución, el usuario busca vehículos, para obtener su información, así como el precio, además de los documentos que muestran esto.

`C:\WINDOWS\system32\cmd.exe`

```
Documento número 0
Documento número 1
Vehículo(s) cuyo precio está comprendido entre 5000 y 7000
  Berlina 5 puertas
  Break 5 puertas
  Utilitario 3 puertas
```

2.5.3. Ampliación

Una ampliación del sistema sería incluir una interfaz llamada por ejemplo tipoDePintura, y una clase ComponenteTipoDePintura, que implemente dicha interfaz. Habría que añadir el método buscaTipoDePinturaDisponible(), que se añadiría a las clases anteriormente mencionadas, además de a la interfaz WebServiceAuto. Por tanto la clase que la implementa también debería de contenerlo.

Así de esta forma, añadiríamos más funcionalidad, manteniendo el mismo punto de entrada (fachada).

2.6. Flyweight

2.6.1. Funcionamiento (base: FlyweightConsoleApp)

En este caso la clase Client, solicita vehículos (VehiculoSolicitado). La clase VehiculoSolicitado, hace uso de FabricaOpcion para mostrar cualquier característica añadida al coche, mediante la clase OpcionVehiculo, que define cada característica de forma única. Por ejemplo, podemos añadir la inclusión de air bag, o dirección asistida como opciones. Y para generar cada una de ellas se hace uso de FabricaOpcion mediante VehiculoSolicitado. Esto nos permite la reducción de número de instancias, ya que separa las partes del vehículo que son comunes (estado intrínseco), en este caso la disposición de opciones, nombre, descripción, y el precio de venta, de la parte única de cada vehículo, es decir, de cada opción o característica (estado extrínseco).



Diagrama de clases de FlyweightConsoleApp.

2.6.2. Ejecución

C:\WINDOWS\system32\cmd.exe

```
Opción
Nombre: air bag
Descripción de air bag
Precio estándar: 100
Precio de venta: 80

Opción
Nombre: dirección asistida
Descripción de dirección asistida
Precio estándar: 100
Precio de venta: 90

Opción
Nombre: elevalunas eléctricos
Descripción de elevalunas eléctricos
Precio estándar: 100
Precio de venta: 85
```

2.6.3. Ampliación

Podemos añadir muchas más características, y seguiría manteniéndose la parte intrínseca, para cada vehículo, cambiando la extrínseca. Vamos a añadir dos características más:

C:\WINDOWS\system32\cmd.exe

```
Nombre: air bag
Descripción de air bag
Precio estándar: 100
Precio de venta: 80

Opción
Nombre: dirección asistida
Descripción de dirección asistida
Precio estándar: 100
Precio de venta: 90

Opción
Nombre: elevalunas eléctricos
Descripción de elevalunas eléctricos
Precio estándar: 100
Precio de venta: 85

Opción
Nombre: cambios automáticos
Descripción de cambios automáticos
Precio estándar: 100
Precio de venta: 75

Opción
Nombre: puertas automáticas
Descripción de puertas automáticas
Precio estándar: 100
Precio de venta: 105
```

2.7. Proxy

2.7.1. Funcionamiento (base: ProxyConsoleApp)

Se define una interfaz Animación para la vista de un vehículo, y una clase Video. AnimacionProxy implementa Animación, y redefine la animación. Normalmente, mostraría un objeto de tipo Video, pero se crea AnimacionProxy, que tiene como objetivo sustituir ese Video. Se crea un sustituto de Video, que sería una foto. Si no hay Video, dibuja una foto. Así se controla el acceso a Video, actuando el patrón como método de mediación.

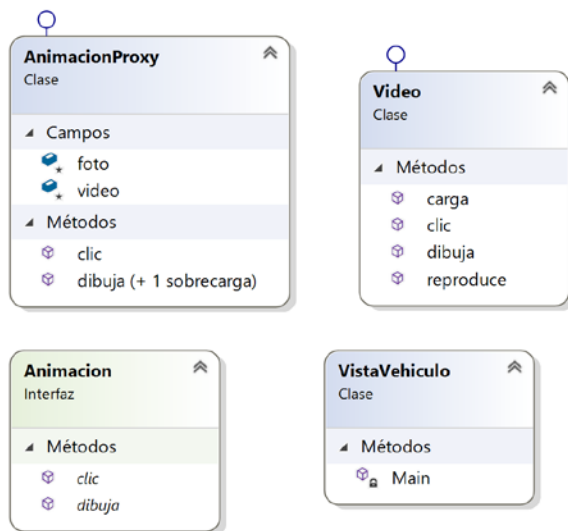


Diagrama de clases de ProxyConsoleApp.

2.7.2. Ejecución

Muestra una foto, y luego sustituye mediante el “proxy” en la animación, y carga, reproduce y muestra un video.

```
C:\WINDOWS\system32\cmd.exe
```

```
mostrar la foto
Cargar el vídeo
Reproducir el vídeo
Mostrar el vídeo
```