Práctica 03 - Patrón Visitor

Los ladrones de cuerpo [1]

[1] En homenaje al clásico de la ciencia ficción *La invasión de los ladrones de cuerpos* (Don Siegel, 1956), donde unas extrañas criaturas se introducen en los cuerpos de los habitantes de un pequeño pueblo estadounidense, los cuales mantienen su apariencia externa pero cambian su comportamiento.

Introducción

Desde el punto de vista de la calidad externa, los productos software creados mediante la utilización de patrones de diseño no se distinguen de los creados sin su utilización. Por ejemplo, supongamos que se creasen dos productos software con exactamente la misma funcionalidad, pero donde un producto se ha creado utilizando patrones de diseño y el otro producto sin ellos. En este caso, ambos productos ofrecerían unas funcionalidades y rendimiento similares. En adelante denominaremos al producto que utiliza los patrones de diseño como el *producto patronizado* y al que los utiliza como el *producto sin patronizar*.

Estos productos no serían sólo iguales desde un punto de visto externo, sino que incluso, en algunas situaciones muy particulares [2], el producto patronizado podría ser ligeramente más lento o consumir más memoria que el producto sin patronizar. Es decir, desde este punto de vista del rendimiento, el producto patronizado se podría considerar peor que el producto que sin patronizar.

Por tanto, si, desde el punto de vista de la calidad externa, ambos productos son prácticamente indistinguibles y la utilización de los patrones de diseño sólo parece servir para empeorar el producto, ¿qué ventaja aportan estos patrones? La principal ventaja que suelen aportar está relacionada con la calidad interna de un producto software y, más concretamente, con su facilidad de evolución y adaptación al cambio. Para ello los patrones de diseño suelen introducir en un producto software *puntos de variación* bien definidos que permiten introducir nuevas funcionalidades, denominadas *variantes*, en dicho producto de manera cómoda, sencilla y elegante, sin necesidad de modificar el producto ya existente.

Por tanto, volviendo a nuestro ejemplo, aunque externamente el producto patronizado pueda parecer exactamente igual que el producto sin patronizar, el producto patronizado es en realidad mucho más fácil de adaptar a nuevas situaciones. Dicho de otra forma, el producto patronizado es mucho más barato de mantener.

El objetivo de esta práctica es que el alumno entienda cómo y por qué los patrones de diseño ayudan a mejorar la calidad interna de un producto software. Para alcanzar este objetivo, el alumno deberá aplicar el *patrón Visitor* a un problema concreto. Siguiendo los principios del *patrón Visitor* podemos incorporar nuevas funcionalidades a una jerarquía o estructura de clases ya existente, sin necesidad de tener que modificar dichas clases. Sin embargo, para aplicar el *patrón Visitor* debemos utilizar de un complejo y exótico mecanismo de *double dispatching*, el puede ser complejo de entender e introduce además una serie de niveles de indirección que teoricamente ralentizaría la ejecución de la aplicación [2]. Para alcanzar este objetivo general, el alumno deberá satisfacer los subobjetivos que se detallan en la siguiente sección.

[2] *(1, 2)* Dadas las características del hardware actual, este fenómeno es prácticamente inapreciable en prácticamente el 100% de las situaciones.

Objetivos

Los objetivos concretos de esta práctica son:

1. Comprender el funcionamiento del *patrón Visitor*.
2. Comprender el concepto de punto de variación y variante.
3. Ser capaz de, utilizando el *patrón Visitor*, introducir puntos de variación dentro de una estructura de clases de manera que se puedan incorporar nuevas funcionalidades a dicha estructura sin necesidad de modificarla.
4. Ser capaz de, utilizando el *patrón Visitor*, extender la funcionalidad de una estructura visitable de clases mediante la implementación de nuevas variantes.
5. Ser capaz de utilizar el *patrón Visitor* para visitar un elemento cualesquiera de una jerarquía de clases.

Para alcanzar dichos objetivos, el alumno deberá aplicar el *patrón Visitor* al problema que se describe a continuación.

Visualización del Sistema de Archivos Sparrow

El Sistema de Archivos Sparrow debe permitir visualizar su estructura en forma arbórea conforme a varios formatos. Actualmente se consideran sólo dos formatos de visualización distintos, conocidos como el *formato compacto* y el *formato extendido*. No obstante, desea que pero se tiene previsto en un futuro cercano añadir más variantes, por lo que se desea que la aplicación sea facilmente extendible a este respecto.

Los *formatos compacto* y *extendido* que se desean implementar comparten una serie de características comunes y una serie de diferencias. Con respecto a sus aspectos comunes, ambos fomatos deben:

- Mostrar un único elemento por línea.
- Mostar el nombre de cada elemento.
- Mostrar el contenido de los directorios debajo de cada directorio, convenientemente tabulado con respecto al nombre del directorio.

Además, el en caso del *formato extendido* se deberá:

- Mostrar el contenido de los archivos comprimidos de la misma forma que los directorios.
- Mostrar delante del nombre una letra que especifique el tipo de elemento del que se trata, de acuerdo con la siguiente tabla.

`d`	Directorio
`e`	Enlace Directo
`c`	Archivo Comprimido
`f`	Fichero Básico

La siguiente figura muestra un ejemplo de visualización en el formato extendido.

Ejemplo de Sistema de Archivos Sparrow¶

```
d Raiz
   d Directorio Vacio
   d Directorio Con Archivo Unico
      f foto001.jpg
   d Directorio Con Archivo Comprimido Simple
      f foto002.jpg
      e foto001.jpg
      c ccSimple.zip
         d Directorio Vacio En Archivo Comprimido
         f foto003.jpg
         e foto001.jpg
   d Directorio con Directorio Anidado
      f foto004.jpg
      e ccSimple.zip
      e Directorio Vacio
      d Directorio con Archivo Comprimido Complejo
         f foto005
         f foto006
         c ccComplejo.zip
            c ccAnidada.zip
               f foto007.jpg
            f foto008.jpg
```

Actividades

El alumno, para poder alcanzar los objetivos perseguidos, deberá completar las siguientes actividades:

1. Hacer que jerarquía de clases creada en la práctica dedicada al patrón Composite sea visitable por las funciones de impresión descritas. Para hacer que la jerarquía sea visitable, el alumno deberá crear un *visitante abstracto* y añadir a la jerarquía los *métodos para aceptar visitantes*, con su correspondiente implementación.

Nota

Las funciones de impresión deberán serializar un sistema de archivos como una cadena de caracteres, de manera que dicha cadena se pueda imprimir en diferentes salidas, tales como un monitor o una impresora. Por tanto, las funciones de impresión o visualización deben retornar una cadena de caracteres.

2. Implementar un *visitante concreto* para el *formato compacto*, sin tener en cuenta la necesidad de tabular el contenido de los directorios.
3. Crear un programa de prueba que verifique el correcto funcionamiento del visitante concreto implementado.

4. Implementar un visitante concreto para el *formato extendido*, sin tener en cuenta la necesidad de tabular el contenido de los directorios y archivos comprimidos.
5. Crear un programa de prueba que verifique el correcto funcionamiento del visitante concreto implementado.
6. Crear una clase `SparrowView` que represente una especie de cuadro de texto para visualizar el contenido de un sistema de archivos sparrow. Dicha clase `SparrowView` tendrá un método `mostrarSistemaSparrow`, que acepte como parámetro cualquier elemento de un sistema Sparrow y lo muestre por pantalla, en formato extendido, entre dos líneas con algún adorno. Una posible implementación de este método se muestra a continuación:

```
Console.Out.WriteLine("======= Sparrow Viewer =======");
Console.Out.WriteLine();
// Imprimir Sistema Sparrow aquí
Console.Out.WriteLine();
Console.Out.WriteLine("==============================");
```

7. Modificar los visitantes concretos implementados para que tengan en cuenta el requisito relativo a la tabulación de los elementos anidados.
8. Con los programas de prueba implementados anteriormente, verificar el correcto funcionamiento de las modificaciones realizadas para soportar las tabulaciones.

Criterios de Autoevaluación

Para verificar que el patrón Visitor ha sido aplicado correctamente, se aconseja verificar que:

- Sólo se ha modificado la jerarquía de clases para añadir los métodos *accept*.
- La jerarquía de clases está libre de atributos relacionados con las funciones de impresión.
- Los métodos *accept* sólo delegan en los vistantes concretos.
- Los métodos *visit* de los visitantes concretos no invocan directamente a otros métodos *visit*, sino que delegan en los métodos *accept* que correspondan.
- Las implementaciones de los métodos *accept* y *visit* está libre de *castings*
- Se pueden añadir nuevos visitantes concretos sin modificar la jararquía de clases.