Práctica 02 - Patrón Composite

La parte contratante de la primera parte [1]

[1] En homenaje a la famosa escena de la película de los hermanos Marx *Una Noche en la Ópera* (Sam Wood, 1935).

Introducción

Los *Patrones de Diseño*, tal como se comentó en la práctica anterior, constituyen uno de los principales avances realizados dentro del campo de la Ingeniería del Software en las últimas décadas. Actualmente, la mayoría de los productos software desarrollados por profesionales, tanto de mediana como de larga escala, hacen uso de un buen número de patrones de diseño.

La principal bondad de los patrones de diseño es que proporcionan soluciones elegantes, sencillas y efectivas a problemas recurrentes del desarrollo de sistemas software. Es precisamente la recurrencia de los problemas que resuelven la razón por la cual estos patrones aparecen tan frecuentemente en el desarrollo de sistemas software.

No obstante, un patrón de diseño no genera una porción de código reutilizable como un elemento de caja negra. De hecho, resulta bastante complejo encapsular un patrón de diseño en un conjunto de clases que puedan ser reutilizables en aplicaciones de diferentes dominios. Lo que proporciona un patrón es un conjunto de reglas que, aplicadas con inteligencia, permiten dar solución a un problema recurrente y bien conocido.

Por tanto, para aplicar un patrón de diseño a un problema determinado necesitaremos adaptar la idea que el patrón nos proporciona a las peculiaridades concretas de dicho problema. De esta forma, se distinguen dos habilidades distintas a la hora de utilizar patrones de diseño para el desarrollo de sistemas software.

La *primera habilidad* consistiría en, dado un sistema concreto a contruir, identificar qué problemas presenta y ser capaz de discernir si existe algún patrón de diseño, dentro de los actualmente disponibles, que pueda ayudar a resolver cada problema. Por ejemplo, un buen Ingeniero Informático debería saber que, ante la necesidad de implementar una herencia múltiple en Java, la aplicación del patrón mixin puede aportar una solución adecuada a dicho problema.

Una vez identificado un patrón a aplicar dentro de un sistema concreto, la *segunda habilidad* consistiría en ser capaz de adaptar la idea propuesta por el patrón a las particularidades de dicho sistema. Es decir, ser capaz de *instanciar* el patrón de un contexto determinado.

A lo largo de este curso el alumno realizará diferentes prácticas relacionadas con la segunda de estas habilidades, la intanciación de un patrón ya identificado dentro de un contexto concreto. La *primera habilidad* queda relegada a estudios superiores.

Dentro de dicho objetivo general, el objetivo concreto de esta práctica es que el alumno aprenda a aplicar correctamente el patrón Composite. Para ello, el alumno deberá satisfacer los subojetivos que se detallan en la siguiente sección.

Objetivos

Los objetivos concretos que se persiguen con la realización de esta práctica son:

1. Consolidar los conocimientos del alumno sobre elaboración de diagramas de clases.
2. Aprender a trabajar con clases abstractas o interfaces.
3. Aprender a aplicar correctamente el patrón Composite
4. Consolidar los buenos hábitos y buenas prácticas de programación.

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

Sistema de Archivos Sparrow

Dentro de un proyecto de mayor envergadura, denominado *Captain Sparrow*, se necesita implementar un sistema de archivos que contenga los siguientes elementos:

Archivos:

Constituyen los elementos básicos del sistema de archivos. Se corresponden con

documentos, fotografías o cualquier otro elemento que se deseen almacenar en el sistema. Cada archivo tiene un nombre y un tamaño en kilobytes. El sistema no distinguirá entre diferentes tipos de arhivos en función de su contenido.

Directorios:

Son contenedores de archivos u otros elementos que se puedan alojar el sistema de archivos Sparrow. Sirven para organizar y estructurar el sistema de manera que resulte más fácil localizar la información conforme al criterio que establezca cada usuario. Cada directorio tiene un nombre y su definición consume 1 Kb de almacenamiento en el sistema. Merece la pena destacar que un directorio puede contener a su vez más directorios dentro.

Archivos Comprimidos:

Son archivos especiales que almacenan en su interior elementos del sistema de archivos de manera comprimida. Cada archivo comprimido, al igual que los archivos normales, tiene un nombre. El sistema de compresión utilizado en el sistema Sparrow asegura que el tamaño del archivo comprimido sea siempre un 30% del tamaño total de los elementos que contiene, más 1Kb por la definición del archivo en sí.

Enlaces Directos:

Son elementos que dirigen al usuario de forma directa a otro elemento del sistema de archivos. El elemento al cual dirige un enlace lo denominaremos elemento destino. El elemento destino de un acceso directo puede ser cualquier elemento del sistema de archivos, menos otro acceso directo. Un acceso directo no tendrá nombre propio y utilizará como nombre el nombre de su elemento destino. Si se renombra un enlace, deberá renombrarse también su elemento destino. Cada enlace directo ocupa en el sistema 1 Kb de espacio de almacenamiento.

Cada elemento del sistema de archivos Sparrow debe permitir obtener de manera cómoda los siguientes datos:

- El tamaño total que ocupa dicho elemento el sistema de archivos. En este caso, hay que aclarar que el tamaño de un directorio se considera la suma del tamaño de la definición del directorio (1 Kb) más el tamaño de todos los elementos que contiene.
- El número total de archivos que contiene dicho elemento. En este caso, los archivos comprimidos cuentan como uno con independencia de los elementos que contengan y los enlaces no cuentan como archivos.

Para aplicar el patrón Composite a este problema, el alumno deberá realizar las actividades que se detallan a continuación.

Actividades

El alumno deberá realizar las siguientes actividades para poder aplicar el patrón Composite correctamente al problema anteriormente descrito:

1. Diseñar una jerarquía de clases conforme al patrón Composite que permita crear sistemas de archivos de profundidad no acotada y tratar dichos sistemas de manera homogénea con independencia de su profundidad. En este primer paso se excluirán del sistema los enlaces directos.
2. Representar dicha jerarquía en una diagrama de clases UML [2].
3. Implementar en C# la jerarquía de clases creada.
4. Implementar las operaciones necesarias para el correcto cómputo de la propiedad `número de archivos` [3].
5. Diseñar e implementar los casos de prueba necesarios para comprobar el correcto funcionamiento de los cómputos creados en el punto anterior.
6. Diseñar e implementar las operaciones necesarias para el cómputo de la propiedad `tamaño` [3].
7. Diseñar e implementar los casos de prueba necesarios para comprobar el correcto funcionamiento de los cómputos creados en el punto anterior.
8. Incorporar los enlaces directos al diseño creado, modificando tanto el diagrama UML como la implementación hasta ahora realizadas. El diseño deberá asegurar, sin necesidad de realizar *castings* o comprobaciones manuales de tipo, que no se puedan crear enlaces a enlaces [4].
9. Comprobar que todos los casos de prueba creados con anterioridad siguen funcionando correctamente bajo la presencia de enlaces.
10. Diseñar e implementar los casos de prueba necesarios para comprobar el correcto funcionamiento de los enlaces.

[2] Para crear el diagrama UML se puede utilizar tanto una herramienta profesional tipo MagicDraw como lápiz y papel. En caso de utilizar lápiz y papel, el alumno deberá escenear adecuadamente el documento y añadirlo a la práctica. Para la utilización de herramientas UML se recuerda que existen licencias académicas de MagicDraw a disposición de los alumnos, aunque cada alumno es libre de utilizar la herramenta UML que más sea de su agrado.

[3] *(1, 2)* No es necesario utilizar las propiedades de C# para implementar la práctica, pudiéndose utilizar los tradicionales *getters* y *setters* de Java. No obstante, se recomienda a todos los alumnos explorar y aprender a trabajar con el concepto de propiedad.

[4] Durante la incorporación de los enlaces, recordar que las clases abstractas y las interfaces se usan frecuentemente para representar *ables*, es decir, conjuntos de clases que comparten una cierta propiedad abstracta, como la de ser *comparable*, *serializable*, *ejectuable* o *cacheable*.

Criterios de Autoevaluación

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

1. Se pueden crear sistemas de archivos de cualquier profundidad.
2. Dentro de cada nivel puede haber elementos de diferente tipo.
3. Desde un punto de vista externo a la jerarquía, para ciertas operaciones, se manipulan igual sistemas de archivos multinivel que ficheros simples.
4. Los métodos para manipular los distintos elementos dentro la jerarquía tratan de manera homogénea a los diferentes elementos del sistema de archivos *Sparrow*, no haciendo distinciones entre los elementos en función de su tipo. Si este punto se cumple, los métodos que calculan el número de archivos y el tamaño de cada elementos deberían estar libres de *castings*.
5. Se puede añadir un nuevo elemento al sistema Sparrow, como `archivo encriptado`, sin tener que modificar ninguno de los elementos ya existentes en la jerarquía.

Para comprobar parte de los puntos 1 y 2, se aconseja al alumno verificar que su diseño permite la creación de un sistema de archivos como el que se muestra en la Figura 1[#f4]_.

Figura 1. 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
```

[5] La letra al lado de cada elemento indica el tipo de elemento del cual se trata: `d` es directorio, `c` es archivo comprimido, `f` es archivo y `e` es enlace directo.

Para la comprobación del punto 3 se aconseja crear una función estática `imprimirPropiedadesElementoSparrow` que acepte cualquier elemento de un sistema de archivos Sparrow e imprima por pantalla su `nombre`, `número de archivos` y `tamaño`. Dicha función deberá funcionar igual con independencia del elemento que se le pase y podría tener una implementación parecida a la que se muestra en la Figura 2, donde `e` sería un elemento cualesquiera del sistema de archivos Sparrow.

Figura 2. Ejemplo de cuerpo para el método `imprimirPropiedadesElementoSparrow` ¶

```
Console.Out.WriteLine("============= Info ==============");
Console.Out.WriteLine();
Console.Out.WriteLine("Nombre        : " + e.Nombre);
Console.Out.WriteLine("Tamaño        : " + e.Tamanho);
Console.Out.WriteLine("Num. Archivos : " + e.NumArchivos);
```

Por último, se aconseja verificar que el modelo UML creado no contiene errores de sintaxis triviales, como la ausencia de nombre y multiplicidad en los extremos navegables de una asociación.