PRÁCTICA 2: LA PARTE CONTRANTE DE LA PRIMERA PARTE[1]

1. Introducción

Los *Patrones de Diseño*, tal como se comentó en la práctica anterior, constituyen uno de los principales avances realizados dentro del campos de la *Ingeniería del Software* en las últimas décadas. Actualmente, la mayoría de los productos software, tanto de mediana como de larga escala, desarrollados por profesionales 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 porque los problemas que resuelven son recurrentes la razón por la cual los patrones de diseño aparecen tan frecuentemente en el desarrollo de sistemas software de diferente ámbito y dominio.

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. Por tanto, para aplicar un patrón de diseño a un problema determinado necesitaremos adaptar la idea que este 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 problema concreto, ser capaz de discernir si existe algún patrón de diseño que pueda ayudar a resolver dicho 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 ser una solución.

Una vez identificado el patrón a aplicar en un determinado contexto, la segunda habilidad consistiría en ser capaz de adaptar la idea propuesta por el patrón a las particularidades del problema a resolver de manera correcta.

A lo largo de este curso el alumno realizará diferentes prácticas relacionadas con la segunda de estas habilidades, la adaptación de un patrón concreto a un problema determinado. No os por tanto responsabilidad del alumno identificar el patrón a aplicar en cada práctica. Dicha habilidad queda relegada a cursos de estudios superiores.

Concretamente, el objetivo de la presente práctica es que el alumno aprenda a aplicar correctamente el patrón *Composite*. El patrón *Composite* es un patrón de diseño utilizado frecuentemente para crear jerarquías de objetos heterogéneos de profundidad arbitraria. A pesar de estar compuestas por objetos heterogéneos, se desea tratar a los elementos de esta jerarquía de la forma más homogénea posible.

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

Este objetivo concreto se descompone en los subobjetivos que se describen en el siguiente apartado.

2. Objetivos

Los objetivos concretos 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 conocimientos sobre el diseño y ejecución de pruebas software.
(5) 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.

3. 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:

(1) *Archivos*: Constituyen los elementos básicos del sistema de archivos. Se corresponden con documentos, fotografías u otros elementos que se deseen almacenar en el sistema. Cada archivo tiene un nombre y un tamaño en kilobytes.
(2) *Directorios*: Son contenedores de archivos (u otros elementos que pueda alojar el sistema de archivos). 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.
(3) *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.
(4) *Enlace Directo*: 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. Por tanto, un enlace directo no debe poder renombrarse. Un enlace directo ocupará en el sistema 1 Kb de espacio de almacenamiento.

Además, cada elemento del sistema de archivos *Sparrow* debe permitir calcular de manera cómoda los siguientes datos:

1. El tamaño total que ocupa dicho elemento. 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.
2. El número total de archivos que contiene dicho elemento. En este caso, los archivos comprimidos cuentan como uno con independencia de los elementos que contienen y los enlaces no cuentan como archivos.

Para poder computar estos datos, los diferentes elementos del sistema de archivos deberán implementar las operaciones correspondientes.

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

4. Actividades

El alumno deberá completar las siguientes actividades para la aplicación del patrón *Composite*:

1. Crear un diagrama de clases UML que ilustre la jerarquía de clases creadas conforme al patrón *Composite*, excluyendo por el momento los enlaces directos de dicha jerarquía.
2. Implementar en C# el diseño creado.
3. Implementar las operaciones necesarias para el cómputo del tamaño de un sistema de archivos.
4. Implementar los casos de prueba necesarios para comprobar la corrección la operación implementada en el punto anterior.
5. Implementar las operaciones necesarias para el cómputo del número de archivos.
6. Implementar los casos de prueba necesarios para comprobar la corrección de la operación implementada en el punto anterior.
7. Incorporar los enlaces directos al diseño creado hasta el momento, modificando tanto el diagrama UML como la implementación hasta ahora realizada. El diseño deberá asegurar, sin necesidad de realizar *castings* o comprobaciones manuales de tipo, que no se puedan crear enlaces a enlaces.
8. Implementar los casos de prueba para para comprobar que las operaciones para el cálculo del tamaño y del número de elementos siguen comportándose correctamente ante la presencia de enlaces directos en el sistema de archivos.

Durante la realización del punto 7, 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 planificable.

5. Criterios de Evaluación y Aclaraciones

La práctica se entregará a través de la plataforma *Moodle* siguiendo las instrucciones en ella proporcionadas. Las entregas fuera de la fecha o el formato establecido tendrán una calificación automática de cero.

La práctica se calificará atendiendo a los siguientes criterios de evaluación:

1. Corrección de la jerarquía de clases creada, incluyendo tanto diseño como implementación, como consecuencia de la aplicación de patrón *Composite* al sistema de archivos *Sparrow*, sin considerar los enlaces directos. En concreto, se verificará que:
    a. Se pueden representar sistemas de archivos de diferentes niveles (0.5 puntos);
    b. No existe limitación al número de niveles que se pueden crear (0.25 puntos);
    c. Dentro de cada nivel puede haber elementos de diferente tipo (0.5 puntos);
    d. No se puedan añadir elementos a elementos que no son contenedores, como, por ejemplo, los archivos (1 punto);
    e. Los métodos para añadir y eliminar elementos a un contenedor son homogéneos para los diferentes contenedores del sistema de archivos *Sparrow* (0.75 puntos).
2. Correcta implementación del método para calcular el tamaño total de un sistema de archivos, sin considerar los enlaces (1 punto).
3. Corrección de las pruebas creadas para el método del punto anterior (0.5 puntos).
4. Correcta implementación del método para calcular el número de archivos de un sistema de archivos, sin considerar los enlaces (1 punto).
5. Corrección de las pruebas creadas para el método del punto anterior (0.5 puntos).
6. Correcta incorporación de los enlaces a la jerarquía anteriormente creada (2 puntos).
7. Corrección de las pruebas creadas para comprobar el correcto comportamiento del sistema de archivos ante la presencia de enlaces directos (0.5 puntos).
8. Cumplimiento de la *Guía de Buenas Prácticas en Programación* (1 punto).

Los errores de notación que se detecten en el modelo UML, como la ausencia de nombre y multiplicidad en los extremos navegables de una asociación, tendrán una penalización, a aplicar en la calificación final de la práctica, de 0.25 puntos por error detectado. La penalización máxima a aplicar por este concepto será de dos puntos.

Si el diseño y la implementación creadas satisficiesen los criterios de evaluación proporcionados, pero no fuese conforme al patrón *Composite*, la calificación de la práctica sería automáticamente de cero.

Pablo Sánchez Barreiro