PRÁCTICA 4: TIEMPOS MODERNOS[1]

1. Introducción

Los principios de *indirección* e *inversión de dependencias* establecen que cuando un módulo A depende de otro módulo B, dicho módulo A sólo debería depender de los detalles de alto nivel de dicho módulo B, permaneciendo sus detalles de bajo nivel de ocultos. De esta forma, variaciones o cambios realizados en los elementos de bajo nivel del módulo B no deberían afectar al módulo A.

Por ejemplo, supongamos que tenemos una clase *Foo* que utiliza una clase *Date* para manipular fechas. Una fecha podría implementarse, al menos, de dos maneras completamente distintas: (1) almacenando los tres valores separados que constituyen la fecha, es decir, el día, el mes y el año; o, (2) almacenando el número de días transcurridos desde una fecha determinada, como, por ejemplo, el 1 de enero de 1970.

La primera alternativa permitiría visualizar la fecha por un dispositivo de salida con mayor comodidad. Por el otro lado, la segunda representación permite calcular con mayor comodidad el número de días transcurridos desde una fecha determinada. Este cálculo se realiza comúnmente para determinar, por ejemplo, si un determinado producto ha caducado o si una oferta concreta ha vencido. Por tanto, dependiendo del uso que la clase *Foo* vaya a realizar de la clase *Date*, una implementación resultaría más adecuada que otra.

En el caso ideal, un módulo *Foo* debería depender exclusivamente de la interfaz pública del módulo *Date*, y no de la forma en la cual se implementa este último módulo. Es decir, debería depender sólo de *Date* y no de módulos referentes a implementaciones concretas como *UserFriendlyDate* o *CounterDate*. Además, debería ser relativamente fácil cambiar la implementación concreta de la clase *Date* que se esté utilizando en cada momento.

En los lenguajes orientados a objetos, esta independencia se alcanza mediante la siguiente estrategia: cuando una clase A necesita utilizar otra clase B, se crea una interfaz o clase abstracta que contenga la interfaz pública de B. La clase A utiliza esta interfaz o clase abstracta como tipo para las referencias a la clase B. De esta forma se consigue que la clase A no dependa de implementaciones concretas de B. En nuestro ejemplo, se crearía una interfaz para la clase *Date* y se haría que la clase *Foo* contuviese referencias exclusivamente a la clase *Date*.

A continuación, se crean una o más clases concretas que hereden de la clase abstracta o interfaz y la implementen Estas clases concretas representarían implementaciones alternativas de la clase B. Por ejemplo, en nuestro caso se crearían dos subclases concretas de la clase *Date*, que denominaríamos *UserFriendlyDate* y *CounterDate*. Cada clase correspondería con una de las estrategias de implementación previamente comentadas.

[1] En referencia a una obra maestra del cine mudo, *Tiempos Modernos* (Charles Chaplin, 1936).

De esta manera, gracias al polimorfismo y la vinculación dinámica, se pueden utilizar instancias de estas clases concretas que sean compatibles con el tipo *Date* utilizado dentro de la clase *Foo*. Por tanto, la clase *Foo* no tendría dependencias con las clases concretas de *Date* y podría funcionar sin necesidad de modificaciones con sus diferentes implementaciones.

Esta técnica, sin embargo, tiene un punto débil. Dado que no se pueden crear instancias ni de clases abstractas ni de interfaces, si la clase A necesita crear una instancia de la clase B, dicha clase A necesitaría hacer referencia a una clase concreta de B. Por ejemplo, si la clase *Foo* necesitase crear una instancia de *Date*, la clase *Foo* tendría que hacer referencia a *UserFriendlyDate* o a *CounterDate*, por lo que al final acabaría acoplada a una de sus implementaciones concretas.

Para solventar este problema, existe un conjunto de patrones de diseño, denominados estructurales, entre los que destaca el *Abstract Factory*. Las factorías abstractas han ido ganando tal popularidad en los últimos años, que se han ido creando, en los últimos años, una serie de librerías y *frameworks*, conocidos como *inyectores de dependencias*, cuyo objetivo es permitir la especificación de factorías a un más alto nivel.

El objetivo general de esta práctica es que el alumno aprenda tanto a crear factorías abstractas desde cero como a crearlas mediante la utilización de un inyector de dependencias. El siguiente apartado refina esta serie de objetivos genéricos en un conjunto de objetivos concretos.

2. Objetivos

Los objetivos concretos de esta práctica son:

(1) Aprender a utilizar el patrón *Abstract Factory*.
(2) Aprender a utiliza el patrón *Singleton*.
(3) Aprender a utilizar el patrón *Prototype*.
(4) Aprender a utilizar un inyector de dependencias.

Para alcanzar dichos objetivos, el alumno deberá aplicar el patrón *Abstract Factory*, complementado con los patrones *Prototype* y *Singleton* a la situación que se describe a continuación. Dicha implementación se comparará con una basada en inyección de dependencias.

3. Configuración del Sistemas de Archivos *Sparrow*.

El Sistema de Archivos *Sparrow*, de acuerdo con la especificación creada por la organización *La Perla Negra*, puede configurarse mezclando los sistemas de visualización y las estrategias para la conversión de nombres de los modos que se describen a continuación.

En caso de que se opte por utilizar un sistema de visualización compacto, sólo se podrá utilizar la estrategia de visualización de nombres conocida como *Internacional Gallega*. Esta configuración se conoce como la configuración *Básica*.

Si se decide utilizar el sistema de visualización extendido, se puede optar tanto por no usar ninguna estrategia como por utilizar la *Internacional Gallega* o la *Internacional Catalana*. Estas configuraciones se conocen como *Estándar*, *Extendida Gallega* y *Extendida Catalana*, respectivamente.

Además, en el caso de la visualización extendida, el sistema de Archivos *Sparrow* debe soportar la existencia de una configuración personalizable donde la estrategia de visualización de nombres no esté predefinida, dejándose a elección del usuario. Esta estrategia, además, no tendría por qué estar entre las inicialmente definidas en la especificación de la *Perla Negra*. Por ejemplo, podría optarse por utilizar una estrategia donde la letra *ñ* se reemplazase por los caracteres *nn*, tal como ocurría en castellano antiguo. Esta última configuración se conoce como la configuración *Abierta*.

4. Actividades

El alumno, para alcanzar los objetivos planteados, deberá realizar satisfactoriamente las siguientes actividades, utilizando la implementación creada en prácticas anteriores:

1. Crear una factoría abstracta que soporte la creación de visitantes y estrategias concretas.
2. Crear factorías concretas para las configuraciones *Básica, Estándar, Extendida Gallega* y *Extendida Catalana*.
3. Aplicar el patrón *Singleton* a las factorías creadas en el punto anterior de manera que sólo pueda existir una instancia de la factoría abstracta en tiempo de ejecución.
4. Crear un programa de prueba que visualice el sistema de archivos de la Figura 1 utilizando la configuración *Extendida Catalana*. Dicho programa de pruebas utilizará las factorías para creadas anteriormente y cuando sea necesario para la instanciación de los visitantes y las estrategias concretas.

```
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
```

Figura 1. Ejemplo de Sistema de Archivos

5. Crear una factoría para la configuración *Abierta*, que utilice el patrón *Prototype* para soportar la configuración de la estrategia de visualización de nombres a utilizar.
6. Crear un programa de prueba que visualice el sistema de archivos de la Figura 1 utilizando la configuración *Abierta* personalizada con la estrategia *Gallega*. Dicho programa de pruebas utilizará las factorías, cuando sea necesario, para crear los visitantes y las estrategias concretas.
7. En un proyecto aparte, crear un programa de prueba que, utilizando inyección de dependencias para la creación de visitantes y estrategias concretas, visualice el sistema de archivos de la Figura 1 utilizando la configuración *Extendida Gallega*. Como inyector de dependencia se deberá utilizar el paquete *Ninject*.

5. Criterios de Evaluación y Aclaraciones

La práctica se entregará a través de la plataforma *moodle* siguiendo las instrucciones en ella proporcionadas.

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

(1) Correcta creación de la *factoría abstracta* (0.5 puntos).
(2) Correcta creación de las factorías concretas para las configuraciones *Básica*, Estándar, *Extendida Gallega* y *Extendida Catalana* (1.5 puntos).
(3) Correcta aplicación del patrón *Singleton* a la factoría abstracta (1 punto).
(4) Correcta utilización de las factorías para la visualización del sistema de archivos de la Figura 1 utilizando la configuración *Extendida Catalana* (1 punto).
(5) Correcta utilización del patrón *Prototype* para la implementación de la configuración *Abierta* (1.5 punto).
(6) Correcta utilización de las factorías y prototipos para la visualización de la Figura 1 utilizando la configuración *Abierta* con la estrategia *Gallega* (1 punto).
(7) Correcta utilización de la inyección de dependencias para la visualización de la Figura 1 utilizando la configuración *Extendida Gallega* (2.5 puntos).
(8) Cumplimiento de la *Guía de Buenas Prácticas en Programación* (1 punto).

Para facilitar la realización de la práctica, el alumno tiene disponible en la plataforma *moodle* un pequeño proyecto de ejemplo donde se utiliza *Ninject* para realizar la inyección de dependencia.

Pablo Sánchez Barreiro