PRÁCTICA 1: ¿SU CHINO ES CANTONÉS, NO?[1]

1. Introducción

Por desgracia, y a diferencia de lo que ocurre en otras Ingenierías, dentro del mundo del desarrollo software existe una notoria ausencia de estandarización. Este fenómeno se debe en parte a la rapidez con la que se suceden los cambios en este campo y a la lentitud normalmente asociada a los procesos de estandarización.

Como consecuencia, han ido apareciendo multitud de lenguajes que comparten el mismo propósito, se basan en los mismos principios y conceptos, pero presentan notaciones diferentes y pequeñas diferencias entre ellos.

Al estar estos lenguajes basados en los mismos principios y fundamentos, poseyendo un sólido conocimiento de dichos principios y fundamentos debería relativamente fácil aprender a utilizar nuevos lenguajes. Bastaría con aprender como una serie de conceptos ya conocidos se expresan en una notación diferente. En algunos casos, también sería necesario conocer como cada lenguaje interpreta algunos de esos conceptos.

Por ejemplo, si se dominan los fundamentos básicos de la programación orientado a objetos, aprender un nuevo lenguaje orientado a objetos, como *Ruby*, sólo requeriría conocer como ciertos conceptos, tales como clases, herencia, visibilidad de atributos o constructores, entre otros, se expresan en dicho lenguaje. Además, deberíamos informarnos de otros detalles que suelen variar entre lenguajes. Por ejemplo, cómo *Ruby* interpreta los modificadores de visibilidad de los atributos de una clase.

Una cuestión diferente es que queramos aprender un lenguaje perteneciente a un paradigma de programación diferente, basado en una nueva serie de principios y conceptos. Este sería el caso, por ejemplo, de un programador expertos en programación orientada a objetos que desease aprender a utilizar un lenguaje funcional como *Scala*. En este caso el principal obstáculo a superar no sería la notación a superar, sino aprender a utilizar conceptos propios de la programación funcional como la *inmutabilidad*, la *evaluación perezosa* o la utilización de *mónadas*. No obstante, una vez interiorizados estos conceptos, este programador debería ser capaz de aprender a utilizar nuevos lenguajes funcionales, como *Clojure*, con cierta facilidad.

El primer objetivo de esta práctica, que también se extiende al resto de la asignatura, es que el alumno comprueba mediante la experiencia las afirmaciones anteriormente realizadas de manera que pierda el miedo a aprender y a enfrentarse a nuevos lenguajes de programación. Para alcanzar este objetivo, el alumno deberá implementar un pequeño diseño *software* en un lenguaje de programación que se presupone que desconoce.

Dado que otro de los principales objetivos de la asignatura es que el alumno aprenda a utilizar e implementar *patrones de diseño software*, el diseño a implementar consistirá en la aplicación de un patrón de diseño sencillo a un problema determinado.

[1] Frase extraída de un famoso chiste de Forges (http://cort.as/67ru).

Los *Patrones de Diseño* representan uno de los mayores avances de la *Ingeniería del Software* en las últimas décadas. No existe prácticamente software de calidad alguno, de mediana o larga escala, que no haga uso de varios patrones de diseño.

La principal bondad de los patrones de diseño es que proporcionan soluciones elegantes, efectivas y sencillas a problemas recurrentes asociados al desarrollo de sistemas software.

Un patrón de diseño representa una idea brillante reutilizable para resolver un problema determinado. Por tanto, aplicar o instanciar un patrón de diseño significa aplicar dicha idea a un problema concreto.

El segundo objetivo de esta práctica es que el alumno aprenda a instanciar patrones de diseño mediante la aplicación de un patrón sencillo a un problema muy concreto.

La siguiente sección desarrolla los objetivos de esta práctica.

2. Objetivos

Los objetivos de alto nivel de esta práctica son:

(1) Corroborar, mediante la experiencia, que el aprendizaje de nuevos lenguajes de programación, asumiendo que se conocen los principios y fundamentos sobre los que se sustenta, es relativamente sencillo.
(2) Familiarizar al alumno con el concepto de patrón, mediante la aplicación e implementación de un patrón sencillo.

Para alcanzar el primer objetivo, para la implementación de esta práctica se utilizará un lenguaje de programación que en principio se presupone desconocido para el alumno. Dicho lenguaje será C#. Se ha optado por C# por ser un lenguaje muy similar a Java, que se entiende que es el lenguaje que el alumno ha estado usando principalmente a lo largo de sus estudios. Además, C# es el principal lenguaje de la plataforma .Net, la cual está muy demandada en el sector empresarial que nos rodea.

Como patrón de diseño a implementar se utilizará el patrón *mixin*. Se ha seleccionado dicho patrón por dos cuestiones principales: (1) su sencillez; y (2) sirve para ilustrar con claridad los conceptos de *herencia de comportamiento*, *herencia de tipo*, *reutilización por composición* y *delegación*. Estos conceptos son básicos para la asignatura de *Diseño Software*.

Por tanto, los objetivos concretos de bajo nivel de esta práctica son:

(1) Ser capaz de implementar en C#, incluyendo las correspondientes pruebas unitarias, un determinado diseño software aplicando previamente el patrón *mixin*.
(2) Comprender y saber utilizar los conceptos de *herencia de comportamiento*, *herencia de tipo*, *reutilización por composición* y *delegación*.

Para alcanzar dichos objetivos, el alumno deberá realizar de forma satisfactoria las actividades que se describen a continuación.

3. Actividades

Para alcanzar los objetivos anteriormente descritos, el alumno deberá completar las siguientes actividades:

1. Instalar el entorno Visual Studio, con la opción del lenguaje C#.
2. Realizar un mini tutorial sobre los aspectos básicos del entorno Visual Studio.
3. Conocer y comprender cómo C# gestiona la vinculación dinámica.
4. Implementar los diseños que se muestran en la Sección 4, utilizando para ello el patrón *mixin*.
5. Probar el correcto funcionamiento de la implementación creada, implementado las pruebas unitarias que se consideren necesarias.

4. Diseño a Implementar

Como es bien conocido, un buen pasiego debe saber hacer tanto ricas quesadas como sabrosos sobaos, dos magníficos productos típicos de la Vega de Pas. Además, se supone que es capaz de elaborar magníficos cocidos montañeses. Por otro lado, un lebaniego debería ser capaz de producir tanto un magnífico orujo como un magnífico cocido lebaniego. Uniendo ambas habilidades, sería posible crear una nueva raza superior de ser humano. Dicho nuevo *superhumano* sería el *PasiegoLebaniego*. Esta nueva especie sería capaz de producir tanto ricos cocidos lebaniegos como orujos, quesadas y sobaos, lo cual le conferirá una fuerza sobrenatural a la par que graves trastornos cardiovasculares.

La Figura 1 muestra un posible diseño, utilizando herencia múltiple, que simularía el comportamiento de *pasiegos*, *lebaniegos* y la nueva superraza, el *pasiego lebaniego*. Dicho diseño obvia, por el momento, las habilidades de pasiegos para la elaboración de cocidos.

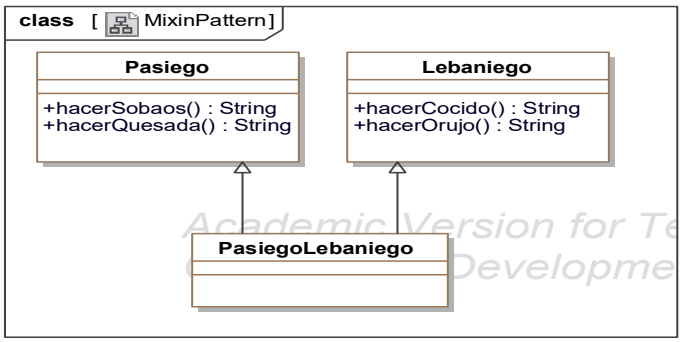

Figura 1. Jerarquía de clases para la clase *PasiegoLebaniego*

En la Figura 1, cada método de devuelve una cadena de caracteres que indica la acción que esté realizando el objeto en cuestión. Por ejemplo, si se invoca *hacerSobaos()*, el método devolvería "*Haciendo Sobaos.*". Ninguno de los métodos de la Figura 1 deben imprimir directamente por pantalla.

Evolucionaos ahora el diseño de la Figura 1 de manera que los pasiegos sean capaces de preparar ricos y reconstituyentes cocidos. Para ello, añadimos un nuevo método *hacerCocido()* a la clase *Pasiego*. Ahora la nueva superraza, el pasiego lebaniego, ha adquirido, gracias a la herencia, la habilidad de hacer cocidos tal como los haría un pasiego. Por tanto, el pasiego lebaniego es capaz ahora de elaborar cocidos tanto lebaniegos como montañeses, y elaborará uno u otro dependiendo de la región en la que se encuentre.

Para simular ese comportamiento, añadimos una propiedad *contexto*, que indique si el pasiego lebaniego se encuentra en la región de Liébana o en la región de la Vega de Pas (ver Figura 2). En función del valor de esta propiedad, al invocarse el método *hacerCocido,* un objeto de la clase *PasiegoLebaniego* cocinará un cocido u otro.

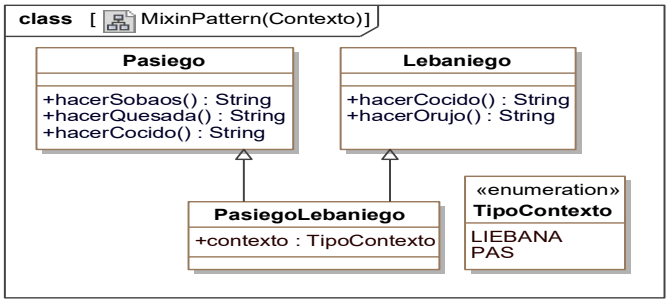

Figura 2. Jerarquía de clases modificada para la clase *PasiegoLebaniego*

5. Criterios de Evaluación y Aclaraciones

La practicará 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 aplicación del patrón *mixin* al diseño de la Figura 1 (6 puntos).
(2) Existencia y corrección de un conjunto de pruebas unitarias que sirvan para comprobar el correcto funcionamiento de la implementación de la Figura 1. (1 punto).
(3) Correcta aplicación del patrón *mixin* al diseño de la Figura 2 (1.5 puntos).
(4) Existencia y corrección de un conjunto de pruebas unitarias que sirvan para comprobar el correcto funcionamiento de la implementación de la Figura 1. (0.5 puntos).
(5) Cumplimiento de la *Guía de Buenas Prácticas en Programación* (1 punto).

Pablo Sánchez Barreiro