Análisis y diseño de software
Informe - Práctica 4
Fecha: 20/4/2017
Profesor: Miguel Ángel Mora
Grupo 2201

Pareja Nº 4
Luis Cárabe
(luis.carabe@estudiante.uam.es)
David García
(david.garcia03@estudiante.uam.es)

Introducción

En este informe explicamos las decisiones de diseño más relevantes y necesarias para la implementación de los distintos ejercicios de los que se compone la *Practica 3* de la asignatura de *Análisis y diseño de software*.

Apartado 1

En este apartado teníamos el cometido de implementar la interfaz IPhysicalUnit, para esto, creamos una clase abstracta **PhysicalUnit**. Esta clase contiene el método de transformTo, que nos sirve para pasar de unidad a otra; para que esta función fuese genérica para todo tipo de cambios de unidad (dentro de que sean compatibles claro está) creamos un atributo double que nos sirviera para dicha acción ya que dividiendo ese atributo de la clase a transformar por el atributo de la unidad física a convertir conseguimos el factor de conversión, simplemente multiplicando esto por la cantidad que deseamos convertir obtenemos la cantidad ya convertida. De esta clase heredan los tipos de unidades físicas, han sido necesarias dos clases, **LengthUnit** y **TimeUnit**, las dos contienen un tipo de unidad (Quantity) que hemos implementado como una enumeración. Por tanto, LengthUnit contiene un Quantity.LONGITUD y representa las unidades físicas de longitud y TimeUnit contiene Quantity.TIEMPO, que representa las unidades físicas de tiempo. En una primera implementación, pensamos que iba a ser útil tener clases como Meter, Inch o Hora, que heredaban de sus correspondientes unidades físicas, pero en apartados sucesivos nos dimos cuenta de que había una mejor implementación, lo explicaremos en el Apartado 2.

Apartado 2

En este apartado implementamos la interfaz IMetricSystem, para esto creamos la clase abstracta **MetricSystem** lo cual nos da soporte para los distintos sistemas métricos. En esta práctica hemos implementado tres, **SILengthMetricSystem**, que nos sirve para almacenar unidades del sistema metrico internacional con respecto a la longitud (metro, kilómetro y milímetro), **SITimeMetricSystem**, que nos sirve para almacenar unidades del sistema métrico internacional con respecto al tiempo (horas, minutos y milisegundos) y **ImperialLengthMetricSystem**, que nos sirve para almacenar unidades del sistema métrico imperial con respecto a la longitud (pulgadas, millas y pies). Todas las clases heredan de MetricSystem como es lógico y son Singleton.Dentro de cada respectiva clase nos encontramos con que la implementación de las unidades de cada sistema se hace con un atributo *public static final PhysicalUnit NOMBRE*, por ejemplo, para los metros, será public static final LengthUnit METERS, a esto le asignaremos un new LengthUnit con las características del metro. Como vemos, es posible implementarlo sin las clases extras dedicadas a cada unidad mencionadas anteriormente, esa parte del código resultaba redundante cuando con una instancia de LengthUnit ya queda claro el tipo de unidad a tratar, con su factor de conversión, abreviatura, Quantity... correspondientes. Lo mismo sucede con TimeUnit.

Apartado 3

En este apartado tuvimos que implementar la interfaz IMagnitude. Esta tarea fue sencilla, ya que simplemente creando la clase **Magnitude** y desarrollando ahí los métodos de dicha interfaz conseguimos la tarea requerida. Como la clase tiene un atributo IPhysicalUnit y un valor double, los métodos de suma y resta fueron fáciles de codificar, simplemente transformando la unidad con el método de Magnitude transformTo, que llama a transformTo de PhysicalUnit, (con lo que conseguimos la genericidad y la reutilización de código deseada en esta práctica) y sumando o restando el valor.

Apartado 4

En este apartado nos pedían implementar los convertidores, es decir, la interfaz IMetricSystemConverter, que contiene métodos para transformar unidades de un sistema métrico a otro. Para poder realizar bien nuestra tarea, creamos la clase **SiLength2ImperialConverter**, que implementa la interfaz anterior y da soporte al convertidor que pasa del sistema internacional de longitud al imperial. Para esto, necesitamos un multiplicador que actúe como factor de conversión. El método *transformTo* es el que se encarga de realizar dicha tarea de conversión. Cuando llegamos a este apartado, solo teníamos un tipo de Exception, **QuantityException** (que implementa la excepcion cuando dos unidades no son compatibles) y fue cuando nos dimos cuenta de que necesitabamos varias excepciones más, así creamos **UnknownUnitException** (cuando se desconoce la unidad) que hereda de QuantityException, **NotMatchingException** (cuando dos unidades a transformar no corresponden con las del conversor invocado) que hereda de UnknownUnitException y **TypeException** (cuando no se puede transformar de una unidad a otra debido al tipo) que hereda de QuantityException. Con esto ya podíamos tener un control exhaustivo. También hubo que cambiar los métodos de transformTo y canTransform de PhysicalUnit, ya que ahora no debían de dar error al transformar unidades de diferentes sistema, simplemente debían buscar entre sus convertidores uno adecuado para poder realizar su cometido.

Apartado 5

Diagrama de clases: (Figura 1, al final del documento)

b) ¿Es extensible tu diseño? Indica qué pasos habría que dar para:

Sí, nuestro diseño es extensible.

b.1) añadir nuevas unidades a un sistema existente

Para esto, sencillamente tendríamos que añadir un atributo del tipo **public static final PhysicalUnit NOMBRE** en la clase del sistema métrico deseado. Simplemente sustituiríamos en lo anterior PhysicalUnit por el tipo de unidad física requerida (como LengthUnit) y se llamaría al constructor de esta clase con los datos de la nueva unidad.

b.2) añadir la cantidad Masa al Sistema Métrico Internacional

Para esta tarea, necesitaríamos crear una clase **UnidadMasa** que heredara de PhysicalUnit. También deberíamos prolongar la enumeración Quantity con el tipo de unidad MASA. Finalmente, tendríamos que crear una clase que representara todas las unidades de masa del sistema metrico internacional, como ya hicimos con SILengthMetricSystem. En dicha clase, ya podríamos añadir todas las únidades de masa que quisieramos.

c) ¿Qué desventajas o limitaciones tiene el diseño de la librería?

No obstante, a pesar de que nuestra librería es extensible, tiene ciertas desventajas, ya que hasta el apartado 4, estaba pensada solamente para tratar unidades simples, y tras el apartado

6, somos capaces de operar con unidades compuestas tales como el m/s. La cuestión resulta ser que, fuera de esos tipos de unidades, hay otras que no conseguirían encajar en nuestra librería (como m/s^2), ya que para ello deberíamos crear nuevas clases y restructurar algunas ya creadas, modificando algunos métodos, lo que limitaría el uso de la librería.

Apartado 6 (opcional)

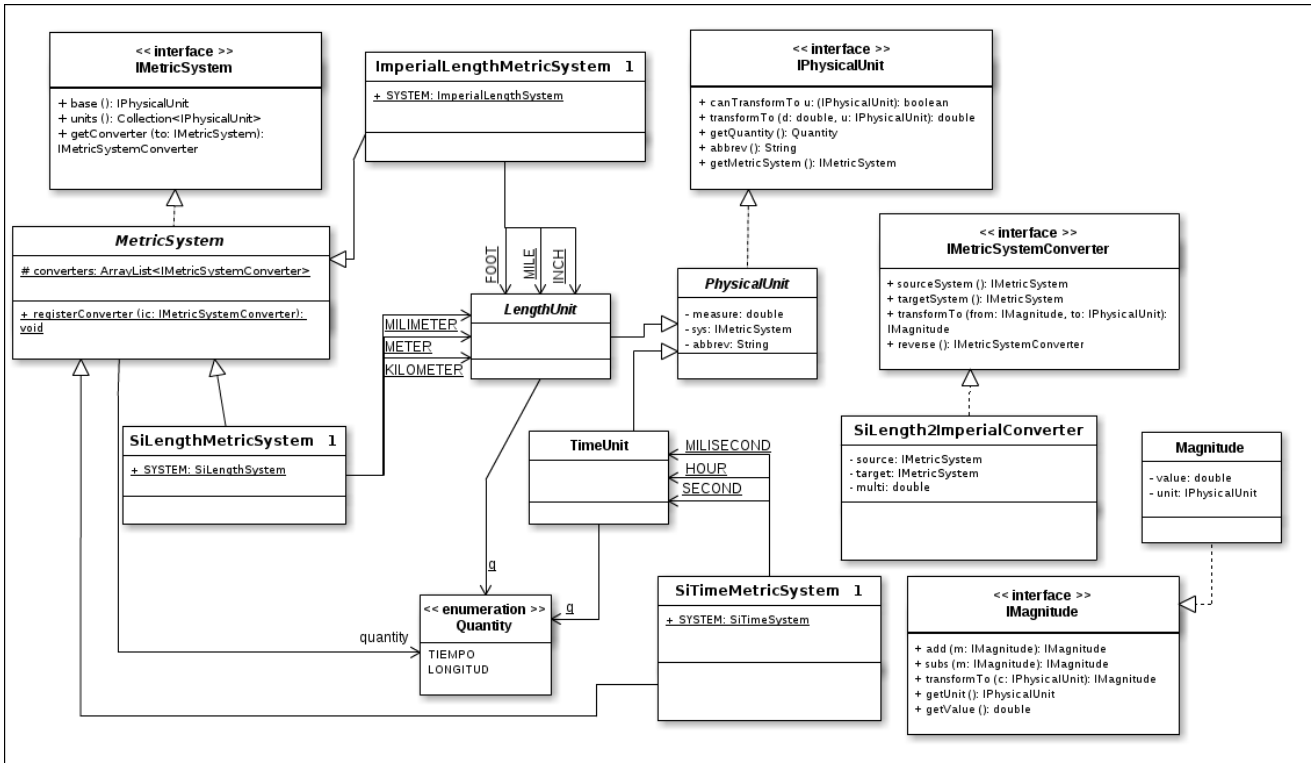

Figura 1: Diagrama de clases Apartados 1 - 4