Apartado 5: Diagrama de clases y explicación del diseño

a) Debes entregar un diagrama de clases que muestre el diseño efectuado. No olvides que el objetivo del diagrama es explicar el diseño con un nivel de abstracción adecuado, no es "Java en dibujos". De esta manera, debes obviar constructores, getters y setters, y debes representar las colecciones, arrays y referencias usadas como asociaciones del tipo más adecuado. No olvides incluir las interfaces que has usado, y las relaciones de implementación entre clases e interfaces. Incluye una pequeña explicación del diseño, así como de las decisiones que has tomado.

.
Originalmente implementamos los sistemas métricos como enumeraciones con atributos, en los que las unidades eran cada uno de los valores del enumerado. Sin embargo, debido a problemas con la constante SYSTEM de los tester, descartamos esta implementación.

Para implementar las unidades hemos creado una clase abstracta MetricUnit que implementa todos los métodos sin tener en cuenta la Quantity concreta de la unidad. Para implementar cada tipo de unidad hemos creado clases que heredan de MetricUnit y que únicamente contienen un constructor que llama al de MetricUnit con la Quantity correspondiente.

Para implementar los sistemas métricos decidimos que las unidades del sistema fueran variables estáticas dentro de la clase, y que la base se definiera por el valor devuelto por base(). Finalmente lo implementamos como una enumeración con un único valor SYSTEM, aunque se podría haber implementado como un Singleton tradicional.

Las magnitudes fueron relativamente sencillas de implementar, pero una decisión que tomamos fue copiar la magnitud de entrada en los métodos add y sub y transformar la copia en la nueva unidad, evitando así modificar la magnitud que se desea sumar o restar.

Los convertidores fueron la parte más compleja: finalmente creamos una clase MetricSystemConverter general, y solo creamos una subclase para el caso particular de SiLength2ImperialConverter, que al igual que las unidades únicamente posee un constructor que llama al de su superclase. Además, añadimos la lista de convertidores del sistema a esta clase como una variable estática, e hicimos que los métodos registerConverter y getConverter llamen a métodos estáticos de MetricSystemConverter. La ventaja de esta implementación respecto a otras que probamos es la simplicidad de registrar los convertidores inversos en el sistema.

Las excepciones que definimos son únicamente las pedidas por las interfaces, y aunque contemplamos hacer que UnknownUnitException heredara de QuantityException, finalmente decidimos que no era la mejor opción, ya que la excepción de unidad desconocida no es exactamente un subtipo de la excepción de cantidad.

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

b.1) añadir nuevas unidades a un sistema existente.

Para añadir una unidad a un sistema existente, es necesario crear una nueva variable estática en el sistema al que se quiere añadir la unidad. Esta variable estática debe ser el mismo tipo que el resto de unidades y debe crearse con su abreviatura, número por el que multiplicar para pasar a la base y SYSTEM como último argumento.

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

-Primero: añadir a Quantity el valor MASA

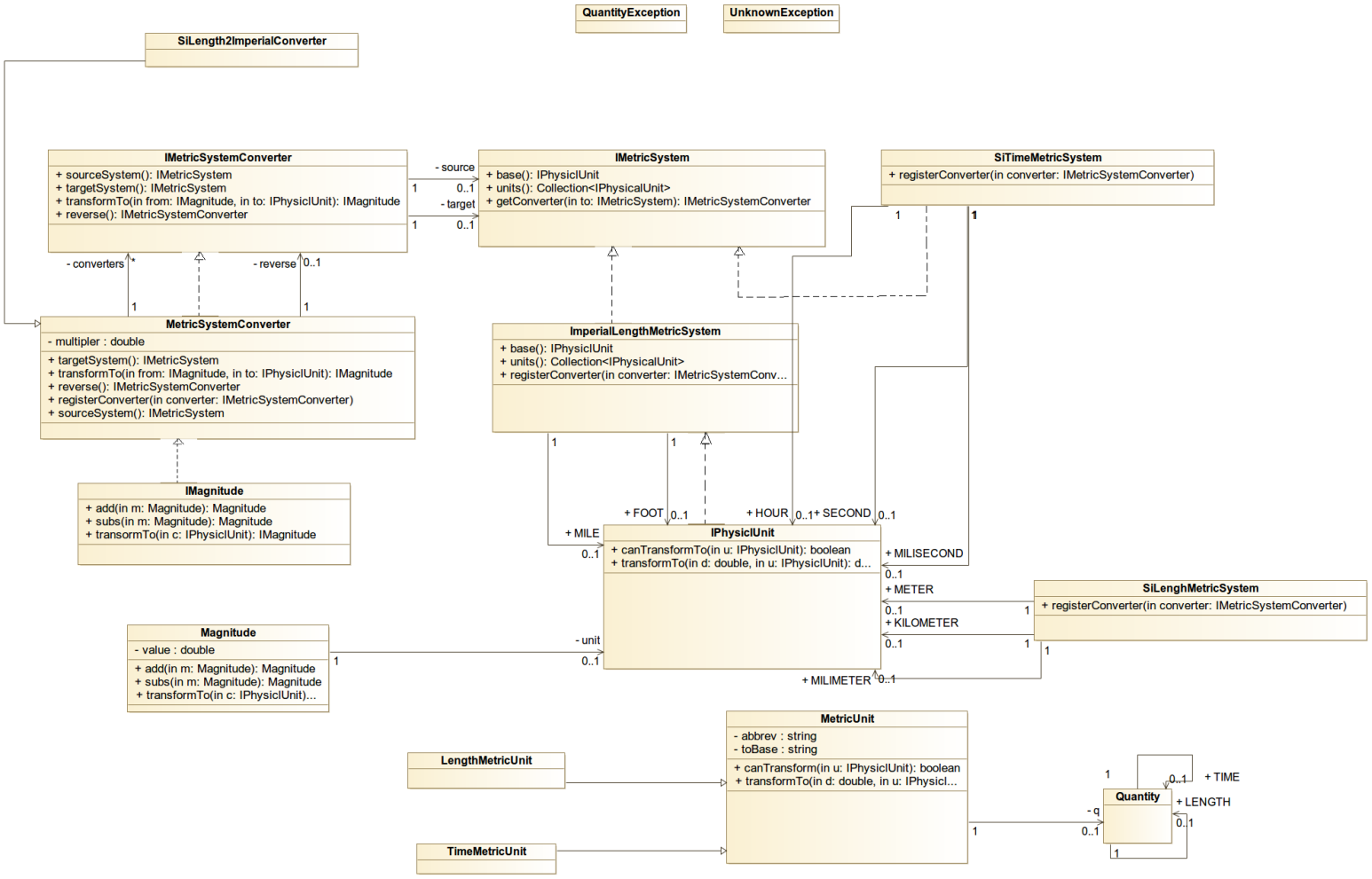
-Segundo: crear una nueva clase en *units* llamada MasaMetricUnit que herede de MetricUnit, y definir un constructor que llame al super pasando MASA como Quantity.
-Tercero: crear una nueva clase en *metricsystem* llamada SIMasaMetricSystem que implemente IMetricSystem, e implementar las unidades que se desee. En concreto, el método base() debe devolver la unidad kilogramo y units() debe devolver todas las unidades que se hayan implementado, mientras que registerConverter y getConverter se implementan de forma análoga a los otros sistemas.

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

Una desventaja es la necesidad de definir cada sistema métrico manualmente, tanto crear un nuevo sistema como definir sus unidades, su base y asegurarse de que units() devuelve todas las unidades que existen.

Otra desventaja es la posibilidad de crear convertidores entre sistemas libremente, lo que podría crear problemas si se crean dos convertidores entre los mismos sistemas con distinto multiplicador.
Por último, una posible desventaja es la dificultad de navegación entre los distintos transformTo, que se llaman entre sí de forma algo caótico una vez entran en juego los convertidores. Esto es en parte debido a la confusión provocada por el enunciado carente de suficiente claridad.

DIAGRAMA DE CLASES:



Miguel García Moya y Lucía Colmenarejo Pérez.