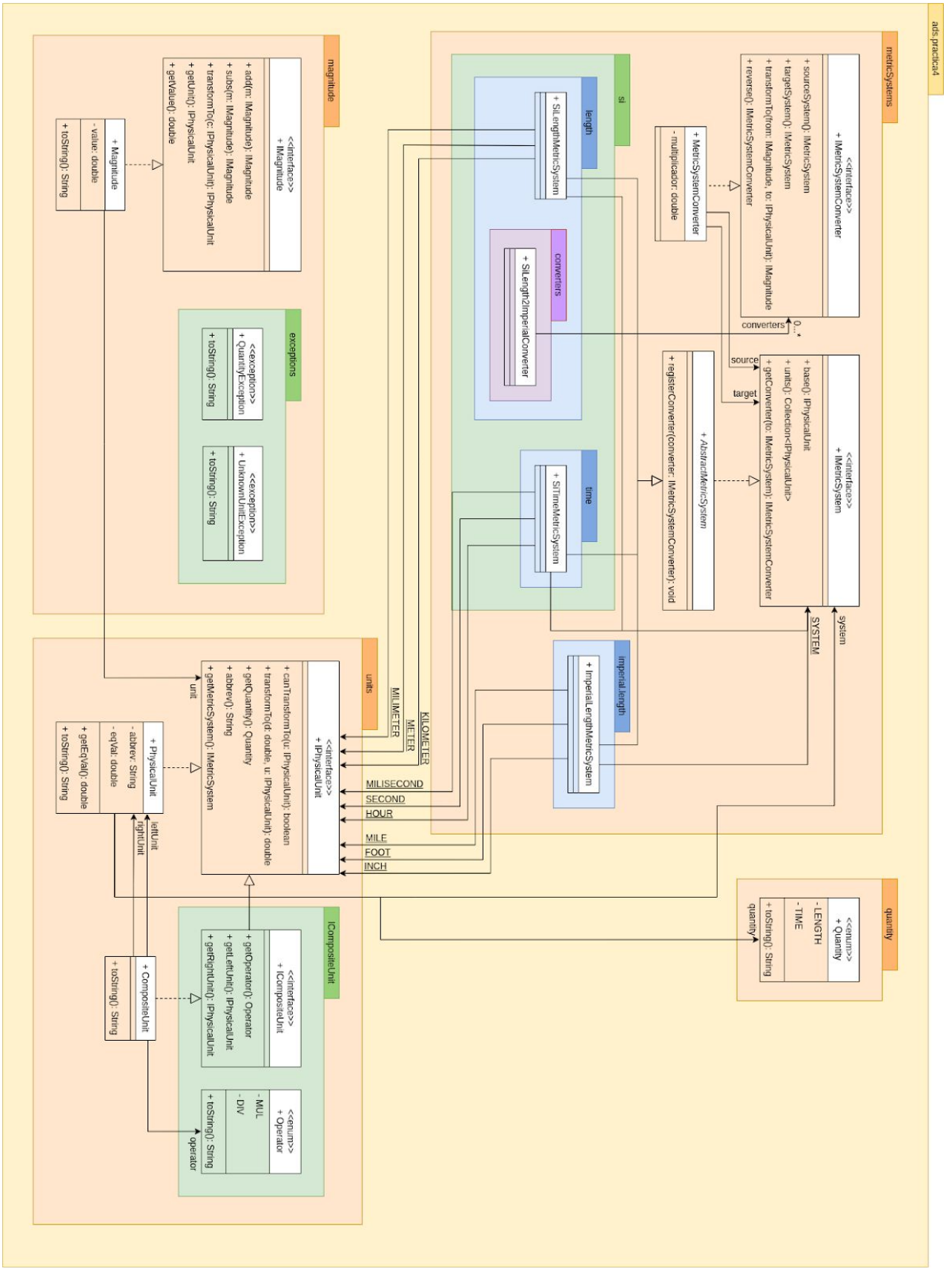
MEMORIA PRÁCTICA 4

ANÁLISIS Y DISEÑO DE SOFTWARE

Laura de Paz Carbajo
Rubén García de la Fuente
Grupo 2201
15/04/2020

Apartado 5:

a) Adjuntamos el diagrama en horizontal para una mejor visualización:



Hemos añadido color para una mejor visualización debido a la cantidad de información que contiene, de manera que los paquetes que son del mismo color están al mismo nivel.
Tenemos 4 paquetes principales:

- quantity: contiene un enumerado con las cantidades medibles por sistemas métricos (tiempo y longitud para esta práctica).
- units: contiene las magnitudes e interfaces relacionadas con las unidades que luego podrán ser usadas por los sistemas métricos (las unidades físicas simples y compuestas).
- magnitude: entendidas como un par valor-unidad, son las que nos permiten realizar operaciones aritméticas con ellas y contienen todas las interfaces y clases para una correcta implementación.
- metricSystems: este paquete es el más extenso, pues contiene toda la información relacionada con los diferentes sistemas métricos (tanto internacional como imperial), los distintos sistemas físicos (de longitud, de tiempo) y sus conversiones entre ellos.

b) El diseño que hemos implementado es extensible ya que para añadir nuevos elementos como sistemas métricos, unidades físicas o conversores únicamente es necesario crear una nueva clase que herede de ciertas clase o implemente ciertas interfaces y codificar sus métodos propios o heredados.

- Para añadir una nueva unidad física a un sistema métrico ya existente es necesario añadir un nuevo atributo a la clase correspondiente del sistema métrico en cuestión, pasándole como argumentos al constructor de PhysicalUnit la abreviación de la unidad, su equivalencia respecto a la unidad base, tipo de Quantity (cantidad que mide) y el sistema métrico al que pertenece. Si la nueva unidad es la base del sistema, en al función base() habrá que devolver este nueva unidad e independientemente de si es la base o no, en units() hay que agregarla al array de unidades físicas del sistema.

- Si queremos añadir la cantidad Masa en el Sistema Métrico Internacional, lo primero que hay que hacer es añadir un elemento a la enumeración Quantity ya que todavía no se han tratado unidades de masa. A continuación, se ha de crear una clase SiMassMetricSystem (o similar) dentro del paquete ads.practica4.metricSystems.si.mass que herede de la clase AbstractMetricSystem, y debe incluir como atributos esta nueva clase las unidades que se deseen pertenecientes a este sistema y también el atributo SYSTEM, que es realmente el sistema métrico que se acaba de crear (por el patrón Singleton). Hay que implementar los métodos base() y units() definidos en la interfaz IMetricSystem para esta nueva clase. Para ello, una de las unidades que incluya esta clase ha de ser la base del sistema (si se tratase del sistema internacional sería KILOGRAM con eqVal = 1) y hay que crear manualmente el array de unidades con cada una de ellas.

c) Un gran inconveniente de esta implementación es la función units() que implementa cada clase que hereda de AbtractMetricSystem y que como se ha comentado en el apartado anterior, hay que añadir manualmente en el array todas las unidades del sistema métrico.

   Una clara desventaja es la implementación de las unidades compuestas, pues al tener que implementar IPhysicalUnit (al tratarse de una unidad que contiene a otras dos) debe implementar métodos como getQuantity o getMetricSystem. El método getQuantity tiene el inconveniente de que las cantidades compuestas (velocidad, fuerza) no han sido necesariamente añadidas al enumerado, el método getMetricSystem tiene el inconveniente por su parte de que una unidad compuesta puede estar formada por unidades simples pertenecientes a distintos sistemas métricos, por lo que el retorno de la misma es ambiguo. Esto se agravaría más si, en una unidad compuesta, utilizáramos un ICompositeUnit en rightUnit o leftUnit (es decir, que una de las unidades de las que se compone una unidad compuesta fuese, a su vez, otra unidad compuesta).

   Por último destacar que existe una cierta ambigüedad con el método transformTo de IMetricSystemConverter. Esto es debido a que una magnitud debería ser capaz de transformarse en otra si existe conversor entre sistema métrico (es decir, Magnitude llama a MetricSystemConverter), por otro lado, el método transformTo de IMetricSystemConverter puede convertir cualquier magnitud del sistema métrico original al sistema métrico fuente (MetricSystemConverter llama a Magnitude). Como podemos observar ambos métodos requieren la ejecución del método homónimo de la otra clase para completarse rompiendo así la jerarquía en la que los métodos más “complejos” van llamando a los métodos más simples para finalizar su ejecución. Creemos por ejemplo que sería más conveniente que el método transformTo de IMetricSystemConverter transformase unidades y no magnitudes, y fuese únicamente el método transformTo de Magnitude el que llamase a transformTo de se IMetricSystemConverter.

*Nota: en la memoria se indica que el apartado 6 (el opcional) se ha de entregar en un directorio diferente. Como nuestro diseño de este apartado no modifica el funcionamiento de lo anterior, lo hemos mantenido en el mismo directorio.