Molinaroli Carlos Atilio, Blanco Valentín Nicolas

Entregable Teorico N1

1) Las magnitudes de longitud del **sistema Internacional** fueron divididas en **4 clases** (Kilometro, Metro, Centímetro y Milímetro) cada una contando con métodos heredados de su clase padre LongInt (que a su vez esta hereda de longitud y esta ultima de magnitud) tales como son convertir (convierte la unidad de longitud internacional a otra de su sistema y recibe como parámetro un String que indica a cual sera transformada), suma (realiza la suma entre dos unidades de longitud sin importar si la pasada por parámetro es del ingles ya que sera transformada a la unidad con la que se trabaja), resta (realiza la resta entre dos unidades de longitud, al igual que la suma tampoco importa si la pasada por parámetro pertenece a otro sistema de longitud) y comparar (realiza la comparación entre dos unidades de longitud devolviendo true en caso de tener el mismo valor, la unidad pasada por parámetro para eso es transformada a la longitud instanciada). Por otra parte se encuentra el **sistema ingles** que fue dividido en **3 clases** (Yarda, Pie y Pulgada) que al igual que las clases de longitud del sistema internacional cuentan con los mismos métodos ya que fueron heredados de la clase Longitud, clase padre de LongInt y LongIng siendo esta ultima de donde heredan estos. Estos métodos trabajan de la misma manera que los de las subclases de LongInt. Otra forma de plantearla podría ser sin el uso de subclases para estas longitudes pero para ello requeriríamos una variable privada extra en la clase magnitud de tipo String (o de tipo enumerativo) que haga referencia a que tipo de unidad de longitud instanciaremos (en ese caso haríamos el set de cual sera la unidad a representar por ejemplo setUnidad(“metro”)) y para la transformación dentro del sistema con el que trabajemos haríamos un switch de este String (switch (super.getUnidad())) donde preguntaríamos a que unidad del sistema hace referencia y dentro de cada unidad usaríamos otro switch donde transformaremos esta unidad al parámetro recibido (al igual que lo hicimos en el método convertir), luego los otros métodos se implementarían de la misma manera que utilizando subclases con la diferencia de hacer el switch para saber con que unidad trabajamos para así por ejemplo en la suma/conversión/resta sabemos a que unidad transformar el parámetro recibido, aunque finalmente optamos por utilizar la forma de división es subclases ya que es mas optima refiriéndonos con esto a que consideramos cada unidad como un objeto que adopta el comportamiento del padre (siendo obligado a esto a través del uso de clases abstractas) y visualizando de esta forma una jerarquía de clases.

2) Para nuestro proyecto utilizamos clases abstractas en vez de interfaces, los **métodos abstractos** usados fueron **suma**, **resta**, **convertir**, **comparar**. Los mismos tuvieron implementación para las subclases de unidades (por ejemplo metro,segundo, pie, kilogramo, libra,etc) por lo que las clases padres eran las que tenían la etiqueta abstract que les obligaba a implementarlos a los hijos. Nuestra decisión fue optar por utilizar clases abstractas ya que no era requerido un sistema similar a herencia múltiple, en nuestro caso todas las subclases de unidades hacen uso de los mismos métodos, de igual forma no consideramos que existe una ventaja entre el uso de interfaces o clases abstractas en nuestra implementación por lo que se podría realizar con interfaces (se requeriría una sola interfaz con los 4 métodos que en nuestro caso declaramos abstractos).

3) Para comparar unidades de longitud diferentes optamos por preguntar si la unidad recibida por parámetro pertenecía al mismo sistema, haciendo uso del comando instanceof (ejemplo de esta utilización `if (m instanceof LongIng)` preguntando en este caso si la variable m era instancia del sistema ingles de longitud), con esto se podían obtener tres

resultados, el primero de ellos siendo que este pertenece al mismo sistema en donde solo se debe realizar la transformación a la misma unidad y preguntar si el resultado de esta transformación era igual al valor de la unidad, el segundo siendo que el parámetro recibido no pertenece al mismo sistema donde antes de transformarlo a la misma unidad debíamos pasarlo al mismo sistema a través de una clase que realizaba este cambio con un método estático (por lo que podíamos instanciarlo en cualquier clase) por ejemplo en nuestras subclases de longInt hacíamos `aux= Conversoraotrosist.cambiarSistemaLong("pie",m.convertir("pie"))` que trabaja recibiendo dos parametros el primero un String que indica a cual sistema se quiere convertir y el segundo el valor en metros/pies, por lo tanto este nos permite guardarnos el cambio de sistema en la variable aux y con ello realizar la transformación dentro del mismo sistema para comparar las mismas unidades, y el tercer caso (no visible en código pero si en practica) en donde la magnitud instanciada es otro tipo de magnitud (por ejemplo una de tiempo como segundo) donde no se debe transformar (ya que no tienen relacion) y es retornado falso.

4) Para la suma y resta entre unidades de diferente sistema métrico (ingles e internacional) se utilizo el mismo recurso que para la comparación con esto nos referimos al instanceof que fue utilizado ademas de la misma forma que en la comparación es decir cuando queríamos sumar/restar unidades de diferentes sistemas ejecuta el `if (m instanceof LongIng)` para el caso de las sumas/restas en sistemas internacionales o `if (m instanceof LongInt)` el en sistemas ingleses, luego de esa instrucción también se hace el uso de la clase Conversoraotrosist y al igual que en la comparación se utiliza el método estático cambiarSistemaLong guardando en un auxiliar el valor de la conversión y transformando ese auxiliar al valor de la unidad para sumárselo o restárselo.

5) La clase **SondaEspacial** recibe parámetros de **cualquier** tipo de **unidad** ya sea como la que utilizamos en test donde realizamos la impresión de una yarda de la siguiente forma `s.imprimir(y);` siendo s una instancia de sonda (`SondaEspacial<Yarda> s= new SondaEspacial<Yarda>();`) e y el objeto yarda (`y= new Yarda();`), otros ejemplos podrían ser también unidades como segundos o kilogramos. En nuestro caso no le pusimos ninguna cota pero debíamos especificar que el parámetro recibido debía extender de magnitud, esto seria realizado de la siguiente manera: `public class SondaEspacial <T extends Magnitud>`. La ventaja que observamos definiendo a sonda espacial como tipo genérico es que nos permite ahorrar código ya que no es necesario declarar un método por cada unidad de magnitud.

6) La interface publica (lo que el usuario puede observar) de la clase sondaEspacial es el metodo imprimir que realiza la impresión en pantalla de la unidad (el parámetro recibido), la interfaz privada por otra parte lo formaria el metodo toString de la unidad que es invocado cuando se realiza la impresion en el metodo imprimir asi como tambien el getCifra que se encontraria en el metodo toString de la unidad (nosotros interpretamos que se debia imprimir el hashcode), lo hubiesemos implementado de la siguiente manera `public String toString(){ return "unidad:"+this.getCifra()+"cm"; }` para el caso de centimetros por ejemplo. En el trabajo elegimos imprimir como interfaz publica porque es lo que el usuario ve ya que se trata de un print y como interfaz privada a los métodos toString y getCifra porque el usuario no es conciente desde su visión de que se están ejecutando