Prácticas de Programación Orientada a Objetos
Curso 2006/2007

Entrega 3

El **plazo de entrega** será establecido el día de presentación del ejercicio en clase y notificado en SUMA.

Introducción.

El objetivo de esta práctica es la implementación de dos nuevos tipos de figuras relacionadas con las características geométricas y físicas del videojuego. Por un lado, se programarán los cubos que hacen de soporte, que son cubos sólidos que pueden tener encima otros objetos que cumplan ciertas propiedades. Y por otro lado, se definirán los cubos contenedor que, como su nombre indica, pueden contener a otros cubos.

Asimismo, en esta entrega se pretende seguir aplicando el concepto de herencia, utilizar el concepto de genericidad para representar la relación de agregación entre los cubos contenedores y los cubos contenidos, y aplicar el concepto de *interface* Java. Se implementarán las operaciones de igualdad, clonación y representación como cadenas de texto de la biblioteca de figuras geométricas siguiendo las convenciones de programación de Java. En esta entrega es necesario trabajar con las colecciones de Java y se introducirá el uso de las excepciones como mecanismo de control de errores y su aplicación en la técnica de Diseño por Contrato de Bertrand Meyer.

El trabajo de esta entrega cierra la primera fase de desarrollo de la práctica. El propósito de este primer ciclo ha sido la implementación de una primera aproximación de una biblioteca de clases de geometría que dé soporte a la construcción del videojuego. No obstante, el desarrollo de software normalmente suele ser iterativo. Es decir, es probable que al profundizar en los requisitos del videojuego surja nueva funcionalidad que haya que incluir en las clases de la biblioteca e incluso cambiar algunas operaciones. Finalmente, con esta entrega tendremos una entrevista de revisión con todos los grupos de prácticas.

Trabajo.

1. Implementa la clase **CuboSoporte** que representa un cubo sólido sobre el que se puede apoyar *cualquier* objeto que cumpla una serie de propiedades. Este clase debe ofrecer la siguiente funcionalidad:
    a. Un método para notificar al cubo soporte que otro objeto se sitúa sobre él. El objeto que se coloque sobre el cubo debe ofrecer dos propiedades: la región que ocupa en el espacio (en general, un paralelepípedo) y su masa. La precondición para situarse sobre el cubo es que la región del objeto sea contigua en la cara de arriba.
    b. Otro método que elimine el registro de alguno de los objetos soportados. La precondición de este método es que el objeto ya estuviera sobre el cubo. Por

tanto, se ofrecerá un método de consulta para poder comprobar si un objeto ya está soportado por un cubo soporte.

c. Hay que tener en cuenta que la masa del cubo soporte varía en función de la masa de los objetos que soporta.

d. Interesa notificar la situación en la que el cubo soporte cambia de masa. Para ello ha de implementarse un mecanismo que permita a cualquier objeto, que cumpla una serie de condiciones, registrarse/borrarse como oyente interesado en ese evento, y que permita, cuando suceda el evento, a los objetos interesados conocer el cambio de masa, es decir, la masa anterior y la masa actual.

e. La operación de desplazamiento afecta tanto al cubo soporte como a los objetos soportados. Por tanto, los objetos soportados deben ofrecer como tercera característica un método para desplazarlos.

2. Programa la clase **CuboContenedor** que represente a un cubo que puede contener a otros cubos. Es importante que la implementación exprese esta relación de agregación y que sea posible restringir el tipo de cubo que puede contener en el momento de la instanciación. Esta clase ofrecerá la siguiente funcionalidad:

   a. Un método para indicar que un cubo va a ser contenido por el cubo contenedor. La precondición para esta operación es que el cubo contenido esté incluido dentro del contenedor, es decir, que geométricamente un cubo esté dentro de otro. En tal caso, la operación *normalizará* las coordenadas del cubo contenido, es decir, las coordenadas del cubo contenido serán modificadas suponiendo que el sistema de referencia es el punto inferior del cubo contenedor (simplemente, a las coordenadas de los dos puntos de referencia del cubo contenido se les resta las del punto inferior del contenedor).

      Por otro lado, la inserción podría no ser legal dependiendo de la política de control que establece cómo ha de ser el desplazamiento e inserción en el cubo contenedor, que pueden ser cualesquiera. Si el nuevo cubo no cumple estas reglas, no se insertará y se devolverá un valor booleano informando de esta situación (más adelante se describe la naturaleza de estas reglas).

      Nótese, que según el diseño propuesto, puede darse la paradoja de que un cubo pueda estar *geométricamente* dentro de un cubo contenedor y no estar *contenido* por él. Aunque esta situación es legal (evitarla sería complejo), lo normal es que un cubo que esté dentro del contenedor forme parte de su contenido. Esta relación de agregación tiene varias implicaciones que más adelante se detallan. Por tanto, serán las clases del videojuego que usen esta biblioteca de geometría las que deberán evitar esa situación anómala.

   b. Otro método para sacar un cubo de su cubo contenedor. La precondición es que el cubo ya esté contenido. Esta operación es contraria a la anterior e implica una *desnormalización* de las coordenadas del cubo (sumarle las del punto inferior del cubo contenedor).

      Es importante destacar que no debe permitirse la desnormalización de las coordenadas de un cubo, si previamente no ha sido normalizado respecto al cubo contenedor de referencia. Por tanto, hay que valorar las implicaciones que tendrá en la clase Cubo las operaciones de normalización y

desnormalización de sus coordenadas, y su relación con el cubo contenedor. Es posible que sea necesario introducir nuevos atributos y métodos en la clase Cubo.

c. La política de control que rige el modo de desplazamiento y de inserción de los cubos contenidos dentro del cubo contenedor podrá ser diferente para cada contenedor. Ejemplos de restricciones de la política de control podrían ser: que no se permita que un cubo se desplace quedando fuera del cubo contenedor, que al desplazarse no colisione con otro cubo, que un cubo normal pueda colisionar con otro cubo normal, pero no con un cubo sólido, que sean ilegales los movimientos hacia abajo o que en la inserción no se permita que un nuevo cubo colisione con otro ya existente. Por tanto, debe programarse una solución que ofrezca esta flexibilidad a la hora de instanciar a los cubos contenedores. Es decir, las políticas de control serán objetos con ciertas características que ofrezcan tanto al cubo contenedor como a los cubos contenidos un modo de controlar si tanto las inserciones como los desplazamientos son válidos.

   A modo de ejemplo para esta entrega, implementaremos la siguiente política: "no se permiten desplazamientos fuera del contenedor y no se permiten colisiones entre los cubos, ni en el desplazamiento ni en la inserción".

   Nótese que la política de control puede limitar los desplazamientos de los cubos dentro de un cubo contenedor. Por tanto, esta situación tiene un impacto en el diseño de la clase Cubo. Es decir, el propio método de desplazamiento de un cubo debe tratar la situación en la que esté dentro de un cubo contenedor y, por tanto, que debe cumplir la política establecida por éste. Si la operación de desplazamiento no es legal, no podrá realizarse.

   Por otro lado, téngase en cuenta que los cubos contenidos tienen sus coordenadas normalizadas respecto al cubo contenedor. Esto significa que, cuando el contenedor se mueva por el espacio, implícitamente también se mueven los cubos contenidos sin necesidad de modificar sus coordenadas, ya que el sistema de referencia de los cubos contenidos es el cubo contenedor.

3. Implementa el método **toString** para todas las clases de la biblioteca de geometría de acuerdo con el formato propuesto en clase, el método **equals** para todas las clases excepto CuboSoporte y CuboContenedor, siguiendo la propuesta de clase, y el método **clone** para todas las clases, pero en este caso razonando qué tipo de copia conviene para cada clase. Por ejemplo, para el Paralelepípedo y el Cubo interesa una copia en profundidad.

4. Utiliza **excepciones** *Runtime* de Java para controlar las **precondiciones** de los métodos. Normalmente en Java las excepciones que se utilizan para notificar la violación de las precondiciones son *IllegalArgumentException*, si el argumento pasado a un método o a un constructor no cumple unas restricciones, e *IllegalStateException*, si se trata de aplicar un método y el objeto no está en el estado adecuado. Para este apartado habrá que revisar el código programado, identificar y documentar las precondiciones de los métodos y constructores, y utilizar este tipo de excepciones para controlar su cumplimiento.

5. Las clases implementadas serán probadas por las clases de prueba **TestCuboSoporte** y **TestCuboContenedor**, que formarán parte del paquete "cubos.geometria.tests". La implementación de las pruebas es a criterio de cada

grupo, aunque se valorará que el código de prueba sea significativo y que esté bien documentado. Al igual que en la entrega anterior, se definirá un lanzador para cada una de las clases con el fin de poder lanzarlas independientemente. En esta entrega sigue siendo importante hacer pruebas que comprueben que se aplica correctamente la ligadura dinámica en las implementaciones de las subclases. Por último, ten en cuenta que será necesario incluir o modificar algunas de las pruebas programadas en entregas anteriores.

Entrega.

Todas las clases deberán estar documentadas adecuadamente en **javadoc** y deberá generarse la **documentación** para cada entrega. El **responsable** del grupo (el primer alumno del grupo) deberá dejar la práctica en su zona de contenidos de la asignatura. La falta de entrega en el plazo indicado supondrá un 5 % de penalización sobre la nota final de la práctica.

Esta entrega también deberá entregarse en **papel**. El formato de la entrega debe ser el siguiente:

- Una portada con el nombre y dirección de correo electrónico de los miembros del grupo de prácticas, la titulación y el profesor
- El código fuente de todas las clases implementadas hasta el momento impreso preferiblemente por las **dos caras** del folio.
- El código impreso no debe encuadernarse, simplemente meterlo en una funda de plástico.
- El trabajo debéis dejarlo en Conserjería.

Criterios de evaluación.

1. Uso adecuado de las colecciones de Java (10 %).
2. Correcta aplicación del concepto de genericidad (10 %).
3. Uso adecuado del concepto de interface Java (15 %).
4. Interpretación correcta de las precondiciones y su control mediante excepciones (10 %)
5. Correcta implementación de los métodos *toString*, *equals* y *clone* (10 %).
6. Corrección y completitud de la implementación de la funcionalidad (15 %).
7. Definición adecuada de las clases de prueba (5 %).
8. Resto de criterios de evaluación aplicados en entregas anteriores: uso de la herencia, legibilidad del código, javadoc, etc. (25 %).