5 DISEÑO Y FUNCIONALIDAD DE LA PRÁCTICA

5.1 DISEÑO

Para gestionar la información principal de la práctica utilizaremos tres clases y un enumerado:

- `GestionarFiguras`: se encargará de guardar un conjunto de figuras por su nombre, para poder consultarlas, recuperarlas o eliminarlas más adelante. Para que estas operaciones sean eficientes se ha decidido utilizar una `Hashtable`.

- `Figura`: se encargará de representar una figura compuesta por una secuencia de trazos. Para representar la secuencia utilizaremos una lista. Entre las operaciones de manipulación se encuentran tres: *homotecia2*, *fusionar* e *insertar* por posición que requieren insertar varios trazos en la lista. Por lo tanto, interesa que esta operación sea eficiente. Vamos a analizar si es mejor utilizar un `ArrayList` o un `LinkedList`.
    - Fusionar es añadir varios trazos al final. Es eficiente en las dos y prácticamente, el mismo tiempo.
    - Homotecia2 requiere insertar un nuevo trazo por cada trazo existente, es decir, una inserción en medio de la lista. Por lo tanto, es más eficiente un `LinkedList`.
    - Insertar por posición: requiere 1) posicionarse en el trazo de la posición *pos* y 2) insertar todos los trazos de segunda figura. El coste total es la suma de las dos operaciones. Siendo *n* y *m* el número de trazos de la primera figura y segunda, respectivamente, el tiempo de ejecución para cada implementación de listas sería el siguiente:
        - `ArrayList`: O(1) + O(m) en el mejor caso, que es insertar al final
          O(1) + O(n*m) en el peor caso, que es insertar al principio
        - `LinkedList`: O(1) + O(m) en el mejor caso, que es insertar al principio
          O(n) + O(m) en el peor caso, que es insertar al final

      Como se puede comprobar en el análisis anterior, el mejor caso de uno es el peor del otro y al revés. ¿Cuál seleccionar? En este caso seleccionaremos LinkedList porque en el peor caso, el crecimiento de O(n) + O(m) es más lento que el de O(n*m). Si resumimos que m es igual a n, entonces, uno es lineal O(n) y el otro $O(n^2)$

- `Trazo`: representa un trazo que puede tener una orientación del `Tipo` (➔,🡻,🡸,🡹) representado mediante las letras D, B, I y S.

La Figura 3 muestra el diseño mediante un diagrama de clases UML.

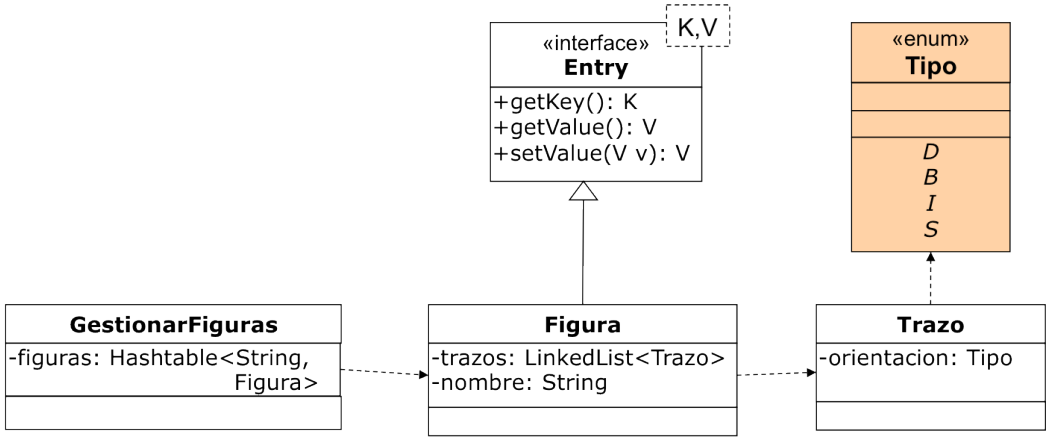

Figura 3. Diagrama de clases para gestionar figuras de trazos hechos a mano

5.2 FUNCIONALIDAD

El fichero *figuras.zip* contiene el proyecto con la funcionalidad a implementar en las clases que contiene la carpeta 'src'.

Nota: los programadores de la funcionalidad, para la comparación de figuras o trazos no podrá utilizar la comparación entre sus representaciones de cadena de caracteres.

5.3 VALIDACION

Mientras la mitad del grupo se encarga de implementar **parte** de la funcionalidad, la otra mitad se tendría que encargar de la validación, sin conocer cómo lo están implementando. Para ello, la forma de trabajar sería:

1) Pensar en varios casos de uso (extremos y generales) para cada funcionalidad que se puede ejecutar.
2) Implementar los test unitarios utilizando JUnit 4 en la carpeta 'tests'. Recomendable utilizar el asistente de Eclipse. Seleccionar la clase y los métodos que se van a validar. Genera una estructura de clase parecida al del ejemplo TrazoTest.java, que luego hay que implementar los casos de prueba que se han pensado.
3) Cuando se tenga la implementación de la funcionalidad, juntar las dos partes y subirlo a Web-CAT.
    a. **Nota**: El 'user' es el mismo usuario que utilizáis en el LDAP de la Universidad y el 'password' vuestro DNI. No os olvidéis de asociar a vuestros compañeros cómo 'Partner'.
4) Analizar el informe de Web-CAT, ver 'Results from Running your tests'.
5) Si alguno de los test fallan:
    a. El grupo que ha escrito la validación, comprueba que los test están bien escritos.
    b. Si el anterior está bien, el otro grupo tendría que revisar el código escrito y corregirlo. En este caso, ejecuta los tests en Eclipse hasta que los tenga bien.
    c. Una vez corregido el código, volver a juntarlo y subirlo a Web-CAT.
6) Subir el proyecto a Web-CAT y analizar el informe de la validación. Encontraréis pistas de cómo mejorar vuestro test unitario en 'Code Coverage from your Tests'.

Una vez se haya conseguido la validación, intercambiar los roles para la otra parte de la funcionalidad.

Nota: los programadores de los tests, para la comparación de figuras o trazos pueden utilizar la comparación entre sus representaciones de cadena de caracteres.

6 BIBLIOGRAFÍA

1. JUnit. http://www.junit.org
2. Tutorial de JUnit, ver 'Material complementario' en eGela.
3. eGela. http://egela.ehu.es
4. Web-CAT. http://lsi.vc.ehu.es/Web-CAT/WebObjects/Web-CAT.woa
5. Resumen de UML, ver 'Material complementario' en eGela.
6. Java. https://www.oracle.com/java/index.html
7. JavaDoc, ver 'Material complementario' en eGela.

8. Eclipe. http://www.eclipse.org