Ampliación de Bases de Datos
Año 2014/2015
Grado en Ing. Informática
Fdi - UCM

Práctica 2

Crucigramas con *Hibernate ORM*

Fecha de entrega: 3 de mayo de 2015.
Porcentaje en la calificación total de prácticas: 20 % (0.60 puntos de la nota final de la asignatura).

Esta práctica se divide en cinco apartados, de los cuales son obligatorios los tres primeros: diseño orientado a objetos, anotación de las clases y acceso a base de datos (consultas sencillas). Con ellos se puede obtener la calificación numérica máxima de 4 sobre 10 (APTO). Para optar a una nota superior se deberán implementar las partes opcionales de la práctica. La calificación máxima varía según los apartados de la práctica implementados:

- Apartados 1, 2 y 3: Calificación máxima de 4/10 (0.24 puntos en la nota final de la asignatura).
- Apartados 1, 2, 3 y 4: Calificación máxima de 6/10 (0.36 puntos en la nota final).
- Todos los apartados: Calificación máxima de 10/10 (0.60 puntos en la nota final).

A la hora de entregar la práctica ten en cuenta lo siguiente:

- La resolución de cada apartado se valida mediante una serie de casos de prueba que se proporcionarán en el Campus Virtual. Los apartados que no hayan sido validados por sus casos de prueba correspondientes no serán tenidos en cuenta para la evaluación de la práctica.
- Además de la validación de los tests, se valorará: el correcto diseño orientado a objetos, su correspondencia con el modelo relacional, y la validez de las consultas realizadas.
- Evita el uso de caracteres no estándar en el código fuente Java (tildes, caracteres ñ y ç).
- Es obligatorio el uso de *Maven*, independientemente del entorno de programación que se utilice.

En esta práctica continuaremos con la base de datos de crucigramas vista en la práctica anterior. En esta ocasión nos centraremos en la información de crucigramas, palabras y etiquetas, sin atender al resto de componentes (usuarios, historial, peticiones, etc.). La práctica tiene como objetivo la implementación de funciones de creación, búsqueda de crucigramas y búsqueda de palabras a partir de sus etiquetas. Para ello utilizaremos la librería *Hibernate*.

Los siguientes apartados tienen como objetivo implementar algunas funciones de acceso a datos siguiendo el patrón DAO (*Data Access Object*[1]). En esta práctica no se exige el desarrollo de ninguna interfaz gráfica. Utiliza el proyecto plantilla proporcionado en el *Campus Virtual* para la realización de la práctica.

[1] Ver: `http://es.wikipedia.org/wiki/Data_Access_Object`

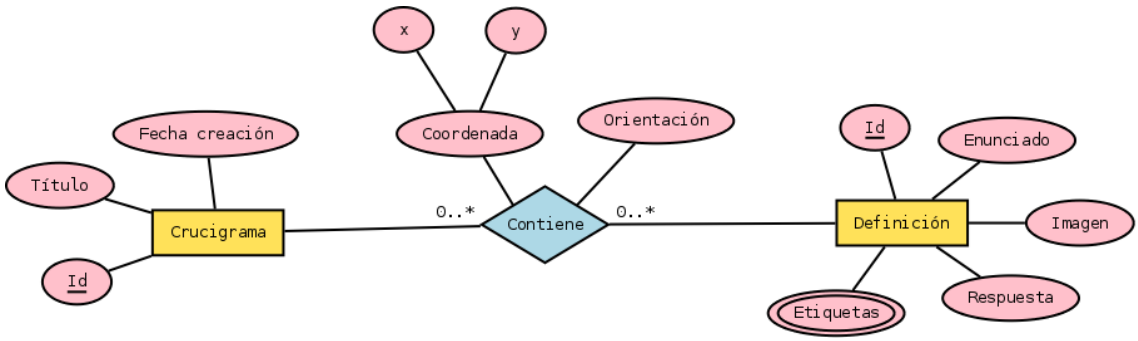

Figura 1: Extracto del diagrama ER de la práctica anterior

Importante: No se permite la modificación de ninguna de las clases proporcionadas en el proyecto plantilla. Para realizar la práctica tan sólo es necesario añadir clases nuevas.

1 Diseño orientado a objetos

En esta práctica modelaremos el diseño de la BD de crucigramas mostrado en la Figura 1.

1. Diseña las clases necesarias para representar las entidades y la relación mostrada en la Figura 1. Las clases han de estar contenidas en el paquete `es.ucm.abd.practica2.model`. No es necesario que introzduzcas aún las anotaciones de *Hibernate*. Además de los métodos de acceso y modificación correspondientes, debes añadir los métodos necesarios para que se puedan realizar las operaciones mostradas en el punto siguiente.

2. En la carpeta `src/test/java` de la plantilla proporcionada existe una interfaz paramétrica llamada `AbstractCrosswordFacade<C,W>`. El tipo `C` hace referencia a la clase que representa un crucigrama, mientras que el tipo `W` hace referencia a la clase que representa una definición. Dentro de esta carpeta, añade una clase que implemente `AbstractCrosswordFacade<`*ClaseCrucigrama*`,` *ClaseDefinición*`>`, sustituyendo los parámetros *ClaseCrucigrama* y *ClaseDefinición* por los nombres de las clases que has creado en el punto anterior para modelar los crucigramas y las definiciones, respectivamente. Implementa los métodos de dicha interfaz[2]:

   `newCrossword(titulo, fecha)` Devuelve un nuevo crucigrama con el título y fecha dados.

   `newDefinition(respuesta, enunciado, etiquetas)` Devuelve una nueva definición con la información pasada como parámetro.

   `addWordToCrossword(crucigrama, definición, fila, columna, orientación)` Añade una definición a un crucigrama con la posición y orientación indicadas.

   `getAnswerOfWord(definición)` Devuelve la respuesta de una definición (es decir, la palabra definida).

   `getTagsOfWord(definición)` Devuelve un array con las etiquetas de una definición.

   `getHintOfWord(W word)` Devuelve el enunciado de una definición.

[2]Utiliza la opción *Source* → *Override/Implement methods* de Eclipse

`getTitleOfCrossword(crucigrama)` Devuelve el título del crucigrama dado como parámetro.

`getDateOfCrossword(crucigrama)` Devuelve la fecha de creación del crucigrama dado.

`getWordsOfCrossword(crucigrama)` Devuelve las palabras contenidas en un crucigrama. Lee atentamente la documentación de la interfaz para conocer el formato en el que se debe devolver esta información.

En cuanto el método restante (`createDAO`), de momento impleméntalo para que devuelva `null`. Posteriormente volveremos a este método.

Importante: Estos métodos no deben realizar ninguna operación de acceso a la base de datos. Tan solo han de llamar a los métodos de acceso de las clases creadas en el primer punto de este apartado. La clase creada en este punto no debe contener ningún atributo.

3. A continuación vamos a validar, mediante los tests proporcionados, la implementación de las operaciones anteriores. En la plantilla (carpeta `src/test/java`) encontrarás una clase llamada `CrosswordTestBase<C,W>`, donde las variables `C` y `W` tienen el mismo significado que antes. Esta clase contiene los casos de prueba. No se puede ejecutar directamente los tests de esta clase desde *Eclipse*, ya que la clase es abstracta. Hemos de crear antes una nueva subclase de `CrosswordTestBase<`*ClaseCrucigrama*, *ClaseDefinición*`>`. Puedes llamar a esta nueva clase como quieras, pero es importante que su nombre acabe en `Test` (por ejemplo, `CrosswordTest`).

   Dentro de esta nueva clase hay que implementar un método `buildFacade`, que deberá crear una instancia de la clase fachada implementada en el punto anterior y devolverla. Una vez hecho esto, podrás ejecutar la clase con *jUnit*, que lanzará todos los casos de prueba heredados.

   En este punto de la práctica deberán poder validarse los siguientes tests:

   ```
   checkWordModelSequences
   checkWordModelDescriptions
   checkWordModelTags
   checkCrosswordTitle
   checkCrosswordDate
   checkCrosswordWords
   ```

2 Anotación de clases

En este apartado estableceremos la correspondencia entre los atributos de las clases creadas en el apartado anterior y la base de datos. Para ello:

1. Crea una base de datos vacía con *phpMyAdmin* con el nombre `Practica2`.
2. Introduce las anotaciones de *Hibernate* necesarias en las clases creadas en el primer punto del apartado anterior. Añade los identificadores numéricos que sean necesarios. En esta práctica es muy recomendable no utilizar claves compuestas.
3. Modifica el fichero de configuración `hibernate.cfg.xml` (carpeta `src/main/resources`) y añade los elementos `<mapping>` necesarios. Recuerda que toda clase anotada con `@Entity` ha de aparecer en este fichero.

4. Vuelve a ejecutar los tests. Además de los anteriores, deberá poder validarse el siguiente test:

   `setUpSessionFactory`

   Observa en *phpMyAdmin* la base de datos creada y comprueba que es similar a las tablas correspondientes de la práctica anterior. Si no fuese así, es recomendable eliminar las tablas creadas antes de volver a ejecutar los tests, ya que algunas tablas podrían pervivir entre distintas ejecuciones.

3 Acceso a la base de datos. Consultas sencillas.

Una vez establecida la correspondencia, utilizaremos las funciones de `Hibernate` para escribir y leer información de la base de datos.

1. En el paquete `es.ucm.abd.practica2.dao` se encuentra la interfaz `AbstractCrosswordDAO<C,W>`. Crea una clase que extienda esta interfaz (de nuevo, sustituyendo `C` y `W` por las clases correspondientes de crucigramas y definiciones). Esta clase ha de contener un constructor sin parámetros y un atributo privado de tipo `SessionFactory`. Implementa el método `setSessionFactory`, que establecerá el valor de este atributo al `SessionFactory` pasado como parámetro. Los casos de prueba se encargarán de crear el objeto `SessionFactory` y pasárselo a este método antes de realizar cualquier operación con la base de datos.

2. Implementa los métodos `insertWord` e `insertCrossword`, que se encargan de añadir una definicion y un crucigrama, respectivamente, a la base de datos. Ten en cuenta que una palabra es una entidad por si misma que puede pervivir independientemente de su pertenencia a un crucigrama, por lo que al insertar un crucigrama no es necesario (ni recomendable) añadir a la BD las palabras contenidas dentro de este crucigrama. Eso sí, el crucigrama ha de manejar la relación de pertenencia que hay entre éste y las palabras que contiene (incluyendo los atributos de posición y orientación). Para ello tendrás que utilizar la opción `cascade` vista en clase.

3. Antes de ejecutar los tests, hemos de implementar la función `createDAO` de la clase fachada que has realizado en el punto 2 del primer apartado de la práctica. Mientras que esta función devolvía antes `null`, ahora tiene que construir y devolver una instancia de la clase DAO creada en este apartado.

   Una vez implementadas estas funciones correctamente, vuelve a ejecutar los casos de prueba. Debería ejecutarse correctamente el siguiente método de prueba:

   `setUpDB`

   Comprueba en *phpMyAdmin* que se ha introducido información en todas las tablas de la BD.

4. Implementa la función `findCrosswordById` de la clase del DAO. Esta función debe recuperar un objeto crucigrama de la BD a partir de su identificador numérico. Vuelve a ejecutar los tests. Se deberán validar los siguientes:

   `checkCrosswordTitleAfterInsertion`
   `checkCrosswordDateAfterInsertion`
   `checkCrosswordWordsAfterInsertion`

5. Implementa la función `getCrosswordData` del DAO. Esta función recibe una cadena `str` y devuelve la siguiente información para aquellos crucigramas cuyo título *contenga* la cadena `str`.

   - Identificador del crucigrama.
   - Título del crucigrama.
   - Fecha de creación del crucigrama.
   - Número de palabras del crucigrama.

   Utiliza, para ello, una sentencia HQL que contenga la cláusula `GROUP BY`. La función debe devolver una lista de arrays. Cada uno de estos arrays ha de tener cuatro elementos: identificador, titulo, fecha y número de palabras. Tras esto, se deberán ejecutar correctamente los siguientes métodos de prueba:

   ```
   checkCrosswordByNameNotFound
   checkCrosswordByName
   checkCrosswordByNameOther
   checkCrosswordBoth
   ```

   Importante: La consulta HQL ha de ser parámetrica. Utiliza marcadores anónimos (?) o con nombre (`:nombre`). Si utilizas marcadores anónimos, recuerda que en *Hibernate* se empiezan a numerar desde el cero, al contrario que en JDBC.

4 Búsqueda por etiqueta.

Implementa el método `findWordsByTags` de la clase DAO. Este método recibe un array de etiquetas (`String`) y devuelve las palabras de la BD que contengan todas las etiquetas del array. Si se recibe un array de etiquetas vacío se deben devolver todas las palabras. Utiliza una consulta HQL con la ayuda de `MEMBER OF`. Los siguientes tests deberán ser aceptados:

```
checkFindByTagsFuncional
checkFindByTagsEmpty
checkFindByTagsTwo
checkFindByTagsNone
```

5 Búsqueda por encaje

En algunas ocasiones disponemos de una serie de palabras en un crucigrama y queremos añadir, en una determinada posición, una palabra que encaje con las ya existentes dentro del crucigrama. Supongamos la situación de la Figura 2, en la que tenemos varias palabras representadas en el crucigrama y queremos añadir una palabra horizontal que comience por la casilla marcada en verde. Las palabras `MAR`, `CARLOTA` y `PALCO` encajarían perfectamente en el crucigrama de la figura. Sin embargo, la palabra `ABETO` no encajaría, ya que la segunda letra de esta palabra colisiona con la primera A del crucigrama. Tampoco nos valdría la palabra `CALVOS` que, pese a que no contradice ninguna de las letras ya existentes del crucigrama, acabaría en una casilla adyacente a la A de más a la derecha, por lo que la palabra que se mostraría en el crucigrama sería `CALVOSA` (Figura 3).

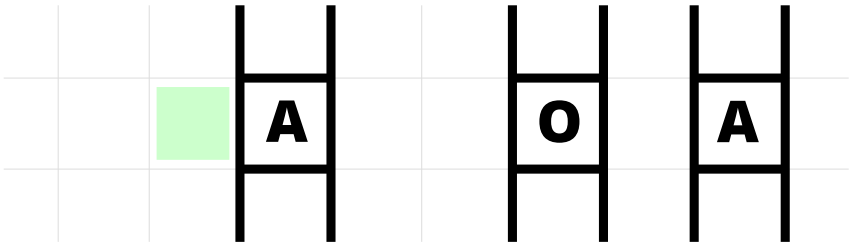

Figura 2: Búsqueda de palabra que encaje en un crucigrama existente.

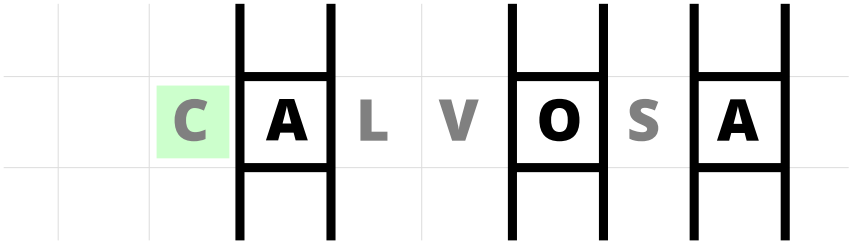

Figura 3: Palabra incorrecta: CALVOS.

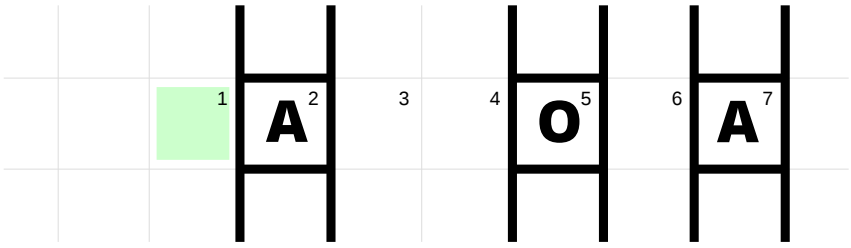

Figura 4: Numeración de las casillas del crucigrama.

Para representar las posibles restricciones que se nos presentan al insertar en una palabra en un crucigrama, numeramos las posiciones de la palabra comenzando desde la número 1 (ver Figura 4). Podemos representar cada restricción como un par (*posición*, *carácter*). Por ejemplo, la situación de la Figura 2 se representa mediante un array de pares `[(2,'A'), (5,'O'), (7,'A')]`. Nosotros representaremos cada uno de estos pares como una instancia de la clase `CharConstraint` (paquete `es.ucm.abd.practica2.dao`), que contiene dos atributos: `position` y `character`.
El objetivo de esta última parte de la práctica consiste en implementar la función `getMatchingWords` del DAO que, dado un array de restricciones `CharConstraints`, busca aquellas palabras en la base de datos que encajen con estas restricciones, en el sentido explicado anteriormente. El método ha de ejecutar una única sentencia HQL. Para obtener el carácter situado en una determinada posición de una cadena puedes utilizar la función `SUBSTRING`. Puedes comprobar la correcta implementación de la práctica utilizando los siguientes tests:

`checkMatchingOneChar`
`checkMatchingTwoChars`
`checkMatchingThreeChars`
`checkMatchingNone`
`checkMatchingNoConstraints`

Si el método `getMatchingWords` recibe un array vacío, deberá devolver todas las palabras de la BD.