Calidad de los Sistemas Informáticos, 2018-2019
Práctica 6 – Entidad relacionada

Objetivo

El objetivo de esta práctica es la ampliación del proyecto creado en las prácticas anteriores para incluir una entidad relacionada que tipifique la entidad principal.

Para ello se requerirá añadir esta nueva entidad en la base de datos, incluir el campo de relación correspondiente, crear la clase que la represente e incluir una lista desplegable en los formularios, con objeto de contener su valor. Se empleará una clase modelo, que hereda de la clase de Java `AbstractListModel` y que se encarga de poblar automáticamente un componente `JComboBox`, evitando el trabajo de hacerlo a mano.

Requisitos previos

Para la realización de esta práctica se requieren los recursos y resultados de las prácticas previas.

Criterios sintácticos

- `Courier`: código fuente, nombres de archivos, paquetes, entidades, atributos, tablas y campos.
- *Cursiva*: términos en otro idioma.
- Negrita: contenido resaltado.
- **$**Entre dólar**$**: contenido no textual, generalmente a decidir por el desarrollador.

Instrucciones previas

- En aquellas partes del código en la que se consideren adecuadas precondiciones (generalmente, al inicio de cada método), que deberán indicarse mediante el comentario hasta el último punto de esta práctica, en el que se propone la implementación de las mismas.

```
// TODO: Preconditions
```

- En esta fase sigue requiriéndose avisar al usuario en caso de error o excepción. Para ello, todas las excepciones y errores deben mostrar un mensaje en pantalla mediante el uso de `JOptionPane`, de la forma en la que se muestra, garantizando que el mensaje de error es suficientemente descriptivo como para garantizar cierto grado de fiabilidad de la aplicación.

```
JOptionPane.showMessageDialog(null,
    "mensaje", "Error", JOptionPane.ERROR_MESSAGE);
```

Pablo de la Torre Moreno
Departamento de Ingeniería Informática
18 de noviembre de 2018

Procedimiento

1. Construcción de la tabla

1. Crear en la base de datos la tabla **$**`TipoEntidad`**$** (o similar: `Raza`, `Color`, `TipoArma`), que tipifique[1] algún aspecto de **$**`Entidad`**$**, tal y como se pide en la práctica 1.
2. Incluir manualmente al menos tres registros en dicha tabla **$**`TipoEntidad`**$**.
3. Crear el campo de relación correspondiente en la tabla **$**`Entidad`**$**, llamado `Id_`**$**`TipoEntidad`**$**, permitiéndose que sea `null` -o no será posible crear dicho campo si ya existen registros en la tabla-. El orden de campos debe ser `Id`, `Id_`**$**`TipoEntidad`**$**, y luego el resto de los campos. De no ser así, deben reordenarse.
4. Para cada registro de la tabla, introducir manualmente el valor de `Id_`**$**`TipoEntidad`**$**, que debe existir como valor en el campo `Id` de **$**`TipoEntidad`**$**; es decir, que no debe haber ningún valor en `Id_`**$**`TipoEntidad`**$** que no exista como `Id` en **$**`TipoEntidad`**$**.
5. Una vez insertado los valores, reconfigurar `Id_`**$**`TipoEntidad`**$** para eliminar la posibilidad de que sea `null`.
6. Desde "Vista de relaciones", configurar la integridad referencial de `Id_`**$**`TipoEntidad`**$** con el campo `Id` de la tabla **$**`TipoEntidad`**$**, con `ON DELETE` y `ON UPDATE` a `RESTRICT`. Esto hace que sea imposible que se introduzcan valores en `Id_`**$**`TipoEntidad`**$** que no se encuentren en `Id` de **$**`TipoEntidad`**$**, y que puedan eliminarse registros de **$**`TipoEntidad`**$** que estén referenciados. Comprobar que esto es así observando el error que devuelve el sistema:
    a. al intentar actualizar mediante `UPDATE` un valor en `Id_`**$**`TipoEntidad`**$** que no exista entre los identificadores de **$**`TipoEntidad`**$**;
    b. al intentar eliminar mediante `DELETE` un registro de **$**`TipoEntidad`**$** cuyo valor de `Id` se encuentre en `Id_`**$**`TipoEntidad`**$**.
7. Exportar la base de datos con el nuevo cambio y actualizar el contenido de la que está en el proyecto.

2. Construcción de la clase

1. Crear la clase de mapeo equivalente para **$**`TipoEntidad`**$** dentro del (sub)paquete `data`, creando tantas variables privadas y métodos de acceso (`get` y `set`) adecuados relativos a los campos de la tabla **$**`TipoEntidad`**$**.
2. Implementar un constructor para dicha clase y que contenga un parámetro `int iId`. Dicho constructor deberá obtener el registro correspondiente a dicha clave primaria y rellenar las variables privadas.
3. Implementar un método estático `Select()` similar al creado en práctica anteriores, pero esta vez sin parámetros y, por tanto, sin condiciones; es decir, que devuelva una lista (`ArrayList`) de todas las instancias existentes en la tabla **$**`TipoEntidad`**$**

[1] Lo normal es que una entidad tipificadora tenga un campo `Id` y otro `Nombre`. En ocasiones, se requieren más campos, por ejemplo, si hay atributos de detalle o de relación -por ejemplo, con la provincia, en una localidad-.

ordenadas por el campo más identificativo para el usuario (el nombre, por ejemplo, de haberlo).

4. Implementar el método `toString()` y que funcione de manera similar al de **$**`Entidad`**$**, tal y como se describe en la práctica 2.
5. En el paquete `test`, construir la clase JUnit **$**`TipoEntidad`**$**`Test`, basándose en la ya existente **$**`Entidad`**$**`Test`. Implementar los métodos de prueba para su constructor y para `Select`.
6. Ejecutar las pruebas con cobertura de código hasta su correcto funcionamiento.

3. Inclusión del tipo en la entidad fuerte

1. En la clase **$**`Entidad`**$**, crear la variable y métodos de acceso para **$**`TipoEntidad`**$**. La variable se declarará como `private` **$**`TipoEntidad`**$** `_`**$**`tipoEntidad`**$**, siendo los métodos de acceso del mismo tipo.
2. Modificar el constructor de **$**`Entidad`**$** para que cargue el valor de dicha variable, para lo cual deberá llamar al constructor de **$**`TipoEntidad`**$**.
3. Modificar el método `Create`, para que reciba **$**`TipoEntidad`**$** -tal cual, no su identificador- como parámetro adicional, incluyéndose en el `INSERT` campo de relación hacia ésta.
4. Modificar el método `Select` de **$**`Entidad`**$**, de forma que reciba un parámetro más, `String s`**$**`TipoEntidad`**$**, que contenga el nombre (o parte del nombre) del tipo, incluyéndolo en su caso en la condición de búsqueda. Será necesario modificar también el método `Where`, en su caso. Debe recordarse que, si se incluye en la condición de búsqueda (es decir, si dicho parámetro no es `null` ni vacío), deberá añadirse también a la consulta la pertinente instrucción `JOIN`. No olvidar tampoco poner el nombre de la tabla como prefijo en todos aquellos campos que puedan prestarse a confusión por duplicación en ambas tablas (por ejemplo, `SELECT` **$**`Entidad`**$**`.Id`, en vez de `SELECT Id`).
5. Modificar el método `Update`, para que en la consulta actualice también el valor de la variable de tipo **$**`TipoEntidad`**$** (es decir, `Id_`**$**`TipoEntidad`**$**).
6. Modificar el método `toString()` de **$**`Entidad`**$** para que aparezca entre los campos (separados por punto y coma) el resultado del método `toString()` del tipo.
7. En este punto no se pueden desarrollar pruebas aún, ya que la modificación habrá generado errores en el resto del código.

4. Inclusión del tipo en la interfaz de detalle

1. Como se explica en los siguientes pasos, incluir en el formulario de detalle de la entidad principal una lista desplegable con los nombres de las posibles instancias del tipo, de tal manera que, seleccionando un nuevo nombre y guardando, se modifique la referencia en la tabla de la base de datos[2]. Esto debe pasar por usar también el correspondiente método `set`.
2. Para ello y de forma similar a como se realizó con la tabla de resultados del formulario de búsqueda, crear en el paquete `gui` una clase **$**`TipoEntidad`**$**`ListModel` que

[2] No debe olvidarse que en la base de datos se trabaja con identificadores, no con nombres ni instancias.

herede de `javax.swing.AbstractListModel<$TipoEntidad$>` e implemente la interfaz `javax.swing.ComboBoxModel<$TipoEntidad$>`.

```
public class $TipoEntidad$ListModel
      extends AbstractListModel<$TipoEntidad$>
      implements ComboBoxModel<$TipoEntidad$>
```

3. Crear en dicha clase una variable privada de tipo `List<$TipoEntidad$>` llamada `_aData`.
4. Crear otra variable privada de tipo `Object` llamada `_oSelectedItem`, a `null` por defecto, y que contendrá la referencia al elemento seleccionado en la lista desplegable. Crear sus métodos de acceso `get` y `set`.
5. Crear un constructor que reciba un parámetro del mismo tipo e iguale `_aData` a dicho parámetro.
6. Implementar los métodos sobrescritos `getElementAt(int iIndex)` y `getSize()`, de forma similar a como se hizo en la práctica anterior con `JTable`.
7. Crear un control `JComboBox` llamado `cmb$TipoEntidad$` en el formulario de detalle `Ifr$Entidad$`, con su correspondiente `JLabel`.
8. Buscar en el código del formulario la instanciación de la lista desplegable y cambiar su tipo por `JComboBox<$TipoEntidad$>`, y añadir, justo debajo de su creación,

   ```
   cmb$TipoEntidad$.setModel(
         new $TipoEntidad$ListModel($TipoEntidad$.Select()));
   ```

   con lo que se asignará un nuevo modelo creado con el resultado de invocar al método `Select` que se implementó en el punto 2.3 y que obtiene todos los posibles registros de la tabla.
9. Solventar el resto de los errores de la aplicación, derivados de la inclusión del nuevo elemento en los parámetros de `$Entidad$`, teniendo en cuenta que el elemento seleccionado en la lista desplegable se obtiene invocando a `cmb$TipoEntidad$.getModel().getSelectedItem()`. Por ahora, no debe modificarse la interfaz del formulario de búsqueda, por lo que habrá que usar `null` como parámetro de `Select`, cuando corresponda.
10. Ejecutar el programa y abrir un nuevo elemento. Abrir la lista desplegable para ver lo que aparece. Cambiar el método `toString()` de `$TipoEntidad$` para que aparezca únicamente el nombre y probar de nuevo.
11. Enviar en los métodos invocados a "Guardar" el valor obtenido de `cmb$TipoEntidad$.getModel().getSelectedItem()`, tanto en `Create` (que ya debería de estar hecho) como en los métodos `set` previos a `Update`.
12. Añadir en dicho código que no se permita guardar si no hay ningún elemento de la lista desplegable seleccionado (es decir, que el resultado de consultar el elemento seleccionado es `null`), mostrando el correspondiente error.
13. Probar (viéndolo directamente en la base de datos) que inserta correctamente, y que posteriormente guarda los cambios.

4. Inclusión del tipo en la interfaz de búsqueda

1. Tal y como se explica a continuación, incluir en el formulario de búsqueda un control de lista desplegable de tal forma que busque también la entidad principal en función del valor del nombre de su tipo (escrito manualmente o seleccionándolo).
2. En la parte superior del formulario `Ifr$Entidades$`, añadir un campo de búsqueda más, `cmb$TipoEntidad$` (con su `JLabel` correspondiente) poniendo en sus características (sección [Properties]) el valor de `editable` a `true`.
3. Aprovechando el modelo que se creó anteriormente, buscar en el código del formulario la instanciación de la lista desplegable, cambiar el tipo a `JComboBox<$TipoEntidad$>` y añadir, justo debajo de su creación,

   ```
   cmb$TipoEntidad$.setModel(
         new $TipoEntidad$ListModel($TipoEntidad$.Select()));
   ```

4. En el código de "Buscar", enviar a la llamada a `Select`, en el parámetro que corresponda al tipo, el texto de la caja: `cmb$TipoEntidad$.getSelectedItem().toString()`, siempre y cuando dicho elemento seleccionado no sea `null`, en cuyo caso se enviará `null`.
5. Corregir posibles errores del proyecto y ejecutar para comprobar que hace las búsquedas correctamente.
6. Cambiar como corresponda el código de los métodos de la clase `$Entidades$TableModel` para que aparezca el nombre de la entidad `$TipoEntidad$` relacionada en la lista de resultados.
7. Modificar el constructor del formulario de detalle `Ifr$Entidad$` para que cargue el valor indicado en la lista desplegable, usando el método `setSelectedItem` del modelo. Ya se adelanta que, aunque aparentemente parece correcto, esto no va a funcionar exactamente como debería; el motivo, relacionado con el comportamiento del método `setSelectedItem`, debe razonarse y si fuera necesario consultar con el docente.

5. Trabajo adicional opcional

1. Completar la clase `$TipoEntidad$` con los métodos `Create`, `Update`, `Delete` y `Select` (sobrecargada con parámetros), o pasar a la fase de pruebas, en otro caso.
2. Revisar todas las clases de prueba, completar los métodos que faltan y volver a ejecutarlas mediante cobertura de código.