Reto 2.1 CRUD desde Java sobre BBDD relacional

Enlace a GitHub:

1. Implementar los métodos vacíos de la clase ModeloAlumnosJDBC en el proyecto gestionAlumnos para implementar el CRUD de la entidad Alumno. (Rubén)

En el documento pasado llamado gestionAlumnos.zip en la clase ModeloAlumnoJDBC aparecen distintos Overrides que te piden:

1. GetAll.

Este método sirve para mostrar la tabla sql que he creado, por lo que he creado una lista que se llame alumnos para almacenar la tabla, luego he creado un String que se llama sql con el código para seleccionar todo el contenido, hay que probar la conexión con la BDD para ver si funciona correctamente si no te saldría un excepción(esto se repite en todas).
Luego he creado un while para que recorra la tabla y meta el contenido en el array para luego mostrarlo.

2. getAlumnoPorDNI

Este método como indica el nombre quieres buscar el alumno por DNI por lo que he creado otro String que se llama sql que se le pase por parámetro un DNI y lo busque en la base de datos. En el caso de que exista se asignará en el objeto Alumno, para comprobar lo que hago es sustituir el parámetro “?” por DNI para que se ejecute la consulta, con el if realizado lee toda la base de datos hasta encontrarlo en caso de encontrar el DNI que queramos te muestra el DNI, nombre, apellidos y código postal.

3. modificarAlumno

Este método al ser boolean te indicará true o false si se ha modificado por lo que tendremos que hacer un String con la consulta para luego cambiar los datos. Entonces habrá que coger desde la primera posición hasta la cuarta porque así están puestos los valores y se tiene que respetar el orden, el último valor sirve para buscar el DNI del alumno que queramos.

4. eliminarAlumno

Este es otro método boolean que en el caso que se pueda eliminar al alumno solicitado te devolverá true y si no puede false.
Tendremos que volver a crear un String sql con la consulta que se

quiera en este caso un “delete”, con el setString establecerá DNI con el parámetro que hayamos pasado.
Por último he creado un int filas para que me cuente a cuantas filas afecta en caso de que me devuelva 1 es que sí existía el DNI y se ha eliminado, si no te devolverá false.

5. **crear**
   Este último método también es boolean por lo que te devolverá true en caso de que se haya creado y false en el caso de que no haya funcionado
   Metemos la consulta que queramos en un String y pasaremos los parámetros que queramos y que se intercambien por donde aparecen las “?”, luego contaré las líneas para ver si ha afectado a alguna fila.

Lo que he sacado en común de todos estos métodos creado es que primero hay que meter en un String la consulta que se quiera en este caso hemos hecho el CRUD y luego comprobar con “Connection”--> getConnection para ver si podemos meternos en la BDD y PreparedStatement para enviar la consulta que hayamos pedido.

Luego tendremos que irnos a ControladorGestionAlumnos a los dos métodos Override que tenemos:

1. **actionPerformed**
   Este método sirve para cuando empiezas en la interfaz se ejecute, he creado un switch case para que te devuelvan los métodos que he creado.

2. **valueChanged**
   Se ejecuta automáticamente cuando el usuario selecciona un elemento de una lista.
   e.getValueIsAdjusting() sirve para evitar que el evento se dispare múltiples veces mientras el usuario está seleccionando.
   a. Obtiene el valor seleccionado de la lista (view.jListaAlumnos.getSelectedValue()).
   b. Usa ese valor (el DNI) para buscar el objeto Alumno en el modelo (model.getAlumnoPorDNI(dni)).
   c. Luego carga los datos del alumno en los campos de texto de la vista (cargarAlumno(a)).

Por último he creado un Main para que se puedan ejecutar los métodos. Lo que he hecho en crear dos alumnos para luego meterlo con el método crear que he creado, en listar los alumnos he recorrido la lista que he creado anteriormente, para buscar el alumno busco el DNI que quiera y me devuelven los datos, para modificar alumno igual pasas por parametro un DNI y si los modificas te aparece true si no false, eliminar alumno se pasa el DNI como parámetro y si se encuentra

se elimina.

2. Sustituir en cualquiera de los proyectos adjuntos que usan ficheros para persistencia (AgendaGUI, FicheroAgendaV1 o GestorAlumnosFinal) la persistencia en ficheros por persistencia en base de datos. (Rubén)

En nuestro caso hemos seleccionado AgendaGUI por lo que todo lo que he realizado ha sido alrededor de esta clase.

1. AppAgenda
    a. getContactos

    Devuelve una lista con todos los contactos almacenados en la base de datos. Para ello, llama al método getAll() del modelo, que ejecuta la consulta SQL correspondiente y devuelve los resultados. Este método sólo obtiene los datos, no los modifica.

    b. añadirContacto

    Sirve para crear un nuevo contacto y guardarlo en la base de datos. Primero se crea un objeto de tipo Contacto con el nombre y teléfono recibidos por parámetro, y después se llama al método insertar(c) del modelo para añadirlo a la base de datos. Finalmente, el método devuelve el contacto que se ha creado.

    c. editarContacto

    Se utiliza para modificar los datos de un contacto ya existente. Crea un nuevo objeto Contacto con los valores actualizados y llama al método actualizar(c) del modelo, que se encarga de realizar el cambio en la base de datos. Una vez hecho esto, devuelve el contacto modificado.

    d. borrarContacto

    Se encarga de eliminar un contacto de la base de datos. Llama al método borrar(nombre) del modelo, pasando el nombre del contacto que se desea eliminar, y después devuelve un objeto Contacto con el nombre del contacto eliminado y el número de teléfono como valor nulo.

2. modeloAgendaJDBC

El objetivo es permitir que otras clases del programa (como el

controlador AppAgenda) puedan obtener, añadir, modificar o eliminar contactos sin tener que preocuparse por los detalles de la conexión con la base de datos.
En el constructor de la clase, se establece dicha conexión utilizando DriverManager.getConnection, donde se indica la URL del servidor MySQL (jdbc:mysql://localhost:3306/adat2), el usuario (dam2) y la contraseña (asdf.1234).
Si ocurre algún error durante la conexión, se captura mediante un bloque catch y se muestra el error con e.printStackTrace().

a. **getAll** devuelve una lista con todos los contactos que existen en la tabla contactos.
Para ello, crea un Statement y ejecuta una consulta SQL
b. **El método insertar** permite añadir un nuevo contacto a la base de datos. Luego, se ejecuta la actualización con ps.executeUpdate().
c. **El método actualizar** sirve para modificar los datos de un contacto existente.
d. **El método borrar** se encarga de eliminar un contacto de la base de datos cuyo nombre coincida con el que se pasa como parámetro.

3. Main

La clase Main se encarga de inicializar la aplicación creando los dos componentes principales:

a. **El controlador (AppAgenda)**, que gestiona la lógica del programa y la comunicación con el modelo,

b. **La vista (VistaAgenda)**, que proporciona la interfaz gráfica para que el usuario interactúe con la agenda.

3 . Construir una aplicación básica en Java (CLI o GUI) que haga un CRUD sobre alguna de las bases de datos de ejemplo en el aula virtual, o bases de datos del curso pasado o alguna base de datos que diseñes sobre algún interés particular. (Adrián)

4.Cualquier de las anteriores pero con más de una tabla, al menos una relación entre ellas, un campo de fecha y funcionalidad de búsqueda. (Adrián)

Todos los métodos relativos a las consultas SQL se ubican en la clase EmpleadoCRUD, el POJO es Empleado y la aplicación se ejecutaría en el Main.

Se han realizado ambos apartados en un solo paquete, el programa está basado en una base de datos llamada scott, cuyo script está en el foro de la unidad, aunque se ha usado una versión modificada para admitir nombres y puestos de trabajo con más carácteres ya que al probarlo resultaba muy limitante, este .sql se encuentra en el GitHub también, para probar la aplicación dejo aquí algunos datos:

ID: 7369 Nombre: SMITH

ID: 7499 Nombre: ALLEN

ID: 7521 Nombre: WARD

ID: 7566 Nombre: JONES

En cuanto a la reflexión de Statement vs PreparedStatement evidentemente me posiciono a favor del PreparedStatement a la hora de realizar consultas principalmente por el siguiente motivo:

- Seguridad: El PreparedStatement es más seguro que Statement ya que en vez de ejecutar un String concatenando los parámetros (lo que lo hace carne de cañón para una SQL Injection) se puede sustituir estos parámetros concatenados por el carácter ? el cúal se parametriza después haciendo que las consultas estén más protegidas.

Buscando por internet encontré una clase llamada CallableStatement la cúal funciona de manera similar a PreparedStatement (de hecho hereda de esta) funciona también con el uso de ? para parametrizar, la diferencia es que CallableStatement se usa para llamar a procedimientos y funciones evitando una posible SQL Injection, en este reto no es aplicable pero seguramente sea útil próximamente.

Haciendo la tarea he descubierto un método útil perteneciente a la clase PreparedStatement por si tienes campos null y quieres evitar errores varios, este se llama setNull(int parameterIndex, int sqlType) en el primer int se indica la posición de ? A parametrizar y en el segundo se indica el tipo de dato que es, esto se indica con la clase Types, por ejemplo: Types.INTEGER.