**Guia de la conexión a base de datos mediante el uso de la clase Jswing como interfaz**

Métodos Requeridos

**Método C.R.U.D.**

Este método se trata de una plantilla para las funciones básicas a la hora de conectarse a una base de datos[[1]](#footnote-1). El método divide las funciones básicas en cuatro; Create, Read, Update y Delete.

**Create**

**public** **void** **create**(**Departamento** dep) **throws** **SQLException** {

**String** **sql** = "INSERT INTO DEPARTAMENTOS VALUES (?,?,?)";

PreparedStatement **sentencia** = connection.prepareStatement(sql);

sentencia.setInt(1, dep.getDept\_no());

sentencia.setString(2, dep.getDnombre());

sentencia.setString(3, dep.getLoc());

**int** **filas**;//

filas = sentencia.executeUpdate();

**System**.***out***.println("Filas afectadas: " + filas);

}

Este método enviará una String a la base de datos, la cual tendrá una sintaxis acorde con el lenguaje de la base de datos, que en este caso es MySQL. Este método devolverá nada, y lanzará el error que se daría en la base de datos(SQLException) un nivel más arriba, que deberá controlarse a la hora de llamar al método.

Usaremos PreparedStatement: que identifica las “?” de nuestra String y la procesa insertando datos, para después mandarla a la base de datos.

Sentencia de creación: PreparedStatement **sentencia** = connection.prepareStatement(sql);

La sintaxis para introducir un dato es:

<nombre\_var>.set<tipo\_de\_dato>(<posición\_interrogacion>,<dato>);

Tras esto se ejecutaría el Update, que la base de datos automáticamente devuelve el número de datos afectados.

**Read**

El método Read se divide en dos, con el mismo principio, pero manejada la salida de diferente forma, uno para leer todos los Departamentos y otro para leer uno por nombre.

**public** List<Departamento> **ReadAll**() **throws** **SQLException** {

List<Departamento> **deps** = **new** ArrayList<Departamento>();

**String** **sql** = "select \* from departamentos ";

Statement **sentencia** = connection.createStatement();

sentencia.execute(sql);

ResultSet **rs** = sentencia.getResultSet();

**while** (rs.next()) {

**Departamento** **dep** = **new** Departamento(rs.getInt("dept\_no"), rs.getString("dnombre"), rs.getString("Loc"));

deps.add(dep);

}

**return** deps;

}

En este método es importante mencionar que, ya que no se necesita insertar datos en la String sql, no utilizaremos PreparedStatement, sino Statement, que enviara la String sin modificarla.

Para manejar la salida del método se utilizará ResultSet, que devuelve una lista con los resultados, siendo este 1 o más. Crearemos un bucle, para cada objeto del ResultSet coger los valores y crear un departamento, para añadirlo a la lista. Esta lista será lo que devolverá el método.

**public** **Departamento** **Read**(**String** nombre) **throws** **SQLException** {

**String** **sql** = "select \* from departamentos where dnombre= ? ";

PreparedStatement **sentencia** = connection.prepareStatement(sql);

sentencia.setString(1, nombre);

sentencia.execute();

ResultSet **rs** = sentencia.getResultSet();

**Departamento** **dep**=**null**;

**while**(rs.next()) {

dep=**new** Departamento(rs.getInt("dept\_no"), rs.getString("dnombre"), rs.getString("Loc"));

}

**return** dep;

}

En este metodo, parecido al anterior simplemente crearemos un departamento, y lo devolveremos, en vez de devolver una lista.

**Update**

**public** **int** **Update**(**Departamento** dep) **throws** **SQLException** {

**int** **filas**;

**String** **sql** = "Update departamentos set loc =?,dnombre=? where dept\_no=?";

PreparedStatement **sentencia** = connection.prepareStatement(sql);

sentencia.setString(1, dep.getLoc());

sentencia.setString(2, dep.getDnombre());

sentencia.setInt(3, dep.getDept\_no());

filas = sentencia.executeUpdate();

**return** filas;

}

Volveremos a usar PreparedStatement, este metodo es muy parecido al créate, devolviendo el número de datos afectados.

**Delete**

**public** **void** **Delete**(**int** dept\_no) **throws** **SQLException** {

**String** **sql** = "Delete from Departamentos WHERE dept\_no= ?";

PreparedStatement **sentencia** = connection.prepareStatement(sql);

sentencia.setInt(1, dept\_no);

sentencia.execute();

}

Este método solo necesitara el Dept\_no, la cual es la clave primaria. Se utilizara este dato con PreparedStatement para que la SQL query de solo un resultado y este sea borrado.

Métodos Adicionales

En este proyecto se ha utilizado la librería JSwing para crear una interfaz interactiva para poder usar los métodos creados de conexión a la base de datos. Esta explicación se dividirá en: Objetos y Métodos.

**Objetos**

Ahora listare cada objeto utilizado por primera vez y una breve descripción de su utilidad.

* Jpanel:

contentPane = **new** JPanel();

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

contentPane.setLayout(**new** BorderLayout(0, 0));

setContentPane(contentPane);

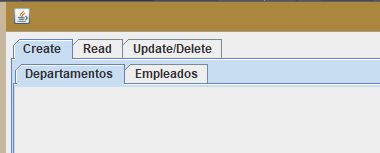
Jpanel es la versión más simple de un espacio vacío, un contenedor. No será el único que se creará a lo largo del programa. Este panel es un objeto, con métodos como setBorder, y setLayout. El único utilizado a conciencia es el setLayout, que define la organización 2d que tendrá los objetos añadidos a este panel.

* JTabbedPane:

**JTabbedPane** **tabbedPane** = **new** JTabbedPane(**JTabbedPane**.***TOP***);

contentPane.add(tabbedPane, **BorderLayout**.***CENTER***);

El JTabbedPane, consiste en un Jpanel, pero con la posibilidad de ver mediante una etiqueta, en que panel se encuentra el usuario.



Aquí se puede apreciar el uso de JTabbedPane. Primero en el mismo nivel jerárquico, están los paneles Create, Read y Update/Delete, y dentro de Create hay dos paneles, Departamentos y Empleados que están un nivel mas abajo que los 3 primeros.

* JTextField:

textField = **new** JTextField();

textField.setColumns(10);

textField.setBounds(116, 145, 209, 20);

panel.add(textField);

Este componente se trata de un rectángulo que permite editar una sola linea de texto. Aquí lo importante son el metodo de setBounds, que marca la posición en el eje X y Y, y después sus dimensiones, en ese orden.

Después añadimos este componente al panel en el que queremos que se encuentre.

* JTextArea:

textArea\_2 = **new** JTextArea();

textArea\_2.setEditable(**false**);

textArea\_2.setBounds(520, 11, 300, 20);

panel\_3.add(textArea\_2);

Muy parecido a JTextField, pero en vez de solo mostrar una linea, esta preparado para tener múltiples líneas de texto plano. Es un derivado de TextArea de la clase AWT, pero tiene diferencias, una de ellas es que si las líneas sobrepasan los limites del componente, este no habilitara el scroll automáticamente, este componente no lo maneja. Para ello pondremos nuestra JTextArea dentro de un ScrollPanel.

* JScrollPane

**JScrollPane** **scrollPane** = **new** JScrollPane(); scrollPane.setBounds(10, 135, 836, 580); panel\_1.add(scrollPane);

Este elemento es un rectángulo con dimensiones, que permite a un componente insertado dentro del jerárquicamente, el sí tiene datos que salen de los límites, habilitar el scroll.

Por ejemplo, aquí introducimos nuestra tabla en un JScrollPane:

scrollPane.setViewportView(table)

* Jtable

table = **new** JTable(); scrollPane.setViewportView(table; table.setFillsViewportHeight(**true**);

table.setEnabled(**false**);

Este elemento es una tabla, básicamente la expresión grafica de un Array bidimensional.

A continuación explicare como se modifica una tabla:

**DefaultTableModel** **model** = **new** DefaultTableModel();

**String**[] **Columnnames** = **new** **String**[] { "Dept\_no", "Dnombre", "Dloc" };

model.setColumnIdentifiers(Columnnames);

table.setAutoResizeMode(1);

**for** (Iterator<Departamento> **iterator** = listadep.iterator(); iterator.hasNext();) {

**Departamento** **depa** = (**Departamento**) iterator.next();

model.addRow(**new** **Object**[] { depa.getDept\_no(), depa.getDnombre(),

depa.getLoc() });

}

table.setModel(model);

Una tabla se manejara mediante un modelo, ya que siempre al inicializarse será un campo con 0 columnas y 0 filas. Nosotros crearemos ese modelo y lo modificaremos a gusto. Primero creamos un Array de String para introducir los nombres, el dato en la posición 0 del Array corresponde con el nombre de la columna 1. Una vez creado se añade al modelo con el método “.setColumnIdentifiers()”.   
Ahora solo vamos creando departamentos con el iterator y los añadimos como filas. Aquí se identifica con un array de objetos, que como en los nombres la posición 0 del Array corresponde con la columna 1. Simplemente creamos el array dentro del constructor en el que añadimos la fila, como se ve en el ejemplo.

Cuando todos los elementos de la tabla que queremos mostrar estén añadidos al modelo que hemos creado, lo añadiremos a la tabla. Aunque la tabla tenga un modelo anterior, este se desechara y se mostrara el nuevo.

* Jlabel

**JLabel** **label\_5** = **new** JLabel("Oficio");

label\_5.setBounds(29, 90, 46, 14);

panel\_3.add(label\_5);

Este elemento es una linea de texto con el mismo comportamiento que JTextField, exceptuando la posibilidad a que reaccione a un input. Esto implica que no podremos interactuar con este campo, solo sirve para mostrar texto plano.

# Estructura de la implementación de la BBDD con JSwing

En este apartado se explicara como se manejan los siguientes eventos introducidos por el usuario, que en esta base de datos se compone de escribir y hacer clic en los botones. También se vera como se manejan las transacciones a la base de datos con JSwing.

El como reacciona JSwing a los eventos es mediante métodos públicos, que están implementados en el código fuente del objeto. Pero estos métodos no funcionarian, ya que no referencian a objetos específicos de nuestra ventana de JSwing ni tienen el código que queremos, que en este caso sería el conectarse a la BBDD.

Por eso usamos esta notación que viene implementada en java: ***@Override*.**

**Esta notación ( toda notación empieza con el carácter “@” y tienen sintaxis diferentes.) hace que el siguiente método se utilice en el caso del evento “MouseEvent** arg0” en vez de utilizar el implementado por defecto.

Entonces, ahora cuando el evento de hacer clic en el botón se envía a la aplicación esta ejecutara el siguiente código:

***@Override***

**public** **void** **mouseClicked**(**MouseEvent** arg0) {

**try** {

depmng = **new** Departamentos();

} **catch** (**SQLException** **e1**) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**Departamento** **dep** = **new** Departamento(**Integer**.*parseInt*(textField\_2.getText()), textField\_1.getText(),textField.getText());

**try** {

depmng.create(dep);

textArea.setText("Departamento creado");

} **catch** (**SQLException** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

textArea.setText(e.getMessage());

}

depmng = **null**;

}

});

Hemos manejado cada conexión de una manera que desconozco si es la óptima, pero si inicializaba una conexión global para cada tabla desde el principio daba error al manejar varias conexiones una tras otra, al no poder operar con transacciones como he aprendido en Hibernate.

Vale, ahora analizando el código vemos como primero manejamos el error de la conexión a la tabla Departamentos. Para crear nuestro departamento, necesitaremos utilizar el texto introducido previamente en los campos de texto JTextField, para esto simplemente llamaremos al objeto en cuestión y utilizaremos el método “.getText() “. Este método nos devolverá el texto en forma de String, pero como vemos , el numero de departamento es un número, no es problema ya que te el método te devuelve una string, y podemos manejarla como tal.

Ya solo sería finalizar el método utilizando el método C.R.U.D. deseado. Tanto como si el método devuelve un error o no, enviaremos un mensaje con un “.setText()” para informar al usuario del estado del programa. Este método simplemente enviara una linea de texto a un componente JTextArea.

1. Esta conexión se ha hecho lo más simple posible con la librería JDBC a MySQL. [↑](#footnote-ref-1)