TECNOLOGICO NACIONAL DE MEXICO



Instituto Tecnológico de la Laguna

Ingenieria en Sistemas Computacionales

**TOPICOS AVANZADOS DE PROGRAMACION**

# Portada

SEMESTRE: Ene - Jun / 2017 GRUPO: “B” 17 – 18 Hrs

PRACTICA No. 5

Aplicacion de base de datos con JavaDB Tarea

ALUMNO:

15131349 José de Jesús Miramontes Gómez

PROFESOR:

Ing. Luis Fernando Gil Vázquez

**Torreón, Coah. a 06 de Junio de 2017**

**Contenido**

[Portada 1](#_Toc484551279)

[Bibliografía 2](#_Toc484551280)

[Ejercicio 1 3](#_Toc484551281)

[Análisis 3](#_Toc484551282)

[Funcionalidades: 3](#_Toc484551283)

[Detalles de conexión a base de datos: 3](#_Toc484551284)

[Esquema de base de datos: 3](#_Toc484551285)

[Tablas: 3](#_Toc484551286)

[Diseño 4](#_Toc484551287)

[Código 4](#_Toc484551288)

[CONNECT.JAVA 4](#_Toc484551289)

[GUI.JAVA 7](#_Toc484551290)

[SPLASHSCREEN.JAVA 8](#_Toc484551291)

[AGENDADECONTACTOS.JAVA 9](#_Toc484551292)

[Prueba de Ejecución 9](#_Toc484551293)

# Bibliografía

Reese, G. (2000). *Database Programming with JDBC & Java, 2nd Edition* (2 ed.). O'Reilly Media. Recuperado el 05 de Junio de 2017, de https://cdn.preterhuman.net/texts/manuals/O'Reilly-Database%20Programming%20with%20JDBC%20and%20Java,%20Second%20Edition\_by\_chimney.Pdf

# Ejercicio 1

Desarrollar una aplicación visual (GUI) basada en Java Swing que utilice conectividad a base de datos mediante JDBC, específicamente a base de datos relacional con JavaDB. La aplicación deberá ser consultada de alguna bibliografía de Java y deberá ser ampliada y/o mejorada si el ejemplo expuesto en la bibliografía es limitado en funcionalidad.

# Análisis

Programa que simula una agenda de contactos almacenada en una base de datos. Capaz de Crear, Leer, Actualizar, y eliminar contactos (CRUD).

## **Funcionalidades:**

* Almacena los campos:
  + **Id\_contacto** (integer).
  + **Nombre** (varchar(30))
  + **Teléfono** (varchar(30))
  + **Correo** (varchar(30))
* Al seleccionar el id, automáticamente se rellenan los datos en los campos para agilizar procedimientos al usuario.
* Evita colisión de ids mostrándole al usuario las acciones que puede realizar.
  + Ej. Si existe un contacto con el id seleccionado el usuario solo será capaz de actualizar los datos o eliminar el registro.
  + Si no existe el id, solo permite insertar datos en la base de datos.

## **Detalles de conexión a base de datos:**

**Nombre:** contactos.

**Usuario:** APP.

**Contraseña:** APP.

**Servidor:** localhost.

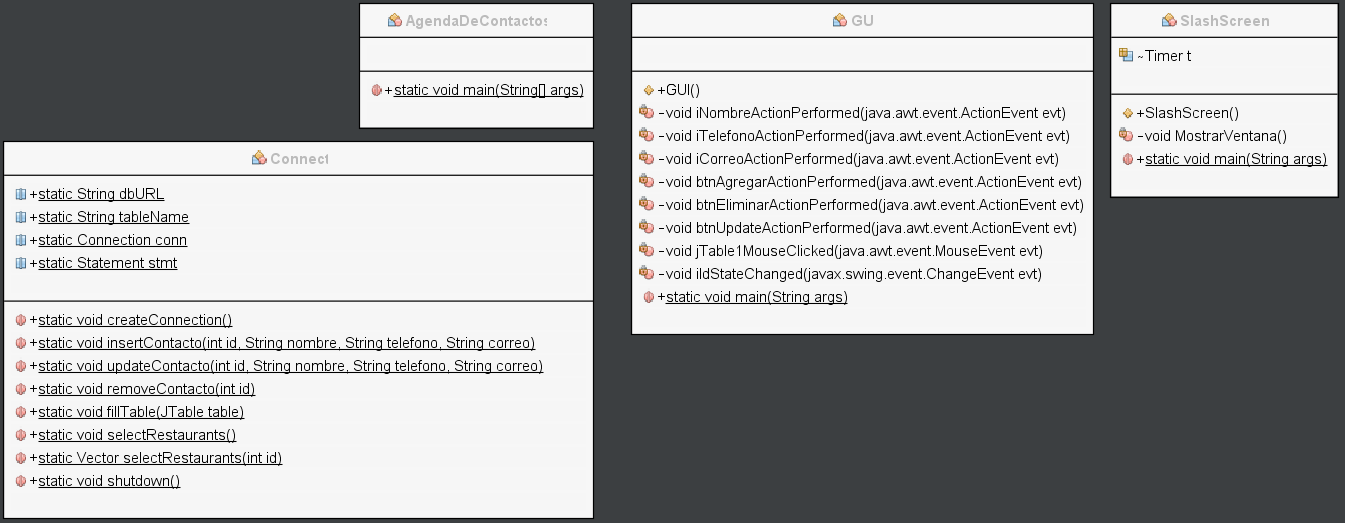
**Puerto:** 1527

## **Esquema de base de datos:**

### **Tablas:**

|  |  |
| --- | --- |
| **ListaContactos** | |
| **Nombre de campo** | **Tipo de dato** |
| Id\_contacto | varchar(30) |
| Nombre | varchar(30) |
| Teléfono | varchar(30) |
| Correo | varchar(30) |

# Diseño



# Código

## **CONNECT.JAVA**

1. /\*\*
2. \* INSTITUTO NACIONAL DE MÉXICO
3. \* INSTITUTO TECNOLÓGICO DE LA LAGUNA
4. \* TÓPICOS AVANZADOS DE PROGRAMACIÓN
5. \* ENE-AGO 2017
6. \* PROFESOR: ING. LUIS FERNANDO GIL VAZQUEZ
7. \* ALUMNO: JOSÉ DE JESÚS MIRAMONTES GÓMEZ
8. \* MATRICULA: 15131349
9. \*/
10. **package** agendadecontactos;
12. **import** java.sql.Connection;
13. **import** java.sql.DriverManager;
14. **import** java.sql.PreparedStatement;
15. **import** java.sql.ResultSet;
16. **import** java.sql.SQLException;
17. **import** java.sql.Statement;
18. **import** java.sql.ResultSetMetaData;
19. **import** java.util.Vector;
20. **import** javax.swing.JTable;
21. **import** javax.swing.table.DefaultTableModel;
23. **public** **class** Connect {
25. **public** **static** String dbURL = "jdbc:derby://localhost:1527/contactos;create=true;user=APP;password=APP;";
26. **public** **static** String tableName = "listacontactos";
27. // jdbc Connection
28. **public** **static** Connection conn = **null**;
29. **public** **static** Statement stmt = **null**;
31. **public** **static** **void** createConnection() {
32. **try** {
33. //stmt.executeUpdate("SET SCHEMA APP");
34. Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
35. //Get a connection
36. conn = DriverManager.getConnection(dbURL);
37. } **catch** (ClassNotFoundException | IllegalAccessException | InstantiationException | SQLException except) {
38. }
39. }
41. **public** **static** **void** insertContacto(**int** id, String nombre, String telefono, String correo) {
42. **try** {
43. stmt = conn.createStatement();
44. stmt.execute("insert into " + tableName + " values ("
45. + id + ",'" + nombre + "','" + telefono + "','" + correo + "')");
46. stmt.close();
47. } **catch** (SQLException sqlExcept) {
48. sqlExcept.printStackTrace();
49. }
50. }
52. **public** **static** **void** updateContacto(**int** id, String nombre, String telefono, String correo) {
53. String updateTableSQL = "UPDATE " + tableName + " SET nombre = ?, telefono = ?, correo = ? WHERE id\_contacto = ?";
54. PreparedStatement preparedStatement = **null**;
55. **try** {
56. createConnection();
57. preparedStatement = conn.prepareStatement(updateTableSQL);
58. preparedStatement.setString(1, nombre);
59. preparedStatement.setString(2, telefono);
60. preparedStatement.setString(3, correo);
61. preparedStatement.setInt(4, id);
63. // execute update SQL stetement
64. preparedStatement.executeUpdate();
66. **if** (preparedStatement != **null**) {
67. preparedStatement.close();
68. }
70. **if** (conn != **null**) {
71. conn.close();
72. }
73. } **catch** (SQLException sqlExcept) {
74. sqlExcept.printStackTrace();
75. }
76. }
78. **public** **static** **void** removeContacto(**int** id)// throws SQLException
79. {
80. String deleteSQL = "DELETE FROM APP.listacontactos WHERE id\_contacto = ?";
81. PreparedStatement preparedStatement = **null**;
83. **try** {
84. createConnection();
85. preparedStatement = conn.prepareStatement(deleteSQL);
86. preparedStatement.setInt(1, id);
88. System.out.println(preparedStatement);
90. // execute delete SQL stetement
91. preparedStatement.executeUpdate();
93. System.out.println("Record is deleted!");
95. **if** (preparedStatement != **null**) {
96. preparedStatement.close();
97. }
99. **if** (conn != **null**) {
100. conn.close();
101. }
103. } **catch** (SQLException e) {
105. System.out.println(e.getMessage());
107. }
108. }
110. **public** **static** **void** fillTable(JTable table) {
111. **try** {
112. createConnection();
114. stmt = conn.createStatement();
115. ResultSet results = stmt.executeQuery("select \* from " + tableName);
116. ResultSetMetaData rsmd = results.getMetaData();
118. //To remove previously added rows
119. **while** (table.getRowCount() > 0) {
120. ((DefaultTableModel) table.getModel()).removeRow(0);
121. }
122. **int** columns = results.getMetaData().getColumnCount();
123. **while** (results.next()) {
124. Object[] row = **new** Object[columns];
125. **for** (**int** i = 1; i <= columns; i++) {
126. row[i - 1] = results.getObject(i);
127. }
128. ((DefaultTableModel) table.getModel()).insertRow(results.getRow() - 1, row);
129. }
131. results.close();
132. conn.close();
133. } **catch** (SQLException s) {
135. }
136. }
138. **public** **static** **void** selectRestaurants() {
139. **try** {
140. stmt = conn.createStatement();
141. ResultSet results = stmt.executeQuery("select \* from " + tableName);
142. ResultSetMetaData rsmd = results.getMetaData();
143. **int** numberCols = rsmd.getColumnCount();
145. results.close();
146. stmt.close();
147. } **catch** (SQLException sqlExcept) {
148. sqlExcept.printStackTrace();
149. }
150. }
152. **public** **static** Vector selectRestaurants(**int** id) {
153. Vector v = **new** Vector();
154. **boolean** flag = **false**;
155. **try** {
156. createConnection();
157. stmt = conn.createStatement();
158. ResultSet results = stmt.executeQuery("select \* from " + tableName + " where id\_contacto = " + id);
159. ResultSetMetaData rsmd = results.getMetaData();
160. **int** numberCols = rsmd.getColumnCount();

163. **while**(results.next())
164. {
165. flag = **true**;
166. v.add(results.getString(1));
167. v.add(results.getString(2));
168. v.add(results.getString(3));
169. v.add(results.getString(4));
170. }
172. results.close();
173. stmt.close();
174. } **catch** (SQLException sqlExcept) {
175. sqlExcept.printStackTrace();
176. }
177. //shutdown();
178. **if** (!flag)
179. v.add("empty");
180. **return** v;
181. }
183. **public** **static** **void** shutdown() {
184. **try** {
185. **if** (stmt != **null**) {
186. stmt.close();
187. }
188. **if** (conn != **null**) {
189. DriverManager.getConnection(dbURL + ";shutdown=true");
190. conn.close();
191. }
192. } **catch** (SQLException sqlExcept) {
194. }
196. }
197. }

## **GUI.JAVA**

//Código autogenerado omitido

1. **private** **void** btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
2. Connect.createConnection();
3. Connect.insertContacto((**int**)iId.getValue(), iNombre.getText(), iTelefono.getText(), iCorreo.getText());
4. Connect.fillTable(jTable1);
5. }
7. **private** **void** btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
8. **int** id = (**int**)iId.getValue();
9. System.out.println(id);
10. Connect.removeContacto(id);
11. Connect.fillTable(jTable1);
12. }
14. **private** **void** btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
15. Connect.updateContacto((**int**)iId.getValue(), iNombre.getText(), iTelefono.getText(), iCorreo.getText());
16. Connect.fillTable(jTable1);
17. }
19. **private** **void** jTable1MouseClicked(java.awt.event.MouseEvent evt) {
21. }
23. **private** **void** iIdStateChanged(javax.swing.event.ChangeEvent evt) {
24. **try**{
25. Vector v = Connect.selectRestaurants((**int**)iId.getValue());
26. **if**(!v.get(0).equals("empty")){
27. btnUpdate.setEnabled(**true**);
28. btnEliminar.setEnabled(**true**);
29. btnAgregar.setEnabled(**false**);
31. iNombre.setEnabled(**true**);
32. iCorreo.setEnabled(**true**);
33. iTelefono.setEnabled(**true**);
35. iNombre.setText((String)v.get(1));
36. iTelefono.setText((String)v.get(2));
37. iCorreo.setText((String)v.get(3));
38. } **else**{
39. btnUpdate.setEnabled(**false**);
40. btnEliminar.setEnabled(**false**);
41. btnAgregar.setEnabled(**true**);
43. iNombre.setEnabled(**true**);
44. iCorreo.setEnabled(**true**);
45. iTelefono.setEnabled(**true**);
47. iNombre.setText("");
48. iTelefono.setText("");
49. iCorreo.setText("");
50. }
51. }**catch**(Exception e){
53. }
54. }

## **SPLASHSCREEN.JAVA**

//Código autogenerado omitido

1. **public** SlashScreen() {

4. al = **new** ActionListener() {
5. @Override
6. **public** **void** actionPerformed(ActionEvent e) {
7. **if**(jProgressBar1.getValue() < 100){
8. jProgressBar1.setValue(jProgressBar1.getValue()+5);
9. }**else**{
10. t.stop();
11. MostrarVentana();
12. }
13. }
14. };
16. t = **new** Timer(80, al);
18. initComponents();
19. jProgressBar1.setStringPainted(**true**);
20. jProgressBar1.setBackground(**new** Color(60,63,65));
21. jProgressBar1.setForeground(**new** Color(60,63,65));
22. t.start();
23. }
25. **private** **void** MostrarVentana(){
26. GUI g = **new** GUI();
27. g.setLocationRelativeTo(**null**);
28. g.setVisible(**true**);
29. **this**.dispose();
30. }

## **AGENDADECONTACTOS.JAVA**

//Código autogenerado omitido

1. **public** **static** **void** main(String[] args) {
2. SlashScreen m = **new** SlashScreen();
3. m.setSize(820, 284);
4. m.setLocationRelativeTo(**null**);
5. m.setVisible(**true**);
7. }

# Prueba de Ejecución

