Henry Post & Jessica Soto

Final Lab

12/08/17

Description of the Code:

The project is about creating a help desk trouble ticket system. In the project, we create a Trouble ticket system helps you get a full control over any given trouble tickets. The admin/user can monitor trouble tickets effectively and improve the IT services you provide. These tickets are in displayed by GUIs, where is going to be navigate the ticket system. Operators or staff receiving a call or query from a user would fill out a small card with the user's details and a brief summary of the request and place it into a position in a column of pending slots for an appropriate engineer, so determining the staff member who would deal with the query and the priority of the request. With the trouble ticket is an implementation in used to track, detect, report and resolve any problem in the ticketing. With the GUI’s asks the admin (like yourself) which will intend of getting a ticket to resolve it. Which each ticket is identified by each ticket It, which will allow the admin/ user to add, delete or view the ticket. With all of this is connected to the server on Papademas.

For Henry, he was the one who made the basis in the project and was the one who implemented the SQL basis to the code. His skills created a new implementations in the code and made them more complex along the way with the code. With his code and with his GUIs he created a good basis and a good organization to the code. With his code, he made things in the project more organized and with that it made the code he made the structure of the code to be smoother, for example with the SQLC. Java, he put the things on order. For Jessica’s contribution in the group keeping the communications going for our final and keeping all in the group together for the team. Is to keep things in order in the group and communicate with the group. She coded more in the login aspect and in the Trouble Ticket GUI, she implemented it and coded it. Overall, we both collaborated our ideas together in the group to keep the ticketing system. With this collaboration, we helped one another in the code and tried to make complex with the code so that it can be easier with the user to use.

**Usage information:**

Valid users:

('**HenryFBP**','**iamapassword**','hpost@hawk.iit.edu','Henry Post',2),

('**peter**','**passwordhere**','peter@hawk.iit.edu','Peter',1),

('**mike**','**mikerocks12345**','mike@hawk.iit.edu','Michael Bernauer',1),

('**max**','**password**','max@hawk.iit.edu','Max Oellean',1),

('**admin**','**<no password.>**','admin@admin.admin','Admin admin',2);

Pressing <ENTER> on the username field will cause it to jump to the password field.

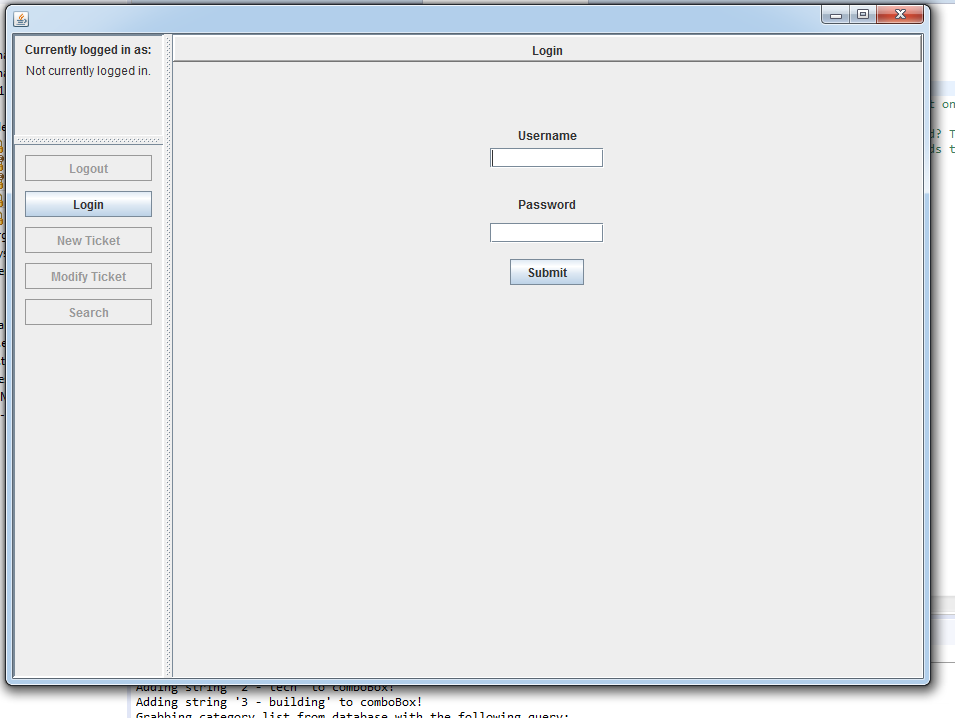
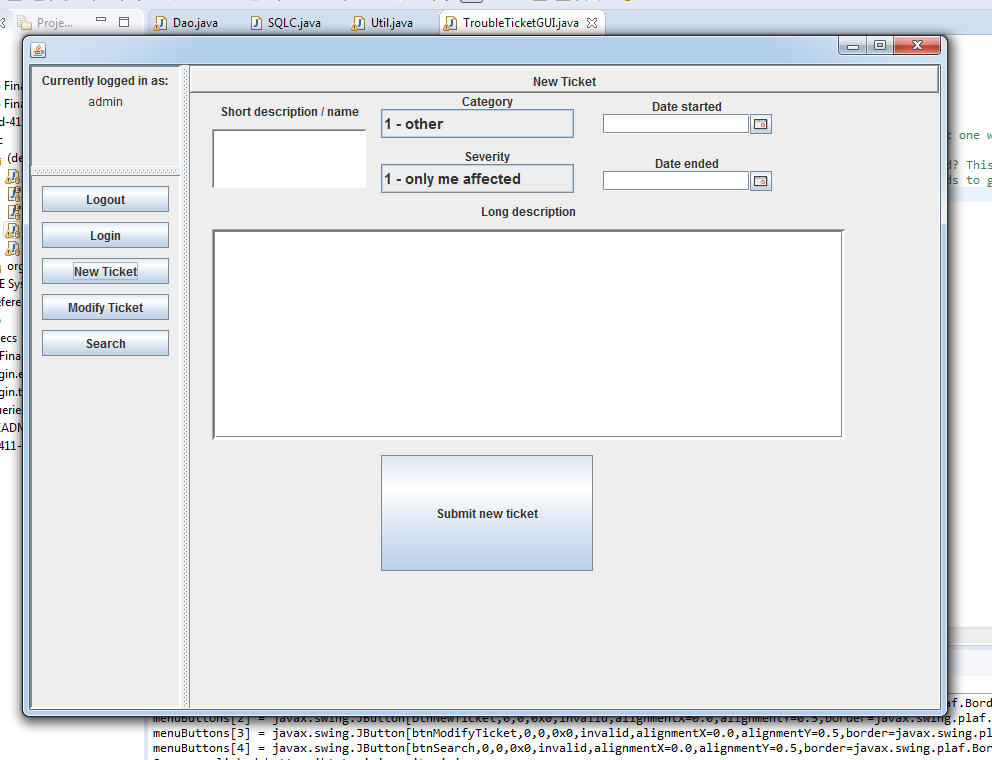
Incorrect fields that are not fixed in most contexts will be highlighted in red and selected.

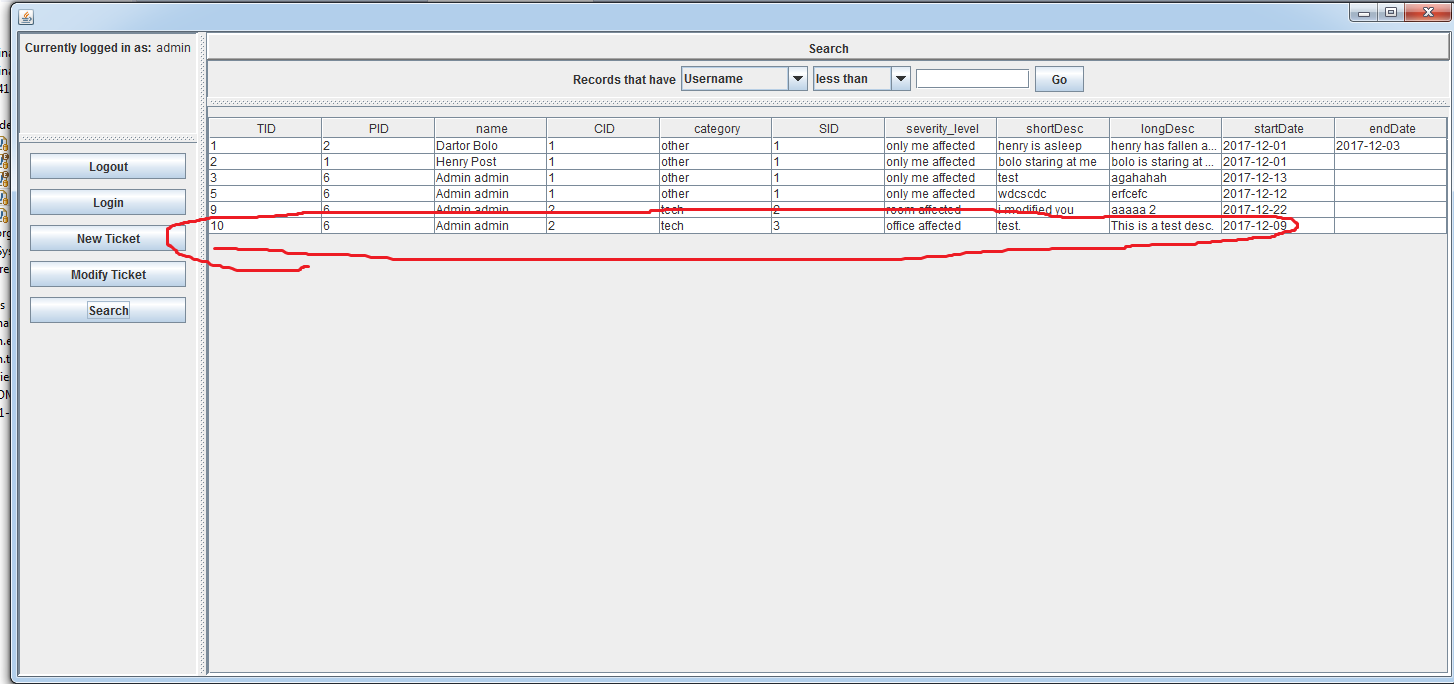
You can right-click on the results table to bring up a context-sensitive menu that allows you to delete or modify a ticket.

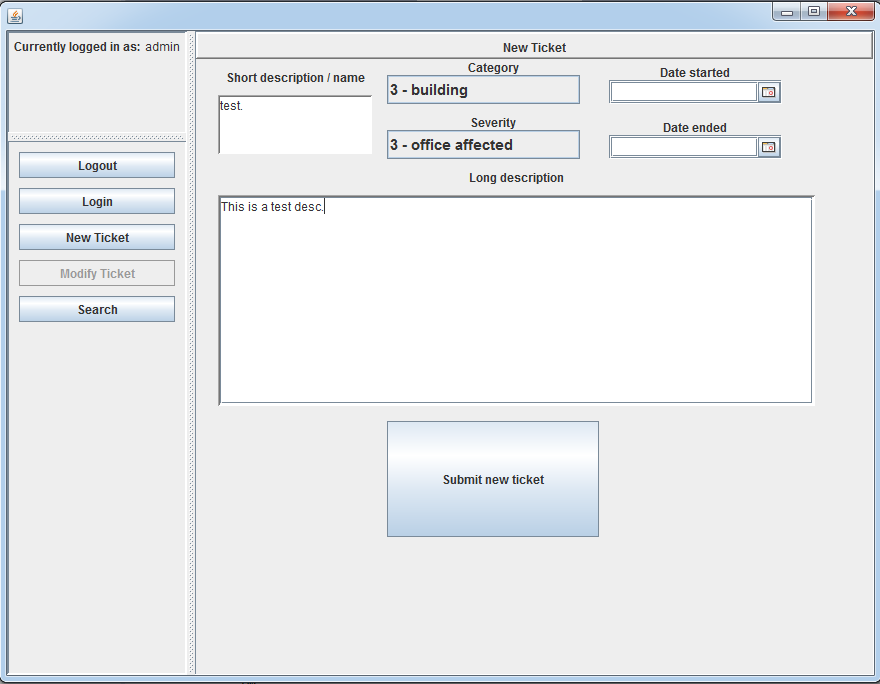
If something is disabled, hover over it. It usually has information about why it’s disabled.

[Source code directory](#Source_code_directory):

* [Dao.java](#DAO)
* [TroubleTicketGUI.java](#TroubleTicketGUI)
* [Util.java](#Util)
* [GUIC.java](#GUIC)
* [SQLC.java](#SQLC)

Screenshots:





# [DAO.java](#Source_code_directory)

1. **import** java.sql.Connection;
2. **import** java.sql.DriverManager;
3. **import** java.sql.ResultSet;
4. **import** java.sql.ResultSetMetaData;
5. **import** java.sql.SQLException;
6. **import** java.sql.Statement;
7. **import** java.text.SimpleDateFormat;
8. **import** java.util.ArrayList;
9. **import** java.util.Date;
11. **public** **class** Dao
12. {
14. **private** **static** **final** String DRIVER\_CLASS\_NAME = "com.mysql.jdbc.Driver";
16. **public** **static** **final** **int** USERNAME\_NOT\_FOUND = -0x0001;
17. **public** **static** **final** String USERNAME\_NOT\_FOUND\_S = "Username '%s' not found!";
19. **public** **static** **final** **int** PASSWORD\_INCORRECT = -0x0002;
20. **public** **static** **final** String PASSWORD\_INCORRECT\_S = "Password for user '%s' incorrect!";
22. **public** **static** **final** **int** NORMAL\_USER = 0x0001;
23. **public** **static** **final** String NORMAL\_USER\_S = "Welcome, user '%s'!";
25. **public** **static** **final** **int** ADMINISTRATOR = 0x0002;
26. **public** **static** **final** String ADMINISTRATOR\_S = "Welcome, admin '%s'!";
28. **public** **static** **final** String[] OPERATOR\_TYPES = { "<", "=", "CONTAINS", ">", "<>" };
29. **public** **static** **final** String[] OPERATOR\_TYPES\_S = { "less than", "is", "contains", "greater than", "is not" };
31. **public** **static** **final** String[] VALID\_SEARCH\_FIELDS\_USERS = { "username", "name" };
32. **public** **static** **final** String[] VALID\_SEARCH\_FIELDS\_USERS\_S = { "Username", "Name" };
34. **public** **static** **final** String[] VALID\_SEARCH\_FIELDS\_TICKETS = { "TID", "PID", "shortDesc", "category", "longDesc",
35. "startDate", "endDate" };
36. **public** **static** **final** String[] VALID\_SEARCH\_FIELDS\_TICKETS\_S = { "Ticket ID", "Person ID", "Short Description",
37. "Category", "Long Description", "Start Date", "End Date" };
39. **public** Connection c = **null**;
41. **public** Integer USER\_TYPE = **null**;
42. **public** String USER\_NAME = **null**;
43. **public** String USER\_PASS = **null**;
44. **public** Integer USER\_ID = **null**;


48. **public** Dao() // default constructor
49. {
51. }
53. **public** Dao(String host, String user, String password) // nondefault constructor
54. {
55. **this**.connect(host, user, password);
56. }
58. /\*\*\*
59. \* Tries to connect to user@host using password as a password.
60. \*
61. \* @param host
62. \* @param user
63. \* @param password
64. \* @return A 1 if succeeded, a -1 if any exceptions happened.
65. \*/
66. **public** **int** connect(String host, String user, String password)
67. {
68. **int** ret = -1;
70. **try**
71. {
72. Class.forName(DRIVER\_CLASS\_NAME);
73. System.out.printf("Connecting at host %s with user='%s', password='%s'\n", host, user, password);
74. **this**.c = DriverManager.getConnection(host, user, password);
75. ret = 1;
76. }
77. **catch** (SQLException e)
78. {
79. System.out.printf("Could not connect with the following credentials: '%s', '%s', '%s'\n\n", host, user,
80. password);
81. System.out.println(e.getMessage());
82. e.printStackTrace();
83. }
84. **catch** (ClassNotFoundException e)
85. {
86. System.out.printf("Could not find class '%s'!", DRIVER\_CLASS\_NAME);
87. e.printStackTrace();
88. }
89. System.out.printf("C = %s\n", c);
90. **return** ret;
91. }

94. /\*\*\*
95. \* Logs out a user. Sets all their fields to null.
96. \*/
97. **public** **void** logout()
98. {
99. **this**.USER\_TYPE = **null**;
100. **this**.USER\_NAME = **null**;
101. **this**.USER\_PASS = **null**;
102. **this**.USER\_ID = **null**;
103. }
105. /\*\*\*
106. \*
107. \* @param username
108. \*            The username of a user.
109. \* @param password
110. \*            The password of a user.
111. \* @return A nonzero number representing if: this user doesn't exist or has an
112. \*         incorrect password, is a normal user, or is an administrator.
113. \*/
114. **public** **int** login(String username, String password)
115. {
116. System.out.printf("Results for logging in user '%s' with password '%s'...\n", username, password);
117. **int** ret = 0;
118. **int** usertype = -1; // user = 1, admin = 2, etc...
119. **int** tempID = -1;
121. Statement s = **null**;
122. **try**
123. {
124. String query = String.format("SELECT \* FROM %s WHERE username = '%s'", SQLC.TABLE\_USERS, username);
125. s = c.createStatement();
127. System.out.printf("About to execute '%s'\n", query);
128. ResultSet rs = s.executeQuery(query); // gets username and password from table of users
129. ResultSetMetaData rsmd = rs.getMetaData();
130. **int** cols = resultSetColumns(rs);
132. System.out.printf("Got %d columns from that query!\n", cols);
134. // if user doesn't exist
135. **if** (cols == 0)
136. {
137. System.out.printf("User with name '%s' does not exist!\n", username);
138. **return** USERNAME\_NOT\_FOUND;
139. }
141. rs.beforeFirst();
142. rs.next();
144. // verify user has entered the right password
145. **if** (rs.getString(SQLC.PASSWORD\_COLUMN\_NAME.s()).compareTo((password)) == 0)
146. {
147. System.out.printf("Welcome, user '%s'!\n", username);
148. ret = rs.getInt(SQLC.USERTYPE\_COLUMN\_NAME.s()); // get user type
149. tempID = rs.getInt(SQLC.USER\_ID\_COLUMN\_NAME.s()); //get user id for later
151. **switch** (usertype)
152. {
153. **case** NORMAL\_USER:
154. **break**;
155. **case** ADMINISTRATOR:
156. **break**;
157. }
159. }
160. **else**
161. {
162. // wrong password, right user...
163. ret = PASSWORD\_INCORRECT;
164. }
166. ArrayList<String> formattedResults = Util.formatResultSet(rs);
168. Util.printList(formattedResults);
170. }
171. **catch** (SQLException e)
172. {
173. e.printStackTrace();
174. }
176. **if** (ret > 0) // if user is valid, modify the current DAO object for use later.
177. {
178. **this**.USER\_TYPE = ret;
179. **this**.USER\_NAME = username;
180. **this**.USER\_PASS = password;
181. **this**.USER\_ID = tempID;

184. }
186. **return** ret;
188. }
190. /\*\*\*
191. \* Returns a list of categories from the {@link SQLC.TABLE\_CATEGORIES} table.
192. \* @return
193. \*/
194. **public** ArrayList<String> getCategories()
195. {
196. ArrayList<String> ret = **new** ArrayList<>();
198. Statement s;
199. **try**
200. {
201. s = **this**.c.createStatement();
203. String query = String.format("SELECT \* FROM %s",SQLC.TABLE\_CATEGORIES);
205. ResultSet rs = s.executeQuery(query);
206. }
207. **catch** (SQLException e)
208. {
209. e.printStackTrace();
210. }

213. **return** ret;
214. }

217. **public** **static** ArrayList<ArrayList<Object>> resultSetToList(ResultSet rs)
218. {
219. ResultSetMetaData rsmd = **null**;
220. ArrayList<ArrayList<Object>> ret = **new** ArrayList<>();
221. ArrayList<Object> tempRow;
223. **try**
224. {
226. rsmd = rs.getMetaData();
228. // System.out.printf("Width of cols are %d\n",rsmd.getColumnCount());
230. **int** col = 0;
231. **while** (rs.next())
232. {
233. // System.out.printf("Col %d\n",col);
234. // System.out.println("rs.getString(1) = "+rs.getString(1));
235. // System.out.println("rs.getObject(1) = "+rs.getObject(1));
237. tempRow = **new** ArrayList<>();
239. **if** (rsmd.getColumnCount() > 1)
240. {
241. **for** (**int** i = 1; i < rsmd.getColumnCount(); i++)
242. {
243. // System.out.printf("about to add %s\n",rs.getObject(i));
244. tempRow.add((Object) rs.getString(i));
245. }
246. }
248. tempRow.add((Object) rs.getString(rsmd.getColumnCount())); // FIXME figure out why we need this...fixes
249. // last col not appearing...
251. // rs.geto
252. ret.add(tempRow);
253. // System.out.println("added this row to ret array: "+tempRow.toString());
254. col++;
255. }
257. }
258. **catch** (SQLException e)
259. {
260. e.printStackTrace();
261. }
262. // System.out.println("Returning this: "+ret.toString());
264. **return** ret;
265. }
267. **public** **static** ArrayList<ArrayList<Object>> resultSetMetadataToList(ResultSet rs)
268. {
269. ResultSetMetaData rsmd = **null**;
270. ArrayList<ArrayList<Object>> ret = **new** ArrayList<>();
272. **try**
273. {
274. rsmd = rs.getMetaData();
275. ArrayList<Object> tempRow = **new** ArrayList<>();
276. ArrayList<Object> tempRow2 = **new** ArrayList<>();
278. // add metadata for headers to ret array
279. **for** (**int** i = 1; i < rsmd.getColumnCount(); i++)
280. {
281. tempRow.add(rsmd.getColumnName(i));
282. tempRow2.add(rsmd.getColumnTypeName(i));
283. }
285. tempRow.add((Object) rsmd.getColumnName(rsmd.getColumnCount())); // FIXME figure out why we need
286. // this...fixes last col not
287. // appearing...
288. tempRow2.add((Object) rsmd.getColumnName(rsmd.getColumnCount())); // FIXME figure out why we need
289. // this...fixes last col not
290. // appearing...
291. ret.add(tempRow);
292. ret.add(tempRow2);
294. }
295. **catch** (SQLException e)
296. {
297. e.printStackTrace();
298. }
300. **return** ret;
301. }
303. /\*\*\*
304. \*
305. \* @param rs
306. \*            A ResultSet object. It will record the current position that the
307. \*            ResultSet is in and will reset it back to that position once done
308. \*            iterating.
309. \* @return The number of columns in the resultSet.
310. \*/
311. **public** **static** **int** resultSetColumns(ResultSet rs)
312. {
314. **int** cols = 0;
315. **int** position = -1;
317. **try**
318. {
319. position = rs.getRow(); // record our position
320. rs.beforeFirst(); // goto 0
322. **while** (rs.next())
323. {
324. cols++;
325. }
327. rs.absolute(position); // reset our position
328. }
329. **catch** (SQLException e)
330. {
331. e.printStackTrace();
332. }
334. **try**
335. {
336. rs.absolute(position); // go back to original position
337. }
338. **catch** (SQLException e)
339. {
340. e.printStackTrace();
341. }
342. **return** cols;
343. }


347. /\*\*\*
348. \* Inserts a ticket.
349. \*/
350. **public** **void** submitTicket(String name, String shortDesc, String longDesc, Integer category, Integer severity,
351. Date startDate, Date endDate)
352. {
353. name = Util.sanitize(name);
354. shortDesc = Util.sanitize(shortDesc);
355. longDesc = Util.sanitize(longDesc);

358. SimpleDateFormat sdf = **new** SimpleDateFormat(SQLC.SQL\_DATE\_FORMAT.s());
359. String query = "";
361. String sDate = sdf.format(startDate);
362. Date ed = endDate;
364. **try**
365. {
366. Statement s = **this**.c.createStatement();
368. query = String.format("INSERT INTO %s (%s, %s, %s, %s, %s, %s) ",
369. SQLC.TABLE\_TICKETS, SQLC.USER\_ID\_COLUMN\_NAME, SQLC.SHORT\_DESC\_COLUMN\_NAME,
370. SQLC.LONG\_DESC\_COLUMN\_NAME, SQLC.CATEGORY\_ID\_COLUMN\_NAME, SQLC.SEVERITY\_ID\_COLUMN\_NAME, SQLC.START\_DATE\_COLUMN\_NAME);
372. query += String.format("VALUES (%d, '%s', '%s', %d, %d, STR\_TO\_DATE('%s','%s'));",
373. **this**.USER\_ID,  shortDesc,   longDesc,   category,   severity,  sDate, SQLC.SQL\_STR\_TO\_DATE\_FORMAT.s());
375. System.out.printf("About to insert a record with the following SQL statement:\n '%s'\n",query);
377. s.execute(query);
378. }
379. **catch** (SQLException e)
380. {
381. e.printStackTrace();
382. }
383. }

386. /\*\*\*
387. \* Deletes a ticket by its ID.
388. \* @return Whether or not deletion was successful.
389. \*/
390. **public** Boolean deleteTicket(**int** id)
391. {
392. Statement s;
394. **try**
395. {
396. s = **this**.c.createStatement();
398. s.execute(String.format("DELETE FROM %s WHERE %s = %d",
399. SQLC.TABLE\_TICKETS,SQLC.TICKET\_ID\_COLUMN\_NAME,id));
400. **return** **true**;
401. }
402. **catch** (SQLException e)
403. {
404. Util.printf("Failed to delete ticket of ID '%d'!",id);
405. System.out.println(e.getMessage());
406. System.out.println(e.getSQLState());
407. e.printStackTrace();
408. **return** **false**;
409. }
410. }
412. /\*\*\*
413. \* @param tid The ticket id.
414. \* @return The resultSet associated with the ticket id.
415. \*/
416. **public** ResultSet getTicketRS(**int** tid)
417. {
418. Statement s = **null**;
419. ResultSet rs = **null**;
421. **try**
422. {
423. s = **this**.c.createStatement();
425. rs = s.executeQuery(String.format("SELECT \* FROM %s WHERE %s = %d",
426. SQLC.TABLE\_TICKETS,SQLC.TICKET\_ID\_COLUMN\_NAME,tid));
428. }
429. **catch** (SQLException e)
430. {
431. Util.printf("Failed to get TID '%d",tid,SQLC.SHORT\_DESC\_COLUMN\_NAME);
432. System.out.println(e.getMessage());
433. System.out.println(e.getSQLState());
434. e.printStackTrace();
435. }
436. **return** rs;
437. }
439. /\*\*\*
440. \*
441. \* @param category A string representing a category.
442. \* @return An int that represents the category's CID in the database.
443. \*/
444. **public** **int** categoryToInt(String category)
445. {
446. //TODO
447. **return** -1;
448. }

451. }

# [TroubleTicketGUI.java](#Source_code_directory)

1. **import** java.awt.BorderLayout;
2. **import** java.awt.CardLayout;
3. **import** java.awt.Color;
4. **import** java.awt.EventQueue;
5. **import** java.awt.Font;
6. **import** java.awt.GridBagConstraints;
7. **import** java.awt.GridBagLayout;
8. **import** java.awt.Insets;
9. **import** java.awt.Point;
10. **import** java.awt.event.ActionEvent;
11. **import** java.awt.event.ActionListener;
12. **import** java.awt.event.WindowAdapter;
13. **import** java.awt.event.WindowEvent;
14. **import** java.sql.ResultSet;
15. **import** java.sql.SQLException;
16. **import** java.util.ArrayList;
18. **import** javax.swing.AbstractAction;
19. **import** javax.swing.BorderFactory;
20. **import** javax.swing.DefaultComboBoxModel;
21. **import** javax.swing.JButton;
22. **import** javax.swing.JComboBox;
23. **import** javax.swing.JFrame;
24. **import** javax.swing.JLabel;
25. **import** javax.swing.JMenuItem;
26. **import** javax.swing.JPanel;
27. **import** javax.swing.JPasswordField;
28. **import** javax.swing.JPopupMenu;
29. **import** javax.swing.JScrollPane;
30. **import** javax.swing.JSplitPane;
31. **import** javax.swing.JTable;
32. **import** javax.swing.JTextArea;
33. **import** javax.swing.JTextField;
34. **import** javax.swing.SpringLayout;
35. **import** javax.swing.SwingConstants;
36. **import** javax.swing.SwingUtilities;
37. **import** javax.swing.border.BevelBorder;
38. **import** javax.swing.border.Border;
39. **import** javax.swing.event.PopupMenuEvent;
40. **import** javax.swing.event.PopupMenuListener;
41. **import** javax.swing.table.DefaultTableModel;
43. **import** com.toedter.calendar.JDateChooser;
45. **public** **class** TroubleTicketGUI **extends** JFrame
46. {
48. **private** **static** **final** **long** serialVersionUID = -5012247422436531433L;
49. **public** **static** **int** DEFAULT\_LOGIN\_MODE = 1; // Which entry should be used? If this is too large, the last one will be
50. // used.
51. **public** **static** **final** String LOGINS\_FILE\_PATH = "login.txt"; // Where are our logins and passwords stored? This is for
52. // security and for not committing passwords to git.
54. **public** **static** **final** **int** PADDING = 10;
55. **public** **static** **final** **int** DEFAULT\_DIVIDER\_WIDTH = 150;
57. **private** Boolean \_ARE\_MODIFYING\_TICKET = **false**;
58. **private** **int** \_PREV\_TICKET\_ID = -1;


62. **public** **static** **final** String NESW = "NESW";
63. **public** **static** **final** String PREFERRED\_TIME\_FORMAT = "MMM d, yyyy, hh:mm:ss aa";
65. **private** **static** **final** String BUTTON\_LOGOUT\_NAME = "btnLogout";
66. **private** **static** **final** String BUTTON\_LOGIN\_NAME = "btnLogin";
67. **private** **static** **final** String BUTTON\_NEW\_TICKET\_NAME = "btnNewTicket";
68. **private** **static** **final** String BUTTON\_NEW\_TICKET\_S = "New Ticket";
69. **private** **static** **final** String BUTTON\_MODIFY\_TICKET\_NAME = "btnModifyTicket";
70. **private** **static** **final** String BUTTON\_SEARCH\_NAME = "btnSearch";
71. **private** **static** **final** String BUTTON\_S\_SUBMIT\_NEW\_TICKET\_S = "Submit new ticket";

74. **private** **static** **final** String PANEL\_RIGHT\_CONTENT\_NAME = "panelRightContent";
76. **public** **static** **final** Border borderDefault = **new** JTextField().getBorder();
77. **public** **static** **final** Border borderBad = BorderFactory.createLineBorder(Color.RED);
79. **public** **static** Dao dao;


83. /\*\*\*
84. \* Changes the right panel to show ONLY JPanel 'jp'.
85. \* @param jp  The JPanel to be shown
86. \* @param e  The actionEvent that tells us what button was clicked.
87. \*/
88. **public** **void** modifyRightPane(JPanel jp, ActionEvent e, String panelName)
89. {
90. CardLayout cl = (CardLayout) panelRightContent.getLayout();
91. JLabel label = lblCurrentPane;
92. JButton button = (JButton) e.getSource();
94. Util.println("Someone clicked button '" + button.getName() + "' or '" + button.getText() + "'");
95. Util.printf("About to do cl.show('%s', '%s')\n", panelRightContent.getName(), jp.getName());
97. cl.show(panelRightContent, jp.getName());
98. label.setText(panelName);
99. }
101. **public** **void** modifyRightPane(JPanel jp, String panelName)
102. {
103. CardLayout cl = (CardLayout) panelRightContent.getLayout();
104. JLabel label = lblCurrentPane;
106. Util.println("Don't know what button someone clicked. Just changing right panel.");
108. cl.show(panelRightContent, jp.getName());
109. label.setText(panelName);
110. }

113. /\*\*\*
114. \* Gets the info from the two login info fields. This is its own method so that
115. \* either pressing ENTER or clicking "Submit" will log you in.
116. \*/
117. **public** String[] getLogin()
118. {
119. String[] ret = { "", "" };
121. ret[0] = textFieldUsername.getText();
122. ret[1] = textFieldPassword.getText();
124. **return** ret;
125. }
127. **private** **void** doLogin(String username, String password)
128. {
129. lblLoginErrorMsg.setText("");
130. textFieldUsername.setBorder(borderDefault);
131. textFieldPassword.setBorder(borderDefault);
133. **int** result = dao.login(username, password);
135. **if** (result == Dao.NORMAL\_USER || result == Dao.ADMINISTRATOR)
136. {// log them in!
138. Util.setTooltips(panelMenu,**new** String[] {}, ""); //reset tooltips
139. Util.enableButtons(panelMenu);                   //enable all buttons

142. **if** (result == Dao.NORMAL\_USER)
143. {
144. lblWhosLoggedIn.setText(String.format(Dao.NORMAL\_USER\_S, username));
145. }
146. **else** **if** (result == Dao.ADMINISTRATOR)
147. {
148. lblWhosLoggedIn.setText(String.format(Dao.ADMINISTRATOR\_S, username));
149. }
150. lblWhosLoggedIn.setText(username);
151. }
152. **if** (result == Dao.PASSWORD\_INCORRECT) // wrong pass
153. {
154. textFieldPassword.setBorder(borderBad);
155. textFieldPassword.requestFocus();
156. lblLoginErrorMsg.setText(String.format(Dao.PASSWORD\_INCORRECT\_S, username));
157. }
158. **if** (result == Dao.USERNAME\_NOT\_FOUND) // username not found
159. {
160. textFieldUsername.setBorder(borderBad);
161. textFieldUsername.requestFocus();
162. lblLoginErrorMsg.setText(String.format(Dao.USERNAME\_NOT\_FOUND\_S, username));
163. }
165. }
167. **private** JFrame frame;
168. **private** JLabel lblWhosLoggedIn;
169. **private** JLabel lblCurrentPane;
171. **private** JButton btnLogout;
172. **private** JButton btnLogin;
173. **private** JButton btnNewTicket;
174. **private** JButton btnModifyTicket;
175. **private** JButton btnSearch;
177. **private** JPasswordField textFieldPassword;
179. **private** JPanel panelRightContent;
181. **private** JPanel panelLogin;
182. **private** JPanel panelNewTicket;
183. **private** JPanel panelSearch;
184. **private** JPanel panelModifyTicket;
185. **private** JButton btnSubmitChanges;
186. **private** JPanel panelRight;
187. **private** SpringLayout sl\_panelRight;
188. **private** SpringLayout sl\_panelLogin;
189. **private** JPanel panelUsername;
190. **private** JLabel lblUsername;
191. **private** JPanel panelRightTop;
192. **private** JPanel panelPassword;
193. **private** JLabel lblPassword;
194. **private** JButton btnNTSubmitNewTicket;
195. **private** JButton btnSearchThroughRecords;
196. **private** SpringLayout springLayout\_1;
197. **private** JPanel panelStatus;
198. **private** JLabel lblCurrentlyLoggedIn;
199. **private** JSplitPane splitPane;
200. **private** JPanel panelLeft;
201. **private** JSplitPane splitPaneLeft;
202. **private** JPanel panelMenu;
203. **private** JLabel lblLoginErrorMsg;
204. **private** JPanel panelLoginErrorMsg;
205. **private** SpringLayout springLayout;
206. **private** SpringLayout sl\_panelMenu;
207. **private** JPanel panelNTShortDesc;
208. **private** JTextArea textAreaShortDesc;
209. **private** JPanel panelTextAreaShortDesc;
210. **private** SpringLayout sl\_panelNTShortDesc;
211. **private** JLabel lblShortDesc;
212. **private** JPanel panelNTCategory;
213. **private** JPanel panelNTButtonSubmitNewTicket;
214. **private** JPanel panelTextAreaLongDesc;
215. **private** JTextArea textAreaLongDesc;
216. **private** JPanel panelUsernameMiddle;
217. **private** JTextField textFieldUsername;
218. **private** JPanel panelPasswordMiddle;
219. **private** JScrollPane scrollPaneLongDesc;
220. **private** SpringLayout sl\_panelNTLongDesc;
221. **private** JPanel panelNTLongDesc;
222. **private** JComboBox comboBoxCategory;
223. **private** SpringLayout sl\_panelUsername;
224. **private** JLabel lblLongDescription;
225. **private** JPanel panelBtnLoginSubmitMiddle;
226. **private** JButton btnLoginSubmit;
227. **private** SpringLayout sl\_panelPassword;
228. **private** JTable tableResults;
229. **private** JLabel lblRecordsThatHave;
230. **private** JComboBox<String> comboBoxField;
231. **private** JComboBox<String> comboBoxOperator;
232. **private** JTextField textField;
233. **private** JSplitPane splitPaneSearch;
234. **private** JPanel panelSearchTop;
235. **private** JPanel panelSearchResultsTop;
236. **private** JLabel labelCategory;
237. **private** JLabel labelSeverity;
238. **private** JComboBox<String> comboBoxSeverity;
239. **private** JPanel panelSearchBottom;
240. **private** JPanel panelNTSeverity;
241. **private** JPanel panelNTDateStarted;
242. **private** JLabel labelDateStarted;
243. **private** JDateChooser dateStartedChooser;
244. **private** JPanel panelNTDateEnded;
245. **private** JLabel labelDateEnded;
246. **private** JDateChooser dateEndedChooser;
247. **private** JTextField textField\_1;
249. /\*\*
250. \* Launch the application.
251. \*/
252. **public** **static** **void** main(String[] args)
253. {
254. ArrayList<ArrayList<String>> logins = Util.getLoginsFromFile(LOGINS\_FILE\_PATH);
256. **int** i = 0;
257. **for** (ArrayList<String> login : logins)
258. {
259. // printf("Login is %d long and is %s",login.size(),login);
260. // printf("Login %2d: '%s','%s','%s'",i,login.get(0),login.get(1),login.get(2));
261. i++;
262. }
264. **if** (DEFAULT\_LOGIN\_MODE >= logins.size())
265. {
266. DEFAULT\_LOGIN\_MODE = logins.size() - 1;
267. }
269. Util.printf("\n\nWe're going to use the %dth entry in '%s', aka '%s'.\n\n", DEFAULT\_LOGIN\_MODE, LOGINS\_FILE\_PATH,
270. logins.get(DEFAULT\_LOGIN\_MODE));
272. ArrayList<String> login = logins.get(DEFAULT\_LOGIN\_MODE);
274. dao = **new** Dao(login.get(0), login.get(1), login.get(2));
276. EventQueue.invokeLater(**new** Runnable()
277. {
278. **public** **void** run()
279. {
280. TroubleTicketGUI window = **new** TroubleTicketGUI();
281. window.frame.setVisible(**true**);
282. }
283. });
284. }
286. /\*\*
287. \* Create the application.
288. \*/
289. **public** TroubleTicketGUI()
290. {
291. initialize();
292. }
294. /\*\*
295. \* Initialize the contents of the frame.
296. \*/
297. **private** **void** initialize()
298. {
299. frame = **new** JFrame();
300. frame.setBounds(100, 100, 926, 682);
301. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
302. springLayout = **new** SpringLayout();
303. frame.getContentPane().setLayout(springLayout);
305. splitPane = **new** JSplitPane();
306. splitPane.setName("splitPane");
307. springLayout.putConstraint(Util.SOUTH, splitPane, 0, Util.SOUTH, frame.getContentPane());
308. springLayout.putConstraint(Util.EAST, splitPane, 0, Util.EAST, frame.getContentPane());
309. springLayout.putConstraint(Util.NORTH, splitPane, 0, Util.NORTH, frame.getContentPane());
310. springLayout.putConstraint(Util.WEST, splitPane, 0, Util.WEST, frame.getContentPane());
311. frame.getContentPane().add(splitPane, splitPane.getName());
313. panelLeft = **new** JPanel();
314. splitPane.setLeftComponent(panelLeft);
315. panelLeft.setLayout(**new** BorderLayout(0, 0));
317. splitPaneLeft = **new** JSplitPane();
318. splitPaneLeft.setOrientation(JSplitPane.VERTICAL\_SPLIT);
319. panelLeft.add(splitPaneLeft);
320. panelMenu = **new** JPanel();
321. splitPaneLeft.setRightComponent(panelMenu);
322. sl\_panelMenu = **new** SpringLayout();
323. panelMenu.setLayout(sl\_panelMenu);
325. btnLogout = **new** JButton("Logout");
326. btnLogout.setName(BUTTON\_LOGOUT\_NAME);
327. panelMenu.add(btnLogout);
329. btnLogin = **new** JButton("Login");
330. btnLogin.setName(BUTTON\_LOGIN\_NAME);
331. panelMenu.add(btnLogin);
333. btnNewTicket = **new** JButton("New Ticket");
334. btnNewTicket.setName(BUTTON\_NEW\_TICKET\_NAME);
335. panelMenu.add(btnNewTicket);
337. btnModifyTicket = **new** JButton("Modify Ticket");
338. btnModifyTicket.setName(BUTTON\_MODIFY\_TICKET\_NAME);
339. panelMenu.add(btnModifyTicket);
341. btnSearch = **new** JButton("Search");
342. btnSearch.setName(BUTTON\_SEARCH\_NAME);
343. panelMenu.add(btnSearch);
345. panelStatus = **new** JPanel();
346. splitPaneLeft.setLeftComponent(panelStatus);
348. lblCurrentlyLoggedIn = **new** JLabel(GUIC.CURRENTLY\_LOGGED\_IN\_TEXT.s());
349. panelStatus.add(lblCurrentlyLoggedIn);
351. lblWhosLoggedIn = **new** JLabel(GUIC.NOT\_LOGGED\_IN+"");
352. lblWhosLoggedIn.setFont(**new** Font("Dialog", Font.PLAIN, 12));
353. panelStatus.add(lblWhosLoggedIn);
354. splitPaneLeft.setDividerLocation(100);
356. panelRight = **new** JPanel();
357. splitPane.setRightComponent(panelRight);
358. sl\_panelRight = **new** SpringLayout();
359. panelRight.setLayout(sl\_panelRight);
361. panelRightContent = **new** JPanel();
362. sl\_panelRight.putConstraint(Util.SOUTH, panelRightContent, 0, Util.SOUTH, panelRight);
363. sl\_panelRight.putConstraint(Util.EAST, panelRightContent, 0, Util.EAST, panelRight);
364. panelRightContent.setName(PANEL\_RIGHT\_CONTENT\_NAME);
365. sl\_panelRight.putConstraint(Util.WEST, panelRightContent, 0, Util.WEST, panelRight);
366. panelRight.add(panelRightContent, panelRightContent.getName());
367. panelRightContent.setLayout(**new** CardLayout(0, 0));
369. panelLogin = **new** JPanel();
370. panelLogin.setName("panelLogin");
371. panelRightContent.add(panelLogin, panelLogin.getName());
372. sl\_panelLogin = **new** SpringLayout();
373. panelLogin.setLayout(sl\_panelLogin);
375. panelNewTicket = **new** JPanel();
376. panelNewTicket.setName("panelNewTicket");
377. panelRightContent.add(panelNewTicket, panelNewTicket.getName());
379. panelSearch = **new** JPanel();
380. panelSearch.setName("panelSearch");
381. panelRightContent.add(panelSearch, panelSearch.getName());
383. panelLoginErrorMsg = **new** JPanel();
384. sl\_panelLogin.putConstraint(Util.NORTH, panelLoginErrorMsg, PADDING, Util.NORTH, panelLogin);
385. sl\_panelLogin.putConstraint(Util.WEST, panelLoginErrorMsg, PADDING, Util.WEST, panelLogin);
386. sl\_panelLogin.putConstraint(Util.EAST, panelLoginErrorMsg, -PADDING, Util.EAST, panelLogin);
387. panelLogin.add(panelLoginErrorMsg);
388. panelLoginErrorMsg.setLayout(**new** BorderLayout(0, 0));
390. lblLoginErrorMsg = **new** JLabel("THIS\_IS\_A\_DEFAULT\_ERROR\_MESSAGE");
391. lblLoginErrorMsg.setHorizontalAlignment(SwingConstants.CENTER);
392. panelLoginErrorMsg.add(lblLoginErrorMsg, BorderLayout.CENTER);
394. panelUsername = **new** JPanel();
395. sl\_panelLogin.putConstraint(Util.NORTH, panelUsername, 60, Util.NORTH, panelLogin);
396. sl\_panelLogin.putConstraint(Util.SOUTH, panelLoginErrorMsg, -6, Util.NORTH, panelUsername);
397. sl\_panelLogin.putConstraint(Util.WEST, panelUsername, PADDING, Util.WEST, panelLogin);
398. sl\_panelLogin.putConstraint(Util.EAST, panelUsername, -PADDING, Util.EAST, panelLogin);
399. panelLogin.add(panelUsername);
400. sl\_panelUsername = **new** SpringLayout();
401. panelUsername.setLayout(sl\_panelUsername);
403. lblUsername = **new** JLabel("Username");
404. sl\_panelUsername.putConstraint(Util.WEST, lblUsername, PADDING, Util.WEST, panelUsername);
405. sl\_panelUsername.putConstraint(Util.EAST, lblUsername, -PADDING, Util.EAST, panelUsername);
406. lblUsername.setHorizontalAlignment(SwingConstants.CENTER);
407. sl\_panelUsername.putConstraint(Util.NORTH, lblUsername, 5, Util.NORTH, panelUsername);
408. panelUsername.add(lblUsername);
409. sl\_panelLogin.putConstraint(Util.WEST, lblUsername, 41, Util.WEST, panelLogin);
410. sl\_panelLogin.putConstraint(Util.SOUTH, lblUsername, -397, Util.SOUTH, panelLogin);
412. panelPassword = **new** JPanel();
413. sl\_panelLogin.putConstraint(Util.NORTH, panelPassword, 129, Util.NORTH, panelLogin);
414. sl\_panelLogin.putConstraint(Util.SOUTH, panelPassword, -36, Util.SOUTH, panelLogin);
415. sl\_panelLogin.putConstraint(Util.SOUTH, panelUsername, 0, Util.NORTH, panelPassword);
416. sl\_panelLogin.putConstraint(Util.WEST, panelPassword, PADDING, Util.WEST, panelLogin);
417. sl\_panelLogin.putConstraint(Util.EAST, panelPassword, -PADDING, Util.EAST, panelLogin);
419. panelUsernameMiddle = **new** JPanel();
420. sl\_panelUsername.putConstraint(Util.NORTH, panelUsernameMiddle, 0, Util.SOUTH, lblUsername);
421. sl\_panelUsername.putConstraint(Util.WEST, panelUsernameMiddle, PADDING, Util.WEST, panelUsername);
422. sl\_panelUsername.putConstraint(Util.SOUTH, panelUsernameMiddle, -PADDING, Util.SOUTH, panelUsername);
423. sl\_panelUsername.putConstraint(Util.EAST, panelUsernameMiddle, -PADDING, Util.EAST, panelUsername);
424. panelUsername.add(panelUsernameMiddle);
426. textFieldUsername = **new** JTextField();
427. textFieldUsername.setToolTipText("username");
428. textFieldUsername.setColumns(10);
429. panelUsernameMiddle.add(textFieldUsername);
430. panelLogin.add(panelPassword);
431. sl\_panelPassword = **new** SpringLayout();
432. panelPassword.setLayout(sl\_panelPassword);
434. lblPassword = **new** JLabel("Password");
435. sl\_panelPassword.putConstraint(Util.WEST, lblPassword, 0, Util.WEST, panelPassword);
436. sl\_panelPassword.putConstraint(Util.EAST, lblPassword, 0, Util.EAST, panelPassword);
437. lblPassword.setHorizontalAlignment(SwingConstants.CENTER);
438. sl\_panelPassword.putConstraint(Util.NORTH, lblPassword, 5, Util.NORTH, panelPassword);
439. panelPassword.add(lblPassword);
441. panelPasswordMiddle = **new** JPanel();
442. sl\_panelPassword.putConstraint(Util.NORTH, panelPasswordMiddle, 6, Util.SOUTH, lblPassword);
443. sl\_panelPassword.putConstraint(Util.WEST, panelPasswordMiddle, PADDING, Util.WEST, lblPassword);
444. sl\_panelPassword.putConstraint(Util.EAST, panelPasswordMiddle, -PADDING, Util.EAST, lblPassword);
445. panelPassword.add(panelPasswordMiddle);
447. textFieldPassword = **new** JPasswordField();
448. panelPasswordMiddle.add(textFieldPassword);
449. sl\_panelPassword.putConstraint(Util.NORTH, textFieldPassword, 26, Util.NORTH, panelPassword);
450. sl\_panelPassword.putConstraint(Util.WEST, textFieldPassword, 12, Util.WEST, panelPassword);
451. textFieldPassword.setToolTipText("password");
452. textFieldPassword.setColumns(10);
454. panelBtnLoginSubmitMiddle = **new** JPanel();
455. sl\_panelPassword.putConstraint(Util.NORTH, panelBtnLoginSubmitMiddle, 6, Util.SOUTH,panelPasswordMiddle);
456. sl\_panelPassword.putConstraint(Util.WEST, panelBtnLoginSubmitMiddle, PADDING, Util.WEST,panelPassword);
457. sl\_panelPassword.putConstraint(Util.SOUTH, panelBtnLoginSubmitMiddle, 42, Util.SOUTH,panelPasswordMiddle);
458. sl\_panelPassword.putConstraint(Util.EAST, panelBtnLoginSubmitMiddle, -PADDING, Util.EAST,panelPassword);
459. panelPassword.add(panelBtnLoginSubmitMiddle);
461. btnLoginSubmit = **new** JButton("Submit");
462. panelBtnLoginSubmitMiddle.add(btnLoginSubmit);
464. panelRightTop = **new** JPanel();
465. sl\_panelRight.putConstraint(Util.NORTH, panelRightContent, 27, Util.NORTH, panelRightTop);
466. panelRightTop.setBorder(**new** BevelBorder(BevelBorder.RAISED, **null**, **null**, **null**, **null**));
467. sl\_panelRight.putConstraint(Util.NORTH, panelRightTop, 0, Util.NORTH, panelRight);
468. sl\_panelRight.putConstraint(Util.WEST, panelRightTop, 0, Util.WEST, panelRightContent);
469. sl\_panelRight.putConstraint(Util.SOUTH, panelRightTop, 0, Util.NORTH, panelRightContent);
470. sl\_panelRight.putConstraint(Util.EAST, panelRightTop, 0, Util.EAST, panelRightContent);
471. panelRight.add(panelRightTop);
473. lblCurrentPane = **new** JLabel("THIS\_SHOULD\_BE\_AUTO\_REPLACED");
474. panelRightTop.add(lblCurrentPane);
475. GridBagLayout gbl\_panelNewTicket = **new** GridBagLayout();
476. gbl\_panelNewTicket.columnWidths = **new** **int**[] { 12, 179, 112, 105, 46, 137, 0, 0, 0 };
477. gbl\_panelNewTicket.rowHeights = **new** **int**[] { 50, 60, 36, 55, 161, 121, 91, 0, 1 };
478. gbl\_panelNewTicket.columnWeights = **new** **double**[] { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, Double.MIN\_VALUE };
479. gbl\_panelNewTicket.rowWeights = **new** **double**[] { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, Double.MIN\_VALUE };
480. panelNewTicket.setLayout(gbl\_panelNewTicket);
482. panelNTShortDesc = **new** JPanel();
483. GridBagConstraints gbc\_panelNTShortDesc = **new** GridBagConstraints();
484. gbc\_panelNTShortDesc.insets = **new** Insets(0, 0, 5, 5);
485. gbc\_panelNTShortDesc.fill = GridBagConstraints.BOTH;
486. gbc\_panelNTShortDesc.gridheight = 2;
487. gbc\_panelNTShortDesc.gridx = 1;
488. gbc\_panelNTShortDesc.gridy = 0;
489. panelNewTicket.add(panelNTShortDesc, gbc\_panelNTShortDesc);
490. sl\_panelNTShortDesc = **new** SpringLayout();
491. panelNTShortDesc.setLayout(sl\_panelNTShortDesc);
493. lblShortDesc = **new** JLabel("Short description / name");
494. sl\_panelNTShortDesc.putConstraint(Util.NORTH, lblShortDesc, PADDING, Util.NORTH, panelNTShortDesc);
495. lblShortDesc.setHorizontalAlignment(SwingConstants.CENTER);
496. sl\_panelNTShortDesc.putConstraint(Util.WEST, lblShortDesc, PADDING, Util.WEST, panelNTShortDesc);
497. sl\_panelNTShortDesc.putConstraint(Util.EAST, lblShortDesc, -PADDING, Util.EAST, panelNTShortDesc);
498. panelNTShortDesc.add(lblShortDesc);
500. panelTextAreaShortDesc = **new** JPanel();
501. sl\_panelNTShortDesc.putConstraint(Util.NORTH, panelTextAreaShortDesc, PADDING, Util.SOUTH,lblShortDesc);
502. sl\_panelNTShortDesc.putConstraint(Util.WEST, panelTextAreaShortDesc, PADDING, Util.WEST,panelNTShortDesc);
503. sl\_panelNTShortDesc.putConstraint(Util.EAST, panelTextAreaShortDesc, -PADDING, Util.EAST,panelNTShortDesc);
504. panelTextAreaShortDesc.setBorder(**new** BevelBorder(BevelBorder.LOWERED, **null**, **null**, **null**, **null**));
505. sl\_panelNTShortDesc.putConstraint(Util.SOUTH, panelTextAreaShortDesc, -PADDING, Util.SOUTH,panelNTShortDesc);
506. panelNTShortDesc.add(panelTextAreaShortDesc);
507. panelTextAreaShortDesc.setLayout(**new** BorderLayout(0, 0));
509. textAreaShortDesc = **new** JTextArea();
510. panelTextAreaShortDesc.add(textAreaShortDesc);
511. sl\_panelNTShortDesc.putConstraint(Util.NORTH, textAreaShortDesc, 6, Util.SOUTH, lblShortDesc);
512. sl\_panelNTShortDesc.putConstraint(Util.WEST, textAreaShortDesc, PADDING, Util.WEST, panelNTShortDesc);
513. sl\_panelNTShortDesc.putConstraint(Util.SOUTH, textAreaShortDesc, -PADDING, Util.SOUTH, panelNTShortDesc);
514. sl\_panelNTShortDesc.putConstraint(Util.EAST, textAreaShortDesc, -PADDING, Util.EAST, panelNTShortDesc);
515. textAreaShortDesc.setColumns(10);
517. panelNTCategory = **new** JPanel();
518. GridBagConstraints gbc\_panelNTCategory = **new** GridBagConstraints();
519. gbc\_panelNTCategory.gridwidth = 2;
520. gbc\_panelNTCategory.fill = GridBagConstraints.BOTH;
521. gbc\_panelNTCategory.insets = **new** Insets(0, 0, 5, 5);
522. gbc\_panelNTCategory.gridx = 2;
523. gbc\_panelNTCategory.gridy = 0;
524. panelNewTicket.add(panelNTCategory, gbc\_panelNTCategory);
525. panelNTCategory.setLayout(**new** BorderLayout(0, 0));
527. comboBoxCategory = **new** JComboBox<Object>();
528. comboBoxCategory.setFont(**new** Font("Dialog", Font.BOLD, 15));
529. comboBoxCategory.setModel(**new** DefaultComboBoxModel(
530. **new** String[] { "THESE", "SHOULD", "BE", "AUTO", "REPLACED", "FROM", "THE", "CATEGORIES", "TABLE" }));
531. panelNTCategory.add(comboBoxCategory);
533. labelCategory = **new** JLabel("Category");
534. labelCategory.setHorizontalAlignment(SwingConstants.CENTER);
535. panelNTCategory.add(labelCategory, BorderLayout.NORTH);
537. Util.setupCategories(comboBoxCategory,**this**.dao); // grab list of possible categories
538. // from server's category table
540. panelNTDateStarted = **new** JPanel();
541. GridBagConstraints gbc\_panelDateStarted = **new** GridBagConstraints();
542. gbc\_panelDateStarted.gridwidth = 2;
543. gbc\_panelDateStarted.anchor = GridBagConstraints.NORTHWEST;
544. gbc\_panelDateStarted.insets = **new** Insets(5, 5, 5, 5);
545. gbc\_panelDateStarted.gridx = 4;
546. gbc\_panelDateStarted.gridy = 0;
547. panelNewTicket.add(panelNTDateStarted, gbc\_panelDateStarted);
548. panelNTDateStarted.setLayout(**new** BorderLayout(0, 0));
550. dateStartedChooser = **new** JDateChooser();
551. dateStartedChooser.setDateFormatString(PREFERRED\_TIME\_FORMAT);
552. panelNTDateStarted.add(dateStartedChooser);
554. labelDateStarted = **new** JLabel("Date started");
555. labelDateStarted.setHorizontalAlignment(SwingConstants.CENTER);
556. panelNTDateStarted.add(labelDateStarted, BorderLayout.NORTH);
558. panelNTSeverity = **new** JPanel();
559. GridBagConstraints gbc\_panelSeverity = **new** GridBagConstraints();
560. gbc\_panelSeverity.fill = GridBagConstraints.HORIZONTAL;
561. gbc\_panelSeverity.gridwidth = 2;
562. gbc\_panelSeverity.insets = **new** Insets(0, 0, 5, 5);
563. gbc\_panelSeverity.gridx = 2;
564. gbc\_panelSeverity.gridy = 1;
565. panelNewTicket.add(panelNTSeverity, gbc\_panelSeverity);
566. panelNTSeverity.setLayout(**new** BorderLayout(0, 0));
568. labelSeverity = **new** JLabel("Severity");
569. labelSeverity.setHorizontalAlignment(SwingConstants.CENTER);
570. panelNTSeverity.add(labelSeverity, BorderLayout.NORTH);
572. comboBoxSeverity = **new** JComboBox<String>();
573. comboBoxSeverity.setModel(**new** DefaultComboBoxModel(**new** String[] { "THESE", "SHOULD", "BE", "POPULATED", "WITH",
574. "ENTRIES", "FROM", "THE", "SEVERITY", "TABLE" }));
575. comboBoxSeverity.setFont(**new** Font("Dialog", Font.BOLD, 15));
576. panelNTSeverity.add(comboBoxSeverity, BorderLayout.CENTER);

579. Util.setupSeverities(comboBoxSeverity, **this**.dao);   // grab list of possible severities
580. // from server's severity table
582. panelNTDateEnded = **new** JPanel();
583. GridBagConstraints gbc\_panelNTDateEnded = **new** GridBagConstraints();
584. gbc\_panelNTDateEnded.anchor = GridBagConstraints.WEST;
585. gbc\_panelNTDateEnded.gridwidth = 2;
586. gbc\_panelNTDateEnded.insets = **new** Insets(5, 5, 5, 5);
587. gbc\_panelNTDateEnded.gridx = 4;
588. gbc\_panelNTDateEnded.gridy = 1;
589. panelNewTicket.add(panelNTDateEnded, gbc\_panelNTDateEnded);
590. panelNTDateEnded.setLayout(**new** BorderLayout(0, 0));
592. dateEndedChooser = **new** JDateChooser();
593. dateEndedChooser.setDateFormatString("MMM d, yyyy, hh:mm:ss aa");
594. panelNTDateEnded.add(dateEndedChooser, BorderLayout.CENTER);
596. labelDateEnded = **new** JLabel("Date ended");
597. labelDateEnded.setHorizontalAlignment(SwingConstants.CENTER);
598. panelNTDateEnded.add(labelDateEnded, BorderLayout.NORTH);
600. panelNTLongDesc = **new** JPanel();
601. GridBagConstraints gbc\_panelNTLongDesc = **new** GridBagConstraints();
602. gbc\_panelNTLongDesc.gridheight = 3;
603. gbc\_panelNTLongDesc.fill = GridBagConstraints.BOTH;
604. gbc\_panelNTLongDesc.insets = **new** Insets(0, 0, 5, 5);
605. gbc\_panelNTLongDesc.gridwidth = 6;
606. gbc\_panelNTLongDesc.gridx = 1;
607. gbc\_panelNTLongDesc.gridy = 2;
608. panelNewTicket.add(panelNTLongDesc, gbc\_panelNTLongDesc);
609. sl\_panelNTLongDesc = **new** SpringLayout();
610. panelNTLongDesc.setLayout(sl\_panelNTLongDesc);
612. lblLongDescription = **new** JLabel("Long description");
613. sl\_panelNTLongDesc.putConstraint(Util.NORTH, lblLongDescription, 0, Util.NORTH, panelNTLongDesc);
614. sl\_panelNTLongDesc.putConstraint(Util.WEST, lblLongDescription, PADDING, Util.WEST, panelNTLongDesc);
615. sl\_panelNTLongDesc.putConstraint(Util.EAST, lblLongDescription, -PADDING, Util.EAST, panelNTLongDesc);
616. lblLongDescription.setHorizontalAlignment(SwingConstants.CENTER);
617. panelNTLongDesc.add(lblLongDescription);
619. panelTextAreaLongDesc = **new** JPanel();
620. panelTextAreaLongDesc.setBorder(**new** BevelBorder(BevelBorder.LOWERED, **null**, **null**, **null**, **null**));
621. sl\_panelNTLongDesc.putConstraint(Util.NORTH, panelTextAreaLongDesc, PADDING, Util.SOUTH,
622. lblLongDescription);
623. sl\_panelNTLongDesc.putConstraint(Util.SOUTH, panelTextAreaLongDesc, -PADDING, Util.SOUTH,
624. panelNTLongDesc);
625. sl\_panelNTLongDesc.putConstraint(Util.WEST, panelTextAreaLongDesc, PADDING, Util.WEST, panelNTLongDesc);
626. sl\_panelNTLongDesc.putConstraint(Util.EAST, panelTextAreaLongDesc, -PADDING, Util.EAST, panelNTLongDesc);
627. panelNTLongDesc.add(panelTextAreaLongDesc);
628. panelTextAreaLongDesc.setLayout(**new** BorderLayout(0, 0));
630. scrollPaneLongDesc = **new** JScrollPane();
631. panelTextAreaLongDesc.add(scrollPaneLongDesc, BorderLayout.CENTER);
633. textAreaLongDesc = **new** JTextArea();
634. scrollPaneLongDesc.setViewportView(textAreaLongDesc);
635. textAreaLongDesc.setColumns(10);
637. btnNTSubmitNewTicket = **new** JButton(BUTTON\_S\_SUBMIT\_NEW\_TICKET\_S);
638. GridBagConstraints gbc\_btnNTSubmitNewTicket = **new** GridBagConstraints();
639. gbc\_btnNTSubmitNewTicket.fill = GridBagConstraints.BOTH;
640. gbc\_btnNTSubmitNewTicket.gridwidth = 2;
641. gbc\_btnNTSubmitNewTicket.insets = **new** Insets(0, 0, 5, 5);
642. gbc\_btnNTSubmitNewTicket.gridx = 2;
643. gbc\_btnNTSubmitNewTicket.gridy = 5;
644. panelNewTicket.add(btnNTSubmitNewTicket, gbc\_btnNTSubmitNewTicket);
645. btnNTSubmitNewTicket.setName("btnNewTiket");
647. panelNTButtonSubmitNewTicket = **new** JPanel();
648. GridBagConstraints gbc\_panelNTButtonSubmitNewTicket = **new** GridBagConstraints();
649. gbc\_panelNTButtonSubmitNewTicket.insets = **new** Insets(0, 0, 5, 5);
650. gbc\_panelNTButtonSubmitNewTicket.fill = GridBagConstraints.BOTH;
651. gbc\_panelNTButtonSubmitNewTicket.gridwidth = 2;
652. gbc\_panelNTButtonSubmitNewTicket.gridx = 2;
653. gbc\_panelNTButtonSubmitNewTicket.gridy = 6;
654. panelNewTicket.add(panelNTButtonSubmitNewTicket, gbc\_panelNTButtonSubmitNewTicket);
655. panelNTButtonSubmitNewTicket.setLayout(**new** BorderLayout(0, 0));
656. panelSearch.setLayout(**new** BorderLayout(0, 0));
658. splitPaneSearch = **new** JSplitPane();
659. splitPaneSearch.setOrientation(JSplitPane.VERTICAL\_SPLIT);
660. panelSearch.add(splitPaneSearch);
662. panelSearchTop = **new** JPanel();
663. splitPaneSearch.setLeftComponent(panelSearchTop);
665. lblRecordsThatHave = **new** JLabel("Records that have");
666. panelSearchTop.add(lblRecordsThatHave);
668. comboBoxField = **new** JComboBox<String>();
669. comboBoxField.setModel(**new** DefaultComboBoxModel<>(**new** String[] { "THESE", "SHOULD", "BE", "REPLACED", "BY",
670. "VALID", "FIELDS", "THAT", "ARE", "PART", "OF", "THE", "TICKETS", "OR", "USERS", "TABLE" }));
671. panelSearchTop.add(comboBoxField);
673. comboBoxOperator = **new** JComboBox<String>();
674. comboBoxOperator.setModel(**new** DefaultComboBoxModel<>(**new** String[] { "THESE", "SHOULD", "BE", "NUMBERICAL", "OR",
675. "LOGICAL", "OPERATORS", "VALID", "FOR", "THEIR", "TWO", "OPERANDS", "", "MAYBE", "MAKE", "A", "HELPER",
676. "FUNCTION", "OR", "AN", "ENUM", "OR", "SOMETHING" }));
677. panelSearchTop.add(comboBoxOperator);
679. textField = **new** JTextField();
680. panelSearchTop.add(textField);
681. textField.setColumns(10);
683. btnSearchThroughRecords = **new** JButton("Go");
685. panelSearchTop.add(btnSearchThroughRecords);
687. panelSearchBottom = **new** JPanel();
688. splitPaneSearch.setRightComponent(panelSearchBottom);
689. panelSearchBottom.setLayout(**new** BorderLayout(0, 0));
691. panelSearchResultsTop = **new** JPanel();
692. panelSearchBottom.add(panelSearchResultsTop, BorderLayout.NORTH);
694. JScrollPane scrollPaneResults = **new** JScrollPane();
696. tableResults = **new** JTable();
697. tableResults.setModel(**new** DefaultTableModel(**new** Object[][] {}, **new** String[] { "hi", "please", "replace", "me" }));
698. scrollPaneResults.setViewportView(tableResults);
700. JPopupMenu popupMenuResults = **new** JPopupMenu();
702. JMenuItem deleteTicketMI = **new** JMenuItem("Delete -----------------");
704. JMenuItem modifyTicketMI = **new** JMenuItem("Modify -----------------"); //TODO why can't I resize these?
706. popupMenuResults.add(deleteTicketMI,0);
707. popupMenuResults.add(modifyTicketMI,1);
709. tableResults.setComponentPopupMenu(popupMenuResults); //add popup menu for table
711. panelSearchBottom.add(scrollPaneResults, BorderLayout.CENTER);

714. panelModifyTicket = **new** JPanel();
715. panelModifyTicket.setName("panelModifyTicket");
716. panelRightContent.add(panelModifyTicket, panelModifyTicket.getName());
717. springLayout\_1 = **new** SpringLayout();
718. panelModifyTicket.setLayout(springLayout\_1);
720. btnSubmitChanges = **new** JButton("Submit Changes");
722. springLayout\_1.putConstraint(SpringLayout.WEST, btnSubmitChanges, 303, SpringLayout.WEST, panelModifyTicket);
723. springLayout\_1.putConstraint(SpringLayout.SOUTH, btnSubmitChanges, -54, SpringLayout.SOUTH, panelModifyTicket);
724. panelModifyTicket.add(btnSubmitChanges);
726. JPanel panel = **new** JPanel();
727. springLayout\_1.putConstraint(SpringLayout.NORTH, panel, 10, SpringLayout.NORTH, panelModifyTicket);
728. springLayout\_1.putConstraint(SpringLayout.WEST, panel, 10, SpringLayout.WEST, panelModifyTicket);
729. panelModifyTicket.add(panel);
731. JLabel lblTicketId = **new** JLabel("Ticket ID");
732. panel.add(lblTicketId);
734. textField\_1 = **new** JTextField();
735. panel.add(textField\_1);
736. springLayout\_1.putConstraint(SpringLayout.NORTH, textField\_1, 60, SpringLayout.NORTH, panelModifyTicket);
737. springLayout\_1.putConstraint(SpringLayout.WEST, textField\_1, 61, SpringLayout.WEST, panelModifyTicket);
738. textField\_1.setColumns(10);
739. splitPane.setDividerLocation(DEFAULT\_DIVIDER\_WIDTH);
741. btnLogin.addActionListener(**new** ActionListener()
742. {
743. /\*\*\*
744. \* 'Login' button clicked.
745. \*/
746. **public** **void** actionPerformed(ActionEvent e)
747. {
748. modifyRightPane(panelLogin, e, btnLogin.getText());
749. }
750. });

753. btnLogout.addActionListener(**new** ActionListener()
754. {
755. /\*\*\*
756. \* 'Logout' button clocked.
757. \*/
759. @Override
760. **public** **void** actionPerformed(ActionEvent e)
761. {
762. Util.disableButtons(panelMenu, **new** String[]{BUTTON\_LOGIN\_NAME}); //disable all but log in
763. modifyRightPane(panelLogin, e, btnLogin.getText());
765. textFieldPassword.setText("");
766. textFieldUsername.setText("");
767. lblWhosLoggedIn.setText(GUIC.NOT\_LOGGED\_IN.s());
769. textFieldUsername.grabFocus();
771. dao.logout(); //log them out
772. }
773. });

776. popupMenuResults.addPopupMenuListener(**new** PopupMenuListener()
777. {
778. /\*\*\*
779. \* When the popup menu gets invoked.
780. \*/
781. @Override
782. **public** **void** popupMenuWillBecomeVisible(PopupMenuEvent pme)
783. {
785. SwingUtilities.invokeLater(**new** Runnable()
786. {
787. /\*\*\*
788. \* This part selects the table when you right-click.
789. \*/
790. @Override
791. **public** **void** run()
792. {
793. JPopupMenu parentMenu = (JPopupMenu)pme.getSource();

796. **int** rowAtPoint = tableResults.rowAtPoint(SwingUtilities.convertPoint(popupMenuResults, **new** Point(0, 0), tableResults));
797. **if** (rowAtPoint > -1)
798. {
799. tableResults.setRowSelectionInterval(rowAtPoint, rowAtPoint);
800. Util.printf("Setting selection to row [%d] from rightclick.\n",rowAtPoint,rowAtPoint);
802. DefaultTableModel dtm = (DefaultTableModel)tableResults.getModel(); //get data model
804. **int** id = (**int**) Integer.parseInt((String)dtm.getValueAt(rowAtPoint, 0)); //get 0th item at nth row
806. JMenuItem jmiDelete = ((JMenuItem)((JPopupMenu)pme.getSource()).getComponent(0)); //get menu item so we can make it say "Delete ticket id {}"
807. JMenuItem jmiModify = ((JMenuItem)((JPopupMenu)pme.getSource()).getComponent(1)); //get menu item so we can make it say "Modify ticket id {}"
809. jmiDelete.setText(String.format("Delete Ticket ID '%d'",id));
810. jmiModify.setText(String.format("Modify Ticket ID '%d'",id));

813. //TODO why do I have to put a bunch of ------'s? Can't repaint the jpopupmenu to scale to width of text? hm..
814. }
815. }
816. });
817. }
819. @Override
820. **public** **void** popupMenuWillBecomeInvisible(PopupMenuEvent arg0)
821. {
822. // TODO Auto-generated method stub
823. }
825. @Override
826. **public** **void** popupMenuCanceled(PopupMenuEvent arg0)
827. {
828. // TODO Auto-generated method stub
829. }
830. });
832. modifyTicketMI.addActionListener(**new** AbstractAction()
833. {
834. /\*\*\*
835. \* When someone clicks the "Modify ticket n" menu item.
836. \*/
837. @Override
838. **public** **void** actionPerformed(ActionEvent arg0)
839. {
841. **int** row = Util.getSelectedPos(tableResults)[0];
843. DefaultTableModel dtm = (DefaultTableModel) tableResults.getModel();
845. **int** id = ((**int**) Integer.parseInt((String)dtm.getValueAt(row, 0)));
847. System.out.printf("Setting up 'new ticket' view for ticket ID '%d'\n",id);

850. ResultSet oneticket = dao.getTicketRS(id);

853. //first, fill in the field and change the name of the "submit" button;
854. **try**
855. {
856. oneticket.next();
858. textAreaShortDesc.setText(oneticket.getString(SQLC.SHORT\_DESC\_COLUMN\_NAME.s()));
859. textAreaLongDesc.setText(oneticket.getString(SQLC.LONG\_DESC\_COLUMN\_NAME.s()));
860. comboBoxCategory.setSelectedIndex(oneticket.getInt(SQLC.CATEGORY\_ID\_COLUMN\_NAME.s()));
861. comboBoxSeverity.setSelectedIndex(oneticket.getInt(SQLC.SEVERITY\_ID\_COLUMN\_NAME.s()));
862. dateStartedChooser.setDate(oneticket.getDate(SQLC.START\_DATE\_COLUMN\_NAME.s()));
863. dateEndedChooser.setDate(oneticket.getDate(SQLC.END\_DATE\_COLUMN\_NAME.s()));
865. btnNTSubmitNewTicket.setText(btnNTSubmitNewTicket.getText() + " (modify)");
866. \_ARE\_MODIFYING\_TICKET = **true**;
867. \_PREV\_TICKET\_ID = id;


871. }
872. **catch** (SQLException e)
873. {
874. System.out.printf("Failed to populate 'new ticket' view with ticket ID %d's info!",id);
875. System.out.println(e.getSQLState());
876. System.out.println(e.getMessage());
877. e.printStackTrace();
879. }


883. modifyRightPane(panelNewTicket, btnModifyTicket.getText()); //use our 'new ticket' GUI elt..why not?


887. }
888. });
890. deleteTicketMI.addActionListener(**new** AbstractAction()
891. {
892. **private** **static** **final** **long** serialVersionUID = 2440586029673540511L;
894. /\*\*\*
895. \* Someone clicks "Delete ticket n" on our right-click menu.
896. \*/
898. @Override
899. **public** **void** actionPerformed(ActionEvent arg0)
900. {
901. **int** row = Util.getSelectedPos(tableResults)[0];
903. DefaultTableModel dtm = (DefaultTableModel) tableResults.getModel();
905. **int** id = ((**int**) Integer.parseInt((String)dtm.getValueAt(row, 0)));
907. System.out.printf("We should delete ticket with ID '%d'.\n",id);
909. **if**(dao.deleteTicket(id)) //if successfully deleted
910. {
911. dtm.removeRow(row);
912. }
913. **else**
914. {
915. System.out.println("Failed to delete...:(");
916. }

919. }
920. });


924. btnModifyTicket.addActionListener(**new** ActionListener()
925. {
926. /\*\*\*
927. \* 'Modify ticket' button clicked.
928. \*/
929. **public** **void** actionPerformed(ActionEvent e)
930. {
931. btnModifyTicket.setEnabled(**false**); //we're not gonna use it.
932. btnModifyTicket.setToolTipText("Use the right-click menu in the \"Search\" view to modify a ticket.");
933. }
934. });




940. btnSubmitChanges.addActionListener(**new** ActionListener()
941. {
942. /\*\*\*
943. \* 'Submit changes' button clicked.
944. \*/
945. **public** **void** actionPerformed(ActionEvent e)
946. {
947. }
948. });
950. btnSearch.addActionListener(**new** ActionListener()
951. {
952. /\*\*\*
953. \* 'Search' button clicked.
954. \*/
955. **public** **void** actionPerformed(ActionEvent e)
956. {
957. modifyRightPane(panelSearch, e, btnSearch.getText());
959. comboBoxField.removeAllItems();
960. comboBoxOperator.removeAllItems();
962. **for** (**int** i = 0; i < Dao.OPERATOR\_TYPES\_S.length; i++)
963. {
964. // add all operator types
965. comboBoxOperator.addItem(Dao.OPERATOR\_TYPES\_S[i]);
966. }
968. **for** (**int** i = 0; i < Dao.VALID\_SEARCH\_FIELDS\_USERS\_S.length; i++)
969. {
970. // add valid fields that belong to users table
971. comboBoxField.addItem(Dao.VALID\_SEARCH\_FIELDS\_USERS\_S[i]);
972. }
974. **for** (**int** i = 0; i < Dao.VALID\_SEARCH\_FIELDS\_TICKETS\_S.length; i++)
975. {
976. // add valid fields that belong to tickets table
977. comboBoxField.addItem(Dao.VALID\_SEARCH\_FIELDS\_TICKETS\_S[i]);
978. }


982. ResultSet rs = **null**; //get results for table
984. **try**
985. {
986. //execute default query to populate search pane
987. System.out.printf("About to execute this query to populate our default results view: \n'%s'\n",SQLC.SEARCH\_PANE\_DEFAULT\_QUERY\_PAPADEMAS.toString());
988. rs = dao.c.createStatement().executeQuery(SQLC.SEARCH\_PANE\_DEFAULT\_QUERY\_PAPADEMAS.toString());
989. }
990. **catch** (SQLException e1)
991. {
992. e1.printStackTrace();
993. }

996. ArrayList<ArrayList<Object>> data = dao.resultSetToList(rs); //get 2d list of data
997. ArrayList<ArrayList<Object>> dataHeaders = dao.resultSetMetadataToList(rs);


1001. Object[][] dataA = data.stream().map(aRow -> aRow.toArray(**new** Object[0])).toArray(Object[][]::**new**);
1002. //ArrayList<ArrayList<Object>> converted to Object[][]
1004. Object[] dataHeadersA = dataHeaders.get(0).toArray();
1005. //0th header converted to Object[]
1007. // set up the table to have a default view of all notes...
1008. DefaultTableModel dtm = **new** DefaultTableModel(dataA,dataHeadersA);

1011. tableResults.setModel(dtm);
1013. //              dao.resultSetToList(rs)
1015. }
1017. });
1019. btnSearchThroughRecords.addActionListener(**new** ActionListener()
1020. {
1021. /\*\*\*
1022. \* Someone wants to search for records!
1023. \*/
1024. **public** **void** actionPerformed(ActionEvent arg0)
1025. {
1027. }
1028. });
1030. btnNewTicket.addActionListener(**new** ActionListener()
1031. {
1032. /\*\*\*
1033. \* 'New ticket' button clicked.
1034. \*/
1035. **public** **void** actionPerformed(ActionEvent e)
1036. {
1037. ((JButton)e.getSource()).setText(BUTTON\_NEW\_TICKET\_S);
1038. modifyRightPane(panelNewTicket, e, btnNewTicket.getText());
1039. }
1040. });
1042. textFieldUsername.addActionListener(**new** ActionListener()
1043. {
1044. /\*\*\*
1045. \* Someone presses ENTER on our username field, goto password field
1046. \*/
1047. **public** **void** actionPerformed(ActionEvent e)
1048. {
1049. textFieldPassword.grabFocus();
1050. }
1051. });
1053. textFieldPassword.addActionListener(**new** ActionListener()
1054. {
1055. /\*\*\*
1056. \* Someone presses ENTER on our password field
1057. \*/
1058. **public** **void** actionPerformed(ActionEvent arg0)
1059. {
1060. btnLoginSubmit.doClick();
1061. }
1063. });
1065. btnLoginSubmit.addActionListener(**new** ActionListener()
1066. {
1067. /\*\*\*
1068. \* Someone clicks on our "submit" for password + username
1069. \*/
1070. **public** **void** actionPerformed(ActionEvent arg0)
1071. {
1072. String[] login = getLogin();
1073. doLogin(login[0], login[1]);
1074. }
1075. });
1077. frame.addWindowListener(**new** WindowAdapter()
1078. {
1079. /\*\*\*
1080. \* When window is opened, select username field.
1081. \*/
1082. **public** **void** windowOpened(WindowEvent we)
1083. {
1084. textFieldUsername.requestFocus();
1086. }
1087. });
1089. btnNTSubmitNewTicket.addActionListener(**new** ActionListener()
1090. {
1091. /\*\*\*
1092. \* Someone clicks "submit new ticket"
1093. \*/
1094. **public** **void** actionPerformed(ActionEvent arg0)
1095. {
1097. **if**(\_ARE\_MODIFYING\_TICKET) //to only delete if they submit
1098. {
1099. dao.deleteTicket(\_PREV\_TICKET\_ID); //delete previous ticket
1101. \_ARE\_MODIFYING\_TICKET = **false**;
1102. \_PREV\_TICKET\_ID = -1;
1103. btnNTSubmitNewTicket.setText(BUTTON\_S\_SUBMIT\_NEW\_TICKET\_S);
1104. }
1106. dateStartedChooser.setBorder(borderDefault);
1107. dateEndedChooser.setBorder(borderDefault);  //reset borders
1109. dateStartedChooser.setToolTipText(PREFERRED\_TIME\_FORMAT);
1110. dateEndedChooser.setToolTipText(PREFERRED\_TIME\_FORMAT);
1111. dateStartedChooser.getCalendarButton().setToolTipText(PREFERRED\_TIME\_FORMAT);
1112. dateEndedChooser.getCalendarButton().setToolTipText(PREFERRED\_TIME\_FORMAT); //reset tooltips
1113. dateStartedChooser.getDateEditor().getUiComponent().setToolTipText(PREFERRED\_TIME\_FORMAT);
1114. dateStartedChooser.getDateEditor().getUiComponent().setToolTipText(PREFERRED\_TIME\_FORMAT);

1117. Boolean allFormsOK = **true**;
1118. **if**(dateStartedChooser.getDate() == **null**) //if start date isn't filled out, complain
1119. {
1120. allFormsOK = **false**;
1121. dateStartedChooser.setBorder(borderBad);
1122. dateStartedChooser.requestFocus();
1123. dateStartedChooser.getDateEditor().getUiComponent().setToolTipText(GUIC.MUST\_ENTER\_DATE + " (" + PREFERRED\_TIME\_FORMAT + ")");
1124. }
1126. **if**(allFormsOK)
1127. {
1128. dao.submitTicket(dao.USER\_NAME,                     //username
1129. textAreaShortDesc.getText(),        //short desc
1130. textAreaLongDesc.getText(),         //long desc
1131. (Integer)Integer.parseInt(((String)comboBoxCategory.getSelectedItem()).substring(0, 1)),    //1st char of category dropdown as Integer
1132. (Integer)Integer.parseInt(((String)comboBoxSeverity.getSelectedItem()).substring(0, 1)),    //1st char of severity dropdown as Integer
1133. dateStartedChooser.getDate(),
1134. dateEndedChooser.getDate());
1136. // clear the text we just used.
1137. textAreaShortDesc.setText("");
1138. textAreaLongDesc.setText("");
1139. comboBoxCategory.setSelectedIndex(0);
1140. comboBoxSeverity.setSelectedIndex(0);
1141. ((JTextField)dateStartedChooser.getDateEditor().getUiComponent()).setText(""); //set text of date started to nothing
1142. ((JTextField)dateEndedChooser.getDateEditor().getUiComponent()).setText(""); //set text of date started to nothing
1144. }
1145. }
1146. });


1150. Util.formatButtons(panelMenu);
1152. btnLogin.doClick(); // to setup the top text
1153. lblLoginErrorMsg.setText(""); // empty login error message
1155. Util.disableButtons(panelMenu,**new** String[] {BUTTON\_LOGIN\_NAME}); //disable all buttons until user logs in
1156. Util.setTooltips(panelMenu, **new** String[] {BUTTON\_LOGIN\_NAME}, "You must log in to be able to use this button.");





1163. }
1164. }

# [Util.java](#Source_code_directory)

1. **import** java.awt.Component;
2. **import** java.io.BufferedReader;
3. **import** java.io.BufferedWriter;
4. **import** java.io.File;
5. **import** java.io.FileNotFoundException;
6. **import** java.io.FileReader;
7. **import** java.io.FileWriter;
8. **import** java.io.IOException;
9. **import** java.io.Writer;
10. **import** java.sql.ResultSet;
11. **import** java.sql.ResultSetMetaData;
12. **import** java.sql.SQLException;
13. **import** java.sql.Statement;
14. **import** java.util.ArrayList;
15. **import** java.util.Arrays;
17. **import** javax.swing.DefaultComboBoxModel;
18. **import** javax.swing.JButton;
19. **import** javax.swing.JComboBox;
20. **import** javax.swing.JComponent;
21. **import** javax.swing.JPanel;
22. **import** javax.swing.JTable;
23. **import** javax.swing.SpringLayout;
24. **import** javax.swing.table.DefaultTableModel;
26. /\*\*\*
27. \*
28. \* @author henryfbp
29. \*  A utility class containing lots of different useful functions.
30. \*/
31. **public** **class** Util
32. {
34. **public** **static** **final** String NORTH = SpringLayout.NORTH;
35. **public** **static** **final** String EAST = SpringLayout.EAST;
36. **public** **static** **final** String SOUTH = SpringLayout.SOUTH;
37. **public** **static** **final** String WEST = SpringLayout.WEST;
39. **public** **static** **void** printf(String s, Object... args)
40. {
41. System.out.printf(s, args);
42. }
44. **public** **static** **void** print(String s)
45. {
46. System.out.print(s);
47. }
49. **public** **static** **void** println(String s)
50. {
51. System.out.println(s);
52. }
54. /\*\*\*
55. \* @param duplicate The string to repeat.
56. \* @param length How many times should we repeat?
57. \* @return The duplicated string.
58. \*/
59. **private** **static** String stringLine(String duplicate, **int** length)
60. {
61. String ret = "";
63. **for** (**int** i = 0; i < length; i++)
64. {
65. ret += duplicate;
66. }
67. **return** ret;
68. }
70. **public** **static** String oppositeDirection(String direction)
71. {
72. **switch** (direction.toUpperCase().charAt(0))
73. {
74. **case** 'N':
75. **return** SOUTH;
76. **case** 'E':
77. **return** WEST;
78. **case** 'S':
79. **return** NORTH;
80. **case** 'W':
81. **return** EAST;
82. **default**:
83. **return** **null**;
84. }
86. }
88. /\*\*\*
89. \* @param ar The list to print.
90. \*/
91. **public** **static** **void** printList(ArrayList<String> ar)
92. {
93. **for** (**int** i = 0; i < ar.size(); i++)
94. {
95. System.out.println(ar.get(i));
96. }
97. **return**;
98. }
100. /\*\*\*
101. \*
102. \* @param rs The resultSet to be formatted.
103. \* @return  A pretty, human-readable list of strings.
104. \*/
105. **public** **static** ArrayList<String> formatResultSet(ResultSet rs)
106. {
107. **int** pos = -1;
108. **try**
109. {
110. pos = rs.getRow(); // record current place
111. rs.beforeFirst(); // set place to zero
113. }
114. **catch** (SQLException e1)
115. {
116. e1.printStackTrace();
117. }
119. **int** largestColumn = getLargestColumn(rs);
120. String stringFormat = "%" + 13 + "s |";
121. String stringFormat2 = "%" + 15 + "s |";
123. ArrayList<String> ret = **new** ArrayList<>();
125. ResultSetMetaData rsmd;
126. **try**
127. {
128. rsmd = rs.getMetaData();
129. ret.add(String.format(stringFormat2, "Column name:"));
130. ret.add(String.format(stringFormat2, "Column type:"));
132. // System.out.println("metadata:");
133. **for** (**int** i = 1; i < rsmd.getColumnCount(); i++)
134. {
135. // printf("i = %2d, current thing = %10s, %10s\n",
136. // i,rsmd.getColumnName(i),rsmd.getColumnTypeName(i));
137. ret.set(0, ret.get(0) + String.format(stringFormat, rsmd.getColumnName(i)) // tack on a name
138. );
140. ret.set(1, ret.get(1) + String.format(stringFormat, rsmd.getColumnTypeName(i)) // tack on a type
141. );
142. }
144. // System.out.println("length of my lines should be " + ret.get(0).length());
145. ret.add(stringLine("-", ret.get(0).length()));
146. ret.add(0, stringLine("\_", ret.get(0).length()));
148. **int** numCols = 0;
150. **while** (rs.next())
151. {
152. String oneColumn = String.format(stringFormat2, "");
153. **for** (**int** i = 1; i < rsmd.getColumnCount(); i++)
154. {
155. // System.out.println("HI. ONECOLUMN = "+oneColumn);
156. oneColumn += String.format(stringFormat, rs.getString(i));
157. }
159. ret.add(oneColumn);
160. numCols++;
161. }
163. **if** (numCols == 0) // our resultSet is empty....
164. {
165. ret.add("    Empty resultset...");
166. }
168. }
169. **catch** (SQLException e)
170. {
171. e.printStackTrace();
172. }
174. **try**
175. {
176. rs.absolute(pos); // reset to position it was at
177. }
178. **catch** (SQLException e)
179. {
180. e.printStackTrace();
181. }
183. **return** ret;
184. }

187. /\*\*\*
188. \*
189. \* @param rs The resultSet you want to find the largest column in.
190. \* @return The column that has the largest amount of characters in it.
191. \*/
192. **public** **static** **int** getLargestColumn(ResultSet rs)
193. {
194. **int** largest = -1;
195. ResultSetMetaData rsmd = **null**;
196. **try**
197. {
198. rsmd = rs.getMetaData();
200. **for** (**int** i = 1; i < rsmd.getColumnCount(); i++)
201. {
202. **if** (rsmd.getColumnDisplaySize(i) > largest)
203. {
204. largest = rsmd.getColumnDisplaySize(i);
205. }
206. }
208. }
209. **catch** (SQLException e)
210. {
211. e.printStackTrace();
212. }
214. **return** largest;
216. }


220. /\*\*\*
221. \* Note: the reason for this existing is me not wanting to store the professor's
222. \* host, login, and password as a hardcoded string on a public git repo.
223. \* Instead, it will be stored in a .gitignore'd file and read in later.
224. \*
225. \* @param path
226. \*            A path to a text file containing a newline-separated list of host,
227. \*            login, password.
228. \* @return A list of logins in the following format:<br>
229. \*         <code>
230. \* {<br>
231. \*  {"localhost:3306","user","pass"},<br>
232. \*  {"129.999.0.1:3604","admin","nimda"},<br>
233. \*  {"goggles.ru:3604","hithere","ello"}<br>
234. \* }<br>
235. \* </code>
236. \*/
237. **public** **static** ArrayList<ArrayList<String>> getLoginsFromFile(String path)
238. {
239. ArrayList<ArrayList<String>> ret = **new** ArrayList<>();
240. BufferedReader br = **null**;
241. **try**
242. {
243. br = **new** BufferedReader(**new** FileReader(**new** File(path)));
245. String line = **null**;
246. String words[] = **null**;
247. ArrayList<String> lineArr = **null**;
249. **while** ((line = br.readLine()) != **null**)
250. {
251. lineArr = **new** ArrayList<>();
252. words = line.split(",");
253. **for** (String word : words)
254. {
255. lineArr.add(word);
256. }
258. **if** (lineArr.size() > 0)
259. {
260. System.out.printf("Adding this: '%s'\n", lineArr);
261. ret.add(lineArr);
262. }
263. }
265. }
266. **catch** (FileNotFoundException e)
267. {
268. printf("File '%s' not found. Creating it with default login info...", path);
270. **try**
271. {
272. BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(**new** File(path)));
274. bw.write(SQLC.DEFAULT\_LOGIN\_DATA+"");
276. bw.close();
277. }
278. **catch** (IOException e1)
279. {
280. e1.printStackTrace();
281. }

284. e.printStackTrace();
285. }
286. **catch** (IOException e)
287. {
288. e.printStackTrace();
289. }
290. **return** ret;
291. }
293. /\*\*\*
294. \* Makes all buttons inside of a JPanel disabled/unclickable.
295. \* @param buttonContainer   The container that has the buttons inside of it
296. \* @param excludedButtons   Names of buttons to not be disabled.
297. \* @param enableAll         Enable all buttons instead of disabling?
298. \*/
299. **public** **static** **void** disableButtons(JPanel buttonContainer, String[] excludedButtons, Boolean enableAll)
300. {
301. ArrayList<String> excludedButtonsAR = **new** ArrayList<>(Arrays.asList(excludedButtons));
303. ArrayList<JButton> menuButtons = getButtonsFromJPanel(buttonContainer);
304. SpringLayout tempSL = (SpringLayout) buttonContainer.getLayout();
306. **for**(**int** i = 0; i < menuButtons.size(); i++)
307. {
308. JButton tempButton = menuButtons.get(i);
310. tempButton.setEnabled(**false**); //assume it should be disabled
312. **if**(enableAll || excludedButtonsAR.contains(tempButton.getName()))
313. {
314. tempButton.setEnabled(**true**);
315. }
316. }
317. }
319. /\*\*\*
320. \*  Disable all buttons EXCEPT those specified in the excludedButtons list.
321. \* @param buttonContainer The container of buttons.
322. \* @param excludedButtons Button names to not be disabled.
323. \*/
324. **public** **static** **void** disableButtons(JPanel buttonContainer, String[] excludedButtons)
325. {
326. disableButtons(buttonContainer, excludedButtons, **false**);
327. }
329. **public** **void** disableButtons(JPanel buttonContainer)
330. {
331. disableButtons(buttonContainer, **new** String[] {});
332. }
334. **public** **static** **void** enableButtons(JPanel buttonContainer)
335. {
336. disableButtons(buttonContainer, **new** String[] {}, **true**);
337. }


341. /\*\*\*
342. \* Populates a comboBox with a list gotten from an SQL table.
343. \* The SQL table should look like this:
344. \* <pre>
345. \* <code>
346. \* +----+-------------+
347. \* | <b>id</b> |     str     |
348. \* +----+-------------+
349. \* | 1  |    bad      |
350. \* | 2  |    good     |
351. \* | 3  |    great    |
352. \* | 4  |  excellent  |
353. \* +----+-------------+
354. \* </pre>
355. \* </code>
356. \*
357. \* and produces this JComboBox:
358. \* <pre>
359. \* <code>
360. \* +------------------+
361. \* | 1 - bad          |
362. \* | 2 - good         |
363. \* | 3 - great        |
364. \* | 4 - excellent    |
365. \* +------------------+
366. \* </pre>
367. \* </code>
368. \* @param comboBox
369. \* @param tableName
370. \*/
371. **public** **static** **void** setupComboBox(JComboBox<String> comboBox, String tableName, Dao dao)
372. {
373. **try**
374. {
375. Statement s = dao.c.createStatement();
377. String query = String.format("SELECT \* FROM %s",tableName);
379. printf("Grabbing category list from database with the following query: \n'%s'\n",query);
381. ResultSet rs = s.executeQuery(query);
382. ResultSetMetaData rsmd = rs.getMetaData();
384. comboBox.removeAll();
385. comboBox.setModel(**new** DefaultComboBoxModel<>());
387. **while**(rs.next())
388. {
389. String onecol = "";
391. **for**(**int** i = 1; i <= rsmd.getColumnCount(); i++)
392. {
393. **if**(i != 1)
394. {
395. onecol += " - ";
396. }
398. onecol += rs.getString(i).toString();
399. }
401. printf("Adding string '%s' to comboBox!\n",onecol);
402. comboBox.addItem(onecol);
403. }



408. }
409. **catch** (SQLException e)
410. {
411. e.printStackTrace();
412. }
413. }
415. **public** **static** **void** setupCategories(JComboBox<String> comboBox, Dao dao)
416. {
417. setupComboBox(comboBox, SQLC.TABLE\_CATEGORIES.ts(), dao);
418. }
420. **public** **static** **void** setupSeverities(JComboBox<String> comboBox, Dao dao)
421. {
422. setupComboBox(comboBox, SQLC.TABLE\_SEVERITIES.ts(), dao);
423. }



428. **public** **static** Component getVisibleComponent(JPanel jp)
429. {
430. **for** (Component c : jp.getComponents())
431. {
432. **if** (c.isVisible())
433. {
434. **return** c;
435. }
436. }
437. **return** **null**;
438. }
440. **public** **static** ArrayList<JButton> getButtonsFromJPanel(JPanel jp)
441. {
442. ArrayList<JButton> ret = **new** ArrayList<>();
444. **for** (Component c : jp.getComponents())
445. {
446. **if** (c **instanceof** JButton)
447. {
448. ret.add((JButton) c);
449. }
450. }
451. **return** ret;
452. }
454. **public** **static** ArrayList<Component> getComponentsFromJPanel(JPanel jp)
455. {
456. ArrayList<Component> ret = **new** ArrayList<>();
458. **for** (Component c : jp.getComponents())
459. {
460. **if** (c **instanceof** Component)
461. {
462. ret.add((Component) c);
463. }
464. }
465. **return** ret;
466. }


470. /\*\*\*
471. \* This grabs the comparator ('>', '=', '!='), and the two operands (name, id,
472. \* date), ("henry", 1, 11/27/2017) and puts it into a query. It then uses the
473. \* results of that query to format the table.
474. \*
475. \* @param s
476. \*/
477. **public** **static** **void** setupResultsView(String s)
478. {
480. }
482. /\*\*\*
483. \* Sets tooltips for all components inside of a container.
484. \* @param container             The container that has the components inside of it.
485. \* @param excludedComponents    Names of components to not get tooltips.
486. \* @param tooltip               The tooltip to be displayed.
487. \*/
488. **public** **static** **void** setTooltips(JPanel container, String[] excludedComponents, String tooltip)
489. {
490. ArrayList<String> excludedComponentsAR = **new** ArrayList<>(Arrays.asList(excludedComponents));
492. ArrayList<Component> components = Util.getComponentsFromJPanel(container);
493. SpringLayout tempSL = (SpringLayout) container.getLayout();
495. **for** (**int** i = 0; i < components.size(); i++)
496. {
497. **if** (!excludedComponentsAR.contains(components.get(i).getName()))
498. {
499. **try**
500. {
501. ((JComponent) components.get(i)).setToolTipText(tooltip); // may not be able to be cast to JComponent...
502. }
503. **catch** (Exception e)
504. {
505. e.printStackTrace();
506. }
507. }
508. }
509. }


513. /\*\*
514. \* @param buttonContainer
515. \*            The container that has the buttons inside of it.
516. \*
517. \*            With a bunch of buttons, they will be laid out using springLayout
518. \*            offset by 10px from eachother.
519. \*/
520. **public** **static** **void** formatButtons(JPanel buttonContainer)
521. {
522. // insert formatting for all buttons.
523. ArrayList<JButton> menuButtons = Util.getButtonsFromJPanel(buttonContainer);
524. SpringLayout tempSL = (SpringLayout) buttonContainer.getLayout();
526. **for** (**int** i = 0; i < menuButtons.size(); i++)
527. {
528. JButton tempButton = menuButtons.get(i);
529. Util.printf("menuButtons[%d] = %s\n", i, tempButton);
531. tempSL.putConstraint(Util.WEST, tempButton, 10, Util.WEST, buttonContainer);
532. tempSL.putConstraint(Util.EAST, tempButton, -10, Util.EAST, buttonContainer);
534. **if** (i == 0) // first button aligns vertically w/ container
535. {
536. tempSL.putConstraint(Util.NORTH, tempButton, 10, Util.NORTH, buttonContainer);
537. }
538. **else** // all others align with south side of prev. button
539. {
540. JButton prevButton = menuButtons.get(i - 1);
541. tempSL.putConstraint(Util.NORTH, tempButton, 10, Util.SOUTH, prevButton);
542. }
543. }
544. }
546. /\*\*
547. \* Return the int[] [row,col] of a JTable object.
548. \* @param t The JTable object.
549. \* @return The selected row and column.
550. \*/
551. **public** **static** Integer[] getSelectedPos(JTable t)
552. {
553. Integer[] ret = {t.getSelectedRow(), t.getSelectedColumn()};
555. **return** ret;
556. }

559. **public** **static** String sanitize(String s)
560. {
561. **return** s.replaceAll("['\"\\\\]", "\\\\$0");
562. }


566. }

# [GUIC.java](#Source_code_directory)

1. /\*\*\*
2. \* GUI Constants for laziness and uniformity.
3. \*/
4. **public** **enum** GUIC
5. {
6. MUST\_ENTER\_DATE("You must enter a date here."),
7. NOT\_LOGGED\_IN("Not currently logged in."),
8. CURRENTLY\_LOGGED\_IN\_TEXT("Currently logged in as:"),
9. ;


13. **private** String constants;
15. **private** GUIC(String s)
16. {
17. **this**.constants = s;
18. }
20. @Override
21. **public** String toString()
22. {
23. **return** constants;
24. }
26. **public** String s()
27. {
28. **return** toString();
29. }
31. **public** String ts()
32. {
33. **return** toString();
34. }
35. }

# [SQLC.java](#Source_code_directory)

/\*\*\*

\* SQL Constants for laziness and uniformity.

\*/

**public** **enum** SQLC

{

***DEFAULT\_LOGIN\_DATA*** ("jdbc:mysql://localhost:3306/testdb,root,password\r\n"

+ "jdbc:mysql://goggle.ru/bestdb:6666,toor,root\r\n"),

***TABLE\_PREFIX*** ("h1\_j1\_"),

***SQL\_STR\_TO\_DATE\_FORMAT*** ("%Y-%m-%d %H:%i:%s"),

***SQL\_DATE\_FORMAT*** ("yyyy-MM-dd hh:mm:ss"),

***TABLE\_USERS*** (***TABLE\_PREFIX***+"users"),

***TABLE\_TICKETS*** (***TABLE\_PREFIX***+"tickets"),

***TABLE\_USER\_TYPES*** (***TABLE\_PREFIX***+"user\_types"),

***TABLE\_CATEGORIES*** (***TABLE\_PREFIX***+"categories"),

***TABLE\_SEVERITIES*** (***TABLE\_PREFIX***+"severities"),

***PASSWORD\_COLUMN\_NAME*** ("password"),

***USERTYPE\_COLUMN\_NAME*** ("user\_type"),

***USER\_ID\_COLUMN\_NAME*** ("PID"),

***TICKET\_ID\_COLUMN\_NAME*** ("TID"),

***SHORT\_DESC\_COLUMN\_NAME*** ("shortDesc"),

***LONG\_DESC\_COLUMN\_NAME*** ("longDesc"),

***CATEGORY\_COLUMN\_NAME*** ("category"),

***CATEGORY\_ID\_COLUMN\_NAME*** ("CID"),

***SEVERITY\_COLUMN\_NAME*** ("severity"),

***SEVERITY\_ID\_COLUMN\_NAME*** ("SID"),

***START\_DATE\_COLUMN\_NAME*** ("startDate"),

***END\_DATE\_COLUMN\_NAME*** ("endDate"),

***SEARCH\_PANE\_DEFAULT\_QUERY1*** ("SELECT \* FROM " + SQLC.***TABLE\_TICKETS***),

***SEARCH\_PANE\_DEFAULT\_QUERY2*** ("SELECT\n\u0009TID 'Ticket ID',\n\u0009PID 'Person ID',\n\u0009name 'Name',\n\u0009RES.CID 'Category ID',\n\u0009category 'Category',\n\u0009RES.SID 'Severity ID',\n severity\_level 'Severity Level',\n\u0009shortDesc 'Short Description',\n\u0009longDesc 'Long Description',\n\u0009startDate 'Start date',\n\u0009endDate 'End date'\nFROM\n\u0009(\n SELECT\n\u0009\u0009TID,\n\u0009\u0009PID,\n\u0009\u0009name,\n\u0009\u0009RES.CID,\n\u0009\u0009category,\n\u0009\u0009SID,\n\u0009\u0009shortDesc,\n\u0009\u0009longDesc,\n\u0009\u0009startDate,\n\u0009\u0009endDate\n\u0009FROM\n\u0009\u0009(\n\u0009\u0009SELECT\n\u0009\u0009\u0009TID,\n\u0009\u0009\u0009t.PID,\n\u0009\u0009\u0009name,\n\u0009\u0009\u0009CID,\n\u0009\u0009\u0009SID,\n\u0009\u0009\u0009shortDesc,\n\u0009\u0009\u0009longDesc, \n\u0009\u0009\u0009startDate, \n\u0009\u0009\u0009endDate \n\u0009\u0009FROM\n\u0009\u0009\u0009tickets t\n\u0009\u0009INNER JOIN\n\u0009\u0009\u0009(\n SELECT\n\u0009\u0009\u0009\u0009name,\n PID -- get users\n\u0009\u0009\u0009FROM \n\u0009\u0009\u0009\u0009users\n\u0009\u0009\u0009) u\n\u0009\u0009ON\n\u0009\u0009\u0009t.PID = u.PID -- put ticket ids on users\n\u0009\u0009) AS RES\n\u0009INNER JOIN\n\u0009\u0009(SELECT CID, category FROM categories) c \n\u0009ON\n\u0009\u0009c.CID = RES.CID -- put category name on CID\n\u0009) AS RES\nINNER JOIN \n\u0009(SELECT \* FROM severities) s \nON\n\u0009s.SID = RES.SID -- put severity name on SID"), //generated using http://snible.org/java2/uni2java.html

***SEARCH\_PANE\_DEFAULT\_QUERY\_PAPADEMAS*** ("SELECT\n\u0009TID 'Ticket ID',\n\u0009PID 'Person ID',\n\u0009name 'Name',\n\u0009RES.CID 'Category ID',\n\u0009category 'Category',\n\u0009RES.SID 'Severity ID',\n severity\_level 'Severity Level',\n\u0009shortDesc 'Short Description',\n\u0009longDesc 'Long Description',\n\u0009startDate 'Start date',\n\u0009endDate 'End date'\nFROM\n\u0009(\n SELECT\n\u0009\u0009TID,\n\u0009\u0009PID,\n\u0009\u0009name,\n\u0009\u0009RES.CID,\n\u0009\u0009category,\n\u0009\u0009SID,\n\u0009\u0009shortDesc,\n\u0009\u0009longDesc,\n\u0009\u0009startDate,\n\u0009\u0009endDate\n\u0009FROM\n\u0009\u0009(\n\u0009\u0009SELECT\n\u0009\u0009\u0009TID,\n\u0009\u0009\u0009t.PID,\n\u0009\u0009\u0009name,\n\u0009\u0009\u0009CID,\n\u0009\u0009\u0009SID,\n\u0009\u0009\u0009shortDesc,\n\u0009\u0009\u0009longDesc, \n\u0009\u0009\u0009startDate, \n\u0009\u0009\u0009endDate \n\u0009\u0009FROM\n\u0009\u0009\u0009h1\_j1\_tickets t\n\u0009\u0009INNER JOIN\n\u0009\u0009\u0009(\n SELECT\n\u0009\u0009\u0009\u0009name,\n PID -- get users\n\u0009\u0009\u0009FROM \n\u0009\u0009\u0009\u0009h1\_j1\_users\n\u0009\u0009\u0009) u\n\u0009\u0009ON\n\u0009\u0009\u0009t.PID = u.PID -- put ticket ids on users\n\u0009\u0009) AS RES\n\u0009INNER JOIN\n\u0009\u0009(SELECT CID, category FROM h1\_j1\_categories) c \n\u0009ON\n\u0009\u0009c.CID = RES.CID -- put category name on CID\n\u0009) AS RES\nINNER JOIN \n\u0009(SELECT \* FROM h1\_j1\_severities) s \nON\n\u0009s.SID = RES.SID -- put severity name on SID"),

;

**private** String constants;

**private** SQLC(String s)

{

**this**.constants = s;

}

@Override

**public** String toString()

{

**return** constants;

}

**public** String s()

{

**return** toString();

}

**public** String ts()

{

**return** toString();

}

}