Purpose of the Project:

To create a trouble ticket system that allows a user to input tickets and admins to manage those tickets. It also utilizes two database tables one that stores the user credentials and another that stores all of the trouble ticket information. Within this project, there are four programs each that has their own role.

* Login.java: Creates a login window for the either a user or admin to enter their credentials. From there, the program will authenticate and verify the credentials to make sure that there are valid.

ADMIN LOGIN: username: admin password: admin1

USER LOGIN: username: Cheryl Gardner password: Wedemhawks1

* Tickets.java: Creates the interactive ticket window where the user and the admins can use their menus. When menu items are selected, it allows for the user to interact including inputting their name, ticket description, status and ticket id.
* Dao.java: Writes the SQL statements to create the database connection to the SQL server, create the two tables, the user and the. Ticket table and write the SQL statements for the various CRUD operations (Create, Read, Update, Delete) that allows one depending on their credentials to insert, read, delete and update tickets.
* ticketsJTable: Incorporates the JTable structure and data into the created JTables within the ticket system.

Within the login windows, the level of the user decides on what functions they have available to them. For users, they have the most basic functions including exiting the system, adding their number to be helped, opening a ticket and selecting their specific ticket to view. For admins, they have a wider variety including exiting the system, refreshing the ticket system, deleting a ticket, inserting a ticket, updating a ticket and viewing all of the tickets. In addition, I have adjusted so that the status represents the closing date that when it is closed, there is a closing date and when it is open there is none.

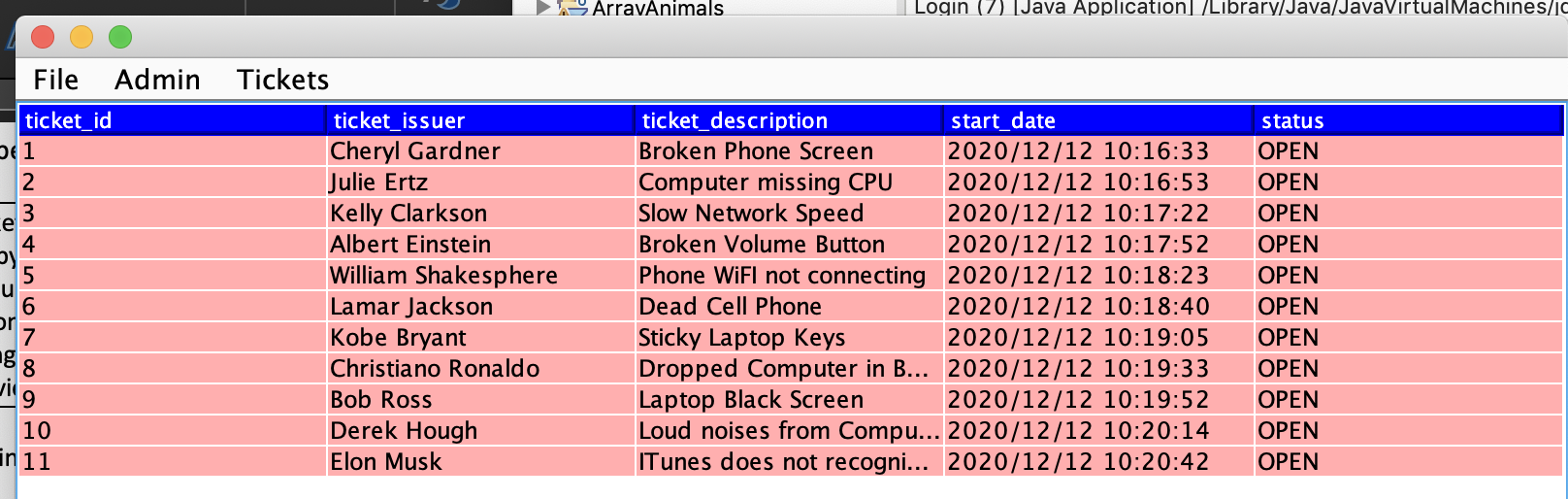
The six things that needed to be tested in the program include:

1. Insert at least 10 tickets into the DB table. *Include a record with your name* into the table.
2. Update your record by changing your ticket description.
3. Show a view of your updated ticket.
4. Delete your ticket from the DB.
5. Close any two existing tickets.
6. Lastly, show a table view of all of your tickets.

**The results of the testing shown in the console:**

1. **Insertion of 10 tables into the tickets DB table, including a record with my name in there**

Text

Description automatically generated

1. **Update the description of my trouble ticket**

Graphical user interface, text, application

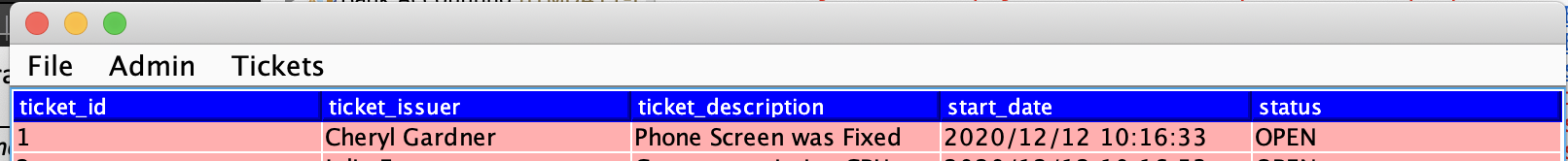
Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated A picture containing text, indoor

Description automatically generated

1. **Showing the view of the updated ticket**



1. **Deletion of my ticket from the DB tickets table**

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedText

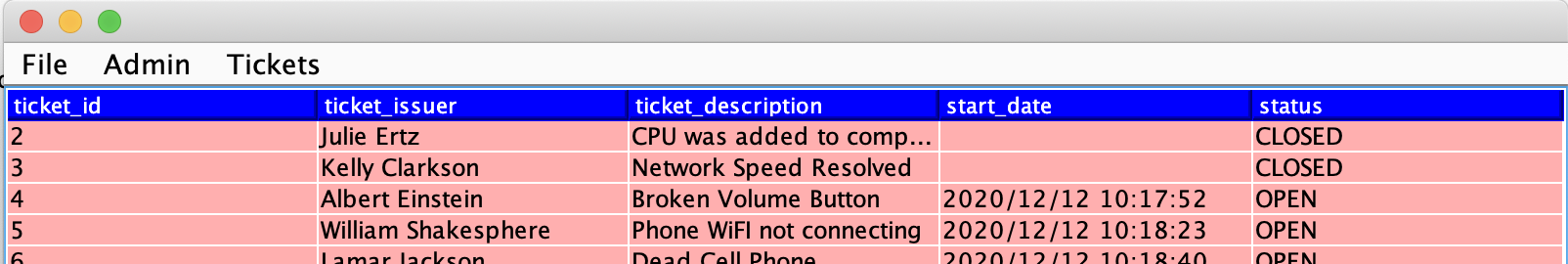
Description automatically generatedA picture containing graphical user interface

Description automatically generated

1. **Close two existing tickets within the DB tickets table**

Text

Description automatically generated



1. **Show a table view of all of the tickets that are left after all of the task**

Text

Description automatically generated

Graphical user interface

Description automatically generated

**Admin Menu Options (Exit, Refresh, Update Ticket, Delete Ticket, Open Ticket, and View Ticket):**

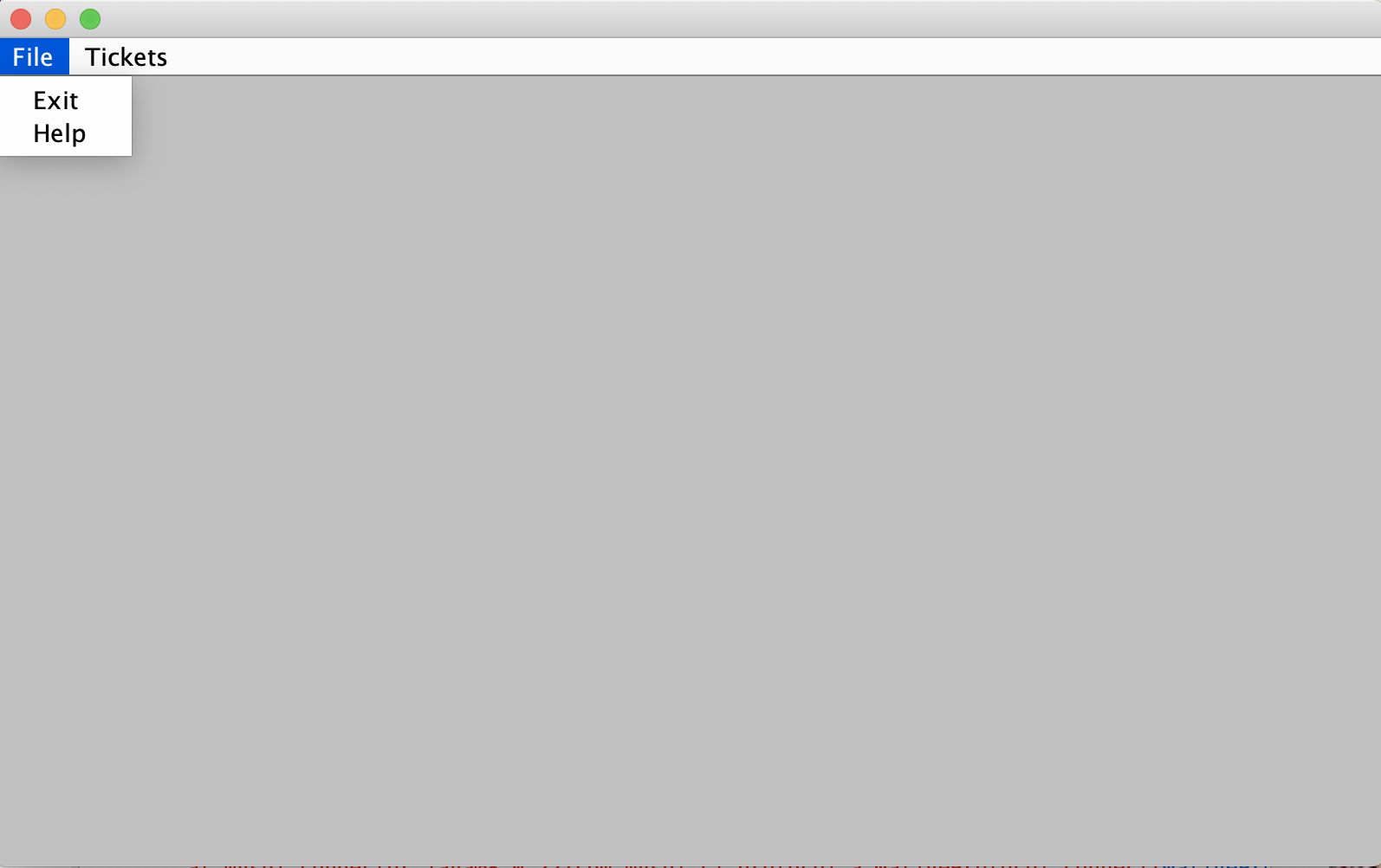
**A picture containing graphical user interface

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generated**

**User Menu Options (Exit, Help, Open Ticket, View Your Ticket):**

**Graphical user interface, application, Word

Description automatically generated**

**Extra Credit:**

**3. GitHub repo with links to all of the labs and the final project:**

Link to Lab 1 GitHub Repository: <https://github.com/cgardner3/ITMD411-Lab1>

Link to Lab 2 GitHub Repository: <https://github.com/cgardner3/ITMD411-Lab2>

Link to Lab 3 GitHub Repository: <https://github.com/cgardner3/ITMD411-Lab3>

Link to Lab 4 GitHub Repository: <https://github.com/cgardner3/ITMD411-Lab4>

Link to Final Project GitHub Repository: <https://github.com/cgardner3/ITMD411-FinalProject>

**12. Prepared statements for selects, inserts, updates, deletes**

**Prepared Statements to Insert/Create Records:**

//Declare a new method to insert the Records into the table

**public** **int** insertRecords(String ticketName, String ticketDesc) {

//Declare a new variable called id

**int** id = 0;

//Create a new Prepared statement to add all of the records into the ticket system

**try** {

PreparedStatement insert = *connect*.prepareStatement("INSERT into cgard\_tickets1 (ticket\_issuer, ticket\_description, start\_date, status) VALUES (?,?,?,?)", Statement.***RETURN\_GENERATED\_KEYS***);

String startTime = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(Calendar.*getInstance*().getTime());

insert.setString(1, ticketName);

insert.setString(2, ticketDesc);

insert.setString(3, startTime);

insert.setString(4, "OPEN");

insert.executeUpdate();

//Retrieve the ticket id from the insertion statement

ResultSet resultSet = **null**;

resultSet = insert.getGeneratedKeys();

**if** (resultSet.next()) {

//Retrieve the first field in the table

id = resultSet.getInt(1);

}

}

//Throw an exception statement if it does not work

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the id variable to the insertion

**return** id;

}

**Prepared statements to Select/View Records:**

//Create the readRecords Method to view the records

**public** ResultSet readRecords() {

//Declare the resultSet as empty

ResultSet results = **null**;

//Create a try block with the Prepared Statement to select all of the data from the ticket table

**try** {

PreparedStatement read = *connect*.prepareStatement("SELECT \* FROM cgard\_tickets1");

results = read.executeQuery();

//read.close();

}

//If the statement does not work then throw an SQL Exception

**catch** (SQLException e1) {

e1.printStackTrace();

}

//Return the result set back to the method

**return** results;

}

//Create a new method called SelectRecords for a user to be able to see their specific ticket

**public** ResultSet selectRecords(**int** tid) {

//Declare resultSet as empty

ResultSet results = **null**;

//Create a new try block to select all of the fields of a specific row using the user's ticket ID

**try** {

PreparedStatement select = *connect*.prepareStatement("SELECT \* FROM cgard\_tickets1 WHERE ticket\_id = ?");

select.setLong(1, tid);

results = select.executeQuery();

//select.close();

}

//If it does not work, throw the error back to the user

**catch** (SQLException e1) {

e1.printStackTrace();

}

//Return the result set back to the method

**return** results;

}

**Prepared statements to Update Records:**

//Declare a new method called updateRecords

**public** **void** updateRecords(**int** tid) {

//Create a new try block to update the Records using SQL Prepared Statements

**try** {

//Declare two variables called newDesc and newStatus to hold the status and description for the update

String newDesc = **null**;

String newStatus = **null**;

//Ask the user if they want to update the description and the status

**int** answer1 = JOptionPane.*showConfirmDialog*(**null**, "Do you want to update the ticket description of Ticket ID: " + tid + " ?", "Confirm", JOptionPane.***YES\_NO\_OPTION***);

**int** answer2 = JOptionPane.*showConfirmDialog*(**null**, "Do you want to close Ticket ID: " + tid + " ?", "Confirm", JOptionPane.***YES\_NO\_OPTION***);

//If the user says they want to update both fields to this

**if** (answer1 == JOptionPane.***YES\_OPTION*** && answer2 == JOptionPane.***YES\_OPTION***) {

//Ask the user what they want the new description and set the status to be closed

newDesc = JOptionPane.*showInputDialog*(**null**, " What would you like the new description of Ticket ID: " + tid + " to be?");

newStatus = "CLOSED";

//Create a new prepared statement that updates the description, clears start time and closes ticket using a specific ticket id

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET ticket\_description = ?, start\_date = ?, status = ? WHERE ticket\_id = ?");

update.setString(1,newDesc);

update.setString(2, "");

update.setString(3, newStatus);

update.setLong(4, tid);

update.executeUpdate();

update.close();

//Tell the user the ticket was updated and closed successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

System.***out***.println("Ticket id: " + tid + " was successfully closed");

JOptionPane.*showMessageDialog*(**null**, "Ticket id: " + tid + " was closed successfully");

}

//If the user only wants to update the description do this

**else** **if** (answer1 == JOptionPane.***YES\_OPTION*** && answer2 == JOptionPane.***NO\_OPTION***) {

//Ask the user what the new description is

newDesc = JOptionPane.*showInputDialog*(**null**, " What would you like the new description to be?");

//Create a new prepared statement that updates the description using a certain ticket id

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET ticket\_description = ? WHERE ticket\_id = ?");

update.setString(1, newDesc);

update.setLong(2, tid);

update.executeUpdate();

update.close();

//Tell the user that the ticket was updated successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

}

//If the user only wants to update the status do this

**else** **if** (answer1 == JOptionPane.***NO\_OPTION*** && answer2 == JOptionPane.***YES\_OPTION***) {

//Set the new status to be closed

newStatus = "CLOSED";

//Create a new prepared statement that clears the start date and updates the status to be closed

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET start\_date = ?, status = ? WHERE ticket\_id = ?");

update.setString(1, "");

update.setString(2,newStatus);

update.setLong(3, tid);

update.executeUpdate();

update.close();

//Tell the user that the ticket was closed and updated successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

System.***out***.println("Ticket id: " + tid + " was successfully closed");

JOptionPane.*showMessageDialog*(**null**, "Ticket id: " + tid + " was closed successfully");

}

//If the user does not want to update either do this

**else** **if** (answer1 == JOptionPane.***NO\_OPTION*** && answer2 == JOptionPane.***NO\_OPTION***) {

//Tell the user that the update fails

JOptionPane.*showMessageDialog*(**null**, "The following ticket could not be updated");

System.***out***.println("Update failed");

}

}

//If the try block does not work, throw an exception

**catch** (SQLException e) {

e.printStackTrace();

}

}

**Prepared statements to Delete Records:**

//Create a new method called deleteRecords to delete a record using a specific ticket id

**public** **int** deleteRecords(**int** tid) {

//Create a new try block

**try** {

//Create a new Prepared statement that deletes the record using the given ticket id

PreparedStatement delete = *connect*.prepareStatement("DELETE FROM cgard\_tickets1 WHERE ticket\_id = ?");

delete.setLong(1, tid);

delete.executeUpdate();

delete.close();

}

//If it does not work then throw the exception

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the ticket id to the method

**return** tid;

}

**Source Code:**

**Dao.java:**

/\*

\* Name: Cheryl Gardner

\* Date: 12/10/2020

\* Program: Dao.java

\* ITMD 411 Final Project

\* Purpose: Define all of the necessary programs including creating the necessary tables, making a connection

\* adding users, and inserting, selecting, view, delete, and update records using a ticket id

\*/

//Import javaapplication1 package

**package** javaapplication1;

//Import all of the necessary packages for the program

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Calendar;

**import** java.util.List;

**import** javax.swing.JOptionPane;

//Create Dao class for connections and record purposes

**public** **class** Dao {

// Declare the necessary instances fields

**static** Connection *connect* = **null**;

Statement statement = **null**;

// Create the Dao Constructor

**public** Dao() {

}

//Define a new method called connection that will create the SQL connection

**public** Connection getConnection() {

// Setup the connection with the DB

**try** {

//Try to connect to the JDBC sql papademas using the user fp411 and password 411

*connect* = DriverManager

.*getConnection*("jdbc:mysql://www.papademas.net:3307/tickets?autoReconnect=true&useSSL=false"

+ "&user=fp411&password=411");

//Tell the user that the connection was made

System.***out***.println("Connection was made");

}

//If the connection cannot be made, then tell the user that the connection failed

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the connection to the Dao program

**return** *connect*;

}

//Declare the Creation of Tables method to create the necessary tables

**public** **void** createTables() {

// Create the tickets table to hold the ticket records and the users table to hold all of the valid users for login

**final** String createTicketsTable = "CREATE TABLE cgard\_tickets1(ticket\_id INT AUTO\_INCREMENT PRIMARY KEY, ticket\_issuer VARCHAR(30), ticket\_description VARCHAR(200), start\_date VARCHAR(50), status VARCHAR(15))";

**final** String createUsersTable = "CREATE TABLE cgard\_users1(uid INT AUTO\_INCREMENT PRIMARY KEY, uname VARCHAR(30), upass VARCHAR(30), admin int)";

//Try this block to see if it works

**try** {

//Create a new statement and update to add the necessary tables for creation

statement = getConnection().createStatement();

statement.executeUpdate(createTicketsTable);

statement.executeUpdate(createUsersTable);

//Tell the user that the tables were created successfully

System.***out***.println("Tables were successfully created in the database");

//Close the statement and the connection

statement.close();

*connect*.close();

}

//Throw an error statement if it does not work

**catch** (Exception e) {

System.***out***.println(e.getMessage());

}

//Call the add users method

addUsers();

}

//Create a new method called add users

**public** **void** addUsers() {

//Declare all of the variables that are needed for this method

String sql;

Statement statement;

BufferedReader br;

List<List<String>> array = **new** ArrayList<>(); // list to hold (rows & cols)

//Try to read the user.csv file

**try** {

br = **new** BufferedReader(**new** FileReader(**new** File("./userlist.csv")));

//Declare a new String variable called line and split all of the tables for the username, password and admin id

String line;

**while** ((line = br.readLine()) != **null**) {

array.add(Arrays.*asList*(line.split(",")));

}

}

//If the try block does not work, tell the user the file cannot be read

**catch** (Exception e) {

System.***out***.println("There was a problem loading the file");

}

//Create a new try block to insert all of the data from user.csv into the user table

**try** {

// Create a new statement based on the connection

statement = getConnection().createStatement();

//As long as the rowData is in the array, keep using the for loop

**for** (List<String> rowData : array) {

//Create a new sql statement to add the username, password and admin for the three values

sql = "insert into cgard\_users1(uname,upass,admin) " + "values('" + rowData.get(0) + "'," + " '"

+ rowData.get(1) + "','" + rowData.get(2) + "');";

//Execute the sql statement to update

statement.executeUpdate(sql);

}

//Tell the user that the insert was successfully

System.***out***.println("The users were successfully inserted into the database");

//Close the statement object

statement.close();

}

//Throw an exception if the sql statement does not work

**catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

//Declare a new method to insert the Records into the table

**public** **int** insertRecords(String ticketName, String ticketDesc) {

//Declare a new variable called id

**int** id = 0;

//Create a new Prepared statement to add all of the records into the ticket system

**try** {

PreparedStatement insert = *connect*.prepareStatement("INSERT into cgard\_tickets1 (ticket\_issuer, ticket\_description, start\_date, status) VALUES (?,?,?,?)", Statement.***RETURN\_GENERATED\_KEYS***);

String startTime = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(Calendar.*getInstance*().getTime());

insert.setString(1, ticketName);

insert.setString(2, ticketDesc);

insert.setString(3, startTime);

insert.setString(4, "OPEN");

insert.executeUpdate();

//Retrieve the ticket id from the insertion statement

ResultSet resultSet = **null**;

resultSet = insert.getGeneratedKeys();

**if** (resultSet.next()) {

//Retrieve the first field in the table

id = resultSet.getInt(1);

}

}

//Throw an exception statement if it does not work

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the id variable to the insertion

**return** id;

}

**public** ResultSet refreshTable() {

ResultSet results = **null**;

**try** {

PreparedStatement refresh = *connect*.prepareStatement("SELECT \* FROM cgard\_tickets1");

results = refresh.executeQuery();

}

**catch** (SQLException e1) {

e1.printStackTrace();

}

**return** results;

}

//Create the readRecords Method to view the records

**public** ResultSet readRecords() {

//Declare the resultSet as empty

ResultSet results = **null**;

//Create a try block with the Prepared Statement to select all of the data from the ticket table

**try** {

PreparedStatement read = *connect*.prepareStatement("SELECT \* FROM cgard\_tickets1");

results = read.executeQuery();

//read.close();

}

//If the statement does not work then throw an SQL Exception

**catch** (SQLException e1) {

e1.printStackTrace();

}

//Return the result set back to the method

**return** results;

}

//Create a new method called SelectRecords for a user to be able to see their specific ticket

**public** ResultSet selectRecords(**int** tid) {

//Declare resultSet as empty

ResultSet results = **null**;

//Create a new try block to select all of the fields of a specific row using the user's ticket ID

**try** {

PreparedStatement select = *connect*.prepareStatement("SELECT \* FROM cgard\_tickets1 WHERE ticket\_id = ?");

select.setLong(1, tid);

results = select.executeQuery();

//select.close();

}

//If it does not work, throw the error back to the user

**catch** (SQLException e1) {

e1.printStackTrace();

}

//Return the result set back to the method

**return** results;

}

//Declare a new method called updateRecords

**public** **void** updateRecords(**int** tid) {

//Create a new try block to update the Records using SQL Prepared Statements

**try** {

//Declare two variables called newDesc and newStatus to hold the status and description for the update

String newDesc = **null**;

String newStatus = **null**;

//Ask the user if they want to update the description and the status

**int** answer1 = JOptionPane.*showConfirmDialog*(**null**, "Do you want to update the ticket description of Ticket ID: " + tid + " ?", "Confirm", JOptionPane.***YES\_NO\_OPTION***);

**int** answer2 = JOptionPane.*showConfirmDialog*(**null**, "Do you want to close Ticket ID: " + tid + " ?", "Confirm", JOptionPane.***YES\_NO\_OPTION***);

//If the user says they want to update both fields to this

**if** (answer1 == JOptionPane.***YES\_OPTION*** && answer2 == JOptionPane.***YES\_OPTION***) {

//Ask the user what they want the new description and set the status to be closed

newDesc = JOptionPane.*showInputDialog*(**null**, " What would you like the new description of Ticket ID: " + tid + " to be?");

newStatus = "CLOSED";

//Create a new prepared statement that updates the description, clears start time and closes ticket using a specific ticket id

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET ticket\_description = ?, start\_date = ?, status = ? WHERE ticket\_id = ?");

update.setString(1,newDesc);

update.setString(2, "");

update.setString(3, newStatus);

update.setLong(4, tid);

update.executeUpdate();

update.close();

//Tell the user the ticket was updated and closed successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

System.***out***.println("Ticket id: " + tid + " was successfully closed");

JOptionPane.*showMessageDialog*(**null**, "Ticket id: " + tid + " was closed successfully");

}

//If the user only wants to update the description do this

**else** **if** (answer1 == JOptionPane.***YES\_OPTION*** && answer2 == JOptionPane.***NO\_OPTION***) {

//Ask the user what the new description is

newDesc = JOptionPane.*showInputDialog*(**null**, " What would you like the new description to be?");

//Create a new prepared statement that updates the description using a certain ticket id

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET ticket\_description = ? WHERE ticket\_id = ?");

update.setString(1, newDesc);

update.setLong(2, tid);

update.executeUpdate();

update.close();

//Tell the user that the ticket was updated successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

}

//If the user only wants to update the status do this

**else** **if** (answer1 == JOptionPane.***NO\_OPTION*** && answer2 == JOptionPane.***YES\_OPTION***) {

//Set the new status to be closed

newStatus = "CLOSED";

//Create a new prepared statement that clears the start date and updates the status to be closed

PreparedStatement update = *connect*.prepareStatement("UPDATE cgard\_tickets1 SET start\_date = ?, status = ? WHERE ticket\_id = ?");

update.setString(1, "");

update.setString(2,newStatus);

update.setLong(3, tid);

update.executeUpdate();

update.close();

//Tell the user that the ticket was closed and updated successfully

System.***out***.println("Ticket ID : " + tid + " was updated successfully");

JOptionPane.*showMessageDialog*(**null**, "Ticket ID " + tid + " was successfully updated");

System.***out***.println("Ticket id: " + tid + " was successfully closed");

JOptionPane.*showMessageDialog*(**null**, "Ticket id: " + tid + " was closed successfully");

}

//If the user does not want to update either do this

**else** **if** (answer1 == JOptionPane.***NO\_OPTION*** && answer2 == JOptionPane.***NO\_OPTION***) {

//Tell the user that the update fails

JOptionPane.*showMessageDialog*(**null**, "The following ticket could not be updated");

System.***out***.println("Update failed");

}

}

//If the try block does not work, throw an exception

**catch** (SQLException e) {

e.printStackTrace();

}

}

//Create a new method called deleteRecords to delete a record using a specific ticket id

**public** **int** deleteRecords(**int** tid) {

//Create a new try block

**try** {

//Create a new Prepared statement that deletes the record using the given ticket id

PreparedStatement delete = *connect*.prepareStatement("DELETE FROM cgard\_tickets1 WHERE ticket\_id = ?");

delete.setLong(1, tid);

delete.executeUpdate();

delete.close();

}

//If it does not work then throw the exception

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the ticket id to the method

**return** tid;

}

}

**Login.java:**

/\*

\* Name: Cheryl Gardner

\* Date: 12/10/2020

\* Program: Login.java

\* ITMD 411 Final Project

\* Purpose: Create a new Login menu that checks to make sure that the user is valid

\* and to create all of the necessary login buttons and usernames

\*/

//Import the necessary package

**package** javaapplication1;

//Import all of the necessary packages for the program

**import** java.awt.Color;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPasswordField;

**import** javax.swing.JTextField;

@SuppressWarnings("serial")

//Create a new class called Login that extends JFrame

**public** **class** Login **extends** JFrame {

//Declare a new Dao connection variable to be used later

Dao conn;

**public** Login() {

//Create the window header

**super**("IIT HELP DESK LOGIN");

//Call the Dao program and the create table method

conn = **new** Dao();

conn.createTables();

//Set the size, grid layout and location for the login window

setSize(800, 500);

setLayout(**new** GridLayout(5, 2));

setLocationRelativeTo(**null**);

//Color the overall login content pane

getContentPane().setBackground(Color.***CYAN***);

//Create a new Header that tells the users what this program us

JLabel header1 = **new** JLabel("Cheryl's Help ", JLabel.***RIGHT***);

JLabel header2 = **new** JLabel("Desk System", JLabel.***LEFT***);

//Color the text for the headers

header1.setForeground(Color.***BLACK***);

header2.setForeground(Color.***BLACK***);

//Create the login and passwords labels

JLabel lblUsername = **new** JLabel("Username", JLabel.***LEFT***);

JLabel lblPassword = **new** JLabel("Password", JLabel.***LEFT***);

//Color the text for the labels

lblUsername.setForeground(Color.***BLACK***);

lblPassword.setForeground(Color.***BLACK***);

//Create a status bar at the bottom and create a password and text field for users to input their credentials

JLabel lblStatus = **new** JLabel(" ", JLabel.***CENTER***);

// JLabel lblSpacer = new JLabel(" ", JLabel.CENTER);

JTextField txtUname = **new** JTextField(10);

JPasswordField txtPassword = **new** JPasswordField();

//Color the fields with the background to blend in and color the text that is inputed

txtUname.setBackground(Color.***CYAN***);

txtPassword.setBackground(Color.***CYAN***);

txtUname.setForeground(Color.***BLACK***);

txtPassword.setForeground(Color.***BLACK***);

//Add two buttons one to submit and one to exit

JButton btn = **new** JButton("Submit");

JButton btnExit = **new** JButton("Exit");

//Color the background of the buttons as well as the text

btn.setBackground(Color.***BLUE***);

btnExit.setBackground(Color.***RED***);

btn.setOpaque(**true**);

btn.setBorderPainted(**false**);

btnExit.setOpaque(**true**);

btnExit.setBorderPainted(**false**);

btn.setForeground(Color.***WHITE***);

btnExit.setForeground(Color.***WHITE***);

//Create the necessary constraints

lblStatus.setToolTipText("Contact help desk to unlock password");

lblUsername.setHorizontalAlignment(JLabel.***CENTER***);

lblPassword.setHorizontalAlignment(JLabel.***CENTER***);

//Add all of the Objects into the frame

add(header1); //1st row

add(header2);

add(lblUsername);//2nd row

add(txtUname);

add(lblPassword); //3rd row

add(txtPassword);

add(btn); //4th row

add(btnExit);

add(lblStatus); //5th row

//Add a new button listener when the submit button is pressed

btn.addActionListener(**new** ActionListener() {

//Declare the count to be 0

**int** count = 0; // count agent

@Override

//Add an addition actionEvent to check the user's credentials

**public** **void** actionPerformed(ActionEvent e) {

//Declare the user to be empty and add one to the count

String user = "";

count = count + 1;

//Create a prepared statement that checks the username and password of the user

String query = "SELECT \* FROM cgard\_users1 WHERE uname = ? and upass = ?;";

**try** (PreparedStatement stmt = conn.getConnection().prepareStatement(query)) {

//Set the two fields to be the inputted username and password

stmt.setString(1, txtUname.getText());

stmt.setString(2, txtPassword.~~getText~~());

//Execute the query

ResultSet rs = stmt.executeQuery();

//If there is a next in the result set

**if** (rs.next()) {

//Check the admin field to be if it is 1 or 0

**int** isAdmin = rs.getInt("admin");

//If it is an admin set the user to be admin and welcome the admin

**if** (isAdmin == 1) {

user = "admin";

JOptionPane.*showMessageDialog*(**null**, "Welcome Admin");

}

//If it is a regular user, welcome the user into the system

**else** **if** (isAdmin != 1) {

JOptionPane.*showMessageDialog*(**null**, "Welcome User");

}

//Pass the user into the tickets progra,

**new** Tickets(user);

//Hide the frame and close out of the window

setVisible(**false**);

dispose();

}

//If the login is not right, update the login saying how many attempts are left and update status

**else**

lblStatus.setText("Try again! " + (3 - count) + " / 3 attempts left");

}

//If the try block does not work, throw the error to the user

**catch** (SQLException ex) {

ex.printStackTrace();

}

}

});

//Add an action listener for the exit

btnExit.addActionListener(e -> System.*exit*(0));

setVisible(**true**);

}

//Call the main method

**public** **static** **void** main(String[] args) {

//Call the new Login program to run

**new** Login();

}

}

**Tickets.java:**

/\*

\* Name: Cheryl Gardner

\* Date: 12/10/2020

\* Program: Tickets.java

\* ITMD 411 Final Project

\* Purpose: Create the gui ticket menus for both admins and users as well as add listeners for each of the buttons

\* when they are pressed by a user or admin

\*/

//Import the javaapplication1 package for the program

**package** javaapplication1;

//Import all of the necessary packages that are needed for the program

**import** java.awt.Color;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

**import** java.sql.SQLException;

**import** javax.swing.JFrame;

**import** javax.swing.JMenu;

**import** javax.swing.JMenuBar;

**import** javax.swing.JMenuItem;

**import** javax.swing.JOptionPane;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTable;

**import** javax.swing.table.DefaultTableModel;

@SuppressWarnings("serial")

//Declare a new public class called tickets that extends JFrame and implements Action Listener

**public** **class** Tickets **extends** JFrame **implements** ActionListener {

//Declare and initialize the class variables

Dao dao = **new** Dao();

Boolean chkIfAdmin = **null**;

//Create the three menus, File, Admin and Tickets

**private** JMenu mnuFile = **new** JMenu("File");

**private** JMenu mnuAdmin = **new** JMenu("Admin");

**private** JMenu mnuTickets = **new** JMenu("Tickets");

//Create all of the menu items for the three menus

JMenuItem mnuItemExit;

JMenuItem mnuItemRefresh;

JMenuItem mnuItemUpdate;

JMenuItem mnuItemDelete;

JMenuItem mnuItemOpenTicket;

JMenuItem mnuItemViewTicket;

JMenuItem mnuItemSelectTicket;

JMenuItem mnuItemHelp;

JTable jt;

//Create a new method called tickets that checks whether the login user is an admin or user

**public** Tickets(String user) {

//If the user is an admin, make chkIfAdmin true and create and admin menu and gui

**if** (user.equals("admin")) {

chkIfAdmin = **true**;

AdminMenu();

AdminGUI();

}

//If the user is not an admin, create a user admin menu and gui

**else** {

UserMenu();

UserGUI();

}

}

//Create a new method called AdminMenu that creates the menu for an admin

**private** **void** AdminMenu() {

/\* Initialize sub menu items \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// initialize sub menu item for File main menu

mnuItemExit = **new** JMenuItem("Exit");

// add to File main menu item

mnuFile.add(mnuItemExit);

mnuItemRefresh = **new** JMenuItem("Refresh");

mnuFile.add(mnuItemRefresh);

// initialize first sub menu items for Admin main menu

mnuItemUpdate = **new** JMenuItem("Update Ticket");

// add to Admin main menu item

mnuAdmin.add(mnuItemUpdate);

// initialize second sub menu items for Admin main menu

mnuItemDelete = **new** JMenuItem("Delete Ticket");

// add to Admin main menu item

mnuAdmin.add(mnuItemDelete);

// initialize first sub menu item for Tickets main menu

mnuItemOpenTicket = **new** JMenuItem("Open Ticket");

// add to Ticket Main menu item

mnuTickets.add(mnuItemOpenTicket);

// initialize second sub menu item for Tickets main menu

mnuItemViewTicket = **new** JMenuItem("View Ticket");

// add to Ticket Main menu item

mnuTickets.add(mnuItemViewTicket);

/\* Add action listeners for each desired menu item \*\*\*\*\*\*\*\*\*\*\*\*\*/

mnuItemExit.addActionListener(**this**);

mnuItemRefresh.addActionListener(**this**);

mnuItemUpdate.addActionListener(**this**);

mnuItemDelete.addActionListener(**this**);

mnuItemOpenTicket.addActionListener(**this**);

mnuItemViewTicket.addActionListener(**this**);

}

//Create a new method called UserMenu to create the menu for the user

**private** **void** UserMenu() {

/\* Initialize sub menu items \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// initialize sub menu item for File main menu

mnuItemExit = **new** JMenuItem("Exit");

// add to File main menu item

mnuFile.add(mnuItemExit);

mnuItemHelp = **new** JMenuItem("Help");

mnuFile.add(mnuItemHelp);

// initialize first sub menu item for Tickets main menu

mnuItemOpenTicket = **new** JMenuItem("Open Ticket");

// add to Ticket Main menu item

mnuTickets.add(mnuItemOpenTicket);

mnuItemSelectTicket = **new** JMenuItem("View Your Ticket");

mnuTickets.add(mnuItemSelectTicket);

/\* Add action listeners for each desired menu item \*\*\*\*\*\*\*\*\*\*\*\*\*/

mnuItemExit.addActionListener(**this**);

mnuItemHelp.addActionListener(**this**);

mnuItemOpenTicket.addActionListener(**this**);

mnuItemSelectTicket.addActionListener(**this**);

}

//Create a new method called AdminGUI to create the GUI for the admin

**private** **void** AdminGUI() {

// create JMenu bar

JMenuBar bar = **new** JMenuBar();

bar.add(mnuFile); // add main menu items in order, to JMenuBar

bar.add(mnuAdmin);

bar.add(mnuTickets);

// add menu bar components to frame

setJMenuBar(bar);

addWindowListener(**new** WindowAdapter() {

// define a window close operation

**public** **void** windowClosing(WindowEvent wE) {

System.*exit*(0);

}

});

// set frame options

setSize(800, 500);

getContentPane().setBackground(Color.***LIGHT\_GRAY***);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

//Create a new method called UserGUI to create the GUI for the user

**private** **void** UserGUI() {

// create JMenu bar

JMenuBar bar = **new** JMenuBar();

bar.add(mnuFile); // add main menu items in order, to JMenuBar

bar.add(mnuTickets);

// add menu bar components to frame

setJMenuBar(bar);

addWindowListener(**new** WindowAdapter() {

// define a window close operation

**public** **void** windowClosing(WindowEvent wE) {

System.*exit*(0);

}

});

// set frame options

setSize(800, 500);

getContentPane().setBackground(Color.***LIGHT\_GRAY***);

setLocationRelativeTo(**null**);

setVisible(**true**);

}

@Override

//Create the action performed method for the actions when each menu is pressed

**public** **void** actionPerformed(ActionEvent e) {

//If the exit button is pressed do this

**if** (e.getSource() == mnuItemExit) {

//Exit the system

System.*exit*(0);

}

//If the Open Ticket Button is pressed do this

**else** **if** (e.getSource() == mnuItemOpenTicket) {

//Get the ticketName and the ticketDescription

String ticketName = JOptionPane.*showInputDialog*(**null**, "Enter your name");

String ticketDesc = JOptionPane.*showInputDialog*(**null**, "Enter a ticket description");

//Insert the ticket information into the records table

**int** id = dao.insertRecords(ticketName, ticketDesc);

//Tell the user whether the ticket creation that was a success or failure

**if** (id != 0) {

System.***out***.println("Ticket ID : " + id + " created successfully!!!");

JOptionPane.*showMessageDialog*(**null**, "Ticket id: " + id + " created");

}

**else** **if** (id == 0){

System.***out***.println("Ticket cannot be created!!!");

}

}

//If the help button is pressed do this

**else** **if** (e.getSource() == mnuItemHelp) {

//Ask the user to enter a contact phone number and check whether it is correct or not

String phoneNumber = JOptionPane.*showInputDialog*(**null**, "Enter a phone number that we can contact you?");

**int** confirm = JOptionPane.*showConfirmDialog*(**null**, "Is this phone number correct? " + phoneNumber);

//If the number is correct tell the user that they will be contacted

**if** (confirm == JOptionPane.***YES\_OPTION***) {

JOptionPane.*showMessageDialog*(**null**, "You will be contacted as soon as a representative becomes available");

}

//If the number is not correct have them enter a new number

**else** **if** (confirm == JOptionPane.***NO\_OPTION***) {

String newNumber = JOptionPane.*showInputDialog*(**null**, "Please enter the correct number");

phoneNumber = newNumber;

}

}

//If the user presses the View Ticket or Refresh button

**else** **if** (e.getSource() == mnuItemViewTicket) {

//Retrieve all of the data from the tables

**try** {

///Create a JTable using all of the ticket data from the records

JTable jt = **new** JTable(ticketsJTable.*buildTableModel*(dao.readRecords()));

jt.setBounds(30, 40, 200, 400);

//Color the background and the table header

jt.setBackground(Color.***PINK***);

jt.setForeground(Color.***BLACK***);

jt.getTableHeader().setBackground(Color.***BLUE***);

jt.getTableHeader().setForeground(Color.***WHITE***);

JScrollPane sp = **new** JScrollPane(jt);

add(sp);

setVisible(**true**); // refreshes or repaints frame on screen

//If it is View Ticket, tell the user that the ticket view was created

JOptionPane.*showMessageDialog*(**null**, "The Ticket View was created");

System.***out***.println("Ticket View was successfully created");

//If it does not work, throw an exception

} **catch** (SQLException e1) {

e1.printStackTrace();

}

}

**else** **if** (e.getSource() == mnuItemRefresh) {

**try** {

**new** Tickets("admin").mnuItemViewTicket.doClick();

setVisible(**false**);

dispose();

}

**catch** (Exception e1) {

e1.printStackTrace();

}

JOptionPane.*showMessageDialog*(**null**, "The Table was refreshed");

System.***out***.println("The table was successfully refreshed");

}

//If the Select Ticket button is pressed do this

**else** **if** (e.getSource() == mnuItemSelectTicket) {

//Ask the user what ticket they would like to select and parse it into an int

String tick\_Id = JOptionPane.*showInputDialog*(**null**, "Enter your selected Ticket ID number");

**int** tid = Integer.*parseInt*(tick\_Id);

//Create a try block

**try** {

//Create a JTable using all of the ticket data from the records

JTable jt = **new** JTable(ticketsJTable.*buildTableModel*(dao.selectRecords(tid)));

jt.setBounds(30, 40, 200, 400);

//Color the background and the table header

jt.setBackground(Color.***MAGENTA***);

jt.setForeground(Color.***WHITE***);

jt.getTableHeader().setBackground(Color.***WHITE***);

jt.getTableHeader().setForeground(Color.***BLACK***);

JScrollPane sp = **new** JScrollPane(jt);

add(sp);

//Refresh or repaint the frame on the screen

setVisible(**true**);

//Tell the user that the view of their selected ticket was a success

JOptionPane.*showMessageDialog*(**null**, "The Selected Ticket was successfully viewed");

System.***out***.println("Selected Ticket View was a success");

}

//If it does not work, throw an exception

**catch** (SQLException e1) {

e1.printStackTrace();

}

}

//If the update button is pressed, do this

**else** **if** (e.getSource() == mnuItemUpdate) {

//Enter the ticket number for the ticket to update and parse that int

String tick\_Id = JOptionPane.*showInputDialog*(**null**, "Enter the ticket id for the ticket you wish to update");

**int** tid = Integer.*parseInt*(tick\_Id);

//Update the record using the id

dao.updateRecords(tid);

//If it is 0, tell the user that the ticket could not be updated

**if** (tid == 0) {

JOptionPane.*showMessageDialog*(**null**, "The following ticket could not be updated");

}

}

//If the delete button is pressed, do this

**else** **if** (e.getSource() == mnuItemDelete) {

//Enter the number of the ticket id that you want to delete and parse that String into an int

String tick\_Id = JOptionPane.*showInputDialog*(**null**, "Enter the ticket id you want to delete");

**int** tid = Integer.*parseInt*(tick\_Id);

//Ask the user on whether or not they want to delete the ticket

**int** response = JOptionPane.*showConfirmDialog*(**null**, "Are you sure you want to delete ticket " + tid, "Warning once you delete it, it cannot be recovered", JOptionPane.***YES\_NO\_OPTION***);

//If they say Yes, delete the record

**if** (response == JOptionPane.***YES\_OPTION***) {

**int** id = dao.deleteRecords(tid);

//If the id is not 0 tell them that it was a success

**if** (id != 0) {

System.***out***.println("Ticket ID : " + id + " was successfully deleted");

JOptionPane.*showMessageDialog*(**null**, "Ticket id " + id + " was deleted");

}

//If not, tell the user it could not be updated

**else** {

JOptionPane.*showMessageDialog*(**null**, "The ticket could not be deleted");

}

}

//If they say no, tell the user it was not deleted

**else** **if** (response == JOptionPane.***NO\_OPTION***) {

JOptionPane.*showMessageDialog*(**null**, "The following ticket: " + tid + " was not deleted");

}

//If they cancel the message, tell them that the deletion was cancelled

**else** **if** (response == JOptionPane.***CANCEL\_OPTION***) {

JOptionPane.*showMessageDialog*(**null**, "The deletion request of ticket: " + tid + " was cancelled");

}

}

}

}

**ticketsJTable.java:**

/\*

\* Name: Cheryl Gardner

\* Date: 12/10/2020

\* Program: ticketsJTable.java

\* ITMD 411 Final Project

\* Purpose: To create the format for the table models of the JTables that is used to display all of the data of the tables

\*/

//Import the javaapplication1 package into the program

**package** javaapplication1;

//Import all of the necessary packages into the program

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.SQLException;

**import** java.util.Vector;

**import** javax.swing.table.DefaultTableModel;

//Create a new class called ticketsJTable that formats the JTables

**public** **class** ticketsJTable {

@SuppressWarnings("unused")

//Create a new final table model and build of that table model

**private** **final** DefaultTableModel tableModel = **new** DefaultTableModel();

//Decare a new method called DefaultTableModel that displays the results to the user and throws SQL Exceptions

**public** **static** DefaultTableModel buildTableModel(ResultSet rs) **throws** SQLException {

//Get all of the meta data from the tables.

ResultSetMetaData metaData = rs.getMetaData();

// names of columns

Vector<String> columnNames = **new** Vector<String>();

**int** columnCount = metaData.getColumnCount();

**for** (**int** column = 1; column <= columnCount; column++) {

columnNames.add(metaData.getColumnName(column));

}

// data of the table

Vector<Vector<Object>> data = **new** Vector<Vector<Object>>();

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

Vector<Object> vector = **new** Vector<Object>();

**for** (**int** columnIndex = 1; columnIndex <= columnCount; columnIndex++) {

vector.add(rs.getObject(columnIndex));

}

data.add(vector);

}

// return data/col.names for JTable

**return** **new** DefaultTableModel(data, columnNames);

}

}