Even Semester (2022)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information:

Surname: Clarin **Given Name:** Maria **Student ID Number:** 2501990331

Course Code : COMP6047001 **Course Name :** Algorithm and Programming

Class : L2BC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignments: Term Final Project

Submission Pattern

Due Date : 10 June 2022 **Submission Date :** 9 June 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Maria Clarin

A. Project Specification

1. Program Name and Logo

- Quizzen. Derived from the word ‘quiz’ and ‘zen’.
- The image attached below is the logo made for the app. The koala operating a laptop suits the name Quizzen. The koala looks peaceful and calm which refers to the meaning of ‘zen’ (also reminds me of nature which then reminds me of zen gardens) while the concept of the quiz app was developed on my own laptop and was made to operate on computers/laptops.

2. Program Description

- Quizzen is a student-teacher desktop quiz application made with Java libraries such as JFrame, Java SQL, and other Java utility libraries. It also utilizes the MySQL JDBC database to store student data and the teacher’s question inputs.
- As the education system has fully adapted to the online learning environment, students and teachers often use apps to make/take quizzes/tests. However, some places

3. Program Flow Summary

- Teachers/admins have to have a registered account login info (declared in the program manually) in order to be able to log into the teacher/admin database. Once logged in, Teachers can add questions to the quiz, update questions in the quiz, view all questions made for the quiz, delete specific questions or all the questions in the quiz, view all students’ quiz results, manage the student database (editing student data, deleting specific student data, or deleting all student data), log out or exit the program.
- Students who are going to take the quiz will be asked to input their data (Name, Class, Gender, Student ID No., Contact No., Email, and Address) in order to be able to proceed with the quiz. They will then be led to the instructions page, where they can choose to start the quiz. Once the quiz starts, a timer will run and keep track of the student’s time working on the quiz. I set the maximum time as 10 minutes (manually programmed in the code) and so the timer will stop when it reaches 10 minutes and automatically submits the result of the student. If the student gets the answer right to a question, their score/marks will increment per correct answer. Students will not receive any score/mark on the question that they answered wrong. They are able to submit before they answer all the questions, however, there will be a pop-up confirm

dialog for them to confirm before submitting. By the end of the quiz, they can then see their total score. If they have any issues regarding their data such as wrong input, etc., they would have to request some changes to be made to the admin/teacher, as the students cannot edit their data once submitted, but the teacher/admin can.

- Once the student(s) have submitted their quiz, the results and the student data will immediately be accessible to the teacher/admin.

4. Lessons Learned

- This is my first time creating a full-stack Java Program that incorporates database and SQL. I had to watch and read tutorials about MySQL and how to utilize it in a Java Program. Aside from that, I also learned how to program the front-end design with Java via Netbeans IDE. It's my first time using this IDE and designing a program with Java that has a GUI. I learned how to use and create Java GUIs with JFrame and other Java Utility Libraries. It was a fun and very good experience as I learned a lot of new materials and how to implement them in Java with OOP.

5. Project Technical Descriptions

- **JFrame Libraries**

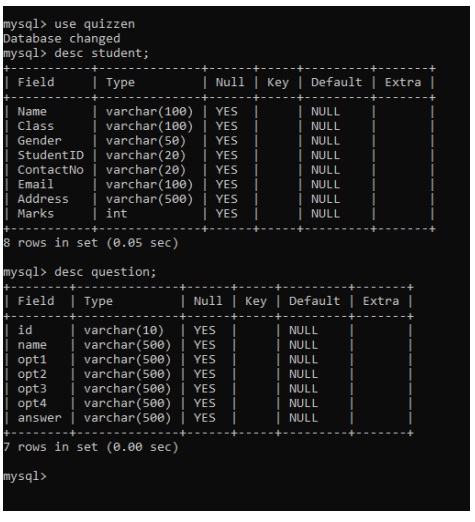
  Jframe is a container that helps create GUIs or window displays on the screen for java programs. They are in the form of a java class with which we can inherit to our class files that we want to create a GUI for. It comes with multiple components that we can insert into our window such as menu bars, panels, labels, text fields, buttons, etc. There are also alot of methods that comes with the javax.swing.Jframe class in order for us to customize our window displays and the components in it. Almost every other Swing application starts with the JFrame window.

  I used the Jframe class to create a display screen that is interactive for all my program files. Therefore all my runnable program files inherit the same JFrame class to create a Jframe form.

- **MySQL Database**

  MySQL is a RDBMS or a relational database management system that is based on structured query language or otherwise shortened as SQL. A database is a structured/organized collection of data. The data is organised within tables. The tables are built with rows and columns, each having its own function/meaning/relation with the data inside it and all having its own correlation to each other.

  I used MySQL database to create a database to store all Quizzen related data. I created it within my local computer. Two tables



```
mysql> use quizzen
Database changed
mysql> desc student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| Name      | varchar(100) | YES  |     | NULL    |       |
| Class     | varchar(100) | YES  |     | NULL    |       |
| Gender    | varchar(50)  | YES  |     | NULL    |       |
| StudentID | varchar(20)  | YES  |     | NULL    |       |
| ContactNo | varchar(20)  | YES  |     | NULL    |       |
| Email     | varchar(100) | YES  |     | NULL    |       |
| Address   | varchar(500) | YES  |     | NULL    |       |
| Marks     | int          | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
8 rows in set (0.05 sec)

mysql> desc question;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| id     | varchar(10)  | YES  |     | NULL    |       |
| name   | varchar(500) | YES  |     | NULL    |       |
| opt1   | varchar(500) | YES  |     | NULL    |       |
| opt2   | varchar(500) | YES  |     | NULL    |       |
| opt3   | varchar(500) | YES  |     | NULL    |       |
| opt4   | varchar(500) | YES  |     | NULL    |       |
| answer | varchar(500) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql>
```

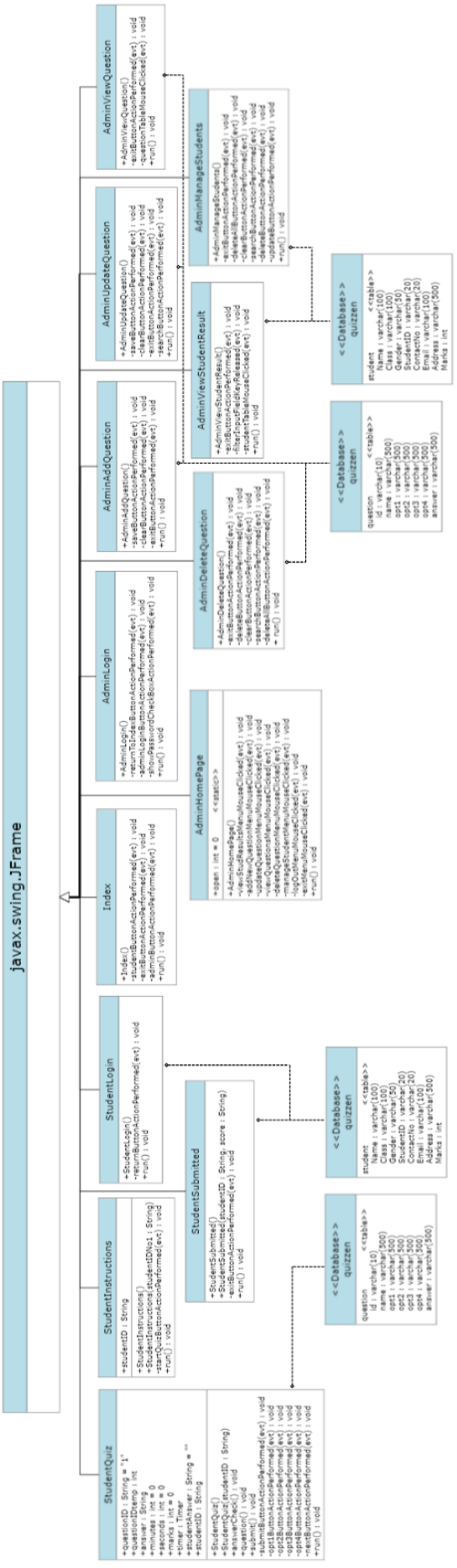
made in it are the ‘question’ table and the ‘student’ table. These tables then can be accessed in my java files. The student table is used to store student data, including their name, class, gender, student ID number, contact number, email, address, and their marks. The question table is used to store the individual quiz question data, including the question ID number, the question, the 4 individual options, and the answer to the question.

6. Program Libraries/Modules

- javax.swing.JFrame
- java.sql.*
- javax.swing.JOptionPane
- javax.swing.ImageIcon
- net.proteanit.sql.DbUtils
- java.text.SimpleDateFormat
- java.util.Date
- java.awt.event.ActionEvent
- java.awt.event.ActionListener
- javax.swing.Timer
- rs2xml
- MySQL Connector Jar File (JDBC)

B. Solution Design

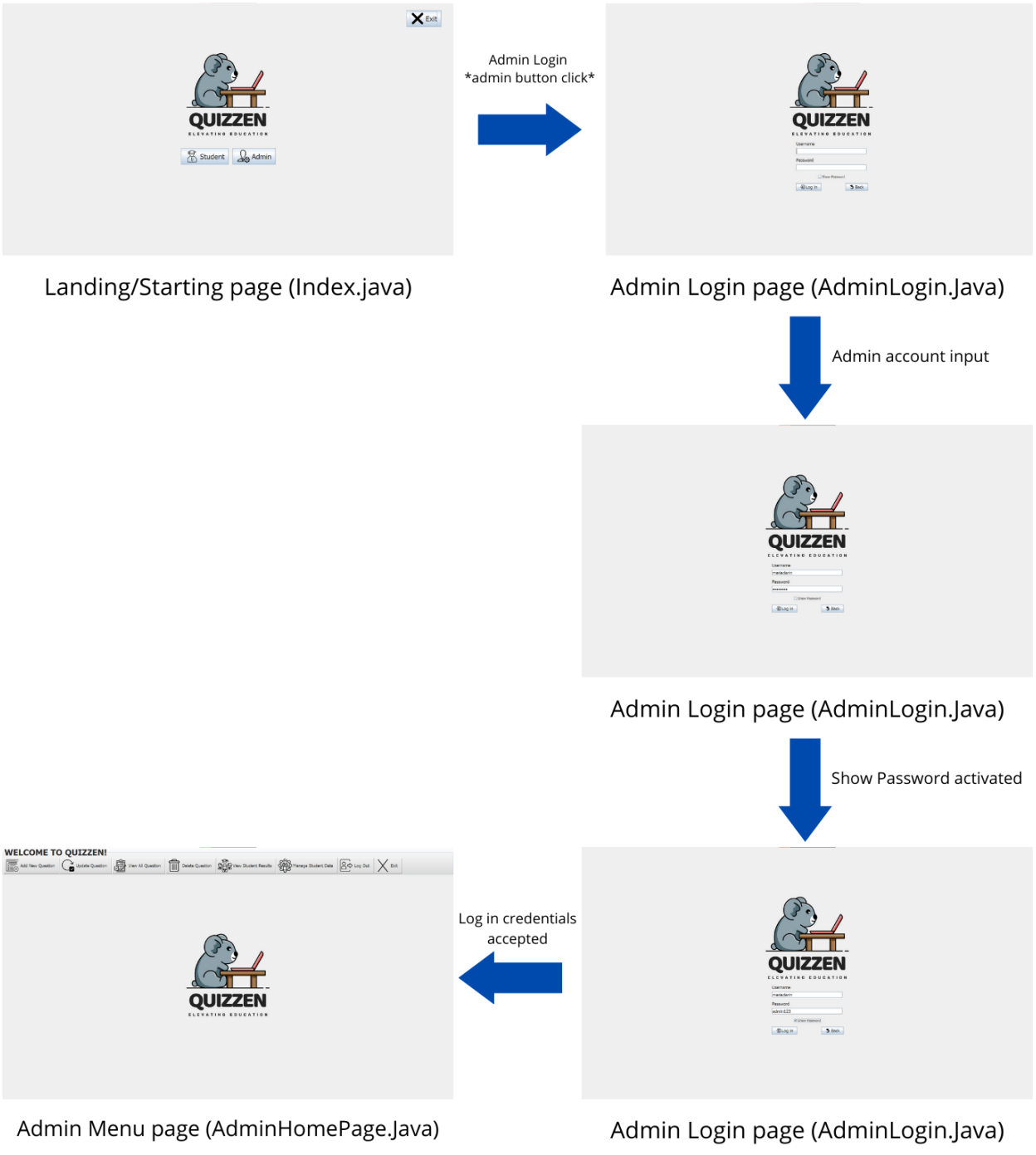
1. Program UML Diagram



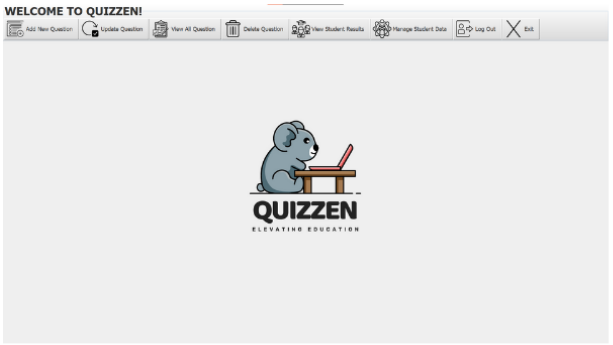
- All my program files inherit the JFrame class in order to create a GUI, therefore showing the inheritance connection to the javax.swing.Jframe class.
- The program files AdminAddQuestion.java, AdminUpdateQuestion.java, AdminViewQuestion.java, AdminDeleteQuestion.java, and StudentQuiz.java realizes/includes a MySQL table named ‘question’ in the database ‘quizzen’.
- The program files StudentLogin.java, StudentSubmitted.java, AdminViewStudentResult.java, and AdminManageStudents.java realizes or includes a MySQL table named ‘question’ in the database ‘quizzen’.

2. Program Flow

- **Admin Login**

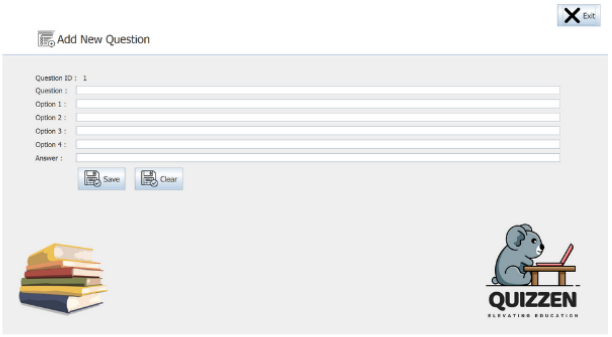


- **Admin Add Question Menu**



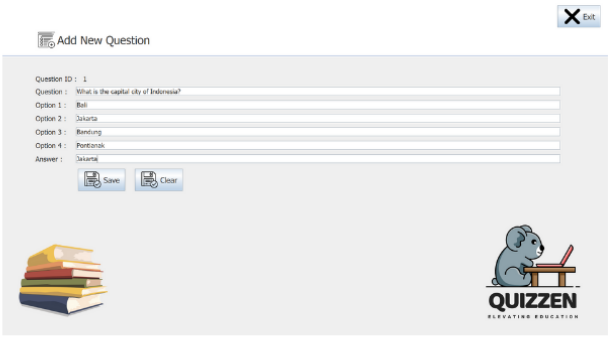
Admin Menu Page (AdminHomePage.java)

Add Question Menu Clicked



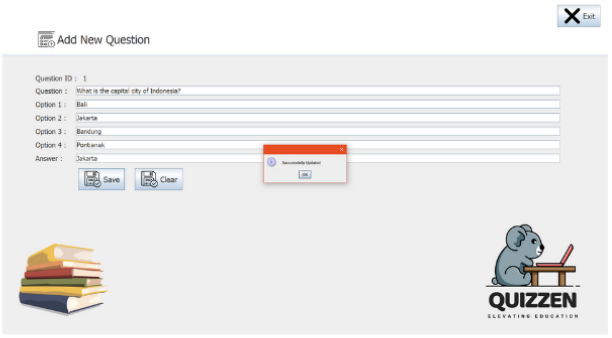
Admin Add Question Page (AdminAddQuestion.java)

Admin Question Input



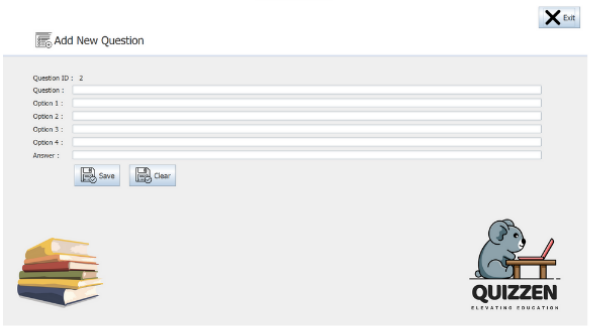
Admin Add Question Page (AdminAddQuestion.java)

Save button click



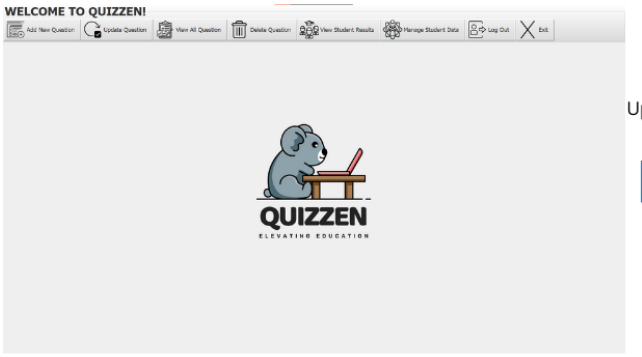
Admin Add Question Page (AdminAddQuestion.java)

Question saved

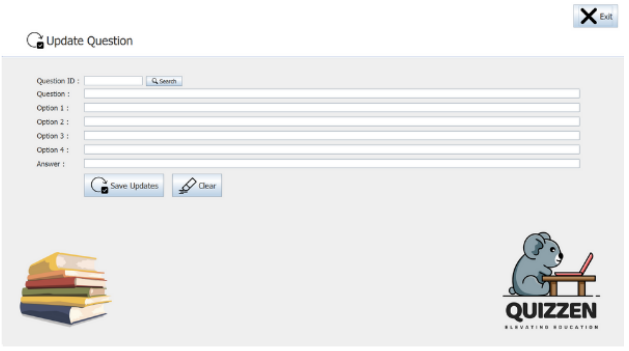


Question ID Updated (AdminAddQuestion.java)

- **Admin Update Question Menu**



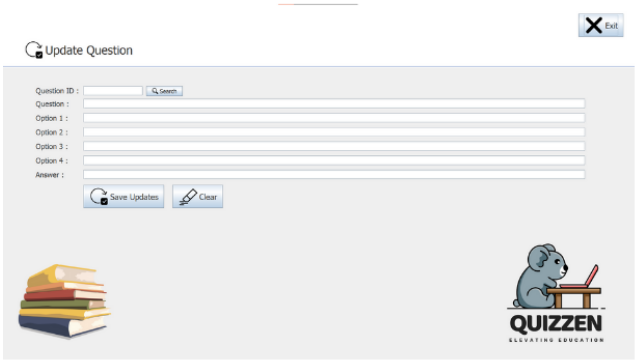
Update Question Menu Clicked

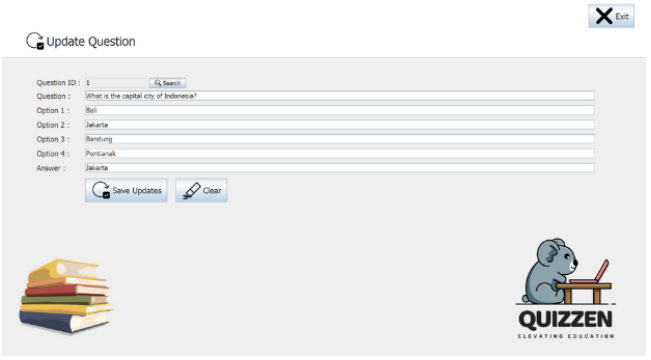


Admin Menu Page (AdminHomePage.java)

Admin Update Question page (AdminUpdateQuestion.java)

Question ID input and search clicked



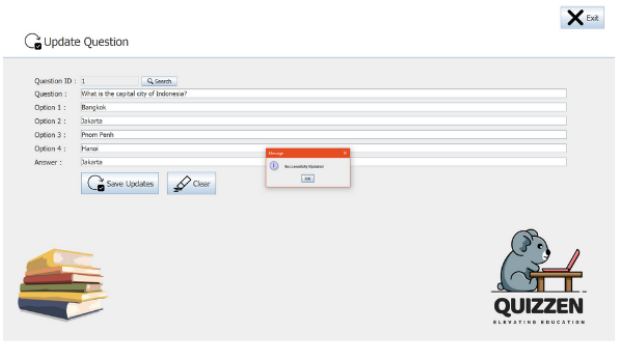


Update Question Page Refresh (AdminUpdateQuestion.java)

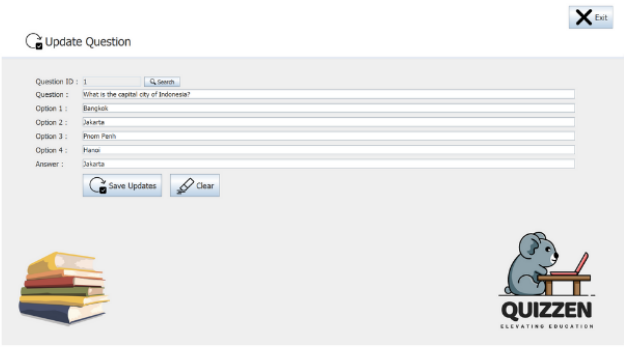
Question ID search result (AdminUpdateQuestion.java)

Exit confirm dialog

Question data edits

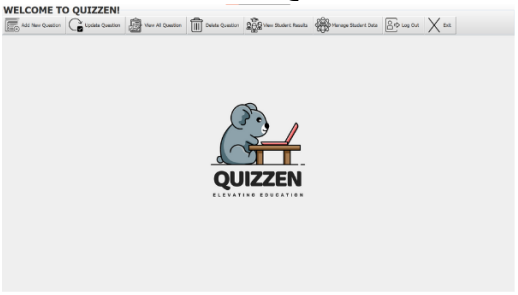


Question saved



Question Data Updated (AdminUpdateQuestion.java)

- **Admin View Questions Menu**

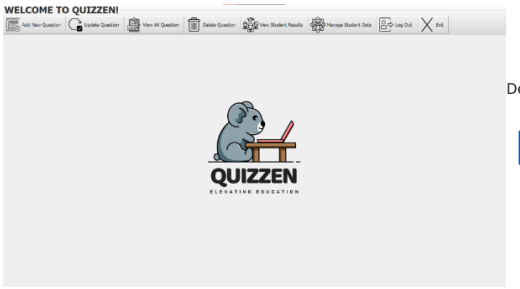


View Questions Menu Clicked

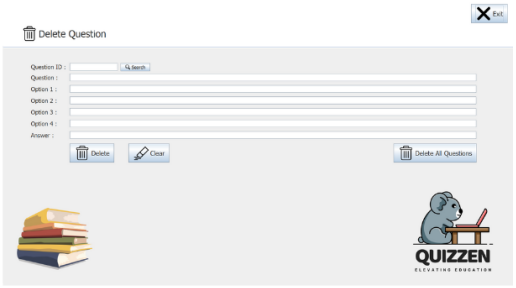
Admin Menu Page (AdminHomePage.java)

Admin View Questions page (AdminViewQuestion.java)

- **Admin Delete Questions Menu**



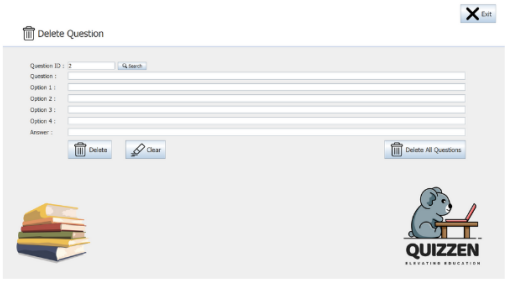
Delete Questions Menu Clicked



Admin Menu Page (AdminHomePage.java)

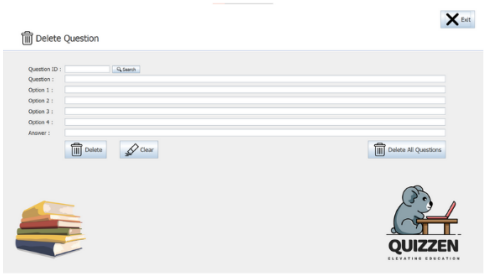
Admin Delete Questions page (AdminDeleteQuestion.java)

Question ID Input

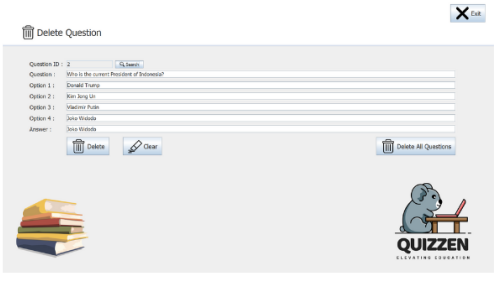


Admin Delete Questions page (AdminDeleteQuestion.java)

search button clicked



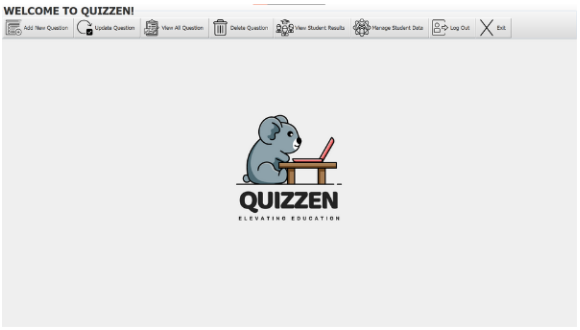
Question deleted



Delete Question page refresh (AdminDeleteQuestion.java)

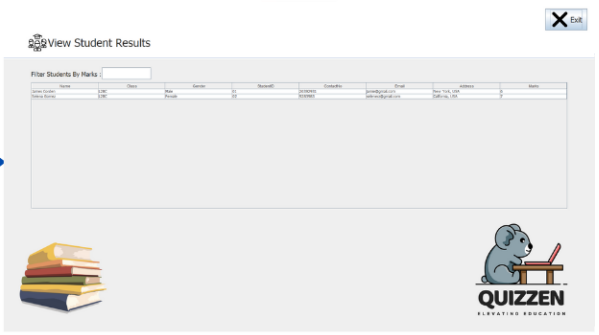
Question ID found (AdminDeleteQuestion.java)

- **Admin View Student Result**



Admin Menu Page (AdminHomePage.java)

View Student Results Menu Clicked

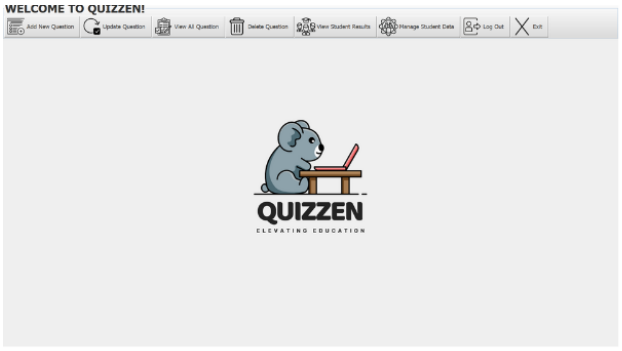


Displaying all student results (AdminViewStudentResult.java)

Filter Marks Input

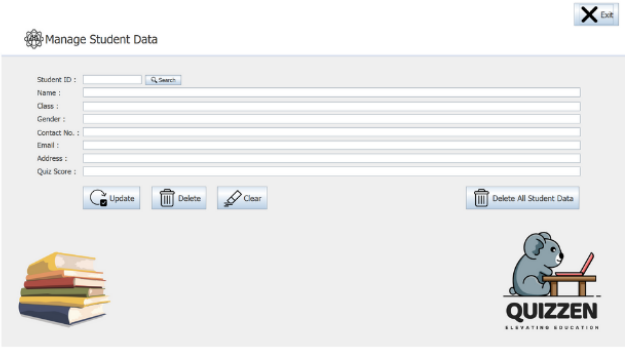
Filtered Result (AdminViewStudentResult.java)

- **Admin Manage Student Data**



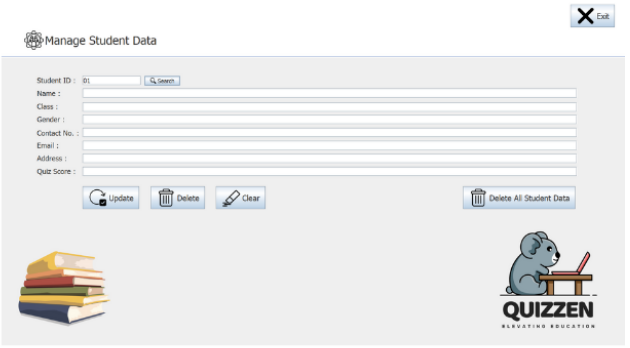
Admin Menu Page (AdminHomePage.java)

Manage Student Data Menu Clicked



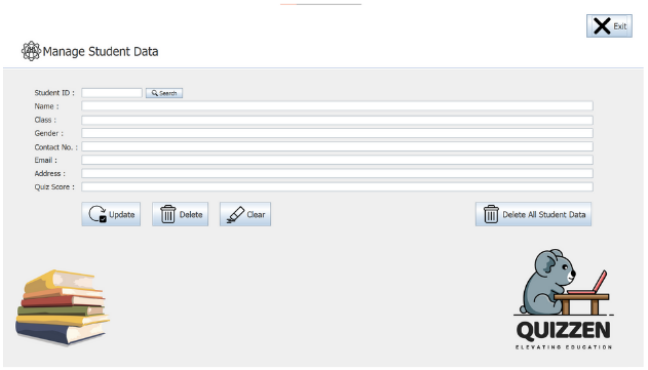
Admin Manage Student Data page (AdminManageStudents.java)

Student ID No. input



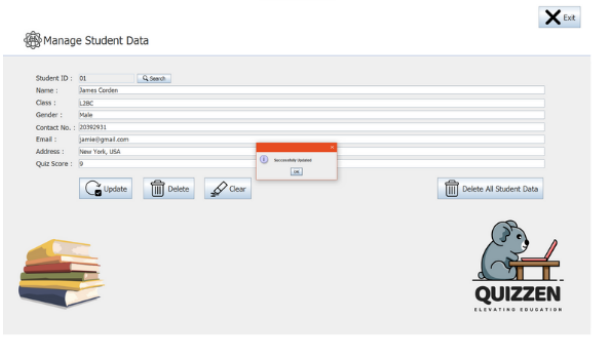
Admin Manage Student Data page (AdminManageStudents.java)

search button clicked



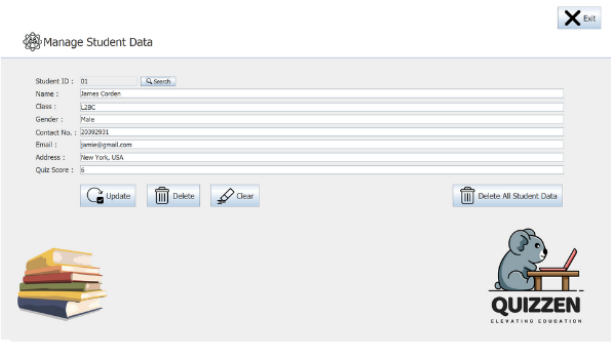
Admin Manage Student Data page refresh (AdminManageStudents.java)

Exit confirm dialog



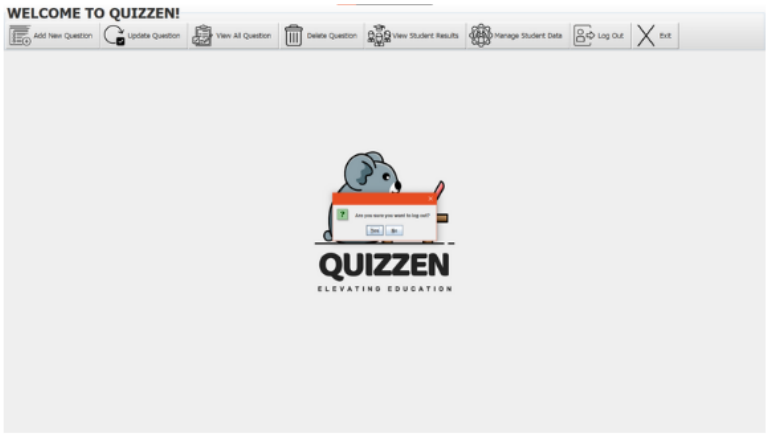
Updates Saved (AdminManageStudents.java)

Input Updates Update Button Clicked



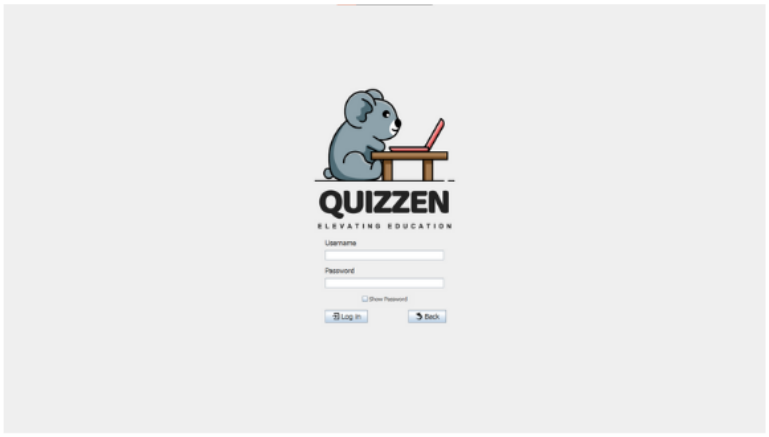
Searched data result (AdminManageStudents.java)

- **Admin Log Out Menu**



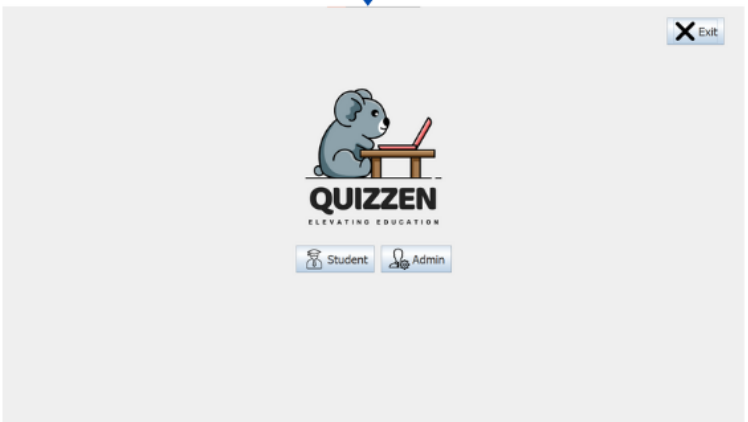
Log Out Menu Clicked

Exit confirm dialog



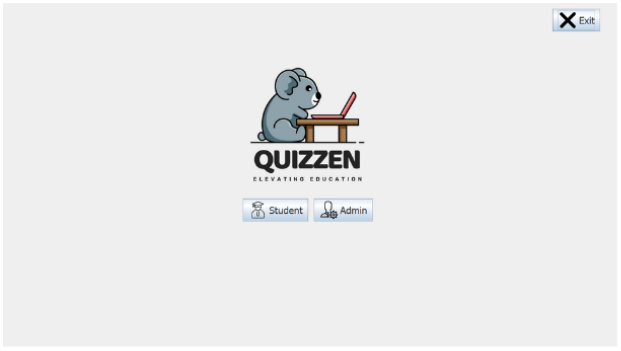
Admin Login Page

Back button clicked



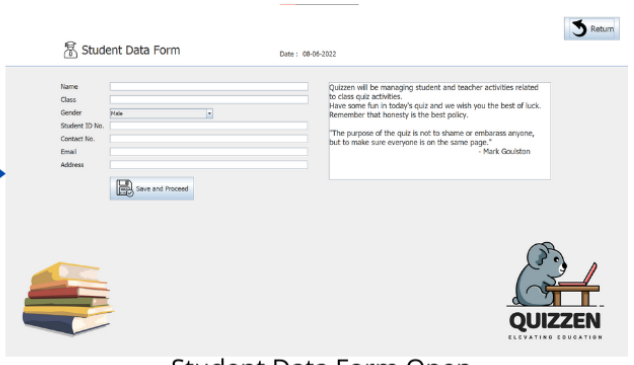
Back to Index page

- **Student Quiz Mechanism**



Landing Page (Index.java)

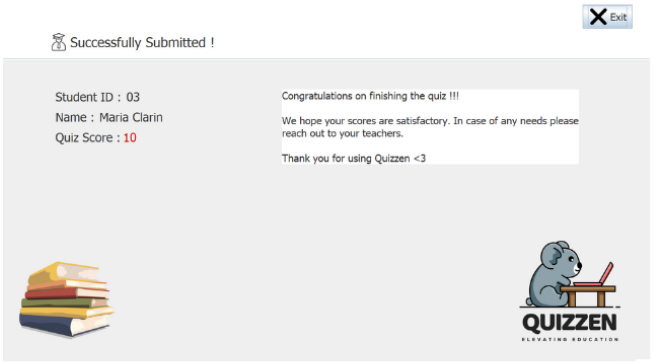
Student Button Menu Clicked



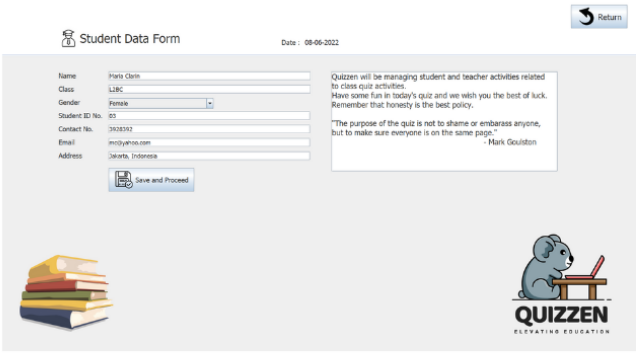
Student Data Form Open (StudentLogin.java)

Exit Button clicked

Student Data Form Input



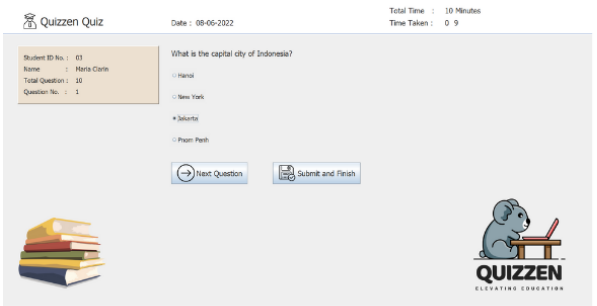
Student Result Page (StudentSubmitted.java)



Student Data Form Filled (StudentLogin.java)

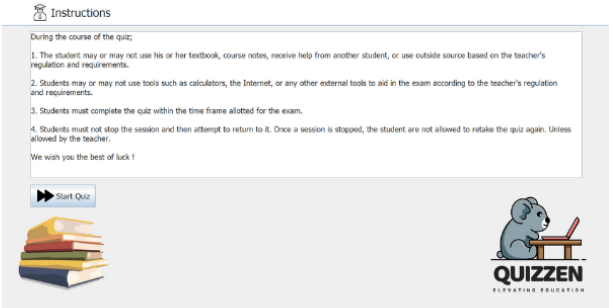
Quiz Submit Button Clicked

save button clicked



Quiz Starts, Timer Starts, Question No. Increments (StudentQuiz.java)

Start Quiz button Clicked

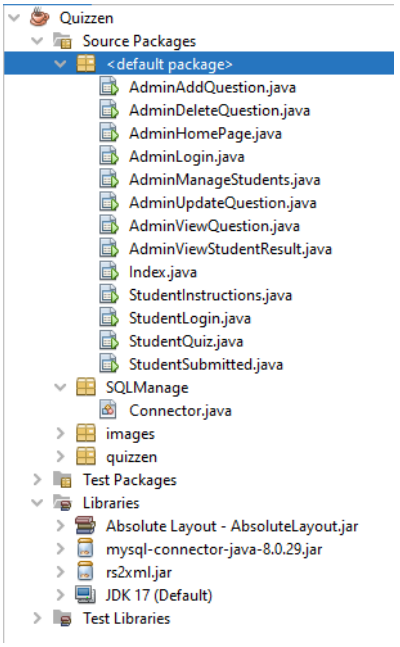


Student Instructions Page (StudentInstructions.java)

C. Code Design and Explanation

1. Project Structure

- All runnable files with a display screen are located in the Source Packages package, inside the <default package> package display in Netbeans IDE.
- A java class file used to establish connections between files and the MySQL database (Connector.java), is located in the SQLManage package inside the Source Packages package.
- All images used in the runnable files are included in the images package.
- All external libraries that need to be imported are included in the Libraries folder.
- To ease display and avoid errors, it is **best to access the Quizzen files with the Netbeans IDE.**



2. Connector.java

```
//importing necessary library and package
package SQLManage;
import java.sql.*;

//creating a new class Connector that will act as a connector between the mysql database and quizzen java interface
public class Connector {

    //private default constructor
    private Connector() {
    }

    //getCon() functions work to connect the database and the file
    public static Connection getCon()
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");

             //this is my local host sql syntax/query
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/quizzen", "root", "098761");
            return con;
        }
        catch(Exception e)
        {
            return null;
        }
    }
}
```

- The Connector class' function is to get a connection from my local MySQL database made for Quizzen. All I need to do is to import this packaged file and call the function of getCon().
- The getCon() function works to fetch and get a connection from my local address for the database.

```
 //this is my local host sql syntax/query
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/quizzen", "root", "098761");
return con;
```

- The green query is the address to access the database. The query “root” and “098761” is my MySQL account info in which the database is stored.
- Use case example (taken from some of my runnable files) :

```
//a public constructor of the AdminViewQuestion class
public AdminViewQuestion() {
    initComponents();

    //the try block commands will establish a connection between this file and the mysql database for quizzen
    try
    {
        Connection con = Connector.getCon();

        //creating a statement to declare as query in the mysql database
        Statement st = con.createStatement();

        //executing the query to select all the rows from the question table and displaying it into the questionTable
        ResultSet rs = st.executeQuery("select *from question");
        questionTable.setModel(DbUtils.resultSetToTableModel(rs));
    }

    //a catch block to go alongside the try block
    catch(Exception e)
    {
        //A message dialog popup that pops up if an error occurs within the try block.
        JOptionPane.showMessageDialog(null, e);
    }
}
```

```
//public constructor of the AdminViewStudentResult class
public AdminViewStudentResult() {
    initComponents();

    //the try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create statement to execute to query

        //executing a query to select all the rows in the student table and displaying them into the studentTable
        ResultSet rs = st.executeQuery("select *from student");
        studentTable.setModel(DbUtils.resultSetToTableModel(rs));
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occors
    catch(Exception e){
        //a message dialog that contains nothing, only for error handling purposes
        JOptionPane.showMessageDialog(null, e);
    }
}
```

```
//QUESTION FUNCTION
public void question(){

    //a try block to create connections between this file and the mysql database for quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create statement to be executed as a query in the mysql database

        //executing the query to select the row of data where the id matches with the questionID variable in the question table
        ResultSet rs = st.executeQuery("select *from question where id='"+questionID+"'");

        //while the mysql database returns a result
        while(rs.next())
        {
            //the questions will be displayed alongside the options and question number
            questionNoLabel.setText(rs.getString(1));
            questionLabel.setText(rs.getString(2));
            opt1Button.setText(rs.getString(3));
            opt2Button.setText(rs.getString(4));
            opt3Button.setText(rs.getString(5));
            opt4Button.setText(rs.getString(6));

            //the variable answer will store the correct answer based on the database to compare with the student's answer
            //in the function answerCheck()
            answer = rs.getString(7);
        }
    }

    //a catch block to go alongside the try block to catch any errors that may occur from the try block
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

3. Index.java

```
//Importing necessary libraries
import javax.swing.JOptionPane;

//Index.java inherits the class from the JFrame STL in order to make it a JFrame Form to create UI
public class Index extends javax.swing.JFrame {

    //Public constructor of the Index class
    public Index() {
        initComponents();
    }
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    studentButton = new javax.swing.JButton();
    adminButton = new javax.swing.JButton();
    exitButton = new javax.swing.JButton();
    indexBackground = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setUndecorated(true);
    getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

    studentButton.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
    studentButton.setIcon(new javax.swing.ImageIcon(getClass().getResource("/images/student.png"))); // NOI18N
    studentButton.setText("Student");
    studentButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            studentButtonActionPerformed(evt);
        }
    });
    getContentPane().add(studentButton, new org.netbeans.lib.awtextra.AbsoluteConstraints(760, 620, -1, -1));

    adminButton.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
    adminButton.setIcon(new javax.swing.ImageIcon(getClass().getResource("/images/admin.png"))); // NOI18N
    adminButton.setText("Admin");
    adminButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            adminButtonActionPerformed(evt);
        }
    });
    getContentPane().add(adminButton, new org.netbeans.lib.awtextra.AbsoluteConstraints(980, 620, -1, -1));

    exitButton.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
    exitButton.setIcon(new javax.swing.ImageIcon(getClass().getResource("/images/exit.png"))); // NOI18N
    exitButton.setText("Exit");
    exitButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            exitButtonActionPerformed(evt);
        }
    });
    getContentPane().add(exitButton, new org.netbeans.lib.awtextra.AbsoluteConstraints(1720, 30, -1, -1));

    indexBackground.setIcon(new javax.swing.ImageIcon(getClass().getResource("/images/backgroundIndex.png"))); // NOI18N
    getContentPane().add(indexBackground, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -1));

    pack();
}// </editor-fold>
// <editor-fold defaultstate="collapsed" desc="Generated Code">
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

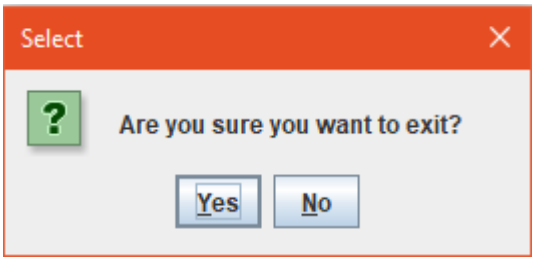
```java
//STUDENT LOGIN BUTTON EVENT
private void studentButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // when the student button is pressed, it will direct the user to the student login page.
    setVisible(false);                      // Index.java page will nolonger be visible
    new StudentLogin().setVisible(true);    // StudentLogin.java will popup and function
}
```

```java
//ADMIN LOGIN BUTTON EVENT
private void adminButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // when the admin button is pressed, it will direct the user to the admin login page.
    setVisible(false);                    // Index.java page will nolonger be visible
    new AdminLogin().setVisible(true);    // AdminLogin.java will popup and function
}
```

- The setup for the studentButton and adminButton for when an action is done with the component.
- For the studentButtonActionPerformed(evt) function, once the user clicks on the student button, the program will set the Index.java file as not visible (exits the display of that specific file) and will start running the StudentLogin.java file by calling the class constructor StudentLogin() and setting it as visible.
- For the adminButtonActionPerformed(evt) function, once the user clicks on the admin button, the program will set the Index.java file as not visible (exits the display of that specific file) and will start running the AdminLogin.java file by calling the class constructor AdminLogin() and setting it as visible.

```java
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
   // when the exit button is pressed, there will be a confirm dialog popup with a Yes or No option.
    int userConfirmation=JOptionPane.showConfirmDialog(null, "Are you sure you want to exit?", "Select", JOptionPane.YES_NO_OPTION);

    //if the user clicks on the button Yes (the syntax is ==0) the system will exit.
    //if the user clicks on the button No, the popup will disappear and the current page will be accessible.
    if(userConfirmation==0)
    {
        System.exit(0);
    }
}
```

- The set up for the exitButton for when an action is done with the component.
- For the exitButtonActionPerformed(evt) function, once the user clicks on the exit button, the program will generate a pop-up confirm dialog for the user to confirm whether or not they really want to exit the program.



- This confirm dialog comes with two options, Yes or No. If the user clicks on the Yes button (the syntax being 0 therefore the if statement would be if(userConfirmation==0)), The program will exit itself and all running files will be terminated. If the user clicks on the No button, the program will exit the confirm dialog and display the previously displayed page (Index.java).
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```java
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Index().setVisible(true);
        }
    });
}
```

4. AdminLogin.java

```
//importing necessary libraries
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

//AdminLogin.java inherits the class from the JFrame STL class in order to make it a JFrame Form to create UI
public class AdminLogin extends javax.swing.JFrame {

    //public constructor of the AdminLogin CLass
    public AdminLogin() {
        initComponents();
    }

    /**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.
- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

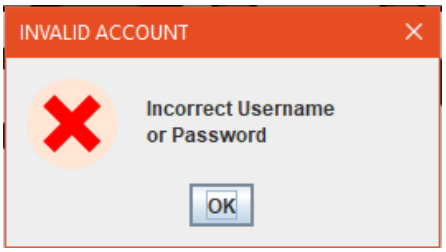
```
//RETURN BUTTON EVENT
private void returnToIndexButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the user clicks on the return button, they will return to the main landing page (index.java)
    setVisible(false);
    new Index().setVisible(true);
}
```

- The setup for the returnToIndexButton for when an action is done with the component.
- For the returnToIndexButtonActionPerformed(evt) function, once the user clicks on the return button, the program will exit the display of the AdminLogin page and return to the Index page display.

```
//ADMIN LOGIN BUTTON ACTION SET UP
private void adminLoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //will check if the admin login credentials match in order to be able to login
    //creating an admin account with the username 'mariaclarin' with the password 'admin123'
    if(adminUserInput.getText().equals("mariaclarin")&& adminPasswordInput.getText().equals("admin123")){

        //if the credentials match, the user will be able to access the admin homepage.
        //it means that the login is successful
        setVisible(false);
        new AdminHomePage().setVisible(true);
    }

    //If the credentials doesnt match, a popup message will appear stating incorrect username or password.
    else
    {
        ImageIcon icon=new ImageIcon("passWrong.PNG");
        JOptionPane.showMessageDialog(null,"<html>Incorrect Username <br> or Password</b></html>", "Show", JOptionPane.INFORMATION_MESSAGE, icon);
    }
}
```

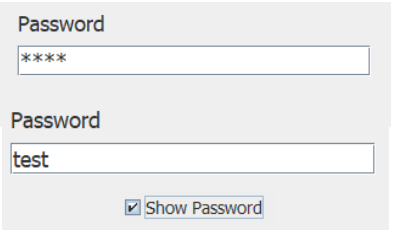
- The setup for the adminLoginButton for when an action is done with the component.
- When the user inputs their login credentials, if the username input is equal to “mariaclarin” and the password input is equal to “admin123”, then the program will exit the display of the AdminLogin page and start running the AdminHomePage.
- For now, this is the only way to register a new account/login credential to be able to log in to access admin menus. I plan to improve the code so the user can register a new admin account via the GUI instead of manually editing the code :D.

- However if the login credentials don't match the declared credentials, a pop-up message dialog will appear stating that they inputted the wrong password or username.




```
//SHOW PASSWORD CHECKBOX ACTION SET UP
private void showPasswordCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
    // if the show password checkbox is selected, it will show the password chars
    if(showPasswordCheckBox.isSelected()){
        adminPasswordInput.setEchoChar((char)0);
    }

    //if it is not selected, it will show the individual chars as *
    else
    {
        adminPasswordInput.setEchoChar('*');
    }
}
```

- The setup for the showPasswordCheckBox for when an action is done with the component.
- If the user clicks on the checkbox, the chars they enter in the password input box will be displayed. Else, if the checkbox is not clicked, every char entered in the password input box will be displayed as * (asterisk symbol).



- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminLogin().setVisible(true);
        }
    });
}
```

5. AdminHomePage.java

```
//importing the necessary libraries
import javax.swing.JFrame;
import javax.swing.JOptionPane;

//AdminHomePage.java inherits the class from the JFrame STL in order to make it a JFrame Form to create UI
public class AdminHomePage extends javax.swing.JFrame {

    //declaring a static attribute called open, will be used to run menu pages for the Admin
    public static int open=0;

    //public constructor of the AdminHomePage Class
    public AdminHomePage() {
        initComponents();
    }

    /**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.
- Declaring a public static int attribute called open that has the value 0. This attribute will be very useful in determining if a menu page is currently running or not.
- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//ADD NEW QUESTION MENU EVENT
private void addNewQuestionMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the Add New Question menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        new AdminAddQuestion().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "An Add Question form is already open.");
    }
}
```

- The setup for the addNewQuestionMenu for when the user clicks on the menu box.
- If the user clicks on the addNewQuestionMenu box while the menu page has not been open yet, the program will launch the AdminAddQuestion page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//UPDATE QUESTION MENU EVENT
private void updateQuestionMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the Update Question menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        new AdminUpdateQuestion().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "An Update Question form is already open.");
    }
}
```

- The setup for the updateQuestionMenu for when the user clicks on the menu box.
- If the user clicks on the updateQuestionMenu box while the menu page has not been open yet, the program will launch the AdminUpdateQuestion page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//VIEW QUESTIONS MENU EVENT
private void viewQuestionsMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the View Questions menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        new AdminViewQuestion().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "A View Question form is already open.");
    }
}
```

- The setup for the viewQuestionMenu for when the user clicks on the menu box.
- If the user clicks on the viewQuestionMenu box while the menu page has not been open yet, the program will launch the AdminViewQuestion page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//DELETE QUESTION MENU EVENT
private void deleteQuestionMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the Delete Question menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        new AdminDeleteQuestion().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "A Delete Question form is already open.");
    }
}
```

- The setup for the deleteQuestionMenu for when the user clicks on the menu box.
- If the user clicks on the deleteQuestionMenu box while the menu page has not been open yet, the program will launch the AdminDeleteQuestion page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//VIEW STUDENT RESULT MENU EVENT
private void viewStudResultsMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the View Student Results menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        //the AdminViewStudentResult page will start running.
        new AdminViewStudentResult().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "A View Student Result form is already open.");
    }
}
```

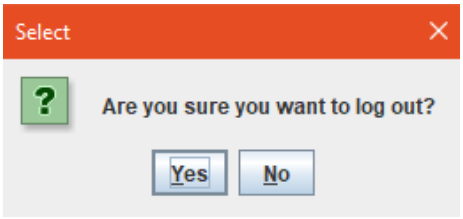
- The setup for the viewStudResultMenu for when the user clicks on the menu box.
- If the user clicks on the viewStudResultMenu box while the menu page has not been open yet, the program will launch the AdminViewStudentResult page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//MANAGE STUDENTS MENU EVENT
private void manageStudentMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the Manage Students menu and the currently opened form for this menu is still 0
    if(open==0)
    {
        new AdminManageStudents().setVisible(true);
        open=1;
    }

    //if there is already a running menu form, a JFrame message dialog will popup if the user clicks to open another form
    else
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "A Manage Student Data form is already open.");
    }
}
```

- The setup for the manageStudentMenu for when the user clicks on the menu box.
- If the user clicks on the manageStudentMenu box while the menu page has not been open yet, the program will launch the AdminManageStudents page and set the open variable to have the value of 1. If the value of the open variable is not equal to 0, a pop-up message dialog will appear to remind the user that an active menu display is already running.

```
//LOG OUT MENU EVENT
private void logOutMenuMouseClicked(java.awt.event.MouseEvent evt) {
    //if the user clicks on the log out menu, a JFrame popup will appear to ask for user confirmation
    JFrame jf = new JFrame();
    jf.setAlwaysOnTop(true);
    int userConfirmation = JOptionPane.showConfirmDialog(jf, "Are you sure you want to log out?", "Select", JOptionPane.YES_NO_OPTION);

    //if the user clicks on the Yes button, they will return to the AdminLogin page.
    if(userConfirmation==0){
        setVisible(false);
        new AdminLogin().setVisible(true);
    }
}
```

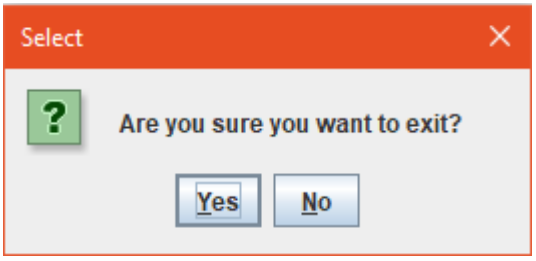
- The set up for the logOutMenu for when the user clicks on it
- Once the user clicks on the logout menu, the program will generate a pop-up confirm dialog for the user to confirm whether or not they really want to log out from the current account.



- This confirm dialog comes with two options, Yes or No. If the user clicks on the Yes button (the syntax being 0 therefore the if statement would be if(userConfirmation==0)), The program will return the user to the AdminLogin page. If the user clicks on the No button, the program will exit the confirm dialog and display the previously displayed page (AdminHomePage.java).

```java
//EXIT MENU EVENT
private void exitMenuMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user clicks on the exit menu, a Jframe popup will appear to ask for user confirmation
    JFrame jf = new JFrame();
    jf.setAlwaysOnTop(true);
    int userConfirmation = JOptionPane.showConfirmDialog(jf, "Are you sure you want to exit?", "Select", JOptionPane.YES_NO_OPTION);

    //if the user clicks on the Yes button, the application will be terminated.
    if(userConfirmation==0){
        System.exit(0);
    }
}
```

- The set up for the exitMenu for when the user clicks on it
- Oonce the user clicks on the exit menu, the program will generate a pop-up confirm dialog for the user to confirm whether or not they really want to exit the program.



- This confirm dialog comes with two options, Yes or No. If the user clicks on the Yes button (the syntax being 0 therefore the if statement would be if(userConfirmation==0)), The program will exit itself and all running files will be terminated. If the user clicks on the No button, the program will exit the confirm dialog and display the previously displayed page (AdminHomePage.java).
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```java
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminHomePage().setVisible(true);
        }
    });
}
```

6. AdminAddQuestion.java

```java
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

//AdminAddQuestion.java inherits the class from the JFrame STL in order to make it a JFrame Form to create UI
public class AdminAddQuestion extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, and inheriting the Jframe class.

```
//public constructor of the AdminAddQuestion Class
public AdminAddQuestion() {
    initComponents();

    //The try block commands will establish a connection between this file and the mysql database made for Quizzen
    try{
        Connection con = Connector.getCon();
        Statement st = con.createStatement();                          //creating a statement in mysql
        ResultSet rs = st.executeQuery("select count(id) from question"); //takes the current question ID from the 'id' collumn of the mysql table 'question'
        if(rs.next())                                                  //if the mysql terminal comes up with a result
        {
            //the value of the current id will be displayed in the questionIDLabel
            int id = rs.getInt(1);
            id = id+1;
            String str = String.valueOf(id);
            questionIDlabel.setText(str);
        }
        else //if the mysql terminal doesnt have a result (the collumn is empty)
        {
            questionIDlabel.setText("1"); //the value '1' will be displayed in the questionIDLabel
        }
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs
    catch(Exception e){
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error.
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- A public constructor of the AdminAddQuestion class. The try and catch block will be run whenever the constructor of the class is called.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of creating a statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- I fetched the data of how much data is stored in the id column of the table to count the current questionID and display it in the questionID label. If the count of the data in the column id is 0, it will display 1 instead. And for each count, the id will be added by 1 to the value before being displayed so it shows the next question ID that is still available to be filled.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
/**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//SAVE BUTTON EVENT
private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // creating variables to store the user inputs from the corresponding input fields.
    String id = questionIDlabel.getText();
    String name = questionInputField.getText();
    String opt1 = opt1InputField.getText();
    String opt2 = opt2InputField.getText();
    String opt3 = opt3InputField.getText();
    String opt4 = opt4InputField.getText();
    String answer = answerInputField.getText();

    //The try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con= Connector.getCon();
        PreparedStatement ps = con.prepareStatement("insert into question values(?,?,?,?,?,?,?)"); //declaring the query into the mysql database

        //inserting the data stored in the previously declared variables into the corresponding collumns of data in a row
        ps.setString(1, id);
        ps.setString(2, name);
        ps.setString(3, opt1);
        ps.setString(4, opt2);
        ps.setString(5, opt3);
        ps.setString(6, opt4);
        ps.setString(7, answer);
        ps.executeUpdate(); //executing the update in the mysql database

        //creating a popup message dialog with JFrame to display success message.
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "Successfully Updated");

        //when the user exits the popup message dialog, the add question page will refresh.
        setVisible(false);
        new AdminAddQuestion().setVisible(true);
    }

    //a catch block to go alongside the try block if the try block fails.
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error.
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the saveButton for when an action is done with the component.
- I created some variables for storage. I stored all the user inputs from each InputField available into the variables.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The prepared statement will declare the query to store all the data by inserting them as a whole new row and organizing the data according to the columns. So when the user is done inputting all the text fields to create a new question and they press the save button, the question will be added to the table in the database.
- A pop-up message dialog will appear to notify the user of the successful update. And once the user exits the message dialog, the page will refresh.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//CLEAR BUTTON EVENT
private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // when the user presses the clear button, they will clear every single input field available on the screen.
    // this will not make changes to the mysql database
    questionInputField.setText("");
    opt1InputField.setText("");
    opt2InputField.setText("");
    opt3InputField.setText("");
    opt4InputField.setText("");
    answerInputField.setText("");
}
```

- The setup for the clearButton for when an action is done to the component.
- It will set all the input fields to have a blank display when the user clicks on the clear button

```
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // when the user clicks on the exit button, they will return to the AdminHomePage while the
    // AdminAddQuestion page will disappear.
    AdminHomePage.open=0;
    setVisible(false);
}
```

- The setup for the exitButton for when an action is done to the component.
- It will return the open value to 0, therefore, allowing another form to run and it will exit the page display and return to the AdminHomePage page.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminAddQuestion().setVisible(true);
        }
    });
}
```

7. AdminUpdateQuestion.java

```
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

//AdminUpdateQuestion.java inherits the class from the JFrame STL in order to make it a JFrame form to create UI
public class AdminUpdateQuestion extends javax.swing.JFrame {

    //public constructor for the AdminUpdateQuestion Class
    public AdminUpdateQuestion() {
        initComponents();
    }

    /**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.
- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//SAVE BUTTON EVENT
private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // variables to store user input from the input field
    String id = questionIDInputField.getText();
    String name = questionInputField.getText();
    String opt1 = opt1InputField.getText();
    String opt2 = opt2InputField.getText();
    String opt3 = opt3InputField.getText();
    String opt4 = opt4InputField.getText();
    String answer = answerInputField.getText();

    //the try command will establish a connection between this file and the mysql database for quizzen
    try
    {
        Connection con= Connector.getCon();

        //declaring the query into the mysql database to update the question table according to the corresponding data in the row where the id matches the user input
        PreparedStatement ps = con.prepareStatement("update question set name=?, opt1=?, opt2=?, opt3=?, opt4=?, answer=?, id=? where id="+id+"");
        ps.setString(1, name);
        ps.setString(2, opt1);
        ps.setString(3, opt2);
        ps.setString(4, opt3);
        ps.setString(5, opt4);
        ps.setString(6, answer);
        ps.setString(7, id);

        //executing the update. After the update, a pop up message will appear
        ps.executeUpdate();
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "Successfully Updated");

        //once the user exits the popup message, the AdminUpdateQuestion page will refresh
        setVisible(false);
        new AdminUpdateQuestion().setVisible(true);
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error orccurs
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the saveButton for when an action is done with the component.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The prepared statement will declare the query to update the data stored in individual columns where the id number matches the user input of the id.
- A pop-up message dialog will appear to notify the user of the successful update. And once the user exits the message dialog, the page will refresh.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//CLEAR BUTTON EVENT
private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks on the clear button, the program will clear every input field available on the page.
    questionIDInputField.setText("");
    questionInputField.setText("");
    opt1InputField.setText("");
    opt2InputField.setText("");
    opt3InputField.setText("");
    opt4InputField.setText("");
    answerInputField.setText("");
    questionIDInputField.setEditable(true);
}
```

- The setup for the clearButton for when an action is done to the component.
- It will set all the input fields to have a blank display when the user clicks on the clear button

```
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks on the exit button, they will return to the AdminHomePage
    AdminHomePage.open=0;
    setVisible(false);
}
```

- The setup for the exitButton for when an action is done to the component.
- It will return the open value to 0, therefore, allowing another form to run and it will exit the page display and return to the AdminHomePage page.

```
//SEARCH BUTTON EVENT
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //a variable to store the user input from the questionIDInputField
    String id = questionIDInputField.getText();

    //the try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con = Connector.getCon();

        //creating a statement to declare as a query for the mysql database
        Statement st= con.createStatement();

        //executing the query to select the data from the row in the question table where the id matches the user input
        ResultSet rs = st.executeQuery("select *from question where id='"+id+"'");

        //if the mysql database comes up with a result
        if(rs.next()){
            //the input fields will display the fetched data from the database
            questionInputField.setText(rs.getString(2));
            opt1InputField.setText(rs.getString(3));
            opt2InputField.setText(rs.getString(4));
            opt3InputField.setText(rs.getString(5));
            opt4InputField.setText(rs.getString(6));
            answerInputField.setText(rs.getString(7));
            questionIDInputField.setEditable(false);
        }

        //if the mysql database does not come up with a result
        else
        {
            //a JFrame popup stating the ID doesnt exist will appear
            JFrame jf = new JFrame();
            jf.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf, "Question ID not found.");
        }
    }

    //a catch block to go alongside the try block
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the searchButton for when an action is done to the component.
- I created some a variable for storage called id, which stores the user input taken from the questionIDInputField.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a create statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The create statement will declare the query to select data from the id column where the id that was inputted by the user is equal to the data.

- If it comes up with a result, all the input fields will be set to display the corresponding data from the same row from different columns (name, opt1, opt2, opt3, opt4, answer). Along with setting the question id input field editable to false.
- If the search doesn't come up with a result, a pop-up message dialog will appear to state that the question id is invalid.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminUpdateQuestion().setVisible(true);
        }
    });
}
```

8. AdminViewQuestion.java

```
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import net.proteanit.sql.DbUtils;

//AdminViewQuestion.java inherits from the JFrame STL in order to make it a Jframe form to create UI
public class AdminViewQuestion extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, and inheriting the Jframe class.

```
//a public constructor of the AdminViewQuestion class
public AdminViewQuestion() {
    initComponents();

    //the try block commands will establish a connection between this file and the mysql database for quizzen
    try
    {
        Connection con = Connector.getCon();

        //creating a statement to declare as query in the mysql database
        Statement st = con.createStatement();

        //executing the query to select all the rows from the question table and displaying it into the questionTable
        ResultSet rs = st.executeQuery("select *from question");
        questionTable.setModel(DbUtils.resultSetToTableModel(rs));
    }

    //a catch block to go alongside the try block
    catch(Exception e)
    {
        //A message dialog popup that pops up if an error occurs within the try block.
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- A public constructor of the AdminViewQuestion class. The try and catch block will be run whenever the constructor of the class is called.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of creating a statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- I selected all the data from the question table and set it to be displayed in the questionTable.

- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
/**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the user clicks on the exit button, they will return to the AdminHomePage
    AdminHomePage.open=0;
    setVisible(false);
}
```

- The setup for the exitButton for when an action is done to the component.
- It will return the open value to 0, therefore, allowing another form to run and it will exit the page display and return to the AdminHomePage page.

```
//TABLE DISPLAY SETUP
private void questionTableMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user tries to edit the display table in this form
    boolean edits = questionTable.isEditing();
    if(edits==false){
        //a popup JFrame will appear stating that they can not do so.
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "You can not edit this table in this form.");
    }
}
```

- Additional setup for the table display. Everytime the user clicks on the questionTable, a pop-up message dialog will appear stating that they cannot edit the table directly from the form.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminViewQuestion().setVisible(true);
        }
    });
}
```

9. AdminDeleteQuestion.java

```java
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

//AdminDeleteQuestion.java inherits the class from the JFrame STL in order to make a JFrame Form to create UI
public class AdminDeleteQuestion extends javax.swing.JFrame {

    //public constructor for the AdminDeleteQuestion Class
    public AdminDeleteQuestion() {
        initComponents();
    }


    /**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.
- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```java
//SEARCH BUTTON EVENT
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // a string variable to store the user input in the questionIDInputField
    String id = questionIDInputField.getText();

    //the try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st= con.createStatement(); //creating a statement to declare into query

        //executing the query command. Selecting the entire row where the questionID matches the data in the collumn id
        ResultSet rs = st.executeQuery("select *from question where id='"+id+"'");

        //if the mysql database comes up with a result
        if(rs.next()){
            //this if statement will set the text in the input fields according to the data fetched in the mysql database.
            questionInputField.setText(rs.getString(2));
            opt1InputField.setText(rs.getString(3));
            opt2InputField.setText(rs.getString(4));
            opt3InputField.setText(rs.getString(5));
            opt4InputField.setText(rs.getString(6));
            answerInputField.setText(rs.getString(7));
            questionIDInputField.setEditable(false);
        }
        else
        {
            //if the mysql database comes up with no result, a popup message will appear to state that the questionID is invalid
            JFrame jf = new JFrame();
            jf.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf, "Question ID not found.");
        }
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs.
    //displays a JFrame that is basically nothing. If the catch block is activated, i take it as an indication that there is an error.
    catch(Exception e)
    {
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the searchButton for when an action is done to the component.

- I created some a variable for storage called id, which stores the user input taken from the questionIDInputField.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a create statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The create statement will declare the query to select data from the id column where the id that was inputted by the user is equal to the data.
- If it comes up with a result, all the input fields will be set to display the corresponding data from the same row from different columns (name, opt1, opt2, opt3, opt4, answer). Along with setting the question id input field editable to false.
- If the search doesn't come up with a result, a pop-up message dialog will appear to state that the question id is invalid.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//CLEAR BUTTON EVENT
private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //when the user presses the clear button, they will clear every single input field available on the screen
    //this will not make changes to the mysql database
    questionIDInputField.setText("");
    questionInputField.setText("");
    opt1InputField.setText("");
    opt2InputField.setText("");
    opt3InputField.setText("");
    opt4InputField.setText("");
    answerInputField.setText("");
    questionIDInputField.setEditable(true);
}
```

- The setup for the clearButton for when an action is done to the component.
- It will set all the input fields to have a blank display when the user clicks on the clear button

```
//DELETE BUTTON EVENT
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // creating the variable 'id' to store the user inputs from the questionIDInputField
    String id = questionIDInputField.getText();

    //The try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con= Connector.getCon();
        PreparedStatement ps = con.prepareStatement("delete from question where id=?"); //declaring the query into the mysql database

        //inserting the data stored in the previously declared variables into the corresponding collumns of data in a row
        ps.setString(1, id);
        ps.executeUpdate(); //executing the update in the mysql database

        //Once the deletion is complete, a new JFrame popup will appear with the successful message.
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "Successfully Deleted");

        //Once the user exits the Jframe popup, the AdminDeleteQuestion Page will refresh.
        setVisible(false);
        new AdminDeleteQuestion().setVisible(true);
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing, if the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the deleteButton for when an action is done to the component.
- I created some a variable for storage called id, which stores the user input taken from the questionIDInputField.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The prepared statement will declare the query to select data from the id column where the id that was inputted by the user is equal to the data.
- If it comes up with a result, the row containing the corresponding data will be deleted.
- A pop-up message dialog will pop up to declare the successful deletion.
- If the search doesn't come up with a result, a pop-up message dialog will appear to state that the question id is invalid.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//DELETE BUTTON EVENT
private void deleteAllButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks on the delete button, a JFrame confirm dialog will appear to ask for user confirmation.
    JFrame jf = new JFrame();
    jf.setAlwaysOnTop(true);
    int userConfirmation=JOptionPane.showConfirmDialog(jf, "Are you sure you want to delete ALL QUESTIONS?", "Select", JOptionPane.YES_NO_OPTION);

    //if the user clicks the button Yes (the syntax ==0)
    if(userConfirmation==0)
    {
        //This try block will run to establish a connection between this file and the mysql database made for Quizzen
        try
        {
            Connection con= Connector.getCon();
            PreparedStatement ps = con.prepareStatement("delete from question"); //declaring the query into the mysql database to delete all rows from the table 'question'

            //executing the update in the mysql database
            ps.executeUpdate();

            //A new JFrame message dialog will appear and show the success message.
            JFrame jf1 = new JFrame();
            jf1.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf1, "Successfully Deleted");

            //once the user exits the JFrame popup, the AdminDeleteQuestion page will refresh
            setVisible(false);
            new AdminDeleteQuestion().setVisible(true);
        }

        //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs.
        catch(Exception e)
        {
            //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error.
            JFrame jf2 = new JFrame();
            jf2.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf2, e);
        }
    }
}
```

- The setup for the deleteAllButton for when an action is done to the component.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- A pop-up confirm dialog will appear to ask for user confirmation upon deletion of all the question table data. If the user clicks on the Yes button, the program will proceed by declaring the prepared statement to delete all the data in all rows in the question table.
- A pop-up message dialog will pop up to declare the successful deletion.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminDeleteQuestion().setVisible(true);
        }
    });
}
```

10. AdminViewStudentResult.java

```
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JOptionPane;
import net.proteanit.sql.DbUtils;
import javax.swing.JFrame;

//AdminViewStudentResult.java inherits the class from the JFrame STL in order to make it a JFrame Form to create UI
public class AdminViewStudentResult extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, and inheriting the Jframe class.

```
//public constructor of the AdminViewStudentResult class
public AdminViewStudentResult() {
    initComponents();

    //the try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create statement to execute to query

        //executing a query to select all the rows in the student table and displaying them into the studentTable
        ResultSet rs = st.executeQuery("select *from student");
        studentTable.setModel(DbUtils.resultSetToTableModel(rs));
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occors
    catch(Exception e){
        //a message dialog that contains nothing, only for error handling purposes
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- A public constructor of the AdminViewStudentResult class. The try and catch block will be run whenever the constructor of the class is called.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of creating a statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- I selected all the data from the student table and set it to be displayed in the studentTable.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
/**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks the exit button, they will return to the AdminHomePage page
    AdminHomePage.open=0;
    setVisible(false);
}
```

- The setup for the exitButton for when an action is done to the component.
- It will return the open value to 0, therefore, allowing another form to run and it will exit the page display and return to the AdminHomePage page.

```
//FILTER INPUT FIELD EVENT
private void filterInputFieldKeyReleased(java.awt.event.KeyEvent evt) {
    //if the filterInputField is empty, the score variable will be set as 0
    int score;
    if(filterInputField.getText().equals("")){
        score=0;
    }

    //if the filterInputField is not empty, the score variable will take value of the input and change it into int
    else
        score=Integer.parseInt(filterInputField.getText());

    //the try block commands will establish a connection between this file and the mysql database made for Quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //a create statement to declare to the query of the mysql database

        //executing the query to select all data in the row where the score variable is >= the data in the Marks Collumn in the student table
        //then displaying it in the studentTable
        ResultSet rs = st.executeQuery("select *from student where Marks >="+score+"");
        studentTable.setModel(DbUtils.resultSetToTableModel(rs));
    }

    //a catch block to go alongside the try block to catch exception errors
    //will display a new JFrame popup message to indicate an error in the try block.
    catch(Exception e){
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- I created a variable, score, to store the user input from the filterInputField. If the input field is empty, the score variable will be set to 0.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- I selected all the data from the student table and set it to be displayed in the studentTable where the data in the marks column is bigger than or equal to the user input stored in the score variable. So the displayed table only shows data of students that have a score that is equal to or higher than the user input.

- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//STUDENT TABLE EDITABLE SETUP
private void studentTableMouseClicked(java.awt.event.MouseEvent evt) {
    // if the user tries to edit the table in this form, a pop up message dialog will appear
    //stating that they can not do so.
    boolean edits = studentTable.isEditing();
    if(edits==false){
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "You can not edit this table in this form.");
    }
}
```

- Additional setup for the table display. Every time the user clicks on the studentTable, a pop-up message dialog will appear stating that they cannot edit the table directly from the form.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminViewStudentResult().setVisible(true);
        }
    });
}
```

11. AdminManageStudents.java

```
//importing necessary libraries
import java.sql.*;
import SQLManage.Connector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

//AdminManageStudent.java inherits the class from the JFrame STL in order to make it a JFrame Form to create UI
public class AdminManageStudents extends javax.swing.JFrame {

    //public constructor for the AdminManageStudents Class
    public AdminManageStudents() {
        initComponents();
    }

    /**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, and creating a default public constructor for the class.
- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```java
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the exit button is clicked, the attribute open from AdminHomePage.java will be set to 0
    //and the program will return to the AdminHomePage page
    AdminHomePage.open=0;
    setVisible(false);
}
```

- The setup for the exitButton for when an action is done to the component.
- It will return the open value to 0, therefore, allowing another form to run and it will exit the page display and return to the AdminHomePage page.

```java
//DELETE ALL BUTTON EVENT
private void deleteAllButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks the delete all button, a JFrame confirmation popup will appear to ask for user confirmation
    JFrame jf = new JFrame();
    jf.setAlwaysOnTop(true);
    int userConfirmation=JOptionPane.showConfirmDialog(jf, "Are you sure you want to delete ALL STUDENT DATA?", "Select", JOptionPane.YES_NO_OPTION);

    //if the user clicks the yes button
    if(userConfirmation==0)
    {
        //the try block commands will establish a connection between this file and the mysql database
        try
        {
            Connection con= Connector.getCon();

            //declaring the query into the mysql database to delete all rows in the 'student' table
            PreparedStatement ps = con.prepareStatement("delete from student");

            //executing the update in the mysql database
            ps.executeUpdate();

            //once the program is done updating, a Jframe popup message will appear
            JFrame jf1 = new JFrame();
            jf1.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf1, "Successfully Deleted");

            //once the user exits the popup message, AdminManageStudents page will refresh
            setVisible(false);
            new AdminManageStudents().setVisible(true);
        }

        //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs
        catch(Exception e)
        {
            //displays a JFrame that is basically nothing. If the catch block is activated I take it as an indication that there is an error.
            JFrame jf2 = new JFrame();
            jf2.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf2, e);
        }
    }
}
```

- The setup for the deleteAllButton for when an action is done to the component.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- A pop-up confirm dialog will appear to ask for user confirmation upon deletion of all the student table data. If the user clicks on the Yes button, the program will proceed by declaring the prepared statement to delete all the data in all rows in the student table.
- A pop-up message dialog will pop up to declare the successful deletion.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//CLEAR BUTTON EVENT
private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks on the clear button, the program will clear all input fields on the screen.
    studentIDInputField.setText("");
    nameInputField.setText("");
    classInputField.setText("");
    genderInputField.setText("");
    contactNoInputField.setText("");
    emailInputField.setText("");
    addressInputField.setText("");
    scoreInputField.setText("");
    studentIDInputField.setEditable(true);
}
```

- The setup for the clearButton for when an action is done to the component.
- It will set all the input fields to have a blank display when the user clicks on the clear button

```
//SEARCH BUTTON EVENT
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //a variable 'id' to store the user input from the studentIDInputField
    String id = studentIDInputField.getText();

    //the try block commands will establish a connection between this file and the mysql database.
    try
    {
        Connection con = Connector.getCon();
        Statement st= con.createStatement(); //creating a statement to execute in the mysql database

        //executing the query to select the data in the row where the id that the user input matches the id that exists in the StudentID collumn from the student table
        ResultSet rs = st.executeQuery("select *from student where StudentID='"+id+"'");

        //if the mysql terminal comes up with a result
        if(rs.next()){
            //the input fields will be set to display the data from the database from the corresponding collumns.
            nameInputField.setText(rs.getString(1));
            classInputField.setText(rs.getString(2));
            genderInputField.setText(rs.getString(3));
            contactNoInputField.setText(rs.getString(5));
            emailInputField.setText(rs.getString(6));
            addressInputField.setText(rs.getString(7));
            scoreInputField.setText(rs.getString(8));
            studentIDInputField.setEditable(false);
        }

        //if the mysql terminal doesnt come up with a result
        else
        {
            //a JFrame popup message will appear to state that the question ID doesnt exist
            JFrame jf = new JFrame();
            jf.setAlwaysOnTop(true);
            JOptionPane.showMessageDialog(jf, "Question ID not found.");
        }
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error ocurs
    catch(Exception e)
    {
        //displays a jFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the searchButton for when an action is done to the component.
- I created some a variable for storage called id, which stores the user input taken from the studentIDInputField.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a create statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- The create statement will declare the query to select data from the StudentID column where the id that was inputted by the user is equal to the data.
- If it comes up with a result, all the input fields will be set to display the corresponding data from the same row from different columns (name, class, gender, contact no., email, address, marks, and student ID). Along with setting the question id input field editable to false.

- If the search doesn't come up with a result, a pop-up message dialog will appear to state that the student id is invalid.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//DELETE BUTTON EVENT
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // a variable to store the user input from the studentIDInputField
    String studentID = studentIDInputField.getText();

    //the try block commands will establish a connection between this file and the mysql database
    try
    {
        Connection con= Connector.getCon();

        //declaring the query to delete a row from the student table where the studentID matches the user input
        PreparedStatement ps = con.prepareStatement("delete from student where StudentID=?");

        //executing the update in the mysql database
        ps.setString(1, studentID);
        ps.executeUpdate();

        //once the update is complete, a JFrame success popup will appear
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "Successfully Deleted");

        //once the user exits the JFrame popup, the AdminManageStudents page will refresh
        setVisible(false);
        new AdminManageStudents().setVisible(true);
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the deleteButton for when an action is done to the component.
- I created some a variable for storage called studentID, which stores the user input taken from the studentIDInputField.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- The prepared statement will declare the query to select data from the StudentID column where the studentID that was inputted by the user is equal to the data.
- If it comes up with a result, the row containing the corresponding data will be deleted.
- A pop-up message dialog will pop up to declare the successful deletion.
- If the search doesn't come up with a result, a pop-up message dialog will appear to state that the question id is invalid.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```java
//UPDATE BUTTON EVENT
private void updateButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //variables to store user input from each input field
    String studentID = studentIDInputField.getText();
    String name = nameInputField.getText();
    String classInput = classInputField.getText();
    String gender = genderInputField.getText();
    String contactNo = contactNoInputField.getText();
    String email = emailInputField.getText();
    String address = addressInputField.getText();
    int score = Integer.parseInt(scoreInputField.getText()); // turning the string to integer for the quiz score

    //the try block commands will establish a connection between this file and the mysql database
    try
    {
        Connection con= Connector.getCon();

        //declaring the query into the mysql database to update the information of the matched student ID row in the student table
        PreparedStatement ps = con.prepareStatement("update student set Name=?, Class=?, Gender=?, ContactNo=?, Email=?, Address=?, Marks=?, StudentID=? where StudentID="+studentID+"");
        ps.setString(1, name);
        ps.setString(2, classInput);
        ps.setString(3, gender);
        ps.setString(4, contactNo);
        ps.setString(5, email);
        ps.setString(6, address);
        ps.setInt(7, score);
        ps.setString(8, studentID);

        //executing the update. After the update a popup JFrame will appear to show the success message.
        ps.executeUpdate();
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, "Successfully Updated");

        //once the user exits the popup JFrame, the AdminManageStudents page will refresh.
        setVisible(false);
        new AdminManageStudents().setVisible(true);
    }

    //a catch block to go alongside the try block if the try block fails which catches an exception if an error occurs
    catch(Exception e)
    {
        //displays a JFrame that is basically nothing. If the catch block is activated, I take it as an indication that there is an error
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(jf, e);
    }
}
```

- The setup for the updateButton for when an action is done with the component.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- The prepared statement will declare the query to update the data stored in individual columns where the id number matches the user input of the studentID.
- A pop-up message dialog will appear to notify the user of the successful update. And once the user exits the message dialog, the page will refresh.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```java
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AdminManageStudents().setVisible(true);
        }
    });
}
```

12. StudentLogin.java

```java
//importing necessary libraries
import java.text.SimpleDateFormat;
import java.sql.*;
import java.util.Date;
import SQLManage.Connector;
import javax.swing.JOptionPane;

//StudentLogin.java inherits the class from the JFrame STL in order to create the UI
public class StudentLogin extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, and inheriting the Jframe class.

```
//public constructor for StudentLogin class
public StudentLogin() {
    initComponents();

    //setting the text area to not be editable
    quizzenTextArea.setEditable(false);

    //creating a automated date generator and displaying it in a Jframe label
    SimpleDateFormat dFormat = new SimpleDateFormat("dd-MM-yyyy");
    Date date = new Date();
    dateLabel.setText(dFormat.format(date));

}
```

- A public constructor of the StudentLogin class. Includes the object instantiation for the dateLabel to generate real-time date display.

```
/**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//RETURN BUTTON EVENT
private void returnButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks on the return button, they will return to the Index page (the initial starting page)
    setVisible(false);
    new Index().setVisible(true);
}
```

- The setup for the returnToIndexButton for when an action is done with the component.
- For the returnToIndexButtonActionPerformed(evt) function, once the user clicks on the return button, the program will exit the display of the StudentLogin page and return to the Index page display.

```java
//SAVE NEXT BUTTON EVENT
private void saveNextButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // variables to store user inputs from the input fields
    String name = nameInputField.getText();
    String classInput = classInputField.getText();
    String gender = (String)genderInput.getSelectedItem();
    String studentID = studentIDNoInputField.getText();
    String contactNo = contactNoInputField.getText();
    String email = emailInputField.getText();
    String address = addressInputField.getText();
    String marks = "0";

    //the try block commands will establish a connection between this file and the mysql database made for quizzen
    try
    {
        Connection con = Connector.getCon();

        // a prepared statement with the query to insert the values of the user input into the corresponding collumns in the student table
        PreparedStatement ps = con.prepareStatement("insert into student values(?,?,?,?,?,?,?,?)");
        ps.setString(1, name);
        ps.setString(2, classInput);
        ps.setString(3, gender);
        ps.setString(4, studentID);
        ps.setString(5, contactNo);
        ps.setString(6, email);
        ps.setString(7, address);
        ps.setString(8, marks);

        //executing the update and starting the StudentInstructions.java file while passing the studentID variable value as the parameter
        ps.executeUpdate();
        setVisible(false);
        new StudentInstructions(studentID).setVisible(true);
    }

    //a catch block to go alongside the try block to indicate any errors that might occur from the try block commands
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- The setup for the saveNextButton for when an action is done with the component.
- I created some variables for storage. I stored all the user inputs from each InputField available into the variables.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- The prepared statement will declare the query to store all the data by inserting them as a whole new row and organizing the data according to the columns. So when the user is done inputting all the text fields of the student data form and they press the save next button, the student data will be added to the table in the database.
- A pop-up message dialog will appear to notify the user of the successful update. And once the user exits the message dialog, the page will refresh.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```java
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new StudentLogin().setVisible(true);
        }
    });
}
```

13. StudentInstructions.java

```
//StudentInstructions.java inherits the class from the JFrame STL in order to create a UI
public class StudentInstructions extends javax.swing.JFrame {
```

- Class initialization. Creating the class, and inheriting the Jframe class.

```
//a public attribute/variable to store the student ID
public String studentID;

//public default constructor for StudentsInstructions class
public StudentInstructions() {
    initComponents();
}

//overloading the constructor for StudentsInstructions class by adding a parameter that is studentIDNo1
public StudentInstructions(String studentIDNo1) {
    initComponents();
    //this constructor is mainly to pass the value of the student ID from the StudentLogin page to the next page (StudentQuiz.java)
    instructionText.setEditable(false);
    studentID = studentIDNo1;
}
```

- I created a public string attribute named studentID, will be used as a parameter to the constructor.
- I have overloaded the default constructor for the StudentInstructions class, one that accepts the parameter/argument of studentIDNo1. I also declared that the instructionTextBox be not editable.

```
/**NETBEANS GENERATED CODE FROM THE FRONT END DESIGN CONSOLE
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code
```

- Front end design setup and code customization to set up the GUI. Code is generated by Netbeans and designed by me with the Netbeans design console.

```
//START QUIZ BUTTON EVENT
private void startQuizButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // a button that runs the StudentQuiz.java file and exits the StudentInstructions.java file
    setVisible(false);
    new StudentQuiz(studentID).setVisible(true);
}
```

- The setup for the startQuizButton for when an action is done with the component.
- Once the user clicks on the start button, the program will exit the display of the StudentInstructions page and move on to the StudentQuiz page display.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new StudentInstructions().setVisible(true);
        }
    });
}
```

14. StudentQuiz.java

```
//importing necessary libraries
import SQLManage.Connector;
import java.sql.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Date;
import javax.swing.Timer;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import java.text.SimpleDateFormat;

//the StudentQuiz class inherits the class from the JFrame STL to create UI
public class StudentQuiz extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, also creating a public default constructor.
- I declared several public variables that will be used in some of the functions in this class.

```
//public attributes/variables declaration
public String questionID="1";
public int questionIDtemp;
public String answer;
public int minutes=0;
public int seconds=0;
public int marks=0;
Timer timer;
public String studentAnswer="";
public String studentID;
```

```
//constructor
public StudentQuiz() {
    initComponents();
}
```

```
public StudentQuiz(String studentID) {
    initComponents();
    studentIDLabel.setText(studentID);
    //date setup
    SimpleDateFormat dFormat = new SimpleDateFormat("dd-MM-yyyy");
    Date date = new Date();
    dateLabel.setText(dFormat.format(date));

    //first question and student details
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select *from student where StudentID ='"+studentID+"'");

        //displaying student name in the data panel
        while(rs.next())
        {
            nameLabel.setText(rs.getString(1));
        }

        //displaying the question no in the data panel, question display and option display
        ResultSet rs1 = st.executeQuery("select *from question where id='"+questionID+"'");
        while(rs1.next())
        {
            questionNoLabel.setText(rs1.getString(1));
            questionLabel.setText(rs1.getString(2));
            opt1Button.setText(rs1.getString(3));
            opt2Button.setText(rs1.getString(4));
            opt3Button.setText(rs1.getString(5));
            opt4Button.setText(rs1.getString(6));
            answer = rs1.getString(7);
        }

        //displaying the total no. of questions in the data panel
        PreparedStatement ps = con.prepareStatement("select count(*) from question");
        ResultSet rs2 = ps.executeQuery();
        while(rs2.next()){
            String questionTotal = rs2.getString("count(*)");
            totalQuestionLabel.setText(questionTotal);
        }
    }

    //a catch block to go alongside the try block to catch any errors that may occur from the try block
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e);
    }
```

- Overloading the public constructor for the StudentQuiz class to take a parameter/argument of studentID.
- The setup to display the date on the dateLabel.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a prepared statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table and the question table.
- This connection is to display the student name and ID number, the question and the options, and also the question number.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//ANSWER CHECK FUNCTION
public void answerCheck(){
    //Synching student input on the radio buttons with the studentAnswer variable
    if(opt1Button.isSelected()){
        studentAnswer = opt1Button.getText();
    }
    else if(opt2Button.isSelected()){
        studentAnswer = opt2Button.getText();
    }
    else if(opt3Button.isSelected()){
        studentAnswer = opt3Button.getText();
    }
    else if(opt4Button.isSelected()){
        studentAnswer = opt4Button.getText();
    }
    else
    {
      studentAnswer = "";
    }

    //checking if the answer of the student is the correct answer
    //if it matches the correct answer, the student's marks will increment by 1
    if(studentAnswer.equalsIgnoreCase(answer))
    {
        marks = marks + 1;
    }

    //question number update
    //incrementing the question ID whenever the answerCheck() function is called
    questionIDtemp = Integer.parseInt(questionID);
    questionIDtemp = questionIDtemp + 1;
    questionID = String.valueOf(questionIDtemp);

    //clear radio button inputs after the answer is checked and the question ID is incremented
    opt1Button.setSelected(false);
    opt2Button.setSelected(false);
    opt3Button.setSelected(false);
    opt4Button.setSelected(false);

    //If the questionID/question number is on last number, hides the next question button will be hidden
    if(questionID.equals("10")){
        nextButton.setVisible(false);
    }
}
```

- The setup for the answerCheck() function. Will be used to check the student answer and traverse through the progress of the quiz.
- This function will start by setting the selected option from the user input as the current user answer. Then if the selected answer of the user is equal to the answer declared by the admin for each of the question, the user score will increment by 1.
- After checking the user answer selection, the questionID display will increment as well. And clears the selection of the radio buttons.
- Lastly, if the questionID is equal to 10, the next button will disappear because that is the last number for the quiz.

```
//QUESTION FUNCTION
public void question(){

    //a try block to create connections between this file and the mysql database for quizzen
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create statement to be executed as a query in the mysql database

        //executing the query to select the row of data where the id matches with the questionID variable in the question table
        ResultSet rs = st.executeQuery("select *from question where id='"+questionID+"'");

        //while the mysql database returns a result
        while(rs.next())
        {
            //the questions will be displayed alongside the options and question number
            questionNoLabel.setText(rs.getString(1));
            questionLabel.setText(rs.getString(2));
            opt1Button.setText(rs.getString(3));
            opt2Button.setText(rs.getString(4));
            opt3Button.setText(rs.getString(5));
            opt4Button.setText(rs.getString(6));

            //the variable answer will store the correct answer based on the database to compare with the student's answer
            //in the function answerCheck()
            answer = rs.getString(7);
        }
    }

    //a catch block to go alongside the try block to catch any errors that may occur from the try block
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- The setup for the question() function.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a create statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the question table.
- The create statement will declare the query to select the data stored in individual columns where the id number matches the user input of the question id. The questionNoLabel, questionLabel, and all the option radio buttons, will be set to display the corresponding data from the table. And the variable answer, will be set to the data in the corresponding answer column.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//SUBMIT FUNCTION
public void submit(){
    //a variable to store the displayed studentID
    studentID = studentIDLabel.getText();

    //calling the answerCheck() function
    answerCheck();

    //the try block commands will create a connection between this file and the mysql database for quizzen
    try{
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create a statement to execute query into the mysql database

        //executing the update with the query to update the student's Marks/score according to the student ID
        st.executeUpdate("update student set Marks='"+marks+"'where StudentID = '"+studentID+"'");

        //once the update is done, the timer ends and the StudentSubmitted file starts running
        setVisible(false);
        new StudentSubmitted(studentID,Integer.toString(marks)).setVisible(true);
        timer.stop();
    }

    //a catch block to go alongside the try block to catch any errors that may occur from the try block
    catch(Exception e){
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- The setup for the question() function.
- The try block section of the code creates a connection between the file and the MySQL database. This is the standard mechanism of initializing a create statement to be entered into the database as a query to access the needed data and table. In this case, I accessed the student table.
- The create statement will declare the query to select the data stored in individual columns where the StudentID number matches the user input of the student id. The submit function will call on the answerCheck() function and after it runs, will update the student data in the table on the Marks column to display the total score of the student.
- The catch block is created to catch any errors that might occur while running the try block. Useful for error handling.

```
//TIMER OBJECT DECLARATION/INSTANTIATION
timer = new Timer(1000, new ActionListener(){//1000 is 1 second 1 = 1 milisecond / program updates the program every 1 second
    @Override
    public void actionPerformed(ActionEvent e){ //what happens when the timer updates

        //displaying the timer labels on the right area, seconds and minutes
        timerSecLabel.setText(String.valueOf(seconds));
        timerMinLabel.setText(String.valueOf(minutes));

        //if the seconds has reach the value of 60, the value will reset to 0 and the minutes variable will be incremented by 1
        if(seconds == 60){
            seconds = 0;
            minutes++;

            //if the timer reaches 10 minutes, then the timer will stop immediately and submit as it is the maximum time of the quiz
            if(minutes == 10){
                timer.stop();
                answerCheck();
                submit();
            }
        }
        //incrementing the seconds variable by 1
        seconds++;
    }
});
timer.start();
```

- The setup of the timer display. For every 60 seconds, a minute will be incremented and the seconds variable will reset to 0.

- If the timer reaches 10 minutes, the quiz will automatically be submitted as it is the maximum amount of time for the students to take the quiz.

```
//NEXT BUTTON EVENT
private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the user clicks on the next button, the answerCheck() and question() functions will be called
    answerCheck();
    question();
}
```

- If the user clicks on the nextButton, the answerCheck() and question() functions will be called.

```
//SUBMIT BUTTON EVENT
private void submitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the student has not reach the total number of questions
    // a confirmation popup message will appear to ask for confirmation as the quiz has not been finished yet
    if(Integer.parseInt(questionID)!=10){
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        int userConfirmation = JOptionPane.showConfirmDialog(jf, "Are you sure you want to submit? You have not finished the quiz.", "Select", JOptionPane.YES_NO_OPTION);
        if(userConfirmation==0){
            answerCheck();
            submit();
        }
    }

    //else the confirmation popup message will appear to ask for confirmation to submit the answers
    else{
        JFrame jf = new JFrame();
        jf.setAlwaysOnTop(true);
        int userConfirmation = JOptionPane.showConfirmDialog(jf, "Are you sure you want to submit?", "Select", JOptionPane.YES_NO_OPTION);
        if(userConfirmation==0){
            answerCheck();
            submit();
        }
    }
```

- The submitButton setup. Includes pop-up confirmation dialogs to ask for user confirmation whether or not they are sure to submit.

```
//RADIO BUTTON SYNCHING
private void opt1ButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the option 1 button is selected, the rest of the button will be deselected
    if(opt1Button.isSelected()){
        opt2Button.setSelected(false);
        opt3Button.setSelected(false);
        opt4Button.setSelected(false);
    }
}
```

```
//RADIO BUTTON SYNCHING
private void opt2ButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the option 2 button is selected, the rest of the button will be deselected
    if(opt2Button.isSelected()){
        opt1Button.setSelected(false);
        opt3Button.setSelected(false);
        opt4Button.setSelected(false);
    }
}
```

```java
//RADIO BUTTON SYNCHING
private void opt3ButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the option 3 button is selected, the rest of the button will be deselected
    if(opt3Button.isSelected()){
        opt2Button.setSelected(false);
        opt1Button.setSelected(false);
        opt4Button.setSelected(false);
    }
}
```

```java
//RADIO BUTTON SYNCHING
private void opt4ButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //if the option 2 button is selected, the rest of the button will be deselected
    if(opt4Button.isSelected()){
        opt2Button.setSelected(false);
        opt3Button.setSelected(false);
        opt1Button.setSelected(false);
    }
}
```

- Radio button sync. Only allows one radio button to be selected.
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```java
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new StudentQuiz().setVisible(true);
        }
    });
}
```

15. StudentSubmitted.java

```java
//importing necessary libraries
import SQLManage.Connector;
import java.sql.*;
import javax.swing.JOptionPane;

//StudentSubmitted class inherits the class from the JFrame STL to create UI
public class StudentSubmitted extends javax.swing.JFrame {
```

- Class initialization. Importing libraries, creating the class, inheriting the Jframe class, also creating a public default constructor.

```
//public default constructor for the StudentSubmitted class
public StudentSubmitted() {
    initComponents();
}

//overloading the constructor for StudentSubmitted class taking in the parameter of studentID and score
public StudentSubmitted(String studentID,String score) {
    initComponents();

    //displaying the quizscore and the student ID
    quizScoreLabel.setText(score);
    studentIDLabel.setText(studentID);

    //the try block will create a connection between the mysql database and this file
    try
    {
        Connection con = Connector.getCon();
        Statement st = con.createStatement(); //create statement to execute query in the mysql database

        //executing query to select all the data in the row where the student id matches
        ResultSet rs = st.executeQuery("select *from student where StudentID ='"+studentID+"'");

        //displaying student name in the data panel
        while(rs.next())
        {
            nameLabel.setText(rs.getString(1));
        }
    }

    //a catch block to go alongside the try block to catch any errors that may occur from the try block
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

- Default public constructor and overloading for the constructor. The try block works on creating a connection between the file and the sql file. To display the nameLabel with the data of the student table.

```
//EXIT BUTTON EVENT
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // if the user clicks the exit button, they will return to Index.java
    setVisible(false);
    new Index().setVisible(true);
}
```

- exitButton setup where if the user clicks on the exitButton, they will return to the Index.java file display
- run() is a file run command that is generated by Netbeans, will be called every time we run the file/ call the class constructor.

```
//RUN COMMAND GENERATED BY NETBEANS
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new StudentSubmitted().setVisible(true);
        }
    });
}
```

References :

- geeksforgeeks.com
- Stackoverflow.com
- programiz.com
- canva.com
- https://www.youtube.com/BTechDays

Program Files :

- **Video :** https://drive.google.com/file/d/1kQNC1ELN08ArUR1GFuksaZQXCeflIIGD/view?usp=sharing
- **Design Canva File :** https://www.canva.com/design/DAFC4A2AoQI/iQIXK-f7XBre1d3FLpbsGA/view?utm_content=DAFC4A2AoQI&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
- **Github Repo :** https://github.com/mariaclarin/FinalProject-OOP-2501990331-MariaClarin