Development

Introduction to Techniques Used

B**asic techniques**: Loops, conditional statements, and data validation.
Advanced techniques: 1. Error handling 2. Parallel array 3. External JAR 4. OOP features: a. Inheritance b. Encapsulation c. Modularity 5. SQLite connection 6. GUI development 7. Dynamic data structure

1. Error Handling

The program implements several methods to effectively manage errors, ensuring it remains operational under various circumstances. By identifying and addressing errors as they occur, the system maintains its functionality and prevents disruptions. Error management is seamlessly integrated into different parts of the program to ensure smooth execution. Techniques such as try-catch blocks**,** exception handling, and user-friendly error messages are utilized to detect, handle, and recover from potential issues.

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. (Gillis, 2022), The exception handling is a part of the try and catch method where the function will output the errors and their cause when it runs. The program runs the functions and if an error occurs it will catch it in the catch block. The exception handling will throw the exception explicitly

In the program, exceptions are also utilized in database-related functions. These exceptions ensure the program remains operational even when issues occur during database access. For instance, SQLException is handled within the database class, as demonstrated in Figures 1 and 2.

```
        JOptionPane.showMessageDialog(parentComponent:null, message:"Account created successfully!");
        jTabbedPane1.setSelectedIndex(index: 6);

        // Clear inputs from the text fields
        jTextField2.setText(t: "");
        jTextField8.setText(t: "");
        jTextField10.setText(t: "");
        jTextField9.setText(t: "");
        jPasswordField1.setText(t: "");
        jPasswordField2.setText(t: "");
        jComboBoxRole2.setSelectedIndex(anIndex:0);

    } catch (SQLException e) {
        /
        JOptionPane.showMessageDialog(parentComponent:null, "Error: " + e.getMessage());
    }
```

Figure 1; An example of exception handling within try catch method of a class database

```
public class DBConnector {
    public static Connection getConnection() {
        Connection con = null;
        try {
            Class.forName(className: "com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection(
                url:"jdbc:mysql://127.0.0.1:3306/project",
                user: "root",
                password: "presidentofTz1!"
            );
            System.out.println(x: "Connection established successfully.");
        } catch (ClassNotFoundException e) {
            System.out.println(x: "MySQL JDBC Driver Not Found");
            e.printStackTrace();
        } catch (Exception e) {
            System.out.println(x: "Could not connect to MySQL");
            e.printStackTrace();
        }
        return con;
```

Figure 2; example of an SQL exception when connecting to the database.

The presence of errors can also be dealt with by displaying error messages. Such types of messages are invoked during specific conditions observable in the system, for example, invalid user input. In such cases, the most effective and powerful mechanism that can be used is `JOptionPane.showMessageDialog` method calls. This attracts the attention of users and provides immediate visual feedback that becomes more pleasing and user-friendly. For example, the message that makes a user aware of incorrect data entry during the sign-up process as illustrated in Figure 1 will guide such a user to correct his input accordingly. The same error message can be seen here below if a user logs in with incorrect details in the database.

```
jLogin.addActionListener(e -> {
    String email = jtxtusername.getText().trim();  // Retrieve email from the text field
    String password = new String(value: jtxtpassword.getPassword());  // Retrieve password from the password field

    if (email.isEmpty() || password.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter both email and password!", title: "Input Error",
                messageType:JOptionPane.WARNING_MESSAGE);
        return;
    }
```

Figure 3; Joption pane showing an output message being produced if not retrieved from the database **2.**

Parallel Arrays

Parallel arrays are two or more arrays that are logically related, with corresponding elements sharing the same index. In this program, parallel arrays are used to manage and organize teacher feedback data. Each feedback attribute—coverage, moreinfo, howprepared, howwell, and how_enjoy—is stored in parallel arrays, ensuring a consistent relationship between the feedback categories. They are also used in the student feedback, teacher details and many other places in the program.

For example, the new feedback entered into the teacherfeedback table would contain value corresponding to attribute. Such link-up is very important because it keeps the data clean and will reduce the work for operations such as showing or analyzing feedback in the future.

The following example is a case of how feedback values are input into the database.

```
String sql = "INSERT INTO teacherfeedback (coverage, moreinfo, howprepared, howwell, how_enjoy) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DBConnector.getConnection();
     PreparedStatement pst = conn.prepareStatement(string: sql)) {

    // Setting the values in the PreparedStatement
    pst.setString(i: 1, string: coverage);
    pst.setString(i: 2, string: moreInfo);
```

Figure 4; example of a parallel array used to store feedback in the database.

This SQL statement maps each feedback attribute to a corresponding value, ensuring that all data fields are populated in parallel. By maintaining this structured approach, the program can easily retrieve, display, and manipulate feedback data, making it integral to its overall functionality.

3. External JAR File

The program relies on several external JAR files to enable critical functionalities, including database connectivity and email communication. These JAR files, as shown in the figure 5 below are imported into the project to provide the necessary libraries for specific operations.

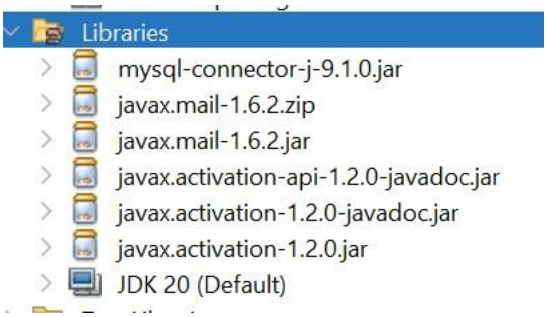


Figure 5; a list of all the jar files insatalled.

a. mysql-connector-j-9.1.0.jar:

This JAR file allows the program to establish a connection with the MySQL database. It facilitates SQL operations such as inserting, retrieving, and updating data in the teacherfeedback table. For example, it enables the program to execute SQL queries like:

```
try {
    // Get the values from the sliders and text field
    int howClearly = jSlider7.getValue();  // Slider for how clearly
    int howEngaging = jSlider5.getValue(); // Slider for how engaging
    int howComfy = jSlider4.getValue();    // Slider for how comfy
    int howPaced = jSlider8.getValue();    // Slider for how paced
    int rating = jSlider6.getValue();      // Slider for rating
    String additionalComments = jTextField5.getText(); // Additional comments text field

    // SQL query to insert the data into the studentfeedback table
    String sql = "INSERT INTO studentfeedback (howclearly, howengaging, howcomfy, howpaced, rating, addcom) "
               + "VALUES (?, ?, ?, ?, ?, ?)";

    // Database connection
    try (Connection conn = DBConnector.getConnection();
         PreparedStatement pst = conn.prepareStatement(string: sql)) {
```

Figure 6; - Code that uses the JAR file for the connection to the database. The code's function is to add a User to the table in the database.

b. javax.mail-1.6.2.jar, javax.mail-1.6.2.zip:

These files are essential for implementing email functionality in the program. The javax.mail library enables the program to send automated emails, which can be useful for notifying users or clients.

```
public void sendEmail(String to, String subject, String body) throws MessagingException {
    final String from = "headofschoolhos@gmail.com";
    final String password = "rzhq ueaw gwhr lvez";

    // Set up mail server properties
    Properties props = new Properties();
    props.put(key:"mail.smtp.auth", value: "true");
    props.put(key:"mail.smtp.starttls.enable", value: "true");
    props.put(key:"mail.smtp.host", value: "smtp.gmail.com");
    props.put(key:"mail.smtp.port", value: "587");

    // Create a session to authenticate with the mail server
    Session session = Session.getInstance(props, new Authenticator() {
        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(userName: from, password);
        }
    });
```

Figure 7; - This code defines a method named sendEmail that is responsible for sending an email using Java's JavaMail JAR file.

c. javax.activation-1.2.0.jar and related files:

These JAR files are for activating the handling of data concerning email attachments and MIME types when sending emails. These maybe related under an integrated level with the javax.mail library.

Here, these files were imported in the program under the Libraries section of the project dependencies (as shown in figure 8). Without these external JAR files, the program would not connect to a MySQL database and hence lend itself to send emails, which are also core functions of the total program.

The modular approach to external library addition ensures the program is efficient else horizontal against scalability to very high loads for the program and heavy tasks.

4. OOP Features:

The program has used some of the OOP concepts which will be stated in Criteria A, for making the programs more functional and structured. The features of OOP like inheritance, encapsulation, and modularization, which could be highly beneficial to such programming applications, are some of the things that would improve the GUI as well as the interaction and communication between classes.

i. Modularity:

The program is divided into some classes, each of them having a role towards better organization and readability. Modularizing this code would made the program more manageable, coherent, and easy to maintain throughout development. It also facilitates smooth communication of these classes.

The arrangement of these classes within the source package is given in Figure 8 showing the logical organization and separation of concerns in the program.

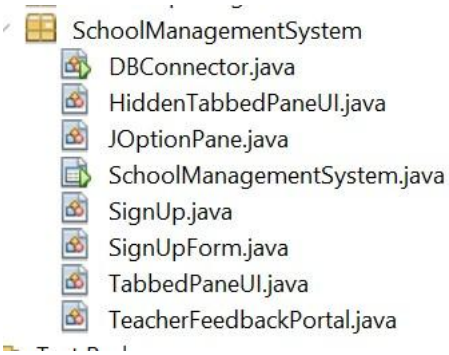


Figure 8; - classes in the source package

ii. Inheritance:

In this program, inheritance helps minimize redundant code and optimize efficiency. By extending javax.swing.JFrame in the SchoolManagementSystem class, the system inherits essential graphical interface functionality without re-implementing it. This approach allows the sharing of common features across classes, ensuring a cleaner structure and more manageable code. Inheritance enables features to be centralized, making updates and maintenance quicker and less error-prone (Figure 9).

```
public class SchoolManagementSystem extends javax.swing.JFrame {

    public SchoolManagementSystem() {

        initComponents();
    }
```

Figure 9; inheritance in the main class

iii. **Encapsulation** ;

Encapsulation hides variables and methods to protect data integrity and prevent unauthorized access. It enables read-only or write-only variables and improves debugging by separating concerns. In this program, feedback values like howPrepared, howWell, and howEnjoy are kept private and accessed through controlled methods, as shown in Figure 10.

```
});

public class Studentfeedback {

    private int howPrepared;
    private int howWell;
    private int howEnjoy;
```

Figure 10; Private variables in student feedback class

5. SQL Database

The program uses a MySQL database to manage essential data. The database contains multiple tables, each serving a specific purpose. The relationships among these tables enable the system to efficiently retrieve, store, and display relevant data. For instance, connections between the teacherfeedback and teacherdetails tables allow users to view feedback tied to specific teachers. The database structure and relationships are displayed in **Figure 11**.

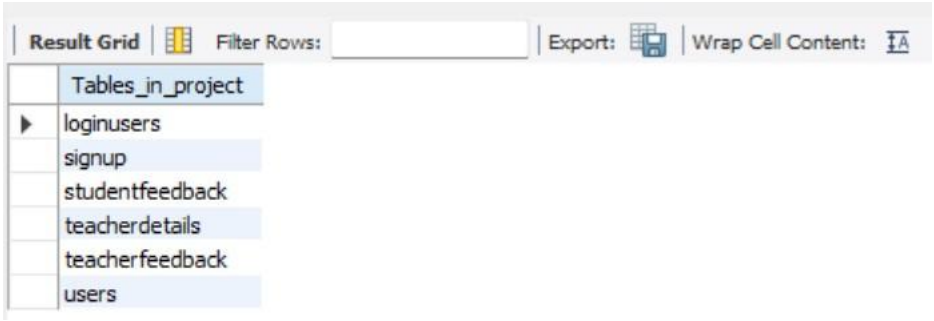


Figure 11; Example of all the tables in the database

The program's class Database creates the link between the database's tables and the program. A number of other functions included in the Database class also depend on the connection in order to function. Figure 12 provides an example of a connection using the Connect function in the Database class.

```
public class DBConnector {
    public static Connection getConnection() {
        Connection con = null;
        try {
            Class.forName(className: "com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection(
                url:"jdbc:mysql://127.0.0.1:3306/project",
                user:"root",
                password:"presidentofTz1!"
            );
            System.out.println(x: "Connection established successfully.");
        } catch (ClassNotFoundException e) {
            System.out.println(x: "MySQL JDBC Driver Not Found");
            e.printStackTrace();
        } catch (Exception e) {
            System.out.println(x: "Could not connect to MySQL");
            e.printStackTrace();
        }
        return con;
    }
```

Figure 12;Class databse function connect

```java
public static void main(String[] args) {
    Connection conn = DBConnector.getConnection();
    if (conn == null) {
        System.out.println(x: "Failed to establish connection.");
        return;
    }

    try {
        PreparedStatement stmt = conn.prepareStatement(string: "SELECT * FROM users;");
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            System.out.println("ID: " + rs.getInt(string: "id"));
            System.out.println("Username: " + rs.getString(string: "username"));
            System.out.println("Email: " + rs.getString(string: "email"));
        }
    } catch (Exception e) {
        System.out.println(x: "Error executing query");
        e.printStackTrace();
    }
}
```

Figure 13; Class database function connect

Several concepts that are used in the database connection code help to establish an interaction between the database and the program:

1. **Prepared statement**: is an object used in Java database connectivity (JDBC) that represents a precompiled SQL statement. This means that the SQL statement is parsed and compiled by the database server beforehand, which can then be executed multiple times with different parameters. (Khorwal, 2024)

2. **Result set** - is a Java object that contains the results of executing an SQL query. In other words, it contains the rows that satisfy the conditions of the query. The data stored in a ResultSet object is retrieved through a set of get methods that allows access to the various columns of the current row (Wesley, 1999).

3. **Queries**: Queries retrieve or manipulate data during execution. For example, in this program, queries fetch teacher details from teacherdetails or handle feedback data in teacherfeedback to ensure accurate and relevant operations.

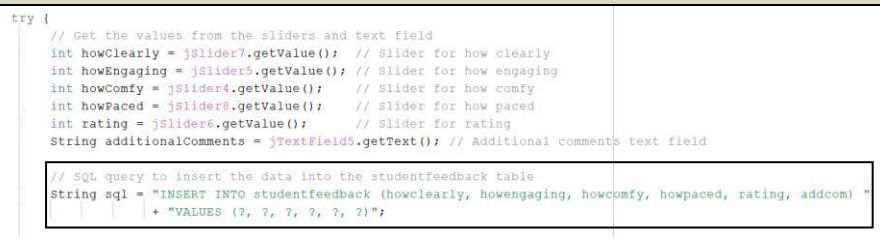
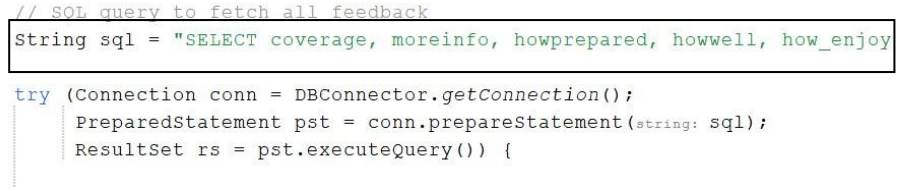
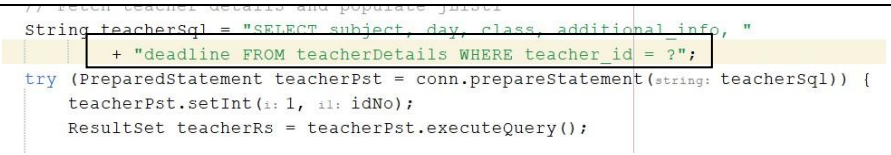
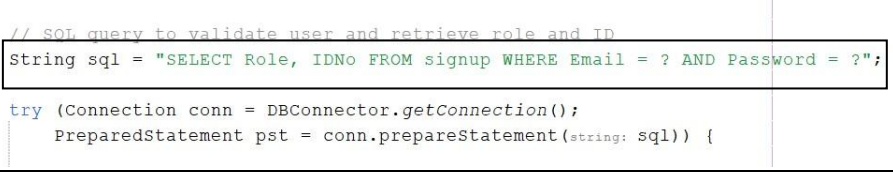
```
private void updateJList3(Connection conn) {
    try {
        // SQL query to get all the feedback from the table
        String sql = "SELECT * FROM studentfeedback";

        // Prepare statement
        try (PreparedStatement pst = conn.prepareStatement(string: sql);
             ResultSet rs = pst.executeQuery()) {
```

Figure 14; Class Database, function update list that uses these concepts.

Command statements in the database class:

In the database class there are several unique statements and operations that makes action from the database easier to operate.

Command/ Operations	Purpose	Picture of code
INSERT	Insert data into a table in the database.	 *Figure 15;- Example of INSERT command in Database class*
SELECT	Extract data from the database	 *Figure 16; Example of SELECT command in the Database class*
WHERE	Used to output a specific data	 *Figure 17; Example of WHERE command in Database class*
FROM	select the database tables	

```
try {
    // Get the values from the sliders and text field
    int howClearly = jSlider7.getValue();  // Slider for how clearly
    int howEngaging = jSlider5.getValue(); // Slider for how engaging
    int howComfy = jSlider4.getValue();    // Slider for how comfy
    int howPaced = jSlider8.getValue();    // Slider for how paced
    int rating = jSlider6.getValue();      // Slider for rating
    String additionalComments = jTextField5.getText(); // Additional comments text field

    // SQL query to insert the data into the studentfeedback table
    String sql = "INSERT INTO studentfeedback (howclearly, howengaging, howcomfy, howpaced, rating, addcom) "
               + "VALUES (?, ?, ?, ?, ?, ?)";
```

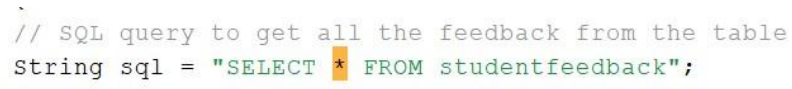
```
// SQL query to fetch all feedback
String sql = "SELECT coverage, moreinfo, howprepared, howwell, how_enjoy

try (Connection conn = DBConnector.getConnection();
     PreparedStatement pst = conn.prepareStatement(string: sql);
     ResultSet rs = pst.executeQuery()) {
```

```
String teacherSql = "SELECT subject, day, class, additional_info, "
        + "deadline FROM teacherDetails WHERE teacher_id = ?";
try (PreparedStatement teacherPst = conn.prepareStatement(string: teacherSql)) {
    teacherPst.setInt(i: 1, i1: idNo);
    ResultSet teacherRs = teacherPst.executeQuery();
```

```
// SQL query to validate user and retrieve role and ID
String sql = "SELECT Role, IDNo FROM signup WHERE Email = ? AND Password = ?";

try (Connection conn = DBConnector.getConnection();
     PreparedStatement pst = conn.prepareStatement(string: sql)) {
```

		Figure 18; Example of FROM operation in Database class
*	Goes with SELECT. It means all columns and rows	 *Figure 19; Example of * operation in Database class*

```
// SQL query to get all the feedback from the table
String sql = "SELECT * FROM studentfeedback";
```

6. GUI Development:

The program uses Java Swing to create a user-friendly interface. Features like buttons, text boxes, and dropdowns enhance usability, making the system approachable for students and teachers.

```
// Variables declaration - do not modify
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.ButtonGroup buttonGroup2;
private javax.swing.ButtonGroup buttonGroup3;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton10;
private javax.swing.JButton jButton12;
private javax.swing.JButton jButton13;
private javax.swing.JButton jButton14;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
```

Figure 20; - Variables GUI features in the Login page class

7. Dynamic data structures

Dynamic data structure is a data structure in the form of a list. A dynamic data structure allows elements to be listed without a size limit. The benefit of this method is the optimization of memory space. The list can increase according to the number of elements that are added and not due to prior setting (W3Schools, 2019).

In this program, a DefaultListModel is used to dynamically populate and display teacher details retrieved from the database. The SQL query fetches information such as subject, day, class, additional_info, and deadline from the teacherdetails table. The data is added to the DefaultListModel, which dynamically updates jList1 to display the retrieved teacher details.

This approach ensures flexibility, as the DefaultListModel can adapt to any number of records without a predefined size, similar to an ArrayList. This dynamic structure allows the program to handle increasing or varying amounts of data over time efficiently, making it scalable and user-friendly.

```
jTabbedPane1.setSelectedIndex(index: 4);
// Fetch teacher details and populate jList1
String teacherSql = "SELECT subject, day, class, additional_info, "
        + "deadline FROM teacherDetails WHERE teacher_id = ?";
try (PreparedStatement teacherPst = conn.prepareStatement(string: teacherSql)) {
    teacherPst.setInt(i: 1, i1: idNo);
    ResultSet teacherRs = teacherPst.executeQuery();

    DefaultListModel<String> listModel = new DefaultListModel<>();
    if (teacherRs.isBeforeFirst()) {  / Check if results exist
        while (teacherRs.next()) {
            String subject = teacherRs.getString(string: "subject");
            String day = teacherRs.getString(string: "day");
            String className = teacherRs.getString(string: "class");
            String additionalInfo = teacherRs.getString(string: "additional_info");
            String deadline = teacherRs.getString(string: "deadline");
            //
```

Figure 21; - Creating ArrayLists.

Word Count ; 1450

References

Gillis, A. S. (2022, June). *definition of exeption handling*. Retrieved from Tech Target: https://www.techtarget.m/searchsoftwarequality/definition/error-handling

Khorwal, R. (2024, Aug 31). *code 360*. Retrieved from naukri.com: https://www.naukri.com/code360/library/prepared-statement-java

W3Schools. (2019). *Dynamic Data Structures* . Retrieved from w3schools: https://www.w3schools.com/vue/vue_dynamic-components.php

Wesley, A. (1999). *5.1 ResultSet Overview*. Retrieved from https://docs.oracle.com/javase/1.5.0/docs/guide/jdbc/getstart/resultset.html#:~:text=A%20ResultSet%20is%20a%20Java,columns%20of%20the%20current%20row.