Criterion C: Development

[x] Appendix (meeting) index.

Introduction:

To complete this project, the Java programming language and IDE Netbeans-14 have been chosen. The NetBeans-14 environment was preferred because of its refactoring abilities, flexibility, and fully featured GUI builder. Due to the special requirements of the success criteria, a range of external libraries have been implemented [C.1] and an additional programming language has been used (SQLite). The most relevant list of techniques used is presented below:

Advanced techniques	1. OOP (Modularity, Polymorphism, Encapsulation, Inclusion)
	2. GUI Graphic User Interface (Java swing)
	3. Data Structures (Array-Lists, 3-D Arrays, Lists, Arrays)
	4. SQLite Database (Library: Sqlite-jdbc-3.7.2.jar, Queries)
	5. Emailing system (Library: Javax.mail.jar, Coding complexity)
Intermediate techniques	6. File Parsing
	6. Error Handling
Basic techniques	8. Control Flow Statement Loops (While, For-each, Nested)
	9. Control Flow Statement Conditionals (If-Statements, Switch-case)

Table 1: Summary of techniques.

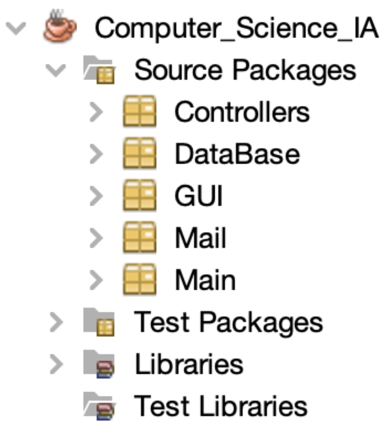
Advanced techniques:

OOP (Object Oriented Programming)

Besides the features mentioned in criteria A, OOP still has a great variety of advantages, promoting the creation of flexible projects optimized for future development and maintenance. This makes OOP suitable for this project allowing it to break down the complex features of the success criteria while making the project suitable for the long-run.

1.1. Modularity:

OOP, allows the organization of classes in a more comprehensive way, facilitating the reading and understanding of code. This project has been separated into 5 source packages:



```
Computer_Science_IA
  Source Packages
    Controllers
    DataBase
    GUI
    Mail
    Main
  Test Packages
  Libraries
  Test Libraries
```

Image 1: Source Packages.

Source Packages:

🟧 **(Controllers)** Includes the controller classes of each GUI class (assigned to store its functions)

🟪 **(DataBase)** Contains all SQLite queries and functions to access/interact with tables.

🟩 **(GUI)** Contains all GUI classes.

🟦 **(Mail)** Contains functions used to send emails.

🟨 **(Main)** The main class that runs the project.

Class	Source Package	Required Imports

LogIn	GUI	import Controllers.LogInController;
		import DataBase.DbConnection1;
		import java.awt.Color;
		import javax.swing.JFrame;
		import javax.swing.JOptionPane;

LogInController	Controllers	import Mail.MailUtil;
		import DataBase.DbConnection1;
		import java.sql.ResultSet;
		import javax.swing.JOptionPane;

SignIn	GUI	import DataBase.DbConnection1;
		import Controllers.SignInController;
		import java.awt.Color;
		import javax.swing.JFrame;
		import javax.swing.JOptionPane;

SignInController	Controllers	import DataBase.DbConnection1;

Menu	GUI	import Controllers.LogInController;
		import javax.swing.JOptionPane;
		import Controllers.MenuController;
		import java.util.ArrayList;

MenuController	Controllers	import java.sql.ResultSet;
		import java.util.ArrayList;

StudyGroups1	GUI	import Controllers.StudyGroups1Controller;
		import java.awt.Color;
		import static java.lang.Thread.sleep;
		import java.sql.ResultSet;
		import java.sql.SQLException;

		import java.util.ArrayList;
		import java.util.logging.Level;
		import java.util.logging.Logger;
		import javax.swing.JOptionPane;

StudyGroups1Controller	Controllers	import DataBase.DbConnection1;
		import GUI.StudyGroups1;
		import java.sql.ResultSet;
		import java.sql.SQLException;
		import java.util.ArrayList;
		import java.util.logging.Level;
		import java.util.logging.Logger;

StudyGroups2	GUI	import Controllers.StudyGroups2Controller;
		import java.util.ArrayList;
		import javax.swing.JList;
		import javax.swing.JOptionPane;

StudyGroups2Controller	Controllers	import DataBase.DbConnection1;
		import java.util.ArrayList;

StudyGroups3	GUI	import java.awt.Color;
		import javax.swing.JOptionPane;
		import Controllers.StudyGroups3Controller;
		import java.util.ArrayList;

StudyGroups3Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;
		import javax.mail.Message;
		import javax.mail.internet.InternetAddress;

StudyGroups4	GUI	import java.util.ArrayList;
		import Controllers.StudyGroups4Controller;
		import javax.swing.JOptionPane;

StudyGroups4Controller	Controllers	import DataBase.DbConnection1;

Tutoring1	GUI	import Controllers.Tutoring1Controller;
		import java.util.ArrayList;
		import javax.swing.JList;
		import javax.swing.JOptionPane;

Tutoring1Controller	Controllers	import DataBase.DbConnection1;
		import java.util.ArrayList;

Tutoring2	GUI	import Controllers.Tutoring2Controller;
		import java.util.ArrayList;

Tutoring2Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;

Tutoring3	GUI	import Controllers.Tutoring3Controller;
		import java.util.ArrayList;

Tutoring3Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;

Settings1	GUI	import javax.swing.JOptionPane;

Settings2	GUI	import Controllers.Settings2Controller;
		import javax.swing.JOptionPane;

Settings2Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;

Settings3	GUI	import Controllers.Settings3Controller;
		import javax.swing.JOptionPane;

Settings3Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;

Settings4	GUI	import Controllers.Settings4Controller;
		import javax.swing.JOptionPane;

Settings4Controller	Controllers	import DataBase.DbConnection1;
		import Mail.MailUtil;
		import java.util.ArrayList;

Settings5	GUI	import Controllers.Settings5Controller;

Settings5Controller	Controllers	import DataBase.DbConnection1;
		import java.sql.ResultSet;

AvailableRooms	GUI	import Controllers.AvailableRoomsController;
		import java.util.Calendar;
		import javax.swing.DefaultListModel;

AvailableRoomsController	Controllers	import java.io.BufferedReader;
		import java.io.FileReader;

DbConnection1	DataBase	import java.sql.*;
		import java.util.ArrayList;
		import java.util.logging.Level;
		import java.util.logging.Logger;

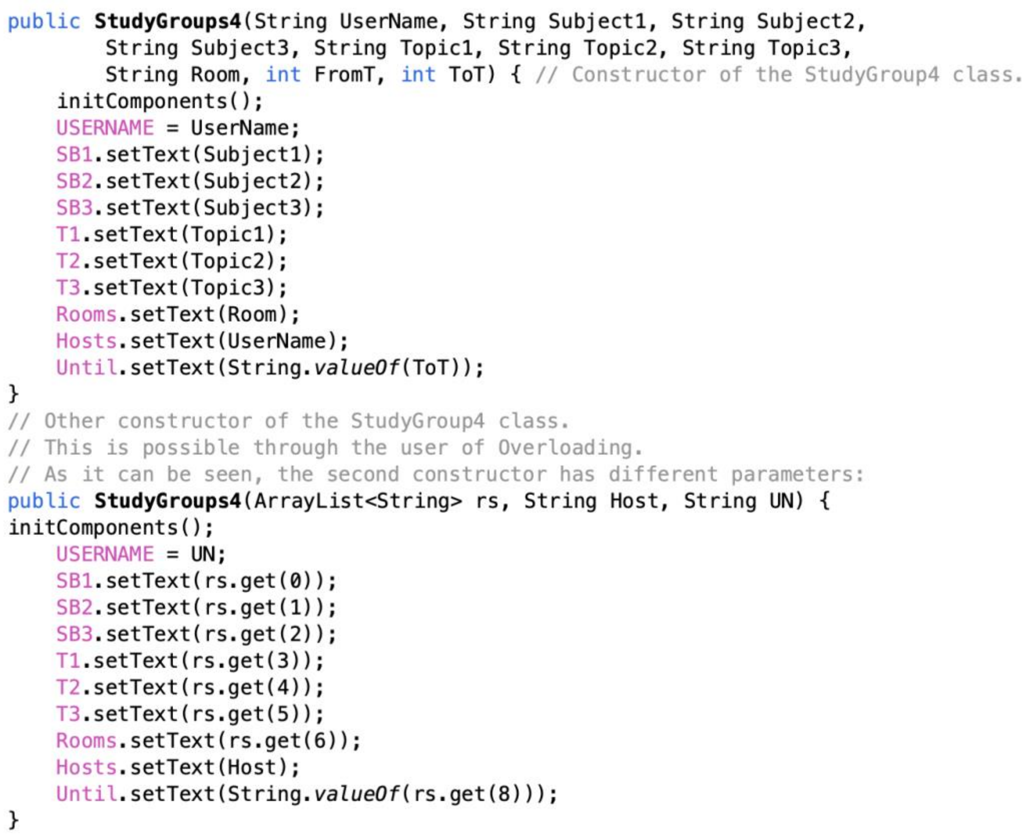
MailUtil	Mail	import java.util.ArrayList;
		import java.util.Properties;
		import java.util.logging.Level;
		import java.util.logging.Logger;
		import javax.mail.Authenticator;
		import javax.mail.Message;
		import javax.mail.PasswordAuthentication;
		import javax.mail.Session;
		import javax.mail.Transport;
		import javax.mail.internet.InternetAddress;
		import javax.mail.internet.MimeMessage;

Main	Main	import GUI.LogIn;

Table 2: Class + Source Packages + Required Imports.

1.2. Polymorphism:

In the context of this project, overloading is the only type of polymorphism used, allowing to have multiple methods of the same name in the same class (Great Learning Team, 2022). As we can see in the following example, the GUI Class "StudyGroups4" has two constructors:



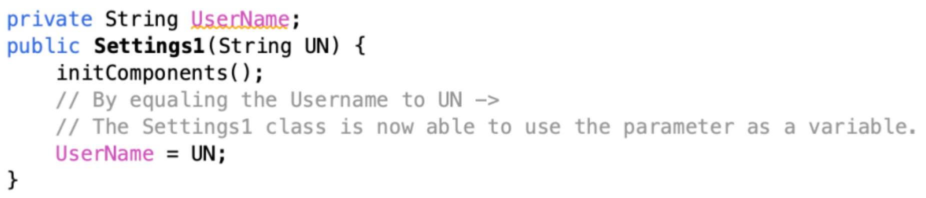
```
public StudyGroups4(String UserName, String Subject1, String Subject2,
        String Subject3, String Topic1, String Topic2, String Topic3,
        String Room, int FromT, int ToT) { // Constructor of the StudyGroup4 class.
    initComponents();
    USERNAME = UserName;
    SB1.setText(Subject1);
    SB2.setText(Subject2);
    SB3.setText(Subject3);
    T1.setText(Topic1);
    T2.setText(Topic2);
    T3.setText(Topic3);
    Rooms.setText(Room);
    Hosts.setText(UserName);
    Until.setText(String.valueOf(ToT));
}
// Other constructor of the StudyGroup4 class.
// This is possible through the user of Overloading.
// As it can be seen, the second constructor has different parameters:
public StudyGroups4(ArrayList<String> rs, String Host, String UN) {
initComponents();
    USERNAME = UN;
    SB1.setText(rs.get(0));
    SB2.setText(rs.get(1));
    SB3.setText(rs.get(2));
    T1.setText(rs.get(3));
    T2.setText(rs.get(4));
    T3.setText(rs.get(5));
    Rooms.setText(rs.get(6));
    Hosts.setText(Host);
    Until.setText(String.valueOf(rs.get(8)));
}
```

Image 2: Polymorphism.

The program does not crash because of the different parameters. This feature of OOP is used to access the "**Menu"** GUI from different classes with different data to be transferred.

1.3. Encapsulation:

After implementing encapsulation a variable and its data will be hidden from other classes, and only accessible through the methods in their current class (*Java - Encapsulation*, n.d.). This helps in protecting variables and methods from outside interference, reducing human errors. I am using this functionality to protect the "UserName" of users, used as ID/Primary-key.

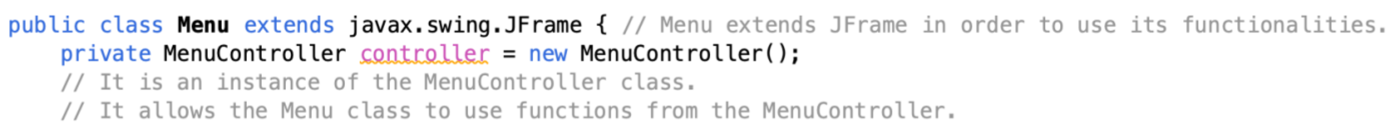


```
private String UserName;
public Settings1(String UN) {
    initComponents();
    // By equaling the Username to UN ->
    // The Settings1 class is now able to use the parameter as a variable.
    UserName = UN;
}
```

Image 3: Encapsulation.

1.4. Inclusion (Instance of a class in another class):

The OOP property of inclusion helped me separate the GUI classes from their actual methods and functions. Inclusion happens when an instance of a class is created in another class (Bodnar, n.d.). I have decided to separate the functionalities that the “Menu” GUI provides into two different classes: the “Menu” & “MenuController”. Through the use of a “MenuController” instance, the “Menu” class has access to all the functions existing in the controller. This fosters a better coding structure while ensuring a concrete barrier between GUI-related and non-GUI-related code.



```
public class Menu extends javax.swing.JFrame { // Menu extends JFrame in order to use its functionalities.
    private MenuController controller = new MenuController();
    // It is an instance of the MenuController class.
    // It allows the Menu class to use functions from the MenuController.
```

Image 4: Inclusion.

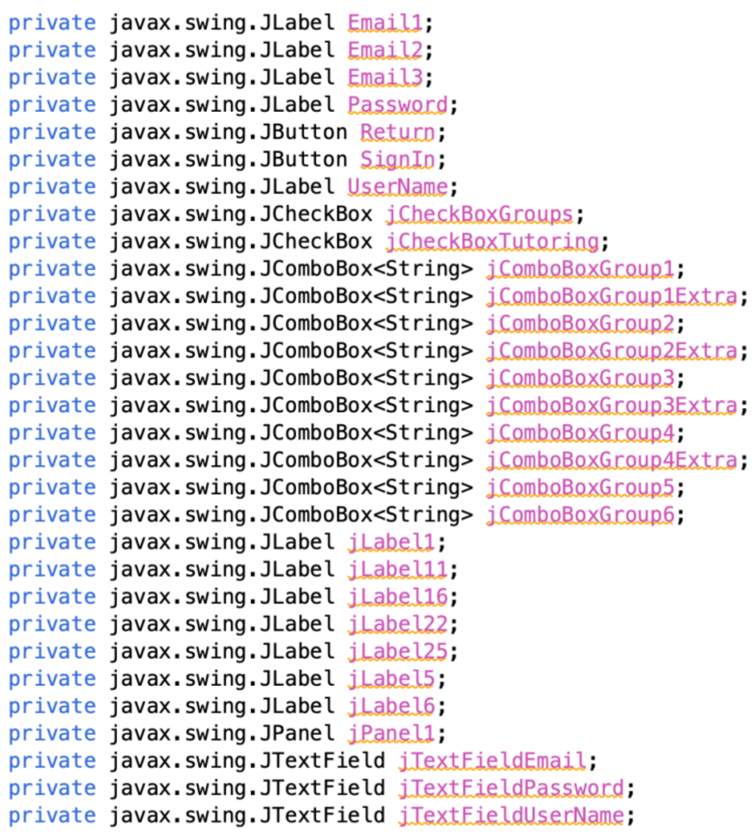
Graphic User Interface (Java Swing):

The rationale to make use of the Java Swing Library is based on the following facts:

• It offers a wide variety of components including standard and third-party-based. (GeeksforGeeks, 2022)
• More powerful components in contrast to other libraries such as ATW. (GeeksforGeeks, 2022)
• Supports MVC which emphasizes a separation between the software's logic and display. (GeeksforGeeks, 2022)
• Java Swing has a legacy UI library fully featured in contrast to other libraries like Javafx which are still being updated. (Pedamkar, 2022)

Table 3: Java Swing advantages.

The java swing library was a perfect fit for the requirements of the success criteria. A large part of the project required complex GUI frames involving specific components to be used. The client has specifically highlighted the importance of a user-friendly interface, not too overwhelming nor too simple [A.1]. Hence, reinforcing the conviction to make use of the Swing library.

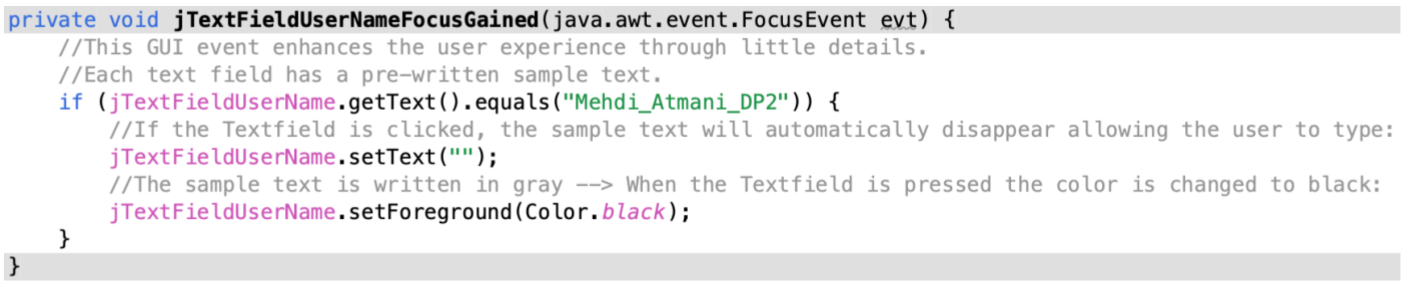


```java
private javax.swing.JLabel Email1;
private javax.swing.JLabel Email2;
private javax.swing.JLabel Email3;
private javax.swing.JLabel Password;
private javax.swing.JButton Return;
private javax.swing.JButton SignIn;
private javax.swing.JLabel UserName;
private javax.swing.JCheckBox jCheckBoxGroups;
private javax.swing.JCheckBox jCheckBoxTutoring;
private javax.swing.JComboBox<String> jComboBoxGroup1;
private javax.swing.JComboBox<String> jComboBoxGroup1Extra;
private javax.swing.JComboBox<String> jComboBoxGroup2;
private javax.swing.JComboBox<String> jComboBoxGroup2Extra;
private javax.swing.JComboBox<String> jComboBoxGroup3;
private javax.swing.JComboBox<String> jComboBoxGroup3Extra;
private javax.swing.JComboBox<String> jComboBoxGroup4;
private javax.swing.JComboBox<String> jComboBoxGroup4Extra;
private javax.swing.JComboBox<String> jComboBoxGroup5;
private javax.swing.JComboBox<String> jComboBoxGroup6;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel22;
private javax.swing.JLabel jLabel25;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JPanel jPanel1;
private javax.swing.JTextField jTextFieldEmail;
private javax.swing.JTextField jTextFieldPassword;
private javax.swing.JTextField jTextFieldUserName;
```

Image 5: GUI elements of the “SignIn” class.

The project has needed the following GUI elements: **JTextField**, **jPasswordField**, **JButton**, **JCheckBox**, **JList**, **JPanel**, **JLable**, **JTextArea**, **JComboBox**.

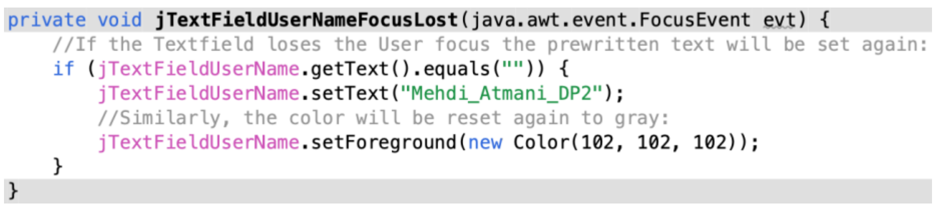
It is worth mentioning, that the project has made use of additional (coded) functionalities for the GUI elements. Focus listeners/events have been implemented to produce a smoother user experience:



```java
private void jTextFieldUserNameFocusGained(java.awt.event.FocusEvent evt) {
    //This GUI event enhances the user experience through little details.
    //Each text field has a pre-written sample text.
    if (jTextFieldUserName.getText().equals("Mehdi_Atmani_DP2")) {
        //If the Textfield is clicked, the sample text will automatically disappear allowing the user to type:
        jTextFieldUserName.setText("");
        //The sample text is written in gray --> When the Textfield is pressed the color is changed to black:
        jTextFieldUserName.setForeground(Color.black);
    }
}
```

Image 6: FocusGained Event.

In this particular example **(Image-6)**, when applying focus to the “jTextFieldUserName” the code will automatically delete the prewritten text and change the color of the font.



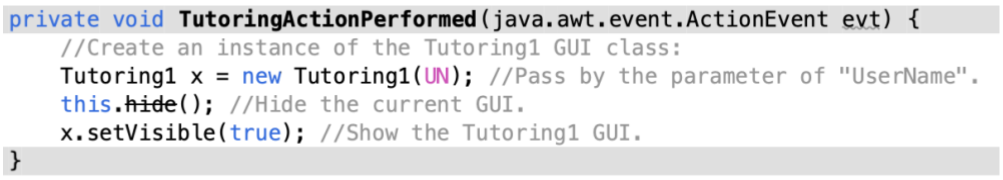
```java
private void jTextFieldUserNameFocusLost(java.awt.event.FocusEvent evt) {
    //If the Textfield loses the User focus the prewritten text will be set again:
    if (jTextFieldUserName.getText().equals("")) {
        jTextFieldUserName.setText("Mehdi_Atmani_DP2");
        //Similarly, the color will be reset again to gray:
        jTextFieldUserName.setForeground(new Color(102, 102, 102));
    }
}
```

Image 7: FocusLost Event.

In this other example **(Image-7)**, when losing focus the code will automatically restore the prewritten text and change the color back to gray.

Additionally, in order to move from **GUI** to **GUI** the system has required the following implementations:

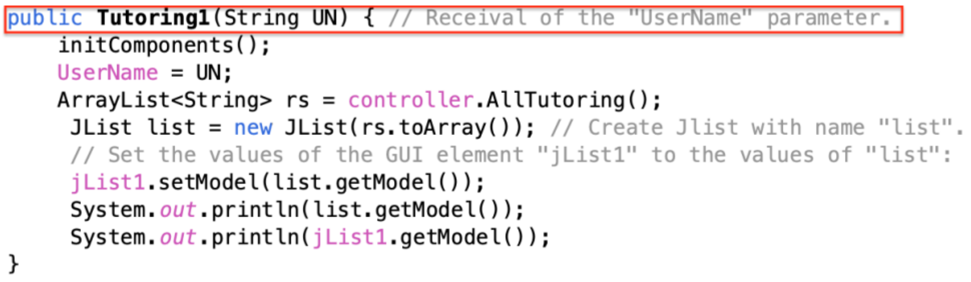
Current class "Menu GUI class":



```
private void TutoringActionPerformed(java.awt.event.ActionEvent evt) {
    //Create an instance of the Tutoring1 GUI class:
    Tutoring1 x = new Tutoring1(UN); //Pass by the parameter of "UserName".
    this.~~hide~~(); //Hide the current GUI.
    x.setVisible(true); //Show the Tutoring1 GUI.
}
```

Image 8: Menu GUI class "Action event".

Desired Class "Tutoring1 GUI class":



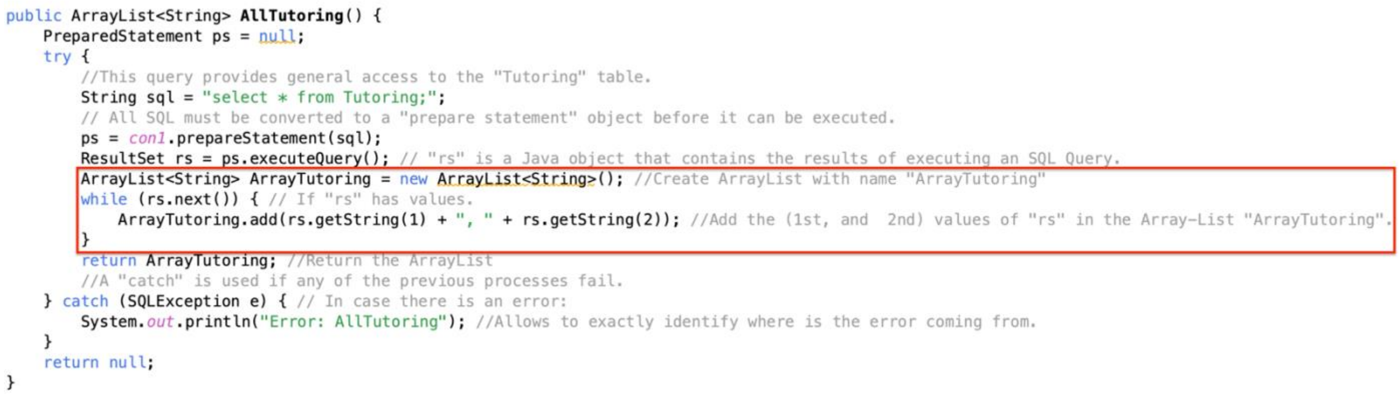
```
public Tutoring1(String UN) { // Receival of the "UserName" parameter.
    initComponents();
    UserName = UN;
    ArrayList<String> rs = controller.AllTutoring();
     JList list = new JList(rs.toArray()); // Create Jlist with name "list".
     // Set the values of the GUI element "jList1" to the values of "list":
     jList1.setModel(list.getModel());
     System.out.println(list.getModel());
     System.out.println(jList1.getModel());
}
```

Image 9: Tutoring1 GUI class "constructor".

Data Structures:

1.1. Array-Lists

It is a Java-exclusive data structure described as a resizable array. The project has required this data structure for the retrieval of information from the SQLite tables. This technique has been chosen above others because of the need for a **dynamic** yet **indexed** data structure. (Java ArrayList, n.d.)

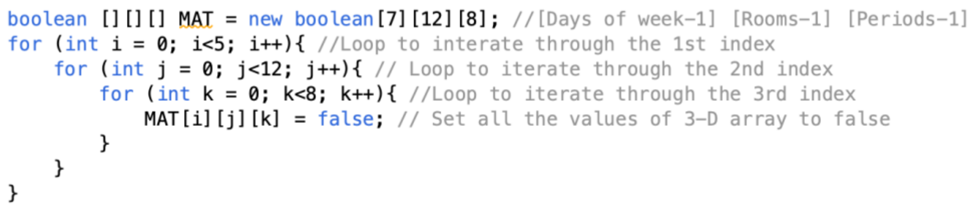


```
public ArrayList<String> AllTutoring() {
    PreparedStatement ps = null;
    try {
        //This query provides general access to the "Tutoring" table.
        String sql = "select * from Tutoring;";
        // All SQL must be converted to a "prepare statement" object before it can be executed.
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); // "rs" is a Java object that contains the results of executing an SQL Query.
        ArrayList<String> ArrayTutoring = new ArrayList<String>(); //Create ArrayList with name "ArrayTutoring"
        while (rs.next()) { // If "rs" has values.
            ArrayTutoring.add(rs.getString(1) + ", " + rs.getString(2)); //Add the (1st, and  2nd) values of "rs" in the Array-List "ArrayTutoring".
        }
        return ArrayTutoring; //Return the ArrayList
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) { // In case there is an error:
        System.out.println("Error: AllTutoring"); //Allows to exactly identify where is the error coming from.
    }
    return null;
}
```

Image 10: Array-List for retrieval of Tutoring information.

1.2. 3-D Arrays

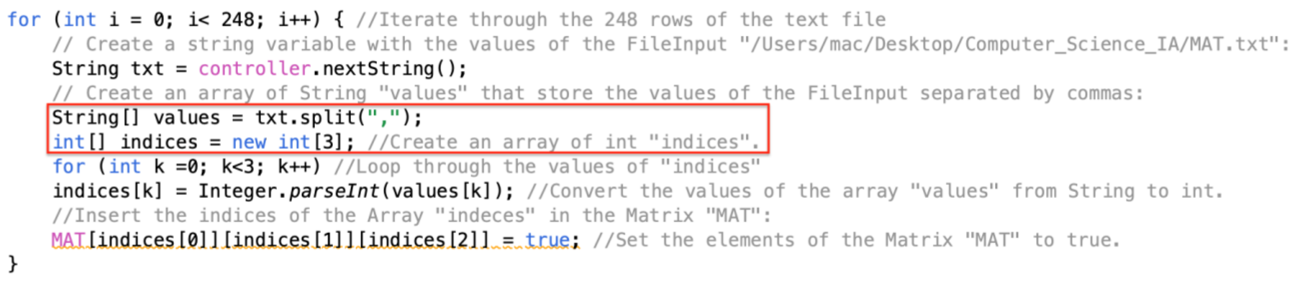
It is a multidimensional array considered as an (array of arrays). It stores the data in a tabular manner making use of a horizontal and vertical system where each intersection represents a cell (Bhandari, 2021). It has been used in this project in order to store the availability of any room at any time of the day anywhere in the week:



```
boolean [][][] MAT = new boolean[7][12][8]; //[Days of week-1] [Rooms-1] [Periods-1]
for (int i = 0; i<5; i++){ //Loop to interate through the 1st index
    for (int j = 0; j<12; j++){ // Loop to iterate through the 2nd index
        for (int k = 0; k<8; k++){ //Loop to iterate through the 3rd index
            MAT[i][j][k] = false; // Set all the values of 3-D array to false
        }
    }
}
```

Image 11: 3-D Array in "AvailableRooms" GUI Class.

1.3. Arrays



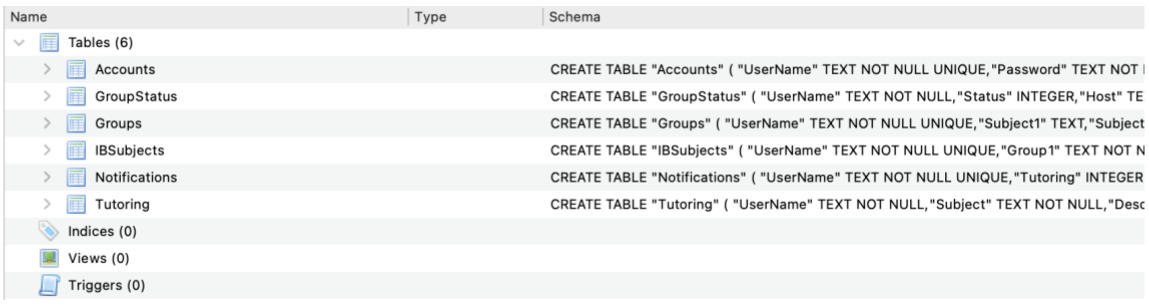
```
for (int i = 0; i< 248; i++) { //Iterate through the 248 rows of the text file
    // Create a string variable with the values of the FileInput "/Users/mac/Desktop/Computer_Science_IA/MAT.txt":
    String txt = controller.nextString();
    // Create an array of String "values" that store the values of the FileInput separated by commas:
    String[] values = txt.split(",");
    int[] indices = new int[3]; //Create an array of int "indices".
    for (int k =0; k<3; k++) //Loop through the values of "indices"
    indices[k] = Integer.parseInt(values[k]); //Convert the values of the array "values" from String to int.
    //Insert the indices of the Array "indeces" in the Matrix "MAT":
    MAT[indices[0]][indices[1]][indices[2]] = true; //Set the elements of the Matrix "MAT" to true.
}
```

Image 12: Array of String and Integers in the "AvailableRooms" GUI class.

The Arrays used in (**Image 12**) helped to efficiently parse the values of a text file into a 3-D Matrix.

SQLite Database:

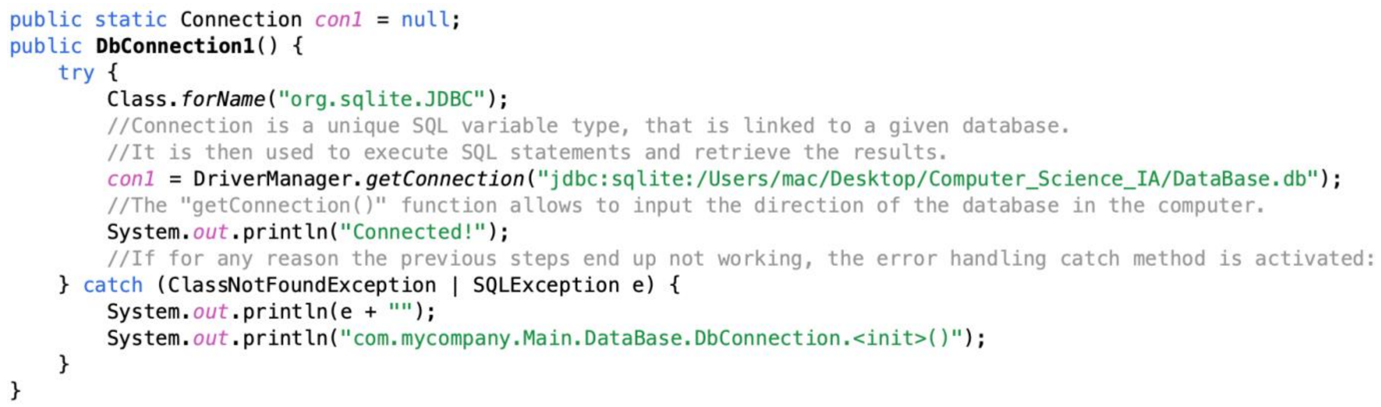
To achieve specific goals of the success criteria, this project has required the use of databases. After consideration, the SQLite option was chosen given that the client was not yet ready to invest in cloud storage. In order to manage and create the **database** the computer application **DB-Browser** has been used. The following **tables** have been created:



Name	Type	Schema
Tables (6)		
Accounts		CREATE TABLE "Accounts" ("UserName" TEXT NOT NULL UNIQUE,"Password" TEXT NOT
GroupStatus		CREATE TABLE "GroupStatus" ("UserName" TEXT NOT NULL,"Status" INTEGER,"Host" TE
Groups		CREATE TABLE "Groups" ("UserName" TEXT NOT NULL UNIQUE,"Subject1" TEXT,"Subject
IBSubjects		CREATE TABLE "IBSubjects" ("UserName" TEXT NOT NULL UNIQUE,"Group1" TEXT NOT N
Notifications		CREATE TABLE "Notifications" ("UserName" TEXT NOT NULL UNIQUE,"Tutoring" INTEGER
Tutoring		CREATE TABLE "Tutoring" ("UserName" TEXT NOT NULL,"Subject" TEXT NOT NULL,"Desc
Indices (0)		
Views (0)		
Triggers (0)		

Image 13: Database tables.

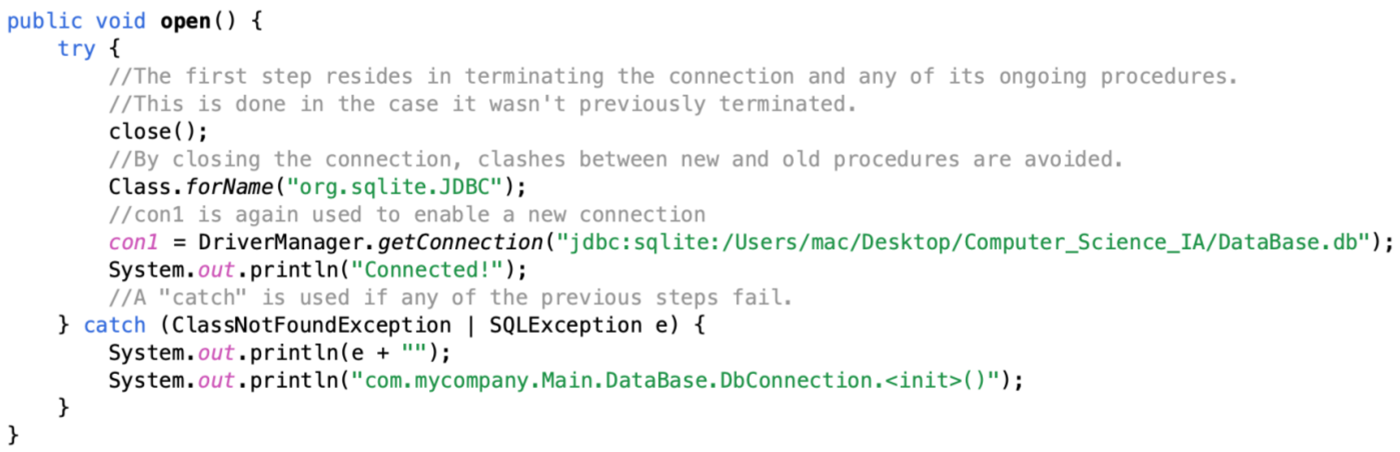
The program connects to the SQLite database by a **connection** object:



```
public static Connection con1 = null;
public DbConnection1() {
    try {
        Class.forName("org.sqlite.JDBC");
        //Connection is a unique SQL variable type, that is linked to a given database.
        //It is then used to execute SQL statements and retrieve the results.
        con1 = DriverManager.getConnection("jdbc:sqlite:/Users/mac/Desktop/Computer_Science_IA/DataBase.db");
        //The "getConnection()" function allows to input the direction of the database in the computer.
        System.out.println("Connected!");
        //If for any reason the previous steps end up not working, the error handling catch method is activated:
    } catch (ClassNotFoundException | SQLException e) {
        System.out.println(e + "");
        System.out.println("com.mycompany.Main.DataBase.DbConnection.<init>()");
    }
}
```

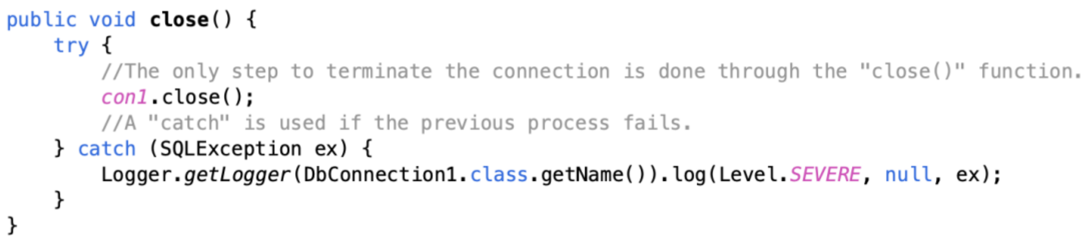
Image 14: Connection function “DbConnection” class.

Every time the Database is opened or closed the following functions need to be called:



```
public void open() {
    try {
        //The first step resides in terminating the connection and any of its ongoing procedures.
        //This is done in the case it wasn't previously terminated.
        close();
        //By closing the connection, clashes between new and old procedures are avoided.
        Class.forName("org.sqlite.JDBC");
        //con1 is again used to enable a new connection
        con1 = DriverManager.getConnection("jdbc:sqlite:/Users/mac/Desktop/Computer_Science_IA/DataBase.db");
        System.out.println("Connected!");
        //A "catch" is used if any of the previous steps fail.
    } catch (ClassNotFoundException | SQLException e) {
        System.out.println(e + "");
        System.out.println("com.mycompany.Main.DataBase.DbConnection.<init>()");
    }
}
```

Image 15: Open function “DbConnection” class.

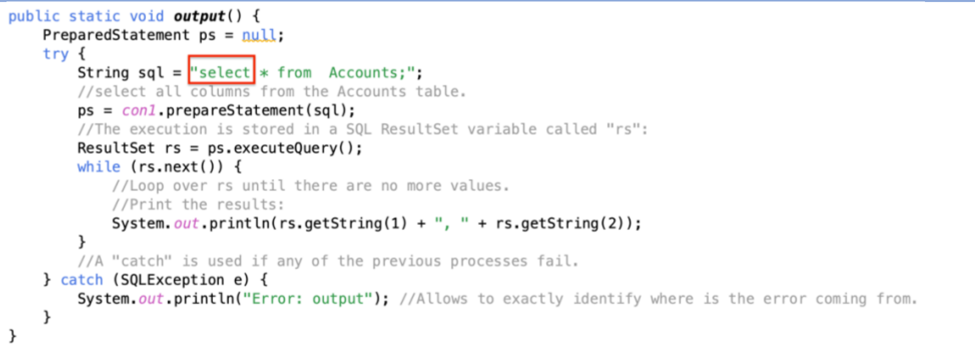
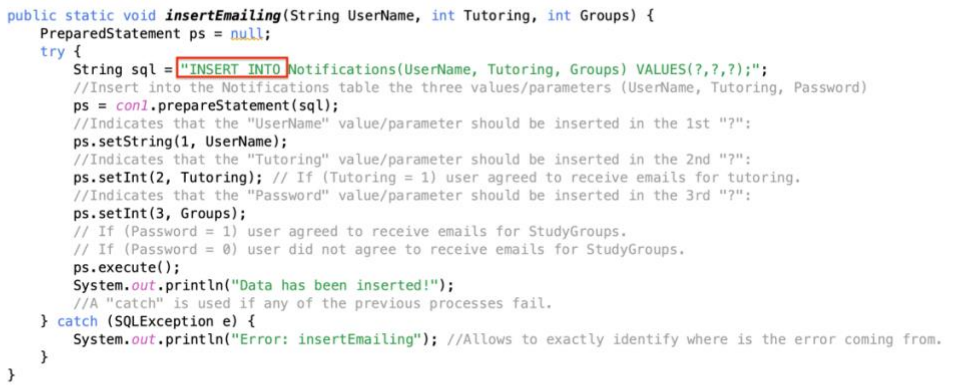
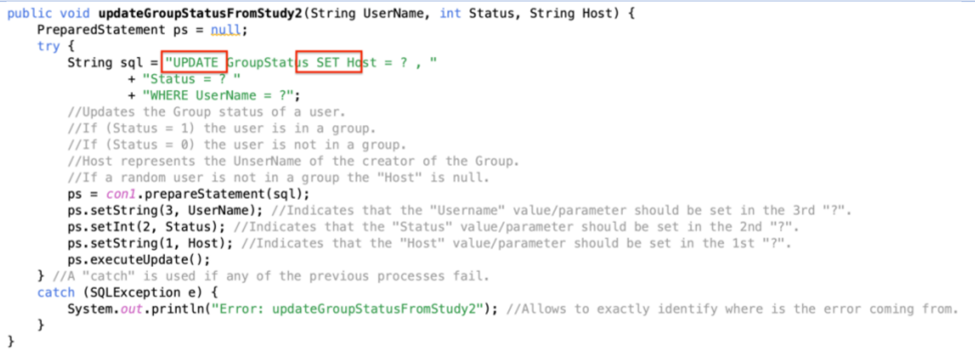
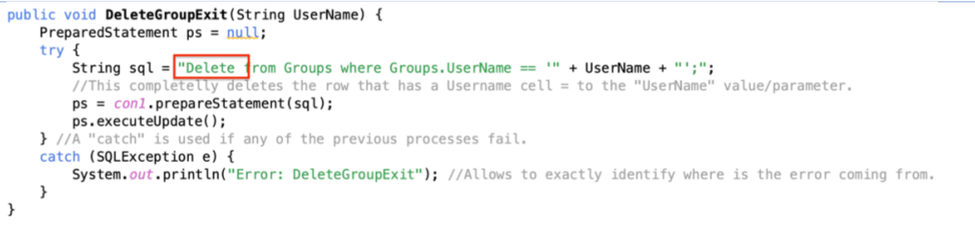


```
public void close() {
    try {
        //The only step to terminate the connection is done through the "close()" function.
        con1.close();
        //A "catch" is used if the previous process fails.
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection1.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Image 16 Close function “DbConnection” class.

To make the **connection**, **access**, and **creation** of **SQLite tables** possible the **JDBC API** has been used (A set of interfaces and classes written in Java). Making use of these standard interfaces and classes allowed me to write functions that connect to tables, send queries (written in structured query language “SQL”), and process results.

Every function in charge of interacting with a database will make use of a query. Each query has specific commands indicating its purpose:

Command	**Purpose**	**Example**
Select	Used to retrieve data from the Database	
Insert Into	Used to insert data in the Database	
Update Set	Used to Update data from the Database	
Delete	Used to delete data from the Database	

```java
public static void output() {
    PreparedStatement ps = null;
    try {
        String sql = "select * from  Accounts;";
        //select all columns from the Accounts table.
        ps = con1.prepareStatement(sql);
        //The execution is stored in a SQL ResultSet variable called "rs":
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            //Loop over rs until there are no more values.
            //Print the results:
            System.out.println(rs.getString(1) + ", " + rs.getString(2));
        }
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: output"); //Allows to exactly identify where is the error coming from.
    }
}
```

```java
public static void insertEmailing(String UserName, int Tutoring, int Groups) {
    PreparedStatement ps = null;
    try {
        String sql = "INSERT INTO Notifications(UserName, Tutoring, Groups) VALUES(?,?,?);";
        //Insert into the Notifications table the three values/parameters (UserName, Tutoring, Password)
        ps = con1.prepareStatement(sql);
        //Indicates that the "UserName" value/parameter should be inserted in the 1st "?":
        ps.setString(1, UserName);
        //Indicates that the "Tutoring" value/parameter should be inserted in the 2nd "?":
        ps.setInt(2, Tutoring); // If (Tutoring = 1) user agreed to receive emails for tutoring.
        //Indicates that the "Password" value/parameter should be inserted in the 3rd "?":
        ps.setInt(3, Groups);
        // If (Password = 1) user agreed to receive emails for StudyGroups.
        // If (Password = 0) user did not agree to receive emails for StudyGroups.
        ps.execute();
        System.out.println("Data has been inserted!");
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: insertEmailing"); //Allows to exactly identify where is the error coming from.
    }
}
```

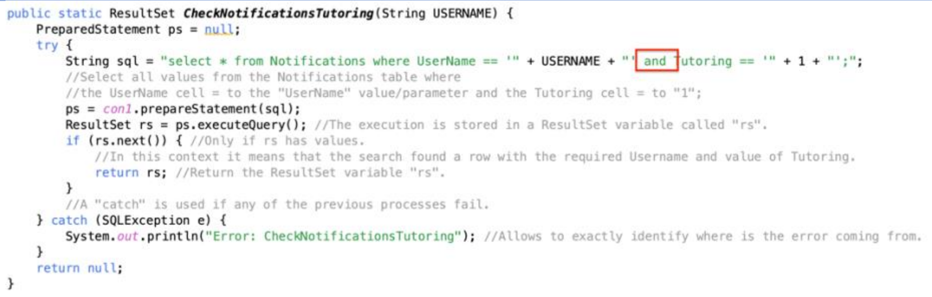
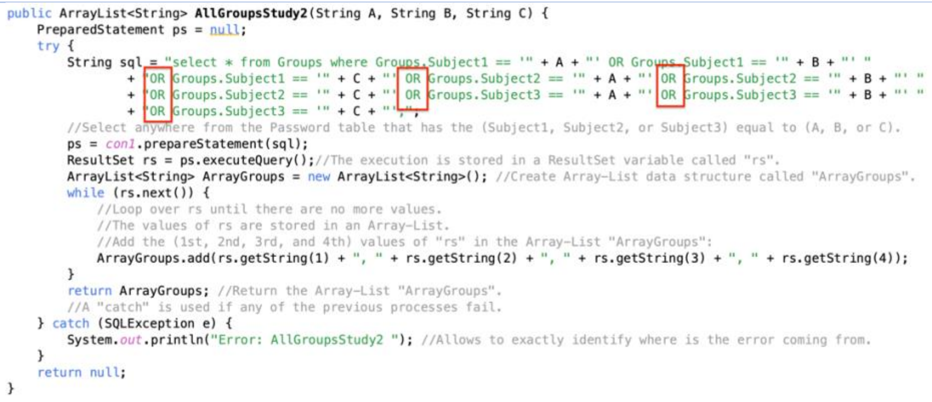
```java
public void updateGroupStatusFromStudy2(String UserName, int Status, String Host) {
    PreparedStatement ps = null;
    try {
        String sql = "UPDATE GroupStatus SET Host = ? , "
                + "Status = ? "
                + "WHERE UserName = ?";
        //Updates the Group status of a user.
        //If (Status = 1) the user is in a group.
        //If (Status = 0) the user is not in a group.
        //Host represents the UnserName of the creator of the Group.
        //If a random user is not in a group the "Host" is null.
        ps = con1.prepareStatement(sql);
        ps.setString(3, UserName); //Indicates that the "Username" value/parameter should be set in the 3rd "?".
        ps.setInt(2, Status); //Indicates that the "Status" value/parameter should be set in the 2nd "?".
        ps.setString(1, Host); //Indicates that the "Host" value/parameter should be set in the 1st "?".
        ps.executeUpdate();
    } //A "catch" is used if any of the previous processes fail.
    catch (SQLException e) {
        System.out.println("Error: updateGroupStatusFromStudy2"); //Allows to exactly identify where is the error coming from.
    }
}
```

```java
public void DeleteGroupExit(String UserName) {
    PreparedStatement ps = null;
    try {
        String sql = "Delete from Groups where Groups.UserName == '" + UserName + "';";
        //This completelly deletes the row that has a Username cell = to the "UserName" value/parameter.
        ps = con1.prepareStatement(sql);
        ps.executeUpdate();
    } //A "catch" is used if any of the previous processes fail.
    catch (SQLException e) {
        System.out.println("Error: DeleteGroupExit"); //Allows to exactly identify where is the error coming from.
    }
}
```

Table 4: Query Commands.

Queries are also composed of (logical operators, comparison operators, conditional clauses, and joins). The ones used in this project are specified down below:

Logical Operators:

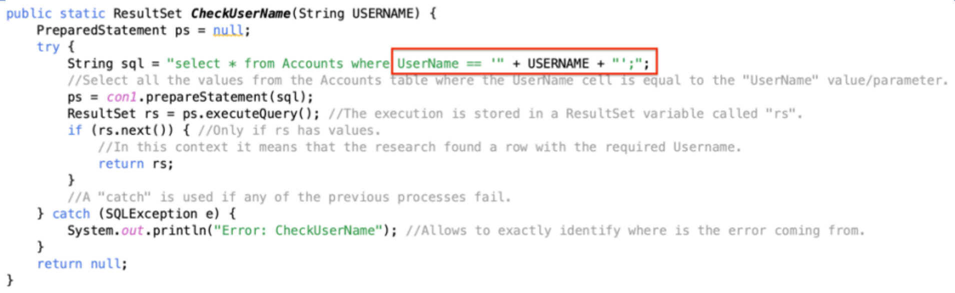
Operator	**Purpose**	**Example**
And	Allows the existence of multiple conditions in a single query.	
Or	It is used to combine multiple conditions in a single query.	

```
public static ResultSet CheckNotificationsTutoring(String USERNAME) {
    PreparedStatement ps = null;
    try {
        String sql = "select * from Notifications where UserName == '" + USERNAME + "' and Tutoring == '" + 1 + "';";
        //Select all values from the Notifications table where
        //the UserName cell = to the "UserName" value/parameter and the Tutoring cell = to "1";
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        if (rs.next()) { //Only if rs has values.
            //In this context it means that the search found a row with the required Username and value of Tutoring.
            return rs; //Return the ResultSet variable "rs".
        }
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: CheckNotificationsTutoring"); //Allows to exactly identify where is the error coming from.
    }
    return null;
}
```

```
public ArrayList<String> AllGroupsStudy2(String A, String B, String C) {
    PreparedStatement ps = null;
    try {
        String sql = "select * from Groups where Groups.Subject1 == '" + A + "' OR Groups.Subject1 == '" + B + "' "
                + "OR Groups.Subject1 == '" + C + "' OR Groups.Subject2 == '" + A + "' OR Groups.Subject2 == '" + B + "' "
                + "OR Groups.Subject2 == '" + C + "' OR Groups.Subject3 == '" + A + "' OR Groups.Subject3 == '" + B + "' "
                + "OR Groups.Subject3 == '" + C + "';";
        //Select anywhere from the Password table that has the (Subject1, Subject2, or Subject3) equal to (A, B, or C).
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();//The execution is stored in a ResultSet variable called "rs".
        ArrayList<String> ArrayGroups = new ArrayList<String>(); //Create Array-List data structure called "ArrayGroups".
        while (rs.next()) {
            //Loop over rs until there are no more values.
            //The values of rs are stored in an Array-List.
            //Add the (1st, 2nd, 3rd, and 4th) values of "rs" in the Array-List "ArrayGroups":
            ArrayGroups.add(rs.getString(1) + ", " + rs.getString(2) + ", " + rs.getString(3) + ", " + rs.getString(4));
        }
        return ArrayGroups; //Return the Array-List "ArrayGroups".
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: AllGroupsStudy2 "); //Allows to exactly identify where is the error coming from.
    }
    return null;
}
```

Table 5: Query Logical Operators.

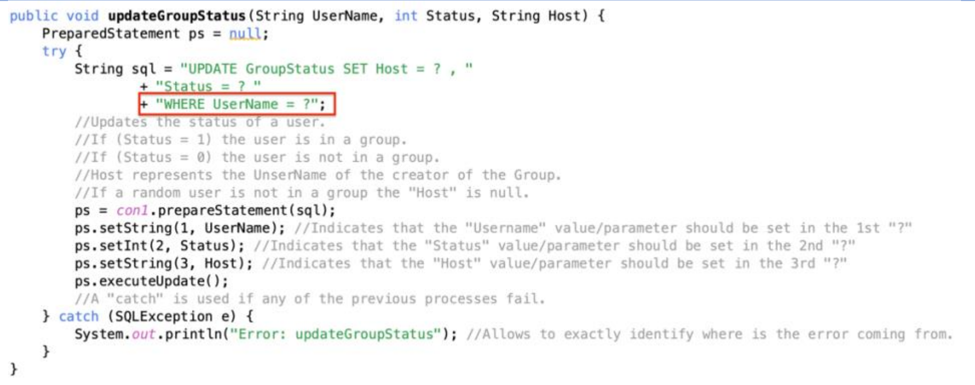
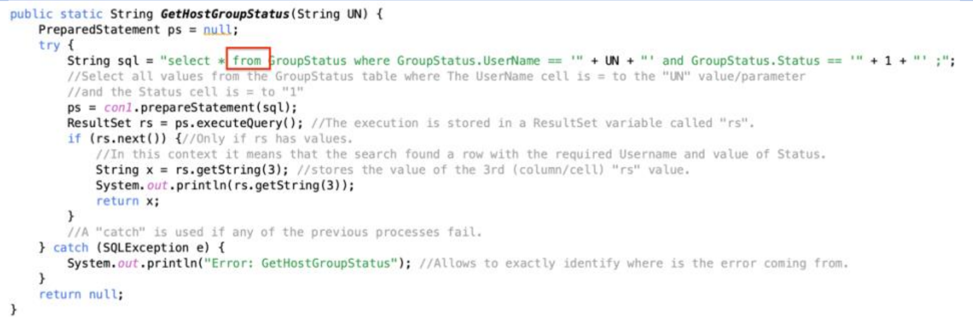
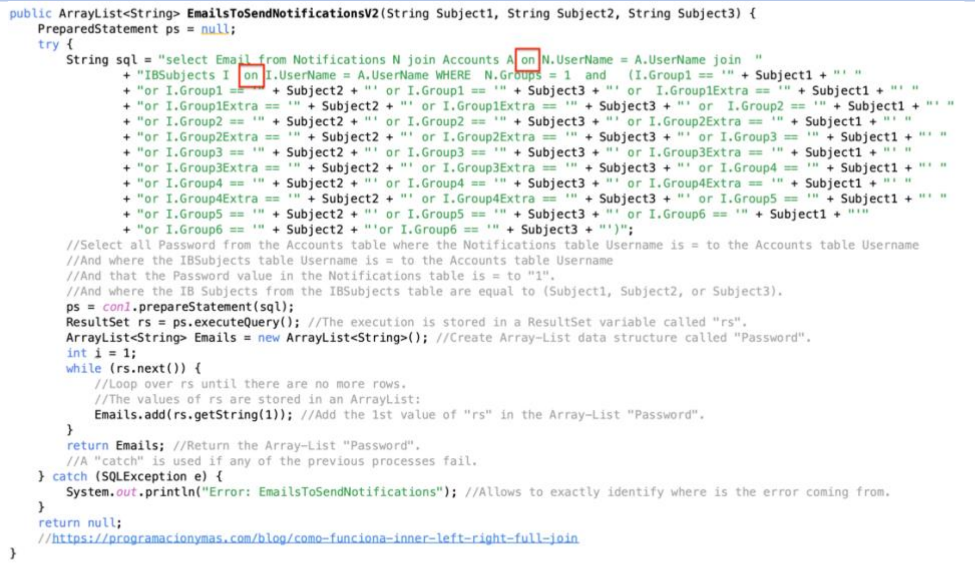
Comparison Operators:

Operator	**Purpose**	**Example**
= ==	Checks if the values of two operands are equal. If yes condition → true, otherwise → false.	

```
public static ResultSet CheckUserName(String USERNAME) {
    PreparedStatement ps = null;
    try {
        String sql = "select * from Accounts where UserName == '" + USERNAME + "';";
        //Select all the values from the Accounts table where the UserName cell is equal to the "UserName" value/parameter.
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        if (rs.next()) { //Only if rs has values.
            //In this context it means that the research found a row with the required Username.
            return rs;
        }
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: CheckUserName"); //Allows to exactly identify where is the error coming from.
    }
    return null;
}
```

Table 6: Query Comparison Operators.

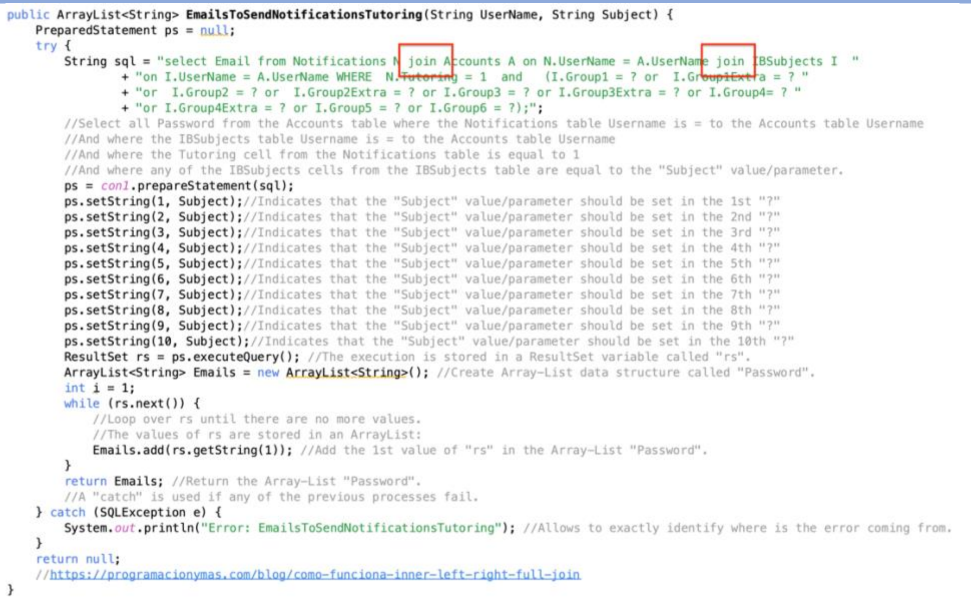
Condition clauses:

Operator	**Purpose**	**Example**
Where	Contains the conditions that must turn true for a row to be returned as a result.	
From	Specifies the table to be used	
On	The On clause is used to specify a Join condition	

```java
public void updateGroupStatus(String UserName, int Status, String Host) {
    PreparedStatement ps = null;
    try {
        String sql = "UPDATE GroupStatus SET Host = ? , "
                + "Status = ? "
                + "WHERE UserName = ?";
        //Updates the status of a user.
        //If (Status = 1) the user is in a group.
        //If (Status = 0) the user is not in a group.
        //Host represents the UnserName of the creator of the Group.
        //If a random user is not in a group the "Host" is null.
        ps = con1.prepareStatement(sql);
        ps.setString(1, UserName); //Indicates that the "Username" value/parameter should be set in the 1st "?"
        ps.setInt(2, Status); //Indicates that the "Status" value/parameter should be set in the 2nd "?"
        ps.setString(3, Host); //Indicates that the "Host" value/parameter should be set in the 3rd "?"
        ps.executeUpdate();
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: updateGroupStatus"); //Allows to exactly identify where is the error coming from.
    }
}
```

```java
public static String GetHostGroupStatus(String UN) {
    PreparedStatement ps = null;
    try {
        String sql = "select * from GroupStatus where GroupStatus.UserName == '" + UN + "' and GroupStatus.Status == '" + 1 + "' ;";
        //Select all values from the GroupStatus table where The UserName cell is = to the "UN" value/parameter
        //and the Status cell is = to "1"
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        if (rs.next()) {//Only if rs has values.
            //In this context it means that the search found a row with the required Username and value of Status.
            String x = rs.getString(3); //stores the value of the 3rd (column/cell) "rs" value.
            System.out.println(rs.getString(3));
            return x;
        }
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: GetHostGroupStatus"); //Allows to exactly identify where is the error coming from.
    }
    return null;
}
```

```java
public ArrayList<String> EmailsToSendNotificationsV2(String Subject1, String Subject2, String Subject3) {
    PreparedStatement ps = null;
    try {
        String sql = "select Email from Notifications N join Accounts A on N.UserName = A.UserName join  "
                + "IBSubjects I  on I.UserName = A.UserName WHERE  N.Groups = 1  and   (I.Group1 == '" + Subject1 + "' "
                + "or I.Group1 == '" + Subject2 + "' or I.Group1 == '" + Subject3 + "' or  I.Group1Extra == '" + Subject1 + "' "
                + "or I.Group1Extra == '" + Subject2 + "' or I.Group1Extra == '" + Subject3 + "' or  I.Group2 == '" + Subject1 + "' "
                + "or I.Group2 == '" + Subject2 + "' or I.Group2 == '" + Subject3 + "' or I.Group2Extra == '" + Subject1 + "' "
                + "or I.Group2Extra == '" + Subject2 + "' or I.Group2Extra == '" + Subject3 + "' or I.Group3 == '" + Subject1 + "' "
                + "or I.Group3 == '" + Subject2 + "' or I.Group3 == '" + Subject3 + "' or I.Group3Extra == '" + Subject1 + "' "
                + "or I.Group3Extra == '" + Subject2 + "' or I.Group3Extra == '" + Subject3 + "' or I.Group4 == '" + Subject1 + "' "
                + "or I.Group4 == '" + Subject2 + "' or I.Group4 == '" + Subject3 + "' or I.Group4Extra == '" + Subject1 + "' "
                + "or I.Group4Extra == '" + Subject2 + "' or I.Group4Extra == '" + Subject3 + "' or I.Group5 == '" + Subject1 + "' "
                + "or I.Group5 == '" + Subject2 + "' or I.Group5 == '" + Subject3 + "' or I.Group6 == '" + Subject1 + "'"
                + "or I.Group6 == '" + Subject2 + "'or I.Group6 == '" + Subject3 + "')";
        //Select all Password from the Accounts table where the Notifications table Username is = to the Accounts table Username
        //And where the IBSubjects table Username is = to the Accounts table Username
        //And that the Password value in the Notifications table is = to "1".
        //And where the IB Subjects from the IBSubjects table are equal to (Subject1, Subject2, or Subject3).
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        ArrayList<String> Emails = new ArrayList<String>(); //Create Array-List data structure called "Password".
        int i = 1;
        while (rs.next()) {
            //Loop over rs until there are no more rows.
            //The values of rs are stored in an ArrayList:
            Emails.add(rs.getString(1)); //Add the 1st value of "rs" in the Array-List "Password".
        }
        return Emails; //Return the Array-List "Password".
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: EmailsToSendNotifications"); //Allows to exactly identify where is the error coming from.
    }
    return null;
    //https://programacionymas.com/blog/como-funciona-inner-left-right-full-join
}
```

Operator	**Purpose**	**Example**
Join	Used to combine rows from two or more tables.	

```
public ArrayList<String> EmailsToSendNotificationsTutoring(String UserName, String Subject) {
    PreparedStatement ps = null;
    try {
        String sql = "select Email from Notifications N join Accounts A on N.UserName = A.UserName join IBSubjects I  "
                + "on I.UserName = A.UserName WHERE  N.Tutoring = 1  and   (I.Group1 = ? or  I.Group1Extra = ? "
                + "or  I.Group2 = ? or  I.Group2Extra = ? or I.Group3 = ? or I.Group3Extra = ? or I.Group4= ? "
                + "or I.Group4Extra = ? or I.Group5 = ? or I.Group6 = ?);";
        //Select all Password from the Accounts table where the Notifications table Username is = to the Accounts table Username
        //And where the IBSubjects table Username is = to the Accounts table Username
        //And where the Tutoring cell from the Notifications table is equal to 1
        //And where any of the IBSubjects cells from the IBSubjects table are equal to the "Subject" value/parameter.
        ps = con1.prepareStatement(sql);
        ps.setString(1, Subject);//Indicates that the "Subject" value/parameter should be set in the 1st "?"
        ps.setString(2, Subject);//Indicates that the "Subject" value/parameter should be set in the 2nd "?"
        ps.setString(3, Subject);//Indicates that the "Subject" value/parameter should be set in the 3rd "?"
        ps.setString(4, Subject);//Indicates that the "Subject" value/parameter should be set in the 4th "?"
        ps.setString(5, Subject);//Indicates that the "Subject" value/parameter should be set in the 5th "?"
        ps.setString(6, Subject);//Indicates that the "Subject" value/parameter should be set in the 6th "?"
        ps.setString(7, Subject);//Indicates that the "Subject" value/parameter should be set in the 7th "?"
        ps.setString(8, Subject);//Indicates that the "Subject" value/parameter should be set in the 8th "?"
        ps.setString(9, Subject);//Indicates that the "Subject" value/parameter should be set in the 9th "?"
        ps.setString(10, Subject);//Indicates that the "Subject" value/parameter should be set in the 10th "?"
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        ArrayList<String> Emails = new ArrayList<String>(); //Create Array-List data structure called "Password".
        int i = 1;
        while (rs.next()) {
            //Loop over rs until there are no more values.
            //The values of rs are stored in an ArrayList:
            Emails.add(rs.getString(1)); //Add the 1st value of "rs" in the Array-List "Password".
        }
        return Emails; //Return the Array-List "Password".
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: EmailsToSendNotificationsTutoring"); //Allows to exactly identify where is the error coming from.
    }
    return null;
    //https://programacionymas.com/blog/como-funciona-inner-left-right-full-join
}
```

Table 7: Query Condition Clauses.

The list of queries used in this project are presented down below:

Name Functions	**Queries**
insertSign	"INSERT INTO Accounts(UserName,Password, Email) VALUES(?,?,?);"
output	"select * from Accounts;"
CheckUserName	"select * from Accounts where UserName == '" + USERNAME + "';"
Check	"select * from Accounts where Accounts.Password == '" + PASSWORD + "' and Accounts.UserName == '" + USERNAME + "';"
CheckExistingAccount	"select * from Accounts where Accounts.UserName == '" + UserName + "' and Accounts.Password == '" + Password + "' and Accounts.Email == '" + Email + "';"
insertSubjects	"INSERT INTO IBSubjects(UserName, Group1, Group1Extra, Group2, Group2Extra, Group3, Group3Extra, Group4, Group4Extra, Group5, Group6) VALUES(?,?,?,?,?,?,?,?,?,?,?);"
insertEmailing	"INSERT INTO Notifications(UserName, Tutoring, Groups) VALUES(?,?,?);"
GroupSubjects	"select * from IBSubjects where IBSubjects.UserName == '" + UN1 + "';"
CheckExistingGroup	"select * from Groups where Groups.Subject1 == '" + A + "' OR Groups.Subject1 == '" + B + "' OR Groups.Subject1 == '" + C + "' OR Groups.Subject2 == '" + A + "' OR Groups.Subject2 == '" + B + "' OR Groups.Subject2 == '" + C + "' OR Groups.Subject3 == '" + A + "' OR Groups.Subject3 == '" + B + "' OR Groups.Subject3 == '" + C + "';";

insertNewGroup	"INSERT INTO Groups(UserName, Subject1, Subject2, Subject3, Topic1, Topic2, Topic3, Room, FromT, ToT) VALUES(?,?,?,?,?,?,?,?,?,?);"
CheckGroupStatus	"select * from GroupStatus where GroupStatus.UserName == '" + UN + "' and GroupStatus.Status == '" +0+ "' ;"
insertGroupStatus	"INSERT INTO GroupStatus (UserName,Status, Host) VALUES(?,?,?);"
CheckGroupStatusInit	"select * from GroupStatus where GroupStatus.UserName == '" + UN + "';"
updateGroupStatus	"UPDATE GroupStatus SET Host = ? , "+ "Status = ? "+ "WHERE UserName = ?"
GetHostGroupStatus	"select * from GroupStatus where GroupStatus.UserName == '" + UN + "' and GroupStatus.Status == '" +1+ "' ;"
GroupSubjectsFromMenu	"select * from Groups where Groups.UserName == '" +Host + "';"
EmailsToSendNotifications	"select Email from Notifications N join Accounts A on N.UserName = A.UserName WHERE N.Groups = 1 ;"
EmailsToSendNotificationsV2	"select Email from Notifications N join Accounts A on N.UserName = A.UserName join IBSubjects I on I.UserName = A.UserName WHERE N.Groups = 1 and (I.Group1 == '" +Subject1+ "' or I.Group1 == '" +Subject2+ "' or I.Group1 == '" +Subject3+ "' or I.Group1Extra == '" +Subject1+ "' or I.Group1Extra == '" +Subject2+ "' or I.Group1Extra == '" +Subject3+ "' or I.Group2 == '" +Subject1+ "' or I.Group2 == '" +Subject2+ "' or I.Group2 == '" +Subject3+ "' or I.Group2Extra == '" +Subject1+ "' or I.Group2Extra == '" +Subject2+ "' or I.Group2Extra == '" +Subject3+ "' or I.Group3 == '" +Subject1+ "' or I.Group3 == '" +Subject2+ "' or I.Group3 == '" +Subject3+ "' or I.Group3Extra == '" +Subject1+ "' or I.Group3Extra == '" +Subject2+ "' or I.Group3Extra == '" +Subject3+ "' or I.Group4 == '" +Subject1+ "' or I.Group4 == '" +Subject2+ "' or I.Group4 == '" +Subject3+ "' or I.Group4Extra == '" +Subject1+ "' or I.Group4Extra == '" +Subject2+ "' or I.Group4Extra == '" +Subject3+ "' or I.Group5 == '" +Subject1+ "' or I.Group5 == '" +Subject2+ "' or I.Group5 == '" +Subject3+ "' or I.Group6 == '" +Subject1+ "'or I.Group6 == '" +Subject2+ "'or I.Group6 == '" +Subject3+ "')";”
AllGroupsStudy2	"select * from Groups where Groups.Subject1 == '" + A + "' OR Groups.Subject1 == '" + B + "' OR Groups.Subject1 == '" + C + "' OR Groups.Subject2 == '" + A +

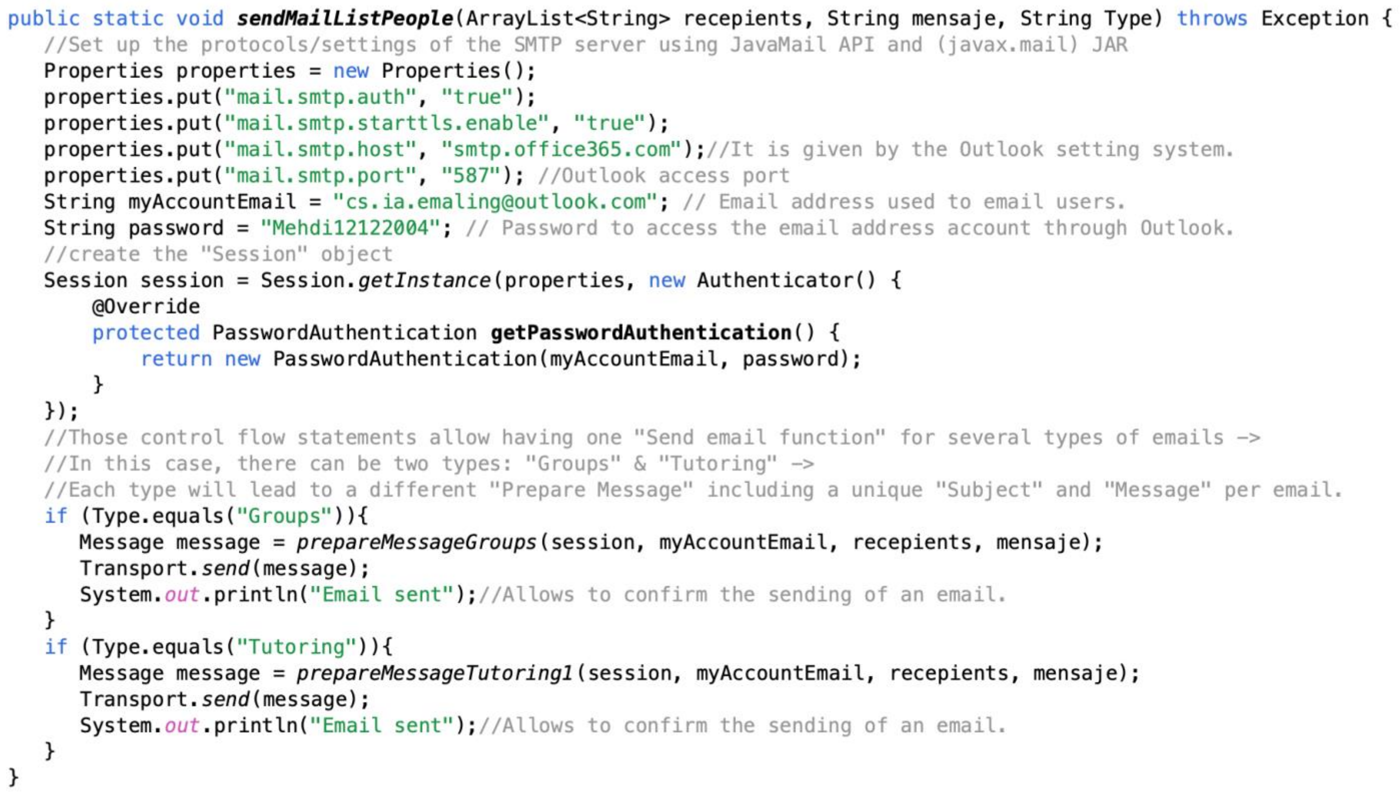
	"' OR Groups.Subject2 == '" + B + "' OR Groups.Subject2 == '" + C + "' OR Groups.Subject3 == '" + A + "' OR Groups.Subject3 == '" + B + "' OR Groups.Subject3 == '" + C + "';";
GetAllGroups2Info	"select * from Groups where Groups.UserName == '" +z+ "';"
updateGroupStatusFromStudy2	"UPDATE GroupStatus SET Host = ? , "+ "Status = ? "+ "WHERE UserName = ?"
InsertTutoring	"INSERT INTO Tutoring(UserName, Subject, Description) VALUES(?,?,?);"
EmailsToSendNotificationsTutoring	"select Email from Notifications N join Accounts A on N.UserName = A.UserName join IBSubjects I on I.UserName = A.UserName WHERE N.Tutoring = 1 and (I.Group1 = ? or I.Group1Extra = ? or I.Group2 = ? or I.Group2Extra = ? or I.Group3 = ? or I.Group3Extra = ? or I.Group4= ? or I.Group4Extra = ? or I.Group5 = ? or I.Group6 = ?);";
DeleteGroupExit	"Delete from Groups where Groups.UserName == '" +UserName+ "';"
updateGroupStatusExit	"UPDATE GroupStatus SET Host = ? , "+ "Status = ? " + "WHERE Host = ?"
AllTutoring	"select * from Tutoring;"
GetAllTutoringInfo	"select * from Tutoring where Tutoring.UserName == '" +z+ "';"
GetEmail	"select * from Accounts where Accounts.UserName == '" + UN1 + "';"
GetTutoringemail	"select * from Accounts where Accounts.UserName == '" +z+ "';"
GetPassword	"select * from Accounts where Accounts.UserName == '" + UN1 + "';"
updatePassword	"UPDATE Accounts\n" +"SET Password == ? \n" +"WHERE UserName = ?;"
UpdateUserName	"UPDATE Accounts\n" +"SET UserName == ? \n" + "WHERE UserName = ?;"
updateEmail	"UPDATE Accounts\n" +"SET Email == ? \n" +"WHERE UserName = ?;"
CheckNotificationsGroups	"select * from Notifications where UserName == '" + USERNAME + "' and Groups == '"+1+"';"
CheckNotificationsTutoring	"select * from Notifications where UserName == '" + USERNAME + "' and Tutoring == '"+1+"';"
updateGroupNotifications	"UPDATE Notifications\n" + "SET Groups == ? \n" +"WHERE UserName = ?;";
updateTutoringNotifications	"UPDATE Notifications\n" + "SET Tutoring == ? \n" +"WHERE UserName = ?;";

Table 8: All Queries used.

Emailing system:

1.1. Javax.mail.jar

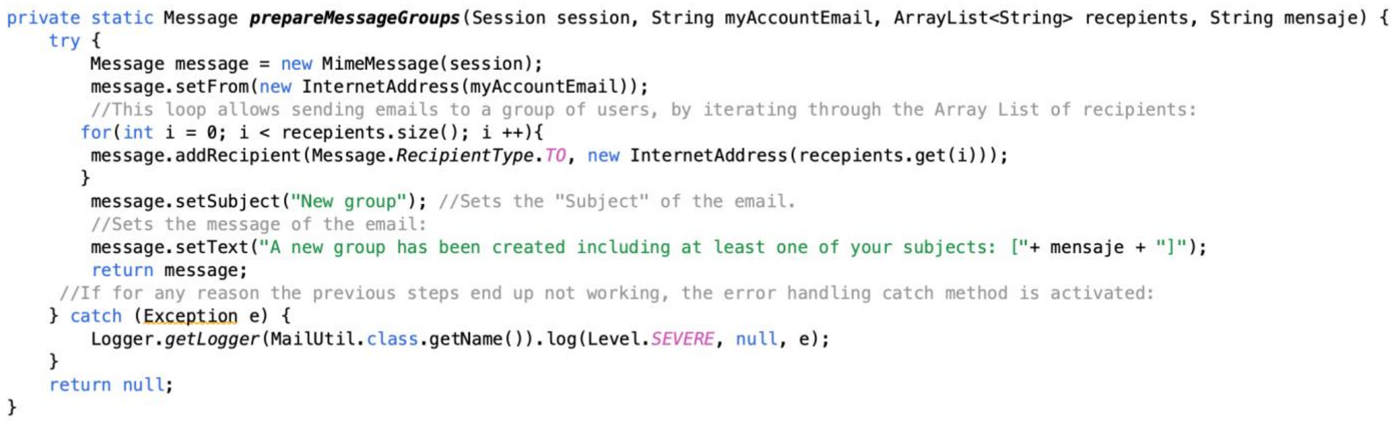
The Java mail library allowed to effectively achieve one of the success criteria of the client. Enabling to send notifications by using email addresses. As it can be referred to in (Criterion A, success criteria: d,e,f,o,p) the user specifically demanded for a fully functional emailing system.



```java
public static void sendMailListPeople(ArrayList<String> recepients, String mensaje, String Type) throws Exception {
   //Set up the protocols/settings of the SMTP server using JavaMail API and (javax.mail) JAR
   Properties properties = new Properties();
   properties.put("mail.smtp.auth", "true");
   properties.put("mail.smtp.starttls.enable", "true");
   properties.put("mail.smtp.host", "smtp.office365.com");//It is given by the Outlook setting system.
   properties.put("mail.smtp.port", "587"); //Outlook access port
   String myAccountEmail = "cs.ia.emaling@outlook.com"; // Email address used to email users.
   String password = "Mehdi12122004"; // Password to access the email address account through Outlook.
   //create the "Session" object
   Session session = Session.getInstance(properties, new Authenticator() {
       @Override
       protected PasswordAuthentication getPasswordAuthentication() {
           return new PasswordAuthentication(myAccountEmail, password);
       }
   });
   //Those control flow statements allow having one "Send email function" for several types of emails ->
   //In this case, there can be two types: "Groups" & "Tutoring" ->
   //Each type will lead to a different "Prepare Message" including a unique "Subject" and "Message" per email.
   if (Type.equals("Groups")){
      Message message = prepareMessageGroups(session, myAccountEmail, recepients, mensaje);
      Transport.send(message);
      System.out.println("Email sent");//Allows to confirm the sending of an email.
   }
   if (Type.equals("Tutoring")){
      Message message = prepareMessageTutoring1(session, myAccountEmail, recepients, mensaje);
      Transport.send(message);
      System.out.println("Email sent");//Allows to confirm the sending of an email.
   }
}
```

Image 17: Emailing system connection.

(**Image-17**) Code used to make a working connection between the program and the outlook emailing system.



```java
private static Message prepareMessageGroups(Session session, String myAccountEmail, ArrayList<String> recepients, String mensaje) {
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(myAccountEmail));
        //This loop allows sending emails to a group of users, by iterating through the Array List of recipients:
       for(int i = 0; i < recepients.size(); i ++){
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(recepients.get(i)));
       }
        message.setSubject("New group"); //Sets the "Subject" of the email.
        //Sets the message of the email:
        message.setText("A new group has been created including at least one of your subjects: ["+ mensaje + "]");
        return message;
     //If for any reason the previous steps end up not working, the error handling catch method is activated:
    } catch (Exception e) {
        Logger.getLogger(MailUtil.class.getName()).log(Level.SEVERE, null, e);
    }
    return null;
}
```

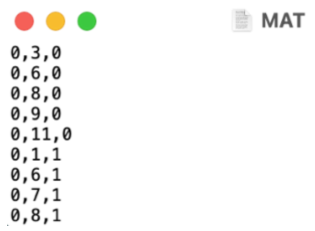
Image 18: Actual message to be sent (Prepare Message)

(**Image-18**) Code used to build a message with specific content and subject.

Intermediate techniques:

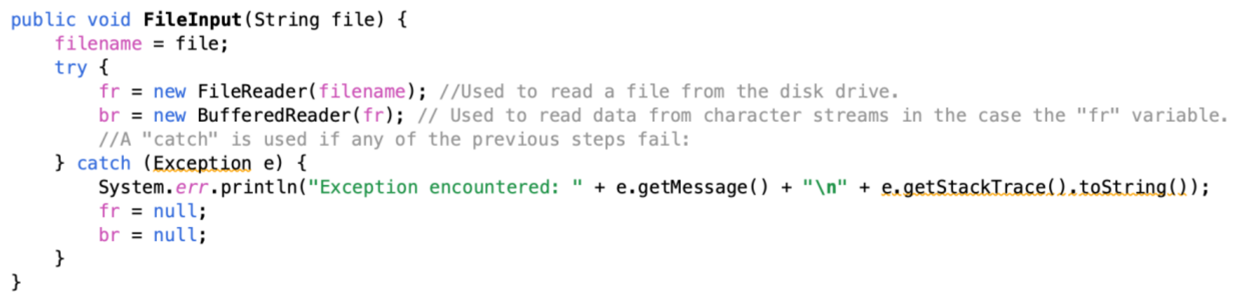
File Parsing

File parsing allowed the storage of the 3-D matrix indexes in a text file. The indexes are supposed to indicate the availability of each room on any given day and hour of the week. This feature avoided storing hundreds of lines of data in the program.



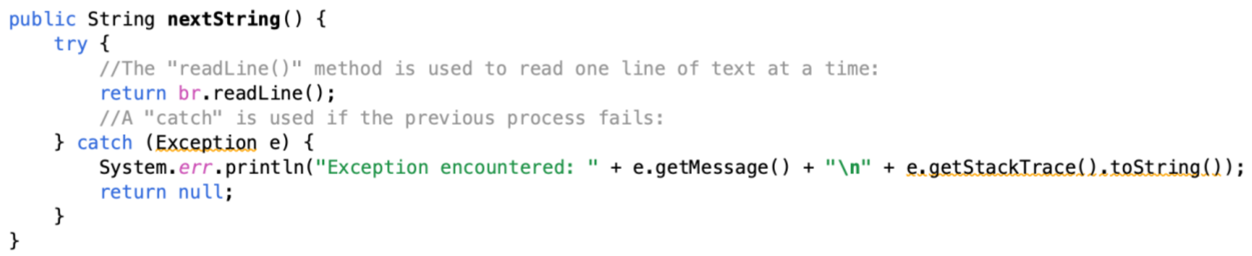
```
0,3,0
0,6,0
0,8,0
0,9,0
0,11,0
0,1,1
0,6,1
0,7,1
0,8,1
```

Image 19: Text file



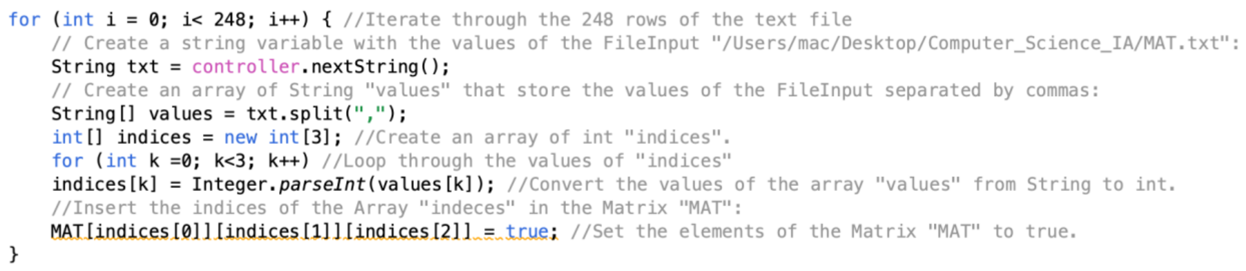
```
public void FileInput(String file) {
    filename = file;
    try {
        fr = new FileReader(filename); //Used to read a file from the disk drive.
        br = new BufferedReader(fr); // Used to read data from character streams in the case the "fr" variable.
        //A "catch" is used if any of the previous steps fail:
    } catch (Exception e) {
        System.err.println("Exception encountered: " + e.getMessage() + "\n" + e.getStackTrace().toString());
        fr = null;
        br = null;
    }
}
```

Image 20: Function used for (txt) file input.



```
public String nextString() {
    try {
        //The "readLine()" method is used to read one line of text at a time:
        return br.readLine();
        //A "catch" is used if the previous process fails:
    } catch (Exception e) {
        System.err.println("Exception encountered: " + e.getMessage() + "\n" + e.getStackTrace().toString());
        return null;
    }
}
```

Image 21: Function used to extract lines from the (txt) file

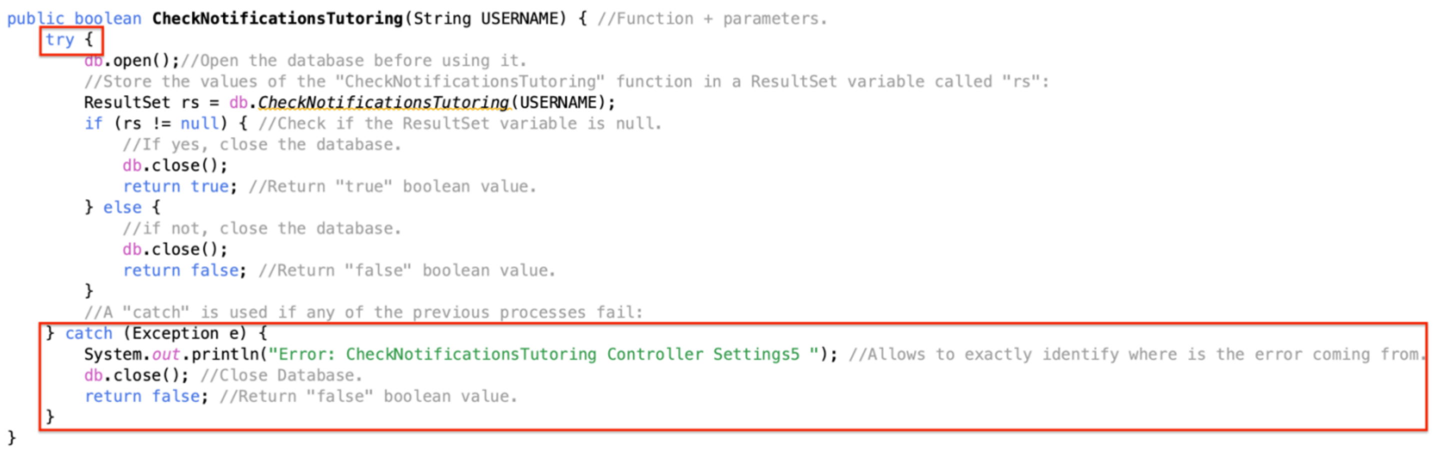


```
for (int i = 0; i< 248; i++) { //Iterate through the 248 rows of the text file
    // Create a string variable with the values of the FileInput "/Users/mac/Desktop/Computer_Science_IA/MAT.txt":
    String txt = controller.nextString();
    // Create an array of String "values" that store the values of the FileInput separated by commas:
    String[] values = txt.split(",");
    int[] indices = new int[3]; //Create an array of int "indices".
    for (int k =0; k<3; k++) //Loop through the values of "indices"
    indices[k] = Integer.parseInt(values[k]); //Convert the values of the array "values" from String to int.
    //Insert the indices of the Array "indeces" in the Matrix "MAT":
    MAT[indices[0]][indices[1]][indices[2]] = true; //Set the elements of the Matrix "MAT" to true.
}
```

Image 22: File parsing algorithm in “AvailableRooms” GUI class.

Error Handling

Error handling processes allow the detection and handling of errors during the run time. They are used to detect, resolve and inform of any error no matter the type. This can be implemented through several coding techniques; however, this project has made use of the try-catch method.



```
public boolean CheckNotificationsTutoring(String USERNAME) { //Function + parameters.
    try {
        db.open();//Open the database before using it.
        //Store the values of the "CheckNotificationsTutoring" function in a ResultSet variable called "rs":
        ResultSet rs = db.CheckNotificationsTutoring(USERNAME);
        if (rs != null) { //Check if the ResultSet variable is null.
            //If yes, close the database.
            db.close();
            return true; //Return "true" boolean value.
        } else {
            //if not, close the database.
            db.close();
            return false; //Return "false" boolean value.
        }
        //A "catch" is used if any of the previous processes fail:
    } catch (Exception e) {
        System.out.println("Error: CheckNotificationsTutoring Controller Settings5 "); //Allows to exactly identify where is the error coming from.
        db.close(); //Close Database.
        return false; //Return "false" boolean value.
    }
}
```

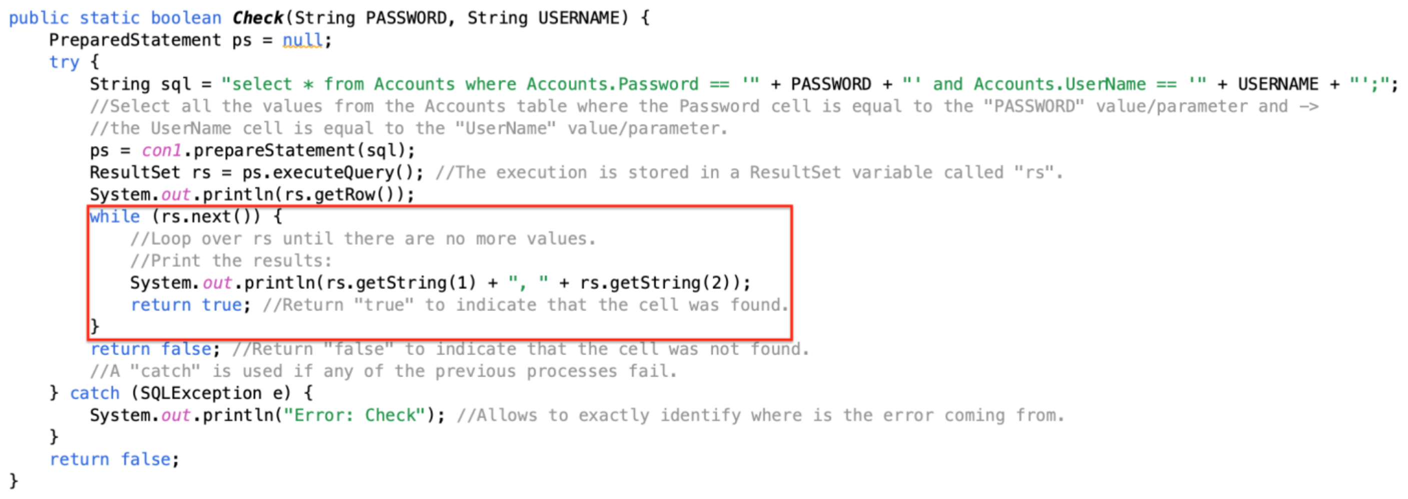
Image 23: Try-catch example in the "Settings5Controller" class.

Basic techniques:

Control flow statements (Loops)

Loops have been used to iterate through the varied data structures used in this project. Different control flow statements were chosen for efficiency and simplicity reasons:

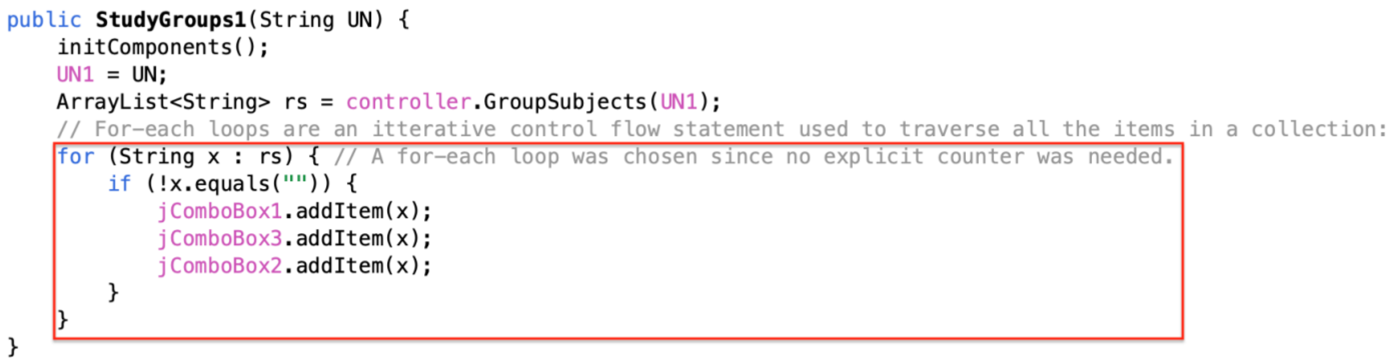
1.1. While loops:



```
public static boolean Check(String PASSWORD, String USERNAME) {
    PreparedStatement ps = null;
    try {
        String sql = "select * from Accounts where Accounts.Password == '" + PASSWORD + "' and Accounts.UserName == '" + USERNAME + "';";
        //Select all the values from the Accounts table where the Password cell is equal to the "PASSWORD" value/parameter and ->
        //the UserName cell is equal to the "UserName" value/parameter.
        ps = con1.prepareStatement(sql);
        ResultSet rs = ps.executeQuery(); //The execution is stored in a ResultSet variable called "rs".
        System.out.println(rs.getRow());
        while (rs.next()) {
            //Loop over rs until there are no more values.
            //Print the results:
            System.out.println(rs.getString(1) + ", " + rs.getString(2));
            return true; //Return "true" to indicate that the cell was found.
        }
        return false; //Return "false" to indicate that the cell was not found.
        //A "catch" is used if any of the previous processes fail.
    } catch (SQLException e) {
        System.out.println("Error: Check"); //Allows to exactly identify where is the error coming from.
    }
    return false;
}
```

Image 24: While loop in "DbConnection" class.

1.2. For-each loops:

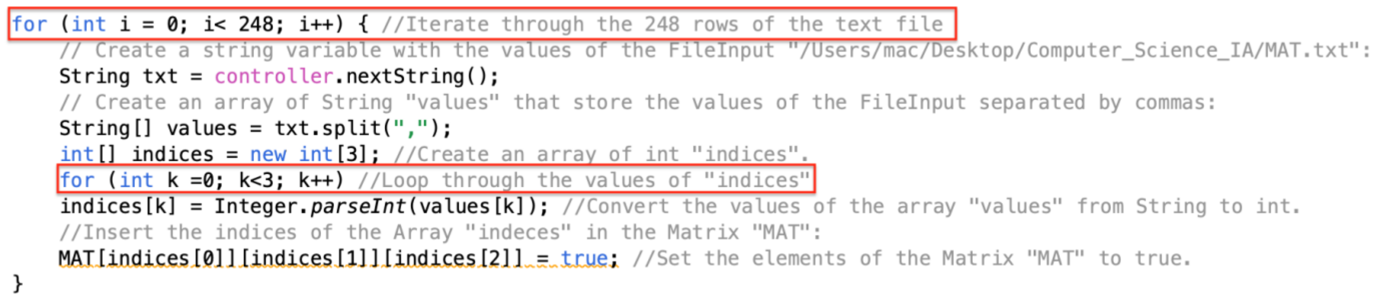


```
public StudyGroups1(String UN) {
    initComponents();
    UN1 = UN;
    ArrayList<String> rs = controller.GroupSubjects(UN1);
    // For-each loops are an itterative control flow statement used to traverse all the items in a collection:
    for (String x : rs) { // A for-each loop was chosen since no explicit counter was needed.
        if (!x.equals("")) {
            jComboBox1.addItem(x);
            jComboBox3.addItem(x);
            jComboBox2.addItem(x);
        }
    }
}
```

Image 25: For-each-loop in “StudyGroups1” GUI class.

1.3. Nested for-Loops:

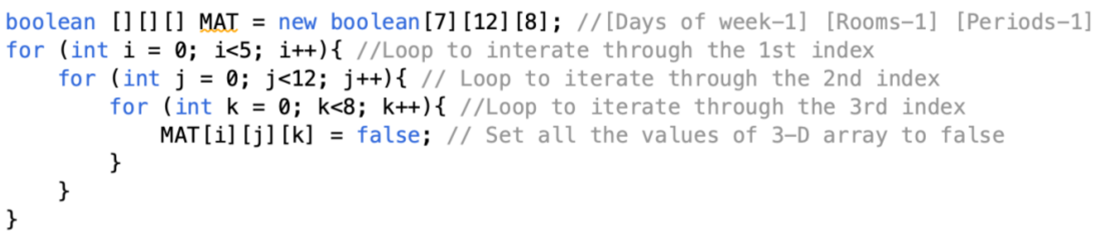
Double nested loops:



```
for (int i = 0; i< 248; i++) { //Iterate through the 248 rows of the text file
    // Create a string variable with the values of the FileInput "/Users/mac/Desktop/Computer_Science_IA/MAT.txt":
    String txt = controller.nextString();
    // Create an array of String "values" that store the values of the FileInput separated by commas:
    String[] values = txt.split(",");
    int[] indices = new int[3]; //Create an array of int "indices".
    for (int k =0; k<3; k++) //Loop through the values of "indices"
    indices[k] = Integer.parseInt(values[k]); //Convert the values of the array "values" from String to int.
    //Insert the indices of the Array "indeces" in the Matrix "MAT":
    MAT[indices[0]][indices[1]][indices[2]] = true; //Set the elements of the Matrix "MAT" to true.
}
```

Image 26: (Double) nested for-loops in “AvailableRooms” GUI class.

Triple nested loops:



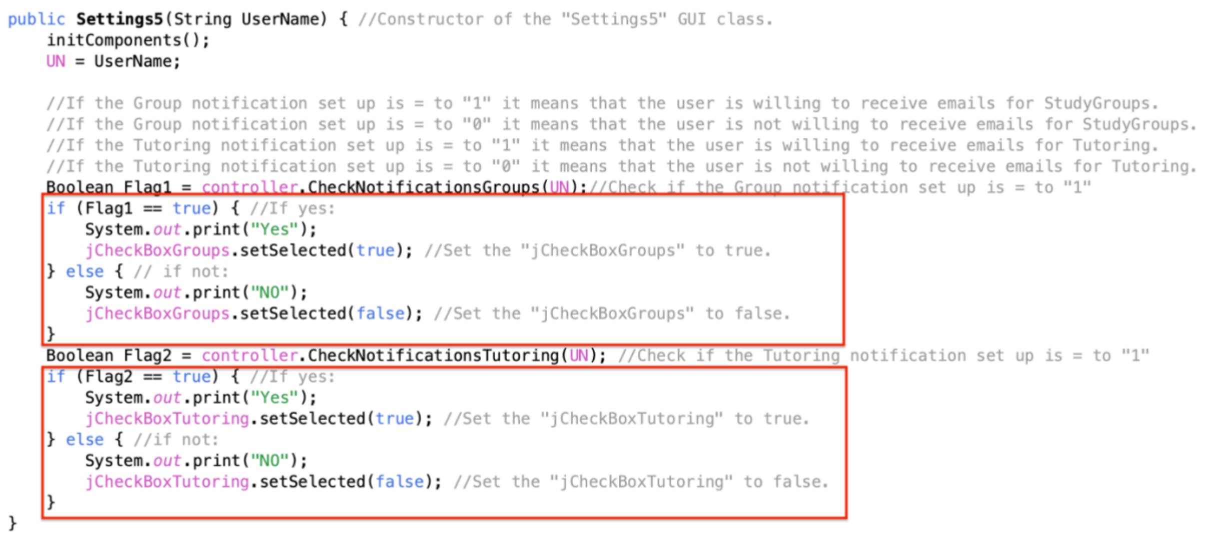
```
boolean [][][] MAT = new boolean[7][12][8]; //[Days of week-1] [Rooms-1] [Periods-1]
for (int i = 0; i<5; i++){ //Loop to interate through the 1st index
    for (int j = 0; j<12; j++){ // Loop to iterate through the 2nd index
        for (int k = 0; k<8; k++){ //Loop to iterate through the 3rd index
            MAT[i][j][k] = false; // Set all the values of 3-D array to false
        }
    }
}
```

Image 27: (Triple) nested for-loops in “AvailableRooms” GUI class.

Control flow statements (Conditionals)

I have made use of conditionals to control the flow of information. This project has made use of two methods, the (if-statement) & (Switch-case):

1.1. If statements

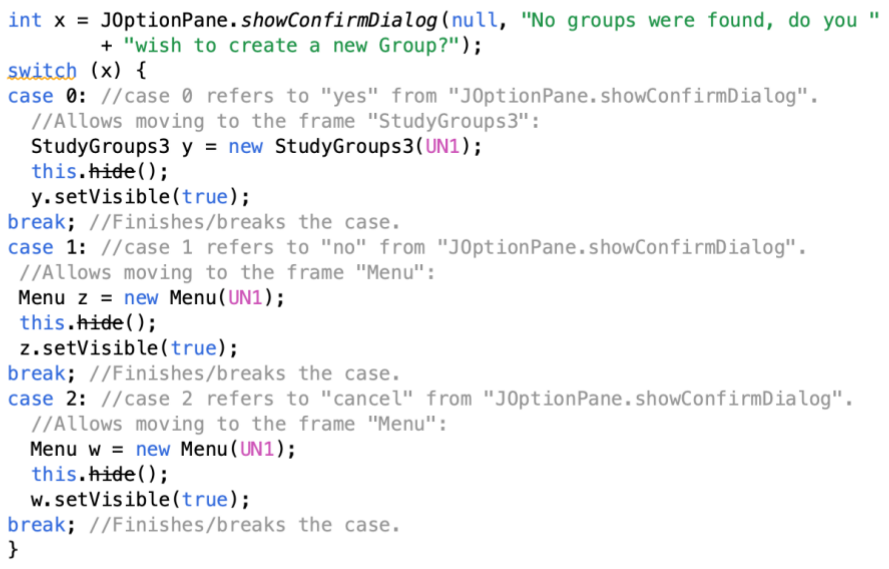


```
public Settings5(String UserName) { //Constructor of the "Settings5" GUI class.
    initComponents();
    UN = UserName;

    //If the Group notification set up is = to "1" it means that the user is willing to receive emails for StudyGroups.
    //If the Group notification set up is = to "0" it means that the user is not willing to receive emails for StudyGroups.
    //If the Tutoring notification set up is = to "1" it means that the user is willing to receive emails for Tutoring.
    //If the Tutoring notification set up is = to "0" it means that the user is not willing to receive emails for Tutoring.
    Boolean Flag1 = controller.CheckNotificationsGroups(UN);//Check if the Group notification set up is = to "1"
    if (Flag1 == true) { //If yes:
        System.out.print("Yes");
        jCheckBoxGroups.setSelected(true); //Set the "jCheckBoxGroups" to true.
    } else { // if not:
        System.out.print("NO");
        jCheckBoxGroups.setSelected(false); //Set the "jCheckBoxGroups" to false.
    }
    Boolean Flag2 = controller.CheckNotificationsTutoring(UN); //Check if the Tutoring notification set up is = to "1"
    if (Flag2 == true) { //If yes:
        System.out.print("Yes");
        jCheckBoxTutoring.setSelected(true); //Set the "jCheckBoxTutoring" to true.
    } else { //if not:
        System.out.print("NO");
        jCheckBoxTutoring.setSelected(false); //Set the "jCheckBoxTutoring" to false.
    }
}
```

Image 28: If and else statements in “Settings5” GUI class.

1.2. Switch case



```
int x = JOptionPane.showConfirmDialog(null, "No groups were found, do you "
        + "wish to create a new Group?");
switch (x) {
case 0: //case 0 refers to "yes" from "JOptionPane.showConfirmDialog".
  //Allows moving to the frame "StudyGroups3":
  StudyGroups3 y = new StudyGroups3(UN1);
  this.~~hide~~();
  y.setVisible(true);
break; //Finishes/breaks the case.
case 1: //case 1 refers to "no" from "JOptionPane.showConfirmDialog".
 //Allows moving to the frame "Menu":
 Menu z = new Menu(UN1);
 this.~~hide~~();
 z.setVisible(true);
break; //Finishes/breaks the case.
case 2: //case 2 refers to "cancel" from "JOptionPane.showConfirmDialog".
  //Allows moving to the frame "Menu":
  Menu w = new Menu(UN1);
  this.~~hide~~();
  w.setVisible(true);
break; //Finishes/breaks the case.
}
```

Image 29: Switch statement in “StudyGroups1” GUI class

Extensibility:

Factors of extensibility	**Details**	**Technique(s)**
Controller classes	By splitting the GUI and non-GUI-related code, future maintenance/updates will render easier via a more readable code infrastructure.	Modularity & instantiation
Database DBConnection class	By storing all the database interaction/access functions in a single global class, it becomes easier to make use of them anywhere in the code. For future implementations, this will promote fast and safe coding via the use of already-tested (working) functions.	Instantiation
Packaging	All classes have been separated into given packages. This should simplify the understanding of code since future developers will be able to differentiate between groups (GUI, Controllers, Emailing, Database, Main).	Modularity & Importing
Encapsulation	By storing given variables under the access modifier public or private, variables can be protected from unexpected future human errors during maintenance or updates.	Encapsulation & Access modifiers.
Names: variables & methods & classes	Names of variables/methods/classes have been carefully chosen to be self-explainable. By doing so, future developers will be able to understand the end goal of a method by simply looking at its name.	N/A
Commented code	The entire project has been clearly and fully commented on to ease the understanding of the code. This will be of special relevance for future developers wishing to add additional functionalities or simply update a given feature.	N/A
Emailing	A specific email address has been created for the product. This will allow future developers to use a dedicated Outlook account specifically made for the required tasks. "*cs.ia.emailing@outlook.com*"	Imports & Javax.mail.jar library

Table 9: Extensibility factors.

Word Count: 1120

Bibliography

Bensos Frequents. (n.d.). YouTube. https://www.youtube.com/c/HackSmile/videos

Bhandari, S. (2021, June 23). *Three dimensional (3D) array in C*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/3d-array-in-c/

Bodnar, J. (n.d.). *Object-oriented programming in Java - OOP concepts*. https://zetcode.com/java/oop/

Exception: java.lang.ClassNotFoundException: javax.activation.DataHandler even though javax.mail.jar is under classpath? (2020, January 10). Stack Overflow. https://stackoverflow.com/questions/59675176/exception-java-lang-classnotfoundexception-javax-activation-datahandler-even-t

GeeksforGeeks. (2022, January 10). *Difference between AWT and Swing in Java*. https://www.geeksforgeeks.org/difference-between-awt-and-swing-in-java/

Genuine Coder. (2019, March 9). *Java - Send Email from Java Program - Java Mail API - (Gmail Example - 2019)*. YouTube. https://www.youtube.com/watch?v=A7HAB5whD6I

Get Day Number of Week in Java. (n.d.). https://www.tutorialspoint.com/get-day-number-of-week-in-java

getting integer values from textfield. (2013, January 5). Stack Overflow. https://stackoverflow.com/questions/14169240/getting-integer-values-from-textfield

Great Learning Team. (2022, November 23). *Polymorphism in Java with Examples – 2023*. Great Learning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/polymorphism-in-java/

how to remove brackets character in string (java). (2014, September 15). Stack Overflow. https://stackoverflow.com/questions/25852961/how-to-remove-brackets-character-in-string-java

Java - Encapsulation. (n.d.). https://www.tutorialspoint.com/java/java_encapsulation.htm

Java ArrayList. (n.d.). https://www.w3schools.com/java/java_arraylist.asp

Java y Otros. (2021, September 27). *Enviar correos en Java desde cuenta de hotmail/outlook (Respuesta comentario)*. YouTube. https://www.youtube.com/watch?v=L_QCGUkVmUg

Mr Teacher Wachs. (2022, March 16). *I.B. Computer Science Internal Assessment (IA) - Part 1 (Section A) Example Walk Through*. YouTube. https://www.youtube.com/watch?v=q_Vue2cABSI

NIMAP INFOTECH. (2022, December 10). *Advantages of Java & Disadvantages | Pros & Cons of Java -* . https://nimapinfotech.com/blog/advantages-and-disadvantages-of-java/

Pedamkar, P. (2022, June 6). *Java Swing vs Java FX*. EDUCBA. https://www.educba.com/java-swing-vs-java-fx/

Ramos, J. (n.d.). *¿Cómo funciona INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN?* Programación Y Más. https://programacionymas.com/blog/como-funciona-inner-left-right-full-join

Static Types vs Dynamic Types. Stop fighting and make my life easier already | Instil. (n.d.). Insights by Instil. https://instil.co/blog/static-vs-dynamic-types/

Wikipedia contributors. (2022, October 6). *Foreach loop*. Wikipedia. https://en.wikipedia.org/wiki/Foreach_loop