|  |
| --- |
| Microsoft |
| I.T. P.A.T. |
| System Technical Document |
|  |
| **Jayden Maree** |
| **Glenwood House** |

Table of Contents

[*Externally Sourced Code* 2](#_Toc491953381)

[ShutDownHook 2](#_Toc491953382)

[Cell Phone Number Verification 2](#_Toc491953383)

[Underlining Clickable Labels 2](#_Toc491953384)

[Parsing MS Access Dates from Strings to Dates. 2](#_Toc491953385)

[Mouse Action Listeners 2](#_Toc491953386)

[*Explanation of Critical Algorithms* 3](#_Toc491953387)

[Section A 3](#_Toc491953388)

[Section B 3](#_Toc491953389)

[Section C 4](#_Toc491953390)

[Section D 4](#_Toc491953391)

[Section E 4](#_Toc491953392)

[Explanation of Sections A, B, C, D and E 5](#_Toc491953393)

[*Advanced Techniques* 6](#_Toc491953394)

[HTML Usage 6](#_Toc491953395)

# *Externally Sourced Code*

## ShutDownHook

Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {

// The shutdownhook makes sure that any non complete registrations are void.

// It also closes the database connection.

// It was sourced online.

public void run() {

try {

if(!storage.isPopulated(username)){

storage.deleteUser(username);

}

storage.closeConnection();

} catch (SQLException | IOException e) {

storage.closeConnection();

//e.printStackTrace();

//expected to catch errors if system is closed during LogIn

}

}

}, "Shutdown-thread"));

## Cell Phone Number Verification

else if (!cellphone.matches("\\d+")) {//learnt online, checks that field contains digits only

MasterFrameGUI.getTutProfScrn().setLblError("Please enter your cellphone number using only numbers 0-9.");

}

## Underlining Clickable Labels

Font font = lblViewProfile.getFont();

Map attributes = font.getAttributes();//Had to google this

attributes.put(TextAttribute.UNDERLINE, TextAttribute.UNDERLINE\_ON);// and this.

lblViewProfile.setFont(font.deriveFont(attributes));

## Parsing MS Access Dates from Strings to Dates.

// This object can interpret strings representing dates in the format MM/dd/yyyy

DateFormat dF = new SimpleDateFormat("yyyy-MM-dd"); //learnt online

//parses from String to Date

Date date = dF.parse(databaseDate);

## Mouse Action Listeners

usernameField.addMouseListener(new MouseAdapter(){//makes the field white and clears errors.

@Override

public void mouseClicked(MouseEvent arg0) {//had to google mouse action listeners initially

usernameField.setBackground(new Color(255,255,255));

lblError.setText("");

}

});

contentPane.add(usernameField);

# *Explanation of Critical Algorithms*

## Section A

//The insert statement makes use of the timetableIDAutoNumber primary key and inserts an empty array.

public void insertUserTable(String username,JTable timetable) throws SQLException{

String [] weekDay = {"Monday","Tuesday","Wednesday","Thursday","Friday"};

for (int i = 0; i < timetable.getRowCount(); i++) {

statement.executeUpdate("INSERT INTO tblTimetable(TimetableID,WeekDay) Values "+ "('"+timetableIDAutoNumber+"','"+weekDay[i]+"');");

}

statement.executeUpdate("UPDATE tblUser\nSET TimetableID = '"+ timetableIDAutoNumber +"' WHERE Username = '"+username+"';");

timetableIDAutoNumber++;

}

## Section B

//The update is used to change any already inserted timetable.

public void updateUserTable(String username,JTable timetable) throws SQLException{

int timetableID = getTimetableID(username);

for (int i = 0; i < timetable.getRowCount(); i++) {

for (int j = 1; j < timetable.getColumnCount(); j++) {

statement.executeUpdate("UPDATE tblTimetable\nSET ["+timetable.getColumnName(j)+"] = '"+ timetable.getValueAt(i, j) +"' WHERE WeekDay = '"+timetable.getValueAt(i, 0)+"' AND TimetableID = '"+timetableID+"';");

}

}

}

## Section C

//inserts a tutor or student subject list and gives them an appropriate ID.

public void insertUserSubjectList(String username, String[] subjectArray) throws SQLException{

for (int i = 0; i < subjectArray.length; i++) {

statement.executeUpdate("INSERT INTO tblSubjects(SubjectID,Subject) Values ('"+subjectsIDAutoNumber+"','"+subjectArray[i]+"');");

}

statement.executeUpdate("UPDATE tblUser SET SubjectID = '"+ subjectsIDAutoNumber +"' WHERE Username = '"+username+"';");

subjectsIDAutoNumber++;

}

## Section D

//deletes the subject records for the user but not their ID. The ID is deleted in the "deleteUser" method

public void deleteUserSubjectList(String username) throws SQLException{

int subjectID = getSubjectID(username);

statement.executeUpdate("DELETE \* FROM tblSubjects WHERE SubjectID = '"+subjectID+"';");

}

## Section E

//these resultSets instantiate the autoNumbers which act as primary key's in the 'PAT Database'.

resultSet = statement.executeQuery("SELECT MAX(TimetableID)FROM tblUser;");

resultSet.next();

timetableIDAutoNumber = resultSet.getInt(1)+1;

resultSet = statement.executeQuery("SELECT MAX(SubjectID)FROM tblUser;");

resultSet.next();

subjectsIDAutoNumber = resultSet.getInt(1)+1;

resultSet = null;

Ucan access dates

## Explanation of Sections A, B, C, D and E

Section E initialises the declared integer fields ‘timetableIDAutoNumber’ and ‘subjectsIDAutoNumber’. These values act as the foreign keys in the table ‘tblUser’ in the database. Each user will need unique ID values and this is achieved by using the ‘MAX’ function. Using ‘MAX’ allows for unique ID values since the values are higher than the highest ID values found in tblUser at the start of the runtime session.

These values are used directly in sections A and C when data gets inserted into the table,

These values are called again from the database, for example in sections B and D they are called to update and delete user timetable data or subject data.

These algorithms are thus critical to the data storage of user subjects and timetables and allows for data to be stored uniquely and in a normalized fashion (since they allow for a one to many relationship between tables in the database).Thus Section E can be seen as a critical Algorithm.

# *Advanced Techniques*

## HTML Usage

//creates the setter that allows users to see errors they have made.

public void setLblError(String lbllError) {//HTML used here to get JLabel to wrap tight around text.

this.lblError.setText("<html><p>"+lbllError+"</p></html>");

}

Using HTML in the setting methods of various labels as shown above, allows the JLabel to wrap around the text dynamically based on the size of the content. This is convenient for changing error messages since the JLabel size can adapt to the message size. It also allows for certain HTML breaks to be used as shown below, which enhance the formatting of the JLabel.

public void setCDLabel(String cDlabel){

this.lblWarning.setText("<html><p><center>"+cDlabel+"</center></p></html>");

}

MasterFrameGUI.getCD().setCDLabel(

"Performing this action causes you to overwrite your current account details.<br><br>"

+ "Are you sure you want to continue?");