Criterion C: Development

INTRODUCTION:
The product is a Java product, with a main function to borrow and return equipment, by which the amount of equipment is defined by the main user. It is also able to alert users to return their equipment when triggered.

Techniques Used:

- **Built-in Objects**
- **Use of Database**
- **Parameterized Query**
- **Handling Exception by Try/Catch**
- **User-defined Objects**
- **User-defined Methods**
- **Simple and Compound Selection**
- **Encapsulation**
- **ArrayList**
- **Loops**
- **EventQueue**
- **Inheritance & GUI**
- **Additional Libraries**

Program Structure

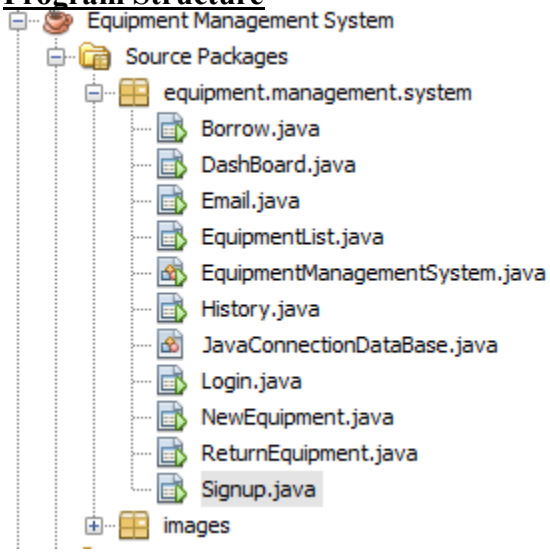

Figure 1.1 - All classes in the program.

Since there are a variety of different functions are required in my program, I decided to divide each functionality into each class and then connect and encapsulate their logic altogether. There are not a lot of code to maintain and use at the same time, thus making it easier for me to fix any errors in the code and test each function out without interfering with other sections of the program.

Built-In Objects

```
13  /**
14   *
15   * @author Bao Son
16   */
17  public class Login extends javax.swing.JFrame {
18      Connection connection;
19      ResultSet resultSet;
20      PreparedStatement preparedStatement;
21      /**
22       * Creates new form Login
23       */
24      public Login () {
25          initComponents();
26          connection = JavaConnectionDataBase.connectToDatabase();
27      }
```

Figure 1.2 - Built-In Objects used.

An object connection is first created in order to create a connection to the database. A preparedStatement object is created here to precompile SQL commands with parameters, where it stores both SQL commands and the precompiled commands. This means that the database can run the statement without the need to compile it. Executing this instead of many Statement objects can reduce time, and also the command can be sent to the database immediately.

Use of Database

In order to communicate and connect with Microsoft SQL Server database, SQL (Structured Query Language) is used. (Docs.oracle.com, n.d.) (Tutorialspoint.com, n.d.) (Docs.microsoft.com, 2019)

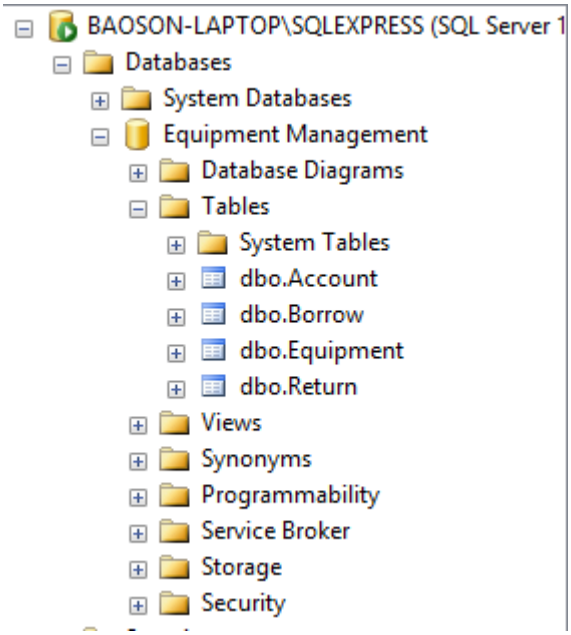

Figure 1.3 - All database tables used.

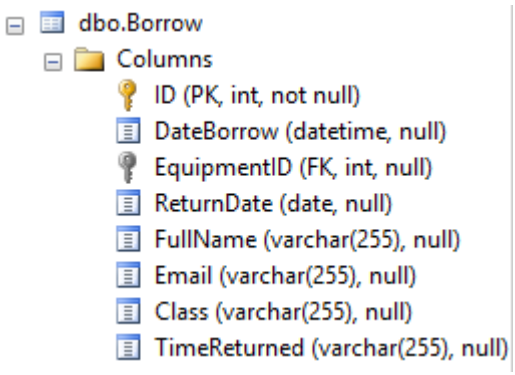

Figure 1.4 - All columns in table Borrow.

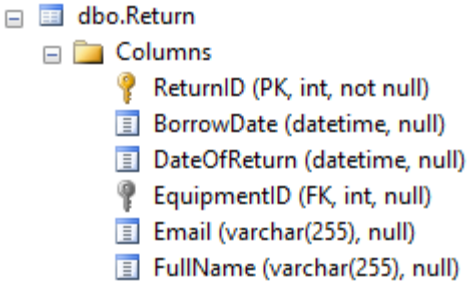

Figure 1.5 - All columns in table Return.

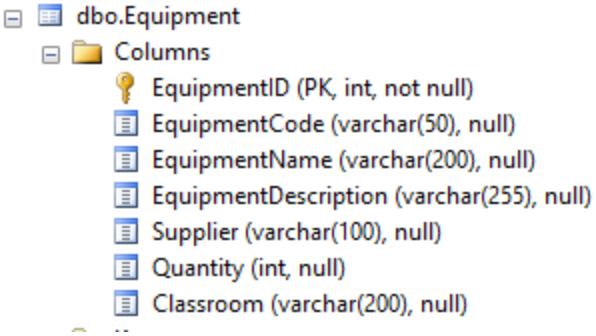

Figure 1.6 - All columns in table Equipment.

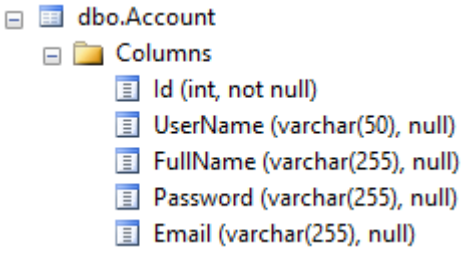

Figure 1.7 - All columns in table Account.

```
        connection = JavaConnectionDataBase.connectToDatabase();
        String sqlEquipmentList = "SELECT EquipmentName AS 'Equipment Name', EquipmentCode AS 'Equipment Code', Quantity,\n" +
"'Supplier' AS 'From Teacher', Classroom FROM Equipment";
```

Figure 1.8 - SELECT query used

```
String SqlBorrowList = "SELECT Borrow.FullName AS \"Full Name\", Borrow.Email AS \"Email\", Equipment.EquipmentName AS \"Equipment Name\","
        + " Equipment.Quantity, Borrow.DateBorrow AS \"Borrow Date\"\n" +
                    "FROM Borrow \n" +
                    "INNER JOIN Equipment ON Borrow.EquipmentID = Equipment.EquipmentID ORDER BY [Borrow Date] DESC";
```

Figure 1.9 - SELECT from table Borrow and ORDER BY the borrow date.

```
String sql = "SELECT * FROM Account Where UserName=? And Password=?";
```

Figure 1.10 - SELECT statement to select all columns from table Account, the WHERE clause limits the return data through two conditions: Username and Password. This satisfies criteria 4 where the client wants a login system.

```
String sqlInsertNewEquipment = "INSERT INTO Equipment(EquipmentCode, EquipmentName, "
        + "EquipmentDescription, Quantity, Supplier, Classroom) VALUES (?,?,?,?,?,?)";
```

Figure 1.11 - INSERT rows into the table Equipment with corresponding inputs.

```
String sqlSearchStudentEmailBorrowEquipment = "SELECT * FROM Borrow \n" +
                                              "  INNER JOIN Equipment ON Borrow.EquipmentID = Equipment.EquipmentID\n" +
                                              "  WHERE Borrow.Email = ? ORDER BY Borrow.DateBorrow DESC";
```

Figure 1.12 - SELECT from table Borrow joining with table Equipment.

```
String sqlDelete = "DELETE FROM Borrow WHERE Email = ? AND EquipmentID = ?";
```

Figure 1.13 - Remove a row in the table Borrow, the WHERE clause limit it to the row with the corresponding Email and EquipmentID inputted by the user.

```
sqlUpdate = "INSERT INTO [Return] (Email, FullName, EquipmentID, BorrowDate, DateOfReturn)  VALUES (?, ?, ?, ?, ?)";
```

Figure 1.14 - INSERT statement to add the information of the borrowing session into the Return table.

Parameterized Query

```
    private void BorrowButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String emailValidate =
    "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
    + "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";
        if (EmailTextField.getText().matches(emailValidate)) {
            String sqlInsertBorrow = "INSERT INTO Borrow(EquipmentID, FullName \n"
                + ", Class, Email, TimeReturned, ReturnDate, DateBorrow) VALUES (?, ?, ?, ?, ?, ?, ?)";
            try {
            preparedStatement = connection.prepareStatement(sqlInsertBorrow);
            preparedStatement.setString(1, EquipmentIDTextField.getText());
            preparedStatement.setString(2, FullNameTextField.getText());
            preparedStatement.setString(3, ClassTextField.getText());
            preparedStatement.setString(4, EmailTextField.getText());
            preparedStatement.setString(5, (String) TimeReturnedCombobox.getSelectedItem());
            preparedStatement.setString(6, ReturnDateCombobox.getText());
            preparedStatement.setString(7, BorrowDateCombobox.getText());
            preparedStatement.execute();
            JOptionPane.showMessageDialog(null, "Equipment Borrowed Successfully");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
        }
    }
        else {
            JOptionPane.showMessageDialog(null, "Please enter a proper email");
        }
            }
```

Figure 1.15 - The process of borrowing a piece of equipment.

Position parameters are used and the values are supplied by what the user types in the textbox. The values only need to be identified right before the execution of the query which allows the customization of which records were to be affected during runtime. This is quite beneficial in terms of performance instead of assigning which values to go into each column one by one. (query? and Parker, 2011)

Handling Exception by Try/Catch

Data handling by using try/catch is utilized across the whole program. This is because almost every class requires a connection to the database, and since a problem with the connection might be likely to happen, with the try and catch block it will allow the error within the block to be caught when executing, making it easier for me to debug and fix the problem. Logger.getLogger in the Figure below is a method to create a new logger to log info messages.

```
28      public EquipmentList() {
29          super("Equipment List");
30          initComponents();
31          //Connect to Database
32          connection = JavaConnectionDataBase.connectToDatabase();
33          String sqlEquipmentList = "SELECT EquipmentName AS 'Equipment Name', EquipmentCode AS 'Equipment Code', Quantity,\n" +
34          "'Supplier' AS 'From Teacher', Classroom FROM Equipment";
35
36          try { //proceed with connection, put in try because there might be a connection error
37              preparedStatement = connection.prepareStatement(sqlEquipmentList); //this create a query from connection
38              resultSet = preparedStatement.executeQuery();
39              //execute queries above, the result of the execution is stored in resultSet
40              EquipmentListTable.setModel(DbUtils.resultSetToTableModel(resultSet));
41              //assign and convert resultSet to Table Model
42          } catch (SQLException ex) {
43              //if there is an error in connection then the error will be caught and displayed
44              Logger.getLogger(EquipmentList.class.getName()).log(Level.SEVERE, null, ex);
45          }
46      }
```

Figure 1.16 - Utilizing the try/catch error handling block in case of an error in the connection to the database.

User-defined Objects:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    NewEquipment newEquipment = new NewEquipment();
    newEquipment.setVisible(true);
}
```

Figure 1.17 - What happened when a button is clicked on in the dashboard.

Since I separated my program into different classes, I decided to instantiate an object that stands for the designated class itself every time that a class (or GUI of that class) needs to be called upon. I then initiate the object and set it to be visible so it can be displayed on the screen.

User-defined Methods

```
16      public static Connection connectToDatabase() {
18          Connection connection = null;
19          ResultSet resultSet;
20          PreparedStatement preparedStatement;
21          try {
22              //a configuration string to store information of database
23              String dbUrl = "jdbc:sqlserver://BAOSON-LAPTOP\\BAOSONSQL;databaseName=Equipment Management;user=sa;password=123456";
24              Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");//a driver to help with the connection to database
25              connection = DriverManager.getConnection(dbUrl);
26              //create a connection from the information from string dbUrl
27              if (connection != null) {
28                  DatabaseMetaData dm = (DatabaseMetaData) connection.getMetaData();
29                  System.out.println("Driver name: " + dm.getDriverName());
30                  System.out.println("Driver version: " + dm.getDriverVersion());
31                  System.out.println("Product name: " + dm.getDatabaseProductName());
32                  System.out.println("Product version: " + dm.getDatabaseProductVersion());
33              }
35              return connection;
36          } catch (Exception e) {
37              JOptionPane.showMessageDialog(null, e);
38              return null;
39          }
40      }
```

Figure 1.18 - A static method created in class JavaConnectionDatabase to use in other classes, it can be invoked later by creating an object to access it or instance of the class.

```
20  public class EquipmentManagementSystem {
21
22      /**
23       * @param args the command line arguments
24       */
25      public static void main(String[] args) {
26          // TODO code application logic here
27         Login a = new Login();
28         JavaConnectionDataBase jcD = new JavaConnectionDataBase();
           jcD.connectToDatabase();
           if (jcD.connectToDatabase() == null) {
31             JOptionPane.showMessageDialog(null, "Cannot connect to database");
32          }
33          else {
34             a.setVisible(true);
35          }
36      }
37  }
```

Figure 1.19 - This class initializes when the program starts.

This is an example of how the method connectToDatabase is used. The class JavaConnectionDatabase is instantiated which allows the properties of that class to be accessed in the class EquipmentManagementSystem, therefore allowing the method to be accessed here. The *new* operator instantiates the class and invokes a constructor. The constructor's name gives the class's name to instantiate. This is applied to other classes that require a connection to the database as well, which makes it more efficient as I don't need to create a new connection for every class.

```
28      public void SendMail() {
29      Properties prop = System.getProperties();
30          prop.put("mail.smtp.host", "smtp.gmail.com");
31          prop.put("mail.smtp.port", "465");
32          prop.put("mail.smtp.auth", "true");
33          prop.put("mail.smtp.socketFactory.port", "465");
34          prop.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
35
36
37
            Session session = null;
39          session = Session.getInstance(prop, new javax.mail.Authenticator() {
41              protected PasswordAuthentication getPasswordAuthentication() {
42                  return new PasswordAuthentication(GmailAccountEmail, GmailPassword);
43              }
              });
44      try{
45       MimeMessage message;
46          message = new MimeMessage(session);
47
48      message.setFrom(new InternetAddress(GmailAccountEmail));
49
50      message.addRecipient(Message.RecipientType.TO,
51                           new InternetAddress(jTextField1.getText()));
52
53      message.setSubject("Your Borrowed Item is Due In TODAY");
54
55      message.setText("Dear Student \n The item you have borrowed is expected by Ms. Goppert to be return today! Please return it on time");
56
```

```
57          Transport.send(message);
58          JFrame frame = new JFrame("JOptionPane showMessageDialog example");
59          JOptionPane.showMessageDialog(frame,
60             "Email Sent Successful ",
61             "",
62             JOptionPane.INFORMATION_MESSAGE);;
63        }catch (MessagingException mex) {
64         mex.printStackTrace();
65        }
66      }
```

Figure 1.20 - Method to send an email to the students.

```
    private void EmailSendButtonActionPerformed(java.awt.event.ActionEvent evt) {
    SendMail();
    setVisible(false);
    }
```

Figure 1.21 - The method is called upon in the same class.

I also defined another method called SendMail() which is used to set up the properties and session required to be able to send an email to alert the student, along with the details of the email itself by instantiating an empty object from class MimeMessage and fill it with desired content. This function satisfies criteria 3. As I was thinking I might be considering setting up the function multiple times within the class, I decided to put it into a method to save time as you can see in the figure above the code takes up a lot of lines and space.

Encapsulation

```
24  public class Email extends javax.swing.JFrame {
25     private final String GmailAccountEmail = "baosonprogramtest@gmail.com";
26     private final String GmailPassword = "programtest123";
27
28      public void SendMail() {
29      Properties prop = System.getProperties();
30          prop.put("mail.smtp.host", "smtp.gmail.com");
31          prop.put("mail.smtp.port", "465");
32          prop.put("mail.smtp.auth", "true");
33          prop.put("mail.smtp.socketFactory.port", "465");
34          prop.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
35
36
37
38          Session session = null;
39          session = Session.getInstance(prop, new javax.mail.Authenticator() {
40              protected PasswordAuthentication getPasswordAuthentication() {
41                  return new PasswordAuthentication(GmailAccountEmail, GmailPassword);
42              }
43              });
44      try{
45       MimeMessage message;
46          message = new MimeMessage(session);
47
48       message.setFrom(new InternetAddress(GmailAccountEmail));
49
50       message.addRecipient(Message.RecipientType.TO,
51                            new InternetAddress(jTextFieldl.getText()));
52
53       message.setSubject("Your Borrowed Item is Due In TODAY");
```

Figure 1.22 - Private variables being accessed by encapsulation. Learned from (YouTube, 2014).

Since I want my email account to not be seen by anyone, I declare the strings involved as private. While the variables are private, it can only be accessed directly in the same class. I used encapsulation here to allow the method SendMail(), which is declared to be public to make it accessible by other classes if necessary. This allows access and retrieve the private variables that are normally inaccessible directly by other classes.

Simple Selection:

```
    private void SignupButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String emailValidate =
    "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
    + "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";
        if  (UserNameTextField.getText().equals(""))  {
            JOptionPane.showMessageDialog(null, "Failed to create new user.");}
        else if (PasswordTextField.getText().equals("")) {
                JOptionPane.showMessageDialog(null, "Failed to create new user.");
            }
        else if (EmailTextField.getText().matches(emailValidate)) {
            try {
            String sqlInsert = "INSERT INTO Account(UserName, FullName, Password, Email)\nVALUES (?,?,?,?)";
            preparedStatement = connection.prepareStatement(sqlInsert);
            preparedStatement.setString(1, UserNameTextField.getText());
            preparedStatement.setString(2, FullNameTextField.getText());
            preparedStatement.setString(3, PasswordTextField.getText());
            preparedStatement.setString(4, EmailTextField.getText());
            preparedStatement.execute();
            JOptionPane.showMessageDialog(null, "New Account Created Successfully");

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Failed to create new user.");    }


    }
        else{
            JOptionPane.showMessageDialog(null, "Failed to create new user.");
    } }
```

Figure 1.23 - Data validation in the signup process.

The method of selection is used here to validate the data entered when creating a new user. It compares the data entered in the UserNameTextField and PasswordTextField by the user to (“ ”), which is an empty field. It first validates the username, and the password, and then to the email. The string emailValidate represents the ordinary structure of an email, and if it matches the structure (on line 180), the inputted data will be inserted into the database, creating a new account for the user. If not then a popup will appear, signifying an error.

Other examples:

```
38      public void BorrowTable () {
39          try {
40              if (BorrowListTable.getRowCount() > 0) {
41                  String SqlBorrowList = "SELECT Borrow.FullName AS \"Full Name\", Borrow.Email AS \"Email\", "
42                          + "Equipment.EquipmentName AS \"Equipment Name\", Borrow.DateBorrow AS \"Borrow Date\"\n, Borrow.TimeReturned AS \"Return At\"" +
43                                  "FROM Borrow \n" +
44                                  "INNER JOIN Equipment ON Borrow.EquipmentID = Equipment.EquipmentID";
45                  preparedStatement = connection.prepareStatement(SqlBorrowList);
46                  resultSet = preparedStatement.executeQuery();
47                  BorrowListTable.setModel(DbUtils.resultSetToTableModel(resultSet));
48              }
49              else {
50                  //BorrowListTable.set
51              }
52
53          } catch (Exception e) {
54              JOptionPane.showMessageDialog(null, e);
55          }
56      }
```

Figure 1.24 - If statement to test if there are any rows in the table BorrowListTable, if there is, the data from the database will be selected and converted to TableModel.

```
229     private void LoginActionPerformed(java.awt.event.ActionEvent evt) {
230         String sql = "SELECT * FROM Account Where UserName=? And Password=?";
231         try {
232             preparedStatement = connection.prepareStatement(sql);
233             preparedStatement.setString(1, UserNameTextField.getText());
234             preparedStatement.setString(2, PasswordField.getText());
235             resultSet = preparedStatement.executeQuery();
236             if (resultSet.next()) {
237                 resultSet.close();
238                 preparedStatement.close();
239                 setVisible(false);
240                 DashBoard dashBoard = new DashBoard();
241                 dashBoard.setVisible(true);
242
243             }
244             else {
245                 JOptionPane.showMessageDialog(null, "Incorrect Username or Password");
246             }
247
249         } catch (Exception e) {
250             JOptionPane.showMessageDialog(null, e);
251         } finally {
252             try {
253                 resultSet.close();
254             preparedStatement.close();
255             }
256             catch (Exception e) {
257             }
258         }
```

Figure 1.25 - If statement to validate the login accounts, if it matches the data from a row in a table, the data will be selected and the current window will close while the dashboard window will appear.

```
private void SearchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String sqlSearchStudentEmailBorrowEquipment = "SELECT * FROM Borrow \n" +
                                                  "  INNER JOIN Equipment ON Borrow.EquipmentID = Equipment.EquipmentID\n" +
                                                  "  WHERE Borrow.Email = ? ORDER BY Borrow.DateBorrow DESC";
    try {
        preparedStatement = connection.prepareStatement(sqlSearchStudentEmailBorrowEquipment);
        preparedStatement.setString(1, EmailTextField.getText());
        resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            EquipmentIDTextField.setText(resultSet.getString("EquipmentID"));
            FullNameTextField.setText(resultSet.getString("FullName"));
            ClassTextField.setText(resultSet.getString("Class"));
            EquipmentCodeTextField.setText(resultSet.getString("EquipmentCode"));
            EquipmentNameTextField.setText(resultSet.getString("EquipmentName"));
            ReturnDateComboBox.setText(resultSet.getString("ReturnDate"));
            BorrowDateTextField.setText(resultSet.getString("DateBorrow").substring(0, 10));
            TimeReturnTextField.setText(resultSet.getString("TimeReturned"));
            resultSet.close();
            preparedStatement.close();
        }
        else {
            JOptionPane.showMessageDialog(null, "Student with this email: " +  EmailTextField.getText() + " have not borrowed any equipment");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e);
```

Figure 1.26 - If statement to get data from the entered email, if it matches the email in the table, then the listed fields above will be displayed with results from the remaining columns associated with that email.

Array List (ADT)

```
private void BorrowEquipementCountButtonActionPerformed(java.awt.event.ActionEvent evt) {
    ArrayList<Object> list = new ArrayList<>();
        try {
        if (BorrowListTable.getRowCount() > 0) {
            String SqlBorrowList = "SELECT Borrow.FullName AS \"Full Name\", Borrow.Email AS \"Email\", Equipment.EquipmentName AS \"Equipment Name\", "
                    + "Equipment.Quantity, Borrow.DateBorrow AS \"Borrow Date\"\n" +
                                "FROM Borrow \n" +
                                "INNER JOIN Equipment ON Borrow.EquipmentID = Equipment.EquipmentID ORDER BY [Borrow Date] DESC";
            preparedStatement = connection.prepareStatement(SqlBorrowList);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                list.add(resultSet.getString("Quantity"));
            }
            int sum = 0;
            for (Object var : list) {
                sum += Integer.parseInt(var.toString());
            }
            JOptionPane.showMessageDialog(rootPane, "Sum of all borrowed equipment: " + sum, "Doing Calculations...", 1);
        }
        else {
            JOptionPane.showMessageDialog(rootPane, "There are no items borrowed", "It's getting lonely in here...", 1);
        }

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

Figure 1.27 - Process of counting total items borrowed.

At first, I thought about using an array. Although later I used an array list instead because it is

resizable, which can be utilized here to store the number of items borrowed as there might be a lot of students borrowing a lot of equipment and the list can go on forever until the pieces of equipment are returned. Thus making an array list more suitable for the job as unlike array list, an array has a fixed size. On line 231 an array list is created to store objects.

Loops

First, the if statement will investigate if there is any row in BorrowListTable. If not, a message will popup saying there is none. If yes, the while loop used inside will iterate through the object resultSet, it keeps going through the data collected from the database, specifically strings from column "Quantity" in resultSet and adds them to the array list until there is nothing left to go through. Next, a variable sum is declared as an integer. Then the for loop will be used to go through each object of the array list, and for each of the object, it is analyzed and converted from a string to an integer and then adds up every time the loop iterate. The total sum at the end will be equal to the addition of all of the converted integers.

EventQueue

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

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

Figure 1.28 - The method run() is being enqueued to the last of the queue containing GUI processes.

EDT (Event Dispatching Thread) is a thread that stores Swing processes. EventQueue.invokeLater enqueue a new event that goes to the end of the list of Swing processes, the event is then processed only after all the other GUI processes have run, just like a queue process of First In First Out. In this case, after all the GUI events are processed, the dashboard is displayed.

Inheritance and GUI:

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

Since this product is for the art department, the visual aspects of it are very essential in the production of the program. That's why I decided to have all the classes in my program to inherit all the public fields and methods of the class JFrame.

```
        NewEquipmentLabel.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
        NewEquipmentLabel.setText("New Equipment");

        HistoryButton.setIcon(new javax.swing.ImageIcon(getClass().getResource("/images/history.png")));
        HistoryButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                HistoryButtonActionPerformed(evt);
            }
        });
```

Figure 1.29 - Structures and fonts of the NewEquipmentLabel and HistoryButton in class DashBoard. All images and icons used at (Iconarchive.com, n.d.). GUI Interface provided by Netbeans IDE and learned from (YouTube, 2013).

I have managed to change the font of the NewEquipmentLabel, along with its colour, setting up a text for it to appear, and also set an icon for HistoryButton so it would enhance more on the user experience.

```
public Borrow() {
    super("Borrow Equipment");
    initComponents();
```

Figure 1.30 - The title of the window are also customized to enhance the visual presentation of the program.

```
private javax.swing.JButton BackButton;
private javax.swing.JButton BorrowButton;
private datechooser.beans.DateChooserCombo BorrowDateCombobox;
private javax.swing.JLabel BorrowDateLabel;
private javax.swing.JLabel ClassLabel;
private javax.swing.JTextField ClassTextField;
private javax.swing.JLabel ClassroomLabel;
private javax.swing.JTextField ClassroomTextField;
private javax.swing.JLabel EmailLabel;
private javax.swing.JTextField EmailTextField;
private javax.swing.JLabel EquipmentCodeLabel;
private javax.swing.JTextField EquipmentCodeTextField;
private javax.swing.JLabel EquipmentDescriptionLabel;
private javax.swing.JTextField EquipmentDescriptionTextField;
private javax.swing.JTextField EquipmentIDTextField;
private javax.swing.JLabel EquipmentNameLabel;
private javax.swing.JTextField EquipmentNameTextField;
private javax.swing.JPanel EquipmentPanel;
private javax.swing.JLabel FullNameLabel;
private javax.swing.JTextField FullNameTextField;
private datechooser.beans.DateChooserCombo ReturnDateCombobox;
private javax.swing.JLabel ReturnDateLabel;
private javax.swing.JButton SearchEquipmentButton;
private javax.swing.JTextField StudentIDTextField;
private javax.swing.JPanel StudentInfoPanel;
private javax.swing.JLabel SupplierLabel;
private javax.swing.JTextField SupplierTextField;
private javax.swing.JComboBox TimeReturnedCombobox;
private javax.swing.JLabel TimeReturnedLabel;
```

Figure 1.31 - All GUI variables declared for class Borrow.

```
private javax.swing.JButton BorrowEquipmentButton;
private javax.swing.JLabel BorrowEquipmentLabel;
private javax.swing.JMenu DashboardMenu;
private javax.swing.JMenuBar DashboardMenuBar;
private javax.swing.JPanel DashboardPanel;
private javax.swing.JButton EquipmentListButton;
private javax.swing.JLabel EquipmentListLabel;
private javax.swing.JMenuItem ExitMenuItem;
private javax.swing.JButton HistoryButton;
private javax.swing.JLabel HistoryLabel;
private javax.swing.JLabel ImageTitle;
private javax.swing.JMenuItem LogoutMenuItem;
private javax.swing.JButton NewEquipmentButton;
private javax.swing.JLabel NewEquipmentLabel;
private javax.swing.JButton ReturnEquipmentButton;
private javax.swing.JLabel ReturnEquipmentLabel;
private javax.swing.JLabel TitleDashboard;
```

Figure 1.32 - All GUI variables declared for class DashBoard.

```
    private javax.swing.JButton EmailSendButton;
    private javax.swing.JLabel EmailTitleLabel;
    private javax.swing.JLabel StudentEmailLabel;
    private javax.swing.JTextField StudentEmailTextField;
```

Figure 1.33 - All GUI variables declared for class Email.

```
    private javax.swing.JButton BackButton;
    private javax.swing.JPanel EquipmentListPanel;
    private javax.swing.JPanel EquipmentListPanel2;
    private javax.swing.JTable EquipmentListTable;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane2;
```

Figure 1.34 - All GUI variables declared for class EquipmentList.

```
    private javax.swing.JButton AlertStudentButton;
    private javax.swing.JButton Back;
    private javax.swing.JButton BorrowEquipementCountButton;
    private javax.swing.JPanel BorrowListPanel;
    private javax.swing.JTable BorrowListTable;
    private javax.swing.JPanel ReturnListPanel;
    private javax.swing.JTable ReturnListTable;
    private javax.swing.JScrollPane jScrollPanel;
    private javax.swing.JScrollPane jScrollPane2;
```

Figure 1.35 - All GUI variables declared for class History.

```
    private javax.swing.JButton LoginButton;
    private javax.swing.JPanel LoginPanel;
    private javax.swing.JPasswordField PasswordField;
    private javax.swing.JLabel PasswordLabel;
    private javax.swing.JButton SignupButton;
    private javax.swing.JPanel SpacePanel;
    private java.awt.Label SubtitleLabel;
    private javax.swing.JLabel TitleImageLabel;
    private java.awt.Label TitleLabel;
    private javax.swing.JTextField UserNameTextField;
    private javax.swing.JLabel UsernameLabel;
    private javax.swing.JPanel jPanel2;
    private java.awt.Label label1;
    private java.awt.Label label3;
```

Figure 1.36 - All GUI variables declared for class Login.

```
    private javax.swing.JLabel ClassroomLabel;
    private javax.swing.JTextField ClassroomTextField;
    private javax.swing.JTextField EquipDescriptionTextField1;
    private javax.swing.JTextField EquipDescriptionTextField3;
    private javax.swing.JTextField EquipDescriptionTextField4;
    private javax.swing.JTextField EquipDescriptionTextField5;
    private javax.swing.JTextField EquipNameTextField;
    private javax.swing.JLabel EquipmentCodeLabel;
    private javax.swing.JTextField EquipmentCodeTextField;
    private javax.swing.JLabel EquipmentDescriptionLabel;
    private javax.swing.JTextField EquipmentDescriptionTextField;
    private javax.swing.JLabel EquipmentNameLabel;
    private javax.swing.JPanel NewEquipmentPanel;
    private javax.swing.JLabel QuantityLabel;
    private javax.swing.JTextField QuantityTextField;
    private javax.swing.JLabel SupplierLabel;
    private javax.swing.JTextField SupplierTextField;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
```

Figure 1.37 - All GUI variables declared for class NewEquipment.

```
    private javax.swing.JButton BackButton;
    private javax.swing.JLabel BorrowedDateLabel;
    private javax.swing.JTextField BorrowedDateTextField;
    private javax.swing.JLabel ClassLabel;
    private javax.swing.JTextField ClassTextField;
    private javax.swing.JLabel EmailLabel;
    private javax.swing.JTextField EmailTextField;
    private javax.swing.JLabel EquipmentCodeLabel;
    private javax.swing.JTextField EquipmentCodeTextField;
    private javax.swing.JTextField EquipmentIDTextField;
    private javax.swing.JLabel EquipmentNameLabel;
    private javax.swing.JTextField EquipmentNameTextField;
    private javax.swing.JLabel FullNameLabel;
    private javax.swing.JTextField FullNameTextField;
    private javax.swing.JButton ReturnButton;
    private datechooser.beans.DateChooserCombo ReturnDateComboBox;
    private javax.swing.JLabel ReturnDateLabel;
    private javax.swing.JPanel ReturnEquipmentPanel;
    private javax.swing.JButton SearchButton;
    private javax.swing.JTextField StudentIDTextField;
    private javax.swing.JLabel TimeReturnedLabel;
    private javax.swing.JTextField TimeReturnedTextField;
    private javax.swing.JLabel jLabel4;
```

Figure 1.38 - All GUI variables declared for class ReturnEquipment.

```
    private javax.swing.JButton BackButton;
    private javax.swing.JLabel EmailLabel;
    private javax.swing.JTextField EmailTextField;
    private javax.swing.JLabel FullNameLabel;
    private javax.swing.JTextField FullNameTextField;
    private javax.swing.JLabel PasswordLabel;
    private javax.swing.JPasswordField PasswordTextField;
    private javax.swing.JButton SignupButton;
    private javax.swing.JPanel SignupLabel;
    private javax.swing.JTextField UserNameTextField;
    private javax.swing.JLabel UsernameLabel;
```

Figure 1.39 - All GUI variables declared for class Signup

Additional Libraries

All libraries used:

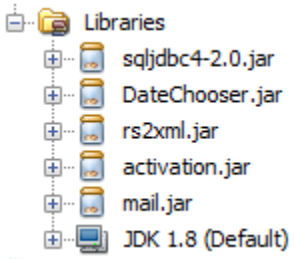

Figure - All libraries presented.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
```

Figure 1.40 - All libraries imported in class Borrow.

```
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
```

Figure 1.41 - All libraries imported in class Email.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import net.proteanit.sql.DbUtils;
```

Figure 1.42 - All libraries imported in class EquipmentList.

```
import javax.swing.JOptionPane;
import javax.swing.UIManager;
```

Figure 1.43- All libraries imported in class EquipmentManagementSystem.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import net.proteanit.sql.DbUtils;
```

Figure 1.44 - All libraries imported in class History.

```
import java.sql.*;
import javax.swing.JOptionPane;
```

Figure 1.45 - All libraries imported in class JavaConnectionDatabase.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
```

Figure 1.46 - All libraries imported in class Login.

```
import com.sun.java.swing.plaf.windows.WindowsBorders;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
```

Figure 1.47 - All libraries imported in class NewEquipment.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
```

Figure 1.48 - All libraries imported in class ReturnEquipment.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JDesktopPane;
import javax.swing.JOptionPane;
```

Figure 1.49 - All libraries imported in class Signup.

Word count: 1141

Bibliography:

All knowledge related to sending email, sources of code of ports and email server:

- Mkyong.com. (2010). JavaMail API – Sending email via Gmail SMTP example – Mkyong.com. [online] Available at: https://www.mkyong.com/java/javamail-api-sending-email-via-gmail-smtp-example/ [Accessed 8 Jul. 2019].
- Tutorialspoint.com. (n.d.). JavaMail API - Quick Guide - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/javamail_api/javamail_api_quick_guide.htm# [Accessed 8 Jun. 2019].

SQL-related knowledge:

- Docs.oracle.com. (n.d.). Processing SQL Statements with JDBC (The Java™ Tutorials > JDBC(TM) Database Access > JDBC Basics). [online] Available at: https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#establishing_connections [Accessed 8 Sep. 2019].
- Tutorialspoint.com. (n.d.). SQL Tutorial - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/sql/index.htm [Accessed 3 Jul. 2019].
- o7planning.org. (n.d.). Hướng dẫn sử dụng Java JDBC kết nối cơ sở dữ liệu. [online] Available at: https://o7planning.org/vi/10167/huong-dan-su-dung-java-jdbc [Accessed 8 Aug. 2019].
- query?, W. and Parker, J. (2011). What is parameterized query?. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/4712037/what-is-parameterized-query [Accessed 8 Oct. 2019].

Java-related knowledge:

- Netbeans Java Swing Tutorial:

    - YouTube. (2013). JAVA Creating a SWING application with NetBeans IDE 7 4. [online] Available at: https://www.youtube.com/watch?v=6IJr5JIY9Yo [Accessed 8 May 2019].
- Encapsulation Java Lesson:
    - YouTube. (2014). 8.1 What is Encapsulation in Java Tutorial. [online] Available at: https://www.youtube.com/watch?v=tt_astMjep0 [Accessed 8 May 2019].

Libraries:
- Sqljdbc4-2.0.jar:
    - Java2s.com. (n.d.). Download sqljdbc4-2.0.jar : sqljdbc4 « s « Jar File Download. [online] Available at: http://www.java2s.com/Code/Jar/s/Downloadsqljdbc420jar.htm [Accessed 8 Oct. 2019].
    - Docs.microsoft.com. (2019). Using the JDBC driver - SQL Server. [online] Available at: https://docs.microsoft.com/en-us/sql/connect/jdbc/using-the-jdbc-driver?view=sql-server-2017 [Accessed 8 Aug. 2019].
- DateChoose.jar:
    - Plugins.netbeans.org. (n.d.). JDateChooser 1.2 - NetBeans Plugin detail. [online] Available at: http://plugins.netbeans.org/plugin/658/jdatechooser-1-2 [Accessed 8 Jul. 2019].
- rs2xml.jar for the use of tables:
    - rs2xml, j., Maya, V. and Taleb, M. (2015). jtable how to use rs2xml. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/27679867/jtable-how-to-use-rs2xml [Accessed 2 Jun. 2019].
    - Karue, B. (n.d.). rs2xml.jar Free Download - Hack Smile. [online] Hack Smile. Available at: https://hacksmile.com/rs2xml-jar-free-download/ [Accessed 15 Jul. 2019].
- activation.jar and mail.jar for sending email:
    - Java2s.com. (n.d.). Programming Tutorials and Source Code Examples. [online] Available at: http://www.java2s.com/ [Accessed 8 Oct. 2019].
- Java SE Development Kit:
    - Oracle.com. (n.d.). Java SE Development Kit 8 - Downloads. [online] Available at: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html [Accessed 8 Oct. 2019].

Icons:
    - Iconarchive.com. (n.d.). Icon Archive - Search 735,802 free icons, desktop icons, download icons, social icons, xp icons, vista icons. [online] Available at: http://www.iconarchive.com/ [Accessed 8 May 2019].