

## Criterion C: Development

### TABLE OF CONTENTS

<b><u>Concept</u></b>	<b><u>PAGE</u></b>
GUI Elements.....	2
Object-Oriented Programming	
Database Creation.....	7
Database Management.....	12
Calculations.....	16
Sources.....	17

## Graphical User Interface

I decided to use GUI as it would provide {Client} with the necessary visual aid and feedback as she went through data input and calculations. In the GUI I decided to use **Swing** and **AWT** instead of JavaFX as a GUI library. Swing provides both additional components and added functionality with AWT-replacement components. Swing components can provide a flexible UI. Swing also provides “extras” for its components, such as icons and tooltips and is the preferred type of UI for {Client} and is thus used within this application.

### Imports

```
package hussainiaproject;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
```

```
package hussainiaproject;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.SwingConstants;
```

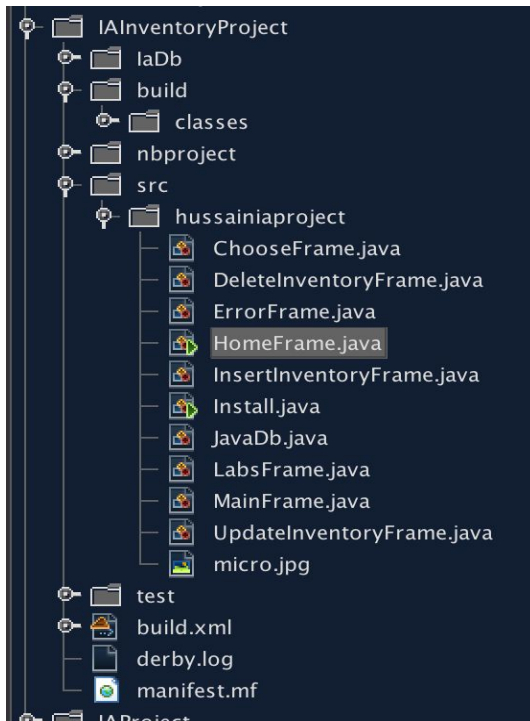
```
package hussainiaproject;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import static javax.swing.JFrame.EXIT_ON_CLOSE;
```

```
package hussainiaproject;
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableColumn;
```

Images above display the relevant imports that were necessary to create and implement the GUI Components and Frames shown below.

## Classes



Classes in this program follow **Java** and are concurrent, object-oriented, and specifically designed to have as few implementation dependencies as possible.

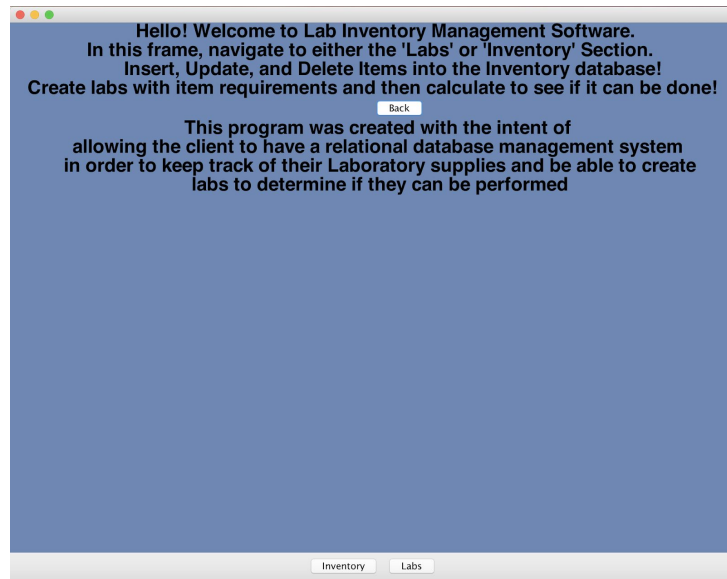
## GUI Implementation/System Flow

The following images show the supposed visual flow/process taken by {Client} whence using the application.

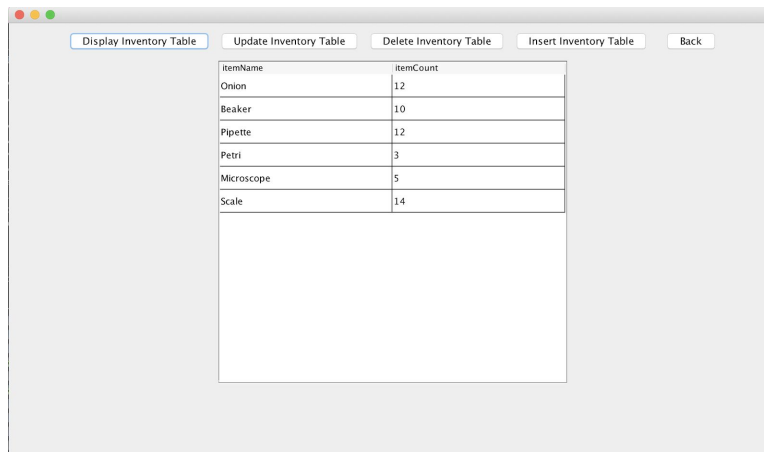
Beginning at the first window, {Client} can start the program as prompted by the JButton and JLabel.



HomeFrame.java

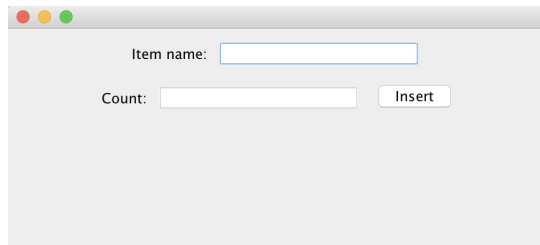


ChooseFrame.java - Giving {Client} insight into program, presenting options for Inventory or Labs.



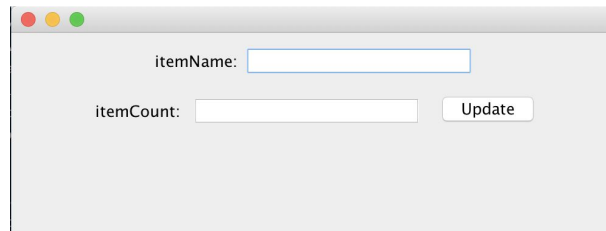
MainFrame.java - Client can insert, update, delete and display values in database into JTables.

## GUI of Functions within **MainFrame.java**



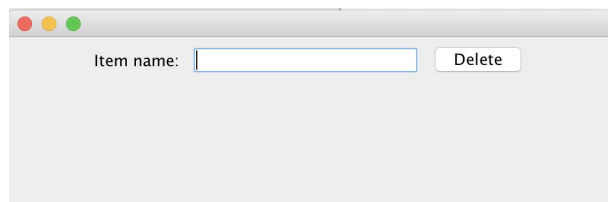
A Java Swing window titled "MainFrame.java" with a light gray background. It contains two text input fields: "Item name:" and "Count:". To the right of the "Count:" field is a button labeled "Insert".

*Insert* function



A Java Swing window titled "MainFrame.java" with a light gray background. It contains two text input fields: "itemName:" and "itemCount:". To the right of the "itemCount:" field is a button labeled "Update".

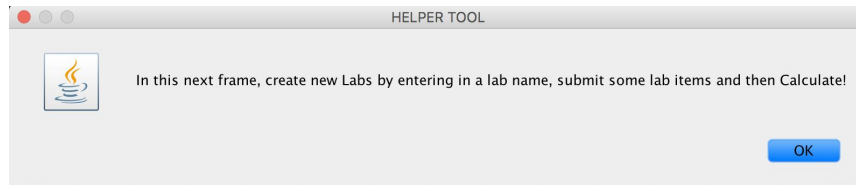
*Update* function



A Java Swing window titled "MainFrame.java" with a light gray background. It contains one text input field labeled "Item name:". To the right of the input field is a button labeled "Delete".

*Delete* function

## Functions within LabsFrame.java



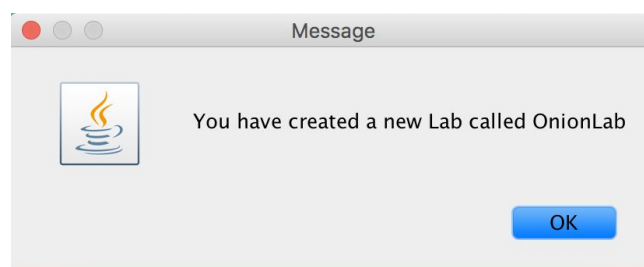
*Helper function* for {Client} for guidance *before* LabsFrame.java is displayed



Labs Frame - evident use of JTextField, JButtons, JLabels



*Bottom of LabsFrame.java* ... shows the area to create a new Lab



*Prompt* after new lab created

## Object-Oriented Programming

One of the key aspects of this program was the usage of *Object Oriented Programming*. OOP was crucial to this program's success and extensibility in terms of methods and accessing information. Through OOP use, I found it useful that I could deal with multiple classes with methods, making my program modular and easier to manage. This also meant that extensibility and maintenance of my program were easier and allowed for use of complex methods for data setting and retrieval in different situations.

## Database Creation

### Install.java

```
package hussainiaproject;

/**
 *
 * @author sameerhussain
 */
public class Install
{
    public static void main(String [] args)
    {
        //Inventory Table
        String inventoryTable;
        JavaDb objDb = new JavaDb();
        objDb.createDb("IaDb");
        inventoryTable = "CREATE TABLE Inventory ( " +
            "itemName varchar(20), " +
            "itemCount int " +
            ")";

        System.out.println(inventoryTable);
        objDb.createTable(inventoryTable, "IaDb");

        //Lab Table
        String labList;
        labList = "CREATE TABLE LabList ( " +
            "labName varchar(20) " +
            ")";
        System.out.println(labList);
        objDb.createTable(labList, "IaDb");

        String labNeeds;
        labNeeds = "CREATE TABLE LabNeeds ( " +
            "name varchar(20), " +
            "count int " +
            ")";
        System.out.println(labNeeds);
        objDb.createTable(labNeeds, "IaDb");
    }
}
```

### Use of Oracle's JavaDB (Apache Derby)

JavaDB is Oracle's supported distribution of the *Apache Derby* open source database. It supports standard *ANSI/ISO SQL* through the *JDBC and Java EE APIs*. Java DB is included in the JDK and Apache Derby is a relational database management system that can be embedded in Java programs (and is implemented into this one).

**Install.java** serves as a class that pre-creates SQL Tables into the Database used in this IA which is named "IaDb".

### Database Tables:

The tables: **Inventory**, **LabList**, and **LabNeeds** are ones that hold values for the lab stock, list of labs, and each lab's requirements, respectively.

These tables are *normalized* to the third normal form (*3NF*) to reduce the duplication of data and ensure referential integrity by having 2NF and all

the attributes in a table are determined only by the primary keys of that relationship and not by any non-prime structures.

**SQL** (Structured Query language) is used as a medium for the program to **communicate with the Java Derby Database used to hold the data.**

I used queries to allow for specific inputs to be provided to the program which allows {Client} to modify the contents of the database when queries are used.

## Establishing Connections

### JavaDb.java

```
/* Sameer Hussain - IB Computer Science 1 - Match 13, 2018 - This program has the database methods */

package hussainiaproject;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
```

```
public class JavaDb {

    private String dbName;
    private Object[][] data;
    private Connection dbConn;

    public JavaDb(String dbName)
    {
        this.dbName = dbName;
        this.data = null;
        setDbConn();
    }

    public JavaDb()
    {
        this.dbName="";
        this.data = null;
        this.dbConn=null;
    }
}
```

**Constructors** for class “JavaDb” that set up data structures of the database name (String), object for data(Two-dimensional), and connection is declared.

#### Encapsulation:

In most of these cases, I utilized encapsulation that ensured that variables couldn’t be accessed directly from another class (maintaining the conformity of user data).

Variables in JavaDb.java have **accessors** and **mutators** which helps with usability and helps



with error prevention and maintenance. Expansion of {Client}'s application or for updates by are easier as variables are encapsulated.

### Inheritance:

In my GUI frames, inheritance of JFrame through *extends* is used as all of GUI runs through a JFrame. Additionally, *super()*; is used for inheriting the contents of the parent class (to create new classes that are built upon existing classes).

```
public class HomeFrame extends JFrame implements ActionListener
{
    private final Color FRAME_COLOR = new Color(100,100,100);
    private final Color TEXT_COLOR = new Color(11,11,11);
    private JLabel openingLabel;
    private final Font TEXT_FONT = new Font("Arial", Font.ITALIC|Font.BOLD,30);
    private final java.net.URL MY_IMAGE = getClass().getResource("micro.jpg");
    private final ImageIcon THE_PICTURE = new ImageIcon(MY_IMAGE);
    private JLabel imageLabel;
    private JPanel homePanel;
    private JButton startButton;
    private JPanel welcomePanel;
    private JLabel descriptionLabel;
    private JButton exitButton;

    public HomeFrame()
    {
        //Setting up frame
        super();
        this.setBounds(200,200,800,600);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.getContentPane().setBackground(FRAME_COLOR);
        this.setLayout(new BorderLayout());
    }
}
```

### **Data structures:**

**ArrayLists** - Regular Arrays are fixed length in data structure. But, ArrayList is variable length, which means ArrayList can grow or shrink its size dynamically, which is necessary as {Client} is constantly updating her Inventory stock.

```

public Object[][] getData(String tableName, String[] tableHeaders)
{
    int columnCount = tableHeaders.length;
    ResultSet rs = null;
    Statement s = null;
    String dbQuery = "SELECT * FROM " + tableName;
    ArrayList<ArrayList> dataList = new ArrayList<>();

    try
    {
        s = this.dbConn.createStatement();
        rs = s.executeQuery(dbQuery);
        while(rs.next())
        {
            ArrayList<String> row = new ArrayList<String>();
            for(int i=0; i<columnCount; i++)
            {
                row.add(rs.getString(tableHeaders[i]));
            }
            dataList.add(row);
        }
        this.data = new Object[dataList.size()][columnCount];
        for (int i=0; i<dataList.size(); i++)
        {
            ArrayList<String> row = new ArrayList<String>();
            row = dataList.get(i);
            for (int j=0; j<columnCount; j++)
            {
                this.data[i][j] = row.get(j);
            }
        }
    }
    catch(Exception e)
    {
        System.exit(0);
    }
    return data;
}

```

Methods in JavaDb.java such as `getData` used *SQL queries* like to display the contents of a database into a `JTable`.

*ArrayLists* and *for loops* are used to ensure all data are processed and within tables by taking list's size and continually adding data until there isn't *.next()* to get.

```

public void setData(Object[][] data) {
    this.data = data;
}

public String getDbName() {
    return dbName;
}

public void setDbName(String dbName) {
    this.dbName = dbName;
}

public Connection getDbConn() {
    return dbConn;
}

//Creating connection

public void setDbConn() {
    String connectionURL = "jdbc:derby:" + this.dbName;
    this.dbConn = null;
    try
    {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        this.dbConn = DriverManager.getConnection(connectionURL);
    }
    catch (SQLException err)
    {
        ErrorFrame objEE = new ErrorFrame();
    }
    catch(ClassNotFoundException ex)
    {
        ErrorFrame objEE = new ErrorFrame();
    }
}

```

Evidence to the left shows techniques of *encapsulating* the data provided by the user as well as the name of the database and connection to the database.

← Embedded driver API for Oracle's Apache Derby

```

if (command.equals("Display Inventory Table"))
{
    inventoryData = objDb.getData(tableName1, columnTitles1);

    invTable = new JTable(inventoryData, columnTitles1);
    invHeader = new JTableHeader();
    invHeader = invTable.getTableHeader();

    column1 = invTable.getColumnModel().getColumn(0);
    column1.setPreferredWidth(50);
    column1 = invTable.getColumnModel().getColumn(1);
    column1.setPreferredWidth(50);
    /*column1 = timeInTable.getColumnModel().getColumn(2);
    column1.setPreferredWidth(50);
    column1 = timeInTable.getColumnModel().getColumn(3);
    column1.setPreferredWidth(50);
    column1 = timeInTable.getColumnModel().getColumn(4);
    column1.setPreferredWidth(50);

    invTable.setRowHeight(30);
    invTable.setGridColor(Color.BLACK);

    this.remove(scrollPanel1);

    scrollPanel1 = new JScrollPane();

    scrollPanel1.getViewport().add(invTable);

    invTable.setFillViewportHeight(true);

    this.add(scrollPanel1, BorderLayout.EAST);

    this.validate();
    this.repaint();
}
else if (command.equals("Update Inventory Table"))
{
    UpdateInventoryFrame objUArt = new UpdateInventoryFrame();
}
else if (command.equals("Delete Inventory Table"))
{
    DeleteInventoryFrame objDArt = new DeleteInventoryFrame();
}
else if (command.equals("Insert Inventory Table"))
{
    InsertInventoryFrame objIArt = new InsertInventoryFrame();
}

this.validate();
this.repaint();
}

```

This coincides with the display of the data into the JTable as shown below (again, with use of SQL Queries and Prepared Statements to execute the changes from Java to SQL).

## *Insert, Update, and Deletion* of data from Java through SQL in Database

```
@Override
public void actionPerformed(ActionEvent e)
{
    String command = e.getActionCommand();
    String fName;
    String cnt;
    int cntInt;
    String query;
    JavaDb dbobj = new JavaDb("IaDb");
    Connection conn;
    int status;
    conn = dbobj.getDbConn();
    if (command.equals("Update"))
    {
        fName = itemNameField.getText();
        cnt = countField.getText();
        cntInt = Integer.parseInt(cnt);
        query = "UPDATE Inventory " +
            "SET itemCount=? " +
            "WHERE itemName=?";
        try{
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setInt(1, cntInt);
            statement.setString(2, fName);
            status = statement.executeUpdate();
        }
        catch(Exception error){
            ErrorFrame objEF = new ErrorFrame();
            error.printStackTrace();
        }
    }
}
```

```
@Override
public void actionPerformed(ActionEvent e)
{
    String command = e.getActionCommand();
    String itemName;
    String lName;
    String count;
    int countInt;
    String gender;
    String job;
    String query;
    JavaDb dbobj = new JavaDb("IaDb");
    Connection conn;
    int status;
    conn = dbobj.getDbConn();
    if (command.equals("Insert"))
    {
        itemName = itemNameField.getText();
        count = countField.getText();
        countInt = Integer.parseInt(count);
        query = "INSERT INTO Inventory (itemName, itemCount) " +
            "VALUES (?,?)";
        try{
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setString(1, itemName);
            statement.setInt(2, countInt);
            status = statement.executeUpdate();
            System.out.println(worked);
        }
        catch(Exception error){
            ErrorFrame objEF = new ErrorFrame();
            error.printStackTrace();
        }
    }
}
```

```
@Override
public void actionPerformed(ActionEvent e)
{
    String command = e.getActionCommand();
    String itemName;
    String query;
    JavaDb dbobj = new JavaDb("IaDb");
    Connection conn;
    int status;
    conn = dbobj.getDbConn();
    if (command.equals("Delete"))
    {
        itemName = itemNameField.getText();
        query = "DELETE FROM Inventory WHERE itemName=?";
        try{
            PreparedStatement statement = conn.prepareStatement(query);
            statement.setString(1, itemName);
            status = statement.executeUpdate();
        }
        catch(Exception error){
            ErrorFrame objEF = new ErrorFrame();
            error.printStackTrace();
        }
    }
}
```

These executions are within the `ActionPerformed` of the Class `MainFrame.java` and correspond to their implementations in aforementioned screenshots with the same name.

**Error Frame** - A technique for robustness in the program - instead of crashing the program errors detected through *catch statements* revert to an Error Frame that notifies the issue. Errors like **SQLException** presented above or a multitude of other errors present problems within the runtime.

```

for(int i=0; i<Names.size(); i++)
{
    out[i][0] = Names.get(i);
    out[i][1] = Counts.get(i);
}
return out;
}
catch(SQLException e)
{
    return null;
}
}

```

```

package hussainiaproject;

import java.awt.FlowLayout;
import javax.swing.JFrame;
import static javax.swing.JFrame.EXIT_ON_CLOSE;
import javax.swing.JLabel;

//Frame for errors

public class ErrorFrame extends JFrame
{
    final JLabel errorLabel;
    public ErrorFrame()
    {
        super();
        this.setBounds(100,200,500,200);
        this.setLayout(new FlowLayout());

        errorLabel = new JLabel("There's been an error in your action "
            + "with this database");
        this.add(errorLabel);
        this.setVisible(true);
    }
}

```

**Navigation through Frames** - a simple technique of setting visible to different booleans based on action required and the calling of the subsequent class. (Example: **HomeFrame.java**)

```

public void actionPerformed(ActionEvent e)
{
    String command = e.getActionCommand();

    //Start button pressed
    if(command.equals("Start"))
    {
        ChooseFrame openFrame = new ChooseFrame(this);
        openFrame.setVisible(true);
        this.setVisible(false);
    }

    if(command.equals("Exit"))
    {
        System.exit(0);
    }
}

public static void main(String[] args) {
    HomeFrame objHomeFrame = new HomeFrame();
}
}

```

## *Techniques of Database Management:*

```
public void createDb(String newDbName)
{
    this.dbName = newDbName;
    String connectionURL = "jdbc:derby:" + this.dbName + ";create=true";
    try
    {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        this.dbConn = DriverManager.getConnection(connectionURL);
        System.out.println("new database created");
        this.dbConn.close();
    }
    catch (Exception err)
    {
        ErrorFrame objEF = new ErrorFrame();
        err.printStackTrace();
    }
}

public void createTable(String newTable, String dbName)
{
    Statement s;
    setDbName(dbName);
    setDbConn();
    try
    {
        s = this.dbConn.createStatement();
        s.execute(newTable);
        System.out.println("New table created");
        this.dbConn.close();
    }
    catch (SQLException err)
    {
        ErrorFrame objEF = new ErrorFrame();
        err.printStackTrace();
    }
}
```

Creation of Db and tables for the Database (used through the Install.java class)

Uses methods setDbconn (accessors) and “Statement” (SQL class) to execute SQL actions.

Application provides methods to query and update (s.execute) data in a database.

JDBC allows multiple implementations to exist and be used.

I use JDBC in my program to support creating and executing statements with SQL’s CREATE, INSERT, UPDATE, and DELETE, or query statements like SELECT.

Stored procedures (**shown above**) are invoked through the JDBC connection in JavaDb.java.

**Statement** – the statement is sent to the database server each and every time.

**PreparedStatement** – the statement is cached and execution path is pre-set on DB server allowing it to be executed.



### *Manipulating data (Labs to Inventory Table):*

```
public int returnCount(String itemName, String dbName)
{
    //statement = smth like "SELECT count FROM Inventory WHERE name='beaker';
    Statement s;
    ResultSet rs;
    String statement = "SELECT itemCount FROM Inventory WHERE itemName='" + itemName + "'";
    setDbName(dbName);
    setDbConn();
    int out = -1;
    try
    {
        s = this.dbConn.createStatement();
        rs = s.executeQuery(statement);
        System.out.println("Who cares");
        if(rs.next())
        {
            out = rs.getInt("itemCount");
        }
        this.dbConn.close();
    }
    catch(SQLException e)
    {
        ErrorFrame objEF = new ErrorFrame();
        e.printStackTrace();
    }
    return out;
}
```

Returns the amount of items (count) for a particular *itemName* for LabsFrame.java and for Lab Tables in *LabsFrame.java* class as shown below:

```
if(command.equals("Enter"))
{
    LabTableLabel = jTextField1.getText().replace(" ", ""); //make it string!!!!
    String newTable = "CREATE TABLE "+LabTableLabel+" (" +
        "itemName varchar(20), " +
        "itemCount int " +
        ")";
    System.out.println(newTable);
    dbobj.createTable(newTable, "IaDb");
    JOptionPane.showMessageDialog(null, "You have created a new Lab called "+LabTableLabel);
    this.validate();
    this.repaint();
}
if(command.equals("Submit Items"))
{
    String itemName = jTextField2.getText();
    String itemCount = jTextField3.getText();
    int itemInt;
    itemInt = Integer.parseInt(itemCount);
    query = "INSERT INTO "+ LabTableLabel + " (itemName, itemCount) "
        + "VALUES (?,?)";
    try{
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setString(1, itemName);
        statement.setInt(2, itemInt);
        status = statement.executeUpdate();
        System.out.println("IT WORKED");
        JOptionPane.showMessageDialog(null, "You have added new item of "+itemName+" with count of "+itemCount);
    }
    catch(Exception error){
        ErrorFrame objEF = new ErrorFrame();
        error.printStackTrace();
    }
    this.validate();
    this.repaint();
}
```

Creation of  
*specific tables* for  
*specific labs* for  
user.

```

if(command.equals("Calculate"))
{
    String output;
    Object[][] supplies;
    String[] names;
    int [] counts;
    int [] finalCount;
    boolean enough = true;
    for(JRadioButton temp : buttonArray)
    {
        if(temp.isSelected()) {
            supplies = dbobj.suppliesNeeded(temp.getText());
            names = new String[supplies.length];
            counts = new int[supplies.length];
            finalCount = new int[supplies.length];
            for(int i=0; i<supplies.length; i++)
            {
                names[i] = (String)supplies[i][0];
                System.out.println(supplies[i][1]);
                counts[i] = Integer.parseInt((String) supplies[i][1]);
            }
            for(int i=0; i<supplies.length; i++)
            {
                finalCount[i] = dbobj.returnCount(names[i], "IaDb") - counts[i];
                if(finalCount[i] < 0)
                {
                    enough = false;
                }
            }
            if(enough)
            {
                output = "You have enough items to do this lab!";
                outputLabel.setText(" "+output);
                outputLabel.setFont(new Font("Helvetica", Font.ITALIC, 26));
                this.validate();
                this.repaint();
            }
            else
            {
                output = "You do not have enough items for this lab, please restock Inventory!";
                outputLabel.setText(" "+output);
                outputLabel.setFont(new Font("Helvetica", Font.ITALIC, 26));
                this.validate();
                this.repaint();
            }
        }
    }
    //JOptionPane.showMessageDialog(this, "Result",output, JOptionPane.PLAIN_MESSAGE);
    System.out.println(output);
}
}

```

Calculating the amount of items needed for each lab by taking the amount needed (*itemCount*) in the *specific Lab table* and subtracting from amount needed in *Inventory* table for specific *itemName*

This is done through *suppliesNeeded* method in JavaDb.java shown left and thus, envelops the entire program through its use in Calculations.

```

public Object[][] suppliesNeeded(String labName)
{
    try
    {
        Object [][] out;
        Statement stmt = dbConn.createStatement();
        ResultSet rs;
        ArrayList<String> Names = new ArrayList<>();
        ArrayList<String> Counts = new ArrayList<>();

        String sql = "SELECT * FROM "+labName;
        rs = stmt.executeQuery(sql);
        while(rs.next())
        {
            Names.add(rs.getString("itemName"));
            Counts.add(Integer.toString(rs.getInt("itemCount")));
        }
        out = new Object[Names.size()][2];

        for(int i=0; i<Names.size(); i++)
        {
            out[i][0] = Names.get(i);
            out[i][1] = Counts.get(i);
        }
        return out;
    }
    catch(SQLException e)
    {
        return null;
    }
}

```



## Bibliography

Group, Documentation. "Apache." *Security Tips - Apache HTTP Server Version 2.4*, <httpd.apache.org/>.

"Java DB." *What Is Big Data?* | Oracle, [www.oracle.com/technetwork/java/javadb/overview/index.html](http://www.oracle.com/technetwork/java/javadb/overview/index.html).