Criterion C: Development

Techniques used in developing the product:

Java Imports
SQL Commands
Use of Validation
Use of database to save/load persistent data
Use of jFreeChart
Use of jTable

Java Imports:

```
import java.awt.*;
import java.util.Random;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Connection;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.plot.CategoryPlot;
```

- "util.Random" is used to generate a random ID for the Sale report
- awt is used for graphical components regarding the chart
- Jfreechart is used to add a chart in the program using these imports:
  - org.jfree.data.category.DefaultCategoryDataset: It stores data in rows and columns, where rows represent categories and columns represent series, and provides methods to add, update, and retrieve data.
  - org.jfree.chart.ChartPanel: This is used to create a panel that allows the chart to appear on it
  - org.jfree.chart.plot.CategoryPlot: is used to render the visuals of the chart in the panel.
  - jfree.chart.ChartFactory: This is used to allow you to easily customize different types of charts (e.g., line charts, bar charts, pie charts) with default configurations.
  - org.jfree.chart.JFreeChart: This import brings in the "JFreeChart" class, which represents a chart object in JFreeChart.
  - org.jfree.chart.plot.PlotOrientation: Used to specify plot orientation, such as vertical or horizontal. Commonly used in creating bar charts or line charts to determine data orientation on the chart.
- sql.SQLException, sql.PreparedStatement, sql,RseultSet, sql.Connection is used to let the program interact with the MS Access database; this is how each import interacts:
  - java.sql.ResultSet: This is used to give the result that was executed and make it appear in the program.

- Java.sql.Connection: This is used to connect the program to the Database through a class that will be created to connect it.
- Java.sql.SQLException: This is used to handle any error occurring in any issue regarding the database.
- Java.sql.PreparedStatement: is used to execute the sql statements in the program

SQL Commands:

```
String SQL = "INSERT INTO ADDPRODUCT (GASTYPE, QUANTITY, PRICEPERKG, MOREINFORMATION) VALUES('" + gt + "','" + qq + "','" + pkg + "','" + info + "')";
```

Figure 1. INSERTING GAS

```
String sql = "SELECT * FROM ADDPRODUCT WHERE GASTYPE='" + selected + "'";
```

Figure 2. SELECTING GAS

```
String SQL = "delete FROM ADDPRODUCT WHERE GASTYPE='" + gt + "'";
```

Figure .3 DELETE GAS

```
String SQL = "UPDATE ADDPRODUCT SET QUANTITY='" + qu + "', PRICEPERCYLINDER='" + price + "', MOREINFORMATION='" + mi + "' WHERE GASTYPE='" + gt + "'";
```

Figure 4. UPDATING GAS

Use of Validation:

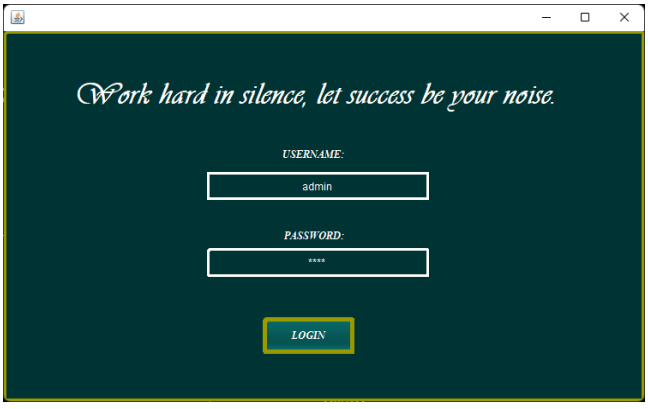

Figure 1. LOGIN PAGE

Figure 2. when data is wrong

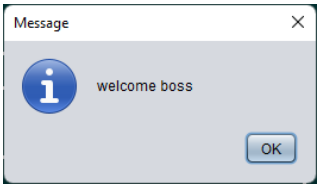

Figure 3. when data is correct

```
void login(){
    String username = jTextField1.getText();
    String password = String.valueOf(jPasswordField1.getPassword());

    if(username.equals("admin") && password.equals("admin")){
        JOptionPane.showMessageDialog(null, "welcome boss");
        Dashboard frame2 = new Dashboard();
        frame2.setVisible(true);
        this.dispose();
    }
    else {
        JOptionPane.showMessageDialog(null, "invalid username or password");

    }
}
```

The following Java code snippet is part of a login function. It starts by getting the username and password that the user entered respectively in the text field (`jTextField1` for username and `jPasswordField1` for password). The code performs a validation action upon the retrieval of the input by verifying if the entered username matches "admin" and the entered password also matches "admin". This check is done through the `if` statement.

If both the username and password are matched to the expected values ("admin" for both), the code produces a welcome message using `JOptionPane.showMessageDialog()` to show that the login is successful. Also, it starts a new dashboard window(`Dashboard`) where user can move forward in the program. The current popup window is closed by `this.dispose()`. If the entered username or password is not "admin", the code uses `JOptionPane.showMessageDialog()` to show an error message, indicating invalid username/password.

Use of database to save/load persistent data:

```
public class tutro {
    public static Connection conn(){
        try {
            String url = "jdbc:ucanaccess://C:/Users/HP/Documents/IA.accdb";
            Connection conn = DriverManager.getConnection(url);
            return conn;
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, e);
        }
        return null;
    }
}
```

The Java code example presents a Java class "Connector" with its static method `conn()` being a database connector that connects to the Microsoft Access database.

This technique constructs the JDBC URL for the 'UCanAccess' JDBC driver which points toward the Access database file. This step attempts to create a connection with 'DriverManager.getConnection()' and a 'Connection' object is returned if this is successful.

'SQLExceptions' are caught by the exception handling and displayed as error messages through 'JOptionPane.showMessageDialog()'. If an exception occurs, `null` is returned otherwise 'Connection' object is returned.

Use of jFreeChart:

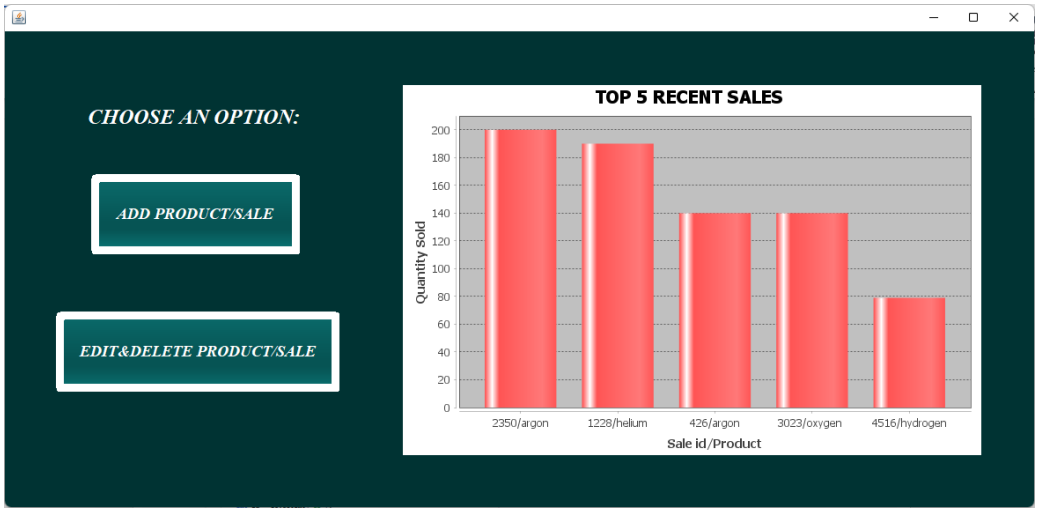

Figure 1. DASHBOARD PAGE

```
public void showChart(){
    try {
    String SQL = "SELECT TOP 5 GASTYPE, QUANTITY, ID FROM ADDSALE ORDER BY QUANTITY DESC";
    pst = conn.prepareStatement(SQL);
    ResultSet rs = pst.executeQuery();

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    JFreeChart chart = ChartFactory.createBarChart("TOP 5 RECENT SALES ", "Sale id/Product", "Quantity Sold", dataset, PlotOrientation.VERTICAL, false, true, false);
    CategoryPlot pr = chart.getCategoryPlot();
    pr.setRangeGridlinePaint(Color.black);

    while (rs.next()) {
        String productName = rs.getString("GASTYPE");
        int salesAmount = rs.getInt("QUANTITY");
        int id = rs.getInt("ID");
        dataset.addValue(salesAmount, "TOP 5 RECENT SALES", id + "/" + productName);
    }
```

This Java snippet applies the technology “JFreeChart” to produce a bar chart of the recent top 5 sales. Initially, we will be executing a query against a `ADDSALE` table in the database which will bring the top 5 records of sales sorted in descending order. SQL query's select statements are used to select the columns `GASTYPE`, `QUANTITY`, and `ID`, these are arranged in descending order with respect to the `QUANTITY` column. After execution of SQL query, the code begins to populate the dataset of the chart. It begins with a `DefaultCategoryDataset` object (`dataset`) which works like a container for the data to be plotted in the chart.

The dataset is now complete and so the code goes on to create a bar graph using the `ChartFactory.createBarChart()` method provided by JFreeChart. The graph is called "TOP 5 RECENT SALES". The x-axis is labeled "Sale id/Product", while the y-axis is labeled "Quantity Sold". The data set that was constructed previously is passed to this function in order to plot the chart with the sales data that has been retrieved.

The code after the chart creation customizes the appearance of the chart. It calls the `GetCategoryPlot` method of the associated chart and sets the color of the range gridlines to black which improves the visual presentation of the chart.

As the code iterates through the `ResultSet` obtained from the executed SQL query, it retrieves the product name, sales quantity, and sale ID for each sales record. For each record, it adds the sales quantity to the dataset along with a label formed by concatenating the sale ID with the product name.

Use of jTable:

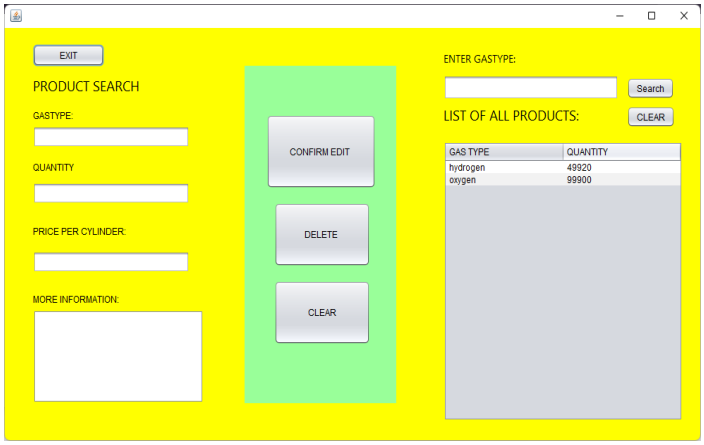

Figure 1. PRODUCT SEARCH/ TABLE OF ALL PRODUCTS

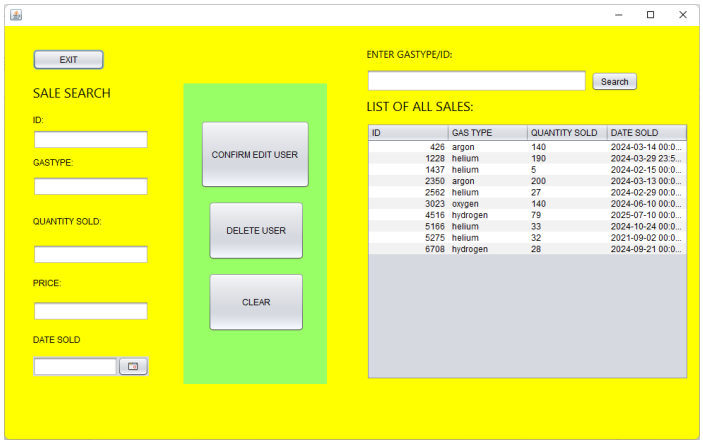

Figure 2. SALES SEARCH/ TABLE OF ALL SALES.

```
public void list(){
    try{

        String SQL="SELECT * FROM ADDPRODUCT";
        pst = conn.prepareStatement(SQL);
        ResultSet rs = pst.executeQuery();

        while(rs.next()){
            String gt = String.valueOf(rs.getString("GASTYPE"));
            String Quan = String.valueOf(rs.getString("QUANTITY"));
            String datatable[] = {gt,Quan};
            DefaultTableModel tbltable = (DefaultTableModel) jTable1.getModel();
            tbltable.addRow(datatable);
        }
    }catch(SQLException e){
        JOptionPane.showMessageDialog(null,e);
    }
}
```

This Java code snippet uses a `JTable` component to show the data retrieved from the database table `ADDPRODUCT`. The `list()` method is the one responsible for populating the `JTable` with data from the `ADDPRODUCT` table. It starts by performing an SQL query to select all the data in the `ADDPRODUCT` table. The `GASTYPE` and `QUANTITY` columns are the retrieved data. They represent the type of gas and its quantity.

The `ResultSet` moves over the retrieved records and the code fetches the values of 'GASTYPE' and 'QUANTITY' one by one. Such values are then converted in strings and stored in variables `gt` (for gas type) and `Quan` (for quantity).

Next the code creates a `string array datatable` which contains the `gt` and `Quan` values for the current row. In order to add the data to the JTable, the code first obtains the table's model by using `jTable1.getModel()` and then casts it to a DefaultTableModel object. This

model also provides the ability of dynamic manipulation of the table's data structure. The `addRow()` method of the `DefaultTableModel` is then called to add the `datatable` array as a new row into the `JTable`. If any SQL error occurs during the query or data population process, the code catches the exception and notifies the user about it using a `JOptionPane`.

Overall word count:970