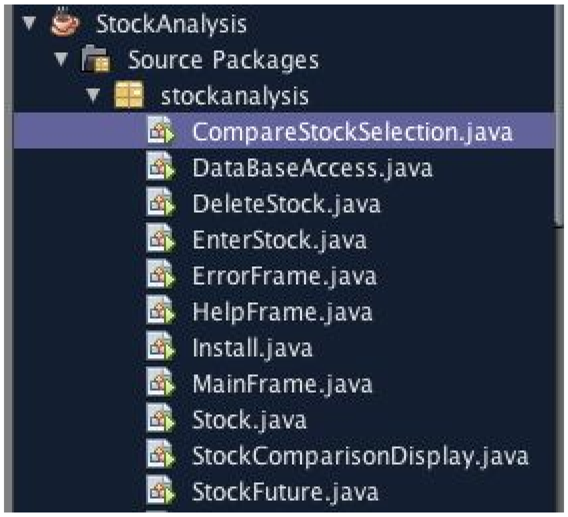
Criterion C: Documentation

Structure of the Product

Figure 1: List of all the classes



```
▼ StockAnalysis
  ▼ Source Packages
    ▼ stockanalysis
        CompareStockSelection.java
        DataBaseAccess.java
        DeleteStock.java
        EnterStock.java
        ErrorFrame.java
        HelpFrame.java
        Install.java
        MainFrame.java
        Stock.java
        StockComparisonDisplay.java
        StockFuture.java
```

Figure 2: The home frame which the user first sees

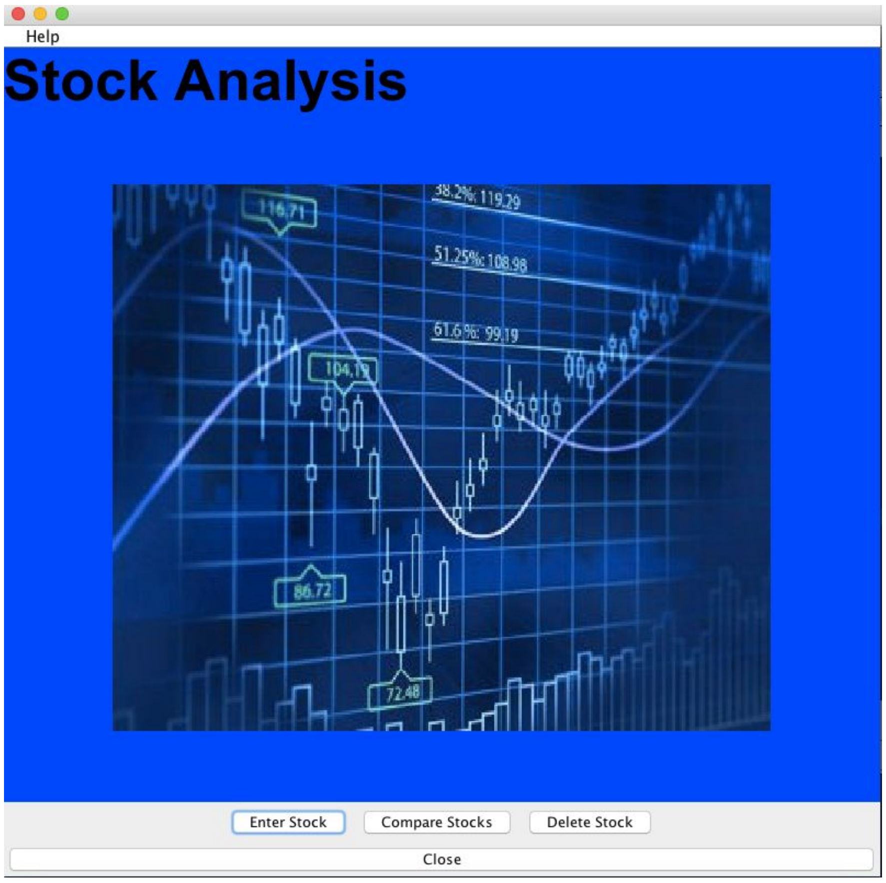
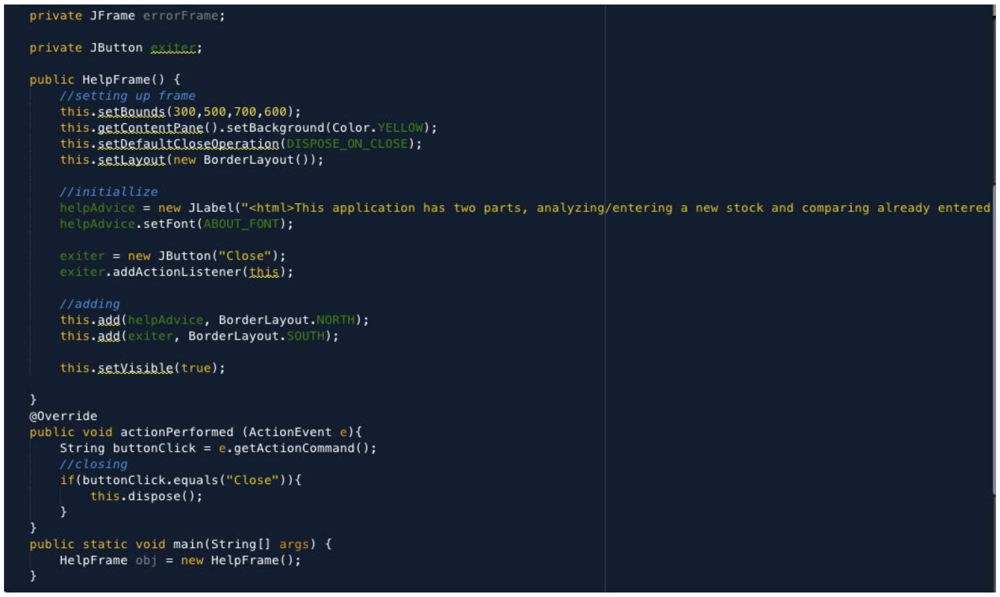
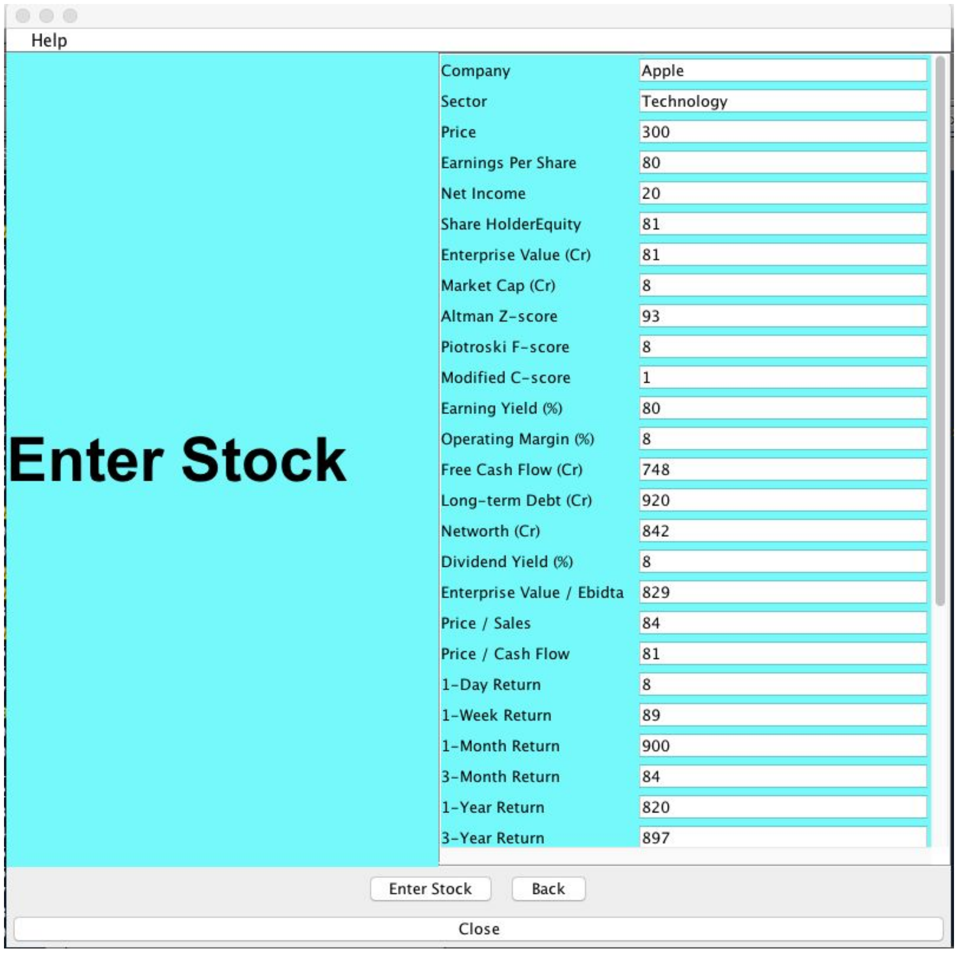


Figure 3: The code for the main frame and it’s buttons



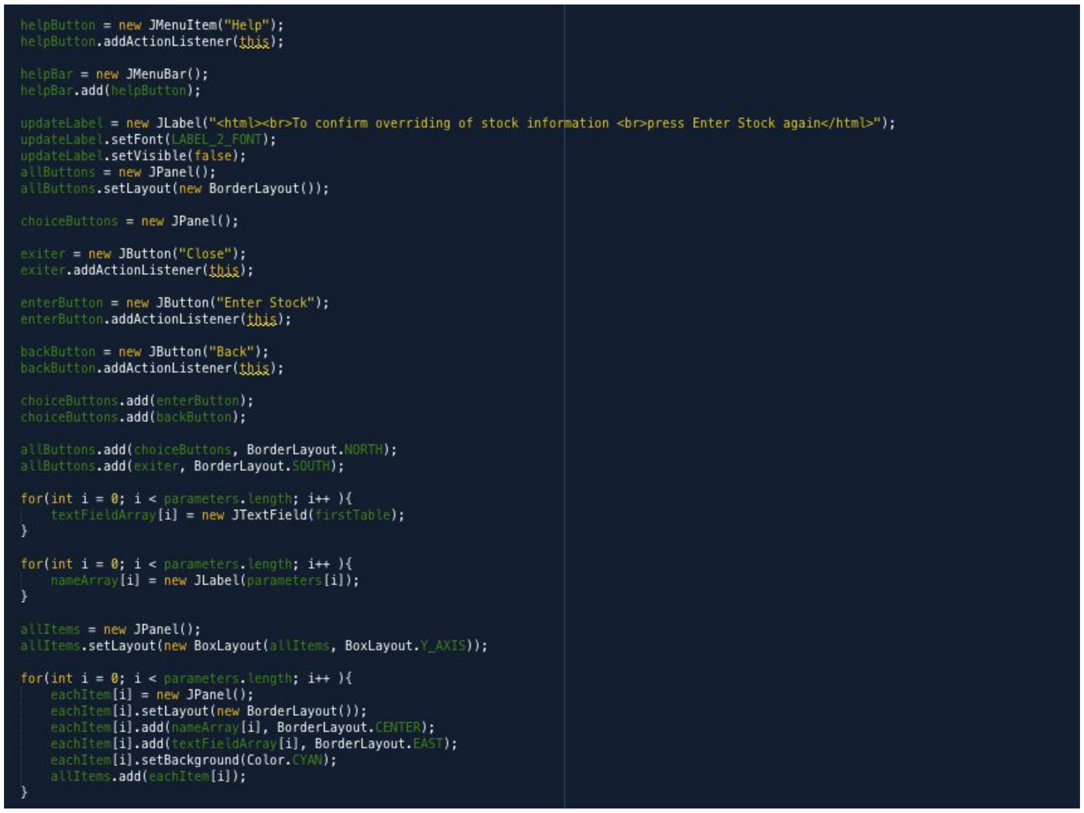
```
private JFrame errorFrame;

private JButton exiter;

public HelpFrame() {
    //setting up frame
    this.setBounds(300,500,700,600);
    this.getContentPane().setBackground(Color.YELLOW);
    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    this.setLayout(new BorderLayout());

    //initiallize
    helpAdvice = new JLabel("<html>This application has two parts, analyzing/entering a new stock and comparing already entered
    helpAdvice.setFont(ABOUT_FONT);

    exiter = new JButton("Close");
    exiter.addActionListener(this);

    //adding
    this.add(helpAdvice, BorderLayout.NORTH);
    this.add(exiter, BorderLayout.SOUTH);

    this.setVisible(true);

}
@Override
public void actionPerformed (ActionEvent e){
    String buttonClick = e.getActionCommand();
    //closing
    if(buttonClick.equals("Close")){
        this.dispose();
    }
}
public static void main(String[] args) {
    HelpFrame obj = new HelpFrame();
}
```

Figure 4: This is the frame where users enter information for a company



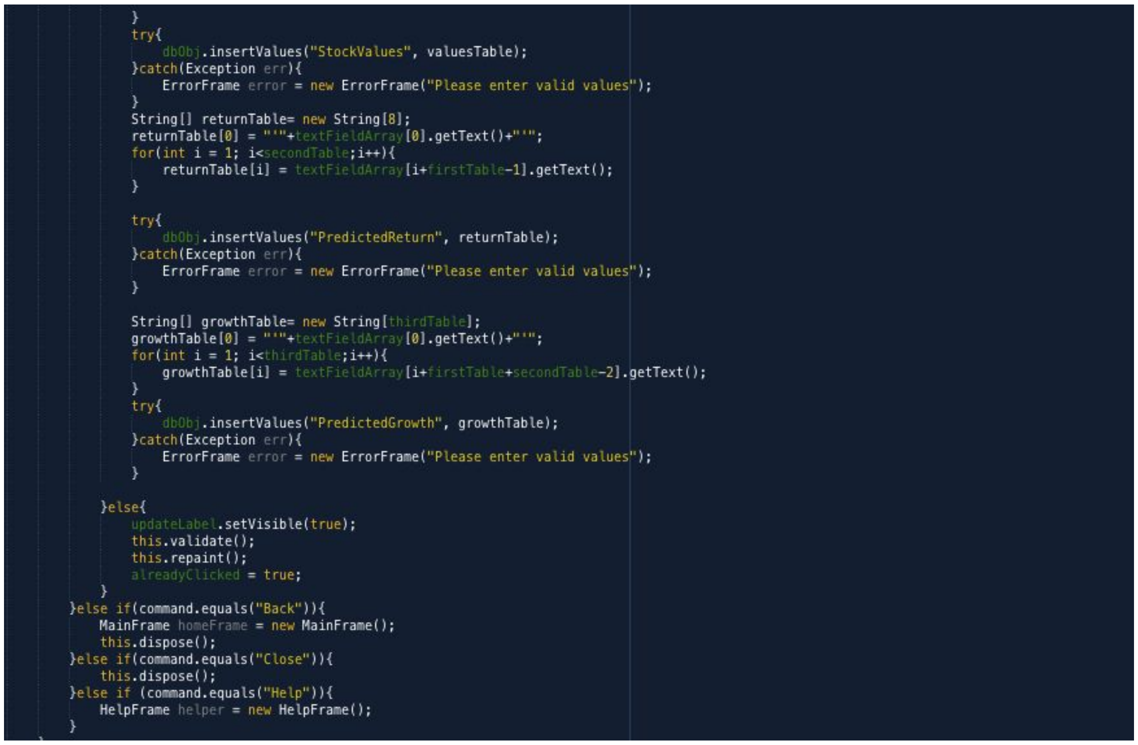
Company	Apple
Sector	Technology
Price	300
Earnings Per Share	80
Net Income	20
Share HolderEquity	81
Enterprise Value (Cr)	81
Market Cap (Cr)	8
Altman Z-score	93
Piotroski F-score	8
Modified C-score	1
Earning Yield (%)	80
Operating Margin (%)	8
Free Cash Flow (Cr)	748
Long-term Debt (Cr)	920
Networth (Cr)	842
Dividend Yield (%)	8
Enterprise Value / Ebidta	829
Price / Sales	84
Price / Cash Flow	81
1-Day Return	8
1-Week Return	89
1-Month Return	900
3-Month Return	84
1-Year Return	820
3-Year Return	897

Figure 5:This is the code that is used to add multiple text fields/buttons



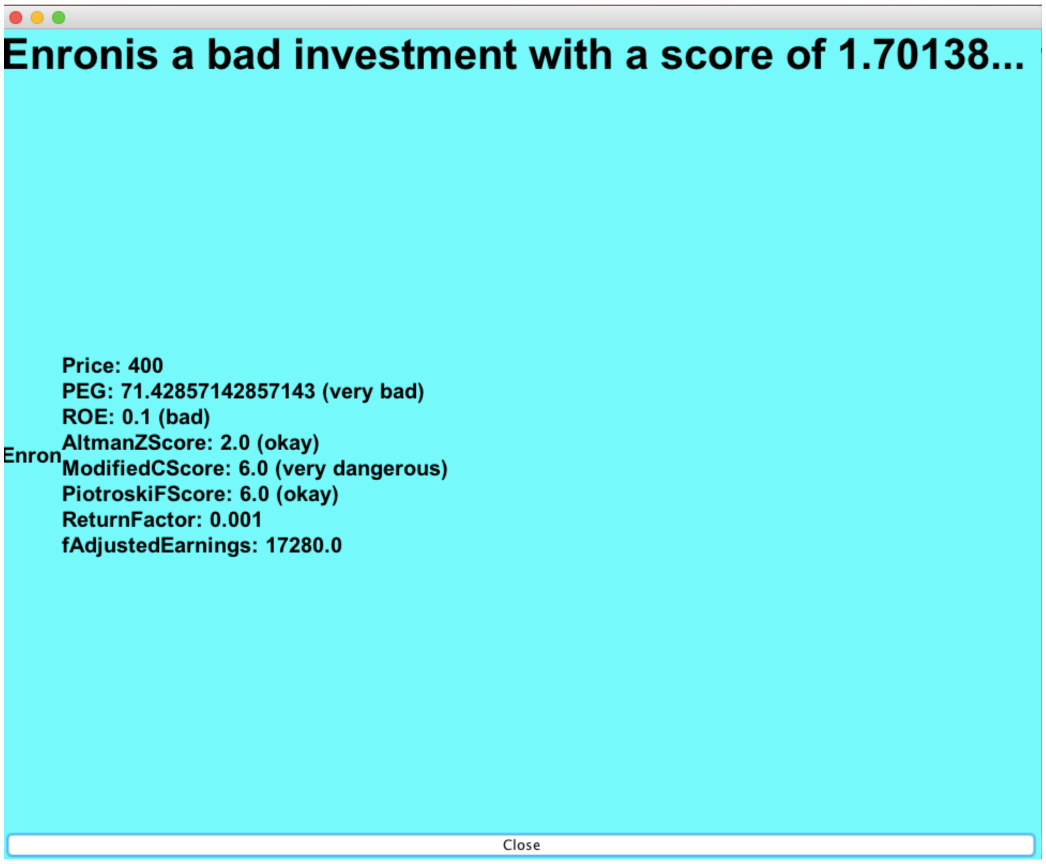
```java
helpButton = new JMenuItem("Help");
helpButton.addActionListener(this);

helpBar = new JMenuBar();
helpBar.add(helpButton);

updateLabel = new JLabel("<html><br>To confirm overriding of stock information <br>press Enter Stock again</html>");
updateLabel.setFont(LABEL_2_FONT);
updateLabel.setVisible(false);
allButtons = new JPanel();
allButtons.setLayout(new BorderLayout());

choiceButtons = new JPanel();

exiter = new JButton("Close");
exiter.addActionListener(this);

enterButton = new JButton("Enter Stock");
enterButton.addActionListener(this);

backButton = new JButton("Back");
backButton.addActionListener(this);

choiceButtons.add(enterButton);
choiceButtons.add(backButton);

allButtons.add(choiceButtons, BorderLayout.NORTH);
allButtons.add(exiter, BorderLayout.SOUTH);

for(int i = 0; i < parameters.length; i++ ){
    textFieldArray[i] = new JTextField(firstTable);
}

for(int i = 0; i < parameters.length; i++ ){
    nameArray[i] = new JLabel(parameters[i]);
}

allItems = new JPanel();
allItems.setLayout(new BoxLayout(allItems, BoxLayout.Y_AXIS));

for(int i = 0; i < parameters.length; i++ ){
    eachItem[i] = new JPanel();
    eachItem[i].setLayout(new BorderLayout());
    eachItem[i].add(nameArray[i], BorderLayout.CENTER);
    eachItem[i].add(textFieldArray[i], BorderLayout.EAST);
    eachItem[i].setBackground(Color.CYAN);
    allItems.add(eachItem[i]);
}
```

Figure 6: This is the code for the buttons



```java
                }
                try{
                    dbObj.insertValues("StockValues", valuesTable);
                }catch(Exception err){
                    ErrorFrame error = new ErrorFrame("Please enter valid values");
                }
                String[] returnTable= new String[8];
                returnTable[0] = "'"+textFieldArray[0].getText()+"'";
                for(int i = 1; i<secondTable;i++){
                    returnTable[i] = textFieldArray[i+firstTable-1].getText();
                }

                try{
                    dbObj.insertValues("PredictedReturn", returnTable);
                }catch(Exception err){
                    ErrorFrame error = new ErrorFrame("Please enter valid values");
                }

                String[] growthTable= new String[thirdTable];
                growthTable[0] = "'"+textFieldArray[0].getText()+"'";
                for(int i = 1; i<thirdTable;i++){
                    growthTable[i] = textFieldArray[i+firstTable+secondTable-2].getText();
                }
                try{
                    dbObj.insertValues("PredictedGrowth", growthTable);
                }catch(Exception err){
                    ErrorFrame error = new ErrorFrame("Please enter valid values");
                }

            }else{
                updateLabel.setVisible(true);
                this.validate();
                this.repaint();
                alreadyClicked = true;
            }
        }else if(command.equals("Back")){
            MainFrame homeFrame = new MainFrame();
            this.dispose();
        }else if(command.equals("Close")){
            this.dispose();
        }else if (command.equals("Help")){
            HelpFrame helper = new HelpFrame();
        }
```

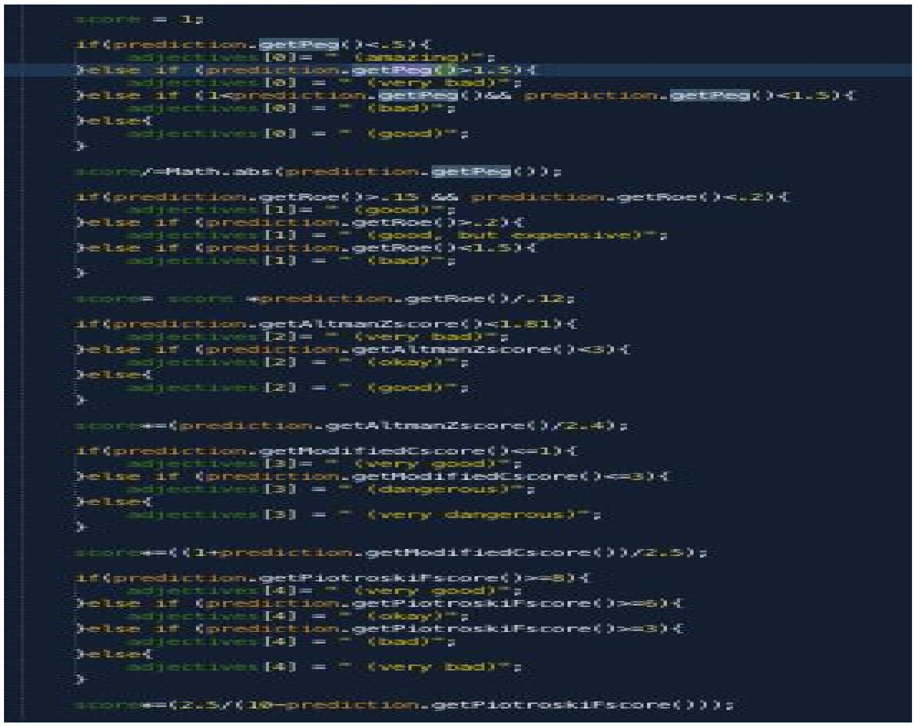
Figure 7: This is the frame after stock entry which evaluates a stock



Enronis a bad investment with a score of 1.70138...

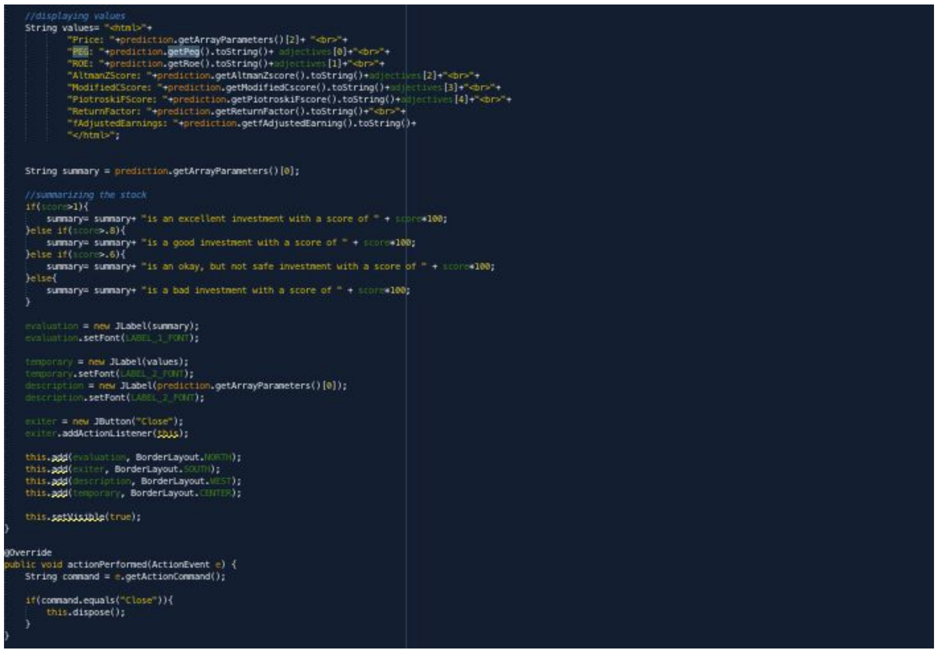
```
       Price: 400
       PEG: 71.42857142857143 (very bad)
       ROE: 0.1 (bad)
       AltmanZScore: 2.0 (okay)
Enron  ModifiedCScore: 6.0 (very dangerous)
       PiotroskiFScore: 6.0 (okay)
       ReturnFactor: 0.001
       fAdjustedEarnings: 17280.0
```

Figure 8: This code calculates, comments, and allows the buttons to work



```
score = 1;

if(prediction.getPeg()<.5){
    adjectives[0]= " (amazing)";
}else if (prediction.getPeg()>1.5){
    adjectives[0] = " (very bad)";
}else if (1<prediction.getPeg()&& prediction.getPeg()<1.5){
    adjectives[0] = " (bad)";
}else{
    adjectives[0] = " (good)";
}

score/=Math.abs(prediction.getPeg());

if(prediction.getRoe()>.15 && prediction.getRoe()<.2){
    adjectives[1]= " (good)";
}else if (prediction.getRoe()>.2){
    adjectives[1] = " (good, but expensive)";
}else if (prediction.getRoe()<1.5){
    adjectives[1] = " (bad)";
}

score= score *prediction.getRoe()/.12;

if(prediction.getAltmanZscore()<1.81){
    adjectives[2]= " (very bad)";
}else if (prediction.getAltmanZscore()<3){
    adjectives[2] = " (okay)";
}else{
    adjectives[2] = " (good)";
}

score*=(prediction.getAltmanZscore()/2.4);

if(prediction.getModifiedCscore()<=1){
    adjectives[3]= " (very good)";
}else if (prediction.getModifiedCscore()<=3){
    adjectives[3] = " (dangerous)";
}else{
    adjectives[3] = " (very dangerous)";
}

score*=((1+prediction.getModifiedCscore())/2.5);

if(prediction.getPiotroskiFscore()>=8){
    adjectives[4]= " (very good)";
}else if (prediction.getPiotroskiFscore()>=6){
    adjectives[4] = " (okay)";
}else if (prediction.getPiotroskiFscore()>=3){
    adjectives[4] = " (bad)";
}else{
    adjectives[4] = " (very bad)";
}

score*=(2.5/(10-prediction.getPiotroskiFscore()));
```

Figure 8 continued



```
    //displaying values
    String values= "<html>"+
            "Price: "+prediction.getArrayParameters()[2]+ "<br>"+
            "PEG: "+prediction.getPeg().toString()+ adjectives[0]+"<br>"+
            "ROE: "+prediction.getRoe().toString()+adjectives[1]+"<br>"+
            "AltmanZScore: "+prediction.getAltmanZscore().toString()+adjectives[2]+"<br>"+
            "ModifiedCScore: "+prediction.getModifiedCscore().toString()+adjectives[3]+"<br>"+
            "PiotroskiFScore: "+prediction.getPiotroskiFscore().toString()+adjectives[4]+"<br>"+
            "ReturnFactor: "+prediction.getReturnFactor().toString()+"<br>"+
            "fAdjustedEarnings: "+prediction.getfAdjustedEarning().toString()+
            "</html>";


    String summary = prediction.getArrayParameters()[0];

    //summarizing the stock
    if(score>1){
        summary= summary+ "is an excellent investment with a score of " + score*100;
    }else if(score>.8){
        summary= summary+ "is a good investment with a score of " + score*100;
    }else if(score>.6){
        summary= summary+ "is an okay, but not safe investment with a score of " + score*100;
    }else{
        summary= summary+ "is a bad investment with a score of " + score*100;
    }

    evaluation = new JLabel(summary);
    evaluation.setFont(LABEL_1_FONT);

    temporary = new JLabel(values);
    temporary.setFont(LABEL_2_FONT);
    description = new JLabel(prediction.getArrayParameters()[0]);
    description.setFont(LABEL_2_FONT);

    exiter = new JButton("Close");
    exiter.addActionListener(this);

    this.add(evaluation, BorderLayout.NORTH);
    this.add(exiter, BorderLayout.SOUTH);
    this.add(description, BorderLayout.WEST);
    this.add(temporary, BorderLayout.CENTER);

    this.setVisible(true);
}

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

    if(command.equals("Close")){
        this.dispose();
    }
}
```

Figure 9: This is where the user selects stocks to compare

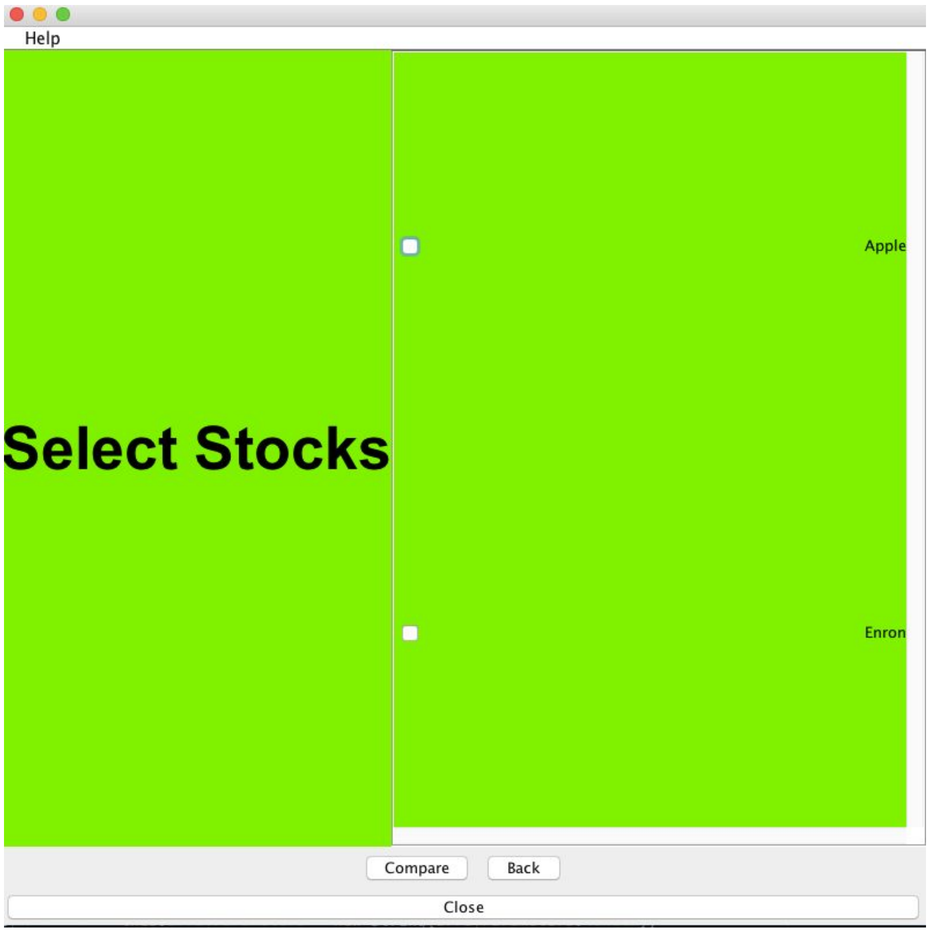
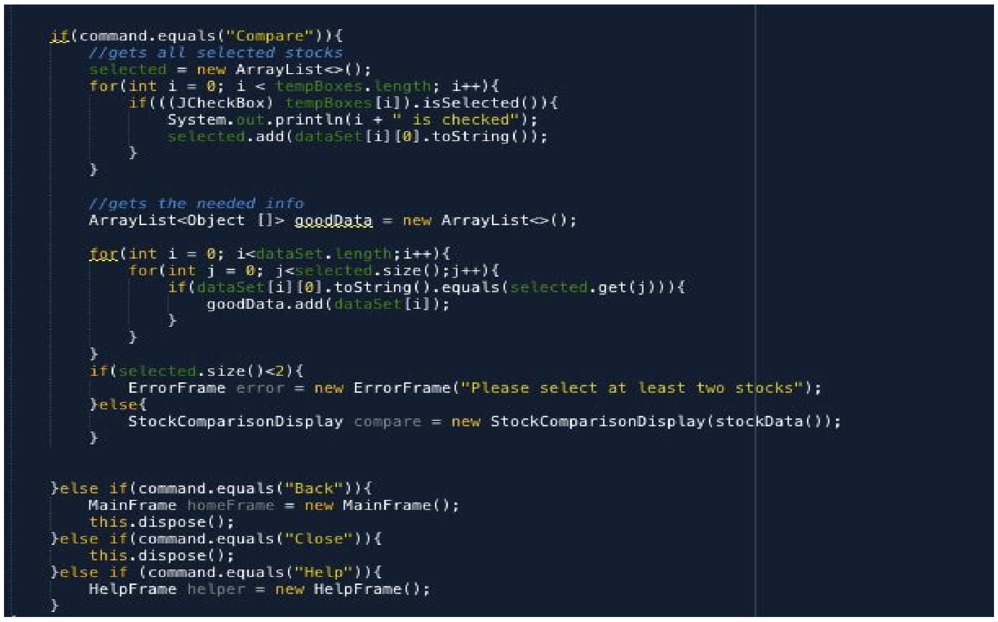
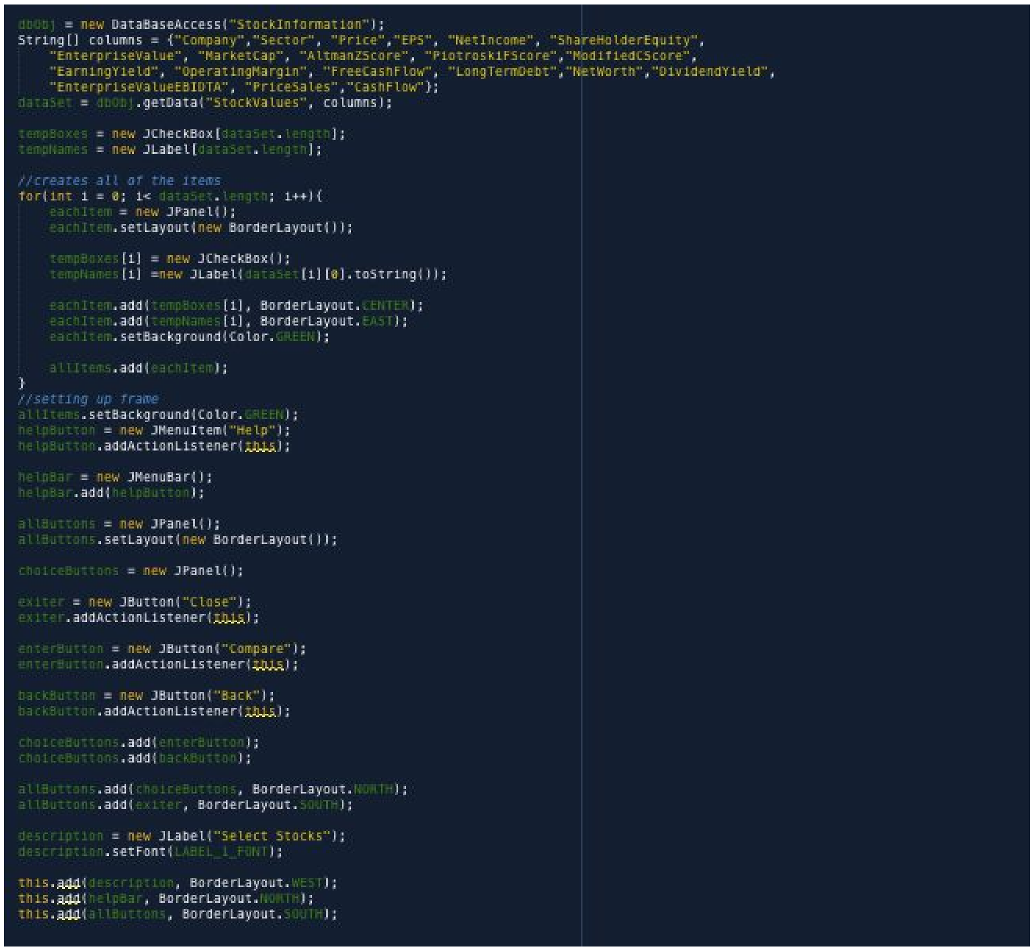


Figure 10: This code is for the buttons, it gets selected info and compares it



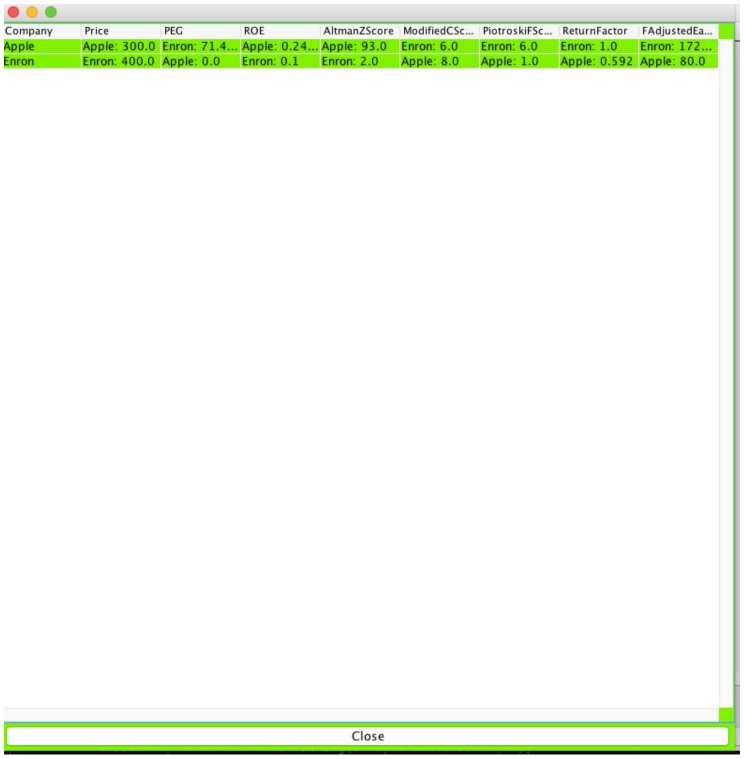
```
if(command.equals("Compare")){
    //gets all selected stocks
    selected = new ArrayList<>();
    for(int i = 0; i < tempBoxes.length; i++){
        if(((JCheckBox) tempBoxes[i]).isSelected()){
            System.out.println(i + " is checked");
            selected.add(dataSet[i][0].toString());
        }
    }

    //gets the needed info
    ArrayList<Object []> goodData = new ArrayList<>();

    for(int i = 0; i<dataSet.length;i++){
        for(int j = 0; j<selected.size();j++){
            if(dataSet[i][0].toString().equals(selected.get(j))){
                goodData.add(dataSet[i]);
            }
        }
    }
    if(selected.size()<2){
        ErrorFrame error = new ErrorFrame("Please select at least two stocks");
    }else{
        StockComparisonDisplay compare = new StockComparisonDisplay(stockData());
    }


}else if(command.equals("Back")){
    MainFrame homeFrame = new MainFrame();
    this.dispose();
}else if(command.equals("Close")){
    this.dispose();
}else if (command.equals("Help")){
    HelpFrame helper = new HelpFrame();
}
```

Figure 11:Code for adding parts to the panel, and the selection for each stock



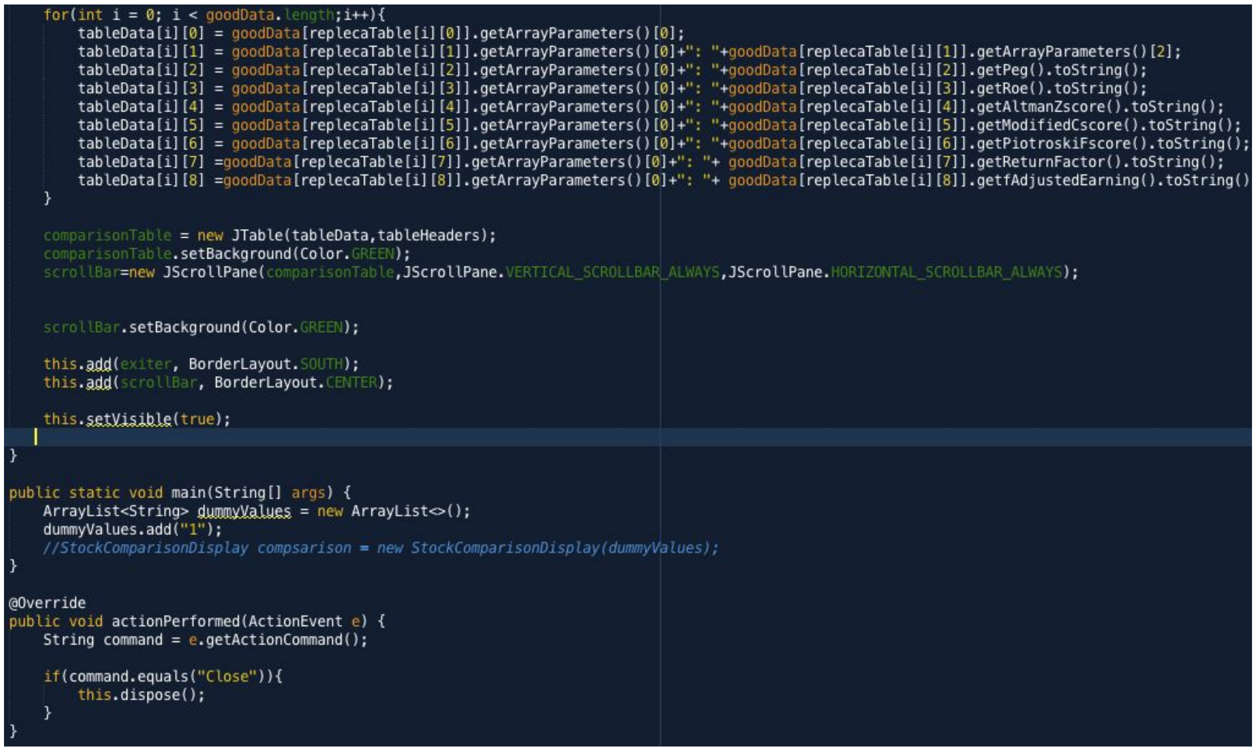
```java
dbObj = new DataBaseAccess("StockInformation");
String[] columns = {"Company","Sector", "Price","EPS", "NetIncome", "ShareHolderEquity",
    "EnterpriseValue", "MarketCap", "AltmanZScore", "PiotroskiFScore","ModifiedCScore",
    "EarningYield", "OperatingMargin", "FreeCashFlow", "LongTermDebt","NetWorth","DividendYield",
    "EnterpriseValueEBIDTA", "PriceSales","CashFlow"};
dataSet = dbObj.getData("StockValues", columns);

tempBoxes = new JCheckBox[dataSet.length];
tempNames = new JLabel[dataSet.length];

//creates all of the items
for(int i = 0; i< dataSet.length; i++){
    eachItem = new JPanel();
    eachItem.setLayout(new BorderLayout());

    tempBoxes[i] = new JCheckBox();
    tempNames[i] =new JLabel(dataSet[i][0].toString());

    eachItem.add(tempBoxes[i], BorderLayout.CENTER);
    eachItem.add(tempNames[i], BorderLayout.EAST);
    eachItem.setBackground(Color.GREEN);

    allItems.add(eachItem);
}
//setting up frame
allItems.setBackground(Color.GREEN);
helpButton = new JMenuItem("Help");
helpButton.addActionListener(this);

helpBar = new JMenuBar();
helpBar.add(helpButton);

allButtons = new JPanel();
allButtons.setLayout(new BorderLayout());

choiceButtons = new JPanel();

exiter = new JButton("Close");
exiter.addActionListener(this);

enterButton = new JButton("Compare");
enterButton.addActionListener(this);

backButton = new JButton("Back");
backButton.addActionListener(this);

choiceButtons.add(enterButton);
choiceButtons.add(backButton);

allButtons.add(choiceButtons, BorderLayout.NORTH);
allButtons.add(exiter, BorderLayout.SOUTH);

description = new JLabel("Select Stocks");
description.setFont(LABEL_1_FONT);

this.add(description, BorderLayout.WEST);
this.add(helpBar, BorderLayout.NORTH);
this.add(allButtons, BorderLayout.SOUTH);
```

Figure 12: Display that orders stock info and compares it



Company	Price	PEG	ROE	AltmanZScore	ModifiedCSc...	PiotroskiFSc...	ReturnFactor	FAdjustedEa...
Apple	Apple: 300.0	Enron: 71.4...	Apple: 0.24...	Apple: 93.0	Enron: 6.0	Enron: 6.0	Enron: 1.0	Enron: 172...
Enron	Enron: 400.0	Apple: 0.0	Enron: 0.1	Enron: 2.0	Apple: 8.0	Apple: 1.0	Apple: 0.592	Apple: 80.0

Figure 13: This is the code for putting data into the table and the exiting button



```
    for(int i = 0; i < goodData.length;i++){
        tableData[i][0] = goodData[replecaTable[i][0]].getArrayParameters()[0];
        tableData[i][1] = goodData[replecaTable[i][1]].getArrayParameters()[0]+": "+goodData[replecaTable[i][1]].getArrayParameters()[2];
        tableData[i][2] = goodData[replecaTable[i][2]].getArrayParameters()[0]+": "+goodData[replecaTable[i][2]].getPeg().toString();
        tableData[i][3] = goodData[replecaTable[i][3]].getArrayParameters()[0]+": "+goodData[replecaTable[i][3]].getRoe().toString();
        tableData[i][4] = goodData[replecaTable[i][4]].getArrayParameters()[0]+": "+goodData[replecaTable[i][4]].getAltmanZscore().toString();
        tableData[i][5] = goodData[replecaTable[i][5]].getArrayParameters()[0]+": "+goodData[replecaTable[i][5]].getModifiedCscore().toString();
        tableData[i][6] = goodData[replecaTable[i][6]].getArrayParameters()[0]+": "+goodData[replecaTable[i][6]].getPiotroskiFscore().toString();
        tableData[i][7] =goodData[replecaTable[i][7]].getArrayParameters()[0]+": "+ goodData[replecaTable[i][7]].getReturnFactor().toString();
        tableData[i][8] =goodData[replecaTable[i][8]].getArrayParameters()[0]+": "+ goodData[replecaTable[i][8]].getfAdjustedEarning().toString()
    }

    comparisonTable = new JTable(tableData,tableHeaders);
    comparisonTable.setBackground(Color.GREEN);
    scrollBar=new JScrollPane(comparisonTable,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);


    scrollBar.setBackground(Color.GREEN);

    this.add(exiter, BorderLayout.SOUTH);
    this.add(scrollBar, BorderLayout.CENTER);

    this.setVisible(true);

}

public static void main(String[] args) {
    ArrayList<String> dummyValues = new ArrayList<>();
    dummyValues.add("1");
    //StockComparisonDisplay compsarison = new StockComparisonDisplay(dummyValues);
}

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

    if(command.equals("Close")){
        this.dispose();
    }
}
```

Figure 14: Frame where the user selects stocks to delete

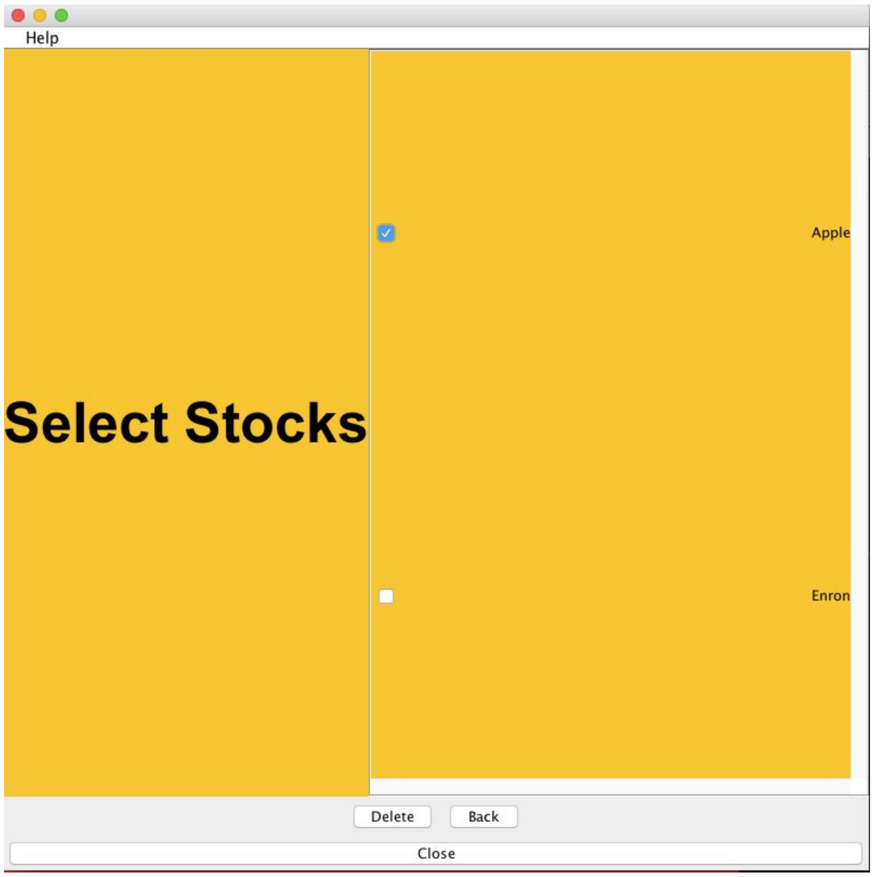
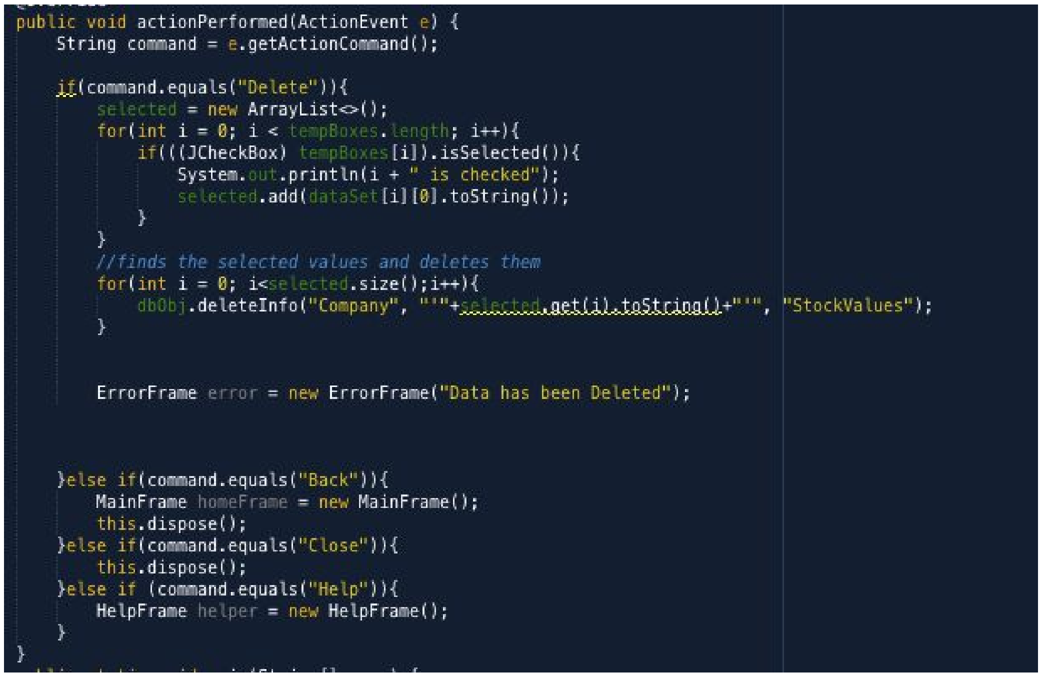
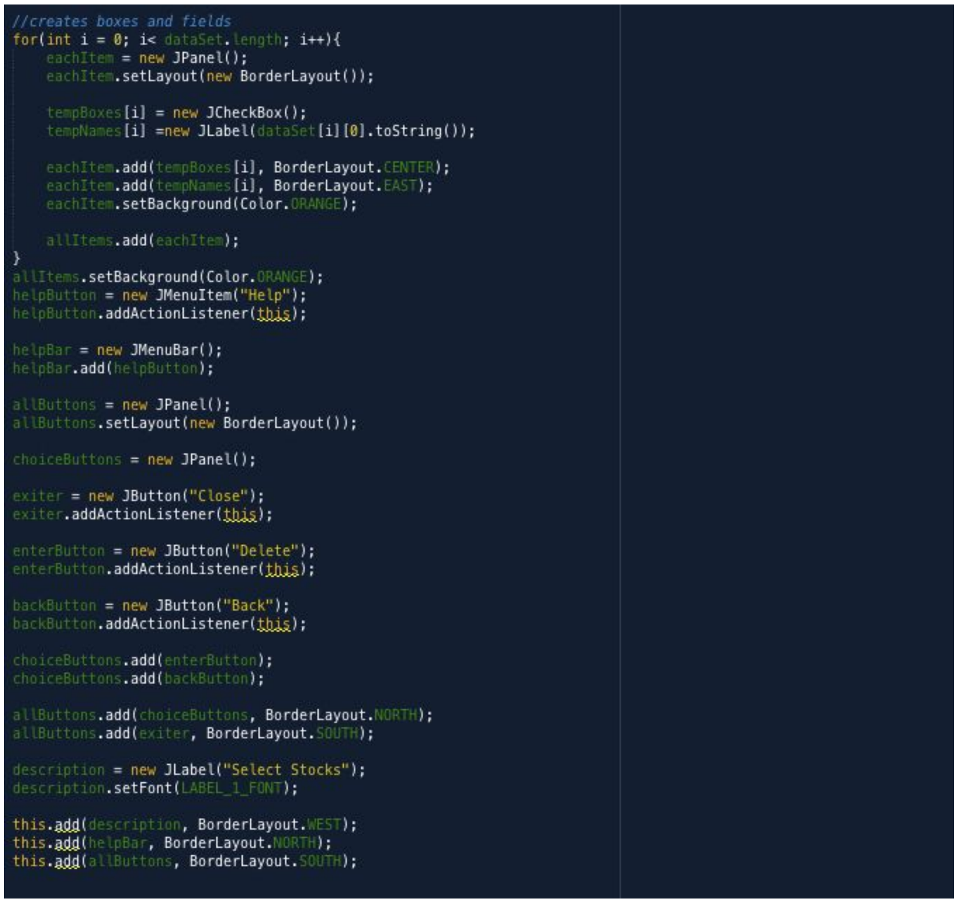


Figure 15: Code for the buttons, that finds the selected stocks and deletes them



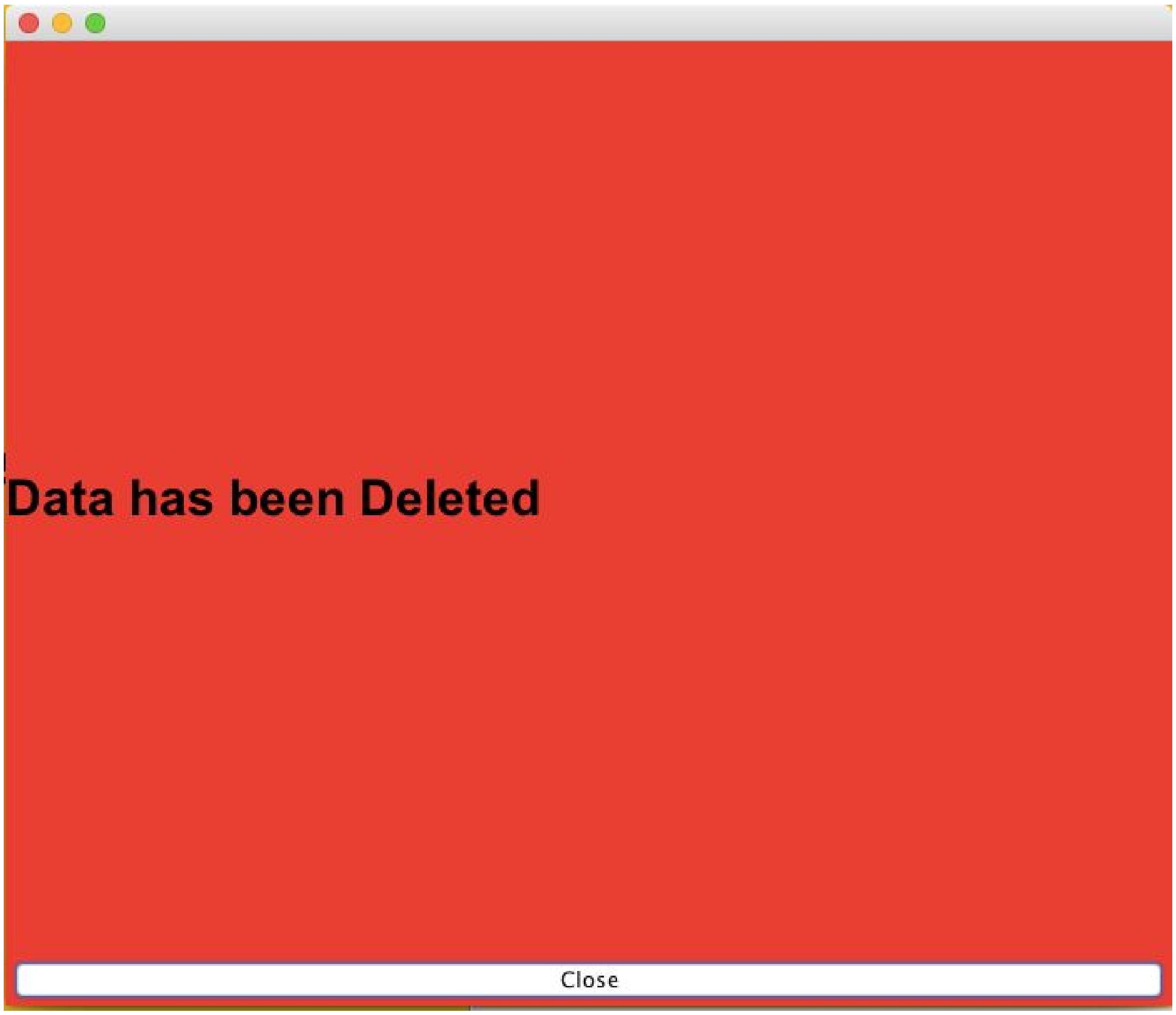
```java
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if(command.equals("Delete")){
        selected = new ArrayList<>();
        for(int i = 0; i < tempBoxes.length; i++){
            if(((JCheckBox) tempBoxes[i]).isSelected()){
                System.out.println(i + " is checked");
                selected.add(dataSet[i][0].toString());
            }
        }
        //finds the selected values and deletes them
        for(int i = 0; i<selected.size();i++){
            dbObj.deleteInfo("Company", "'"+selected.get(i).toString()+"'", "StockValues");
        }


        ErrorFrame error = new ErrorFrame("Data has been Deleted");



    }else if(command.equals("Back")){
        MainFrame homeFrame = new MainFrame();
        this.dispose();
    }else if(command.equals("Close")){
        this.dispose();
    }else if (command.equals("Help")){
        HelpFrame helper = new HelpFrame();
    }
}
```

Figure 16: Code for adding items to the frame, and for each stock



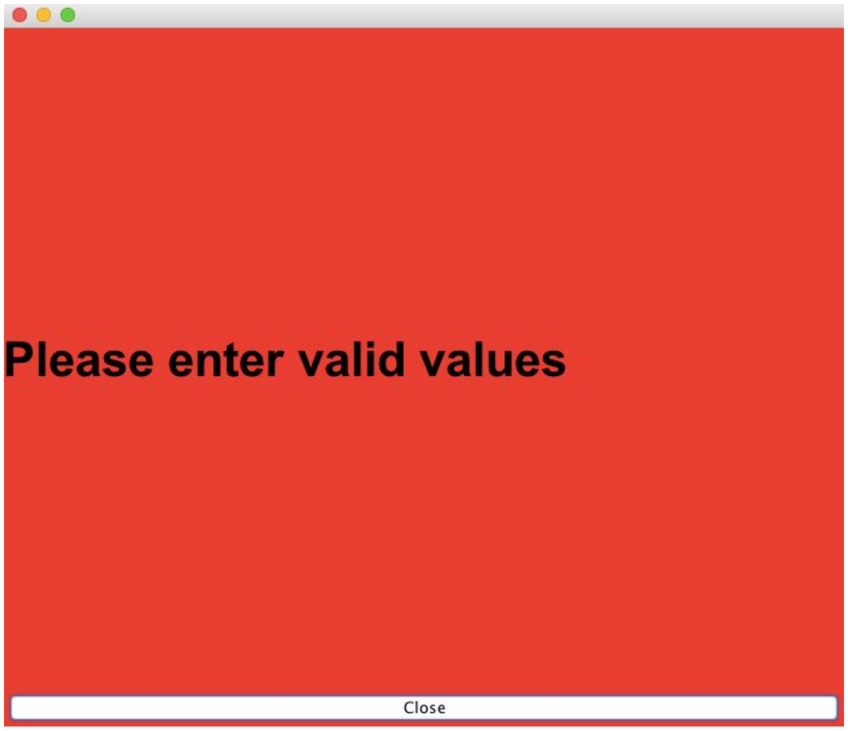
```java
//creates boxes and fields
for(int i = 0; i< dataSet.length; i++){
    eachItem = new JPanel();
    eachItem.setLayout(new BorderLayout());

    tempBoxes[i] = new JCheckBox();
    tempNames[i] =new JLabel(dataSet[i][0].toString());

    eachItem.add(tempBoxes[i], BorderLayout.CENTER);
    eachItem.add(tempNames[i], BorderLayout.EAST);
    eachItem.setBackground(Color.ORANGE);

    allItems.add(eachItem);
}
allItems.setBackground(Color.ORANGE);
helpButton = new JMenuItem("Help");
helpButton.addActionListener(this);

helpBar = new JMenuBar();
helpBar.add(helpButton);

allButtons = new JPanel();
allButtons.setLayout(new BorderLayout());

choiceButtons = new JPanel();

exiter = new JButton("Close");
exiter.addActionListener(this);

enterButton = new JButton("Delete");
enterButton.addActionListener(this);

backButton = new JButton("Back");
backButton.addActionListener(this);

choiceButtons.add(enterButton);
choiceButtons.add(backButton);

allButtons.add(choiceButtons, BorderLayout.NORTH);
allButtons.add(exiter, BorderLayout.SOUTH);

description = new JLabel("Select Stocks");
description.setFont(LABEL_1_FONT);

this.add(description, BorderLayout.WEST);
this.add(helpBar, BorderLayout.NORTH);
this.add(allButtons, BorderLayout.SOUTH);
```

Figure 17: This confirms that data has been deleted



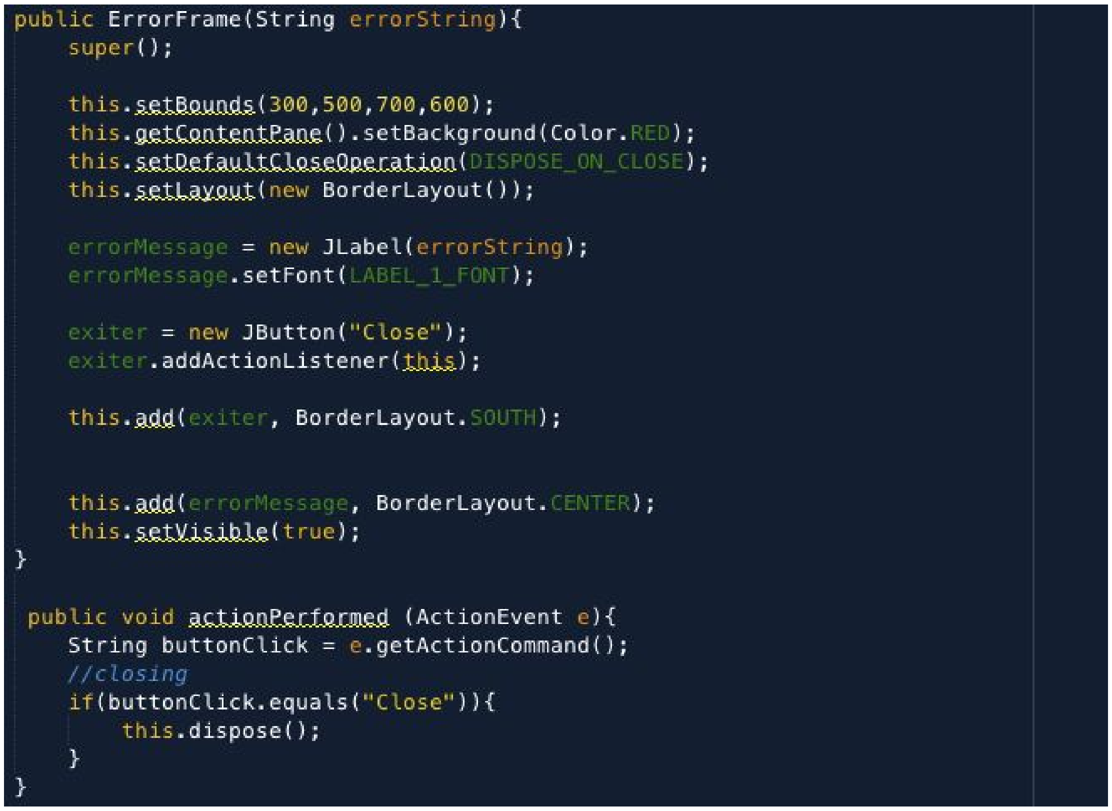
Data has been Deleted

Figure 18: This frame warns the user of an error



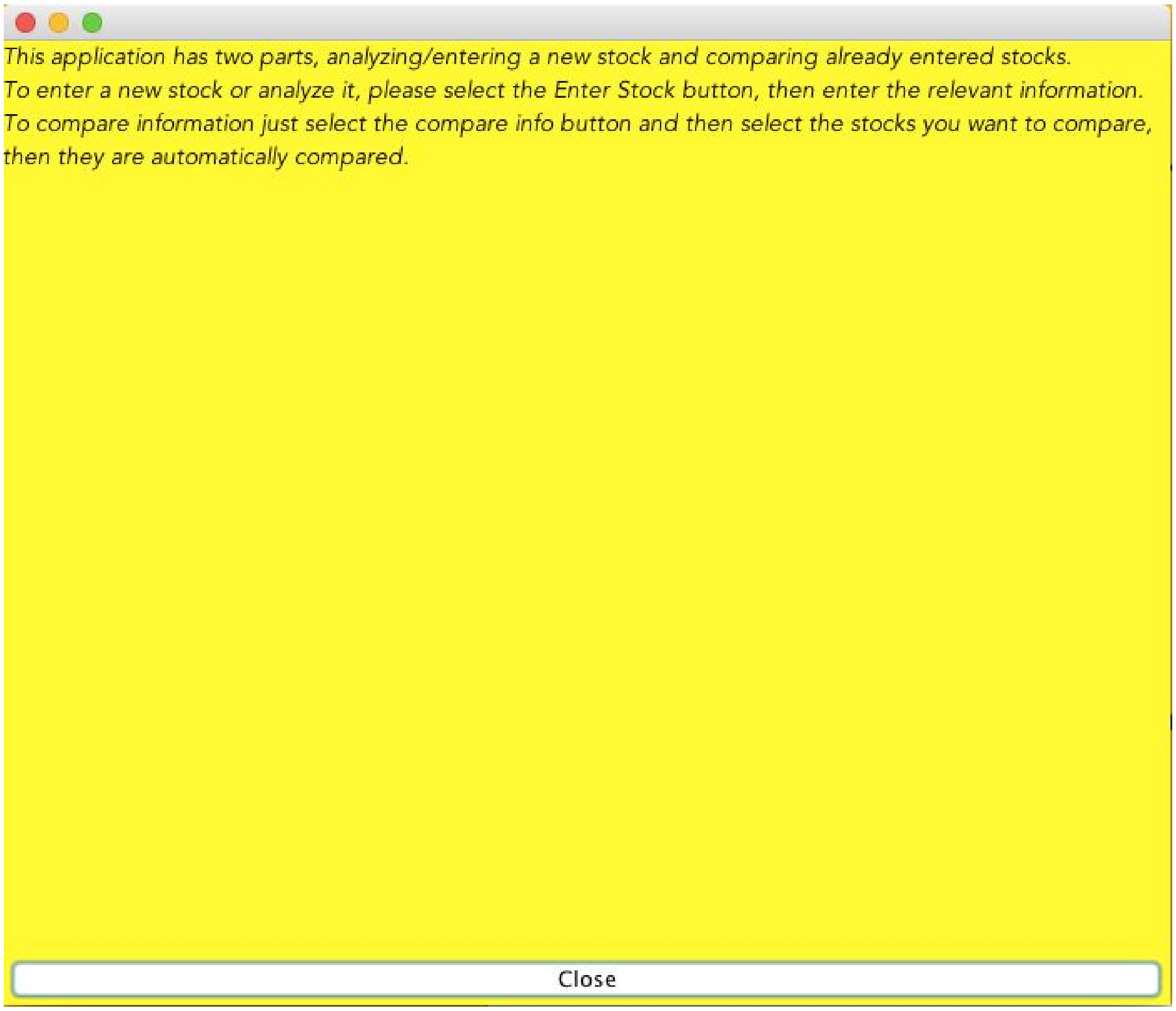
Please enter valid values

Figure 19: This code receives an error message and displays it



```java
public ErrorFrame(String errorString){
    super();

    this.setBounds(300,500,700,600);
    this.getContentPane().setBackground(Color.RED);
    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    this.setLayout(new BorderLayout());

    errorMessage = new JLabel(errorString);
    errorMessage.setFont(LABEL_1_FONT);

    exiter = new JButton("Close");
    exiter.addActionListener(this);

    this.add(exiter, BorderLayout.SOUTH);


    this.add(errorMessage, BorderLayout.CENTER);
    this.setVisible(true);
}

 public void actionPerformed (ActionEvent e){
    String buttonClick = e.getActionCommand();
    //closing
    if(buttonClick.equals("Close")){
        this.dispose();
    }
}
```

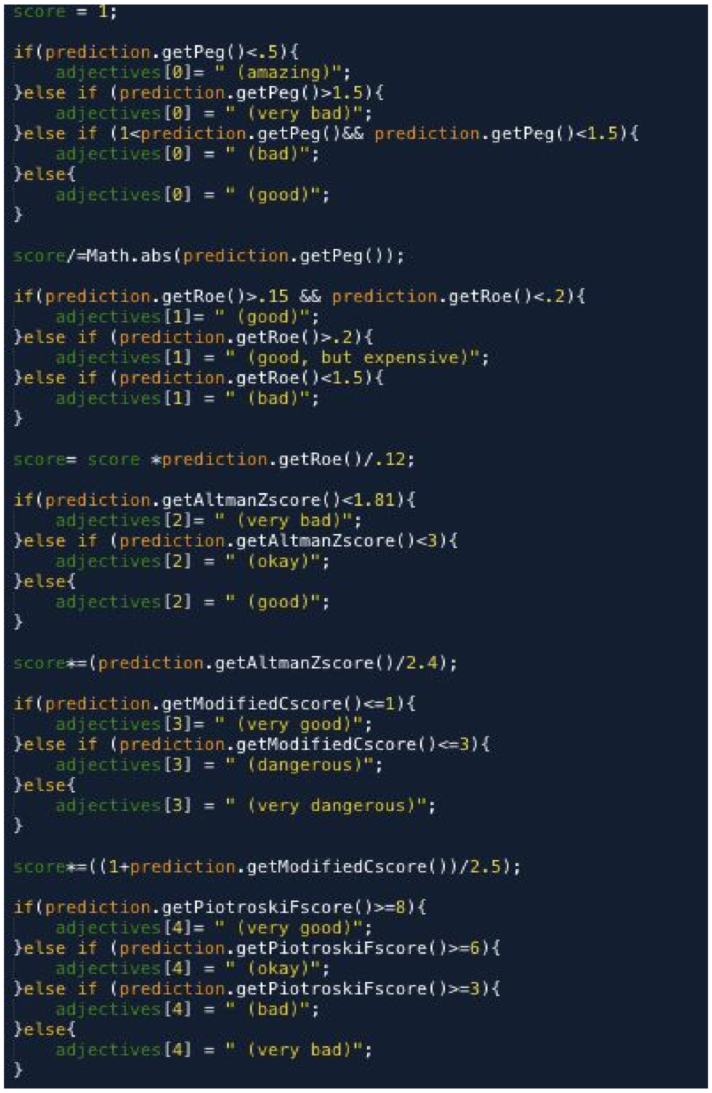
Figure 20: Help frame that can be accessed anywhere with the “help” button



This application has two parts, analyzing/entering a new stock and comparing already entered stocks.
To enter a new stock or analyze it, please select the Enter Stock button, then enter the relevant information.
To compare information just select the compare info button and then select the stocks you want to compare, then they are automatically compared.

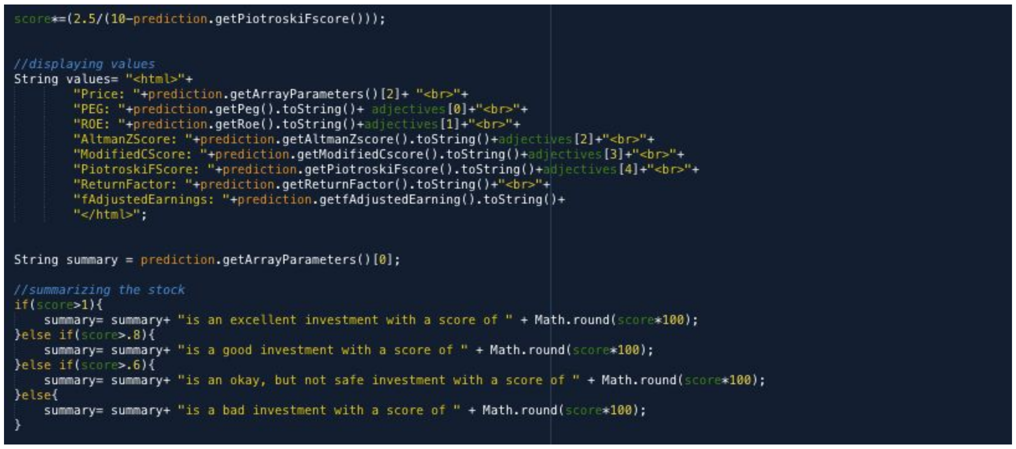
Evidence of Algorithmic Thinking

Figure 1.1: Stock Future evaluation



```
score = 1;

if(prediction.getPeg()<.5){
    adjectives[0]= " (amazing)";
}else if (prediction.getPeg()>1.5){
    adjectives[0] = " (very bad)";
}else if (1<prediction.getPeg()&& prediction.getPeg()<1.5){
    adjectives[0] = " (bad)";
}else{
    adjectives[0] = " (good)";
}

score/=Math.abs(prediction.getPeg());

if(prediction.getRoe()>.15 && prediction.getRoe()<.2){
    adjectives[1]= " (good)";
}else if (prediction.getRoe()>.2){
    adjectives[1] = " (good, but expensive)";
}else if (prediction.getRoe()<1.5){
    adjectives[1] = " (bad)";
}

score= score *prediction.getRoe()/.12;

if(prediction.getAltmanZscore()<1.81){
    adjectives[2]= " (very bad)";
}else if (prediction.getAltmanZscore()<3){
    adjectives[2] = " (okay)";
}else{
    adjectives[2] = " (good)";
}

score*=(prediction.getAltmanZscore()/2.4);

if(prediction.getModifiedCscore()<=1){
    adjectives[3]= " (very good)";
}else if (prediction.getModifiedCscore()<=3){
    adjectives[3] = " (dangerous)";
}else{
    adjectives[3] = " (very dangerous)";
}

score*=((1+prediction.getModifiedCscore())/2.5);

if(prediction.getPiotroskiFscore()>=8){
    adjectives[4]= " (very good)";
}else if (prediction.getPiotroskiFscore()>=6){
    adjectives[4] = " (okay)";
}else if (prediction.getPiotroskiFscore()>=3){
    adjectives[4] = " (bad)";
}else{
    adjectives[4] = " (very bad)";
}
```

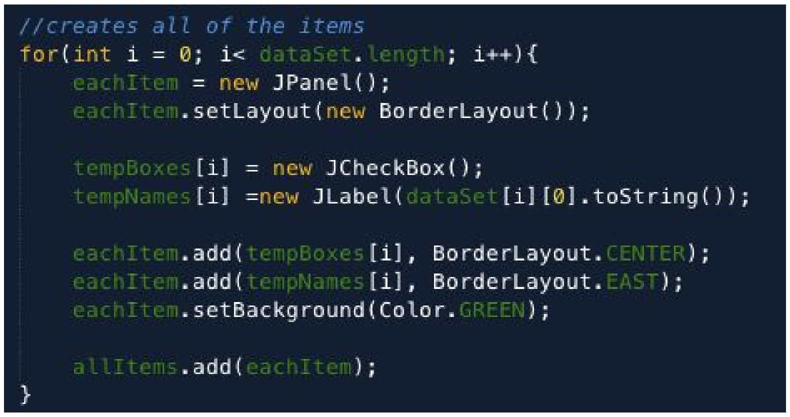
Figure 1.2: Stock Future evaluation continued



```
score*=(2.5/(10-prediction.getPiotroskiFscore()));


//displaying values
String values= "<html>"+
        "Price: "+prediction.getArrayParameters()[2]+ "<br>"+
        "PEG: "+prediction.getPeg().toString()+ adjectives[0]+"<br>"+
        "ROE: "+prediction.getRoe().toString()+adjectives[1]+"<br>"+
        "AltmanZScore: "+prediction.getAltmanZscore().toString()+adjectives[2]+"<br>"+
        "ModifiedCScore: "+prediction.getModifiedCscore().toString()+adjectives[3]+"<br>"+
        "PiotroskiFScore: "+prediction.getPiotroskiFscore().toString()+adjectives[4]+"<br>"+
        "ReturnFactor: "+prediction.getReturnFactor().toString()+"<br>"+
        "fAdjustedEarnings: "+prediction.getfAdjustedEarning().toString()+
        "</html>";


String summary = prediction.getArrayParameters()[0];

//summarizing the stock
if(score>1){
    summary= summary+ "is an excellent investment with a score of " + Math.round(score*100);
}else if(score>.8){
    summary= summary+ "is a good investment with a score of " + Math.round(score*100);
}else if(score>.6){
    summary= summary+ "is an okay, but not safe investment with a score of " + Math.round(score*100);
}else{
    summary= summary+ "is a bad investment with a score of " + Math.round(score*100);
}
```

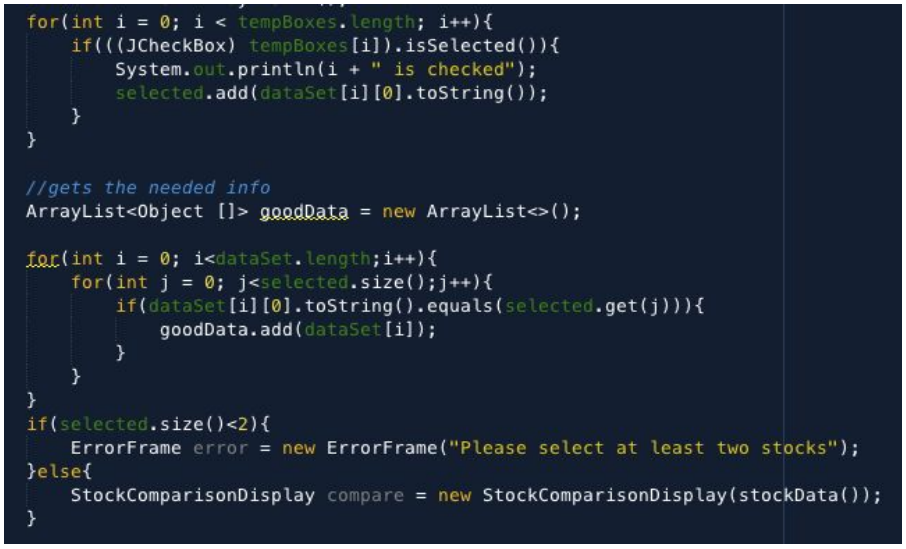
This code utilizes a series of if/else if/else statements to evaluate the individual properties of a given company. It then manipulates the value of “score” which is used to give a holistic view of how the stock is performing. Neither return factor, not F adjusted Earnings were included because while the client felt that they were important, the client noted that these values were situational and outside of context could not be evaluated.

Figure 2.1: Creating all the elements



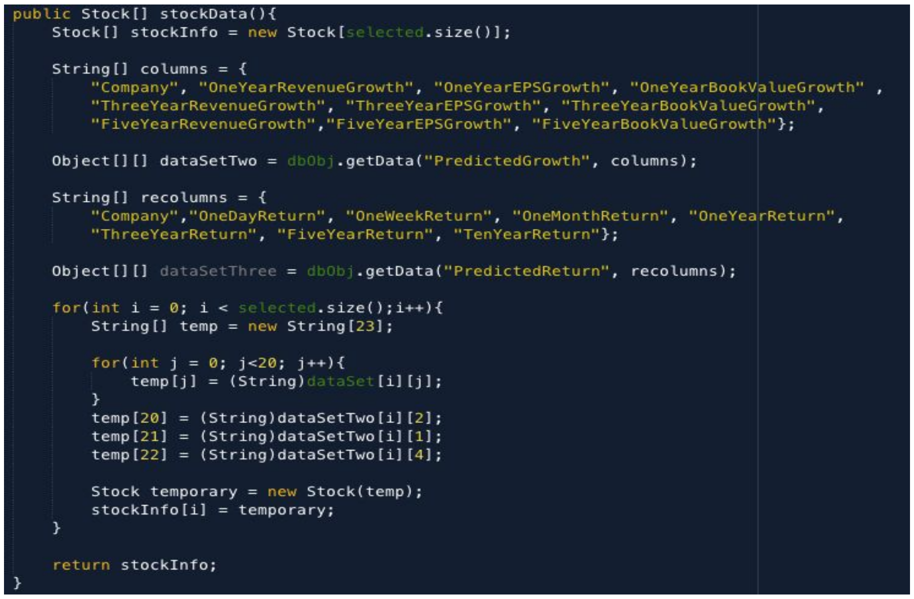
```java
//creates all of the items
for(int i = 0; i< dataSet.length; i++){
    eachItem = new JPanel();
    eachItem.setLayout(new BorderLayout());

    tempBoxes[i] = new JCheckBox();
    tempNames[i] =new JLabel(dataSet[i][0].toString());

    eachItem.add(tempBoxes[i], BorderLayout.CENTER);
    eachItem.add(tempNames[i], BorderLayout.EAST);
    eachItem.setBackground(Color.GREEN);

    allItems.add(eachItem);
}
```

Figure 2.2: Checking selected fields



```java
for(int i = 0; i < tempBoxes.length; i++){
    if(((JCheckBox) tempBoxes[i]).isSelected()){
        System.out.println(i + " is checked");
        selected.add(dataSet[i][0].toString());
    }
}

//gets the needed info
ArrayList<Object []> goodData = new ArrayList<>();

for(int i = 0; i<dataSet.length;i++){
    for(int j = 0; j<selected.size();j++){
        if(dataSet[i][0].toString().equals(selected.get(j))){
            goodData.add(dataSet[i]);
        }
    }
}
if(selected.size()<2){
    ErrorFrame error = new ErrorFrame("Please select at least two stocks");
}else{
    StockComparisonDisplay compare = new StockComparisonDisplay(stockData());
}
```

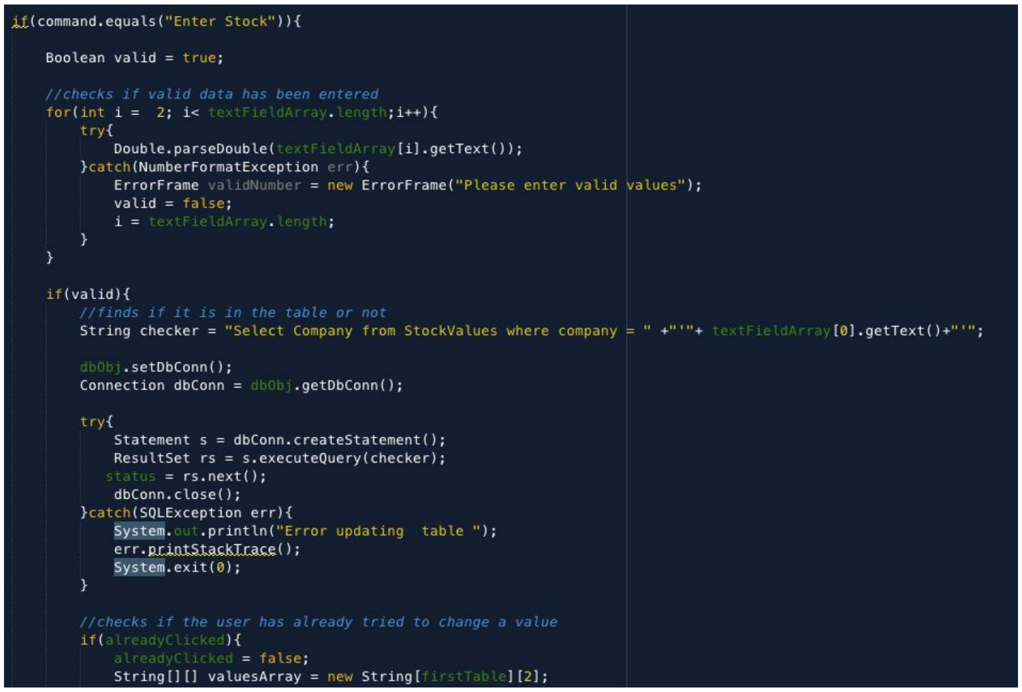
Figure 2.3: Extracting selected information



```java
public Stock[] stockData(){
    Stock[] stockInfo = new Stock[selected.size()];

    String[] columns = {
        "Company", "OneYearRevenueGrowth", "OneYearEPSGrowth", "OneYearBookValueGrowth" ,
        "ThreeYearRevenueGrowth", "ThreeYearEPSGrowth", "ThreeYearBookValueGrowth",
        "FiveYearRevenueGrowth","FiveYearEPSGrowth", "FiveYearBookValueGrowth"};

    Object[][] dataSetTwo = dbObj.getData("PredictedGrowth", columns);

    String[] recolumns = {
        "Company","OneDayReturn", "OneWeekReturn", "OneMonthReturn", "OneYearReturn",
        "ThreeYearReturn", "FiveYearReturn", "TenYearReturn"};

    Object[][] dataSetThree = dbObj.getData("PredictedReturn", recolumns);

    for(int i = 0; i < selected.size();i++){
        String[] temp = new String[23];

        for(int j = 0; j<20; j++){
            temp[j] = (String)dataSet[i][j];
        }
        temp[20] = (String)dataSetTwo[i][2];
        temp[21] = (String)dataSetTwo[i][1];
        temp[22] = (String)dataSetTwo[i][4];

        Stock temporary = new Stock(temp);
        stockInfo[i] = temporary;
    }

    return stockInfo;
}
```

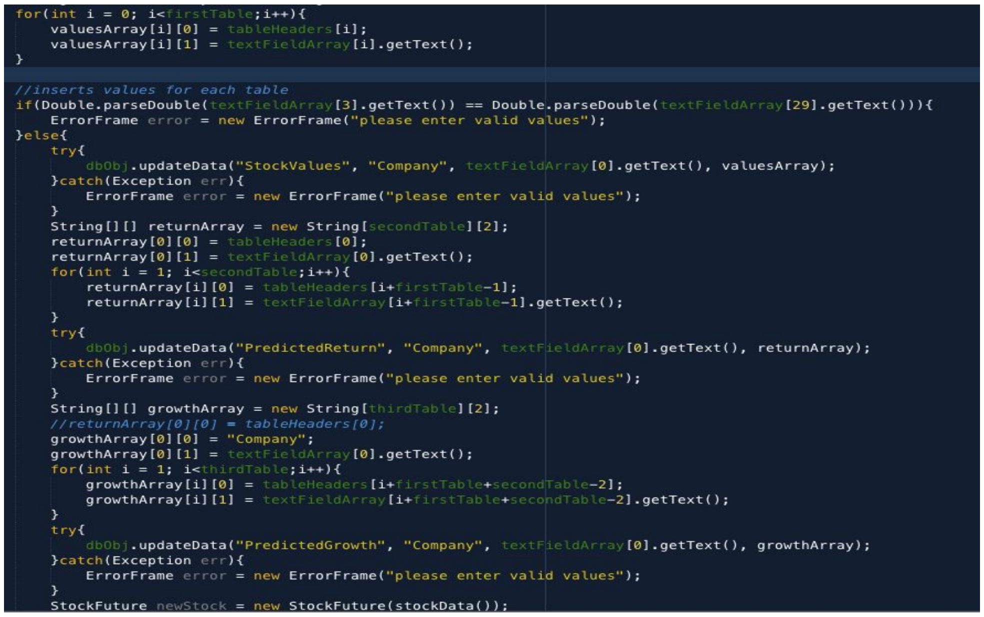
A similar algorithm is used elsewhere in the program, specifically in the frames of “DeleteStock” and “EnterStock”, however, this extract is taken from “CompareStockSelection”. This code uses a for loop to initialize each relevant item in the array for example data types like JCheckBoxes, JLabels, and JTextFields. Then it uses 2 more for loops, one that finds the selected elements to add them to an ArrayList. Then the second for loop loops through each selected element and gets information using the stockData() method shown bellow.

Figure 3.1: Reads/enters data, found in “EnterStock”, checks if stock exists

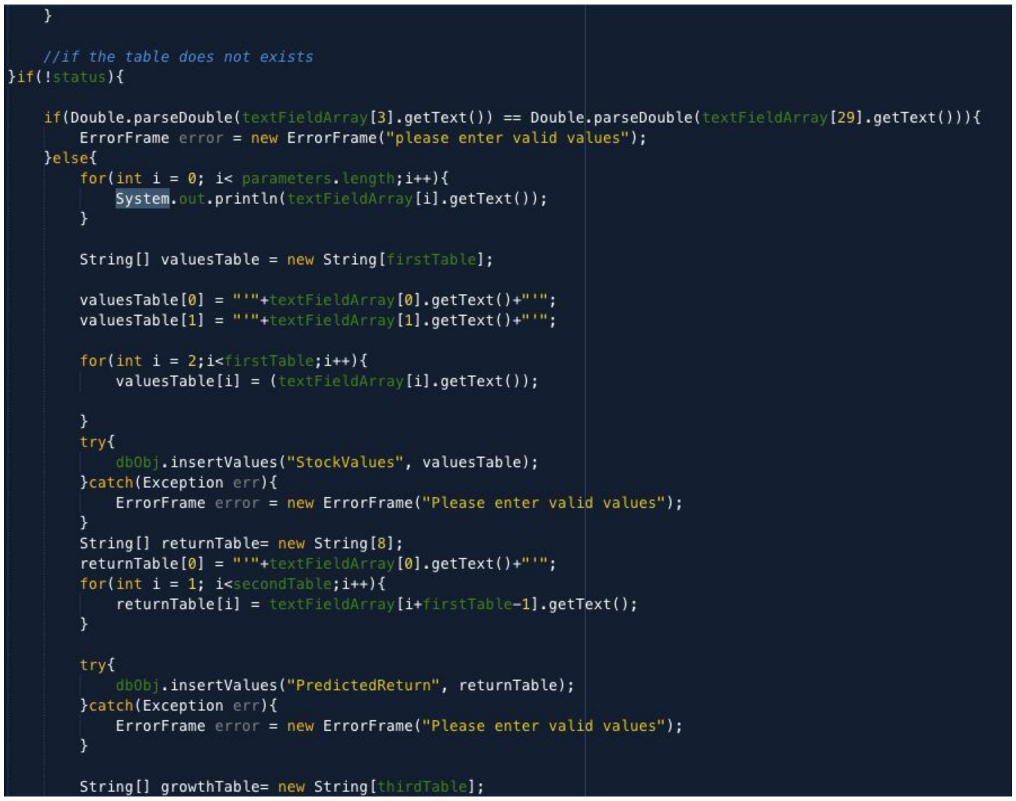


```java
if(command.equals("Enter Stock")){

    Boolean valid = true;

    //checks if valid data has been entered
    for(int i =  2; i< textFieldArray.length;i++){
        try{
            Double.parseDouble(textFieldArray[i].getText());
        }catch(NumberFormatException err){
            ErrorFrame validNumber = new ErrorFrame("Please enter valid values");
            valid = false;
            i = textFieldArray.length;
        }
    }

    if(valid){
        //finds if it is in the table or not
        String checker = "Select Company from StockValues where company = " +"'"+ textFieldArray[0].getText()+"'";

        dbObj.setDbConn();
        Connection dbConn = dbObj.getDbConn();

        try{
            Statement s = dbConn.createStatement();
            ResultSet rs = s.executeQuery(checker);
           status = rs.next();
            dbConn.close();
        }catch(SQLException err){
            System.out.println("Error updating  table ");
            err.printStackTrace();
            System.exit(0);
        }

        //checks if the user has already tried to change a value
        if(alreadyClicked){
            alreadyClicked = false;
            String[][] valuesArray = new String[firstTable][2];
```

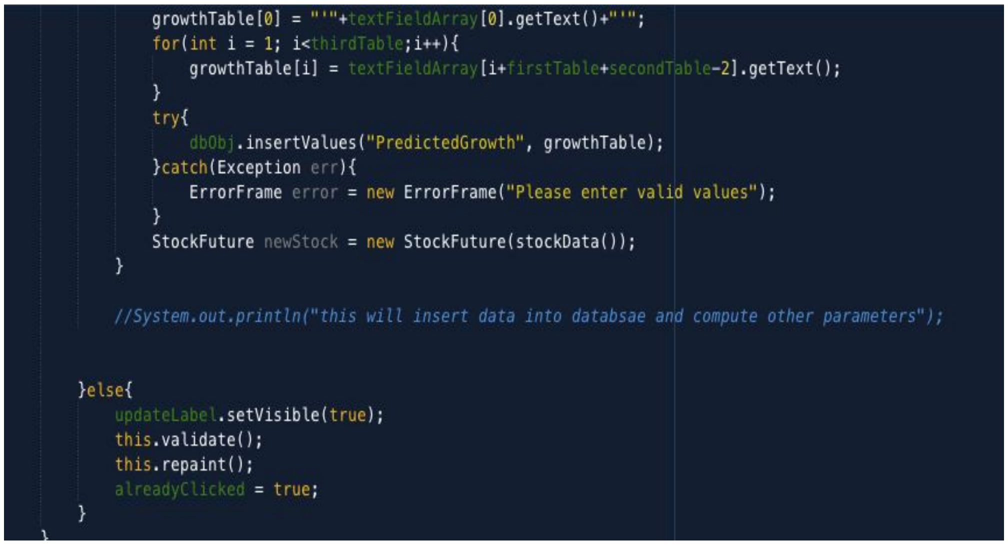
Figure 3.2: Entering data in table



```java
for(int i = 0; i<firstTable;i++){
    valuesArray[i][0] = tableHeaders[i];
    valuesArray[i][1] = textFieldArray[i].getText();
}

//inserts values for each table
if(Double.parseDouble(textFieldArray[3].getText()) == Double.parseDouble(textFieldArray[29].getText())){
    ErrorFrame error = new ErrorFrame("please enter valid values");
}else{
    try{
        dbObj.updateData("StockValues", "Company", textFieldArray[0].getText(), valuesArray);
    }catch(Exception err){
        ErrorFrame error = new ErrorFrame("please enter valid values");
    }
    String[][] returnArray = new String[secondTable][2];
    returnArray[0][0] = tableHeaders[0];
    returnArray[0][1] = textFieldArray[0].getText();
    for(int i = 1; i<secondTable;i++){
        returnArray[i][0] = tableHeaders[i+firstTable-1];
        returnArray[i][1] = textFieldArray[i+firstTable-1].getText();
    }
    try{
        dbObj.updateData("PredictedReturn", "Company", textFieldArray[0].getText(), returnArray);
    }catch(Exception err){
        ErrorFrame error = new ErrorFrame("please enter valid values");
    }
    String[][] growthArray = new String[thirdTable][2];
    //returnArray[0][0] = tableHeaders[0];
    growthArray[0][0] = "Company";
    growthArray[0][1] = textFieldArray[0].getText();
    for(int i = 1; i<thirdTable;i++){
        growthArray[i][0] = tableHeaders[i+firstTable+secondTable-2];
        growthArray[i][1] = textFieldArray[i+firstTable+secondTable-2].getText();
    }
    try{
        dbObj.updateData("PredictedGrowth", "Company", textFieldArray[0].getText(), growthArray);
    }catch(Exception err){
        ErrorFrame error = new ErrorFrame("please enter valid values");
    }
    StockFuture newStock = new StockFuture(stockData());
```



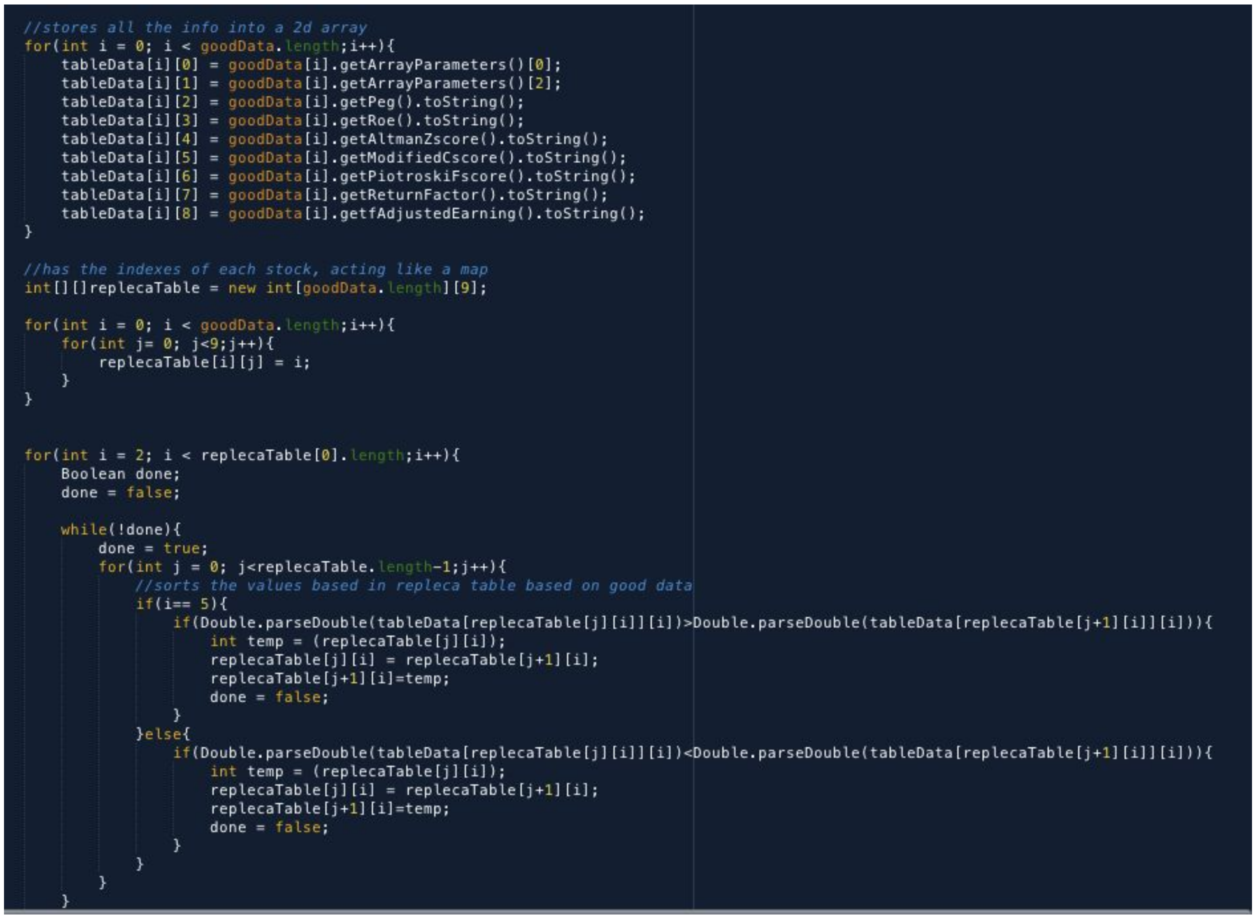
```
    }

    //if the table does not exists
}if(!status){

    if(Double.parseDouble(textFieldArray[3].getText()) == Double.parseDouble(textFieldArray[29].getText())){
        ErrorFrame error = new ErrorFrame("please enter valid values");
    }else{
        for(int i = 0; i< parameters.length;i++){
            System.out.println(textFieldArray[i].getText());
        }

        String[] valuesTable = new String[firstTable];

        valuesTable[0] = "'"+textFieldArray[0].getText()+"'";
        valuesTable[1] = "'"+textFieldArray[1].getText()+"'";

        for(int i = 2;i<firstTable;i++){
            valuesTable[i] = (textFieldArray[i].getText());

        }
        try{
            dbObj.insertValues("StockValues", valuesTable);
        }catch(Exception err){
            ErrorFrame error = new ErrorFrame("Please enter valid values");
        }
        String[] returnTable= new String[8];
        returnTable[0] = "'"+textFieldArray[0].getText()+"'";
        for(int i = 1; i<secondTable;i++){
            returnTable[i] = textFieldArray[i+firstTable-1].getText();
        }

        try{
            dbObj.insertValues("PredictedReturn", returnTable);
        }catch(Exception err){
            ErrorFrame error = new ErrorFrame("Please enter valid values");
        }

        String[] growthTable= new String[thirdTable];
```



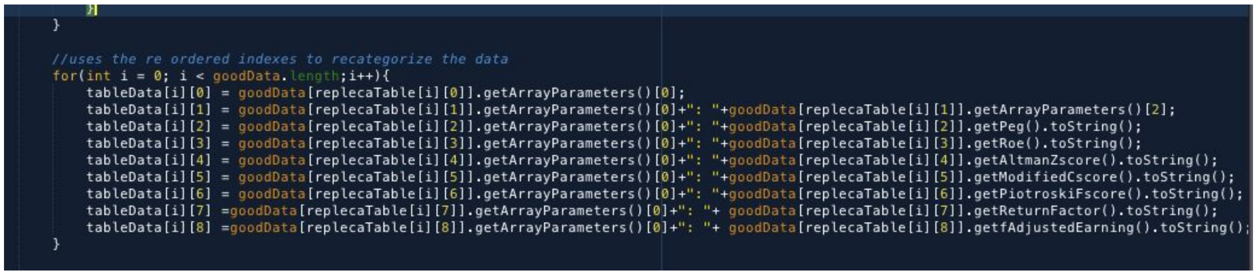
```
            growthTable[0] = "'"+textFieldArray[0].getText()+"'";
            for(int i = 1; i<thirdTable;i++){
                growthTable[i] = textFieldArray[i+firstTable+secondTable-2].getText();
            }
            try{
                dbObj.insertValues("PredictedGrowth", growthTable);
            }catch(Exception err){
                ErrorFrame error = new ErrorFrame("Please enter valid values");
            }
            StockFuture newStock = new StockFuture(stockData());
        }

        //System.out.println("this will insert data into databsae and compute other parameters");


    }else{
        updateLabel.setVisible(true);
        this.validate();
        this.repaint();
        alreadyClicked = true;
    }
}
```

Before reading then inserting the data it must first determine whether the user is updating or inserting data. So the program first checks if a record with the company name exists by selecting all instances of that name in the database and trying to go to the first instance. If it exists the user is prompted to confirm to update the data, otherwise the data is inserted using a similar algorithm to the previous one shown. It gets data from each field and inserts them into the 3 respective tables, then it creates a stock with the information and passes it to a StockFuture object, newStock.

Figure 4: Code that orders selected stocks for each parameter



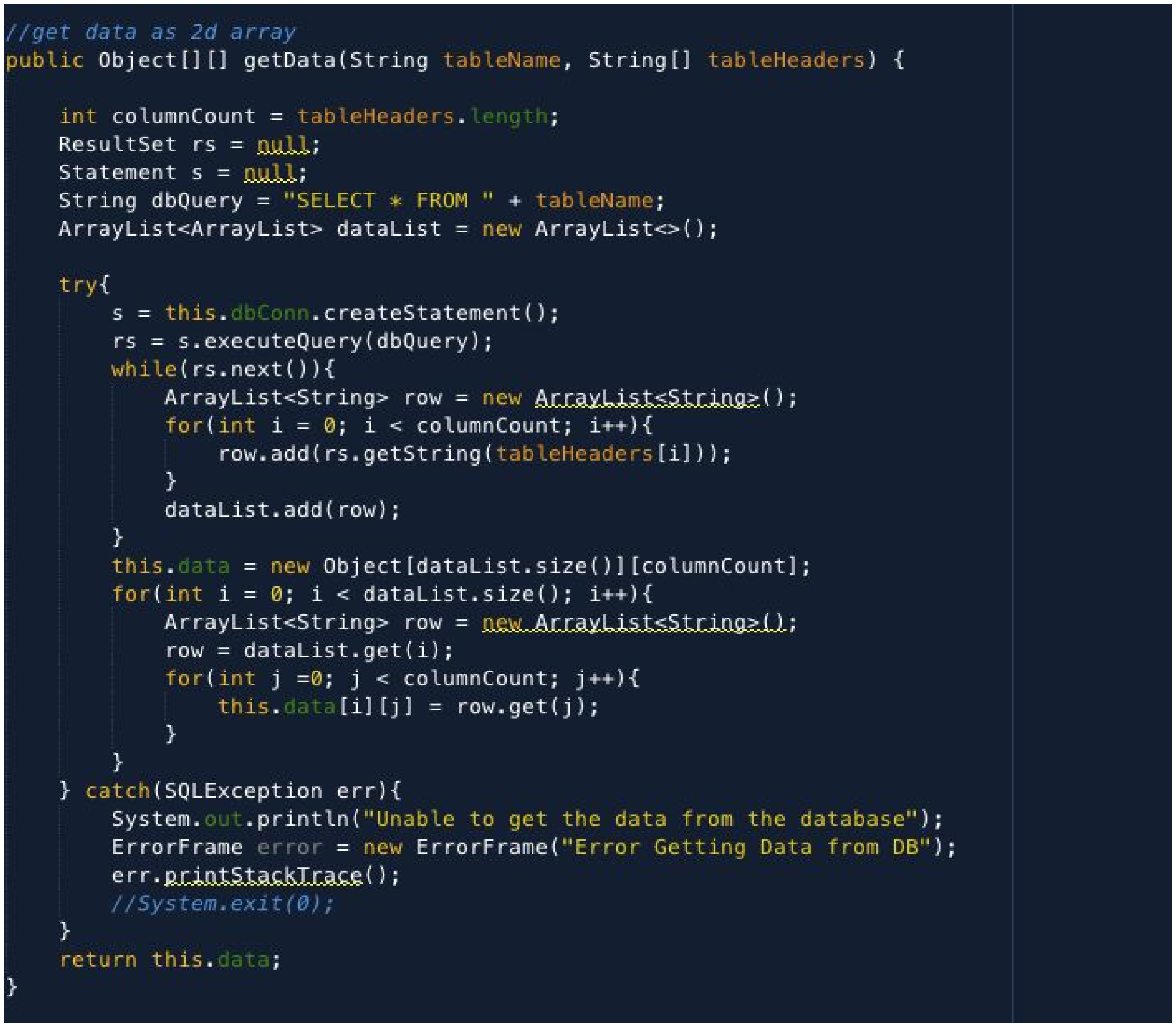
```java
//stores all the info into a 2d array
for(int i = 0; i < goodData.length;i++){
    tableData[i][0] = goodData[i].getArrayParameters()[0];
    tableData[i][1] = goodData[i].getArrayParameters()[2];
    tableData[i][2] = goodData[i].getPeg().toString();
    tableData[i][3] = goodData[i].getRoe().toString();
    tableData[i][4] = goodData[i].getAltmanZscore().toString();
    tableData[i][5] = goodData[i].getModifiedCscore().toString();
    tableData[i][6] = goodData[i].getPiotroskiFscore().toString();
    tableData[i][7] = goodData[i].getReturnFactor().toString();
    tableData[i][8] = goodData[i].getfAdjustedEarning().toString();
}

//has the indexes of each stock, acting like a map
int[][]replecaTable = new int[goodData.length][9];

for(int i = 0; i < goodData.length;i++){
    for(int j= 0; j<9;j++){
        replecaTable[i][j] = i;
    }
}


for(int i = 2; i < replecaTable[0].length;i++){
    Boolean done;
    done = false;

    while(!done){
        done = true;
        for(int j = 0; j<replecaTable.length-1;j++){
            //sorts the values based in repleca table based on good data
            if(i== 5){
                if(Double.parseDouble(tableData[replecaTable[j][i]][i])>Double.parseDouble(tableData[replecaTable[j+1][i]][i])){
                    int temp = (replecaTable[j][i]);
                    replecaTable[j][i] = replecaTable[j+1][i];
                    replecaTable[j+1][i]=temp;
                    done = false;
                }
            }else{
                if(Double.parseDouble(tableData[replecaTable[j][i]][i])<Double.parseDouble(tableData[replecaTable[j+1][i]][i])){
                    int temp = (replecaTable[j][i]);
                    replecaTable[j][i] = replecaTable[j+1][i];
                    replecaTable[j+1][i]=temp;
                    done = false;
                }
            }
        }
    }
```



```java
        }
    }

    //uses the re ordered indexes to recategorize the data
    for(int i = 0; i < goodData.length;i++){
        tableData[i][0] = goodData[replecaTable[i][0]].getArrayParameters()[0];
        tableData[i][1] = goodData[replecaTable[i][1]].getArrayParameters()[0]+": "+goodData[replecaTable[i][1]].getArrayParameters()[2];
        tableData[i][2] = goodData[replecaTable[i][2]].getArrayParameters()[0]+": "+goodData[replecaTable[i][2]].getPeg().toString();
        tableData[i][3] = goodData[replecaTable[i][3]].getArrayParameters()[0]+": "+goodData[replecaTable[i][3]].getRoe().toString();
        tableData[i][4] = goodData[replecaTable[i][4]].getArrayParameters()[0]+": "+goodData[replecaTable[i][4]].getAltmanZscore().toString();
        tableData[i][5] = goodData[replecaTable[i][5]].getArrayParameters()[0]+": "+goodData[replecaTable[i][5]].getModifiedCscore().toString();
        tableData[i][6] = goodData[replecaTable[i][6]].getArrayParameters()[0]+": "+goodData[replecaTable[i][6]].getPiotroskiFscore().toString();
        tableData[i][7] =goodData[replecaTable[i][7]].getArrayParameters()[0]+": "+ goodData[replecaTable[i][7]].getReturnFactor().toString();
        tableData[i][8] =goodData[replecaTable[i][8]].getArrayParameters()[0]+": "+ goodData[replecaTable[i][8]].getfAdjustedEarning().toString();
    }
```

This sorts the data by first taking the data that has been passed, and putting that in a 2d array. Then a “replecaTable” is created. This table holds the indices of each stock in each column. This is the table that ultimately gets sorted with reference to the values in goodData. After sorting the replecaTable, the data is then put in the visual table by, like a map, in each cell using the found index as a key in the goodData array. Then The information can be put into a table.

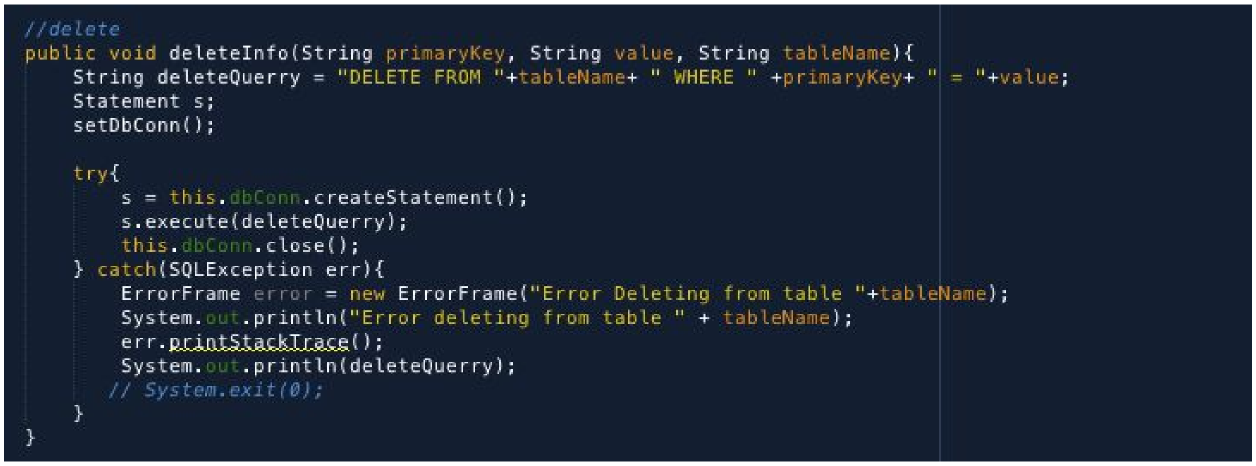
Figure 5: GetData method found in DataBaseAccess



```
//get data as 2d array
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(SQLException err){
        System.out.println("Unable to get the data from the database");
        ErrorFrame error = new ErrorFrame("Error Getting Data from DB");
        err.printStackTrace();
        //System.exit(0);
    }
    return this.data;
}
```

This code was made with reference to what my teacher taught during class. However, first, the data is obtained using a “SELECT * FROM + tableName” statement. The data is obtained as a ResultSet. Then I go through each row in the ResultSet and create an ArrayList that holds the data. That ArrayList is then added to an ArrayList of ArrayList of String. Then at the end, I convert that to a 2d object array using a nested for loop.

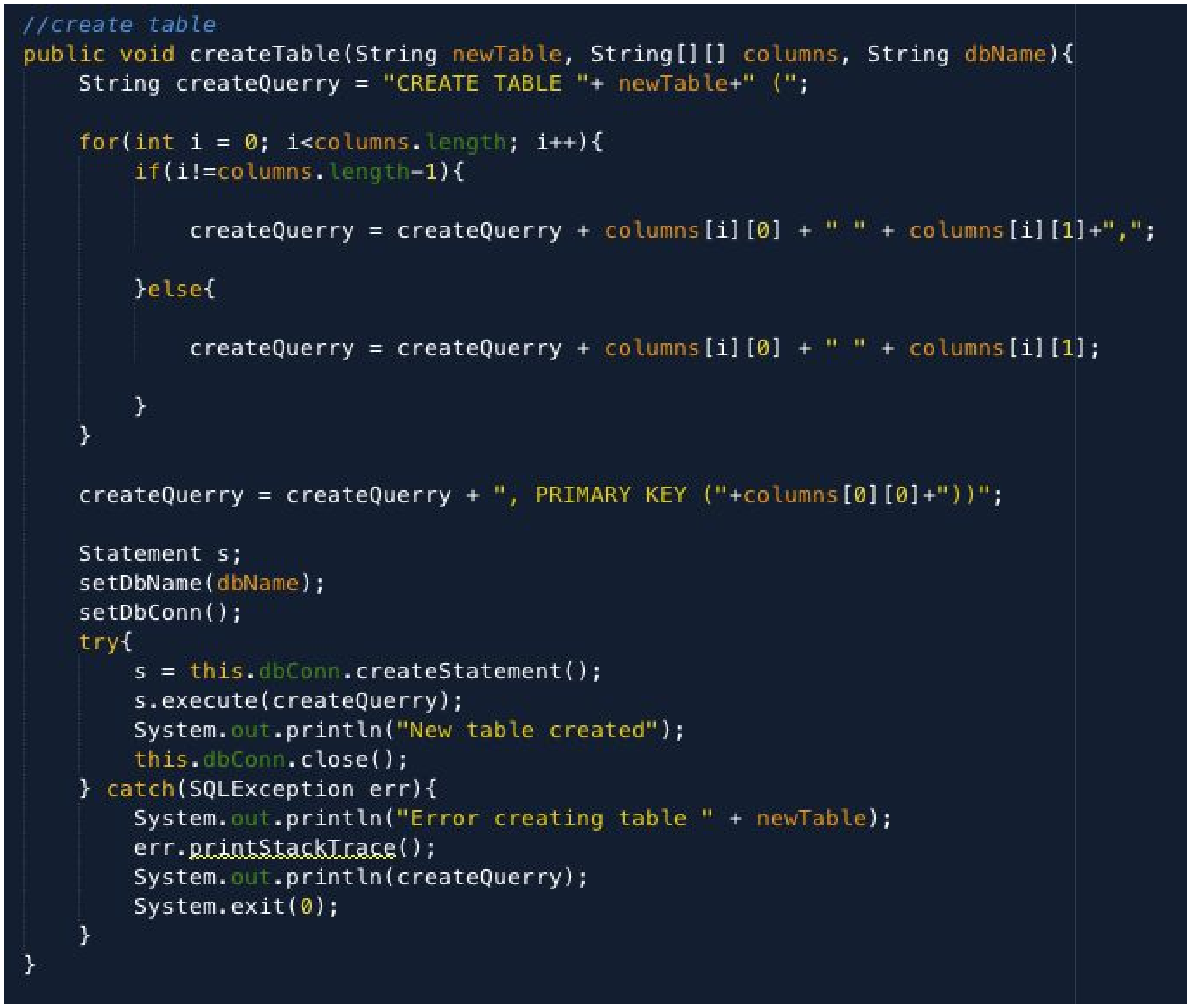
deleteInfo method found in DataBaseAccess



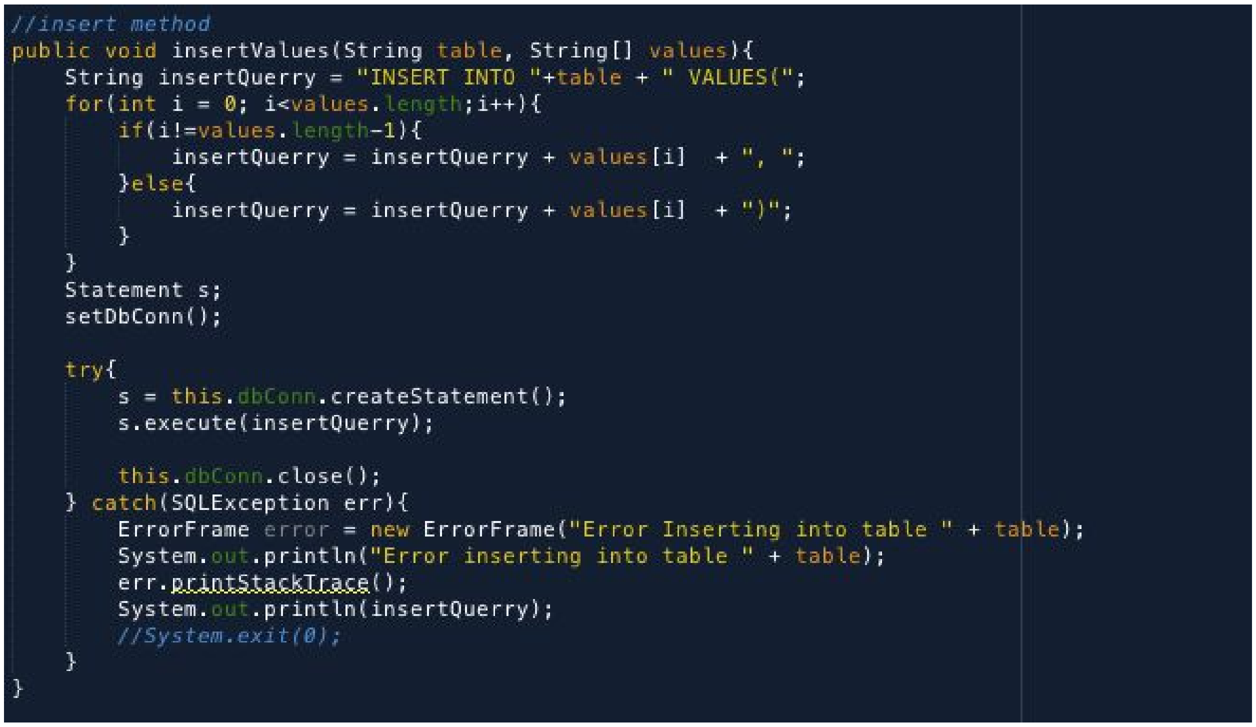
```
//delete
public void deleteInfo(String primaryKey, String value, String tableName){
    String deleteQuerry = "DELETE FROM "+tableName+ " WHERE " +primaryKey+ " = "+value;
    Statement s;
    setDbConn();

    try{
        s = this.dbConn.createStatement();
        s.execute(deleteQuerry);
        this.dbConn.close();
    } catch(SQLException err){
        ErrorFrame error = new ErrorFrame("Error Deleting from table "+tableName);
        System.out.println("Error deleting from table " + tableName);
        err.printStackTrace();
        System.out.println(deleteQuerry);
       // System.exit(0);
    }
}
```

This is the code used to delete information from the database. It takes the primary key, value, and table name as parameters, and constructs a statement using them to delete a distinct element from the table.

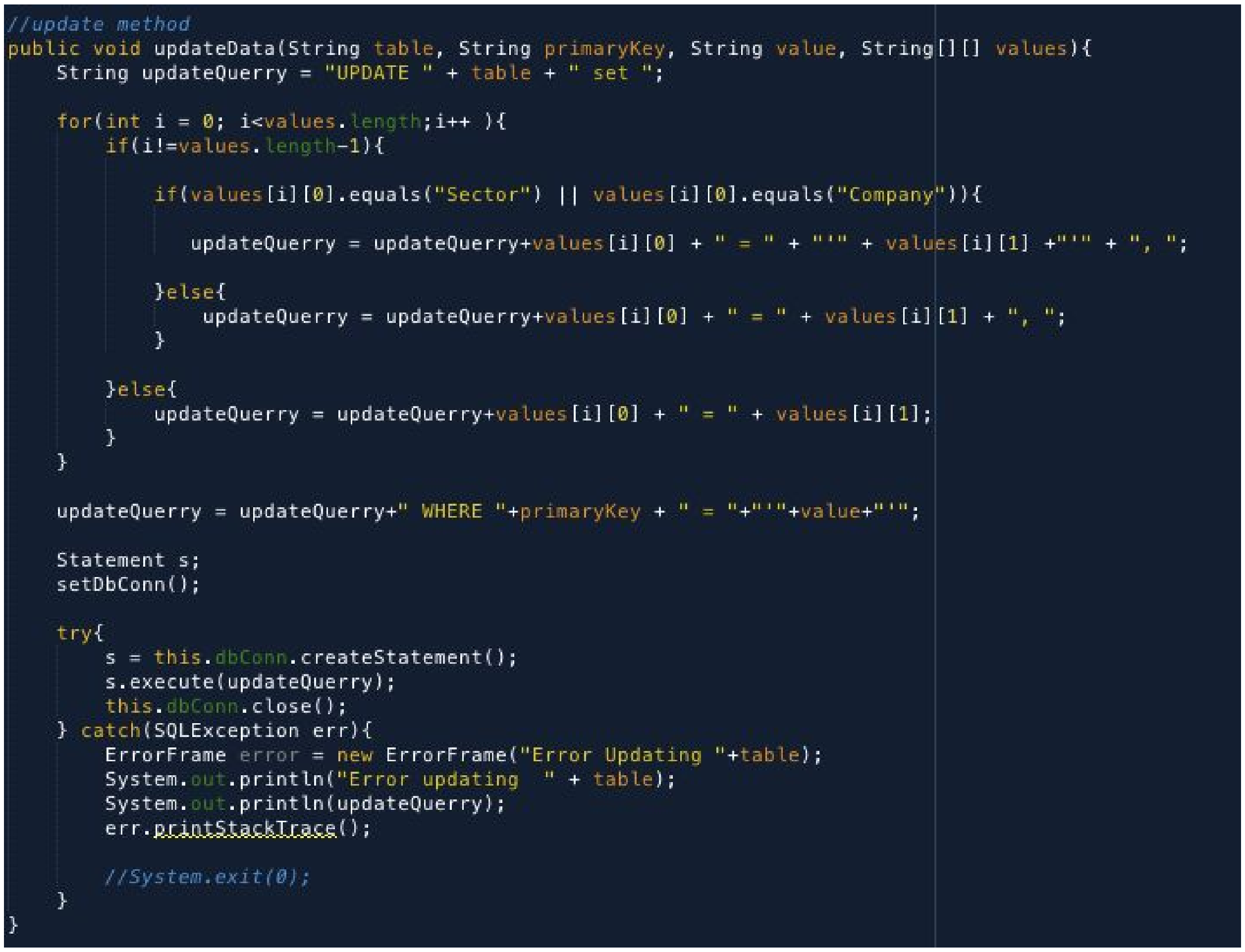
Figure 6: createTable, insertValues, and updateData found in DataBaseAccess



```
//create table
public void createTable(String newTable, String[][] columns, String dbName){
    String createQuerry = "CREATE TABLE "+ newTable+" (";

    for(int i = 0; i<columns.length; i++){
        if(i!=columns.length-1){

            createQuerry = createQuerry + columns[i][0] + " " + columns[i][1]+",";

        }else{

            createQuerry = createQuerry + columns[i][0] + " " + columns[i][1];

        }
    }

    createQuerry = createQuerry + ", PRIMARY KEY ("+columns[0][0]+"))";

    Statement s;
    setDbName(dbName);
    setDbConn();
    try{
        s = this.dbConn.createStatement();
        s.execute(createQuerry);
        System.out.println("New table created");
        this.dbConn.close();
    } catch(SQLException err){
        System.out.println("Error creating table " + newTable);
        err.printStackTrace();
        System.out.println(createQuerry);
        System.exit(0);
    }
}
```



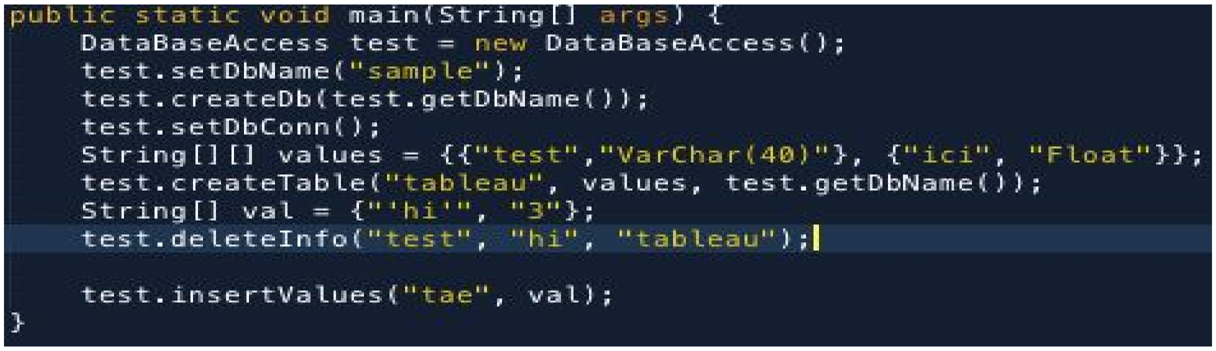
```java
//insert method
public void insertValues(String table, String[] values){
    String insertQuerry = "INSERT INTO "+table + " VALUES(";
    for(int i = 0; i<values.length;i++){
        if(i!=values.length-1){
            insertQuerry = insertQuerry + values[i]  + ", ";
        }else{
            insertQuerry = insertQuerry + values[i]  + ")";
        }
    }
    Statement s;
    setDbConn();

    try{
        s = this.dbConn.createStatement();
        s.execute(insertQuerry);

        this.dbConn.close();
    } catch(SQLException err){
        ErrorFrame error = new ErrorFrame("Error Inserting into table " + table);
        System.out.println("Error inserting into table " + table);
        err.printStackTrace();
        System.out.println(insertQuerry);
        //System.exit(0);
    }
}
```



```java
//update method
public void updateData(String table, String primaryKey, String value, String[][] values){
    String updateQuerry = "UPDATE " + table + " set ";

    for(int i = 0; i<values.length;i++ ){
        if(i!=values.length-1){

            if(values[i][0].equals("Sector") || values[i][0].equals("Company")){

               updateQuerry = updateQuerry+values[i][0] + " = " + "'" + values[i][1] +"'" + ", ";

            }else{
                updateQuerry = updateQuerry+values[i][0] + " = " + values[i][1] + ", ";
            }

        }else{
            updateQuerry = updateQuerry+values[i][0] + " = " + values[i][1];
        }
    }

    updateQuerry = updateQuerry+" WHERE "+primaryKey + " = "+"'"+value+"'";

    Statement s;
    setDbConn();

    try{
        s = this.dbConn.createStatement();
        s.execute(updateQuerry);
        this.dbConn.close();
    } catch(SQLException err){
        ErrorFrame error = new ErrorFrame("Error Updating "+table);
        System.out.println("Error updating  " + table);
        System.out.println(updateQuerry);
        err.printStackTrace();

        //System.exit(0);
    }
}
```

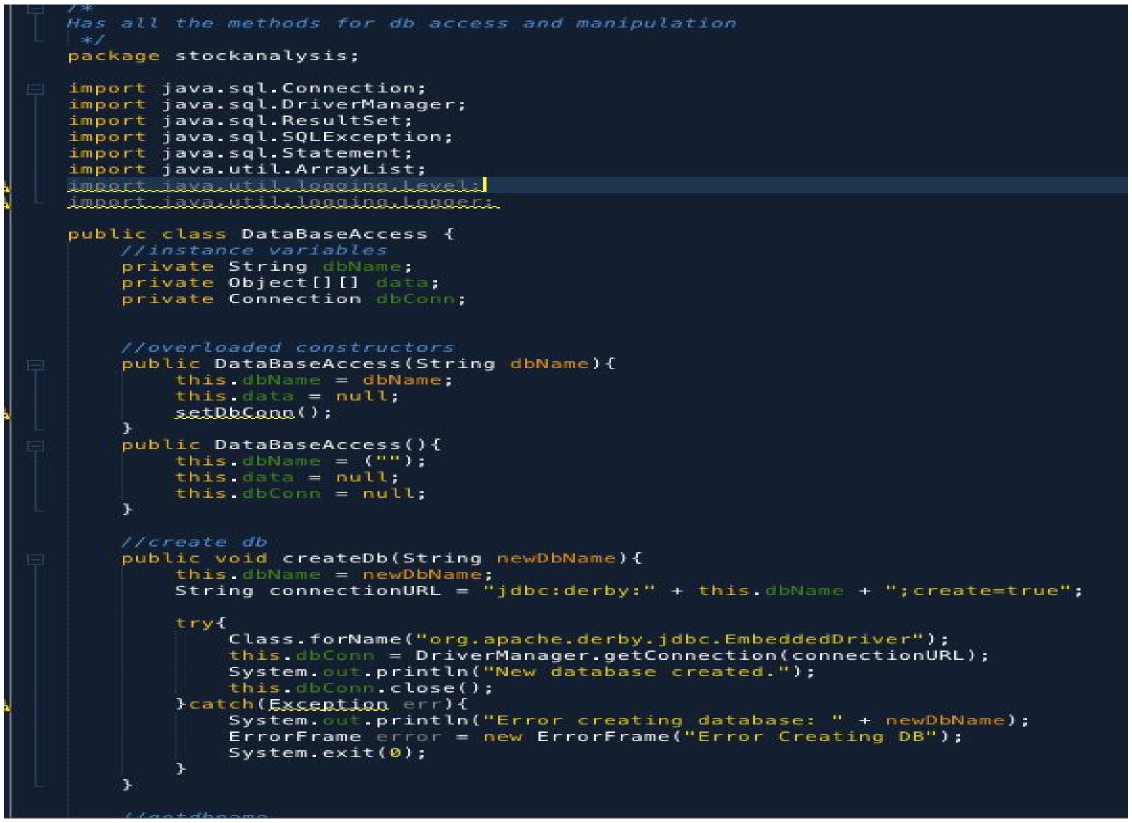
These methods have a similar core algorithm. I receive a 2d array and the table name, and for update, a primary key. The 2d array holds both the values and the data type. So I use a for loop to be able to construct a string with both the data type and value of each distinct item, and I use the primary key to identify the distinct element in update or, in createTable, I just use the first element in the 2d array.

Techniques Used

- Main methods for testing
    - 

```
public static void main(String[] args) {
    DataBaseAccess test = new DataBaseAccess();
    test.setDbName("sample");
    test.createDb(test.getDbName());
    test.setDbConn();
    String[][] values = {{"test","VarChar(40)"}, {"ici", "Float"}};
    test.createTable("tableau", values, test.getDbName());
    String[] val = {"'hi'", "3"};
    test.deleteInfo("test", "hi", "tableau");

    test.insertValues("tae", val);
}
```

    - Several of my classes required testing of the methods, like DataBaseAccess. For them, I included a main method, so that instead of running the program and entering values, I could just run the main method with the values I want, allowing me to conduct alpha testing on each part of my program, as I know the kind of inputs that my program struggles with, or might cause issues, and can test if these have been adressed properly.
- Data Base Access

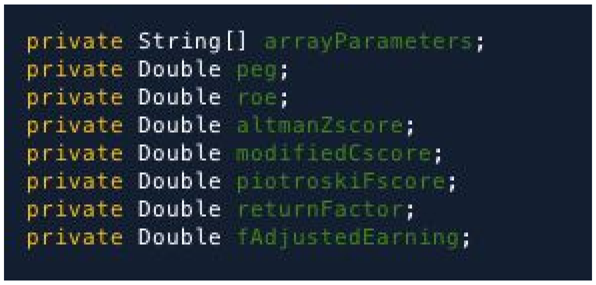


```
/*
Has all the methods for db access and manipulation
 */
package stockanalysis;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DataBaseAccess {
    //instance variables
    private String dbName;
    private Object[][] data;
    private Connection dbConn;


    //overloaded constructors
    public DataBaseAccess(String dbName){
        this.dbName = dbName;
        this.data = null;
        setDbConn();
    }
    public DataBaseAccess(){
        this.dbName = ("");
        this.data = null;
        this.dbConn = null;
    }

    //create db
    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){
            System.out.println("Error creating database: " + newDbName);
            ErrorFrame error = new ErrorFrame("Error Creating DB");
            System.exit(0);
        }
    }
```

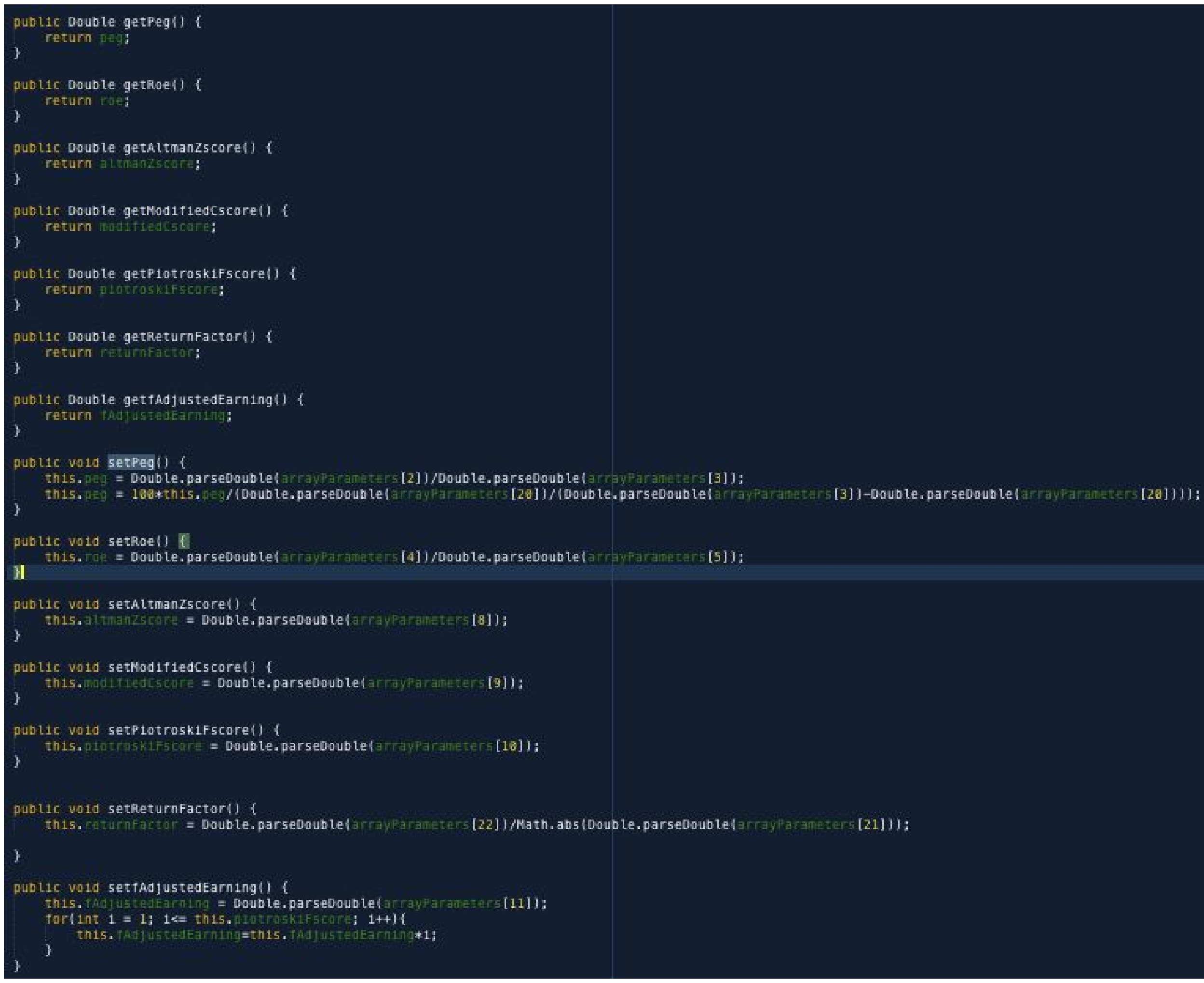
-

- I largely used a SQL connection and statements to interact with the database. This allowed me a nonvolatile source of storing the information for the program. Thus I could compare multiple stocks and recall information of each one.

- Encapsulation

- 

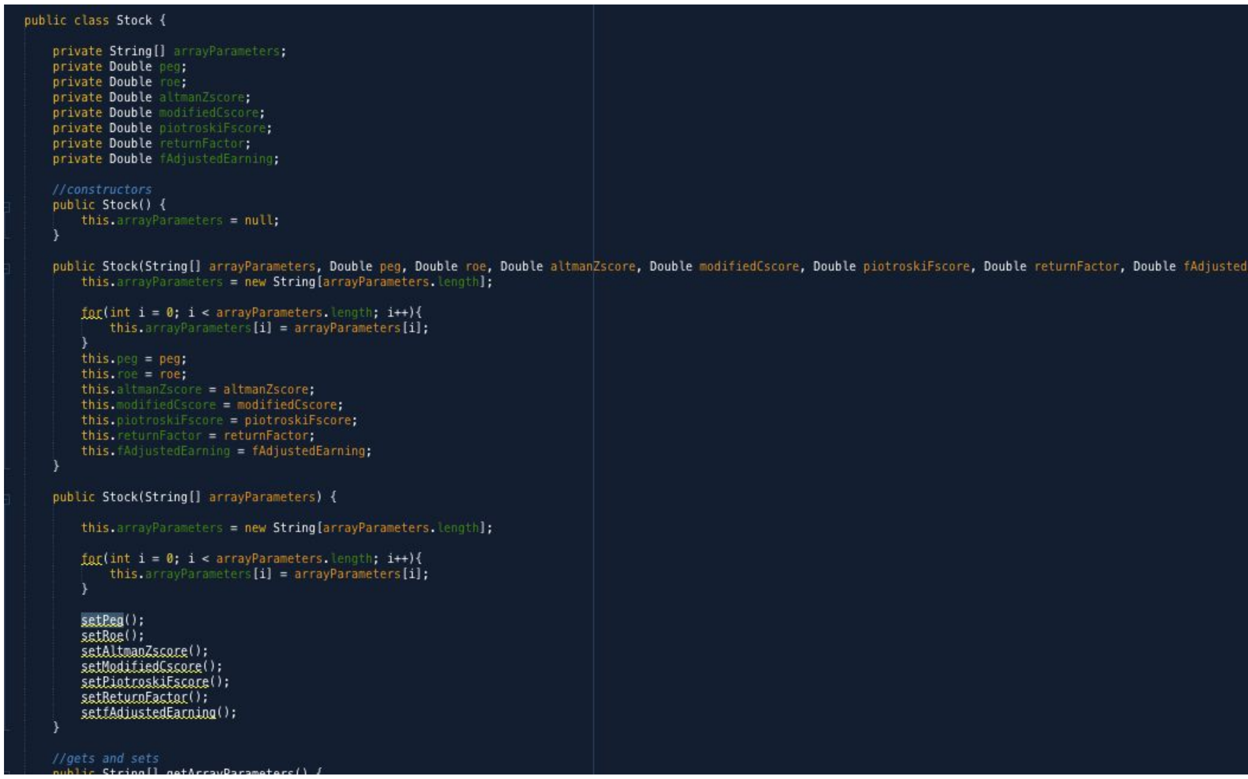
```java
private String[] arrayParameters;
private Double peg;
private Double roe;
private Double altmanZscore;
private Double modifiedCscore;
private Double piotroskiFscore;
private Double returnFactor;
private Double fAdjustedEarning;
```



```java
public Double getPeg() {
    return peg;
}

public Double getRoe() {
    return roe;
}

public Double getAltmanZscore() {
    return altmanZscore;
}

public Double getModifiedCscore() {
    return modifiedCscore;
}

public Double getPiotroskiFscore() {
    return piotroskiFscore;
}

public Double getReturnFactor() {
    return returnFactor;
}

public Double getfAdjustedEarning() {
    return fAdjustedEarning;
}

public void setPeg() {
    this.peg = Double.parseDouble(arrayParameters[2])/Double.parseDouble(arrayParameters[3]);
    this.peg = 100*this.peg/(Double.parseDouble(arrayParameters[20])/(Double.parseDouble(arrayParameters[3])-Double.parseDouble(arrayParameters[20])));
}

public void setRoe() {
    this.roe = Double.parseDouble(arrayParameters[4])/Double.parseDouble(arrayParameters[5]);
}

public void setAltmanZscore() {
    this.altmanZscore = Double.parseDouble(arrayParameters[8]);
}

public void setModifiedCscore() {
    this.modifiedCscore = Double.parseDouble(arrayParameters[9]);
}

public void setPiotroskiFscore() {
    this.piotroskiFscore = Double.parseDouble(arrayParameters[10]);
}


public void setReturnFactor() {
    this.returnFactor = Double.parseDouble(arrayParameters[22])/Math.abs(Double.parseDouble(arrayParameters[21]));

}

public void setfAdjustedEarning() {
    this.fAdjustedEarning = Double.parseDouble(arrayParameters[11]);
    for(int i = 1; i<= this.piotroskiFscore; i++){
        this.fAdjustedEarning=this.fAdjustedEarning*i;
    }
}
```

- Encapsulation, the practice of data hiding by using private variables and public get/set methods was useful for me, noticeably so in my stock class. Less so in data hiding, but instead for making sure that I could easily access the values from any class and easily test/debug the code.

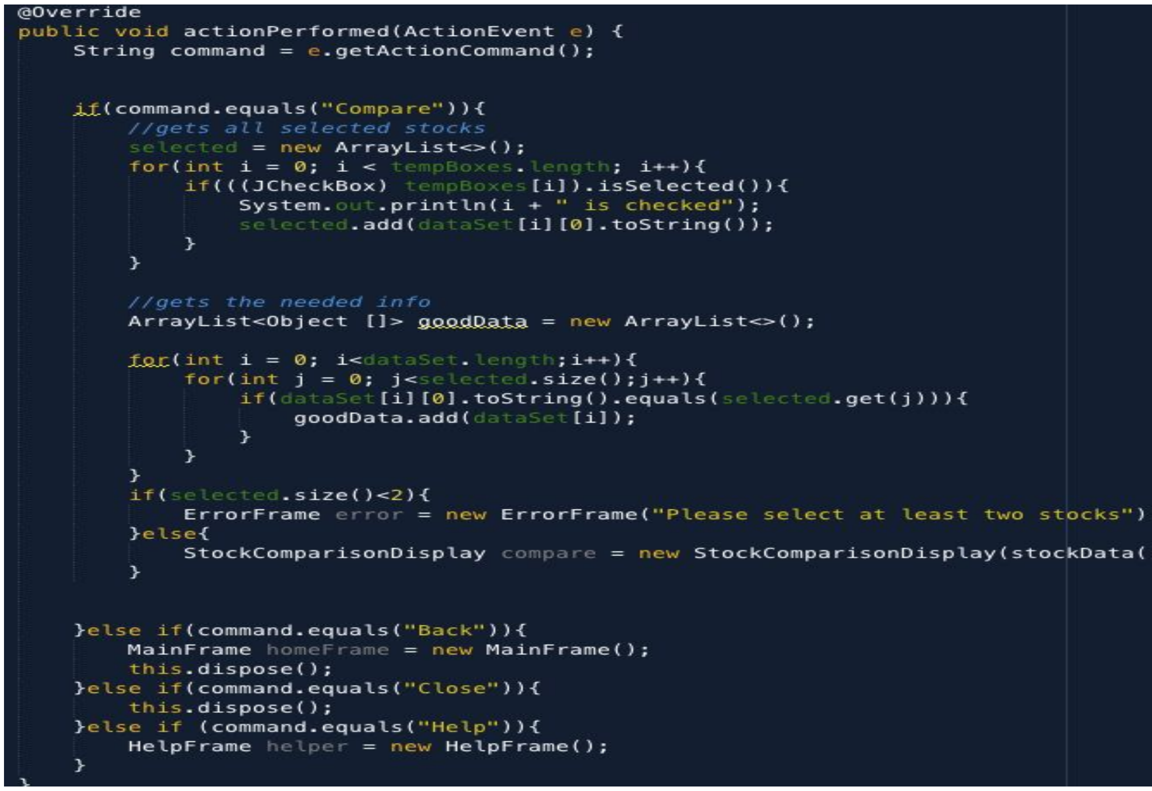
- Abstraction

- 

```java
public class Stock {

    private String[] arrayParameters;
    private Double peg;
    private Double roe;
    private Double altmanZscore;
    private Double modifiedCscore;
    private Double piotroskiFscore;
    private Double returnFactor;
    private Double fAdjustedEarning;

    //constructors
    public Stock() {
        this.arrayParameters = null;
    }

    public Stock(String[] arrayParameters, Double peg, Double roe, Double altmanZscore, Double modifiedCscore, Double piotroskiFscore, Double returnFactor, Double fAdjusted
        this.arrayParameters = new String[arrayParameters.length];

        for(int i = 0; i < arrayParameters.length; i++){
            this.arrayParameters[i] = arrayParameters[i];
        }
        this.peg = peg;
        this.roe = roe;
        this.altmanZscore = altmanZscore;
        this.modifiedCscore = modifiedCscore;
        this.piotroskiFscore = piotroskiFscore;
        this.returnFactor = returnFactor;
        this.fAdjustedEarning = fAdjustedEarning;
    }

    public Stock(String[] arrayParameters) {

        this.arrayParameters = new String[arrayParameters.length];

        for(int i = 0; i < arrayParameters.length; i++){
            this.arrayParameters[i] = arrayParameters[i];
        }

        setPeg();
        setRoe();
        setAltmanZscore();
        setModifiedCscore();
        setPiotroskiFscore();
        setReturnFactor();
        setfAdjustedEarning();
    }

    //gets and sets
    public String[] getArrayParameters() {
```

- The stock class was an abstraction of a stock. Thus I was able to use it to mimic the real world stock and apply characteristics in the real world to the abstraction.

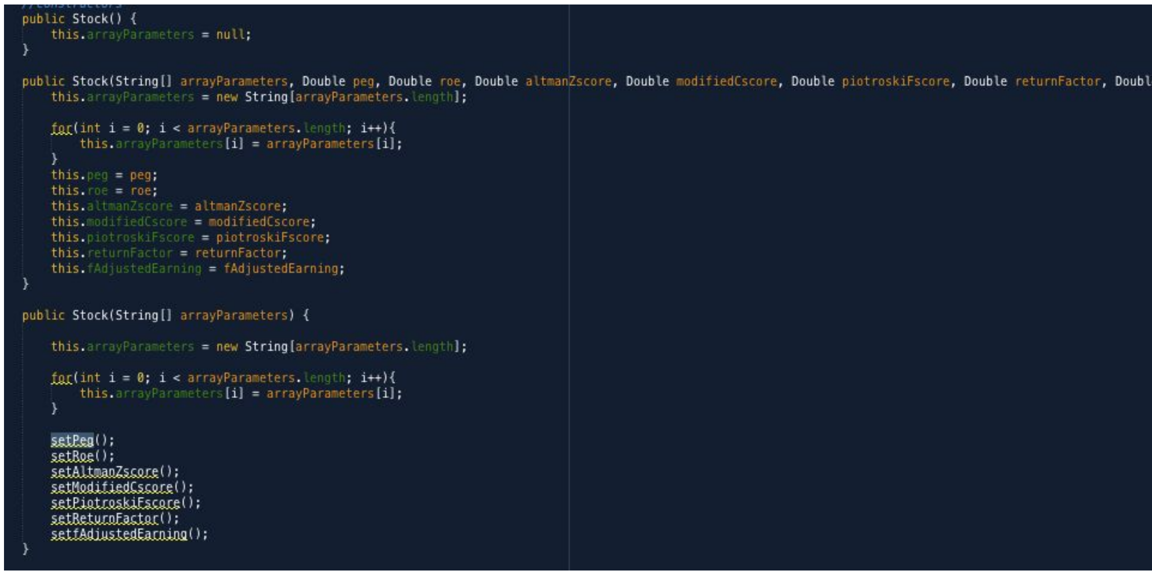
- Polymorphism
    - Overriding

- 

```java
@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();


    if(command.equals("Compare")){
        //gets all selected stocks
        selected = new ArrayList<>();
        for(int i = 0; i < tempBoxes.length; i++){
            if(((JCheckBox) tempBoxes[i]).isSelected()){
                System.out.println(i + " is checked");
                selected.add(dataSet[i][0].toString());
            }
        }

        //gets the needed info
        ArrayList<Object []> goodData = new ArrayList<>();

        for(int i = 0; i<dataSet.length;i++){
            for(int j = 0; j<selected.size();j++){
                if(dataSet[i][0].toString().equals(selected.get(j))){
                    goodData.add(dataSet[i]);
                }
            }
        }
        if(selected.size()<2){
            ErrorFrame error = new ErrorFrame("Please select at least two stocks")
        }else{
            StockComparisonDisplay compare = new StockComparisonDisplay(stockData(
        }


    }else if(command.equals("Back")){
        MainFrame homeFrame = new MainFrame();
        this.dispose();
    }else if(command.equals("Close")){
        this.dispose();
    }else if (command.equals("Help")){
        HelpFrame helper = new HelpFrame();
    }
}
```

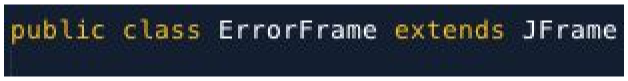
- Overriding was used in all of my actionPerformed methods. It allowed me to override the general actionPerformed method and instead implement my own definition to make my buttons do what I wanted them to do.

- Overloading

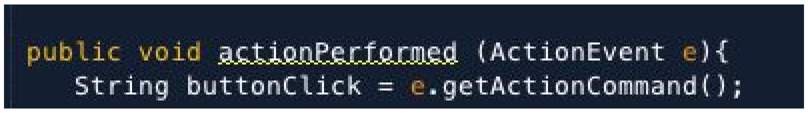
- 

```java
public Stock() {
    this.arrayParameters = null;
}

public Stock(String[] arrayParameters, Double peg, Double roe, Double altmanZscore, Double modifiedCscore, Double piotroskiFscore, Double returnFactor, Doubl
    this.arrayParameters = new String[arrayParameters.length];

    for(int i = 0; i < arrayParameters.length; i++){
        this.arrayParameters[i] = arrayParameters[i];
    }
    this.peg = peg;
    this.roe = roe;
    this.altmanZscore = altmanZscore;
    this.modifiedCscore = modifiedCscore;
    this.piotroskiFscore = piotroskiFscore;
    this.returnFactor = returnFactor;
    this.fAdjustedEarning = fAdjustedEarning;
}

public Stock(String[] arrayParameters) {

    this.arrayParameters = new String[arrayParameters.length];

    for(int i = 0; i < arrayParameters.length; i++){
        this.arrayParameters[i] = arrayParameters[i];
    }

    setPeg();
    setRoe();
    setAltmanZscore();
    setModifiedCscore();
    setPiotroskiFscore();
    setReturnFactor();
    setfAdjustedEarning();
}
```

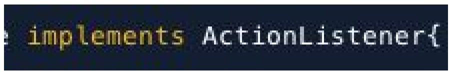
- By overloading the constructor of the stock class, I had the fallback of using different arguments. But also the ability to easily test my code with only the inputs that I want to use at a given time, not extraneous values.

- OOP
    - I used object oriented programming, not only in terms of calculations with my stock class and other calculations/methods. But also in the entire GUI, I used OOP with JFrame to construct my GUI.
- Relations
    - Inheritance
        - 

```java
public class ErrorFrame extends JFrame
```

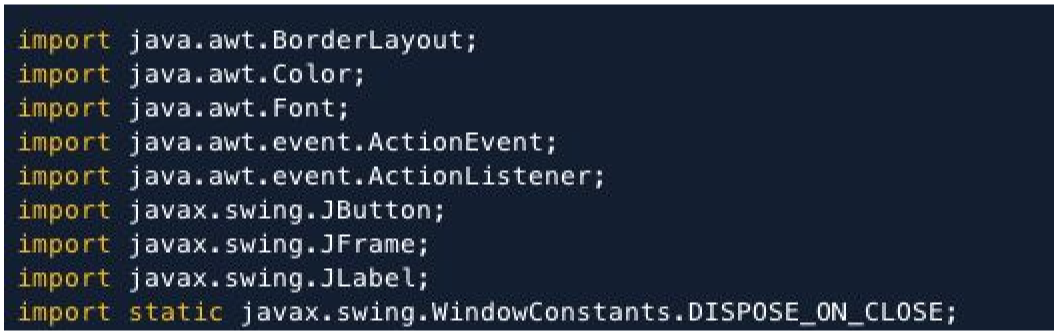
        - All my frames inherit JFrame to be able to use the behaviors/attributes in JFrame
    - Aggregation
        - 

```java
public void actionPerformed (ActionEvent e){
    String buttonClick = e.getActionCommand();
```



```java
implements ActionListener{
```

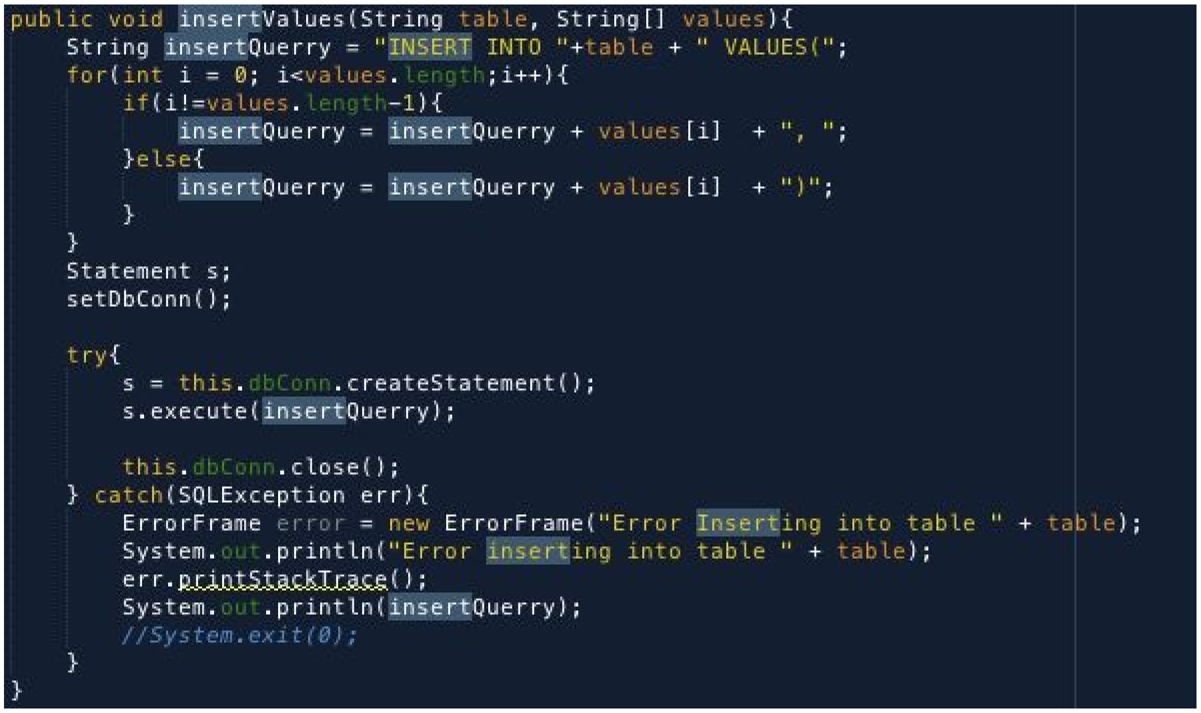
        - All my JFrames use aggregation with ActionListener to be able to override the ActionPerformed method with my own implementation
    - Dependency

- 

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import static javax.swing.WindowConstants.DISPOSE_ON_CLOSE;
```

- I used several dependencies, from the internal like Stock class to the external like ArrayList. They each had independent uses, but on the whole, enabled code reuse and reduced the need for testing or coding things beyond my abilities, like JFrames.

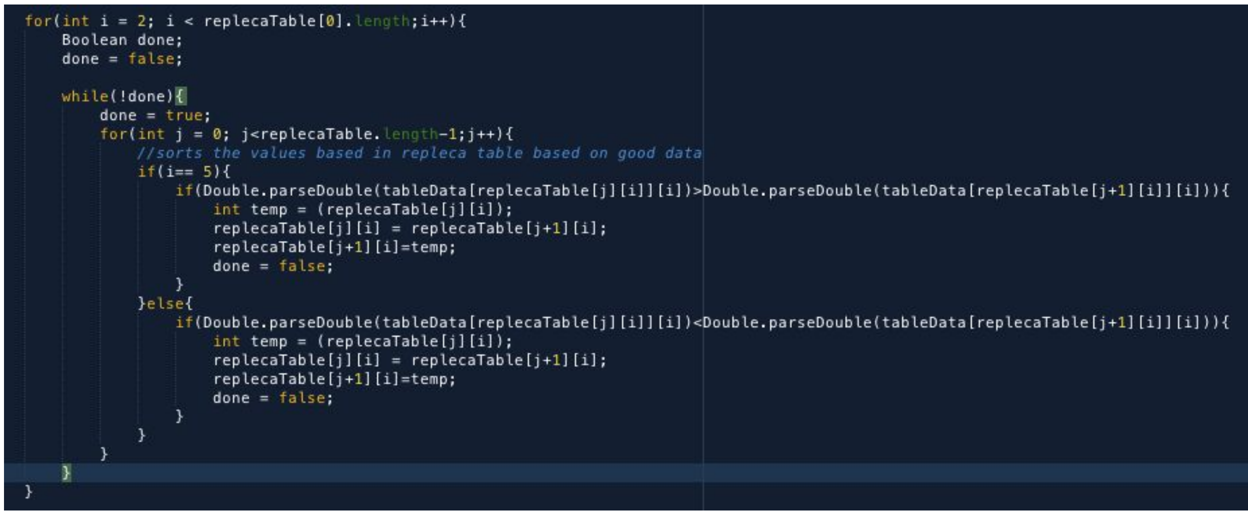
- Loops

  - 

```
public void insertValues(String table, String[] values){
    String insertQuerry = "INSERT INTO "+table + " VALUES(";
    for(int i = 0; i<values.length;i++){
        if(i!=values.length-1){
            insertQuerry = insertQuerry + values[i]  + ", ";
        }else{
            insertQuerry = insertQuerry + values[i]  + ")";
        }
    }
    Statement s;
    setDbConn();

    try{
        s = this.dbConn.createStatement();
        s.execute(insertQuerry);

        this.dbConn.close();
    } catch(SQLException err){
        ErrorFrame error = new ErrorFrame("Error Inserting into table " + table);
        System.out.println("Error inserting into table " + table);
        err.printStackTrace();
        System.out.println(insertQuerry);
        //System.exit(0);
    }
}
```

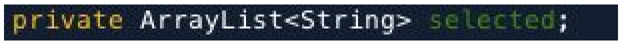
  - I used loops at several points for iterations. I used them in calculations, getting data, etc., essentially anywhere I needed to streamline a process with a loop for a number of times I didn't know yet.

- If statements

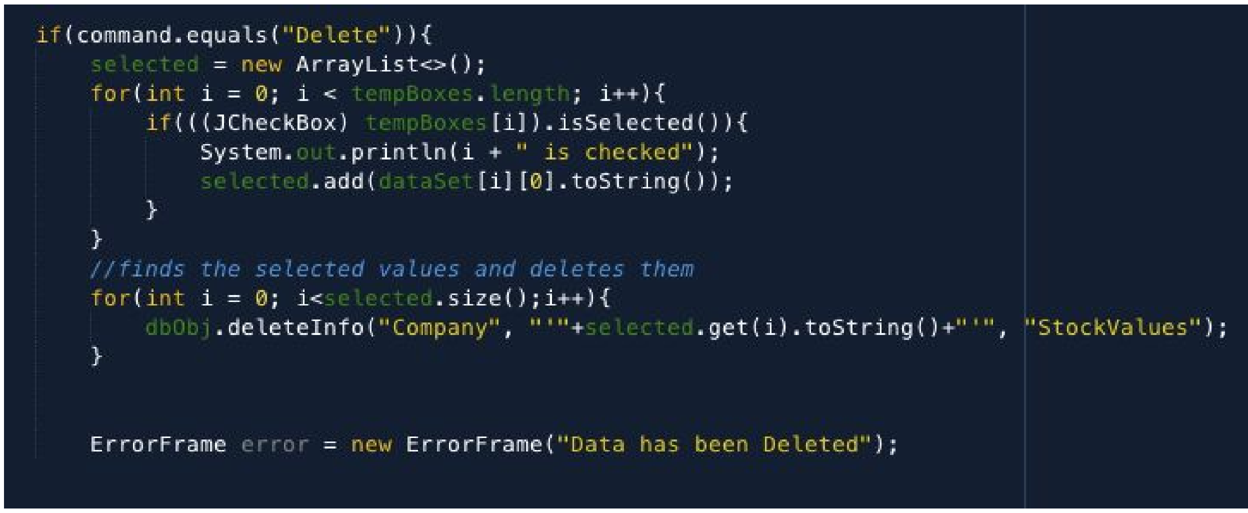
- 

```java
for(int i = 2; i < replecaTable[0].length;i++){
    Boolean done;
    done = false;

    while(!done){
        done = true;
        for(int j = 0; j<replecaTable.length-1;j++){
            //sorts the values based in repleca table based on good data
            if(i== 5){
                if(Double.parseDouble(tableData[replecaTable[j][i]][i])>Double.parseDouble(tableData[replecaTable[j+1][i]][i])){
                    int temp = (replecaTable[j][i]);
                    replecaTable[j][i] = replecaTable[j+1][i];
                    replecaTable[j+1][i]=temp;
                    done = false;
                }
            }else{
                if(Double.parseDouble(tableData[replecaTable[j][i]][i])<Double.parseDouble(tableData[replecaTable[j+1][i]][i])){
                    int temp = (replecaTable[j][i]);
                    replecaTable[j][i] = replecaTable[j+1][i];
                    replecaTable[j+1][i]=temp;
                    done = false;
                }
            }
        }
    }
}
```

- They allowed logic handling at several points, allowing me to determine what to do like on button presses, or evaluating things like in StockFuture.

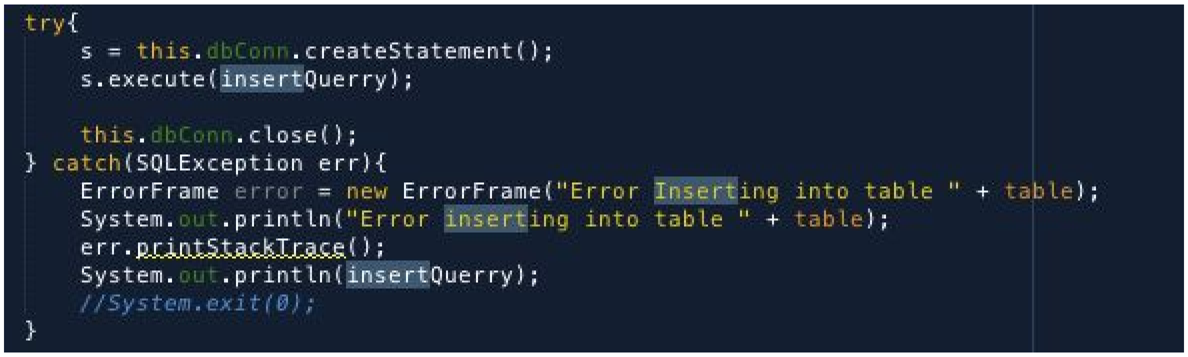
- ArrayLists
    - 

```java
private ArrayList<String> selected;
```



```java
if(command.equals("Delete")){
    selected = new ArrayList<>();
    for(int i = 0; i < tempBoxes.length; i++){
        if(((JCheckBox) tempBoxes[i]).isSelected()){
            System.out.println(i + " is checked");
            selected.add(dataSet[i][0].toString());
        }
    }
    //finds the selected values and deletes them
    for(int i = 0; i<selected.size();i++){
        dbObj.deleteInfo("Company", "'"+selected.get(i).toString()+"'", "StockValues");
    }


    ErrorFrame error = new ErrorFrame("Data has been Deleted");
```

    - I used ArrayLists to stare information if I didn’t know how many spots I needed. This was useful in cases like getData, or the DeleteStock ActionPerformed.
- Exception Handling

- 

```
try{
    s = this.dbConn.createStatement();
    s.execute(insertQuerry);

    this.dbConn.close();
} catch(SQLException err){
    ErrorFrame error = new ErrorFrame("Error Inserting into table " + table);
    System.out.println("Error inserting into table " + table);
    err.printStackTrace();
    System.out.println(insertQuerry);
    //System.exit(0);
}
```

    - This allowed robustness, like NullPointerExceptions and SQLConnectionErrors, which allowed me to ensure the meeting of one of my success criteria.
- Design process
    - While developing my application I had my client conduct user acceptance testing, so I could implement their feedback.

Tools Used

- Java API
    - I used the Java API and several classes in it such as JFrame and ArrayList to be able to construct my GUI, carry out certain complex calculations, and more.
- Java Derby
    - I used the Java Derby databases to be able to store non-volatile information.