Criterion C – Development

Word Count: 831

Contents

Program Structure 4

Packages 4

Object-Oriented Programming 5

Encapsulation 6

Aggregation 6

Method Overloading 7

Error Handling 8

Using Try/Catch 8

User Input Validation 9

Registration/Login 9

Company Addition 9

Use of Files 11

Authentication 11

User 12

Creating User File 12

Retrieving User's Companies 12

Company 13

Structure 13

Retrieving Company Data 14

Retrieving Statistic Data 16

Functionality 18

Company Merging (ArrayList of Objects, For Each Loop & Comparator) 18

Company Valuation (ArrayList of Objects, For Each Loop, For Loop) 21

Data Extrapolation 21

Final Calculation 24

Statistic Graphing (ArrayList of Objects, For Each Loop) 24

Linked List of Company Objects 26

Interface 28

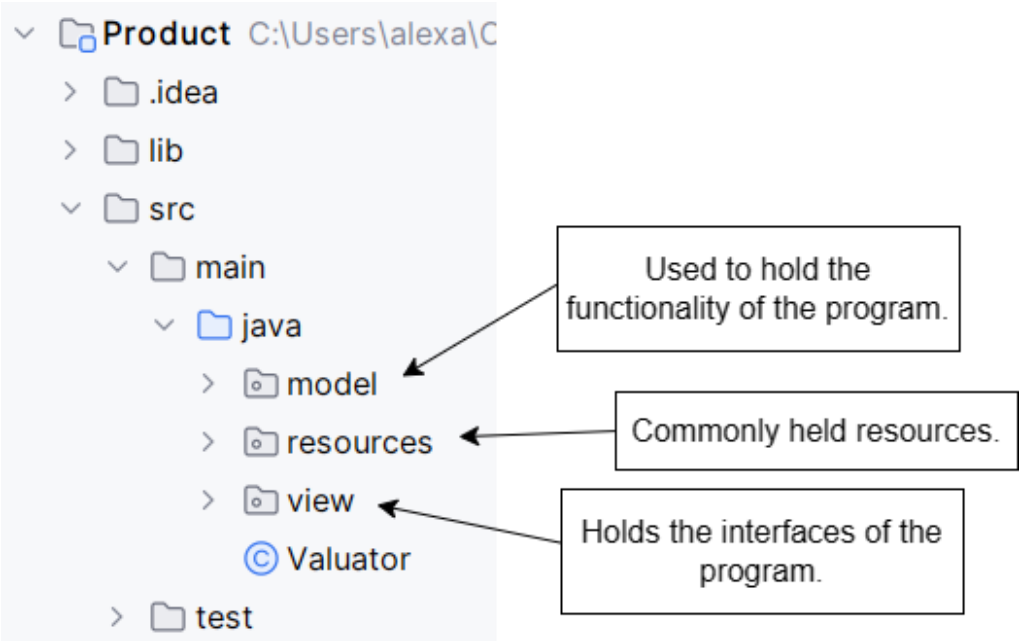
Libraries 31

References 35

Program Structure

Packages

The program is organized into packages for easier maintainability, the main packages being: “model”, “resources,” and “view.”



Screenshot 1 Program structure.

Object-Oriented Programming

Object-Oriented Programming (OOP)[1] is a method of programming that takes problems, such as simulating real world entities such as, in my use cases, companies and users, and breaks them down into smaller tasks and representations using code[2]. I hate utilizing OOP in my program to model the relationship more easily between users and companies, and companies and their data.

Code 1 Object-Oriented Programming.

Class acts as the structure for the object to be created.

```
/**
 * Represents a company with name, description, country, and financial statistics.
 */
 Alexandros Lekkas
public class Company {

    // Logger.
    22 usages
    private static final Logger logger = Logger.getLogger(Company.class.getName());

    // Next Company in CompanyList.
    2 usages
    private Company next = null;

    // Company File.
    2 usages
    private File file;

    // File path to the Company.
    7 usages
    private final String filePath;
```

The "private" access modifier is used for securiry.

When object is created it has these attributes.

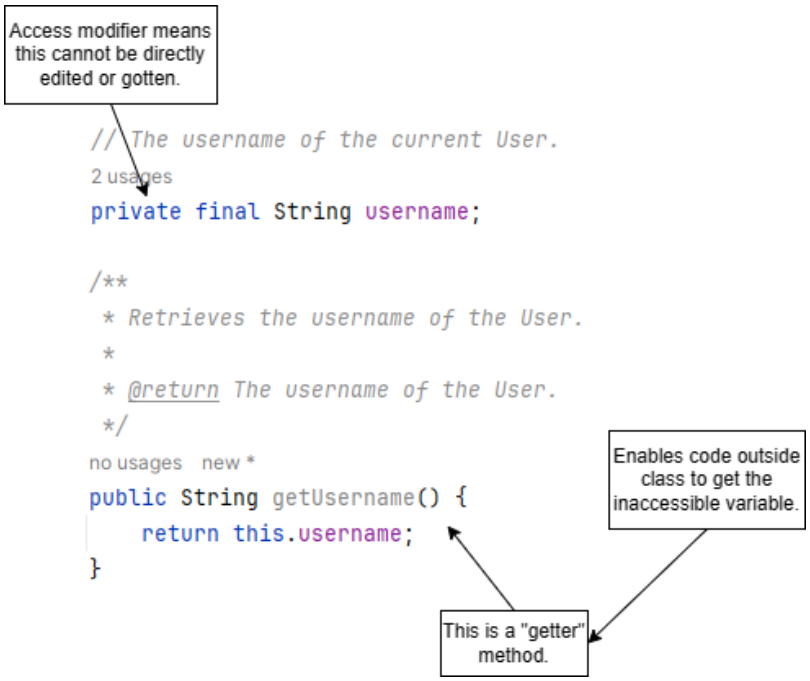
Attributes are in the class.

[1] (Gillis, n.d.)
[2] (Liang, 2017)

Encapsulation

Encapsulation[3] refers to the building of data with the mechanisms that operate the data. This means advanced data security, meaning that we can define how different classes or users of the program are allowed to access certain pieces of data.

Code 2 Encapsulation.



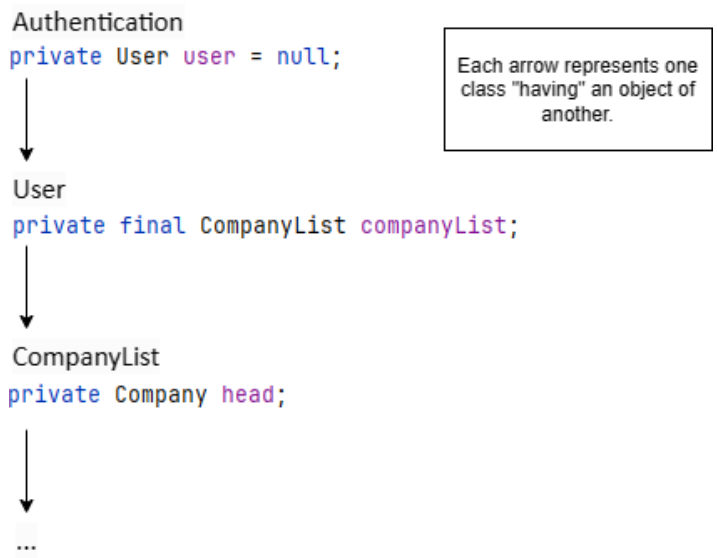
Aggregation

To represent the complex relationships contained in my program, I used aggregation[4], where one class has a 'has' relationship with another. What this means is that one class contains an object of another class.

---

[3] (S, 2023)
[4] (Aggregation in Java, n.d.)

Code 3 Aggregation.



Method Overloading

Method overloading[5] is a feature in Java is a feature allowing for a class to have more than one method with the same name but with different attributes. In my program, I used this for creating a Company object in different scenarios.

Code 4 Method overloading.

```
/** Constructs a Statistic object with the given name and file path. ...*/
2 usages   Alexandros Lekkas
public Statistic(String name, String filePath) {

    this.name = name;
    this.filePath = filePath;
    this.data = new ArrayList<>(); // Initialize the data ArrayList.

}

/** Statistic constructor with data existing. Used to model a statistic separately from a company. ...*/
1 usage   Alexandros Lekkas
public Statistic(String name, ArrayList<Data> data) {

    this.name = name;
    this.data = data;

}
```

[5] (Java Method Overloading, n.d.)

Error Handling

Using Try/Catch

Try/Catch is a convenient functionality of Java[6], where if there is an error during the running of the program, if the "catch" implementation is there, the program will not fail. Instead, we can create a way for the program to respond to the error (this is used to fulfill success criterion 3).

Code 5 Try/catch error handling.

Attempts to perform what is in brackets.

The first line executed within the "try".

```
try (BufferedReader bufferedReader = new BufferedReader(new FileReader(this.filePath))) {

    processStatistics(bufferedReader);

} catch (IOException ioException) {

    // Output error message.
    logger.severe(ioException.getMessage());
    JOptionPane.showMessageDialog(

            parentComponent: null,
            message: "Error with IO. File potentially not found.",
            title: "IO Error",
            JOptionPane.ERROR_MESSAGE

    );

}
```

In this case we are only catching errors relating to IO.

Catch followed by the exception we are looking for...

Severe error message logged using Logger (see Libraries).

"getMessage()" method gets detailed error message.

Simple popup message prevents complication.

[6] (Java Exceptions - Try...Catch, n.d.)

User Input Validation

Registration/Login

During login and registration, the user inputs their username and password, we validate if they are blank or equal to the placeholder text using if statements (this technique was used to satisfy success criterion 3).

Code 6 Registration/login input validation.

```
if (usernameTextField.getText().isBlank() || passwordTextField.getText().isBlank()) {

    // Output error message and return.
    JOptionPane.showMessageDialog(
            parentComponent: null,
            message: "Please enter a username and password.",
            title: "Error",
            JOptionPane.ERROR_MESSAGE
    );
    return;

}

if (usernameTextField.getText().equals("Enter your username") || passwordTextField.getPassword().equals("Password")) {

    // Output error message and return.
    JOptionPane.showMessageDialog(
            parentComponent: null,
            message: "Please enter a valid username and password.",
            title: "Error",
            JOptionPane.ERROR_MESSAGE
    );
    return;

}
```

Check if the fields are empty.

Check if the fields contain their respective placeholder text.

Display error message to user.

Company Addition

When adding a Company from a file, the file selected by the user is checked to see if it is a CSV file[7] (this technique was used to satisfy success criterion 3).

[7] (Comma Separated Values (CSV) Standard File Format, n.d.)

Code 7 Company file checking.

```
File selectedFile = companyFileChooser.getSelectedFile();

// Check if the file has a .csv extension.
if (selectedFile.getName().toLowerCase().endsWith(".csv")) {

    // Add to the company.
    model.CompanyList companyList = user.getCompanyList(); // Retrieve the list of companies.
    company = new model.Company(selectedFile.getAbsolutePath());
    companyList.add(company); // Add the company to the company list.
    companyList.save();
    createCompanyButtons();

} else {

    // The selected file is not a CSV file.
    JOptionPane.showMessageDialog( parentComponent: this,  message: "Please select a CSV file.",  title: "Invalid File Type", JOptionPane.ERROR_MESSAGE);

}
```

Check if the file selected by the user is a CSV file.

Output an error dialog if the file is not a CSV type.

Use of Files

Authentication

For storing the data of authenticated users, I implemented a RandomAccessFile[8]. A random-access file is a file which "behaves like a large array of bytes stored in a file system." I used this file for my user authentication system. I use the Random-Access File because it is easy to create it when the program is run, and it has improved efficiency in CRUD operations[9]. In this file, I store data in an expected structure of a string paired with another string to represent the username and password.

Code 8 RandomAccessFile.

```
// Access the userbase file as a random access file.
try (RandomAccessFile file = new RandomAccessFile(userbaseFilePath, mode: "rw")) {

    // Check if the username already exists in the file.
    while (file.getFilePointer() < file.length()) {

        if (file.readUTF().equals(password)) {

            file.close();

            JOptionPane.showMessageDialog( parentComponent: null,
                                           message: "Username already exists!",
                                               title: "Error",
                                                   JOptionPane.ERROR_MESSAGE);
            return null;

        }

        file.readUTF(); // Consume password.

    }
```

[8] (Oracle, 2024)
[9] (Debnath, 2021)

User

Creating User File

When a new user creates an account, we create a new CSV file to hold which companies belong to them—named after their username, aimed towards fulfilling criterion 1.

Code 9 Creating User file.

```
import java.io.PrintWriter;

// Handle user creation.
String userFilePath = resources.Variables.dataFolderPath + "/Users/" + username + ".csv"; // Define user file.
try (PrintWriter printWriter = new PrintWriter(userFilePath)) {

    this.user = new User(username, userFilePath); // Create user obje
    return this.user; // Return new user.
```

Import the PrintWriter library to create a new file.

File is named after the user.

We create a variable for the file path of the new file.

New PrintWriter created at specified file path.

Since it does not exist a new file is created.

Retrieving User's Companies

To be able to generate the dashboard interface with the correct companies belonging to the user showing the program must first load the company files the user has.

Code 10 Retrieving companies for the user.

```
public CompanyList(String filePath) {

    // Create the linked list of Company objects
    this.head = null;
    this.filePath = filePath;

    // Loop through user file and create and add Company objects.
    try {

        // Use a Scanner object to read the first line in the file.
        Scanner scanner = new Scanner(new File(this.filePath));
        String line = scanner.nextLine();
        String[] companyFileNames = line.split( regex: ",");

        // Loop through the line, creating a new Company object for each file location written.
        for (String companyFileName : companyFileNames) {

            if (companyFileName.toLowerCase().endsWith(".csv")) { // If the file is a CSV file...

                // Create a new Company object and add it to the linked list.
                Company company = new Company( filePath: resources.Variables.dataFolderPath + "/Companies/" + companyFileName);
                this.add(company);
```

Initialises the new Linked List of Company objects for the user.

Loop through company file and create Company objects.

New company object if file name is CSV file.

Finally, add the Company to the Linked List.

Company

Each company has their data stored in a CSV file, a text file format that uses commas to separate values[10]. Acting somewhat as a table, this file is useful as it can be opened and modified in programs like Microsoft Excel[11], a program used frequently by the client, hence, why the file format was chosen. This fulfills success criterion 2.

Structure

The file was broken into two sections using headers, details, and statistics. The details section holds basic information about the company (such as its name, description, and country), while the statistics section holds different revenues and costs as well as the associated data with each.

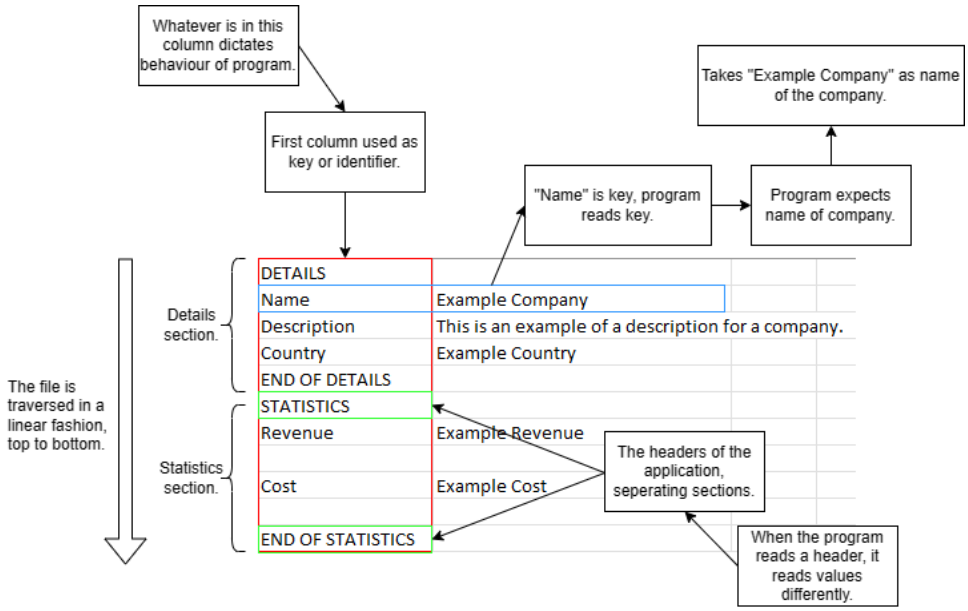


Figure 1 Company CSV file structure.

[10] (Working with CSV files, n.d.)
[11] (Microsoft, n.d.)

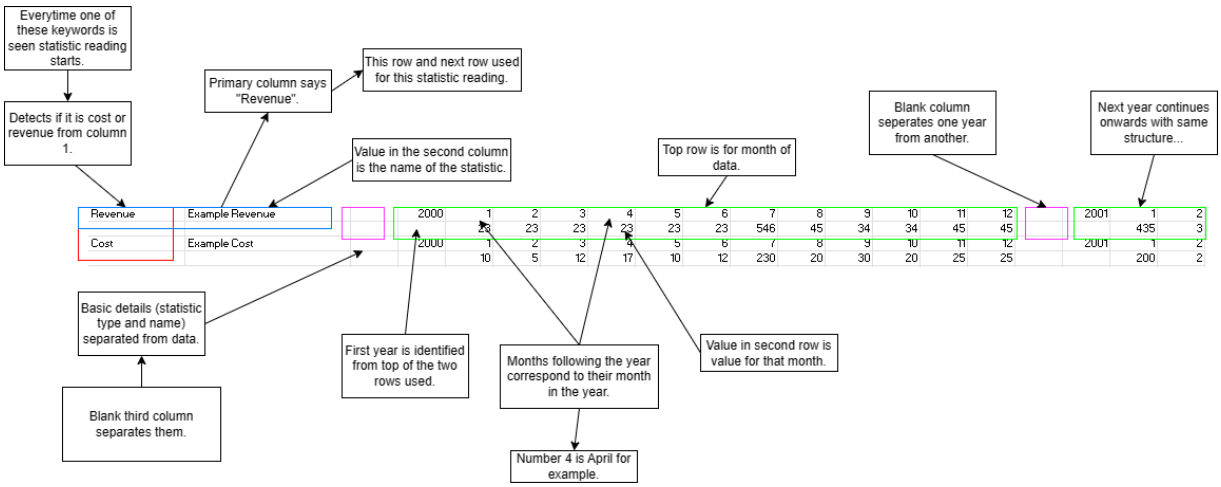


Figure 2 Statistic section and data structure.

Retrieving Company Data

The use of CSV files for storing dynamic company data leads to complicated file loading and sorting. When checking for the data of a company the program looks for specific headers within the file, such as “DETAILS” and “STATISTICS”.

Code 11 Reading company details method code.

Create a new BufferedReader to read the file.

```
private void readCompanyDetails() {

    try (BufferedReader bufferedReader =  new BufferedReader(new FileReader(filePath))) {

        String line;
        while ((line = bufferedReader.readLine()) != null) { processLine(line); }
```

Loop through the file processing each line.

Code 12 Checking for the statistics header code.

Loop through the file.

```
// Find where the statistics data begins in the file.
found = false;
while (!found) { // Loop through the file until statistics data is found...

    String[] currentLine = bufferedReader.readLine().split( regex: ",");

    try {

        // Check if the current line is equal to "STATISTICS".
        if (currentLine[0].equals("STATISTICS")) { // If the current line contains "STATISTICS"...

            found = true;

        }
```

If start of statistic section is found we save some details.

Code 13 Adding statistics from file.

Simple check to see if it is revenue or cost.

```
private void addStatistic(String[] currentLine) {

    if (currentLine[0].equalsIgnoreCase( anotherString: "REVENUE")) {

        revenues.add(new Statistic(currentLine[1], this.filePath));

    } else if (currentLine[0].equalsIgnoreCase( anotherString: "COST")) {

        costs.add(new Statistic(currentLine[1], this.filePath));

    }

}
```

Retrieving Statistic Data

Code 14 Starting to read Statistic data code.

```
public void readData() {

    logger.info( msg: "Reading Data for Statistic " + this.name + ".");

    try {

        FileReader fileReader = new FileReader(this.filePath);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        findStatisticInFile(bufferedReader);
```

Method is used to read data for statistic.

Calls method to find statistic within file

Code 15 Searching for Statistic data code.

```
private void findStatisticInFile(BufferedReader bufferedReader) throws IOException {

    boolean found = false;

    while (!found) {

        String[] currentLine = bufferedReader.readLine().split( regex: ",");
        logger.info( msg: "Current line: " + Arrays.toString(currentLine) + ".");

        try {

            if (currentLine.length > 1 && currentLine[1].equalsIgnoreCase(name)) {

                String[] nextLine = bufferedReader.readLine().split( regex: ",");
                processDataLines(currentLine, nextLine);
                found = true;

            }
```

Loop through file until statistic data is found.

If found line data is processed and loop breaks.

Code 16 Adding Statistic data code.

Method processes two statistic lines.

```
private void processDataLines(String[] currentLine, String[] nextLine) {

    int i = 3;

    while (i < currentLine.length && !currentLine[i].isEmpty()) {

        int year = Integer.parseInt(currentLine[i]);
        int j = 4;

        processYearData(currentLine, nextLine, i, j, year);
        i += 14;

    }
}
```

This method processes one year's data.

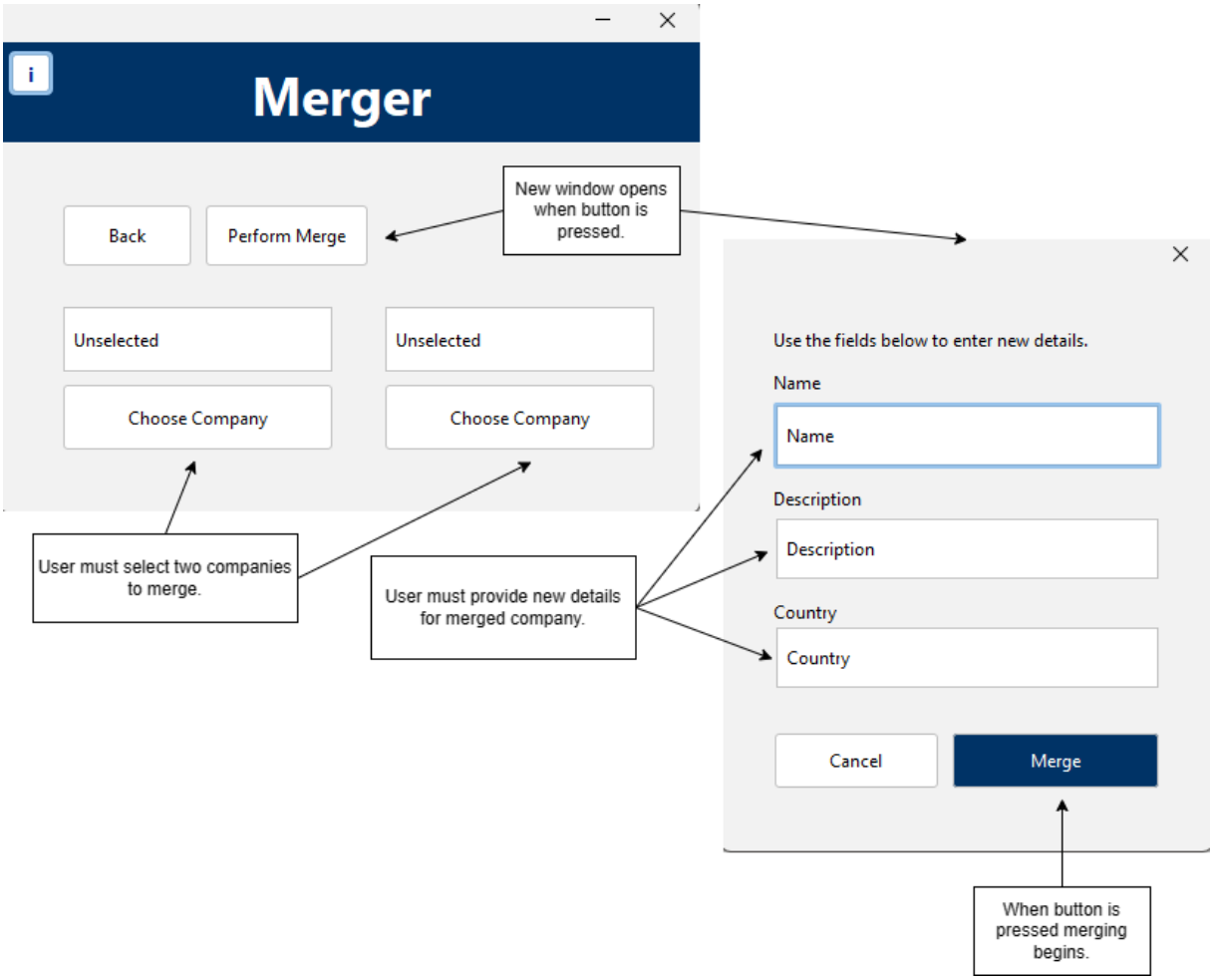
```
private void processYearData(String[] currentLine, String[] nextLine, int i, int j, int year) {

    try {

        while (j < currentLine.length && Integer.parseInt(currentLine[j]) <= 12) {

            int value = (int) Double.parseDouble(nextLine[j]);
            data.add(new Data(year, Integer.parseInt(currentLine[j]), value));
            logger.info(data.toString());
            j++;

        }
```

Loop through year's data and add to Statistic data.

Functionality

Company Merging (ArrayList of Objects[12], For Each Loop[13] & Comparator[14])

This feature was implemented to address success criterion 11.



Screenshot 2 Company merging flow.

[12] (Lawrence University, n.d.)
[13] (GeeksforGeeks, 2023)
[14] (Oracle, 2024)

Code 17 Merge companies method code snippet.

```java
public void mergeCompanies(Company company1, Company company2, String name, String description, String country) {

    String filePath = resources.Variables.dataFolderPath + "/Companies/" + name + ".csv";

    File file = new File(filePath); // Create a new file from the file path that is common to companies.
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(file, append: true))) {

        // Write down the company details to file.
        logger.info( msg: "Writing details to new File.");
        writer.write( str: "DETAILS\n");
        writer.write( str: "Name," + name + "\n");
        writer.write( str: "Description," + description + "\n");
        writer.write( str: "Country," + country + "\n");
        writer.write( str: "END OF DETAILS\n");
```

Creates a new file to store data for the merged Company.

Write all the basic Company details to file.

Code 18 Combination of statistics code snippet.

New ArrayList holds combined Statistics.

```java
ArrayList<Statistic> combinedStatistics = new ArrayList<>(); // Create the new ArrayList to save the merged details.

// Duplicate lists to use for re-adding.
ArrayList<Statistic> statistics1Copy = new ArrayList<>(statistics1);
ArrayList<Statistic> statistics2Copy = new ArrayList<>(statistics2);

// Loop through the two statistics finding similar ones                                    nding logic.
for (Statistic statistic1 : statistics1) {

    // Loop through specifically the second one, doesn't                                    things are.
    for (Statistic statistic2 : statistics2) {

        logger.info( msg: "Comparing " + statistic1.getName() + " to " + statistic2.getName());

        // Check if statistics have the same name, made to uppercase to avoid character issues.
        if (statistic1.getName().equalsIgnoreCase(statistic2.getName())) {

            // Remove statistics from their ArrayLists.
            statistics1Copy.remove(statistic1);
            statistics2Copy.remove(statistic2);

            // Combine data points from stat1 and stat2.
            ArrayList<model.Data> combinedData = combineData(statistic1.getData(), statistic2.getData());
            combinedStatistics.add(new Statistic(statistic1.getName(), combinedData));

        }

    }

}
```

Duplicate Statistic ArrayLists for revenues and costs created.

We can easily reference the current object.

For each loop allows for easy looping.

I used this as it works with ArrayLists.

Existing data stored to re-add non-combined statistics.

We combine data from statistics with the same name and type.

Code 19 Combination of data.

Used for each loops.

Same reasons as previously explained (see previous Code).

```
// Loop through the data and check if data points are existing.
for (model.Data dataPoint1 : data1) {

    System.out.println(dataPoint1.toString());

    // Loop through specifically second data here, order does not matter.
    for (model.Data dataPoint2 : data2) {

        System.out.println(dataPoint2.toString());

        // If they have the same data of data then...
        if (dataPoint1.getYear() == dataPoint2.getYear() && dataPoint1.getMonth() == dataPoint2.getMonth()) {

            // Remove the data values.
            data1copy.remove(dataPoint1);
            data2copy.remove(dataPoint2);

            combinedData.add(new model.Data(dataPoint1.getYear(), dataPoint1.getMonth(), (dataPoint1.getValue() + dataPoint2.getValue())));
```

Using ArrayList of objects allows for easy list object removal.

We can reference the object with the ".remove" method.

Add the new combined data point to the combined data.

Code 20 Sorting of combined Data objects using a Comparator.

Use of Comparator allows for easy definition and clear use.

Comparator allows for complex sorting rules inline with collection sorting methods.

Comparator defined between two Data objects.

Custom comparator used to sort through ArrayList.

If year is not the same as year compared to...

Compare objects based on month.

```
// Sort the combined data.
combinedData.sort((dataPoint1, dataPoint2) -> {

    if (dataPoint1.getYear() != dataPoint2.getYear()) {

        return Integer.compare(dataPoint1.getYear(), dataPoint2.getYear());

    } else {

        return Integer.compare(dataPoint1.getMonth(), dataPoint2.getMonth());

    }

});
```

Company Valuation (ArrayList of Objects, For Each Loop, For Loop[15])

The ability to predict the value of a company was added in response to success criterion 9. This also addresses criterion 10, as the code snippets can be used to predict the value of a company over varying timespans, meaning, that the user can input a different number of years to predict for.

Data Extrapolation

Code 21 All data extrapolation.

```
private ArrayList<Data> extrapolateAllData(ArrayList<Statistic> statistics, int yearsToExtrapolate) {

    logger.info( msg: "Extrapolating all data.");

    ArrayList<Data> combinedData = new ArrayList<>();

    for (Statistic statistic : statistics) {

        logger.info( msg: "Statistic: " + statistic.getData());
        logger.info( msg: "Data before: " + statistic.getData());

        ArrayList<Data> extrapolatedData = statistic.extrapolateData(yearsToExtrapolate);
        combinedData.addAll(extrapolatedData);

        logger.info( msg: "Data after: " + extrapolatedData);
        logger.info( msg: "Combined data size: " + combinedData.size());

    }

    return combinedData;

}
```

New ArrayList to hold old and new data created.

Extrapolate the data for each statistic in the ArrayList of statistics.

[15] (Simplilearn, 2022)

Code 22 Data extrapolation.

```java
public ArrayList<Data> extrapolateData(int yearsToExtrapolate) {

    readData(); // Read Data from Statistic.

    // Get Data for the latest year of the Statistic.
    ArrayList<Data> dataLastYear = new ArrayList<>(data.subList(data.size() - 12, data.size()));
    if (yearsToExtrapolate == 0) {

        return dataLastYear;

    }

    ArrayList<Data> extrapolatedData = new ArrayList<>(dataLastYear);

    int lastYear = data.get(data.size() - 1).getYear(); // Get the latest year.
    logger.info( msg: "Last year: " + lastYear);

    // Loop through for every year.
    for (int i = 0; i < yearsToExtrapolate; i++) {

        // Add 12 data points for every year.
        for (int month = 0; month < 12; month++) {


            // Extrapolate a new data point, based on data from the last 12 months in the extrapolated data
            Data newDataPoint = calculateLinearExtrapolationData(extrapolatedData, month, nextYear: lastYear + i);

            extrapolatedData.add(newDataPoint);

        }
```

Re-read the Statistic data just to be sure.

Sublist of data for full last year created.

Use linear extrapolation to extrapolate the current data point.

Code 23 Extrapolation of data points using linear regression calculations.

Predicts data based on the data from the latest year.

```
private static Data calculateLinearExtrapolationData(ArrayList<Data> dataLastYear, int month, int nextYear)

    int n = dataLastYear.size(); // Number of data points
    double sumX = 0; // Sum of years (as x values)
    double sumY = 0; // Sum of values (as y values)
    double sumXY = 0; // Sum of x*y
    double sumX2 = 0; // Sum of x squared

    // Loop through data
    for (Data dataPoint : dataLastYear) {
        if (dataPoint.getMonth() == month + 1) {
            int x = dataPoint.getYear();
            double y = dataPoint.getValue();

            sumX += x;
            sumY += y;
            sumXY += x * y;
            sumX2 += x * x;
        }
    }

    // Linear regression equations
    double exaggeratedFactor = 2.0;
    double slope = ((n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX)) * exaggeratedFactor;
    double intercept = (sumY - slope * sumX) / n;

    // Use the linear model to project the value into the future
    double projectedValue = intercept + slope * nextYear;
```

Accumulators for sums needed.

Part of linear regression formula.

For each loop used to iterate over ArrayList.

Certify only data from correct month is being used.

Running totals are calculated for the regression components.

Artificial amplification added to increase data sensitivity.

Without this data predictions are not realistic.

The slope is computed using formula for linear regression.

Value if projected using components of linear regression.

Final Calculation

Code 24 Final value calculation.

All revenues and costs are extrapolated.

```
// Read and extrapolate data once.
ArrayList<Data> allRevenueData = extrapolateAllData(newRevenues, monthsToExtrapolate);
ArrayList<Data> allCostData = extrapolateAllData(newCosts, monthsToExtrapolate);

// Calculate the valuation only for the most recent complete year
double revenueSum = sumDataValues(allRevenueData);
double costSum = sumDataValues(allCostData);

double netProfit = revenueSum - costSum;
int profitMultiplier = 5; // Adjust this multiplier as necessary.

// Return the final prediction for the company's value.
return (int) (netProfit * profitMultiplier);
```

Revenue and cost data summed up for year.

Profit is calculated from revenues - costs.

Sensible multiplier is set to do correct valuation.

Final company value based on year's profit multiplied by profit multiplier.

Statistic Graphing (ArrayList of Objects, For Each Loop)

Statistics are graphed using the JFreeChart library[16]. This feature aims to fulfill success criterion 7.

[16] See Libraries section of document.

Code 25 Converting Statistic data to JFreeChart Month.

```
statistic.readData();

TimeSeries series = new TimeSeries( name: "Data Value");

// Loop through the data of the statistic.
ArrayList<model.Data> statisticData = statistic.getData();
for(model.Data data : statisticData) {

    Month month = new Month(data.getMonth(), data.getYear());
    System.out.println(month.toString());
    series.addOrUpdate(month, data.getValue());

}
```

Series is essentially x and y data.

X is data and y is time.

ArrayList of objects used.

We loop through the Statistic data using a for each loop.

A new JFreeChart Month object is created.

Contains both year and month.

We add current time and value to the series.

Code 26 Creating JFreeChart.

```
TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(series);

// Create a chart with proper labels and dataset.
JFreeChart chart = ChartFactory.createTimeSeriesChart(

    statistic.getName(), // Title.
    "Month", // X-axis label.
    "Revenue ($)", // Y-axis label.
    dataset, // Dataset.
    true, // Legend.
    true, // Tooltips.
    false // URLs.

);
```

Collection of TimeSeries gets the series of Statistic data added.

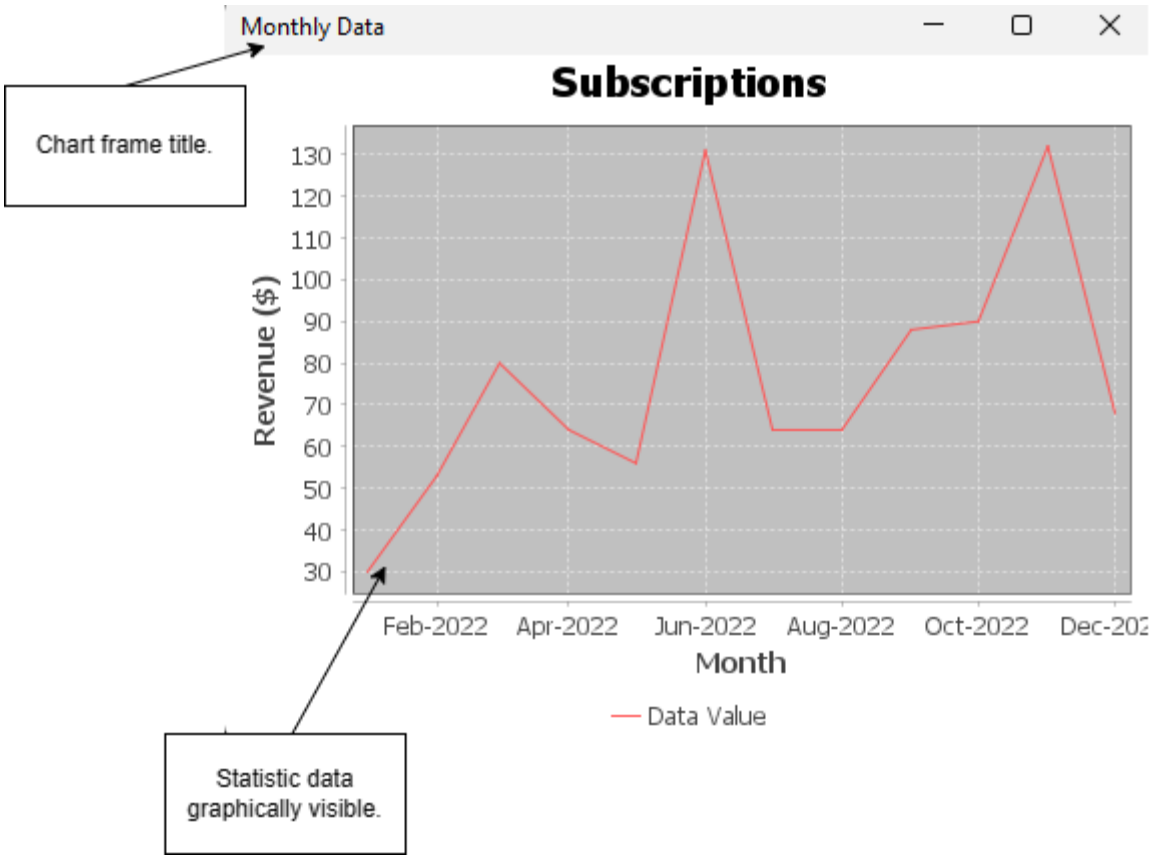
Ability to add multiple series means in future statistic comparison.

Code 27 Displaying JFreeChart.

```
ChartFrame frame = new ChartFrame( title: "Monthly Data", chart);

frame.setVisible(true);
frame.setSize( width: 480, height: 380);
```

Creates a new frame with a title.

Size and visible property changed. Frame 'pops up'.



Screenshot 3 JFreeChart graphed statistic.

Linked List of Company Objects

I decided to structure the program in such a way that the User class aggregates or 'has' a CompanyList object. The CompanyList object acts as the LinkedList controller, while each

Company object is a node within the LinkedList. To make this work, the CompanyList held by the User object has a root Company object, and each Company object has a reference to the next Company.

My main reason for using a LinkedList instead of a fixed array of objects is because it is a dynamic data structure. This is advantageous because it means that if a user wishes to add more companies or remove companies they can do so, and the data structure does not need to have a fixed or set size limitation. The only downside of this is that this is less memory efficient, however, with modern devices and the expected workload the memory efficiency is not a large concern for the client.

Interface

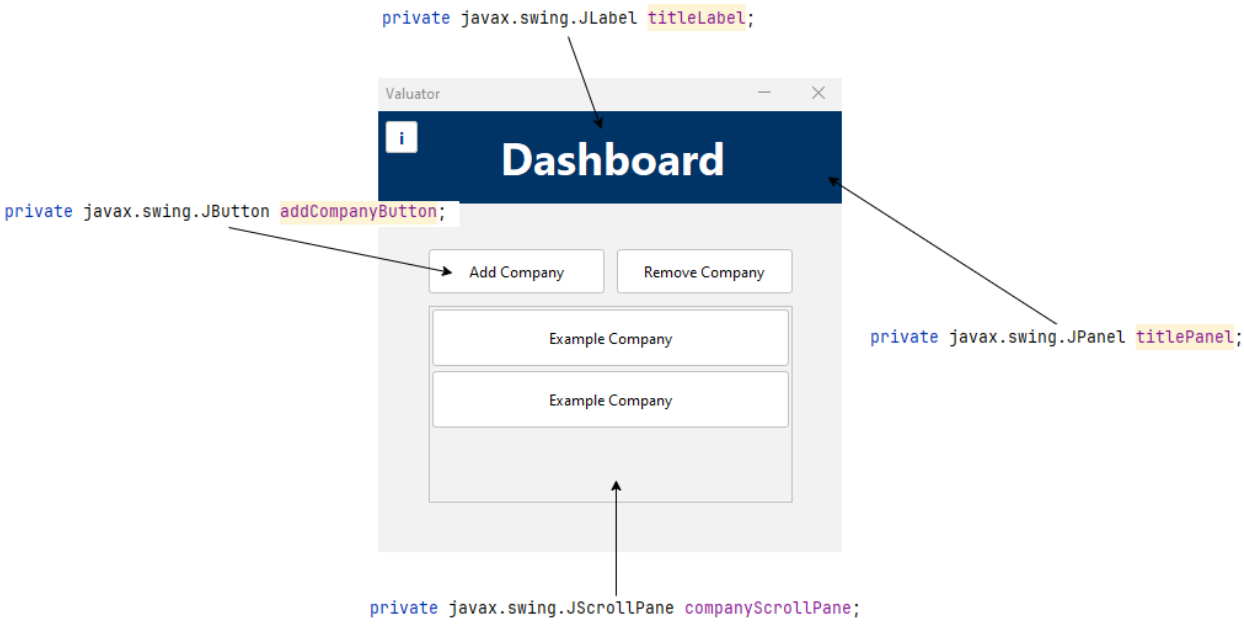
Code 28 Setting of interfaces LookAndFeel.

```
// Attempt to set the preferred look and feel.
try {

    UIManager.setLookAndFeel(new FlatIntelliJLaf());

} catch (UnsupportedLookAndFeelException unsupportedLookAndFeelException) { // Catch a look and feel error.

    // Output error message.
    logger.severe(unsupportedLookAndFeelException.getMessage());
    JOptionPane.showMessageDialog(

            parentComponent: null,
            message: "Unsupported look and feel.",
            title: "Look & Feel Error",
            JOptionPane.ERROR_MESSAGE

    );

}
```

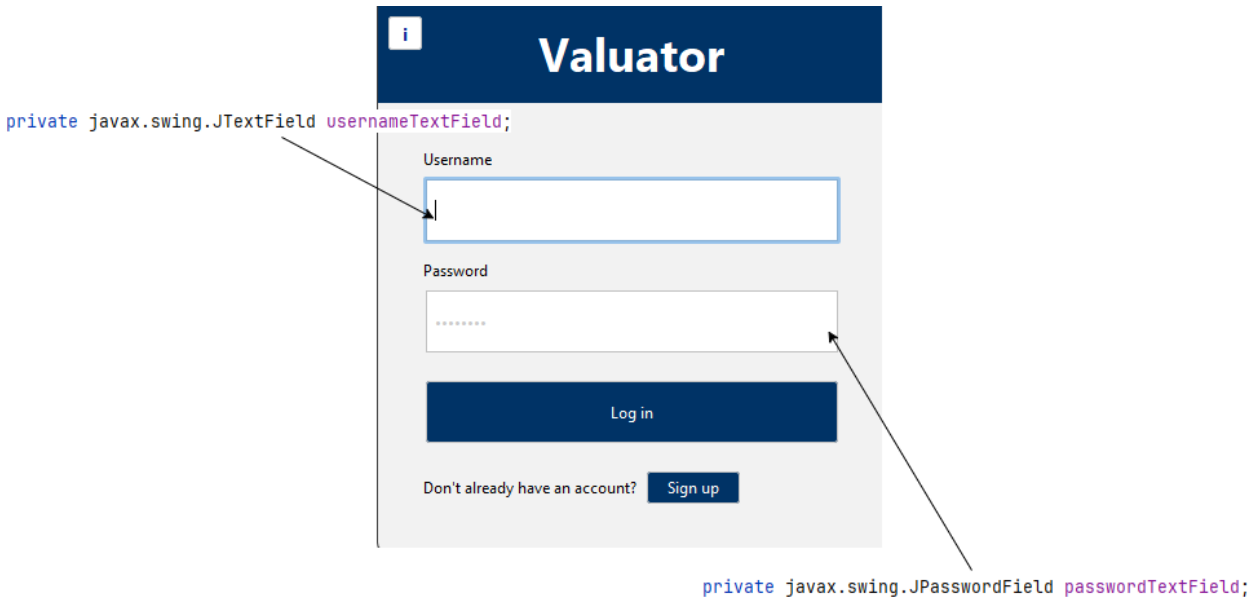
Set LookAndFeel to FlatIntelliJLaf. This is from a library.

See Libraries section in document.

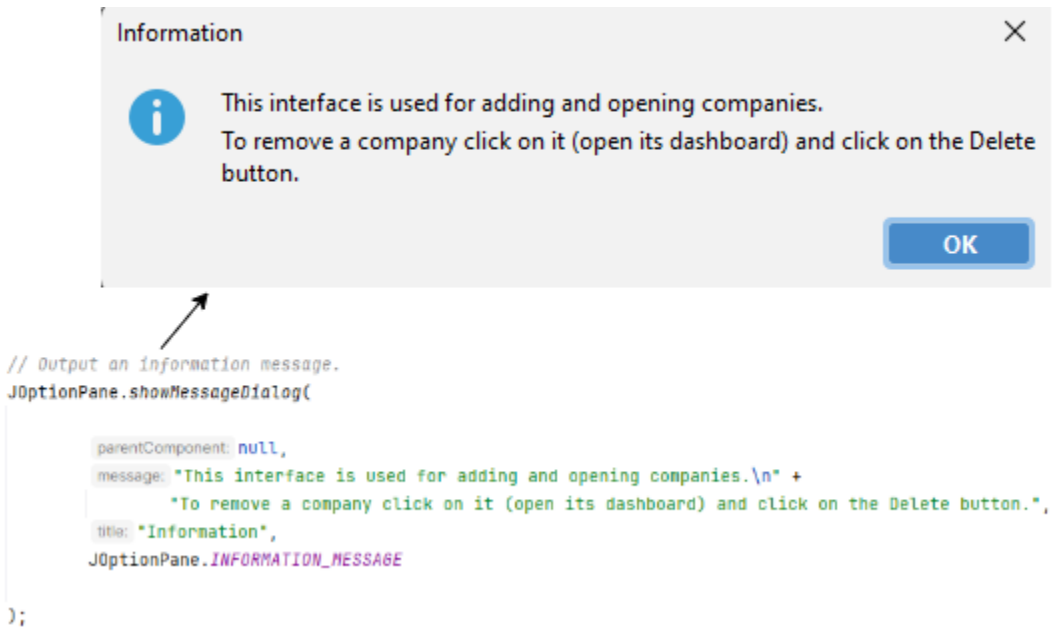
This code uses a library to make the UI prettier.



Screenshot 4 Dashboard interface with components annotated.

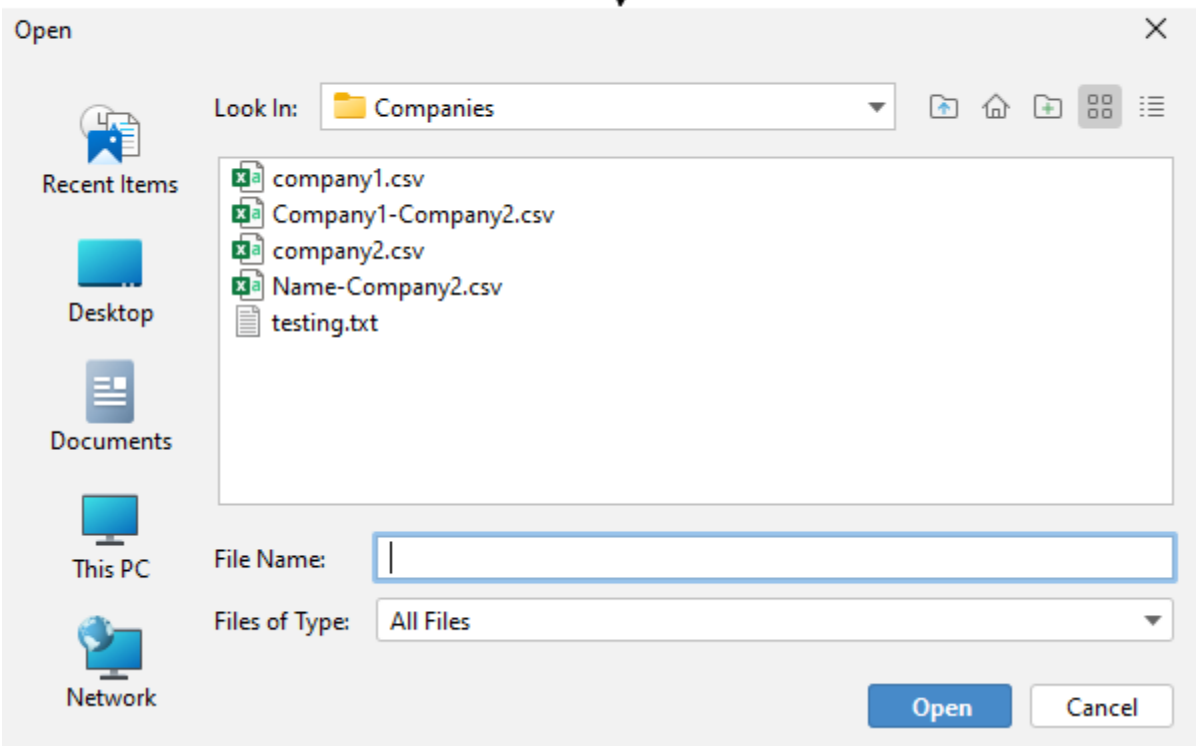


Screenshot 5 Login/signup interface with components annotated.

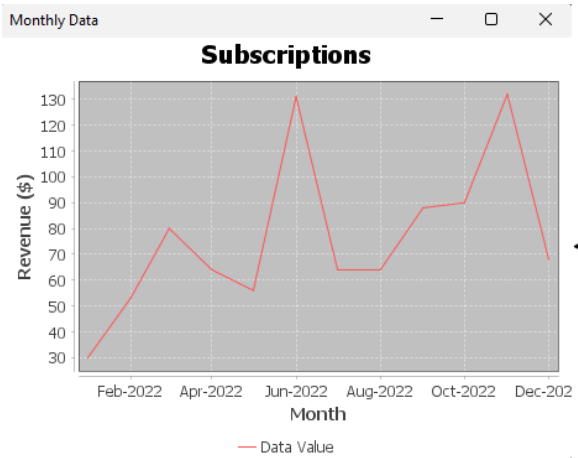


Screenshot 6 Popup with components annotated.

```
private javax.swing.JFileChooser companyFileChooser;
```



Screenshot 7 File chooser with annotated components.



This uses a library, see Libraries section in document.

```
ChartFrame frame = new ChartFrame( title: "Monthly Data", chart);

frame.setVisible(true);
frame.setSize( width: 480, height: 380);
```

Screenshot 8 Statistic graph with annotated components.

Libraries

Table 1 Libraries used in the program.

Library	Reasons for Use	Link
com.formdev.flatlaf.FlatIntelliJLaf	• Used to make interface look more visually appealing. • Makes interface 'cleaner'. Simply visual changes.	https://www.javadoc.io/static/com.formdev/flatlaf/2.0-rc1/com/formdev/flatlaf/FlatIntelliJLaf.html
java.awt.Color	• Used for setting the color of various visual elements.	https://docs.oracle.com/javase/8/docs/api/java/awt/Color.html
java.awt.Component	• Used for alignment property.	https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html
java.awt.Desktop	• Used to launch a file in an application.	https://docs.oracle.com/javase/8/docs/api/java/awt/Desktop.html
java.awt.Dimension	• Width and height of an object in a single component. • Used for sizing buttons.	https://docs.oracle.com/javase/8/docs/api/java/awt/Dimension.html
java.awt.event.ActionEvent	• Primarily used for button pressing events. • Can be used to trigger actions.	https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionEvent.html
java.io.*	• General use of different parts of the IO library. • Used for input and output in the application.	https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html
java.io.BufferedWriter	• Used through program to get data from file line by line. • Can easily read CSV file line to array.	https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html
java.io.File	• Used to act as simply a file.	https://docs.oracle.com/javase/8/docs/api/java/io/File.html

	• Used for when reading a file. • Can store the read file as a file object. • Used in a program for easy file writing and reading. ○ Benefit is path does not need to be re-specified.	
java.io.FileWriter	• Used to write details to files.	https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html
java.io.IOException	• Exception thrown if there is an issue with input output. • Used during file reading and writing. • In program catches issues and we can output to user.	https://docs.oracle.com/javase/8/docs/api/java/io/IOException.html
java.io.PrintWriter	• Also used for writing files in the program. • Can be used to create a new file (used to create user file).	https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html
java.io.RandomAccessFile	• DAT file reading. • Used for userbase (username and password storage).	https://docs.oracle.com/javase/8/docs/api/java/io/RandomAccessFile.html
java.util.ArrayList	• Used by program to store list without specified size. • Also stores in order.	https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
java.util.Arrays	• Used primarily for methods such as converting to String. • Used in program for logging.	https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

java.util.Collections	• Used for its method to find max value from ArrayList. ○ Max value of complete years for prediction.	https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html
java.util.logging.Logger	• Used in the program to log details.	https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html
java.util.NoSuchElementException	• Thrown by program when an element does not exist. ○ If a company does not exist.	https://docs.oracle.com/javase/8/docs/api/java/util/NoSuchElementException.html
java.util.Scanner	• Used to read through first file of user file with ease.	https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
javax.swing.*	• Lightweight interface components that work on multiple platforms. • Used by program for visual interface.	https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/package-summary.html
javax.swing.Box		https://docs.oracle.com/javase/8/docs/api/javax/swing/Box.html
javax.swing.BoxLayout	• A layout used on every interface.	https://docs.oracle.com/javase%2F8%2Fdocs%2Fapi%2F%2F/javax/swing/BoxLayout.html
javax.swing.Jbutton	• A simple button. • Used numerous times in program.	https://docs.oracle.com/javase/8/docs/api/javax/swing/Jbutton.html
javax.swing.JfileChooser	• Used when selecting company files.	https://docs.oracle.com/javase/8/docs/api/javax/swing/JFileChooser.html
javax.swing.JOptionPane	• Used for popups in the program.	https://docs.oracle.com/javase%2F8%2Fdocs%2Fapi%2F%2F/javax/swing/JOptionPane.html
javax.swing.JPanel	• Used for panels holding elements.	https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html
javax.swing.JScrollPane	• Used for elements such	https://docs.oracle.com/javase

	as: ○ Statistics buttons ○ Company buttons • Allows for dynamic adding of elements without compromising page size.	/8/docs/api/javax/swing/JScrollPane.html
javax.swing.UIManager	• Used to manage the LookAndFeel of the program.	https://docs.oracle.com/javase/8/docs/api/javax/swing/UIManager.html
javax.swing.UnsupportedLookAndFeelException	• Used if the library for LookAndFeel is loaded improperly. • Allows the program to still run without the desired LookAndFeel.	https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/UnsupportedLookAndFeelException.html
org.jfree.chart.ChartFactory	• Used for chart creation. ○ Statistics charts	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/ChartFactory.html
org.jfree.chart.ChartFrame	• Creating frame holding the chart.	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/ChartFrame.html
org.jfree.chart.JFreeChart	• Main object for a new chart.	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/JFreeChart.html
org.jfree.data.time.Month	• Month of data point for a statistic.	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/data/time/Month.html
org.jfree.data.time.TimeSeries	• X-axis of a graph which uses time.	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/data/time/TimeSeries.html
org.jfree.data.time.TimeSeriesCollection	• Holds a collection of data. • Is placed onto a TimeSeries. • Used when adding chronological statistic data to graph.	https://www.jfree.org/jfreechart/api/javadoc/org/jfree/data/time/TimeSeriesCollection.html

References

Aggregation in Java. (n.d.). Retrieved from Javatpoint: https://www.javatpoint.com/aggregation-in-java

Comma Separated Values (CSV) Standard File Format. (n.d.). Retrieved 2 27, 2024, from Edoceo, Inc: http://edoceo.com/utilitas/csv-file-format

Debnath, M. (2021, March 25). *Random File Access Using Java*. Retrieved from Developer.com: https://www.developer.com/database/random-file-access-using-java/

GeeksforGeeks. (2023, February 16). *For-each loop in Java*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/for-each-loop-in-java/

Gillis, A. S. (n.d.). *TechTarget*. (S. Lewis, Editor) Retrieved from What is object-oriented programming?: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

Java Exceptions - Try...Catch. (n.d.). Retrieved from W3schools: https://www.w3schools.com/java/java_try_catch.asp

Java Method Overloading. (n.d.). Retrieved from W3schools: https://www.w3schools.com/java/java_methods_overloading.asp

Lawrence University. (n.d.). *Working with ArrayLists*. Retrieved from Lawrence: https://www2.lawrence.edu/fast/GREGGJ/CMSC150/062ArrayLists/ArrayLists.html#:~:text=What%20is%20an%20ArrayList%3F,a%20size%20for%20the%20array.

Liang, Y. D. (2017). Object Oriented Thinking. In Y. D. Liang, *Introduction to Java Programming, Brief Version, Eleventh Edition, Global Edition* (pp. 390-407). Pearson.

Microsoft. (n.d.). *Free online spreadsheet software: Excel: Microsoft 365*. Retrieved from Free Online Spreadsheet Software: Excel | Microsoft 365: https://www.microsoft.com/en-us/microsoft-365/excel

Oracle. (2024, January 8). *Comparator (java platform SE 8)*. Retrieved from Oracle: https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

Oracle. (2024, January 8). *RandomAccessFile (Java Platform SE 8)*. Retrieved from Oracle: https://docs.oracle.com/javase/8/docs/api/java/io/RandomAccessFile.html

S, R. A. (2023, October 11R). *What is Encapsulation in Java and How to Implement It*. Retrieved from simplilearn: https://www.simplilearn.com/tutorials/java-tutorial/java-encapsulation#:~:text=Encapsulation%20in%20Java%20is%20the,the%20class%20can%20access%20them.

Simplilearn. (2022, December 22). *Understanding for loop in java with examples and syntax*. Retrieved from Simplilearn: https://www.simplilearn.com/tutorials/java-tutorial/for-loop-in-

java#:~:text=For%20loop%20in%20Java%20iterates,the%20condition%20is%20not%20met.

Working with CSV files. (n.d.). Retrieved from Fund Recs: https://www.fundrecs.com/blog/working-with-csv-files#:~:text=CSV%20(Comma%20Separated%20Values)%20format,it%20in%20slightly%20different%20ways.