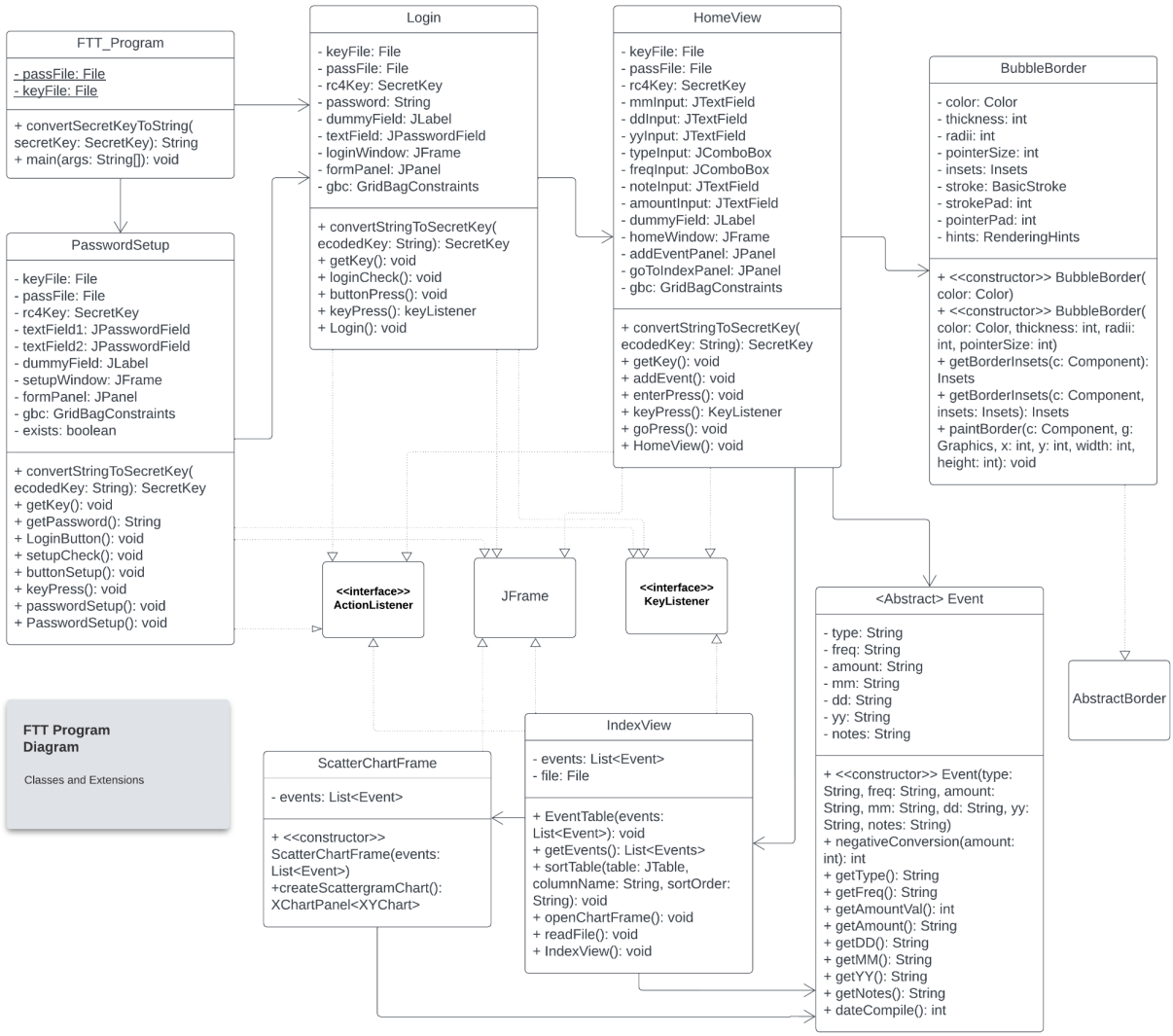
Criterion C: Development

UML Diagram:

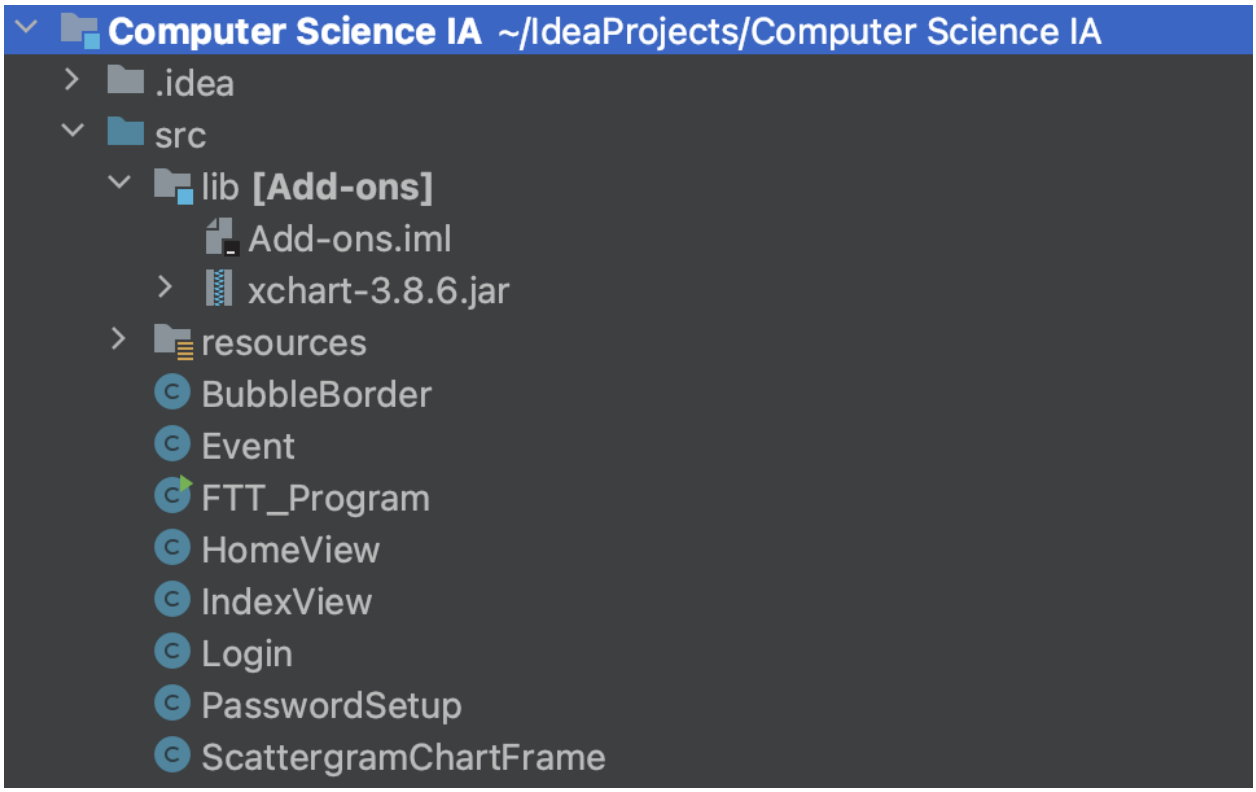


List of techniques:

- ArrayList
- OOP
- Encryption and decryption
- Inheritance
- RandomAccessFile amend and delete
- GUI
- Sorting algorithms
- External libraries/functions

“Relevant” screenshots of “complex code” with Design methodologies (sentences explaining images)

All Classes:



RandomAccessFiles:

Akq3WktMOiNg.txt
eo3cgEUjJjNT.txt
m87UonM12v5h.txt

First Class:

```
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.io.File;
import java.io.FileWriter;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
```

These imports are set up in the beginning class FTT_Program to set up vital write functions and the creation of an encryption key the entire program utilizes.

```
public static void main(String[] args) {
    try {
        if (!file.exists()) {
            KeyGenerator keyGenerator = KeyGenerator.getInstance( algorithm: "RC4");
            SecretKey rc4Key = keyGenerator.generateKey();
            FileWriter writeTo = new FileWriter(file,  append: true);
            writeTo.write(convertSecretKeyToString(rc4Key));
            writeTo.close();
        }
    } catch (Exception e) {
        System.out.println("Exception in encryption generation");
        e.printStackTrace();
    }
    PasswordSetup startScreen = new PasswordSetup();
}
```

Main method begins by checking if the application has amended a password to a specific .txt file.

Catch exception prevents errors with accessing files.

Final call to password creation GUI.

```
public static String convertSecretKeyToString(SecretKey secretKey) {
    byte[] rawData = secretKey.getEncoded();
    return Base64.getEncoder().encodeToString(rawData);
}
```

Passing the conditional uses cryptography imports to create a randomized encryption key, which is then recorded in a .txt file. The data type of the key undergoes a casting method that converts its type into a byte array and then a suitable string for writing.

Password Setup GUI:

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import java.io.FileWriter;
import java.util.Base64;
import java.util.Scanner;
```

Continuation of previous imports, and new imports for GUI

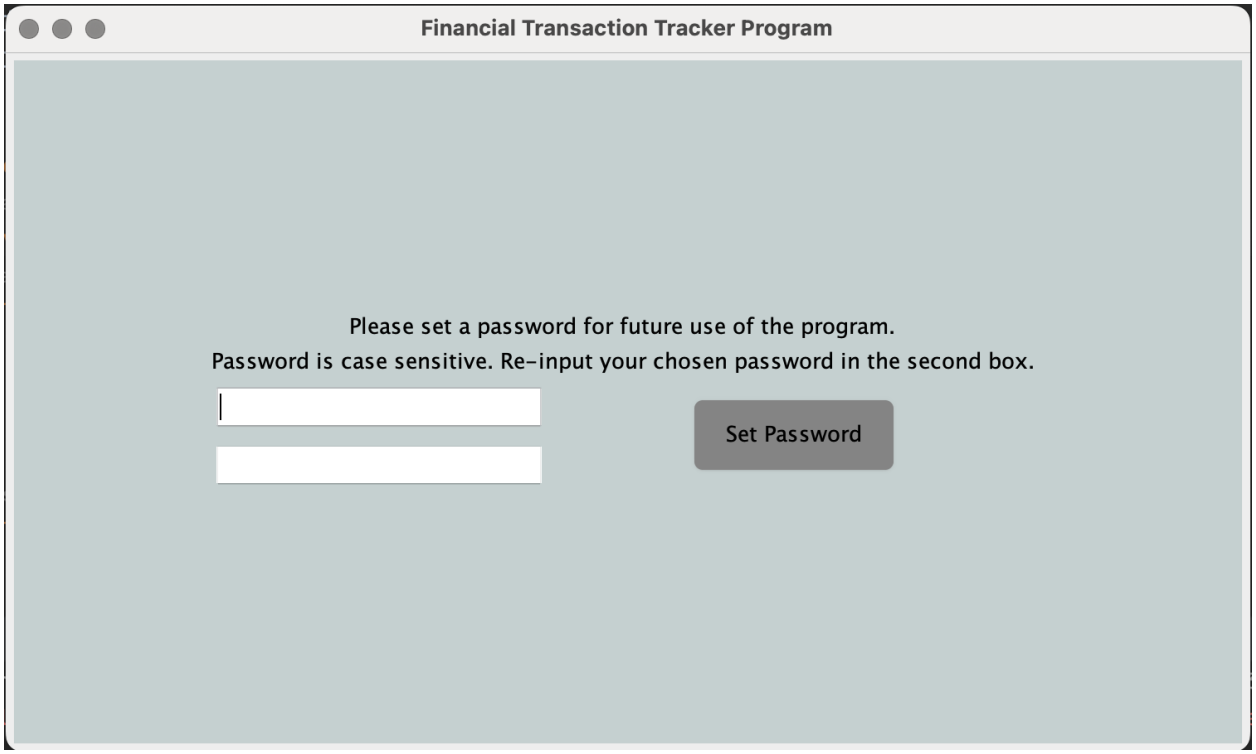
```
public class PasswordSetup extends JFrame {
    2 usages
    private File file = new File( pathname: "/Users/*********/eo3cgEUjJjNT.txt");
    1 usage
    private File keyFile = new File( pathname: "/Users/*********/Akq3WktMOiNg.txt");
    2 usages
    private SecretKey rc4Key;
    3 usages
    private JPasswordField textField1 = new JPasswordField( columns: 16);
    3 usages
    private JPasswordField textField2 = new JPasswordField( columns: 16);
    4 usages
    private JLabel dummyField = new JLabel();
    7 usages
    private JFrame setupWindow = new JFrame( title: "Financial Transaction Tracker Program");
    9 usages
    private JPanel formPanel = new JPanel(new GridBagLayout());
    26 usages
    private GridBagConstraints gbc = new GridBagConstraints();
    3 usages
    private boolean exists = false;
```

Class extends JFrame for GUI creation.

Declaration of JFrame elements and file routes at the beginning of class for easy access and alterability.

```
public PasswordSetup() {
    try {
        if (file.exists()) {
            Login directLoginScreen = new Login();
        } else {
            passwordSetup();
        }
    } catch (Exception exception) {
        System.out.println("Exception in login directory.");
    }
}
```

Conditional to check if specific file exists, and hence an already defined password. Passing streamlines to next GUI class immediately. Failing resumes execution of the class.



JFrame title, JPanel, 2 JLabels, 2 JPasswordBoxes, and a JButton.

GridBoxLayout for JFrame allows full customization of layout and sizing.

Use of JPasswordBoxes for privacy when entering inputs.

Background color uses Hex Code for specific color.

```
public void passwordSetup() {
    JPanel panel = new JPanel(new BorderLayout());
    panel.setBorder(BorderFactory.createEmptyBorder( top: 5,  left: 5,  bottom: 5,  right: 5));
```

Constructor GUI class uses in-lay panel and imports for stylized border and elements.

```java
private class buttonSetup implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (!exists) {
            setupCheck();
        } else {
            LoginButton();
        }

    }
}

2 usages
private class keyPress implements KeyListener {
    public void keyTyped(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ENTER) {
            if (!exists) {
                setupCheck();
            } else {
                LoginButton();
            }
        }
    }

    public void keyPressed(KeyEvent e) {

    }

    public void keyReleased(KeyEvent e) {

    }
}
```

ActionListener for JButton press for Set Password button. KeyListener for keyboard press for Set Password button. KeyListener method checks for enter key press.

Conditional in both methods to differentiate between button presses allowing for reuse of method with other JButton.

```
private void setupCheck() {
    String userInput1 = textField1.getText();
    String userInput2 = textField2.getText();

    if (userInput1.isEmpty() || userInput2.isEmpty()) {
        dummyField.setText("Please enter a password in both fields.");
    } else if (userInput1.equals(userInput2)) {
        try {
            getKey();
            FileWriter writeTo = new FileWriter(file,  append: true);
            Cipher rc4Cipher;
            rc4Cipher = Cipher.getInstance( transformation: "RC4");
            byte[] text = userInput1.getBytes( charsetName: "UTF8");
            rc4Cipher.init(Cipher.ENCRYPT_MODE,rc4Key);
            byte[] textEncrypted = rc4Cipher.doFinal(text);
            String s = new String(textEncrypted);
            writeTo.write(s);
            writeTo.close();
            exists = true;
            dummyField.setText("Password set successfully.");
            JButton loginButton = new JButton( text: "Go to Login →");
            loginButton.addActionListener(new buttonSetup());
            loginButton.addKeyListener(new keyPress());
            setupWindow.getRootPane().setDefaultButton(loginButton);
            gbc.insets = new Insets( top: 20,  left: 0,  bottom: 0,  right: 0);
            formPanel.add(loginButton, gbc);
        } catch (Exception exception) {
            System.out.println("Exception in password setup.");
        }
    } else {
        dummyField.setText("Please ensure that both password inputs match.");
    }
}
```

Casting of dual JTextFields to String for comparison.

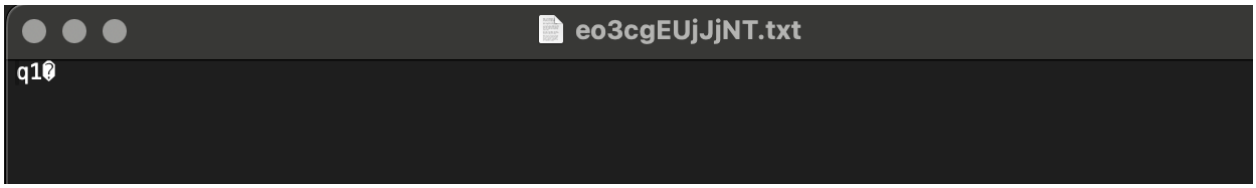
Edge case conditionals with accompanying error messages to appear in the GUI.

Calling of .txt file with encryption key. Casting key from String back into byte array into SecretKey type. Use of key to encrypt the client's input, and writing result into text file for storage.

Adding of new JButton and confirmation message in GUI to initiate next GUI.

```
private void LoginButton() {
    setupWindow.dispose();
    Login loginScreen = new Login();
}
```

Method call from button to start new class.



Random example of written password in .txt file.

Login GUI:

```
import java.io.BufferedWriter;
```

Existing as well as new import for further writing control over .txt files.

```
public Login() {
    try {
        Scanner scanner = new Scanner(file);
        password = scanner.nextLine();
    } catch (Exception exception) {
        System.out.println("Exception in password check.");
    }
```

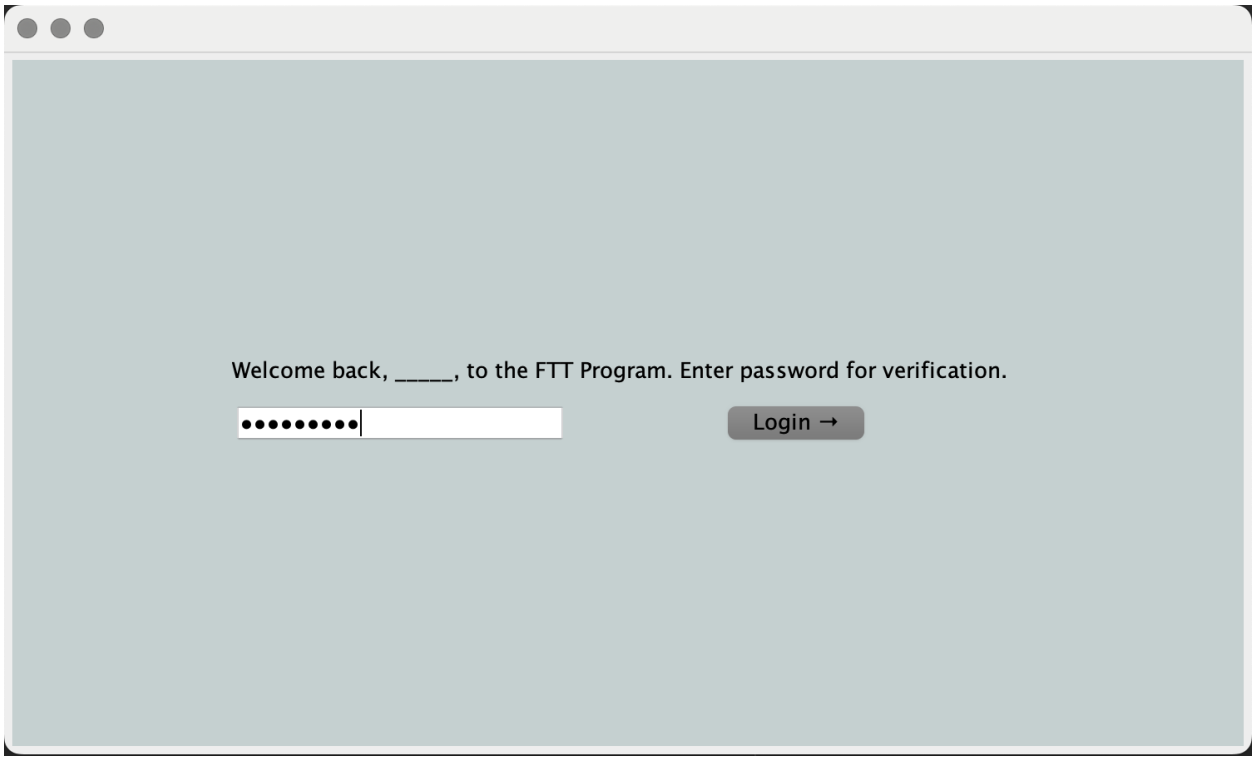
Scanner reads first line of .txt file containing the set password while still encrypted.

```
private void getKey() {
    try {
        Scanner scanner = new Scanner(keyFile);
        String temp = scanner.nextLine();
        rc4Key = convertStringToSecretKey(temp);
    } catch (Exception e) {
        System.out.println("Exception in key retrieval.");
    }
}
```

Encryption key retrieval from .txt file.

```
public static String convertSecretKeyToString(SecretKey secretKey) {
    byte[] rawData = secretKey.getEncoded();
    return Base64.getEncoder().encodeToString(rawData);
}
```

Method call to convert from SecretKey to byte array to Base64 to String.



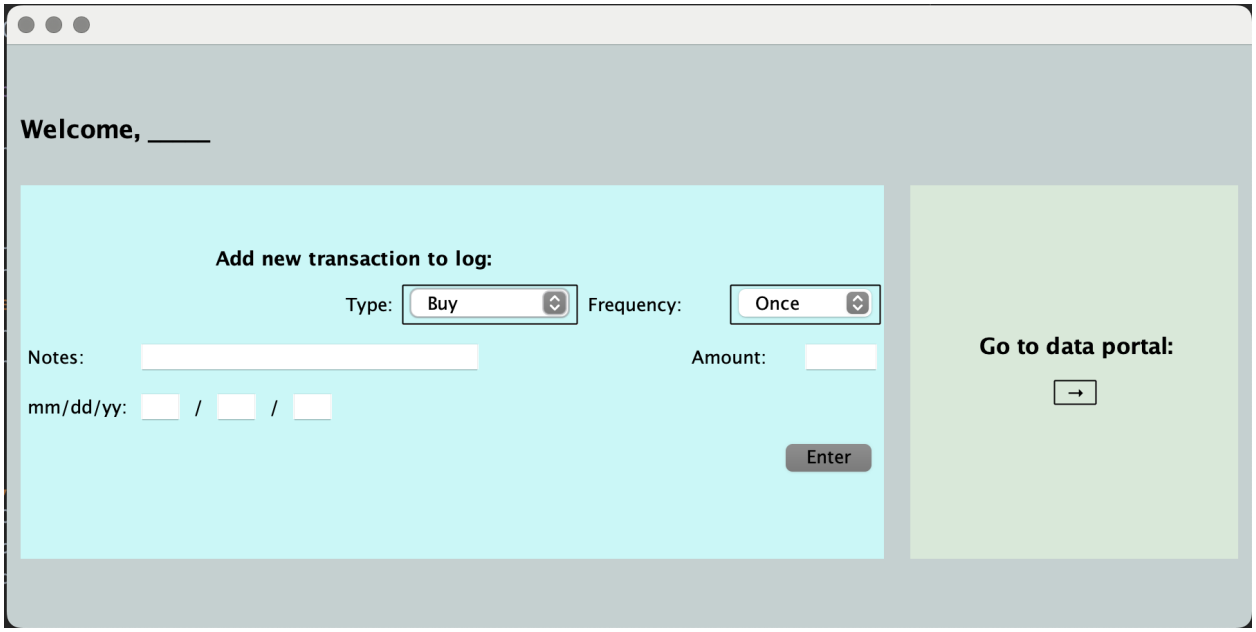
JLabel, JPasswordField, JButton

Privacy with JPasswordField and JButton with click and enter key listener.

Home/Main Page GUI:

```
private JTextField mmInput = new JTextField( columns: 2);
2 usages
private JTextField ddInput = new JTextField( columns: 2);
2 usages
private JTextField yyInput = new JTextField( columns: 2);
1 usage
String[] c1 = {"Buy", "Sell", "Deposit", "Withdrawal", "Transfer"};
3 usages
private JComboBox typeInput = new JComboBox(c1);
1 usage
String[] c2 = {"Once", "Twice", "Multiple", "Later"};
3 usages
private JComboBox freqInput = new JComboBox(c2);
2 usages
private JTextField noteInput = new JTextField( columns: 20);
2 usages
private JTextField amountInput = new JTextField( columns: 4);
4 usages
private JLabel dummyField = new JLabel();
6 usages
private JFrame homeWindow = new JFrame();
19 usages
private JPanel addEventPanel = new JPanel(new GridBagLayout());
4 usages
private JPanel goToIndexPanel = new JPanel(new GridBagLayout());
75 usages
private GridBagConstraints gbc = new GridBagConstraints();
no usages
int prevent = 0;
```

Controls for GUI. Use of Array for both JComboBoxes.



2 JPanel in-lays and JLabel header. 7 JLabels, 2 JComboboxes, 5 JTextFields, and JButton in left-side JPanel for adding transactions. JLabel and JButton in right-side JPanel to traverse to new GUI.

ActionListener and KeyListener for Enter button.

Background colors for differentiation.

JComboBoxes with multiple options in each.

```
freqInput.setBorder(new BubbleBorder(Color.BLACK, thickness: 1, radii: 1, pointerSize: 0));
```

JComboBoxes have outline through OOP border class creator

```
private class goPress implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        homeWindow.dispose();
        IndexView indexWindow = new IndexView();
    }
}
```

ActionListener for data portal button that calls next class.

```
String typeString = typeInput.getSelectedItem().toString();
String freqString = freqInput.getSelectedItem().toString();
String amountString = amountInput.getText();
String mmString = mmInput.getText();
String ddString = ddInput.getText();
String yyString = yyInput.getText();
String noteString = noteInput.getText();
```

Casting of swing types to readable Strings for comparison and checks.

```
FileWriter writeTo = new FileWriter(file,  append: true);
Cipher rc4Cipher;
rc4Cipher = Cipher.getInstance( transformation: "RC4");
String line = typeString + "|" + freqString + "|" + amountString + "|" + mmString + "|" + ddString + "|" + yyString + "|" + noteString;
byte[] text = line.getBytes( charsetName: "UTF8");
rc4Cipher.init(Cipher.ENCRYPT_MODE,rc4Key);
byte[] textEncrypted = rc4Cipher.doFinal(text);
String s = new String(textEncrypted);
buffWrite = new BufferedWriter(writeTo);
buffWrite.newLine();
buffWrite.write(s);
buffWrite.close();
```

Writing of data to .txt file after passing conditional for existence of inputs in fields. Use of BufferWrite import to write to the next line after inputting a singular event. Different events on separate lines for other classes that call the file and read the encrypted text.

```
finally {
    try {
        if (buffWrite != null) {
            buffWrite.close();
        }
    } catch (Exception e) {
        System.out.println("Error closing text file.");
    }
```

Finally catch for BufferWrite functions.

BubbleBorder Class

Adapted from K. Nie. Modified for use on specific objects within the program.

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.border.AbstractBorder;
```

New imports for GUI swing alteration.

```
public class BubbleBorder extends AbstractBorder {
```

Class extends AbstractBorder for its unique job.

```
BubbleBorder(Color color) {
    new BubbleBorder(color, thickness: 4, radii: 8, pointerSize: 7);
}
```

Call method for other classes is filled with dummy values for easy application.

```
private Color color;
3 usages
private int thickness = 4;
6 usages
private int radii = 8;
4 usages
private int pointerSize = 7;
2 usages
private Insets insets = null;
2 usages
private BasicStroke stroke = null;
9 usages
private int strokePad;
3 usages
private int pointerPad = 4;
2 usages
RenderingHints hints;
```

Adding on new BubbleBorder creates new object for the modified element to use as a border.

```
public Insets getBorderInsets(Component c) {
    return insets;
}
public Insets getBorderInsets(Component c, Insets insets) {
    return getBorderInsets(c);
}
```

Getter methods to restrict variable access to class.

Event Abstract Class

```
public Event(String type, String freq, String amount, String mm, String dd, String yy, String notes) {
    this.type = type;
    this.freq = freq;
    this.amount = amount;
    this.mm = mm;
    this.dd = dd;
    this.yy = yy;
    this.notes = notes;
    if (type.equals("Withdrawal") || type.equals("Buy")) {
        this.amount = String.valueOf(-getAmountVal());
    }
}
```

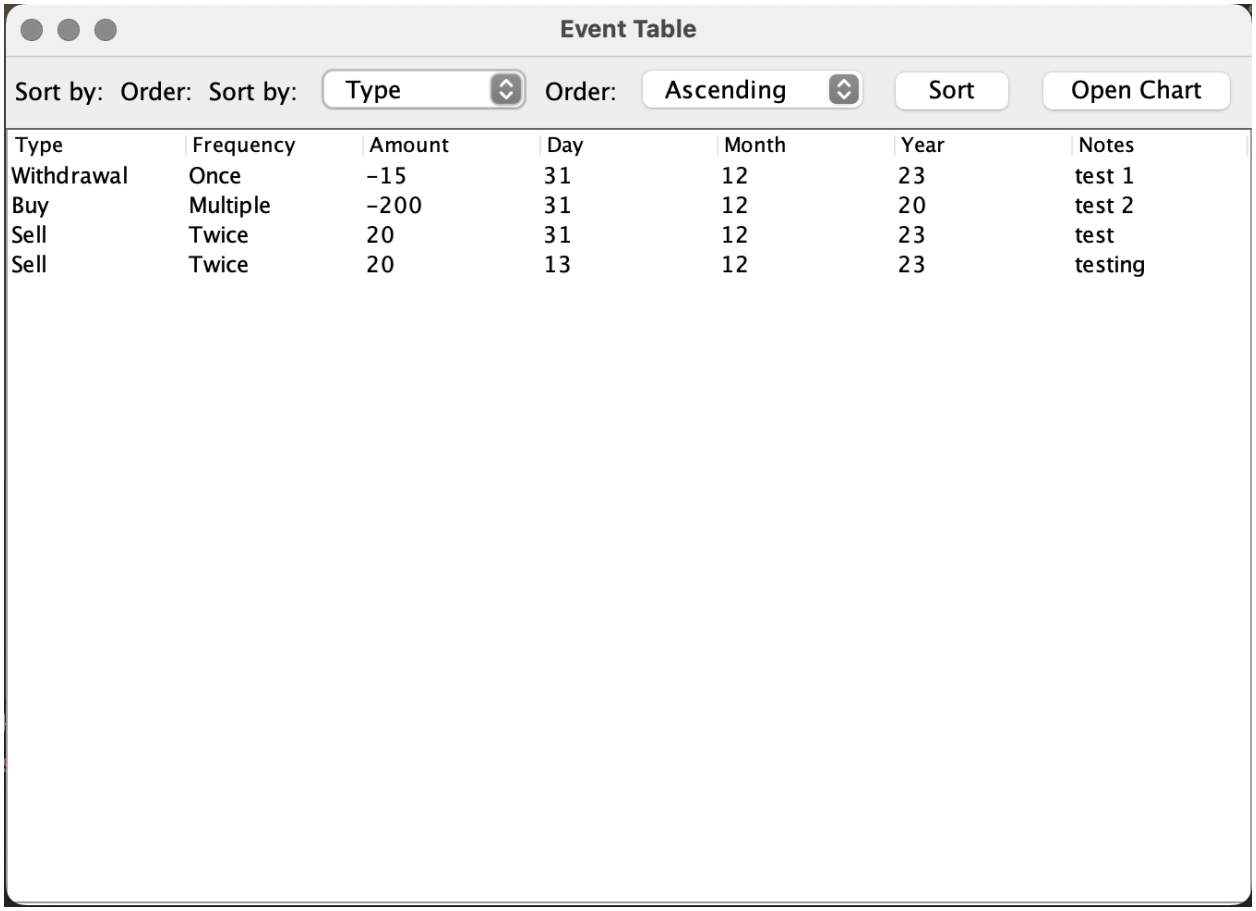
Mass constructor for creation of event objects. Event objects streamline reading the data written to files in previous classes while staying resource effective.

If conditional in constructor automatically converts amounts to negative for transactions that result in net-loss.

```
public String getType() { return type; }
3 usages
public String getFreq() { return freq; }
2 usages
public String getAmount() { return amount; }
3 usages
public int getAmountVal() { return Integer.parseInt(amount); }
3 usages
public String getDD() { return dd; }
3 usages
public String getMM() { return mm; }
3 usages
public String getYY() { return yy; }
3 usages
public String getNotes() { return notes; }
1 usage
public int dateCompile() {
    return (Integer.parseInt(yy) * 1000) + (Integer.parseInt(mm) * 100) + Integer.parseInt(dd);
}
```

Getter methods for quickly accessing specific data values. Special method for combining date values together for comparison with other dates.

IndexView Class



```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
```

Table model import for JTable component. ArrayList and List imports for access to abstract data structures. Comparator import for sorting functionality of the data structures.

```
private static List<Event> events = new ArrayList<>();
```

Initialization of ArrayList to hold event objects allows for dynamic scaling and sorting methods.

```
public IndexView() {
    readFile();
    EventTable(events);
}
```

Method readFile() fills the ArrayList with elements then method EventTable(events) uses List to populate dynamic Table

```java
private void readFile() {
    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) {
                continue;
            }
            String[] parts = line.split( regex: "\\|");
            String tempType = parts[0];
            String tempFreq = parts[1];
            String tempAmount = parts[2];
            String tempDay = parts[3];
            String tempMonth = parts[4];
            String tempYear = parts[5];
            String tempNotes = parts[6];
            Event event = new Event(tempType, tempFreq, tempAmount, tempDay, tempMonth, tempYear, tempNotes);
            events.add(event);
```

Method reads all lines. Condition checks for data in line then splits line if data is present. An array consisting of data held between “|” then intializes a new object with each value filled is created. The entire object is then stored in the ArrayList once created.

```java
DefaultTableModel tableModel = new DefaultTableModel(new Object[]{"Type", "Frequency", "Amount", "Day", "Month", "Year", "Notes"},
for (Event event : events) {
    tableModel.addRow(new Object[]{event.getType(), event.getFreq(), event.getAmountVal(), event.getDD(), event.getMM(), event.getY
```

Table creator method intializes table with column titles then loops through entire contents of ArrayList. All objects in list then provide data for a new row of data in the table.

```java
JComboBox<String> columnComboBox = new JComboBox<>(columns);
String[] orders = {"Ascending", "Descending"};
JComboBox<String> orderComboBox = new JComboBox<>(orders);
JButton sortButton = new JButton( text: "Sort");
sortButton.addActionListener(e -> sortTable(jTable, columnComboBox.getSelectedItem().toString(), orderComboBox.getSelectedItem().toString()));
JButton openChartButton = new JButton( text: "Open Chart");
openChartButton.addActionListener(e -> openChartFrame());
```

JPanel features 2 JComboBoxes and 2 JButtons. JComboBoxes provide option to sort specific column by high to low and vice versa. JButton calls the sort method that intializes sorting process of the column then changes table. Other JButton opens new GUI class with chart visualization.

```java
private void sortTable(JTable table, String columnName, String sortOrder) {
    Comparator<Event> comparator = Comparator.comparing(e -> {
        switch (columnName) {
            case "Type":
                return e.getType();
            case "Frequency":
                return e.getFreq();
            case "Amount":
                return e.getAmount();
            case "Day":
                return e.getDD();
            case "Month":
                return e.getMM();
            case "Year":
                return e.getYY();
            case "Notes":
                return e.getNotes();
            default:
                return "";
        }
    });
```

Sort method takes client's input from JComboBox and matches with column.

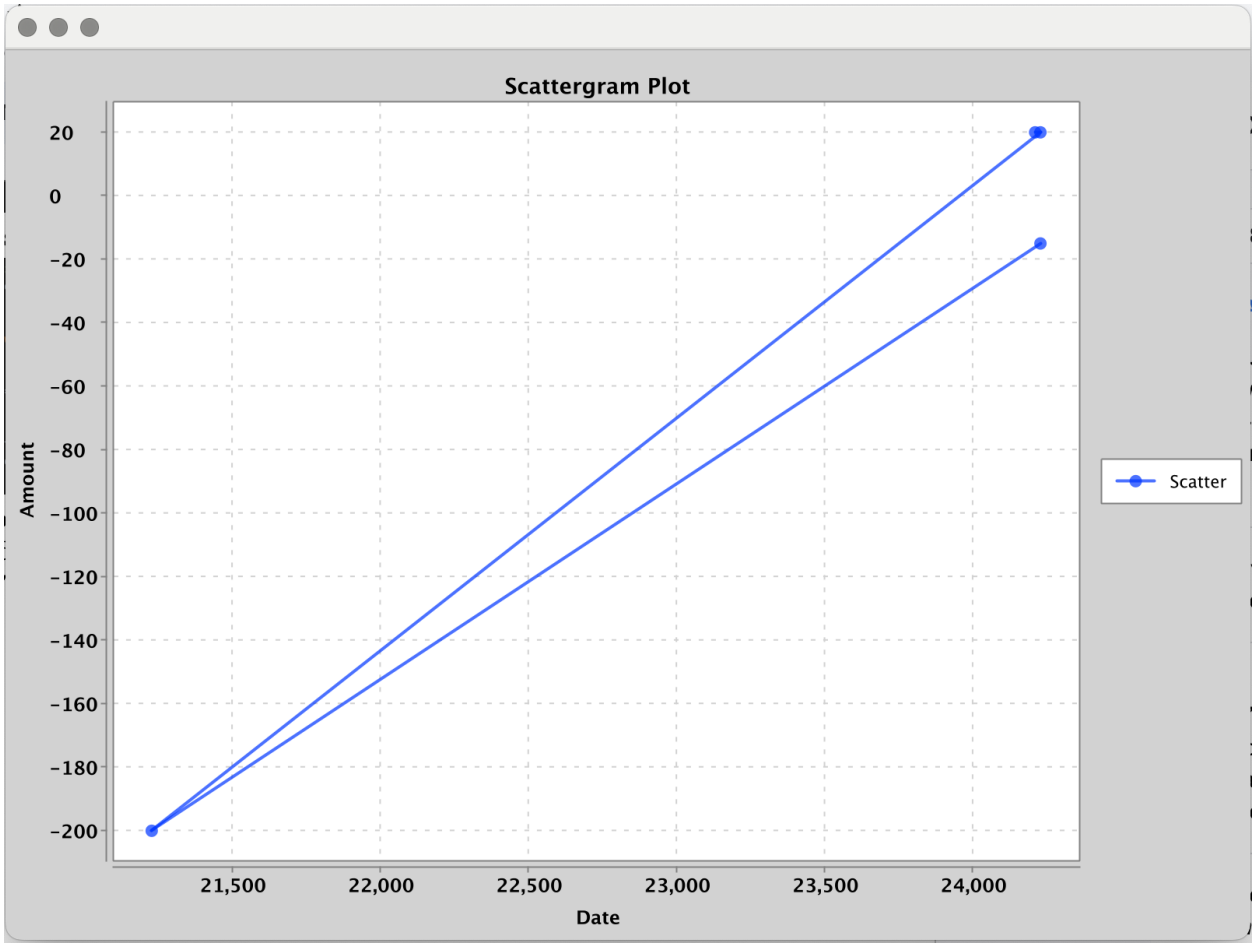
```java
if (sortOrder.equals("Descending")) {
    comparator = comparator.reversed();
}

List<Event> sortedEvents = new ArrayList<>(events);
sortedEvents.sort(comparator);

DefaultTableModel tableModel = (DefaultTableModel) table.getModel();
tableModel.setRowCount(0);

for (Event event : sortedEvents) {
    tableModel.addRow(new Object[]{event.getType(), event.getFreq(), event.getAmount(), event.getDD(), event.getMM(), event.getYY(), event.getNotes()});
}
```

Use of comparator utility keeps sort process simple for large compile cases. Comparator defaults to ascending order, so it is reversed for descending. Table resets and then for loop cycles through ArrayList of sorted values and repopulates table in sorted order.

```java
private void openChartFrame() {
    dispose();
    ScattergramChartFrame chartView = new ScattergramChartFrame(events);
}
```

Chart button closes index GUI then intializes Chart Class with events ArrayList threaded through.

ScattergramChartFrame Class




```
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XChartPanel;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.style.markers.SeriesMarkers;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
```

Class uses .jar library for custom chart.

```
private XChartPanel<XYChart> createScattergramChart() {
    List<Integer> dates = new ArrayList<>();
    List<Integer> amounts = new ArrayList<>();
    for (Event event : events) {
        dates.add(event.dateCompile());
        amounts.add(event.getAmountVal());
    }
    XYChart chart = new XYChartBuilder().width(800).height(600).title("Scattergram Plot").xAxisTitle("Date").yAxisTitle("Amount").build();
    chart.addSeries( seriesName: "Scatter", dates, amounts).setMarker(SeriesMarkers.CIRCLE);
    return new XChartPanel<>(chart);
```

Chart creation method calls amount and date value of each object in element list. Date is assigned to X-Axis and amount to Y-Axis in style of a scattergram.

Bibliography

Baeldung. (2023). *Baeldung*. “Encrypting and Decrypting Files in Java.” Available from: https://www.baeldung.com/java-cipher-input-output-stream. [Accessed 18 November 2023].

Baeldung. (2021). *Baeldung*. “Secret Key and String Conversion in Java.” Available from: https://www.baeldung.com/java-secret-key-to-string. [Accessed 20 November 2023].

(n.d.). *HTML Color Codes*. “Color Picker — HTML Color Codes.” Available from: https://htmlcolorcodes.com/color-picker/. [Accessed 21 November 2023].

Oracle. (n.d.). *Oracle Help Center*. “How to Use GridBagLayout (The Java™ Tutorials > Creating a GUI With Swing > Laying Out Components Within a Container).” Available from: https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html. [Accessed 18 November 2023]

Jain, S. (2023). *GeeksforGeeks*. “Searching in Binary Search Tree (BST).” Available from: https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/. [Accessed 1 December 2023].

Nie, K. (2023). *Bubble Border Class*. [Accessed 23 November 2023].

Oracle. (n.d.). *Oracle Help Center*. “BufferedWriter (Java Platform SE 8).” Available from: https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html. [Accessed 20 November 2023].

Knowm. (2023). *Knowm Inc.* “Overview (XChart Parent 3.8.6 API).” Available from: https://knowm.org/javadocs/xchart/index.html. [Accessed 2 December 2023].