Criterion C: Development

1. Introduction

The product is a Java program coded on Netbeans. Netbeans was selected because graphical user interface for the application can be made easily with Java's swing tools. The application is an investor transactions tracker, which tracks all stock exchange transactions that user enters. Netbeans organizes classes nicely and allows programmer to set the interaction between user and application conveniently.

Word Count: 59

2. Summary List of All Techniques

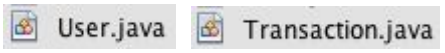
- For loops with i++, and i--
- Nested loops
- Methods returning a value, Methods taking in parameters
- Arrays, ArrayLists [of Strings, of Objects]
- User defined objects made from an OOP "template" class
- Encapsulation of private attributes with accessors and modifier methods
- Simple and compound selection (if/else)
- Sorting, and in particular sorting ArrayLists of objects based on one key attribute
- Saving to a file
- Retrieving data from file, turning them into ArrayLists
- Error handling (outlined in criterion A under functions and B under testing plan)
- GUI tabs
- Use of a flag value (such as -999, or "not set yet")
- Parsing a file using StringTokenizer
- Inheritance between a superclass and a subclass
- SpecificArrayList.add(), SpecificArrayList.remove()
- Loop to move all elements of ArrayList down one index after removing value
- Use of ||, &&, ^, !, .equals() on conditionals
- Use of global variables that are manipulated accordingly in each method in order to determine when user can call certain methods
- Adding elements into combobox while program is running
- Coded user login, create account
- Parsing values into appropriate data type.
    - Parsing ints to doubles, doubles/ints to string
- Use of GUI features: Textfield.setText(), ComboBox.getSelectedItem(), etc.
- Printing out data on JTable using for loop (with conditionals(if/else))
- Math.round()

Word Count: 0 (Bulleted List)

3. Structure of the Program

The main class which is ran when when the user uses the program is a graphical user interface class, or GUI class. GUI allows users to interact with the program, input data and view outputs such as the different variables of each previously entered transactions in a neat and organized way.

A User and Transaction object template classes was also made in order store users and transactions data in an organized way. With these template classes, ArrayLists of them can be made and each variable can be stored within each instance of a User or Transaction and the objects together in an ArrayList. The information can then be access through arrayList.TemplateClass.getVariable(), such as users.User.getUserName(0). **Aggregation** is applied as users have transactions with a 'has a' or ownership relationship between the two classes.



Three of the remaining classes are for searching transaction class and saving and retrieving users information and transactions. The first class is for sorting and searching transactions with 8 methods to sort transaction according to 4 variables from min to max and max to min. Reading and writing transactions with 2 methods, for saving and retrieving transactions and a class for reading and writing users' information which has 2 methods for reading and writing usernames and passwords and 2 methods for reading and writing stock names list for each user.

With classes, and object oriented programming, properties of encapsulation, use of constructors and abstractions are used. With encapsulation, attributes are private and can only be accessed and changed when appropriate accessor and modifier methods are called. Abstraction allowed me to work more efficiently with testing, coding and identifying problems by focusing on one problem at a time.

Word Count: 294

4. Data Structures Used

1. ArrayList

```
private ArrayList<User> users = new ArrayList<User>();
private ArrayList<Stock> stocksList = new ArrayList<Stock>();
private ArrayList<String> stocksNameList = new ArrayList<String>();
private ArrayList<Transaction> transactions = new ArrayList<Transaction>();
```

ArrayLists were used to store elements of objects and strings as ArrayLists are dynamic, thus its size changes according to how many elements it contains, which saves memory. ArrayLists also share some properties with arrays, and can be searched and sorted like arrays. In this case, ArrayLists of transactions are made and can be sorted according to specific variables.

2. Files

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.StringTokenizer;
```

```
public static void savingTransactions(String username, ArrayList<Transaction> transactions) {
    try {
        String fileName = username + " Transactions.txt";
        FileWriter fw = new FileWriter(fileName);
        for (int i = 0; i < transactions.size(); i++) {
            fw.write(transactions.get(i).getStockName());
            fw.write(":");
            fw.write(transactions.get(i).getDate());
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getNumberOfShares()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getDividends()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getPrice()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getBought()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getTimeStamp()));
            fw.write(":");
```

The File class was used to save/store data of attributes and can be accessed after program stops running and restarts again. This is done through the use of the FileWriter and FileReader classes.

Word Count: 95

5. Main Unique Algorithms

1. User Log In

```
private void returningUserLoginTFMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    boolean usernameExists = false;
    for (int i = 0; i < users.size(); i++) {
        if (returningUsernameTF.getText().equals(users.get(i).getUserName())) {
            usernameExists = true;
            usingUserNumber = i;
            i = users.size();
        }
    }
    if (usernameExists == false) {
        returningUsernameTF.setText("Error");
        loginErrorTF.setText("Invalid Username, username does not exist.");
    } else if (usernameExists
            && returningPasswordTF.getText().equals(users.get(usingUserNumber).getPassword()) == false) {
        returningPasswordTF.setText("Error");
        loginErrorTF.setText("Incorrect Password");

    } else if (usernameExists && returningPasswordTF.getText().equals(users.get(usingUserNumber).getPassword())) {
        loggedIn = true;
        transactions = readingTransactionsFile(users.get(usingUserNumber).getUserName());
        stocksNameList = readingStocksNameListFile(users.get(usingUserNumber).getUserName());
        returningUsernameTF.setText("Logged In");
        returningPasswordTF.setText("Logged In");
        loginErrorTF.setText("Logged In");

        //create new String array with new stock name
        String[] stocksNameListForCB = new String[stocksNameList.size() + 1];
        stocksNameListForCB[0] = "";
        for (int i = 0; i < stocksNameList.size(); i++) {
            stocksNameListForCB[i + 1] = stocksNameList.get(i);
        }

        //display combo box with new stock added
        existingStockNameCB.setModel(new javax.swing.DefaultComboBoxModel<>(stocksNameListForCB));

    } else if (usernameExists && returningPasswordTF.getText() != (users.get(usingUserNumber).getPassword())) {
        returningPasswordTF.setText("Error");
        loginErrorTF.setText("Incorrect password, please try again.");
    }
}
```

Every time a user logs in, read transaction and read user information files methods are called to retrieve the appropriate data of the user input. The stocks for the combo box is retrieved as an ArrayList, put into an array, and entered as parameter into combobox model, as combo boxes can only take in String arrays. Turns boolean loggedIn true in order to access functions in application.

2. Entering transactions and all associated error handling

```
private void enterTransactionButtonMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    if (loggedIn) {
        String date = String.valueOf(dateDayTransactionCB.getSelectedItem())
                + dateMonthTransactionCB.getSelectedItem()
                + dateYearTransactionCB.getSelectedItem();

        double averagePriceToDate = -999.99;
        int totalSharesOwned = -99;
        double profit = -99.99;
        double netProfit = -99.99;
        boolean canSell = true;
        //These values are obscure values, for testing purposes, so when these numbers show or obscure values are
        // displayed, it is known that code has semantics error

        //If one of the transaction data is not entered
        if ((newStockNameTransactionTF.getText().equals("") && existingStockNameCB.getSelectedItem().equals(""))
                || enterNumberOfSharesTF.getText().equals("")
                || enterStockDividendsTF.getText().equals("")
                || enterStockPriceTF.getText().equals("")
                || dateDayTransactionCB.getSelectedItem().equals(" ")
                || dateMonthTransactionCB.getSelectedItem().equals(" ")
                || dateYearTransactionCB.getSelectedItem().equals(" ")) {
            transactionErrorTF.setText("Please enter all transaction information");
```

^ All attributes are defined and calculated within method before being entered into Transaction constructor, as a form of abstraction. Each attributes are assigned obscure values that would be instantly noticeable in program testing with semantic error. If any of the information textfield is blank, an error will occur.

```
        } else {
            for (int i = transactions.size() - 1; i > -2; i--) {
                if (i == -1) {
                    totalSharesOwned = (Integer.parseInt(enterNumberOfSharesTF.getText() + ""));
                    averagePriceToDate = (Double.parseDouble(enterStockPriceTF.getText() + ""));
                    profit = 0.00;
                    netProfit = 0.00;
                } //
                else if ((transactions.get(i).getStockName()).equals(newStockNameTransactionTF.getText())
                        || (transactions.get(i).getStockName()).equals
                                (String.valueOf(existingStockNameCB.getSelectedItem()))) {
                    if (buyTransactionCheckBox.isSelected() == true) {
                        totalSharesOwned = ((transactions.get(i)).getTotalSharesOwnedToDate())
                                + (Integer.parseInt(enterNumberOfSharesTF.getText() + ""));

                        averagePriceToDate = ((((transactions.get(i)).getAveragePriceToDate())
                                * ((transactions.get(i)).getTotalSharesOwnedToDate()))
                                + ((Integer.parseInt(enterNumberOfSharesTF.getText() + ""))
                                * (Double.parseDouble(enterStockPriceTF.getText() + ""))))
                                / totalSharesOwned;

                        profit = 0;
                        netProfit = transactions.get(i).getProfitToDate();
```

```
                    } else if (sellTransactionCheckBox.isSelected() == true) {

                        //check if user has enough shares to sell
                        totalSharesOwned = ((transactions.get(i)).getTotalSharesOwnedToDate());
                        if (totalSharesOwned < (Integer.parseInt(enterNumberOfSharesTF.getText() + ""))) {
                            canSell = false;
                        }

                        //calculates values for new transaction to be added
                        totalSharesOwned = ((transactions.get(i)).getTotalSharesOwnedToDate())
                                - (Integer.parseInt(enterNumberOfSharesTF.getText() + ""));

                        if (totalSharesOwned == 0) {
                            averagePriceToDate = 0;
                        } else {
                            averagePriceToDate = ((((transactions.get(i)).getAveragePriceToDate())
                                    * ((transactions.get(i)).getTotalSharesOwnedToDate()))
                                    - ((Integer.parseInt(enterNumberOfSharesTF.getText() + ""))
                                    * (Double.parseDouble(enterStockPriceTF.getText() + ""))))
                                    / totalSharesOwned;
                        }
```

^ Buy and selling will result in different calculation and assignment of values for the attributes in Transaction constructor, thus conditional/compound selection is applied to see whether user has entered buy or sell transaction.

```
                double moneySpent = transactions.get(i).getAveragePriceToDate()
                        * (Integer.parseInt(enterNumberOfSharesTF.getText() + ""));
                double moneyReceive = (Integer.parseInt(enterNumberOfSharesTF.getText() + ""))
                        * (Double.parseDouble(enterStockPriceTF.getText() + ""));
                profit = moneyReceive - moneySpent;

                netProfit = transactions.get(i).getProfitToDate() + profit;

                //if user tries to sell more stocks than owned
                if ((Integer.parseInt(enterNumberOfSharesTF.getText() + ""))
                        > ((transactions.get(i)).getTotalSharesOwnedToDate())) {
                    transactionErrorTF.setText("Selling more stocks than owned");
                }

            }
            i = -1;
        } else {
        }
    }
```

Error handling above and below for every different case.

```
    //Error Handling of Enter Transaction Tab Starts Here
    if ((buyTransactionCheckBox.isSelected() == true && sellTransactionCheckBox.isSelected() == true)
            || (buyTransactionCheckBox.isSelected() == false && sellTransactionCheckBox.isSelected() == false))
        transactionErrorTF.setText("Please select either Buy or Sell");
    } //Error Handling of Enter Transaction Tab Starts Here
    else if (buyTransactionCheckBox.isSelected() == true ^ sellTransactionCheckBox.isSelected() == true) {
        //Error handling of new/existing stock transaction
        if ((existingStockButton.isSelected() == true && newStockButton.isSelected() == true)) {
            transactionErrorTF.setText("Please select if stock is new or existing");
        } else if ((existingStockButton.isSelected() == true && newStockButton.isSelected() == true)
                || (existingStockButton.isSelected() == false && newStockButton.isSelected() == false)) {
            transactionErrorTF.setText("Please check either 'existing stock' or 'new stock' check box");
        } else if (canSell == false) {
            transactionErrorTF.setText("Selling more shares than owned");
        } //existing stock or new stock transaction
        else if (newStockButton.isSelected() == true && existingStockButton.isSelected() == false) {
            transactions.add(new Transaction(
                    newStockNameTransactionTF.getText(),
                    date,
                    Integer.parseInt(enterNumberOfSharesTF.getText() + ""),
                    Double.parseDouble(enterStockDividendsTF.getText() + ""),
                    Double.parseDouble(enterStockPriceTF.getText() + ""),
                    buyTransactionCheckBox.isSelected(),
                    System.currentTimeMillis(),
                    averagePriceToDate,
                    totalSharesOwned,
                    profit,
                    netProfit
            ));
            transactionErrorTF.setText("Transaction entry successful");
```

```
            //add new stock name to combo box
            //error handling for if new stock name already exists in Combo Box
            //and entered in new stock textfield
            boolean stockExists = false;
            for (int i = 0; i < stocksNameList.size(); i++) {
                if (newStockNameTransactionTF.getText().equals(stocksNameList.get(i))) {
                    stockExists = true;
                    i = stocksNameList.size();
                }
            }
            if (stockExists == false) {
                stocksNameList.add(transactions.get(transactions.size() - 1).getStockName());
            }

            //create new String array with new stock name
            String[] stocksNameListForCB = new String[stocksNameList.size()];
            stocksNameListForCB[0] = "";
            for (int i = 1; i < stocksNameList.size(); i++) {
                stocksNameListForCB[i] = stocksNameList.get(i);
            }

            //display combo box with new stock added
            existingStockNameCB.setModel(new javax.swing.DefaultComboBoxModel<>(stocksNameListForCB));
```

^ Compound selection/conditional for if new stock checkbox is checked, but old stock name is entered, the stock name will not be added to combobox stockNamesList ArrayList. If new stock is entered and stock name does not exist, it is added to stocksNameList and combobox array.

(Below) Resetting all information fillers to default (empty)

```
                //display combo box with new stock added
                existingStockNameCB.setModel(new javax.swing.DefaultComboBoxModel<>(stocksNameListForCB));

                // Clear all input textfields, check boxes, combo boxes to orginal state.
                newStockNameTransactionTF.setText("");
                dateDayTransactionCB.setSelectedItem(" ");
                dateMonthTransactionCB.setSelectedItem(" ");
                dateYearTransactionCB.setSelectedItem(" ");
                newStockButton.setSelected(false);
                existingStockButton.setSelected(false);
                enterNumberOfSharesTF.setText("");
                enterStockDividendsTF.setText("");
                enterStockPriceTF.setText("");
                buyTransactionCheckBox.setSelected(false);
                sellTransactionCheckBox.setSelected(false);
                existingStockNameCB.setSelectedItem("");

            } else if (existingStockButton.isSelected() == true && newStockButton.isSelected() == false) {
                transactions.add(new Transaction(
                        String.valueOf(existingStockNameCB.getSelectedItem()),
                        date,
                        Integer.parseInt(enterNumberOfSharesTF.getText() + ""),
                        Double.parseDouble(enterStockDividendsTF.getText() + ""),
                        Double.parseDouble(enterStockPriceTF.getText() + ""),
                        buyTransactionCheckBox.isSelected(),
                        System.currentTimeMillis(),
                        averagePriceToDate,
                        totalSharesOwned,
                        profit,
                        netProfit
                ));
```

^ Adding new transaction with all calculated attributes with parsing of textfield entries.

```
                transactionErrorTF.setText("Transaction entry successful");
                // Clear all input textfields, check boxes, combo boxes to orginal state.
                newStockNameTransactionTF.setText("");
                dateDayTransactionCB.setSelectedItem(" ");
                dateMonthTransactionCB.setSelectedItem(" ");
                dateYearTransactionCB.setSelectedItem(" ");
                newStockButton.setSelected(false);
                existingStockButton.setSelected(false);
                enterNumberOfSharesTF.setText("");
                enterStockDividendsTF.setText("");
                enterStockPriceTF.setText("");
                buyTransactionCheckBox.setSelected(false);
                sellTransactionCheckBox.setSelected(false);
                existingStockNameCB.setSelectedItem("");
            }
        }
    }
} else {
    //***
    transactionErrorTF.setText("Login Required");
}
}
```

3. Displaying transactions on transaction table (after applying sorting methods)

(Below) transactions are sorted appropriately (screenshot of sorting methods called not attached, see full code for reference)

```
int N = transactions.size();
ArrayList<Transaction> transactionsDisplay = new ArrayList<Transaction>();

if (stockToDisplayTransactionsTableTF.getText().equals("All")) {
    transactionsDisplay = transactions;
} else {
    for (int i = 0; i < N; i++) {
        if ((transactions.get(i).getStockName()).equals(stockToDisplayTransactionsTableTF.getText())) {
            transactionsDisplay.add(transactions.get(i));
        }
    }
}
```

^ New ArrayList made so transactions displayed list can be manipulated without changing ArrayList with all transactions. Conditionals to add stocks with specific name when specific stock name is entered into textfield on transactions table tab.

```
if (transactionsDisplay.size() <= transactionsTable.getRowCount()) {

    for (int row = 0; row < transactionsDisplay.size(); row++) {
        if (transactionsDisplay.get(row).getStockName().equals("preventNull")) {

        } else {
            transactionsTable.setValueAt(transactionsDisplay.get(row).getStockName(), row, 0);
            transactionsTable.setValueAt(transactionsDisplay.get(row).getDate(), row, 1);
            transactionsTable.setValueAt(transactionsDisplay.get(row).getNumberOfShares(), row, 2);
            transactionsTable.setValueAt(transactionsDisplay.get(row).getDividends(), row, 3);
            transactionsTable.setValueAt(transactionsDisplay.get(row).getPrice(), row, 4);
            if (transactionsDisplay.get(row).getBought() == true) {
                transactionsTable.setValueAt("Bought", row, 5);
            } else {
                transactionsTable.setValueAt("Sold", row, 5);
            }
            if (transactionsDisplay.get(row).getTotalSharesOwnedToDate() == 0) {
                transactionsTable.setValueAt("n/a", row, 6);
            } else {
                transactionsTable.setValueAt(String.valueOf(transactionsDisplay.get(row).getAveragePriceToDate()), row, 6);
            }
            transactionsTable.setValueAt(transactionsDisplay.get(row).getTotalSharesOwnedToDate(), row, 7);
            if (transactionsDisplay.get(row).getBought() == true) {
                transactionsTable.setValueAt("n/a", row, 8);
            } else {
                transactionsTable.setValueAt(String.valueOf((transactionsDisplay.get(row).getTransactionProfit())),
                        row, 8);
            }
            transactionsTable.setValueAt(transactionsDisplay.get(row).getProfitToDate(), row, 9);
        }
    }

}
```

^ Displaying transactions attributes using for loop to go through all boxes on transaction table.

4. Sorting

```
public void selectionSortOfTransactionsTime(ArrayList<Transaction> transactions) {
    for (int i = 0; i < transactions.size() - 1; i++) {
        int minIndex = i;      // Assumed index of smallest remaining value.
        for (int j = i + 1; j < transactions.size(); j++) {
            if (transactions.get(j).getTimeStamp() < transactions.get(minIndex).getTimeStamp()) {
                minIndex = j;  // Remember index of new minimum
            }
        }
        if (minIndex != i) {
            //Exchange current element with smallest remaining.
            //But note that this only happens once each outer loop iteration, at the end of the inner loop's looping
            Transaction temp = transactions.get(i);
            transactions.set(i, transactions.get(minIndex));
            transactions.set(minIndex, temp);
        }
    }
}
```

*For max to min, change minIndex to maxIndex and change ‘<’ to ‘>’
Code based off of from johnrayworth.info selection sort code.

5. File Writing

```
public static void savingTransactions(String username, ArrayList<Transaction> transactions) {
    try {
        String fileName = username + " Transactions.txt";
        FileWriter fw = new FileWriter(fileName);
        for (int i = 0; i < transactions.size(); i++) {
            fw.write(transactions.get(i).getStockName());
            fw.write(":");
            fw.write(transactions.get(i).getDate());
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getNumberOfShares()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getDividends()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getPrice()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getBought()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getTimeStamp()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getAveragePriceToDate()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getTotalSharesOwnedToDate()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getTransactionProfit()));
            fw.write(":");
            fw.write(String.valueOf(transactions.get(i).getProfitToDate()));
            fw.write(":");
        }
        fw.close();
    } catch (IOException ex) {

    }
}
```

6. File Reading

```
public static ArrayList<String> readingStocksNameListFile(String username) {
    ArrayList<String> stocksNameList = new ArrayList<String>();
    try {
        FileReader fr = new FileReader(username + " StocksNameList.txt");
        BufferedReader br = new BufferedReader(fr);
        String readInFile = br.readLine();
        StringTokenizer st = new StringTokenizer(readInFile, ":");
        while (st.hasMoreTokens()) {
            stocksNameList.add(st.nextToken());
        }
    } catch (IOException ex) {
        stocksNameList.add("");
    }
    return stocksNameList;
}
```

Both file reading and writing code manipulated based off of John Rayworth's work. "Full File Writing and Reading Project" on Youtube.
Link: https://www.youtube.com/watch?v=3bcl246jlSg&feature=youtu.be

File reading and writing used so user can save data and retrieve information after program has been relaunched. StringTokenizer class used to separate data on textedit file. Since program does not work with null ArrayLists, first element is added as empty string "" to prevent null ArrayList.

7. Deleting Account

```
private void deleteAccountButtonMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    boolean usernameExists = false;
    for (int i = 0; i < users.size(); i++) {
        if (returningUsernameTF.getText().equals(users.get(i).getUserName())) {
            usernameExists = true;
            usingUserNumber = i;
            i = users.size();
        }
    }
    if (usernameExists == false) {
        returningUsernameTF.setText("Error");
        loginErrorTF.setText("Invalid Username, username does not exist.");
    } else if (loggedIn == false) {
        returningUsernameTF.setText("Error");
        loginErrorTF.setText("Please login in before deleting account.");
    } else if (returningUsernameTF.equals("") || returningPasswordTF.equals("")) {
        returningUsernameTF.setText("Error");
        loginErrorTF.setText("Please enter username and password of account to delete");
```

^ Error handling, making sure user wants to delete account with layers of user interaction before account deletion.

```
    } else if (usernameExists && returningPasswordTF.getText().equals(users.get(usingUserNumber).getPassword())) {
        //Account deleted means no user is logged in, relogin required.
        loggedIn = false;
        //needs to be done before user is removed, or username will permanently be taken by unempty file
        for (int i = transactions.size() - 1; i > 0; i--) {
            transactions.remove(i);
        }
        savingTransactions(users.get(usingUserNumber).getUserName(), transactions);
        for (int i = stocksNameList.size() - 1; i > 0; i--) {
            stocksNameList.remove(i);
        }
        savingStocksNameList(users.get(usingUserNumber).getUserName(), stocksNameList);
        //removing users and moving later users down an index
        users.remove(usingUserNumber);
        for (int i = usingUserNumber; i < users.size() - 1; i++) {
            users.add(i, users.get(i + 1));
            if (i == users.size() - 2) {
                users.remove(users.size() - 1);
            }
        }
        savingUsersList(users);
        //deleting transactions
        for (int i = 0; i < transactions.size(); i++) {
            transactions.remove(transactions.get(i));
        }
        //deleting stocks Name List
        for (int i = 0; i < stocksNameList.size(); i++) {
            stocksNameList.remove(stocksNameList.get(i));
        }
        returningUsernameTF.setText("Account");
        returningPasswordTF.setText("Deleted");
    } else if (usernameExists && returningPasswordTF.getText() != (users.get(usingUserNumber).getPassword())) {
        returningPasswordTF.setText("Error");
        loginErrorTF.setText("Incorrect password, please try again.");
    }
}
```

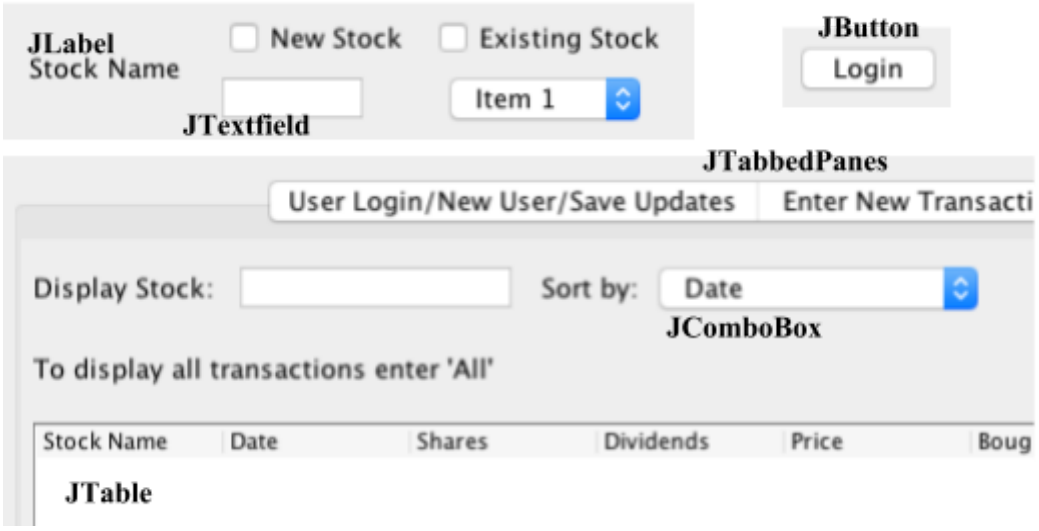
After account is deleted, in the users ArrayList, all the elements greater than the index of the deleted account must be moved down one to close the empty gap and save memory (the purpose of using ArrayList in the first place).

Saving/reading methods must be implemented right away as the files will not be deleted, but must make sure that those files are empty so if new user wants to use the username and thus save their data in those existing files with old file names, they can without carrying over old transactions of deleted account user.

Word Count: 452

6. User Interface/GUI Work

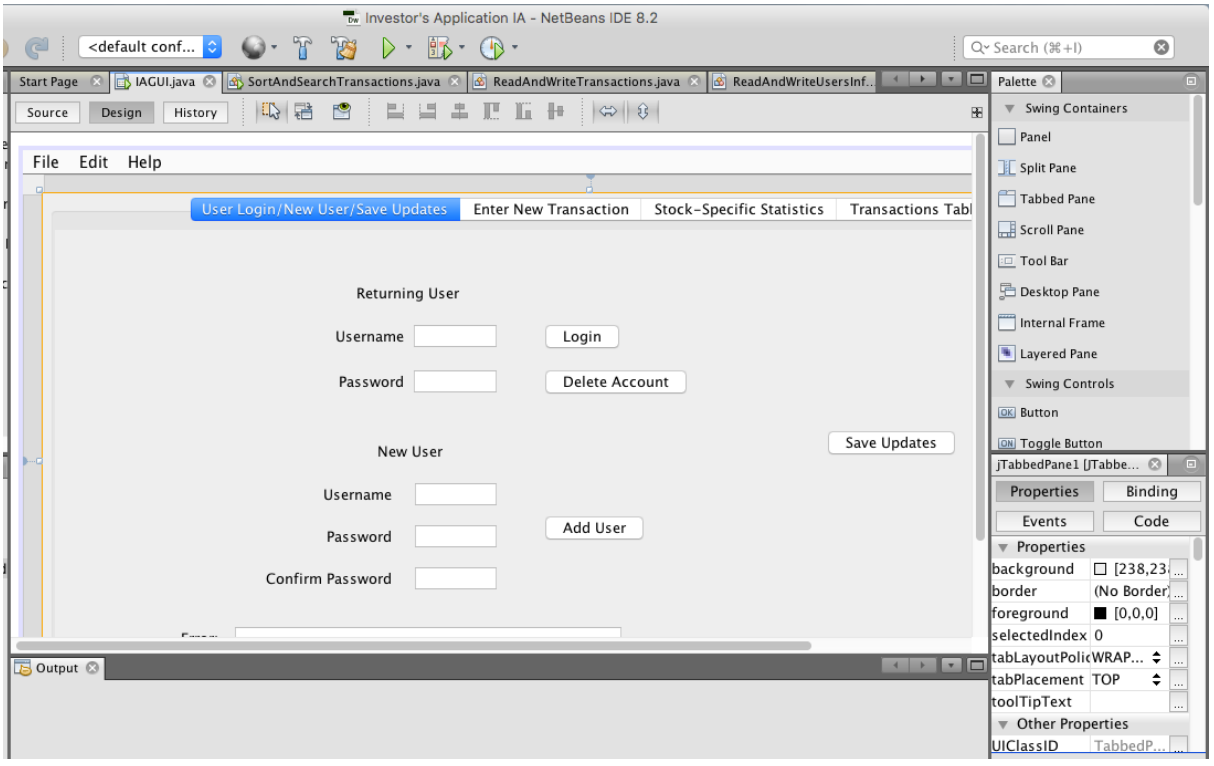
- Use of GUI: For users to be able to interact with program in a visual user friendly setting.
- GUI components used:
    - JTextfields: For user to enter information
    - JCombobox: For users to enter information from specific list.
    - JButtons: For users to click when entering information.
    - JLabels: Notify users what each GUI component is used for.
    - JTabbedPane: Organizes user interface (makes it more user friendly) and separate each function or similar types/related functions into their individual pages
    - JCheckbox: Easy way for user to assign true/false for boolean in program code.
    - JTable: Used to display large amounts of data in an organized way.

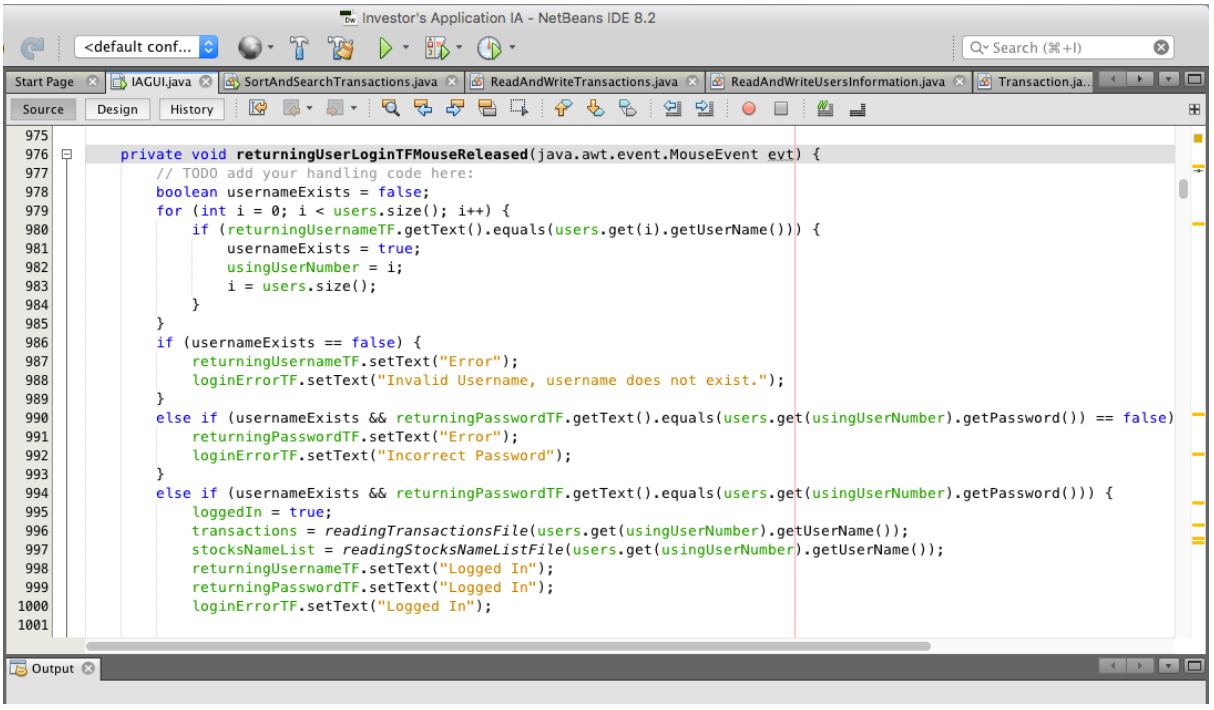


Word Count: 0 (Bulleted List)

7. Software Tools Used

NetBeans, an integrated developmental program for java, was used in the development and coding of the application. NetBeans software was chosen as it has user friendly GUI interface and prewritten code libraries and can be simply used for object oriented programming, which is preferred for the development of this application.





Word Count: 50 Full document Word Count: 950 Criterion A+B+C: 1596