Criteria C - Product Development

(1) While Loops Setup for Entire Menu (Algorithmic Complexity)

(2) Use of .csv Files for Data Storage (System Design & Algorithmic Complexity)

(3) Iteration (loops) used to make calculations through Array (Algorithmic Complexity)

(4) Breakdown of code into separate methods (Organization)

(5) User Interface (Command Line) Simplified Look (Organization)

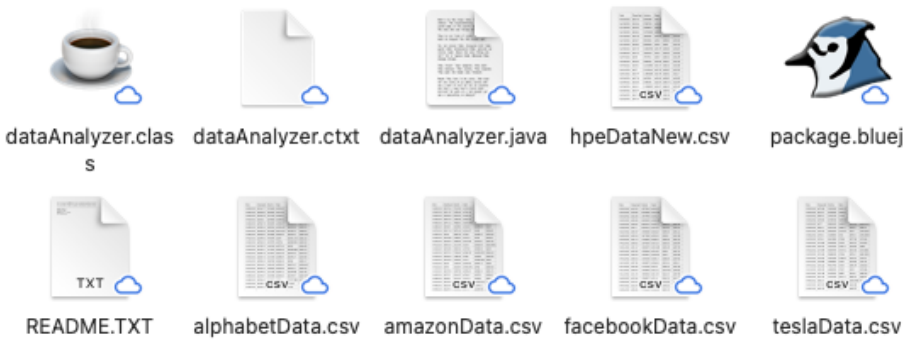
1. While Loops Setup for Entire Menu (Algorithmic Complexity)

```
String input = "";

while(!input.equals("Q"))
{
    // clear();
    System.out.println("1. Read Data From File");
    System.out.println("2. Average of Last Month of Trade Days");
    System.out.println("3. Average of Last Year of Trade Days");
    System.out.println("4. Display Data");
    System.out.println("5. Lowest Trade Date in the Last Month of Trade Days");
    System.out.println("6. Lowest Trade Date in the Last Year of Trade Days");
    System.out.println("7. Highest Trade Date in the Last Month of Trade Days");
    System.out.println("8. Highest Trade Date in the Last Year of Trade Days");
    System.out.println("Q. Quit");

    input = keyboard.nextLine();

    if (input.equals("1"))
    {
        clear();
        System.out.println("Please enter the name of the stock to analyze: "); // Determine the Stock File to Load
        System.out.println("(1.) Hewlett Packard Enterprise, HPE"); // Determine the Stock File to Load
        System.out.println("(2.) Amazon, AMZN"); // Determine the Stock File to Load
        System.out.println("(3.) Alphabet, GOOGL"); // Determine the Stock File to Load
        System.out.println("(4.) Tesla, TSLA"); // Determine the Stock File to Load
        System.out.println("(5.) Facebook, FB"); // Determine the Stock File to Load

        // String stockName = keyboard.nextLine();
        String inputStockType = keyboard.nextLine(); // Record the Stock Type to Load
        // currentStockType = integer.parseInt(inputStockType); // Set the Inputed Stock (Company) (Loaded) as the current Stock Company

        try
        {
           if (inputStockType.equals("1"))
           {
               dataHPE = readDataFromFile("hpeDataNew.csv");
               clear();
               System.out.println("Succesfully imported data for " + inputStockType);
               currentStockData = dataHPE; // Set: Current Stock Data (Array)
                                   System.out.println();


           }
           else if (inputStockType.equals("2"))
           {
               // System.out.println("Amazon");
               dataAMZN = readDataFromFile("amazonData.csv");
               clear();
               System.out.println("Succesfully imported data for " + inputStockType);
               currentStockData = dataAMZN;
                                          // System.out.println("Test2");

               System.out.println();


           }
           else if (inputStockType.equals("3"))
           {
               dataGOOGL = readDataFromFile("alphabetData.csv");
               clear();
               System.out.println("Succesfully imported data for " + inputStockType);
               currentStockData = dataGOOGL;

                                   System.out.println();


           }
           else if (inputStockType.equals("4"))
           {
               dataTSLA = readDataFromFile("teslaData.csv");
               clear();
               System.out.println("Succesfully imported data for " + inputStockType);
               currentStockData = dataTSLA;

                                   System.out.println();
```

```
            }
            else if (input.equals("6"))
            {
            double lowestDay = parsePrice(currentStockData.get(1)[1]);

            for (int i = 1; i < 253; i++)
            {
                double dailyClose = parsePrice(currentStockData.get(i)[1]);
                if (dailyClose < lowestDay)
                {
                    lowestDay = dailyClose;
                }
                // else lowestDay = lowestDay;
            }
            if(currentStockData.size() > 1)
            {
                clear();
                System.out.println();
                System.out.println("The lowest trading price over the last year of trade days is $" + lowestDay);
            }
            System.out.println("Press 'Enter' when ready to return to main menu");
            System.out.println();
            input = keyboard.nextLine();
            }
            else if (input.equals("7"))
            {
            double highestDay = parsePrice(currentStockData.get(1)[1]);

            for (int i = 1; i < 21; i++)
            {
                double dailyClose = parsePrice(currentStockData.get(i)[1]);
                if (dailyClose > highestDay)
                {
                    highestDay = dailyClose;
                }
                // else lowestDay = lowestDay;
            }
            if(currentStockData.size() > 1)
            {
                clear();
                System.out.println();
                System.out.println("The highest trading price over the last month of trade days is $" + highestDay);
            }
            System.out.println("Press 'Enter' when ready to return to main menu");
            System.out.println();
            input = keyboard.nextLine();
            }
            else if (input.equals("8"))
            {
            double highestDay = parsePrice(currentStockData.get(1)[1]);

            for (int i = 1; i < 253; i++)
            {
                double dailyClose = parsePrice(currentStockData.get(i)[1]);
                if (dailyClose > highestDay)
                {
                    highestDay = dailyClose;
                }
                // else lowestDay = lowestDay;
            }
            if(currentStockData.size() > 1)
            {
                clear();
                System.out.println();
                System.out.println("The highest trading price over the last year of trade days is $" + highestDay);
            }
            System.out.println("Press 'Enter' when ready to return to main menu");
            System.out.println();
            input = keyboard.nextLine();
            }
        }
```

Notes/Observations: The ‘while’ loop is most likely the most vital part of the program because it holds together the entire “menu” option that the user sees (*photo 1 and 2*). It is incorporated throughout the ‘main’ class, allowing the user to continuously pick between what analysis/company/display of data they would like to see with the company they have currently selected. This technique of employing a ‘while’ loop to continuously run the menu is taken from a previous in-class assignment of reading and analyzing.csv files, essentially creating a similar environment in the ‘main’ method, where the program continuously cycles in a menu of options until the user of the program enters “Q” to quit the program.

2. **Use of .csv Files for Data Storage (System Design & Algorithmic Complexity)**




```java
 * Method to read in the csv File.
 */
public static ArrayList<String[]> readDataFromFile(String filename) throws FileNotFoundException
{
    Scanner myFile = new Scanner(new File(filename)); // Open the File
    // System.out.println ("Read Data 1");

    // ArrayList<String[]> dataHPE = new ArrayList<String[]>(); // Array of Row Data
    // String[] row; // Array of Row Contents

    ArrayList <String[]> data = new ArrayList<String[]>(); // Array of Row Data
                    String[] row; // Array of Row Contents



    // loop through all the lines in the file
    while (myFile.hasNextLine())
    {
        String line = myFile.nextLine();
        row = line.split(",");
        data.add(row);
    }

    myFile.close();
    // System.out.println(data);
    return data;
```

```java
if (input.equals("1"))
{
    clear();
    System.out.println("Please enter the name of the stock to analyze: "); // Determine the Stock File to Load
    System.out.println("(1.) Hewlett Packard Enterprise, HPE"); // Determine the Stock File to Load
    System.out.println("(2.) Amazon, AMZN"); // Determine the Stock File to Load
    System.out.println("(3.) Alphabet, GOOGL"); // Determine the Stock File to Load
    System.out.println("(4.) Tesla, TSLA"); // Determine the Stock File to Load
    System.out.println("(5.) Facebook, FB"); // Determine the Stock File to Load

    // String stockName = keyboard.nextLine();
    String inputStockType = keyboard.nextLine(); // Record the Stock Type to Load
    // currentStockType = integer.parseInt(inputStockType); // Set the Inputed Stock (Company) (Loaded) as the current Stock Compan

    try
    {
        if (inputStockType.equals("1"))
        {
            dataHPE = readDataFromFile("hpeDataNew.csv");
            clear();
            System.out.println("Succesfully imported data for " + inputStockType);
            currentStockData = dataHPE; // Set: Current Stock Data (Array)
                                System.out.println();
```

Notes/Observations: The stock analyzer has the option to pick between five different high profile stock trades, and all of the data for those stocks is taken from .csv files from the official NASDAQ website. These five .csv files have been preloaded into the java BlueJ file, for the sake of time constraints of the project (*photo 1*). Within the StockAnalyzer methods, the method ReadDataFromFile utilizes a preloaded Java utility scanner to run through the .csv file sent from the 'main' method company picker -- The scanner runs through the file and creates a new ArrayList to return properly to the main function (*photo 2*) and to eventually be used to make data analysis calculations in the 'main' class. It is important to note that this method has been written outside of the 'main' class because it can be sent five different pre-loaded csv files, and is more efficient to be written as a separate method to return to the main class. (*photo 3*) The last screenshot demonstrates how this method ReadDataFromFile is called within the main class, utilizing the first option under option 1, "Please enter the name of the Stock to analyze", and the user selects from a numbered list of the five companies. Said user chooses option 1, "Hewlett Packard", for example, and the variable dataHPE is sent to ReadDataFromFile, which creates the ArrayList mentioned earlier. Once ReadDataFromFile is complete in its call, the if statement returns a confirmation that the data is successfully imported.

3. Iteration (loops) used to make calculations through Array (Algorithmic Complexity)

```
else if (input.equals("2"))
{
    double totalTwenty = 0;
    double finalAverage = 0;
    for (int i = 1; i < 21; i++)
    {
        double dailyClose = parsePrice(currentStockData.get(i)[1]);
        totalTwenty += dailyClose;
    }
```

```
else if (input.equals("5"))
{
double lowestDay = parsePrice(currentStockData.get(1)[1]);

for (int i = 1; i < 21; i++)
{
    double dailyClose = parsePrice(currentStockData.get(i)[1]);
    if (dailyClose < lowestDay)
    {
       lowestDay = dailyClose;
    }
    // else lowestDay = lowestDay;
}
if(currentStockData.size() > 1)
{
    clear();
    System.out.println();
    System.out.println("The lowest trading price over the last month of trade days is $" + lowestDay);
}
System.out.println("Press 'Enter' when ready to return to main menu");
System.out.println();
input = keyboard.nextLine();
}
```

Notes/Observations: As mentioned earlier in the criterion C explanation, loops are a vital part of the StockAnalyzer code, and the large 'while' loop is a cited example. Other examples, employed throughout the main class and other methods utilized 'for' loops to analyze the data for specific metrics, such as data averages, high stock days, low stock days, etc. The 'for' loop was most commonly used for this solution because it runs through the ArrayLists as needed. For example, in (*photo 1*), the user selects option 2 in the main menu, "2. Average of Last Month of Trade Days", and this method creates an instance variable that will eventually sum up to the requested amount of stock data days (in this case one month). Utilizing the 'for' loop, the loop runs through the "Closing Value" column in the selected stock data's ArrayList, going through the last twenty days (representing a month of trade days) and adding each of those day values to that instance variable. It is important to note that within this for loop another method is called, parsePrice(), which alters the string from the ArrayList because it originally contains a "$" character, and converts the remaining dollars and cents value to a double; this an extremely important method because it make the "average" math calculation actually possible, instead of trying to process String data which is not a possibility with Java. Once said 'for' loop is finished, the average is calculated by dividing the instance sum variable by the number of trade days taken.

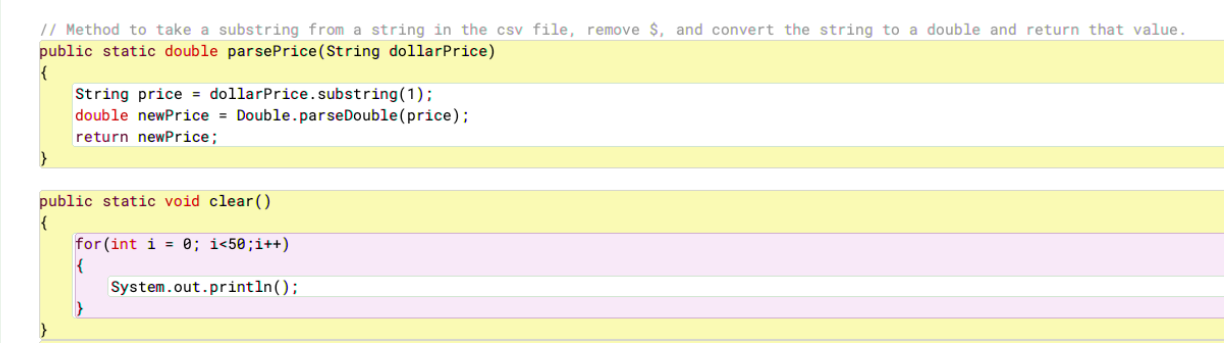
(*Photo 2*) This option "5", "5. Lowest Trade Date in the Last Month of Trade Days" is even more complex than the last option because it involves a conditional statement that is housed within a 'for' loop again. The method again begins with an initialized variable that is set to the first available piece of (processed parsePrice()) data. The if statement within the for loop checks if

the data the 'for' loop sends is less than the original instance variable, and if so, it is redefined. This is an example of a more complex algorithm within the StockAnalyzer code, because it takes into account several factors: the converted data, the conversion of said data into double values, and those double values being compared to each other's value only the specific amount of times that the user has asked it to (one month versus one year).

Notable methods called:

parsePrice(): this method is used to take a substring of the data in the ArrayList and convert this data to a double value that can actually be processed mathematically.

4. Breakdown of code into separate methods (Organization)



```
// Method to take a substring from a string in the csv file, remove $, and convert the string to a double and return that value.
public static double parsePrice(String dollarPrice)
{
    String price = dollarPrice.substring(1);
    double newPrice = Double.parseDouble(price);
    return newPrice;
}

public static void clear()
{
    for(int i = 0; i<50;i++)
    {
        System.out.println();
    }
}
```

Notes/Observations: This is an often overlooked aspect of the solution coding, however, it is of marginal importance because it has allowed the code to be more free from redundancies and extra processing. In an effort to not have to convert each created ArrayList from the ReadDataFromFile() method, a parsePrice() method was written outside of the main() class in order to convert specific values from the ArrayList, removing the "$" character and converting the remaining data to doubles. This was extremely efficient to write as a separate method, because otherwise it would've had to have been written and copied for every single other data analysis calculation in the main menu, because every other option also required the data to be converted successfully to double variables for calculation. Another method that was written separately for efficiency and organization was the clear() method, which is used to clear the screen between interactions of the user and the menu functions. This method was also called for every single selection of the main() method, and rather than be written redundantly and copied

for each method as (for) loop, it was written one time and is called in one simple command clear();

5. User Interface (Command Line) Simplified Look (Organization)

```
public static void clear()
{
    for(int i = 0; i<50;i++)
    {
        System.out.println();
    }
}
```

```
lease enter the name of the stock to analyze:
1.) Hewlett Packard Enterprise, HPE
2.) Amazon, AMZN
3.) Alphabet, GOOGL
4.) Tesla, TSLA
5.) Facebook, FB
```

```
The average trading price over the last year of trade days is $11.21
Press 'Enter' when ready to return to main menu
```

Notes/Observations: It was very important to the client and myself that the command line interface carry the appearance that it is neat and not confusingly ordered, which happens because otherwise the previous interactions in any other command line code would just stay on the screen as the user continues using their respective program. The use of the clear() method, as written in the previous paragraph, helps fulfill this criteria because it creates the appearance that the screen has been “wiped” in between interactions by the user, making it very clear that what is presented on the screen is only new material and not an old interaction. This is a very important aspect of the code, not only for organization but also user-ability and intuitiveness of the program. A user would otherwise be very confused if they read their code from the top with every interaction, because the command-line screen doesn’t erase (or, in this case, provide a visual distinction) with each keyboard request that the user enters.