# **Programming Assignment Unit 4 (v2)**

Computer Science, University of the People

CS 1101-01 Programming Fundamentals - CS 1102-01 - AY2024-T2

Instructor, Noman Shihadeh

December 18, 2023

## Stock Data Analysis Project

For this assignment, we were asked to write a program that analyses stock data provided to us in an array of data containing 10 days of stock prices as float values. The analysis is done using four functions as described here:

1. Calculate the average stock price.
2. Find the maximum stock price.
3. Find the occurrence count of a given stock price.
4. Calculate the cumulative sum of all the stock prices.

Source Code:

// StockAnalysis\_v2.java

import java.util.ArrayList;

import java.util.Random;

/\*\*

 \* This class provides methods to analyze stock prices,

 \* including:

 \* 1. Calculating average price

 \* 2. Finding maximum price

 \* 3. Counting occurrences of a specific price

 \* 4. Computing cumulative sums

 \*/

public class StockAnalysis\_v2 {

    // Length of the stock prices array

    private final static int STOCK\_DATA\_LENGTH = 10;

    /\*\*

     \* Main entry point

     \* Builds the array of demo stock prices and calls the methods to analyze them.

     \*

     \* @param args The command line arguments.

     \*/

    public static void main(String[] args) {

        // Random generator for stock prices

        Random random = new Random();

        // Creating an array of random stock prices

        float[] stockPrices = new float[STOCK\_DATA\_LENGTH]; // Array of randomly generated stock prices

        for (int i = 0; i < stockPrices.length; i++) {

            // Generating a random value between 100.0 and 200.0

            stockPrices[i] = 100.0f + random.nextFloat() \* 100.0f;

        }

        // Print out the randomly generated stock prices for reference

        System.out.print("Generated Stock Prices: ");

        int i = 0;

        while (i < stockPrices.length) {

            System.out.print(stockPrices[i] + " | ");

            i++;

        }

        System.out.println("--------------------------------");

        // Displaying the average stock price

        float average = calculateAvgStockPrice(stockPrices);

        System.out.println("Average Stock Price: " + average);

        System.out.println("--------------------------------");

        // Displaying the maximum stock price

        float maximum = findMaxStockPrice(stockPrices);

        System.out.println("Maximum Stock Price: " + maximum);

        System.out.println("--------------------------------");

        // Pick a random value from the generated stock prices to use as the target

        // price for counting occurrences

        float targetPrice = stockPrices[random.nextInt(stockPrices.length)];

        // Counting occurrences of a specific price

        int occurrences = countPriceOccurrences(targetPrice, stockPrices);

        System.out.println("Occurrences of " + targetPrice + ": " + occurrences);

        System.out.println("--------------------------------");

        // Converting the array of stock prices to an ArrayList

        ArrayList<Float> stockPricesList = convertArrayToArrayList(stockPrices);

        // Displaying the cumulative sum of stock prices

        ArrayList<Float> cumulativeSumList = calculateTheCumulativeSum(stockPricesList);

        System.out.println("Cumulative Sums: " + cumulativeSumList);

    }

    /\*\*

     \* Converts an array of floats to an ArrayList of floats.

     \*

     \* @param floatArray The array of floats to be converted.

     \* @return An ArrayList containing the elements of the given float array.

     \*/

    public static ArrayList<Float> convertArrayToArrayList(float[] floatArray) {

        ArrayList<Float> floatList = new ArrayList<>();

        for (float value : floatArray) {

            floatList.add(value);

        }

        return floatList;

    }

    /\*\*

     \* Calculates the average based on an array of stock prices.

     \* This method uses a traditional for loop.

     \*

     \* @param stockList The array of stock prices.

     \* @return The average price of the stock prices.

     \*/

    public static float calculateAvgStockPrice(float[] stockArray) {

        // Create and initialize a local variable to store the running total

        float result = 0;

        // Iterate through the given array of stock prices using a traditional for loop

        for (int i = 0; i < stockArray.length; i++) {

            // Add the current stock price to the running total

            result += stockArray[i];

        }

        // Return the average of the stock prices by taking the running total and

        // dividing by the length of the given array

        return result / stockArray.length;

    }

    /\*\*

     \* Finds the highest value in an array of stock prices.

     \* This method uses a do-while loop even though it might not be the most

     \* efficient method to do so.

     \* Since we are using a do-while loop, we need to check if the array is empty

     \* to avoid running into an IndexOutOfBoundsException.

     \*

     \* @param stockArray The array of stock prices.

     \* @return The highest stock price.

     \*/

    public static float findMaxStockPrice(float[] stockArray) {

        // Check if the array is empty to avoid IndexOutOfBoundsException

        if (stockArray == null || stockArray.length == 0) {

            throw new IllegalArgumentException("Array is empty");

        }

        // Create and initialize a local variable to store the highest value

        float max = stockArray[0];

        // Iterate through the given array of stock prices using a traditional for loop

        int i = 1;

        do {

            // Check if the current stock price is higher than the highest value

            if (stockArray[i] > max) {

                // If so, update the highest value

                max = stockArray[i];

            }

            i++;

        } while (i < stockArray.length);

        // Return the highest value

        return max;

    }

    /\*\*

     \* Counts how many times a specific price occurs in an array of stock prices.

     \* This method uses a classic for loop to iterate through the array.

     \*

     \* @param lookingForValue The specific price to count.

     \* @param stockArray      The array of stock prices.

     \* @return The count of the specific price.

     \*/

    public static int countPriceOccurrences(float lookingForValue, float[] stockArray) {

        // Create and initialize a local variable to store the count

        int count = 0;

        // Iterate through the given array of stock prices using a traditional for loop

        for (int i = 0; i < stockArray.length; i++) {

            // Check if the current stock price is equal to the specific price

            if (stockArray[i] == lookingForValue) {

                // If so, increment the count

                count++;

            }

        }

        // Return the count

        return count;

    }

    /\*\*

     \* Calculates the cumulative sum of an ArrayList of stock prices.

     \* This method uses an for..each style loop to iterate through the ArrayList.

     \*

     \* @param stockList The ArrayList of stock prices.

     \* @return An ArrayList representing the cumulative sum.

     \*/

    public static ArrayList<Float> calculateTheCumulativeSum(ArrayList<Float> stockList) {

        // Create and initialize a local variable to store the running total

        ArrayList<Float> resultList = new ArrayList<>();

        // create a variable to store the running sum

        float runningSum = 0;

        // Iterate through the given ArrayList of stock prices using a for..each style

        // loop

        for (float stock : stockList) {

            // Add the current stock price to the running total

            runningSum += stock;

            // Add the current running total to resultList

            resultList.add(runningSum);

        }

        // Return the ArrayList representing the cumulative sum

        return resultList;

    }

}

Output (Values are generated randomly so the output will change each time the code is run):

Generated Stock Prices: 170.58276 | 111.227776 | 111.27254 | 162.02481 | 136.24854 | 155.14088 | 190.6275 | 176.93219 | 105.752594 | 193.86183 | --------------------------------

Average Stock Price: 151.36713

--------------------------------

Maximum Stock Price: 193.86183

--------------------------------

Occurrences of 111.227776: 1

--------------------------------

Cumulative Sums: [170.58276, 281.81055, 393.08307, 555.1079, 691.35645, 846.4973, 1037.1248, 1214.0569, 1319.8094, 1513.6713]

The main parts of the code:

1. The “main” method is our program entry point, it is also where we build our demo stock data to be passed into the different methods and then we print the results to the user interface. The first step in this method is to build our demo stock list, we do so using random values generated using the python “Random” library.
2. The “calculateAvgStockPrice” this method takes in a single parameter called “stockArray” of type float array. The method then iterates over the array using the classic for loop for each iteration it accumulates the value in the array’s current location into a helper variable. After the loop exits, we return the helper variable containing the sum of the arrays values as a return value after we divide the it by the length of the array.
3. The “findMaxStockPrice” method takes in a single parameter called “stockArray” of type float array. Since this method is going to iterate the loop using a do..while loop we first verify that the array passed into the function is not empty. After that we initialize and create a local helper variable to the value located in the first position of the array. Next, we iterate over the array using a do..while loop while for each value in the array, we check to see if it is a value that is greater than the value currently found in our helper variable. If it is, we replace the value in the helper variable with the current array value. Finally, when we exit the loop, we return the value last found in the helper variable since it now should hold the largest value found in the array.
4. The “countPriceOccurrences” method takes in two parameters the first, is called “lookingForValue” of type float and represents the value we want to find and count in the array. The second is called “stockArray” of type float array, that represents the array of stock prices we are going to search. In the method body, we first create and initialize a local helper variable which will contain the incrementor that will represent and count the number of times the value “lookingForPrice” was found in the array. We next iterate through the array using a classic for loop each time checking if the current value in the array is equal to the value found in “lookingForPrice” If yes, we increment the helper variable if not, we continue with no action. At the end of the loop, we return the value in the helper variable that represents the number of times we encountered the value in the array.
5. The “calculateTheCumulativeSum” method takes in a single parameter called “stockList” of type ArrayList of floats. The first thing we do is to create and initialize a new local variable of ArrayList of floats that we will use and populate with the new values from the calculations we will perform in this method. We will also need another helper variable to represent a running summary of the values as we iterate the list. We now iterate over the list using a for..each loop each time adding the current value to the helper variable and then adding the running sum value to the local list variable. At the end of the loop, we return the local list which now contains a running cumulative sum of the list that was passed in the parameters.
6. The “convertArrayToArrayList” function is a helper function that takes in an array of floats and converts it into a ArrayList of floats which is then returned. The function does this by creating and empty ArrayList of floats and then iterates over the array of floats using a for..each loop and for each value it adds the value from the array into the ArrayList.

## References

Java Language and Virtual Machine Specifications

<https://docs.oracle.com/javase/specs/index.html>

Introduction to Programming Using Java - Version 9.0, JavaFX Edition

<https://math.hws.edu/javanotes/>

Source Code:

