

Criterion C

Contents

Contents	1
Summary	2
Techniques	2
Programming paradigms	2
Polymorphism	3
Method Overriding	3
Class structure	4
Package layout	4
UML class diagram	7
Data Structures	9
Binary Search Tree	9
JSON File	11
MySQL Database	13
Monte Carlo Stock Analysis	14
Encryption	19
API	20
Serialisation	20
References	22

Summary

The following analysis will include the techniques used to create the program and the data structures used. The product is based on the product described in criteria A and B, and the product was designed based on criterion B. Some changes were made in its development and will be justified in what will follow.

Techniques

Programming paradigms

The following programming paradigms were used:

- Object Oriented Programming
- Event driven programming

Feature of OOP I used	Justification of why the specific OOP feature was used
Encapsulation	<ul style="list-style-type: none">• OOP makes users think about what is being exposed to the project• It allows one to alter their implementation of an object without affect any other code¹
Polymorphism	<ul style="list-style-type: none">• It allows one to have many different functions with the same method name• Methods have the same job to do On different pieces of data depending on what is passed as arguments.²
Inheritance	<ul style="list-style-type: none">• Minimization of the amount of duplicate code – subclass has access to code of superclass• Reusability of code – subclasses can use superclass methods• Extensibility of code – creates more logical connections instead of storing all classes together• Data hiding – superclass can have private variables that can't be altered by subclass• Overriding – subclass can override methods of superclass³

Table 1: Reasons why OOP was used

¹ <https://www.codeproject.com/Questions/285934/why-we-use-object-oriented-programming>

² Ibid.

³ <https://www.quora.com/What-are-all-the-advantages-of-inheritance-in-Java>

Polymorphism

Method Overriding

The use of the above technique was necessitated to deal with the large range of graphing that had to occur such as graphing historical values (figure 2), predictions (figure 1) and distributions of prices (figure 3). Thus, in the spirit of time saving and efficiency, I chose to override the same method.

Functionality of tracing and zooming added to graph (present in all methods)

Preconditions: (left) company name, (middle) list of historical stock prices in given date range previously specified by user, (right) predictions based on historical close values.

```
// open a line graph frame
public LineChart(String symbol, List<StockInfo> st, ArrayList<Prediction> pred) {
    this.title = "Stock time series: " + symbol;
    //Free chart chart
    chartPanel = createChart(this.title, st, pred);
    JFrame f = new JFrame(this.title);
    f.setTitle(this.title);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setLayout(new BorderLayout(0, 5));
    f.add(chartPanel, BorderLayout.CENTER);
    //Enable mouse controls
    chartPanel.setMouseWheelEnabled(true);
    chartPanel.setHorizontalAxisTrace(true); //sets Axis
    chartPanel.setVerticalAxisTrace(true);

    //
    JPanel panel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    //add functionality to the panel to allow zoom in and format dates
    panel.add(createTrace());
    panel.add(createDate());
    panel.add(createZoom());

    f.add(panel, BorderLayout.SOUTH);
    f.pack();
    f.setLocationRelativeTo(null);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}
```

Method that plots the datasets of the dates horizontally (retrieved from the stock objects, see figure 10) and the historical stock values in addition to the predictions vertically, on a chartPanel which is to be added to a JFrame.

Figure 1: method for plotting historical and future time series of stock prices

```
public LineChart(String symbol, List<StockInfo> st) {
    this.title = symbol;
    //Free chart chart
    chartPanel = createChart(this.title, st);
    JFrame f = new JFrame(this.title);
    f.setTitle(this.title);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setLayout(new BorderLayout(0, 5));
    f.add(chartPanel, BorderLayout.CENTER);
    //Enable mouse controls
    chartPanel.setMouseWheelEnabled(true);
    chartPanel.setHorizontalAxisTrace(true); //sets Axis
    chartPanel.setVerticalAxisTrace(true);

    //
    JPanel panel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    //add functionality to the panel to allow zoom in and format dates
    panel.add(createTrace());
    panel.add(createDate());
    panel.add(createZoom());
    f.add(panel, BorderLayout.SOUTH);
    f.pack();
    f.setLocationRelativeTo(null);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}
```

Preconditions: (left) company name, (right) list of historical stock prices in given date range specified by the user.

Method that plots the datasets of the dates horizontally (retrieved from the stock objects, see figure 10) and the historical stock values vertically, on a chartPanel which is to be added to a JFrame.

Figure 2: method for plotting historical stock prices

```

public LineChart(ArrayList<Double> d1, ArrayList<Double> d2) {
    this.title = ""; this.title += "Distribution of Stock Prices";
    //Free chart chart
    chartPanel = createChart(this.title, d1, d2);
    JFrame f = new JFrame(title);
    f.setTitle(title);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setLayout(new BorderLayout(0, 5));
    f.add(chartPanel, BorderLayout.CENTER);
    //Enable mouse controls
    chartPanel.setMouseWheelEnabled(true);
    chartPanel.setHorizontalAxisTrace(true); //sets Axis
    chartPanel.setVerticalAxisTrace(true);

    //
    JPanel panel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    //add functionality to the panel to allow zoom in and format dates
    panel.add(createTrace());
    panel.add(createZoom());
    f.add(panel, BorderLayout.SOUTH);
    f.pack();
    f.setLocationRelativeTo(null);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

```

Preconditions: (left) generic dataset of floats – used for the average values intervals wherein future generated stock values lie, (right) generic dataset of floats – used for a measure of the frequency of future prices falling in the above intervals.

Method that plots the dataset d1 horizontally and the dataset d2 vertically, on a chartPanel which is to be added to a JInternalFrame, the postcondition.

Figure 3: method for plotting distribution of stock prices from historical and generated data

Class structure

Package layout

Package structure of product:

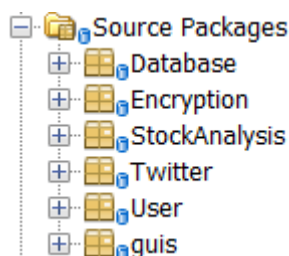
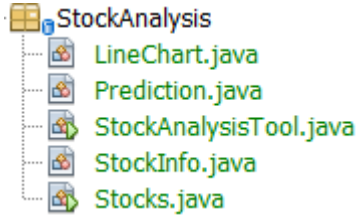

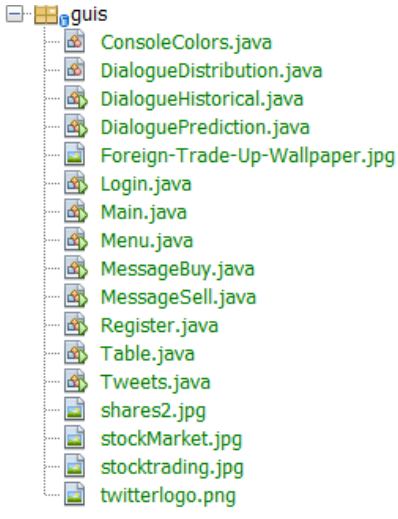
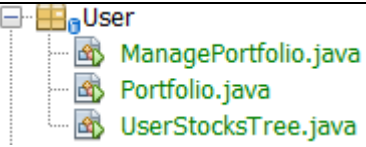


Figure 4: packages for product

Package name	What classes are included within package	Image of package	Why I chose to group classes into specific package
StockAnalysis	<ul style="list-style-type: none"> • LineChart: handles drawing of a line plot given either historical stock data and predictions or distribution and • Prediction: data structure that handles the creation and storage of a prediction for creating multiple objects with different names in a loop • StockAnalysisTool: implements the stochastic algorithm responsible for the creation of future predictions and also handles the creation of datasets with historical stock values and predictions that are to be plotted • StockInfo: data structure that encapsulates all retrieved information concerning stock prices from JSON file <p>Stocks: uses stock API to retrieve historical and real time data from online</p>		I wanted to group all the stock analysis classes into one package for simplicity
Encryption	<p>Password: handles the hashing of usernames and passwords using the SHA 256 cryptographic hash algorithm</p>		<ul style="list-style-type: none"> • All classes that have to do with encryption kept in same

			<p>location</p> <p>For future improvements, any changes to encryption can be made within this package</p>
guis	<ul style="list-style-type: none"> • Login: handles users signing into program • Register: handles users signing up • Main: initialises the program • Menu: handles the central graphical user interface of the product • Dialogue: handles pre conditions of display of historical values, predictions and stock price distribution • Message: handles pre conditions for buying and selling stocks • Tweets: handles the preconditions and the interface for the social media section of the program, involving displaying tweet searches based on a query 		<ul style="list-style-type: none"> • Keep all GUI frames together <p>Allow access to variables that aren't public</p>
User	<ul style="list-style-type: none"> • ManagePortfolio: handles buying, selling stocks in addition to file handling methods for serialisation of the UserStockTree in order for it to be included in the MySQL database • Portfolio: handles the display of portfolio of user in jTable form • UserStockTrees: binary 		<ul style="list-style-type: none"> • All classes dealing with user functionality

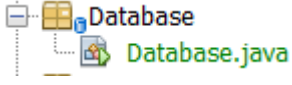
	search tree data structure used for storing transaction history (or contents of portfolio) of individual user		
Database	<ul style="list-style-type: none"> ● Database: handles creation and addition of entries as well as searching methods are included 		<ul style="list-style-type: none"> ● All classes dealing with the functioning of the database

Table 2: Explanation of what classes are included within the package, and reasoning behind packaging decisions

UML class diagram

Figure 5: UML class diagram of final product

Data Structures

Binary Search Tree

In the product, I decided to use a binary search tree as a means of storing the portfolio of a user. For my purposes, due to the small size of datasets, the time it would take for the execution of a search would be similar for all data structures such as linked lists, stacks and queues. However, the advantage of a binary search tree is **scalability** as it is a **dynamic data structure**. This means that the product would perform better for larger data sets than the aforementioned alternatives (see figure 6). More specifically, my method would allow for an average algorithmic complexity of $O(\log n)$ for searching and adding/deleting elements, compared to a linked-list, stack and queue average search complexity of $O(n)$.⁴

In figures 7 and 8, the **recursive** implementation of a binary search tree for the user portfolio is showcased. In the former, the construction of a 'node' object of a tree is detailed and, in the latter, a recursive search method is shown where a search is carried out to determine whether a user has a stock under a specific company or not. The effectiveness of this recursive implementation is that any insertion, deletion and search methods will be built using the same structure as they will rely on the recursive definition of the binary tree.

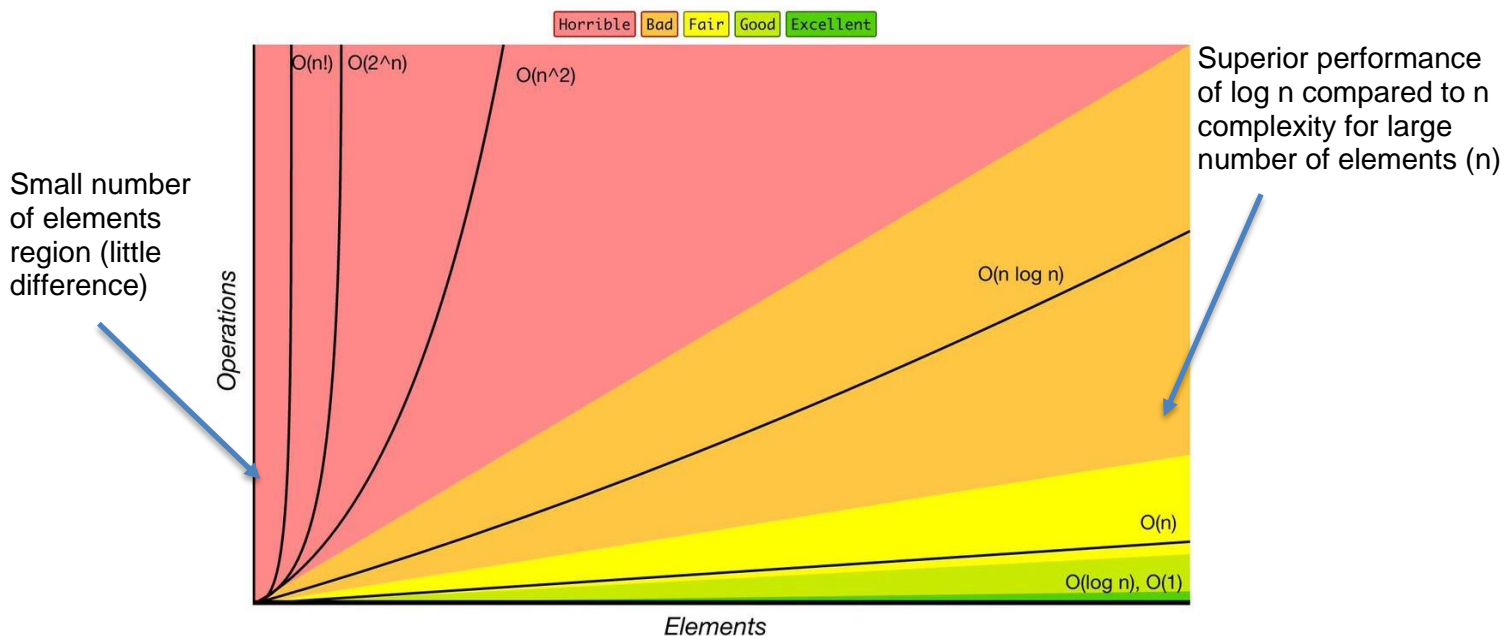


Figure 6: Algorithmic complexity illustration (<https://www.bigocheatsheet.com/>)

⁴ <https://www.bigocheatsheet.com/>

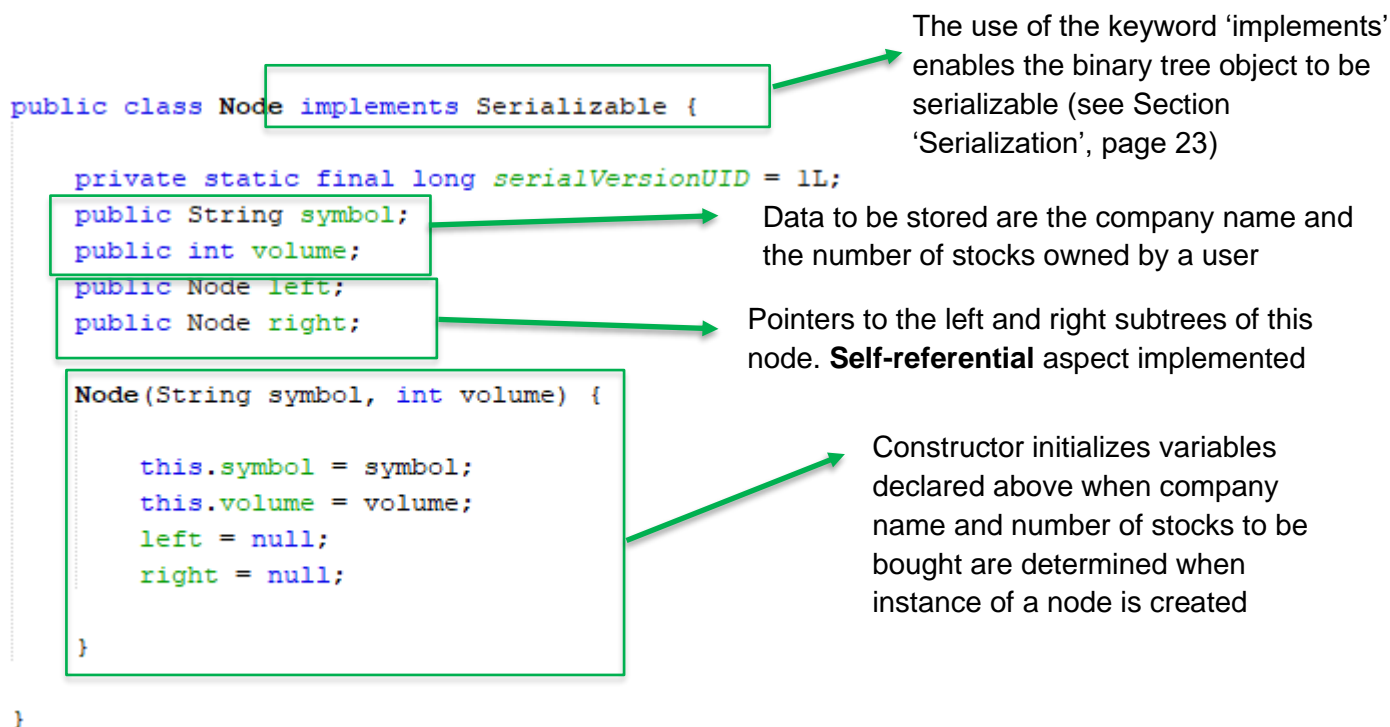


Figure 7: Class that defines the 'node' object that is the building block of the binary tree

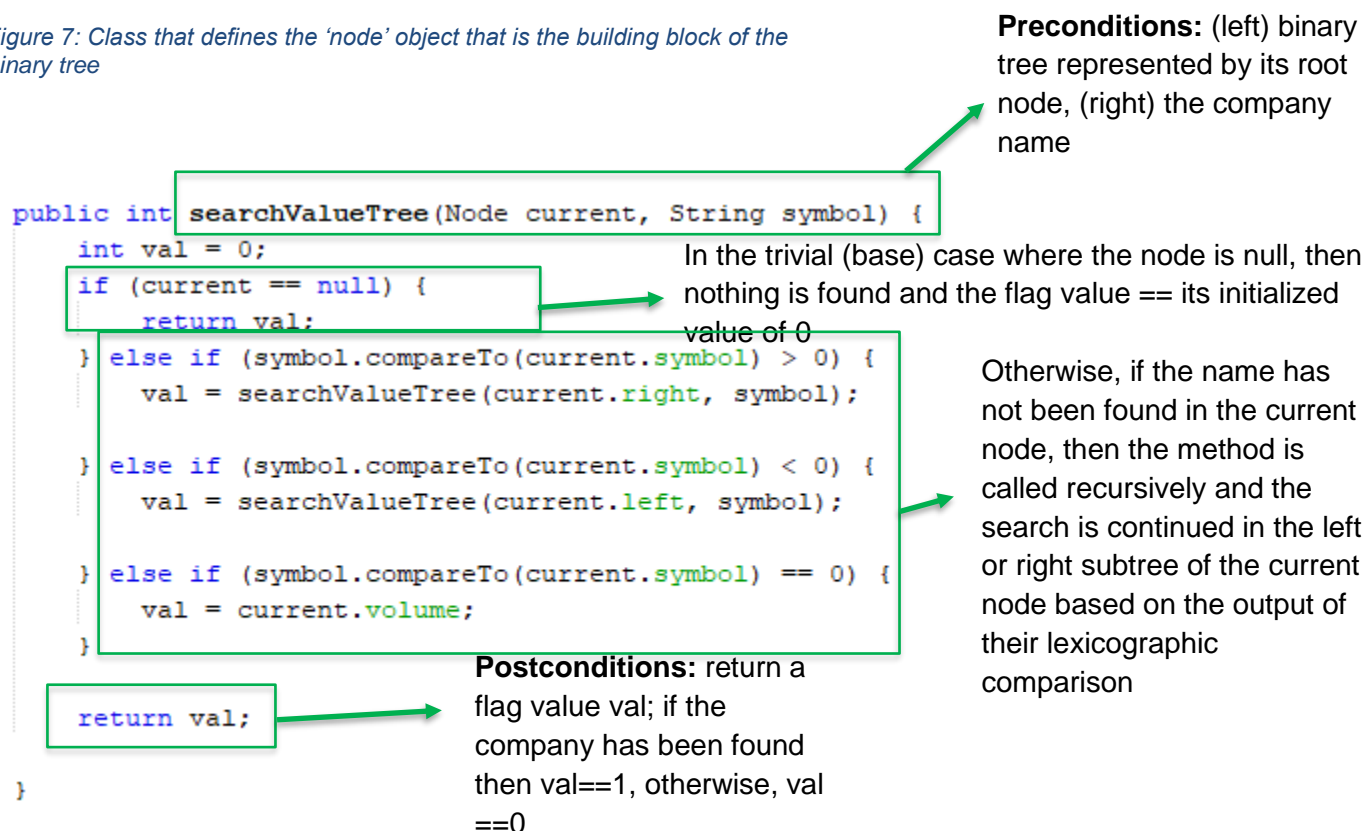


Figure 8: Recursive search file for binary search tree

JSON File

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language for me to use in my product as it is a Java application.⁵ Also, the format is intuitive and provides me with access to real-time data from the web of stock prices. An example of a part of a dataset from a company (Microsoft) can be seen in figure 9. Also, figure 10 shows a method for accessing stocks given a specific time period (start and end date).

```
▼ Meta Data:
  1. Information:      "Daily Prices (open, high, low, close) and Volumes"
  2. Symbol:          "MSFT"
  3. Last Refreshed:  "2019-11-27"
  4. Output Size:      "Full size"
  5. Time Zone:       "US/Eastern"
▼ Time Series (Daily):
  ▼ 2019-11-27:
    1. open:           "152.3300"
    2. high:           "152.5000"
    3. low:            "151.5200"
    4. close:          "152.3200"
    5. volume:         "15123806"
```

Figure 9: JSON file format with example stock prices for a given date

⁵ <https://www.json.org/>

Figure 10: Method for obtaining historical close values of a company between specified dates

ArrayList of 'stock' objects is initialized

Preconditions of method are the start and end dates from which stock values will be taken

```
public static List<StockInfo> getStocksBetweenDates(String symbol, String str_date, String end_date) {  
    ArrayList<StockInfo> stList = new ArrayList<>();  
    try {  
        JSONObject json = readJsonFromUrl("https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=" + symbol + "&outputsize=full&apikey=BYNBOF38V12BFVA0");  
        //System.out.println(json.toString()); // Keys: demo, 0GID9C19NNAK183U, 0IQV6FBQ2420110R, FSNVNRNQ4RHL635, 0ALDNZ5XSDOK14RB, BFRDPF091A413CW0  
        Object js1;  
        js1 = json.get("Time Series (Daily)");  
        List<String> dates = getDateBetween(str_date, end_date);  
        for (String date : dates) {  
            if (((JSONObject) js1).has(date)) {  
                Object js2 = ((JSONObject) js1).get(date);  
                double open = Double.parseDouble(String.valueOf(((JSONObject) js2).get("1. open")));  
                double high = Double.parseDouble(String.valueOf(((JSONObject) js2).get("2. high")));  
                double low = Double.parseDouble(String.valueOf(((JSONObject) js2).get("3. low")));  
                double close = Double.parseDouble(String.valueOf(((JSONObject) js2).get("4. close")));  
                double vol = Double.parseDouble(String.valueOf(((JSONObject) js2).get("5. volume")));  
                stList.add(new StockInfo(date, open, high, low, close, vol));  
            }  
        }  
    } catch (IOException ex) {  
        Logger.getLogger(Stocks.class.getName()).log(Level.SEVERE, null, ex);  
    } catch (JSONException ex) {  
        Logger.getLogger(Stocks.class.getName()).log(Level.SEVERE, null, ex);  
    }  
    return stList;  
}
```

URL of JSON file is used to create a JSON object instance from which all data will be accessed.

This method call is responsible for generating dates in interval specified by the preconditions

Iterating through all of the dates in the above interval, this if-loop checks and whether there are 'gaps' in the JSON file and skips them

Newly created stock object is then added to the aforementioned Array List of stock objects which is the postcondition of this method.

MySQL Database

The use of a database was a critical part of storing data in the product. Each user had a record which contains the fields: "Username", "Password", "Wallet" for the value of available money and "Stock Values" for the user portfolio. The method for initialising the database used by the users can be seen in figure 11 and the part of the method enabling users to buy stocks can be seen in figure 19.

```
public static String name;

static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:sqlite:C:\\Users\\pade\\Portfolio.db";

static final String USER = "username";
static final String PASS = "password";

public Database(String name) {
    this.name = name;
    Connection conn = null;
    Statement stmt = null;
    try {
        conn = DriverManager.getConnection(DB_URL);
        stmt = conn.createStatement();
        stmt.execute("CREATE TABLE " + this.name + " (username TEXT, password TEXT, Wallet TEXT, StockValues TEXT)");
    } catch (SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try {
            if (stmt != null) {
                stmt.close();
            }
        } catch (SQLException se) {
            //do nothing
        }
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException se) {
            se.printStackTrace();
        }
    }
}
```

Establishing connection with JDBC driver

Precondition: name of database

Establish connection to Database using the Driver Manager Class

Create statement which creates table with aforementioned fields

Figure 11: Initialising the database used by the users

Monte Carlo Stock Analysis Algorithm

Due to the size and complexity of the algorithm, in the following section, parts of the Javadoc (see figures 12 to 16), a documentation generator for “API documentation in HTML format from Java source code” will be shown.⁶

StockAnalysis

Class StockAnalysisTool

java.lang.Object
StockAnalysis.StockAnalysisTool

```
public class StockAnalysisTool  
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
java.lang.String	endDate Name of company start date of historical values end date of historical values
int	N Number of predictions
java.lang.String	startDate Start date of historical values
java.lang.String	symbol Name of company
int	val Number of days in the future of the stock projection

Figure 12: Fields used in StockAnalysisTool Class, see Table 2.

⁶<https://en.wikipedia.org/wiki/Javadoc>.

Field Detail

symbol

```
public java.lang.String symbol
```

Name of company

startDate

```
public java.lang.String startDate
```

Start date of historical values

endDate

```
public java.lang.String endDate
```

Name of company start date of historical values end date of historical values

val

```
public int val
```

Number of days in the future of the stock projection

N

```
public int N
```

Number of predictions

Figure 13: Field details for StockAnalysisTool Class, see Table 2.

Figure 14: Constructor and Method summary for Class StockAnalysisTool, see Table 2

Constructor Summary			
Constructors			
Constructor and Description			
StockAnalysisTool()			

Method Summary			
All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description	
void		DisplayFuture()	Displays the future stock prices given
void		Extrapolate()	Implements stochastic algorithm based on geometric Brownian motion for the generation of predicted stocks
void		getDistribution(int A)	Divides the range of generated and historical stock values and subdivides it into bins in each of which the frequency of stock prices in that range is counted.
void		getHistorical()	Displays historical values of stocks given preconditions being the fields in the class
static void		main(java.lang.String[] args)	
static double		mean(java.util.ArrayList<java.lang.Double> m)	Finds the mean of a list of 'real' numbers
static int		meanInt(java.util.ArrayList<java.lang.Integer> n)	Finds the mean of a list of integers
static double		Var(java.util.ArrayList<java.lang.Double> m)	Finds the variance of a list of integers
Methods inherited from class java.lang.Object			
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait			

Figure 15: Method detail for StockAnalysisTool Class, see Table 2.

Method Detail
<div>mean</div> <pre>public static double mean(java.util.ArrayList<java.lang.Double> m)</pre> <p>Finds the mean of a list of 'real' numbers</p> <p>Parameters:</p> <p>m - list of integers for which the mean is calculated</p> <p>Returns:</p> <p>mean</p> <p>See Also:</p> <p>meanInt</p>
<div>meanInt</div> <pre>public static int meanInt(java.util.ArrayList<java.lang.Integer> n)</pre> <p>Finds the mean of a list of integers</p> <p>Parameters:</p> <p>n - list of integers for which the mean is calculated</p> <p>Returns:</p> <p>mean</p> <p>See Also:</p> <p>mean</p>
<div>Var</div> <pre>public static double Var(java.util.ArrayList<java.lang.Double> m)</pre> <p>Finds the variance of a list of integers</p> <p>Parameters:</p> <p>m - list of integers for which the mean is calculated</p> <p>Returns:</p> <p>variance</p>
<div>getHistorical</div> <pre>public void getHistorical() throws java.text.ParseException</pre> <p>Displays historical values of stocks given preconditions being the fields in the class</p> <p>Throws:</p> <p>java.text.ParseException</p>

Figure 16: Method detail continued for StockAnalysisTool Class, see Table 2.

DisplayFuture

```
public void DisplayFuture()  
    throws java.text.ParseException
```

Displays the future stock prices given

Throws:

java.text.ParseException

Extrapolate

```
public void Extrapolate()  
    throws java.text.ParseException
```

Implements stochastic algorithm based on geometric Brownian motion for the generation of predicted stocks

Throws:

java.text.ParseException

getDistribution

```
public void getDistribution(int A)  
    throws java.text.ParseException
```

Divides the range of generated and historical stock values and subdivides it into bins in each of which the frequency of stock prices in that range is counted. Finally, the frequency of the stock prices are plotted against the bins in a LineChart.

Parameters:

A - the number of bins/intervals

Throws:

java.text.ParseException

main

```
public static void main(java.lang.String[] args)  
    throws java.text.ParseException,  
           java.security.NoSuchAlgorithmException
```

Throws:

java.text.ParseException

java.security.NoSuchAlgorithmException

Encryption

Below is the method for hashing the user's password in order to achieve confidentiality and increased security.

Precondition: user's password

```
public static byte[] getSHA(String password) throws NoSuchAlgorithmException
{
    // Static getInstance method is called with hashing SHA
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    // digest() method called
    // to calculate message digest (A digest is a small value generated by a hash function from a whole message) of an input
    // and return array of byte
    return md.digest(password.getBytes(StandardCharsets.UTF_8));
}

public static String toHexString(byte[] hash)
{
    // Convert byte array into signum representation
    BigInteger number = new BigInteger(1, hash);

    // Convert message digest into hex value
    StringBuilder hexString = new StringBuilder(number.toString(16));

    // Pad with leading zeros
    while (hexString.length() < 32)
    {
        hexString.insert(0, '0');
    }

    return hexString.toString();
}
```

Using the SHA 256 cryptographic hash algorithm

Password (String) is converted to an array of bytes (hash)

Preconditions: byte array hash

Byte array is converted into a hex-string which is the **postcondition** of this encryption algorithm

Figure 17: getSHA and toHexString methods converting input into array of bits by applying the cryptographic hash SHA 256 algorithm and then converting the hash into string-hex form respectively.

API

API and frameworks used	Reason API was used
Alphavantage ⁷	<ul style="list-style-type: none">• Provides me with a way of obtaining realtime and historical stock data through access to online JSON files• Intuitive and easy to use• Provides the building blocks for the creation of a large product• Excellent documentation
Twitter4j ⁸	<ul style="list-style-type: none">• Enables me to search for tweets through a query• Simple documentation with examples of how to use methods thereby helping with the learning curve• Only requires setting up a twitter account and app through twitter developer site to 'configure'

Table 3: Displaying APIs and frameworks used and the reasons why they were used

Serialisation

The need for serialisation arose from the need to store the user's transaction data (in the form of a binary tree) on a MySQL Database that could only support string and integer fields. Thus, the solution was to convert an object into a file with a unique name, which would be used in the database as a string. There were other alternatives such as: csv, JSON, text or random-access files. However, upon serialisation, the object at hand (the binary tree) is converted to a binary file and hence its security is enhanced, which would be desired in the real world; this is because it is hard to decode the contents of the binary file. This can be seen in figure 19 which displays a sample output of serialising a student Portfolio binary tree object. Below (see figure 18) is an example where serialisation is used when the user buys stocks.

⁷ <https://www.alphavantage.co/documentation/>

⁸ <http://twitter4j.org/en/code-examples.html>

The serialized Portfolio object of the user is retrieved from the local database

```
ResultSet rs = stmt.executeQuery("SELECT StockValues FROM " + Database.name + " as t WHERE username == '" + user + "'");

String stocks = rs.getString(1);
System.out.print(stocks);
String i = Integer.toString(j);
UserStocksTree Tree = ManagePortfolio.deserialise(stocks);
Tree.add(symbol, volume);

//System.out.print("After " + Tree.TotalValue(Tree.root));
String s = ManagePortfolio.serialise(Tree);
//System.out.print(s);

PreparedStatement ps = conn.prepareStatement(
    "Update " + Database.name + " Set StockValues='" + s + "' where username = '" + user + "'");
PreparedStatement psWallet = conn.prepareStatement(
    "Update " + Database.name + " Set Wallet='" + i + "' where username = '" + user + "'");

psWallet.executeUpdate();
psWallet.close();
ps.executeUpdate();
ps.close();

conn.close();
JOptionPane.showMessageDialog(null,
    "Transaction made");
```

The binary tree is deserialised and the transaction is changing the value of the portfolio, as per Criterion 3 and assuming the preconditions (such as sufficient income are met)

The updated binary tree is then serialized

The database is updated by storing in the user's record the path in memory of the newly created binary file

Figure 18: Part of "Buy" method enabling the user to 'acquire' stocks of a company.

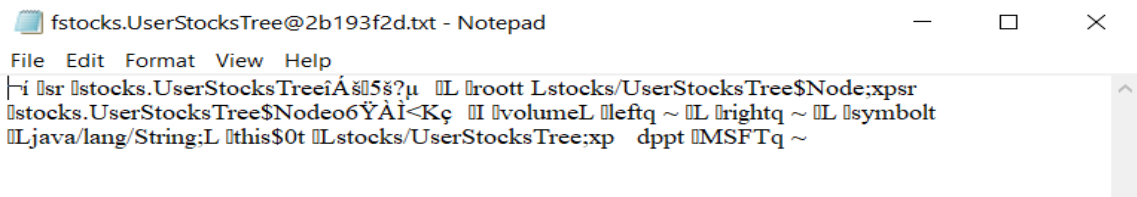


Figure 19: Figure: example of binary fine generalised upon serialisation of binary tree of user's portfolio

Word Count: 813

References

Alpha Vantage API Documentation. (n.d.). Retrieved from <https://www.alphavantage.co/documentation/>

Code Examples. (n.d.). Retrieved from <http://twitter4j.org/en/code-examples.html>

Garavaglia, E. (n.d.). why we use object-oriented programming. Retrieved from <https://www.codeproject.com/Questions/285934/why-we-use-object-oriented-programming>

Javadoc. (2020, February 3). Retrieved from <https://en.wikipedia.org/wiki/Javadoc>

JSON. (n.d.). Retrieved from <https://www.json.org/>

Know Thy Complexities! (n.d.). Retrieved from <https://www.bigocheatsheet.com/>