Das Projekt

Dieses Dokument dient dazu den aktuell stand des SWP- Projekts außenstehenden Personen näherzubringen. Dabei wird die eine Gliederung von

1. Verwendung/ Verwaltung API ->
2. Verwendung/ Verwaltung der Datenbank ->
3. Aufarbeitung der Daten für Basisprojekt ->
4. Basisprojekt aktueller Stand (nicht kombiniert)

Eingehalten. Das Pflichtenheft für das Projekt finden Sie unter folgendem Link:

https://github.com/flostaudacher/ETF_TLT_Backtesting/blob/master/Mein%20Pflichtenheft.pdf

1. Verwendung/ Verwaltung API (Download.Java)

Als API wird Alpha Vantage herbeigezogen. Dabei handelt es sich um eine API zur Beziehung von Daten aus dem Aktienmarkt. In unserem Fall verwenden wir die Funktion Extended History von Alpha Vantage.

Intraday (Extended History)

This API returns historical intraday time series for the trailing 2 years, covering over 2 million data points per ticker. The intraday data is computed directly from the Securities Information Processor (SIP) market-aggregated data feed. You can query both raw (as-traded) and split/dividend-adjusted intraday data from this endpoint. Common use cases for this API include data visualization, trading simulation/backtesting, and machine learning and deep learning applications with a longer horizon.

API Parameters

❚ **Required:** `function`

The time series of your choice. In this case, `function=TIME_SERIES_INTRADAY_EXTENDED`

❚ **Required:** `symbol`

The name of the equity of your choice. For example: `symbol=IBM`

❚ **Required:** `interval`

Time interval between two consecutive data points in the time series. The following values are supported: `1min`, `5min`, `15min`, `30min`, `60min`

❚ **Required:** `slice`

Two years of minute-level intraday data contains over 2 million data points, which can take up to Gigabytes of memory. To ensure optimal API response speed, the trailing 2 years of intraday data is evenly divided into 24 "slices" - `year1month1`, `year1month2`, `year1month3`, ..., `year1month11`, `year1month12`, `year2month1`, `year2month2`, `year2month3`, ..., `year2month11`, `year2month12`. Each slice is a 30-day window, with `year1month1` being the most recent and `year2month12` being the farthest from today. By default, `slice=year1month1`.

❚ Optional: `adjusted`

By default, `adjusted=true` and the output time series is adjusted by historical split and dividend events. Set `adjusted=false` to query raw (as-traded) intraday values.

❚ **Required:** `apikey`

Hier bekommen wir Daten zu jedem Tag zu einer Vielzahl an Zeitpunkten zu jeder Aktie. Erhalten wird eine CSV Datei zu jedem Monat (bis zu 2 Jahren). Um die gewollten Daten zu beziehen muss eine URL aufgerufen werden. Da wir die komplette Historie beziehen wollen müssen wir also 24 verschiedene URLs aufrufen. Das heißt wir müssen uns verschiedene URLs in Java zusammensetzten die den benötigten URLs entsprechen. Hier benötigen wir also mehrere Methoden:

Create URL zur Zusammensetzung:

```
static String createurl(int Month, String wantedSymbol) {
    String baseURL = "https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY_EXTENDED&symbol=";
    String symbol = wantedSymbol;
    String extdendetBaseOne = "&interval=";
    String interval = "15min";
    String extdendetBaseTwo = "&slice=";
    String usedMonth = months[Month];
    String extdendetBaseThree = "&apikey=";
    String apiKey = "QGEDE2GAG8OY3OBU";
    String URL = baseURL + symbol +  extdendetBaseOne + interval + extdendetBaseTwo + usedMonth + extdendetBaseThree + apiKey;
    //System.out.println(URL);
    return URL;
}
```

Eine Methode die uns den jeweiligen Monat für die URL liefert -> Monat für URL

```
private static void fillMonthsArray() {
    for (int i = 0; i < PARTS_OF_API; i++) {
        if (i < 12) {
            months [i] = "year1month" + (i+1);
        } else {
            months [i] = "year2month" + (i+1-12);
        }
    }
}
```

Eine Methode die dem User eine Eingabe eines Symbols ermöglicht -> Aktien Symbol für URL

```
public static String getSymbol() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Zu untersuchenedes Symbol : ");
    String s = scanner.nextLine();
    return s;
}
```

Da wir uns jetzt die jeweiligen URL s kreieren können, können wir jetzt den Aufruf der URL umsetzen und uns alle Teile der API mit Verzögerung da Alpha Vantage ohne Premium nur 5 calls die Minute zulässt in unser Data Directory downloaden.

```
static String fileNameBase = "C:\\Users\\flost\\eclipse-workspace\\ETF_BACKTESTING_BONUS\\data\\";
private static final File DIRECTORYDATA= new File ("C:\\Users\\flost\\eclipse-workspace\\ETF_BACKTESTING_BONUS\\data");
public static void download() {
    fillMonthsArray();
    String symbol = getSymbol();
    setSymbol(symbol);
    int zeit = 1;
    for (int i = 0; i < PARTS_OF_API; i++) {
        url[i] = createurl(i,symbol);
        fileName[i] =  fileNameBase + symbol +"_"+ months[i] +".csv";
        try (BufferedInputStream in = new BufferedInputStream(new URL(url[i]).openStream());
                FileOutputStream fileOutputStream =  new FileOutputStream(fileName[i])) {
            byte dataBuffer[]= new byte[1024];
            int bytesRead;
            while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
                fileOutputStream.write(dataBuffer, 0, bytesRead);
            }
        } catch (IOException e) {
            System.out.println("");
        }
        if (i +1 == 5*zeit) {
            try{
                zeit++;
                Thread.sleep(60000);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
    zeit = 0;
}
```

Durch diesen Programmteil werden die Teile der API in das DIRECTORYDATA geladen unter den jeweiligen Filename, der zusammengesetzt wird. Um die API nicht auszulasten wird nach 5 Downloads eine Auszeit von 60 000 Millisekunden eingehalten -> 1 Minute.

Um unser File System übersichtlich zu halten führen wir zum Start der Main noch eine Methode zum Löschen aller Files vorhandenen Files aus.

```
public static void deleteCurrentFiles() {
    File location= new File ("C:\\Users\\flost\\eclipse-workspace\\ETF_BACKTESTING_BONUS\\data");
    File[] files = location.listFiles();
    for (File file : files) {
        if (!file.delete()) {
            System.out.println("failed to delete");
        }
    }
}
```

1.1 Verwaltung der von der API bezogenen Daten (combination.java)

Bevor wir zu Schritt 2 kommen (Datenbank) machen wir uns diese Daten erst einmal handlicher kombinieren alle Files in eine Datenstruktur:

```
public static void combine() {
    // TODO Auto-generated method stub
    Rowc = 0;
    System.out.println(download.fileName.length);
    for (int i = download.fileName.length-1 ; i >= 0  ; i = i -1 ) {
        Scanner sc= null;
        try
        {
            System.out.println();
            sc= new Scanner (new BufferedReader(new FileReader(download.fileName[i])));
            sc.nextLine();
            while (sc.hasNextLine()) {
                InputLine = sc.nextLine();
                InputLine = InputLine.replaceAll(" ", ",");
                String[] inArr = InputLine.split(",");
                for (int x = 0; x < inArr.length; x++) {
                    CombinationArray[Rowc][x%7]=inArr[x];
                }
                Rowc++;
            }
        }catch (Exception e) {
            System.out.println(e);
        }
        NewArray();
    }
}
```

Es werden alle Daten nach deren zeitlichen Ablauf in einem Array geladen -> 24 CSV s zu einem String Array.

Dieser Kombination Array wird dann verwendet um unseren eigentlich Verwendeten Array den stock Array zu befüllen.

```
static void fillStockArray(String[][] stock) {
    for (int Rowc = 0; Rowc < combination.getLengthOfNeededArray(); Rowc++) {
        for (int Colc = 0; Colc < combination.getCols(); Colc++) {
            String s = "" + combination.CombinationArray[Rowc][Colc];
            stock[Rowc][Colc] = s;
        }
    }
}
```

2. Verwendung/ Verwaltung der Datenbank (MySQL & Java)

Als erstes müssen wir eine Datenbank mit geeigneten Tabellen in mySQL erzeugen

```
create database aktien;
use aktien;

create table aktienListe (
ID int (10) auto_increment primary key,
    Symbol varchar(50)
);
create table aktie(
    Symbol int (50) not null primary key,
    Datum date not null,
    Zeitpunkt varchar(50) not null,
    StockValue varchar (50) not null
);
```

2.1 Für diese Datenbank benötigen wir nun jeweils eine Klasse pro Tabelle. Eine Class für die Tabelle AktienListe mit dem jeweiligen Getter und Settern.

```
public class aktienListe {
    private int AktienID;
    private String symbol;
    public aktienListe() {
        super();
        // TODO Auto-generated constructor stub
    }
    public aktienListe(int aktienID, String symbol) {
        super();
        AktienID = aktienID;
        this.symbol = symbol;
    }
    public int getAktienID() {
        return AktienID;
    }
    public void setAktienID(int aktienID) {
        AktienID = aktienID;
    }
    public String getSymbol() {
        return symbol;
    }
    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }
    @Override
```

Und eine Class für die Tabelle aktie mit dem jeweiligen Getter und Settern.

```
public class aktie {
    private String date;
    private int symbol;
    private String timestamp;
    private String value;
    public aktie(int symbol, String date, String stock, String value) {
        super();
        this.date = date;
        this.symbol = symbol;
        this.timestamp = stock;
        this.value = value;
    }
    public aktie() {
        super();
    }
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }
    public int getSymbol() {
        return symbol;
    }
    public void setSymbol(int symbol) {
        this.symbol = symbol;
    }
    public String getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }
    public String getValue() {
        return value;
    }
    public void setValue(String value) {
        this.value = value;
    }
```

2.2 Nun geht es weiter zum DB Manager der uns mit MySql verknüpfen soll und uns etwaige Methoden zu Verfügung stellen soll.

Methoden Name	**Code**

getConnection	`public Connection getConnection()`
	`throws ClassNotFoundException, SQLException {`
	
	`Connection con = null;`
	`Class.forName("com.mysql.jdbc.Driver");`
	`con = DriverManager.getConnection(`
	`"jdbc:mysql://localhost:3306/aktien?serverTimezone=UTC&useSSL=false"`
	`"root",`
	`"13456stau"`
	`);`
	`return con;`
	`}`

Neue Aktie speichern	`public static void saveNewSpecificStockValue(Connection con, aktie stock) throws SQLException`
	`String sql = "insert into aktie (Symbol,Datum,Zeitpunkt,StockValue) values (?,?,?,?)";`
	`PreparedStatement stm = null;`
	`try {`
	`stm = con.prepareStatement(sql);`
	`System.out.println(stock.getDate());`
	`Date date = Date.valueOf(stock.getDate());`
	`stm.setInt(1, stock.getSymbol());`
	`stm.setDate(2, date);`
	`stm.setString(3, stock.getTimestamp());`
	`stm.setString(4, stock.getValue());`
	`stm.executeUpdate();`
	`}`
	`finally {`
	`if (stm != null)`
	`stm.close();`
	`}`
	`}`

Bestimmte Aktie auslesen	`public ArrayList<aktie> readStockValues (Connection con, int Symbol) throws SQLException{`
	`PreparedStatement stm = null;`
	`ResultSet rs = null;`
	`ArrayList<aktie> result = new ArrayList<aktie>();`
	`try {`
	`String sql = "select Datum, Zeitpunkt, StockValue from aktie where Symbol = ?";`
	`stm = con.prepareStatement(sql);`
	`stm.setInt(1, Symbol);`
	`rs = stm.executeQuery();`
	`while(rs.next()) {`
	`String Datum = rs.getString(1);`
	`String Zeitpunkt = rs.getString(2);`
	`String StockValue = rs.getString(3);`
	`aktie a = new aktie(Symbol,Datum,Zeitpunkt,StockValue);`
	`result.add(a);`
	`}`
	`}`
	`finally`
	`{`
	`if(stm != null)`
	`stm.close();`
	`}`
	`return result;`
	`}`

Überprüfen ob ein Eintrag schon existiert	`public boolean stockRowAlreadyExists(Connection con, String datum, String zeitpunkt) throws SQLException{`
	`boolean result = false;`
	`PreparedStatement stm = null;`
	`ResultSet rs = null;`
	`try {`
	`String sql = "select count(*) from aktie where Datum = ? and Zeitpunkt = ?";`
	`stm = con.prepareStatement(sql);`
	`stm.setString(1, datum);`
	`stm.setString(2, zeitpunkt);`
	`rs = stm.executeQuery();`
	`if (rs.next()) {`
	`int anzahl = rs.getInt(1);`
	`result = anzahl == 1;`
	`}`
	`}`
	`finally {`
	`if (stm != null) {`
	`stm.close();`
	`}`
	`}`
	`return result;`
	`}`

Eine neue Aktie in die Aktien Liste eintragen	
Überprüfen ob ein Eintrag in der AktienListe bereits existiert	
ID einer Aktie beziehen	
Beziehen des Zeitpunktes an dem der Minimalwert an einem Tag auftrat	
Beendigen der Verbindung	

```java
public void newAktieInAktienListe(Connection con, aktienListe l) throws SQLException {
    String sql = "insert into aktienListe (symbol) values (?)";
    PreparedStatement stm = null;
    try {
        stm = con.prepareStatement(sql);
        stm.setString(1, l.getSymbol());
        stm.executeUpdate();
    }
    finally {
        if (stm != null)
            stm.close();
    }
}
```

```java
public boolean stockAlreadyExistsInAktienList (Connection con, String symbol) throws SQLException {
    boolean result = false;
    PreparedStatement stm = null;
    ResultSet rs = null;
    try {
        String sql = "select count(*) from aktienListe where symbol = ?";
        stm = con.prepareStatement(sql);
        stm.setString(1, symbol);
        rs = stm.executeQuery();
        if (rs.next()) {
            int anzahl = rs.getInt(1);
            result = anzahl == 1;
        }
    }
    finally {
        if (stm != null) {
            stm.close();
        }
    }
    return result;
}
```

```java
public int getIDfromStock(Connection con, String symbol) throws SQLException
    PreparedStatement stm = null;
    ResultSet rs = null;
    int result = 0;
    try {
        String sql = "select ID from aktienListe where Symbol = ?";
        stm = con.prepareStatement(sql);
        stm.setString(1, symbol);
        rs = stm.executeQuery();
        while(rs.next()) {
            result = rs.getInt(1);
        }
    }
    finally
    {
        if(stm != null)
            stm.close();
    }
    return result;
}
```

```java
public ArrayList<String> getTimeofMinofDay(Connection con, int symbol) throws SQLException {
    PreparedStatement stm = null;
    ResultSet rs = null;
    String zeitpunkt = "";
    String minStockValue ="";
    ArrayList<String> result = new ArrayList<String>();
    try {
        String sql = "select Zeitpunkt,min(StockValue) as minVal from aktie where Symbol = ? group by Datum";
        stm = con.prepareStatement(sql);
        stm.setInt(1, symbol);
        rs = stm.executeQuery();
        while(rs.next()) {
            zeitpunkt = rs.getString(1);
            minStockValue = rs.getString(2);
            result.add(zeitpunkt);
        }
    }
    finally
    {
        if(stm != null)
            stm.close();
    }
    return result;
}
```

```java
public void releaseConnection (Connection con)
        throws SQLException {
    if (con != null)
        con.close();
}
```

Nun da wir eine Verbindung zur Datenbank und eine Variation von wichtigen Methoden, die uns den Zugriff auf die Datenbank erleichtern haben, werden nun die Daten unseres Stock Array in die Datenbank geladen.

```
static void pushToDatabase(Connection con, aktie[] a, String[][] stock, DBmanager db) throws SQLException {
    int Passcounter = 0;
    int Errorcounter = 0;
    if (db.stockAlreadyExistsInAktienList(con, download.stockSymbol) == false ){
        aktienListe l = new aktienListe (1, download.stockSymbol);
        db.newAktieInAktienListe(con, l);
    }
    for (int Rowc = 0; Rowc < combination.getLengthOfNeededArray(); Rowc++) {
        if (db.stockRowAlreadyExists(con, stock[Rowc][0], stock[Rowc][1]) == false) {
            a[Rowc] = new aktie(db.getIDfromStock(con, download.stockSymbol),stock[Rowc][0],stock[Rowc][1],stock[Rowc][4]);
            DBmanager.saveNewSpecificStockValue(con,a[Rowc]);
            Passcounter++;
        }
        else {
            Errorcounter++;
        }
    }
    System.out.println("Es wurden " + Passcounter + " neue Einträge eingetragen " + Errorcounter + " Einträge waren schon vorhanden");
}
```

Main um daten zu erhalten und in die Datenbank zuladen:

```
public static void main(String[] args) throws ClassNotFoundException, SQLException {
    // TODO Auto-generated method stub
    download.deleteCurrentFiles();
    DBmanager db = new DBmanager();
    Connection con = db.getConnection();
    download.deleteCurrentFiles();
    download.download();
    combination.combine();
    String [][] stock = new String[combination.getLengthOfNeededArray()][combination.getCols()];
    aktie[] a = new aktie[combination.getLengthOfNeededArray()];
    combination.fillStockArray(stock);
    combination.pushToDatabase(con, a, stock,db);
```

3. Aufarbeitung der Daten für Basisprojekt

Nun sollte noch herausgefunden werden zu welchem Zeitpunkt welche Häufigkeit für den Minimalwert einer Aktie besteht. Dies geschieht durch folgenden Code:

```
ArrayList<String> minimalValueTimeList = db.getTimeofMinofDay(con,1);
HashMap<String,Integer> frequenzyMap = new HashMap<String,Integer>();
for ( String str : minimalValueTimeList) {
    Integer i = frequenzyMap.get( str);
    if ( i == null) {
        i = new Integer( 1);
    } else {
        i = new Integer( i.intValue()+1);
    }
    frequenzyMap.put( str, i);
}
for ( String key : frequenzyMap.keySet()) {
    System.out.println( key + ": " + frequenzyMap.get( key));
}
```

Die Häufigkeit des Minimalwerts wird dann stand heute 02.12.2020 auf der Konsole ausgegeben:

```
08:00:00: 1     16:30:00: 3     06:15:00: 1
13:15:00: 4     17:00:00: 37    16:15:00: 2
17:30:00: 12    08:45:00: 2     18:15:00: 20
19:45:00: 66    18:45:00: 19    19:00:00: 25
18:00:00: 20    20:00:00: 136   08:30:00: 1
19:30:00: 37    19:15:00: 37    17:45:00: 19
05:00:00: 1     09:15:00: 1     07:45:00: 5
17:15:00: 11    09:00:00: 2     08:15:00: 2
16:45:00: 19    06:30:00: 2     18:30:00: 21
```

Dies ist der aktuelle Stand des Bonus-Teils. Der Hauptteil wurde bereits in der Schule vorgezeigt und muss nun noch kombiniert werden.

4. Basisprojekt (nicht kombiniert)

Dieser Teil des Projektes ist bereits veraltet da er nicht mit API und Datenbank programmiert worden ist sondern rein eine CSV zu Geschichte des ETFs TLT verwendet. Den Download der CSV werde ich hier nicht mehr zeigen er kann unter folgendem Link gefunden werden:

https://github.com/flostaudacher/ETF_TLT_Backtesting/blob/master/src/Import.java

Grundsätzlich wird in dieser Klasse ein 2D String Array Befüllt. Dies sieht dann aus wie eine Tabelle, die erste Dimension stellt die Zeilen dar die Zweite stellt die Spalten dar.

Dieser Array wird dann um eine weitere Spalte erweitert, diese Spalte stellt dar um welchen Wochentag es sich am jeweiligen Datum handelt. Dies wird in der Classe Friday Check behandelt.

```
public class Fridaycheck {
    static String [] days = {"Mo","Di","Mi","Do","Fr"};
    public static String checkForFriday(int Rowc) throws ParseException {
        String s = toSimpleDate(Import.arr[Rowc][0]); ;
        if (s.equals(days[4])) {
            return days[4];
        }
        for (int dayCounter = 0; dayCounter <= days.length; dayCounter++) {
            if (s.equals(days[dayCounter])) {
                return days[dayCounter];
            }
        }
        return "";
    }
    private static String toSimpleDate(String string) throws ParseException {
        Date date=new SimpleDateFormat("yyyy-MM-dd").parse(string);
        SimpleDateFormat simpleDateformat = new SimpleDateFormat("E");
        return simpleDateformat.format(date);
    }
```

Außerdem wird herausgefunden ob eine Tag börsenfrei war, da an einem Wochenende dem ein Börsenfreier Tag entweder folgt oder zuvorkommt keine Transaktion durchgeführt werden soll. Ob diese Bedingung greift wird durch folgende Methode klar.

```
public static void börsenfrei() {
    int zaehlerWoMaSein=0;
    do {
        if (Import.arr[zaehlerWoMaSein][7].equals("Do")) {
            if (Import.arr[zaehlerWoMaSein+1][7].equals("Mo")) {
                Import.arr[zaehlerWoMaSein][7] = "Fr";
            }
        }
        zaehlerWoMaSein++;
    }while (Import.arr.length > zaehlerWoMaSein);


}
```

Als nächstes wird eine Klasse implementiert, die jegliche Methoden für eine Transaktion beinhaltet, das heißt die Methoden für einen Kauf und einen Verkauf von Positionen.

```
static void buyStock(String arr) {
    double closingValueBuy = Double.parseDouble(arr); // ohne gebühren
    //double closingValueBuy = (Double.parseDouble(arr))*1.01; // mit gebühren.
    amountOfStockAvailable = (int) ((availabeMoney - 5) / (closingValueBuy)); // formel wo die Gebühren hin gehören
    buyValue = closingValueBuy;
    getValueAtBuy();
    availabeMoney = availabeMoney - amountOfStockAvailable * (closingValueBuy);
}


static void SellStock(String arr) {
    double closingValueSell = Double.parseDouble(arr); //without gebühren
    //double closingValueSell = (Double.parseDouble(arr))*0.99;//with
    availabeMoney = availabeMoney + amountOfStockAvailable * (closingValueSell);
    sellValue = closingValueSell;
    getValueAtSell();
    amountOfStockAvailable = 0;

}
```

Als nächstes Erstellen wir eine Flowpane in welches 3 Objekte Geladen werden.

4.1 AreaChart unseres Depot Values nach Anzahl der Trades

Als erstes wird in der Klasse vizualization.java ein Area Chart erstellt:

```
public static AreaChart<Number, Number> areachart() {

    NumberAxis xAxis= new NumberAxis();
    xAxis.setLabel("Years passed");
    NumberAxis yAxis= new NumberAxis();
    yAxis.setLabel("Depot Value");

    AreaChart<Number, Number> areaChart = new AreaChart<Number,Number>(xAxis,yAxis);
    areaChart.setTitle("depot value in relation to times traded");

    XYChart.Series<Number, Number> data = new XYChart.Series<Number, Number>();
    data.setName("Backtesting result if you try the tactic with 10k in the year 2010");

    vizualizingMethods.create(data);
    areaChart.setCreateSymbols(false);
    areaChart.getData().add(data);
    return areaChart;
}
```

Dieser Area Chart wird dann mit vizualizationMethods.create(data) mit Daten befüllt

Den Code dafür finden wir in der Klasse vizualizationMethods.

Hier wird die geplante Transaktion durchgeführt und nach jedem Trade in den Areachart eingetragen

```
public static void createTheChart(Series<Number, Number> data) {

    for (int Rowc = 1; Rowc < Import.anzahlZeilen; Rowc++) {
        if (Import.arr[Rowc][7].equals("Fr")) {
            tradingClass.buyStock(Import.arr[Rowc-1][4]);
            int positions = tradingClass.amountOfStockAvailable;
            tradingClass.SellStock(Import.arr[Rowc][4]);
            tradingCounter++;
            if ((tradingClass.getValueAtBuy() - tradingClass.getValueAtSell()) < 0) {
                PN[0]++;
            }
            else {
                PN[1]++;
            }
            data.getData().add(new XYChart.Data<Number, Number>( (double)tradingCounter / 50,tradingClass.availabeMoney));
        }
    }
}
```

Als Resultat wird ein befüllter Area Chart in das Flowpane eingetragen.

4.2 Barchart für die Chance auf einen Erfolgreichen Trade

Das Zweite Objekt das in das Flowpane eingetragen wird ist eine Visualisierung der Wahrscheinlichkeit auf einen positiven/ negativen Trade anhand eines Barcharts.
Das heißt wir erstellen einen Barchart mit zwei Säulen:

```
public static BarChart<String, Number> Barchart() {
    CategoryAxis xAxis = new CategoryAxis();
    xAxis.setCategories(FXCollections.<String>
    observableArrayList(Arrays.asList("positiv", "negativ")));
    xAxis.setLabel("Chance");

    NumberAxis yAxis= new NumberAxis();
    yAxis.setLabel("Chance for positiv/negativ Trade");

    BarChart<String, Number> barChart = new BarChart<>(xAxis,yAxis);
    barChart.setTitle("Positiv/Negativ");

    XYChart.Series<Number, Number> data = new XYChart.Series<Number, Number>();
    data.setName("Backtesting result if you try the tactic with 10k in the year 2010");


    barChart.getData().addAll(barChartMethods.create(new XYChart.Series<String, Number>(), "positiv"));
    barChart.getData().addAll(barChartMethods.create(new XYChart.Series<String, Number>(), "negativ"));

    barChart.setCategoryGap(20);

    return barChart;
}
```

Dieser wird dann durch die Methode create in der Klasse barChartMethods mit den jeweiligen Daten befüllt.

```
static Series<String, Number> create(Series<String, Number> s, String p) {
    s.setName("" + p);
    if (p.equals("positiv")) {
        s.getData().add(new XYChart.Data<>(p,(vizualizingMethods.PN[0] / vizualizingMethods.tradingCounter) * 100.0));
    }
    if (p.equals("negativ")) {
        s.getData().add(new XYChart.Data<>(p,(vizualizingMethods.PN[1] / vizualizingMethods.tradingCounter) * 100.0));
    }
    return s;
}
```

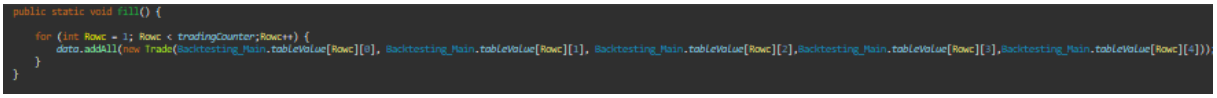
4.3 Tabelle für Trading Historie

Als drittes und letztes Objekt das in das Flowpane eingetragen wird handelt es sich um eine Tabelle die die Historie unserer Trades darstellt.

Als erstes brauchen wir eine Methode die unsere Trades errechnet.

```
public static void calc() {
    for (int Rowc = 1; Rowc < Import.anzahlZeilen; Rowc++) {
        if (Import.arr[Rowc][7].equals("Fr")) {
            tradingClass.buyStock(Import.arr[Rowc-1][4]);
            int positions = tradingClass.amountOfStockAvailable;
            tradingClass.SellStock(Import.arr[Rowc][4]);
            tradingCounter++;
            if ((tradingClass.getValueAtBuy() - tradingClass.getValueAtSell()) < 0) {
                Backtesting_Main.tableValue[tradingCounter][0] = Import.arr[Rowc][0]; //Datum in TabellenArray
                Backtesting_Main.tableValue[tradingCounter][1] = String.valueOf(positions);
                Backtesting_Main.tableValue[tradingCounter][2] = String.valueOf(f.format(tradingClass.buyValue));
                Backtesting_Main.tableValue[tradingCounter][3] = String.valueOf(f.format(tradingClass.sellValue));
                Backtesting_Main.tableValue[tradingCounter][4] = "positiver Trade";
            }
            else {
                Backtesting_Main.tableValue[tradingCounter][0] = Import.arr[Rowc][0]; //Datum in TabellenArray
                Backtesting_Main.tableValue[tradingCounter][1] = String.valueOf(positions);
                Backtesting_Main.tableValue[tradingCounter][2] = String.valueOf(f.format(tradingClass.buyValue));
                Backtesting_Main.tableValue[tradingCounter][3] = String.valueOf(f.format(tradingClass.sellValue));
                Backtesting_Main.tableValue[tradingCounter][4] = "negativer Trade";
            }
        }
    }
}
```

Danach müssen wir unsere Daten in der Methode fill zu Data hinzufügen.



```
public static void fill() {

    for (int Rowc = 1; Rowc < tradingCounter;Rowc++) {
        data.addAll(new Trade(Backtesting_Main.tableValue[Rowc][0], Backtesting_Main.tableValue[Rowc][1], Backtesting_Main.tableValue[Rowc][2],Backtesting_Main.tableValue[Rowc][3],Backtesting_Main.tableValue[Rowc][4]));
    }
}
```

Als aller letztes in Punkt 4.3 müssen wir nun diese Daten in die Tabelle eintragen und die Tabelle eine Vbox laden die dann in das Flowpane übergeben wird.

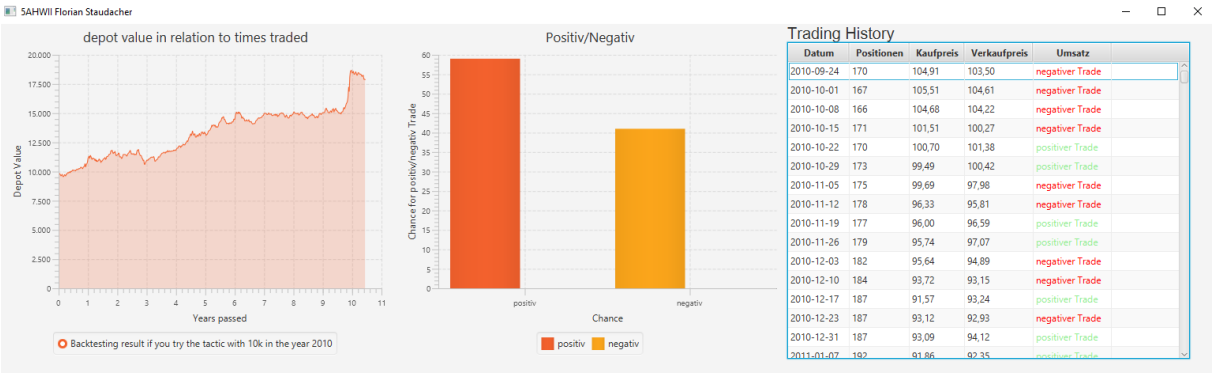
```
TableColumn Datum = new TableColumn("Datum");
Datum.impl_setReorderable(false);
Datum.setCellValueFactory(
        new PropertyValueFactory<Trade ,String>("Datum")
        );
TableColumn Positionen = new TableColumn("Positionen");
Positionen.impl_setReorderable(false);
Positionen.setCellValueFactory(
        new PropertyValueFactory<Trade ,String>("Positionen")
        );
TableColumn Kaufpreis = new TableColumn("Kaufpreis");

Kaufpreis.impl_setReorderable(false);
Kaufpreis.setCellValueFactory(
        new PropertyValueFactory<Trade ,String>("Kaufpreis")
        );
TableColumn Verkaufspreis = new TableColumn("Verkaufpreis");
Verkaufspreis.impl_setReorderable(false);
Verkaufspreis.setCellValueFactory(
        new PropertyValueFactory<Trade ,String>("Verkaufpreis")
        );
TableColumn Umsatz = new TableColumn("Umsatz");
Umsatz.impl_setReorderable(false);
Umsatz.setCellValueFactory(
        new PropertyValueFactory<Trade ,String>("Umsatz")
        );
Umsatz.setCellFactory(column -> {
    return new TableCell<Trade, String>() {
        @Override
        protected void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
            if (item == null || empty) {
                setText("");
                setStyle("");
            } else {
                if (item.equals("positiver Trade")) {
                    setTextFill(Color.LIGHTGREEN);
                    setText("positiver Trade");
                } else {
                    setTextFill(Color.RED);
                    setText("negativer Trade");
                    setStyle("");
                }
            }
        }
    };
});

table.setItems(data);
table.getColumns().addAll(Datum, Positionen, Kaufpreis, Verkaufspreis,Umsatz);
table.minWidth(510);
final VBox vbox = new VBox();
vbox.setMinWidth(510);
vbox.getChildren().addAll(label, table);

return vbox;
```

Die Visualisierung des Projekts sieht dann so aus:



Falls fragen zum Code bestehen entweder mich fragen oder im jeweiligen Repository recherchieren da nicht der komplette Code in diesem File ist.

Vielen Dank dass Sie bis zum Schluss gelesen haben!

Mit freundlichen Grüßen

Florian Staudacher