Super Simple Stocks

Version 1.1

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Comments/Changes from Previous Version |
| 1.0 | Javier Fernandez | 5th May 2017 | Creation of document. Main sections added. |
| 1.1 | Javier Fernandez | 7th May 2017 | Appendixes added. |

Contents

[Version History 3](#_Toc481957742)

[Introduction 4](#_Toc481957743)

[Purpose of Document 4](#_Toc481957744)

[Documentation Basis 4](#_Toc481957745)

[Scope 4](#_Toc481957746)

[Formulae 5](#_Toc481957747)

[Common dividend yield 5](#_Toc481957748)

[Preferred dividend yield 5](#_Toc481957749)

[P/E Ratio 5](#_Toc481957750)

[Volume-weighted average price 6](#_Toc481957751)

[Geometric Mean 6](#_Toc481957752)

[Design 7](#_Toc481957753)

[Appendix A: Classes 8](#_Toc481957754)

[Stock class 8](#_Toc481957755)

[Trade class 10](#_Toc481957756)

[StockType enum 11](#_Toc481957757)

[TradeBuySell enum 11](#_Toc481957758)

[StockTradeControl class 12](#_Toc481957759)

[IStockTradeControl interface 15](#_Toc481957760)

[GBCEMain class 16](#_Toc481957761)

[StockTest methods 19](#_Toc481957762)

# Introduction

## Purpose of Document

This solution has been developed in C# using Microsoft Visual Studio Professional 2013 (Version 12.0.31101.00 Update 4). It contains two projects:

* **JavierFernandez0517\_Assignment** – Class Library (C#)
* **Test\_JavierFernandez0517\_Assignment** – Unit Test Project (C#)

This solution has been sent to Jamie Milligan via email, but it can also be found on GitHub: <https://github.com/JavierJimenez83/JavierFernandez0517_Assignment>

## Documentation Basis

* **Super Simple Stocks.docx**

****

# Scope

#### In Scope

* **Formulae**: Dividend yield, PE Ratio, Volume Weighted Stock Price, Geometric Mean
* Written in one of these languages: Java, **C#**, C++, Python
* Analysis, design and implementation of the **solution**
* **Test unit project**
* **Documentation**

#### Out of Scope

* Graphical User Interface
* Database

# Formulae

## Common dividend yield

A **dividend** is a distribution of a portion of a company's earnings. The dividend rate may be quoted in terms of a percent of the current market price, which is referred to as the **common** **dividend yield**:

**Common stock** is a security that represents ownership in a corporation. Holders of common stock exercise control by electing a board of directors and voting on corporate policy.

Preferred dividend yield

The fixed dividend multiplied by the par value equates to the total annual preferred dividend.   
The **preferred dividend yield** is equal to the annual preferred dividend divided by the current market price.

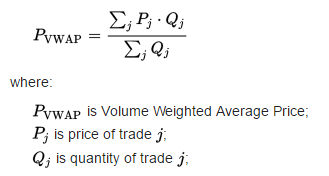
**Preferred stock** is a class of ownership in a corporation that has a higher claim on its assets and earnings than common stock. Preferred shares usually do not carry voting rights.

## P/E Ratio

The price-earnings ratio (P/E ratio) is the ratio for valuing a company that measures its current [share](http://www.investopedia.com/terms/s/shares.asp) price relative to its [per-share earnings](http://www.investopedia.com/terms/e/eps.asp). The P/E ratio can be calculated as:

## Volume-weighted average price

**Volume** **weighted average price (VWAP)** is the ratio of the value traded to total [volume](https://en.wikipedia.org/wiki/Volume_(finance)) traded over a particular time horizon (usually one day). It is a measure of the average price at which a [stock](https://en.wikipedia.org/wiki/Stock) is traded over the trading horizon. VWAP is calculated using the following formula:

{\displaystyle j};

{\displaystyle j}*j* is each individual trade that takes place over the defined period of time, excluding cross trades and basket cross trades.[[2]](https://en.wikipedia.org/wiki/Volume-weighted_average_price#cite_note-investopedia-2)

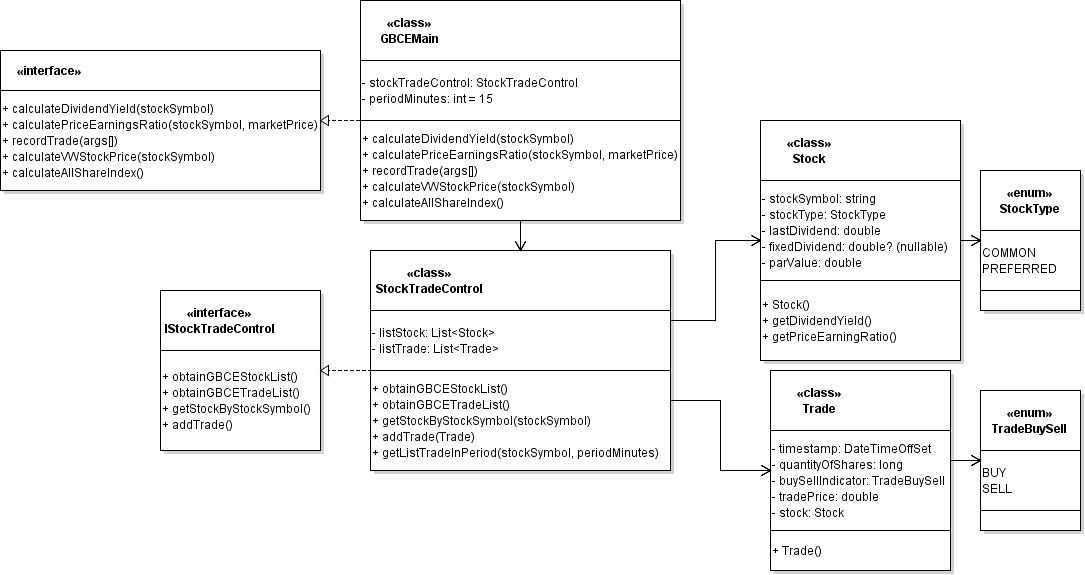
## Geometric Mean

The geometric mean is mainly used to evaluate the data covering several orders of magnitude, the ratios of a company, the percentage changes of a company's fundamentals or other data sets bound by zero. Geometric means should not be used to find the average of a data set if it covers a very small range or if the data set is highly skewed.

**Geometric mean** can be calculated as:

# Design

# 



# Appendix A: Classes

## Stock class

public class Stock

{

// Stock class properties

private string stockSymbol = null;

private StockType stockType = StockType.COMMON;

private double lastDividend = 0.0;

private double? fixedDividend = 0.0;

private double parValue = 0.0;

// Stock class getters-setters

public string GetStockSymbol(){ return stockSymbol; }

public void SetStockSymbol(string value) { this.stockSymbol = value; }

public StockType GetStockType() { return stockType; }

public void SetStockType(StockType value) { this.stockType = value; }

public double GetLastDividend() { return lastDividend; }

public void SetLastDividend(double value) { this.lastDividend = value; }

public double? GetFixedDividend() { return fixedDividend; }

public void SetFixedDividend(double? value) { this.fixedDividend = value; }

public double GetParValue() { return parValue; }

public void SetParValue(double value) { this.parValue = value; }

/// <summary>

/// This is the Stock class constructor

/// </summary>

/// <param name="sto">Stock Symbol (to identify stock)</param>

/// <param name="typ">Stock Type (Common or Preferred)</param>

/// <param name="las">Last Dividend</param>

/// <param name="fix">Fixed Dividend</param>

/// <param name="par">Par Value</param>

public Stock(string sto, StockType typ, double las, double? fix, double par)

{

this.SetStockSymbol(sto);

this.SetStockType(typ);

this.SetLastDividend(las);

this.SetFixedDividend(fix);

this.SetParValue(par);

}

/// <summary>

/// This method calculates the dividend yield for a specific Stock (both COMMON and PREFERRED)

/// </summary>

/// <param name="marketPrice">Market price</param>

/// <returns>This method returns the dividend yield; it returns -1.0 if and only if market price is less or equal than 0.0</returns>

public double getDividendYield(double marketPrice)

{

if (marketPrice > 0.0)

{

// If Stock Type is COMMON

if(this.GetStockType() == StockType.COMMON)

{

return this.GetLastDividend() / marketPrice ;

}

// If Stock Type is PREFERRED

else

{

// fixedDividend is a nullable double; try and catch block throws an exception when fixedDividend is null

try

{

// Casting to double

double fixDiv = (double)this.GetFixedDividend();

return (fixDiv \* this.GetParValue()) / marketPrice;

}

catch

{

return -1.0;

}

}

}

else

{

return -1.0;

}

}

/// <summary>

/// This method calculates the price-earnings ratio for a specific Stock (both COMMON and PREFERRED, through the DividendYield method)

/// </summary>

/// <param name="marketPrice">Market price</param>

/// <returns>This method returns the price-earnings ratio; it returns -1.0 if market price is less or equal than 0.0 and dividend yield is not equal to -1.0 </returns>

public double getPriceEarningsRatio(double marketPrice)

{

double dividendYield = getDividendYield(marketPrice);

if (marketPrice > 0.0 && dividendYield != -1.0)

{

return marketPrice / dividendYield;

}

else

{

return -1.0;

}

}

}

## Trade class

public class Trade

{

// Trade class properties

private DateTimeOffset timestamp = DateTimeOffset.Now; /\* DateTimeOffset has been chosen over DateTime because it identifies a single point in time unambiguously\*/

private long quantityOfShares = 0;

private TradeBuySell buySellIndicator = TradeBuySell.BUY;

private double tradePrice = 0.0;

private Stock stock = null;

// Trade class getters-setters

public DateTimeOffset GetTimestamp() { return timestamp; }

public void SetTimestamp(DateTimeOffset value) { this.timestamp = value; }

public long GetQuantityOfShares() { return quantityOfShares; }

public void SetQuantityOfShares(long value) { this.quantityOfShares = value; }

public TradeBuySell GetBuySellIndicator() { return buySellIndicator; }

public void SetBuySellIndicator(TradeBuySell value) { this.buySellIndicator = value; }

public double GetTradePrice() { return tradePrice; }

public void SetTradePrice(double value) { this.tradePrice = value; }

public Stock GetStock() { return stock; }

public void SetStock(Stock value) { this.stock = value; }

/// <summary>

/// This is the Trade class constructor

/// </summary>

/// <param name="tim">Timestamp</param>

/// <param name="qua">Quantity of shares</param>

/// <param name="tbs">Trade buy or sell indicator</param>

/// <param name="pri">Trade price</param>

public Trade(DateTimeOffset tim, long qua, TradeBuySell tbs, double pri, Stock sto)

{

this.SetTimestamp(tim);

this.SetQuantityOfShares(qua);

this.SetBuySellIndicator(tbs);

this.SetTradePrice(pri);

this.SetStock(sto);

}

}

## StockType enum

public enum StockType

{

COMMON,

PREFERRED

}

## TradeBuySell enum

public enum TradeBuySell

{

BUY,

SELL

}

## StockTradeControl class

public class StockTradeControl : IStockTradeControl

{

// StockTradeControl class properties

List<Stock> listStock = new List<Stock>(); // = obtainGBCEStockList();

List<Trade> listTrade = new List<Trade>();

// StockTradeControl class getters-setters

public List<Stock> GetListStock() { return listStock; }

public void SetListStock(List<Stock> value) { this.listStock = value; }

public List<Trade> GetListTrade() { return listTrade; }

public void SetListTrade(List<Trade> value) { this.listTrade = value; }

/// <summary>

/// This method return the list of stocks (Global Beverage Corporation Exchange)

/// THIS SHOULD BE RETRIEVED FROM A DATABASE. HARDCODED AS NO DATABASE IS REQUIRED.

/// </summary>

/// <returns>List of stocks from the Global Beverage Corporation Exchange</returns>

public List<Stock> obtainGBCEStockList()

{

List<Stock> listStock = new List<Stock>();

listStock.Add(new Stock("TEA", StockType.COMMON, 0.0, null, 100.0));

listStock.Add(new Stock("POP", StockType.COMMON, 8.0, null, 100.0));

listStock.Add(new Stock("ALE", StockType.COMMON, 23.0, null, 100.0));

listStock.Add(new Stock("GIN", StockType.PREFERRED, 8.0, 0.02, 100.0));

listStock.Add(new Stock("JOE", StockType.COMMON, 13.0, null, 100.0));

return listStock;

}

/// <summary>

/// This method return the list of trades (Global Beverage Corporation Exchange)

/// THIS SHOULD BE RETRIEVED FROM A DATABASE.

/// </summary>

/// <returns>List of stocks from the Global Beverage Corporation Exchange</returns>

public List<Trade> obtainGBCETradeList()

{

List<Trade> listTrade = new List<Trade>();

/\*

\* DATABASE QUERYING

\*/

return listTrade;

}

/// <summary>

/// This method returns a Stock with a specific stock symbol

/// </summary>

/// <param name="stockSymbol">Stock symbol (to identify stock)</param>

/// <returns>This returns a stock; it returns null if there is not a stock with such stock symbol</returns>

public Stock getStockByStockSymbol(string stockSymbol)

{

try

{

Stock stock = this.GetListStock().Find(x => x.GetStockSymbol() == stockSymbol);

return stock;

}

catch

{

return null;

}

}

/// <summary>

/// This method adds a trade to the list of trades (Global Beverage Corporation Exchange)

/// </summary>

/// <param name="trade">Trade</param>

/// <returns>It returns 'true' if the trade has been added successfully; it returns 'false' if the trade has not been added</returns>

public bool addTrade(Trade trade)

{

try

{

this.GetListTrade().Add(trade);

return true;

}

catch

{

return false;

}

}

/// <summary>

/// Get all the trade of a particular stock that have been traded in a given period

/// </summary>

/// <param name="stockSymbol">Stock symbol (to identify stock)</param>

/// <param name="periodMinutes">Given period (15 minutes from requirements, but this can be modified in GBCEMain class)</param>

/// <returns></returns>

public List<Trade> getListTradeInPeriod(string stockSymbol, int periodMinutes)

{

List<Trade> listTradeInPeriod = new List<Trade>();

foreach(Trade x in this.GetListTrade())

{

try

{

// Geting stock symbol of that trade

string tradeStockSymbol = x.GetStock().GetStockSymbol();

bool isCorrectStock = tradeStockSymbol == stockSymbol ? true : false;

// Comparing DateTime now to time trade happened

DateTimeOffset timeNow = DateTimeOffset.Now;

DateTimeOffset timeTrade = x.GetTimestamp();

int minutes = timeNow.Subtract(timeTrade).Minutes;

bool isWithinPeriod = minutes <= periodMinutes ? true : false;

// Adding trade to list if: is correct stock AND has been traded in given period

if (isCorrectStock && isWithinPeriod)

{

listTradeInPeriod.Add(x);

}

}

catch

{

}

}

return listTradeInPeriod;

}

}

## IStockTradeControl interface

public interface IStockTradeControl

{

/// <summary>

/// List of stocks (Global Beverage Corporation Exchange)

/// </summary>

/// <returns>List of stocks (Global Beverage Corporation Exchange)</returns>

public List<Stock> obtainGBCEStockList();

/// <summary>

/// List of trades (Global Beverage Corporation Exchange)

/// </summary>

/// <returns>List of trades (Global Beverage Corporation Exchange)</returns>

public List<Trade> obtainGBCETradeList();

/// <summary>

/// This method returns a Stock with a specific stock symbol

/// </summary>

/// <param name="stockSymbol">Stock symbol (to identify stock)</param>

/// <returns>This returns a stock; it returns null if there is not a stock with such stock symbol</returns>

public Stock getStockByStockSymbol(string stockSymbol);

/// <summary>

/// This method adds a trade to the list of trades (Global Beverage Corporation Exchange)

/// </summary>

/// <param name="trade">Trade</param>

/// <returns>It returns 'true' if the trade has been added successfully; it returns 'false' if the trade has not been added</returns>

public bool addTrade(Trade trade);

}

## GBCEMain class

/// <summary>

/// Main entry point of this library

/// </summary>

public class GBCEMain

{

// GBCEMain class property

private StockTradeControl stockTradeControl = new StockTradeControl();

private static int periodMinutes = 15;

// GBCEMain class getter

public StockTradeControl GetStockTradeControl() { return stockTradeControl; }

/// <summary>

/// This method returns the dividend yield of a specific stock. Market price is an input (requirement)

/// </summary>

/// <param name="stockSymbol">Stock symbol (to identify stock)</param>

/// <param name="marketPrice">Market Price</param>

/// <returns>Dividend yield of a specific stock</returns>

public double calculateDividendYield(string stockSymbol, double marketPrice)

{

try

{

// Get stock

Stock stock = this.GetStockTradeControl().getStockByStockSymbol(stockSymbol);

// Get dividend yield

return stock.getDividendYield(marketPrice);

}

catch

{

return -1.0;

}

}

/// <summary>

/// This method returns the price-earning ratio of a specific stock. Market price is an input (requirement)

/// </summary>

/// <param name="stockSymbol">Stock symbol (to identify stock)</param>

/// <param name="marketPrice">Market Price</param>

/// <returns>Price-earning ratio of a specific stock</returns>

public double calculatePriceEarningsRatio(string stockSymbol, double marketPrice)

{

try

{

// Get stock

Stock stock = this.GetStockTradeControl().getStockByStockSymbol(stockSymbol);

// Get dividend yield

return stock.getPriceEarningsRatio(marketPrice);

}

catch

{

return -1.0;

}

}

/// <summary>

/// This method records a new trade

/// </summary>

/// <param name="timestamp">Timestamp of trade</param>

/// <param name="quantityOfShares">Quantity of shares</param>

/// <param name="tradeBuySell">Buy or Sell indicator</param>

/// <param name="tradePrice">Trade price</param>

/// <param name="stockSymbol">Stock Symbol (to identify stock)</param>

/// <returns>It returns 'true' if the trade has been added successfully; it returns 'false' if the trade has not been added</returns>

public bool recordTrade(DateTime timestamp, long quantityOfShares, TradeBuySell tradeBuySell, double tradePrice, string stockSymbol)

{

try

{

// Get stock

Stock stock = this.GetStockTradeControl().getStockByStockSymbol(stockSymbol);

// Create trade instance

Trade trade = new Trade(timestamp, quantityOfShares, tradeBuySell, tradePrice, stock);

// Record trade

return this.stockTradeControl.addTrade(trade);

}

catch

{

return false;

}

}

/// <summary>

/// This method returns the volume weighted stock price of trades for a specific stock in a give period

/// (15 minutes from requirements, but this can be modified in GBCEMain class)

/// </summary>

/// <param name="stockSymbol">Stock Symbol (to identify stock)</param>

/// <returns>Volume Weighted Stock Price</returns>

public double calculateVWStockPrice(string stockSymbol)

{

try

{

// Get list of trades in a give period (15 minutes from requirements, but this can be modified in GBCEMain class)

List<Trade> listTradeInPeriod = this.GetStockTradeControl().getListTradeInPeriod(stockSymbol, periodMinutes);

// Get Summatory(TradePrice[i] \* QuantityShares[i]) and Summatory(QuantityShares[i]) for all those trades

double summPriceTimesQuant = 0.0;

double summQuant = 0.0;

foreach(Trade x in listTradeInPeriod)

{

summPriceTimesQuant += x.GetTradePrice() \* x.GetQuantityOfShares();

summQuant += x.GetQuantityOfShares();

}

// Get Volume Weighted Stock Price

return summPriceTimesQuant / summQuant;

}

catch

{

return -1.0;

}

}

/// <summary>

/// This method returns the Global Beverage Corporation Exchange All Share Index

/// </summary>

/// <returns>Global Beverage Corporation Exchange All Share Index</returns>

public double calculateAllShareIndex()

{

double GBCEAllIndex = 0.0;

// Get list of stocks

List<Stock> listStock = stockTradeControl.GetListStock();

// TO BE FINISHED

return GBCEAllIndex;

}

}

## StockTest methods