DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

# One Basket Finance Project

## Computer Science II Project

**Jared Ott and Ryan Wallace**

**3/3/2017**
**Version 0.3**

This document contains up to date information on the application for One Basket Finance. The goal of the application is to develop an effective application by using Object-Oriented Programming that can meet all of One Basket Finance's functional needs.

## Revision History

| Version | Description of Change(s) | Author(s) | Date |
|---|---|---|---|
| 0.1 | Initial draft of this design document | Ryan Wallace/Jared Ott | 2017/02/01 |
| 0.2 | First revision<br>Added XMLable and JSONable interfaces<br>Added Portfolio Class and description<br>Included more definitions<br>Included more abbreviations<br>Were more descriptive | Jared Ott/Ryan Wallace | 2017/15/02 |
| 0.3 | Revised Class information<br>Removed XMLable and JSONable interfaces<br>Added robust database design | Ryan Wallace/Jared Ott | 2017/03/01 |

## Contents

# Introduction

One Basket Finance needs a better way to keep track of their assets, portfolios, and the people that own and manage them. It is the job of Task Force .JAR to create a program that makes this task easy and automated for our client. At this phase in the production, the program is centered around class construction and implementation, as well as outputting the objects in human readable formats. The end goal of the production is to have a database to easily and efficiently keep track of the assets.

## Purpose of this Document

The purpose of this document is to effectively and clearly communicate how the program works and is designed at a high level. This document provides basic information about the program and it's working parts.

## 1.2 Scope of the Project

The task is to replace One Basket Finance's previous financial application. This involves reading in the data from the previous program and storing it in a database. Furthermore, the goal is to create methods and functions that aid in accessing the data, changing it, and performing other operations that return valuable information to the operator. Our plain is to automate the creation and storage of assets and portfolios and all that acts upon them.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- SEC Identifier Number – Takes the form of sec###. This number is used to identify and verify brokers.
- Quarterly Dividend – The money dealt out to shareholders of a company every fiscal quarter.
- Base Rate of Return – The gain or loss on an investment over a period of time.
- Beta Measure – The measure of relative risk for highly traded commodities I.e. stocks. It measures the return of a certain stock against the return of the overall market (usually an index like the S&P 500) using the Covariance (asset, market) divided by the variance of (market).
- Omega Measure – Assesses the risk of private investments. For the use of this project we will represent final risk as a 0 or 1 with 0 being low risk and 1 being risky.
- Object Oriented Programming – a programming paradigm that predominantly uses Objects.
- Object – a grouping of data in programming that can be implemented with multiple instances. This is the basis of Object-Oriented Programming.
- Instance – a single implementation of an object.
- Class – a way in Java to group information of a real-world object into a computable Object.
- Abstract Class – a way to group classes into an object and assign abstract functions to be implemented in the subclasses.
- Interface – a purely abstract class used to group classes into broad groups and still assign abstract functions to be implemented.

### 1.3.2 Abbreviations & Acronyms Alphabetically

- APR – Annual Percentage Rate
- FK – Foreign Key
- JDBC – Java Database Connectivity
- OOP – Object Oriented Programming
- PK – Primary Key
- SQL – Structured Query Language

## Overall Design Description

The project includes the implementation of multiple classes that represent real world objects, and it is the goal to provide an interface to interact with the objects and provide methods that help the user access the information about these objects. As previously stated, the goal of the project is to (eventually) store information into an optimized database for easy automated information retrieval. At this point in production, we have implemented class-based organization of elements provided using Object Oriented Programming. The design is simple yet efficient: take in and process a *.dat file, convert the information into classes, and output those classes into a human readable formatted *.txt file.

## 2.1 Alternative Design Options

Our solution to One Basket Finance's needs is centered around Object Oriented Programming. The best way to represent real life entities through programming is by using objects. The Asset class was chosen to be defined as an abstract class to prevent anyone from trying to instantiate an "asset," for there is no such thing. We could have made Asset an interface, however this would have taken away some of its functionality as an abstract class, such as defining a base constructor. Only the subclasses are actual real world entities, while they are all under the broad category of "asset." Furthermore, some variables that may be represented as integers were chosen to be represented with Strings because there may be varying input that include non-integers. For example, the ZIP code *could* have been represented with an integer, but it was chosen to represent zip code as a String because it could be the ZIP code +4 which usually includes a hyphen. This should create no problem because no integer operations should be performed on the ZIP code, and it will mainly be used to print out an address.

## 3. Detailed Component Description

Section 3 details the structure design of the program. This includes databases, classes, and their methods. This section's goal is to explicitly outline the program and allow easy identification of how the program's components fit together.

## 3.1 Database Design

Table 1 - Person

| | |
|---|---|
| - personID | integer [PK] |
| - lastName | varchar(255) |
| - firstName | varchar(255) |
| - addressID | integer [FK] (Table 3) |

Table 1.1 – BrokerStatus

| | | |
|---|---|---|
| - | brokerId | integer [PK] |
| - | brokerType | char |
| - | secId | varchar(10) |
| - | personId | integer [FK] (Table 1) |

Table 2 - Asset

| | | |
|---|---|---|
| - | assetID | integer [PK] |
| - | assetType | char |
| - | apr | float |
| - | quarterlyDividend | double |
| - | rateOfReturn | float |
| - | risk | float |
| - | symbol | varchar(5) |
| - | value | double |

Table 3 - Address

| | | |
|---|---|---|
| - | addressID | integer [PK] |
| - | streetAddress | varchar(255) |
| - | zipCode | varchar(10) |
| - | city | varchar(255) |
| - | stateID | integer [FK] (Table 3.1) |

Table 3.1 - State

| | | |
|---|---|---|
| - | stateID | integer [PK] |
| - | name | varchar(255) |
| - | countryID | integer [FK] (Table 3.2) |

Table 3.2 - Country

| | | |
|---|---|---|
| - | countryID | integer [PK] |
| - | name | varchar(75) |

Table 4 - Portfolio

| | | |
|---|---|---|
| - | portfolioID | integer [PK] |
| - | title | varchar(100) |

Table 5 - Email

| | | |
|---|---|---|
| - | emailID | integer [PK] |
| - | address | varchar(255) |

Join Table 1 - AssetPortfolio

| | | |
|---|---|---|
| - | assetPortfolioID | integer [PK] |
| - | number | float |
| - | assetID | integer [FK] (Table 2) |
| - | portfolioID | integer [FK] (Table 4) |

Join Table 2 - PersonPortfolio

| | | |
|---|---|---|
| - | personPortfolioID | integer [PK] |

|  |  |
|---|---|
| - ownerID | integer [FK] (Table 1) |
| - brokerID | integer [FK] (Table 1) |
| - beneficiaryID | integer [FK] (Table 1) |
| - portfolioID | integer [FK] (Table 4) |

Join Table 3 - PersonEmail

|  |  |
|---|---|
| - personEmailID | integer [PK] |
| - personID | integer [FK] (Table 1) |
| - emailID | integer [FK] (Table 5) |

### 3.1.1 Component Testing Strategy

This component can be easily tested by running queries using MySQLWorkbench in order to troubleshoot the tables and the info contained therein.

## 3.2 Class/Entity Model

Class 0 – Main Class

- Contains main method. Uses all other classes in some way to get the job done.

Class 1 – Person

- Person Code a String
  - Unique designation (from the old system)
- First name as String
- Last name as String
- Address as Address
  - (see class 3)
- (optional) Email address or multiple email addresses, stored in an ArrayList of Strings.
  Subclass 1.1 – Broker
  - Includes Broker status as BrokerType
    - Enumerated type which indicates Junior or Expert
  - Broker's SEC identifier as String.

Class 2 – Asset (Abstract Class)

- Label as String
- Code as String

  Subclass 2.1 – Deposit Account
  - APR as double

  Subclass 2.2 – Stock
  - Quarterly Dividend as double
  - Base Rate of Return as double
  - Beta Measure as double
  - Stock Symbol as String
  - Share Price as double

  Subclass 2.3 – Private Investments

    - Quarterly Dividend as double
    - Base Rate of Return as double
    - Omega Measure as double
    - Total value as double

Class 3 – Address

- Street address as String
- City as String
- State as String
- ZIP as String
- Country as String

Class 4 – Portfolio

- Portfolio Code as String
    - Uniquely identifies portfolio.
- Owner as a Person
    - Corresponds to the Person who owns the account.
- Manager as a Person
    - Corresponds to the Person who manages account.
- Beneficiary as a Person (Optional)
    - Corresponds to the designated beneficiary for the portfolio.
- Asset map as a Map mapping Assets to Doubles
    - Maps the assets to the special provided numbers.

Class 5 – Portfolio Comparator

- Allows that comparison of two portfolios for ordering the output

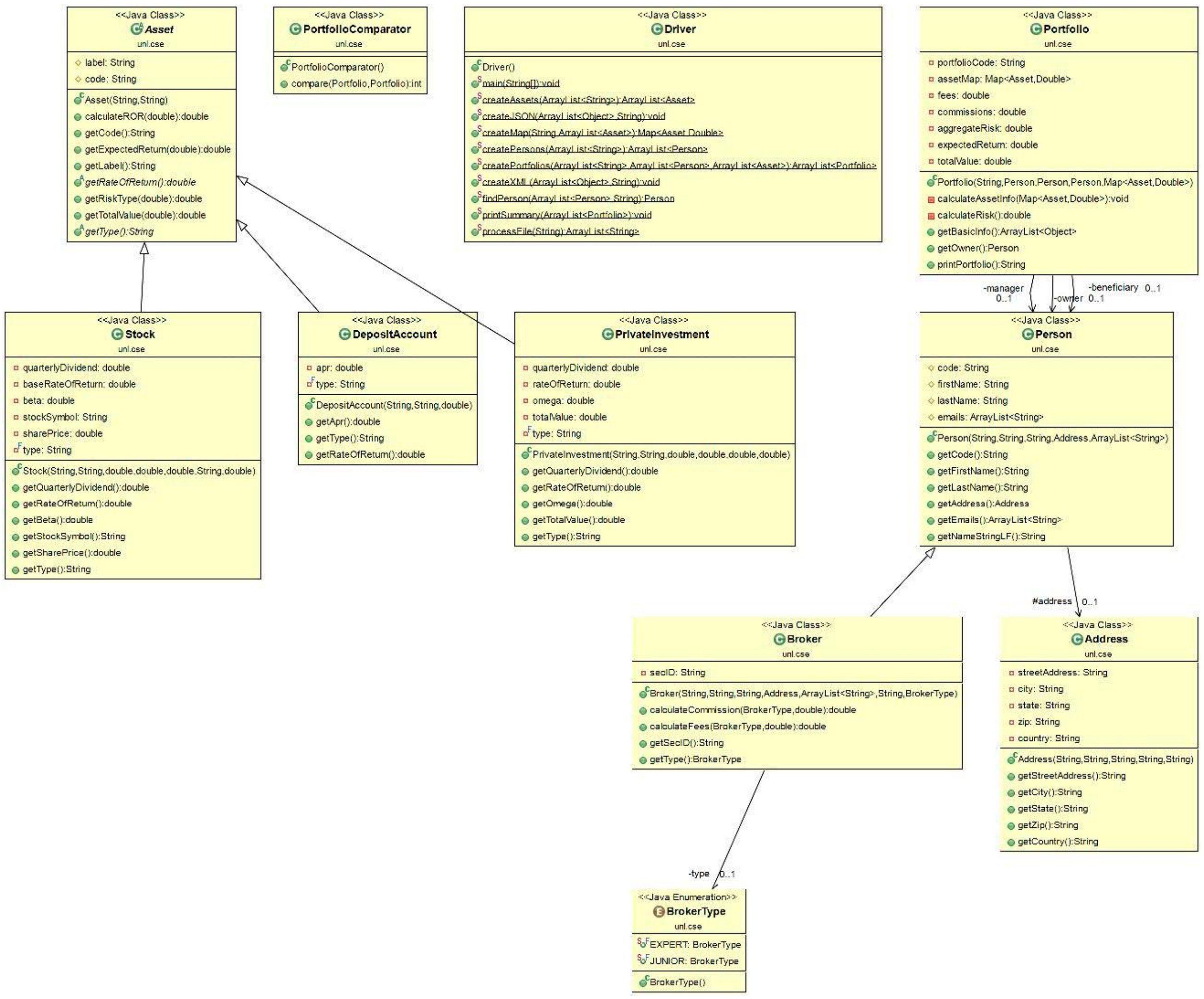

**Figure 1: Above is a Class diagram of the class structure. An arrow with a white head represents inheritance, while an arrow with a black head represents composition.**

### 3.2.1 Component Testing Strategy

For this component, the best testing strategy is to provide the program *.dat files as input and make sure the program outputs a well-formatted *.txt file. The idea is for all the portfolios, persons, and assets to print as they are intended. If it doesn't output correctly or if something went wrong, some re-implementation of the methods is necessary until it does as is expected and as it should.

## 3.3 Database Interface

The program will allow for the use of the previously implemented classes to interact with the SQL database by using JDBC.

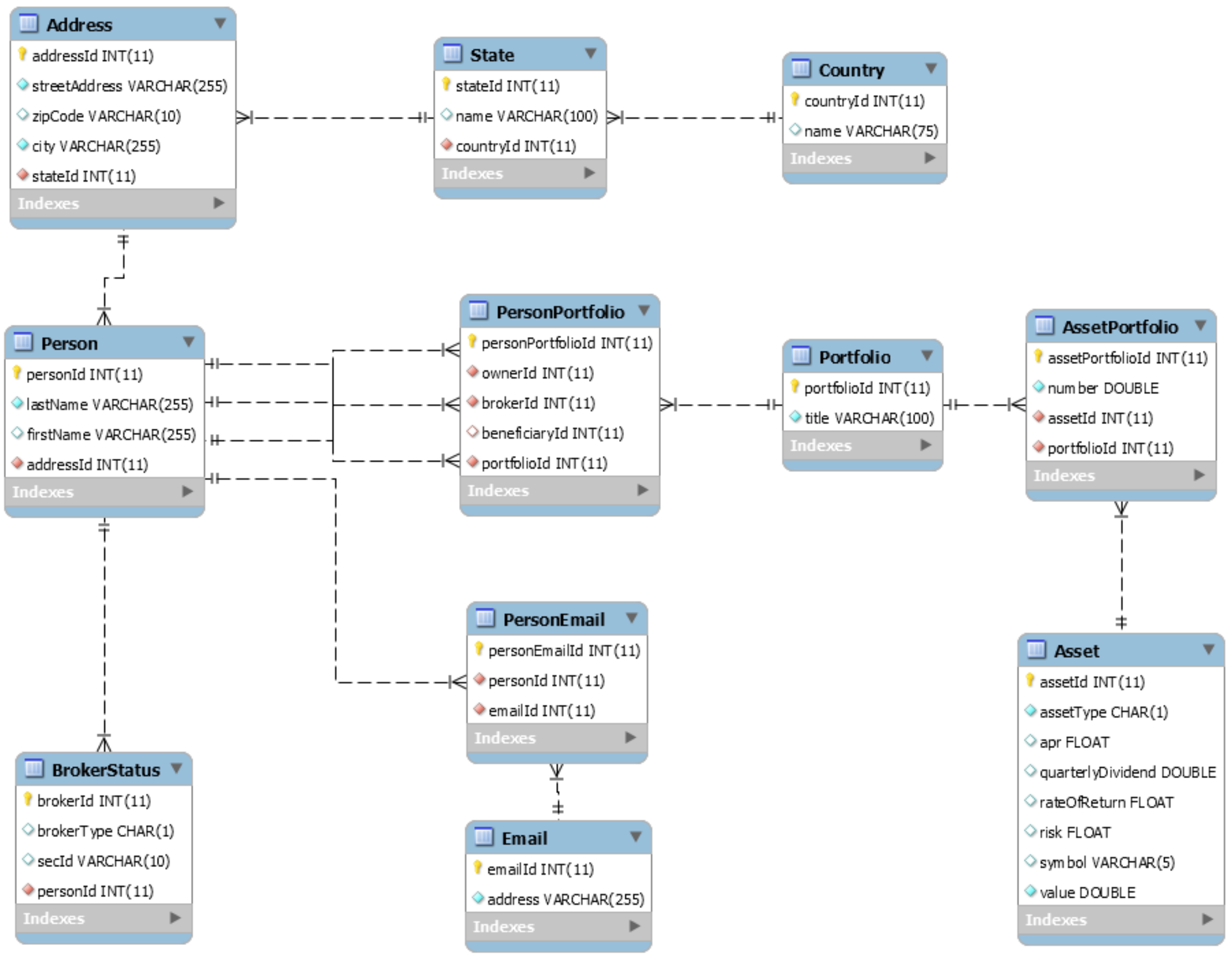

**Figure 2: Above is a diagram of the SQL database table structure. A connection with one prong represents an n-to-1 relationship, while a connection with multiple prongs represents n-to-many relationships.**

### 3.3.1 Component Testing Strategy

TODO: Implement in Phase IV of document. <Only leaving in placeholder so we don't forget later :P>

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

## 3.4 Design & Integration of Data Structures

TODO: Implement in Phase V [5] of document.

[This section will be used to detail phase V where you design an original data structure and integrate it into your application. In earlier phases this section may be omitted or a short note indicating that details will be provided in a subsequent revision of this document?]

#### 3.4.1 Component Testing Strategy

TODO: Implement in Phase V of document.

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

### 3.5 Changes & Refactoring

No major changes or refactors have been incurred thus far.

## 4. Additional Material

### 4.1 External Libraries

We imported a few external libraries to make our XML and JSON output more automated. These libraries are:

- Jackson-annotations-2.8.0.jar
- Jackson-core-2.8.6.jar
- Jackson-databind-2.8.6.jar
- Jackson-dataformat-xml-2.8.6.jar
- Jackson-module-jaxb-annotations-2.8.6.jar
- Stax2-api-3.0.1.jar
- Woodstox-core-asl-4.2.0.jar
- DataConverter.jar

These JAR files work together to allow easy exportation of our Person and Asset objects into human and machine readable formats.

## 5. Bibliography

TODO: Properly implement in future release of design document<Only leaving in placeholder so we don't forget later :P>

[This section will provide a bibliography of any materials, texts, or other resources that were cited or referenced by the project and/or this document. You *must* consistently use a standard citation style such as APA or MLA (good reference: http://www.cws.illinois.edu/workshop/writers/citation/).]

[1] Bourke, C. (2017). *Computer Science II,* http://cse.unl.edu/~cbourke/CSCE156/

[2] Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.