|  |
| --- |
| Department of computer science & Engineering  University of Nebraska—Lincoln |
| The Data Handling Code |
| CSCE 156 – Computer Science II Project |
|  |
| **[Eli Morton & Deverick Simpson]** |
| **November 15, 2013**  **[V0.5]** |

|  |
| --- |
| This document is the Design Document for the Data Handling Code for the Financial group DHC. It is replace the Current aging AS4000 System. |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description of Change(s) | Author(s) | Date |
| .1 | Initial draft of this design document | Eli Morton | 2013/09/17 |
| .2 | Add in Database file | Eli Morton | 2013/10/17 |
| .3 | Add details and updated some information | Deverick Simpson | 2013/10/24 |
| .4 | Updated Database file to current version | Eli Morton | 2013/10/28 |
| .5 | Updated with suggestion from pre-grade | Eli Morton | 2013/11/11 |

Contents

[Revision History 1](#_Toc370907046)

[1.1 Purpose of this Document 3](#_Toc370907047)

[1.2 Scope of the Project 3](#_Toc370907048)

[1.3 Definitions, Acronyms, Abbreviations 3](#_Toc370907049)

[1.3.1 Definitions 3](#_Toc370907050)

[1.3.2 Abbreviations & Acronyms 3](#_Toc370907051)

[2. Overall Design Description 4](#_Toc370907052)

[2.1 Alternative Design Options 4](#_Toc370907053)

[3. Detailed Component Description 4](#_Toc370907054)

[3.1 Database Design 4](#_Toc370907055)

[3.1.1 Component Testing Strategy 5](#_Toc370907056)

[3.2 Class/Entity Model 5](#_Toc370907057)

[3.2.1 Component Testing Strategy 5](#_Toc370907058)

[3.3 Database Interface 5](#_Toc370907059)

[3.3.1 Component Testing Strategy 6](#_Toc370907060)

[3.4 Design & Integration of Data Structures 6](#_Toc370907061)

[3.4.1 Component Testing Strategy 6](#_Toc370907062)

[3.5 Changes & Refactoring 6](#_Toc370907063)

[4. Additional Material 6](#_Toc370907064)

[5. Bibliography 6](#_Toc370907065)

1. **introduction**

This project is to replace the handling of data by the aging AS4000 System with a newer more Flexible approach based on java Object Oriented. It is to be design with the ability to have the data contain in it to XML or JSON. It will also be able to output summary and Detailed Reports of the portfolios contain in the program.

## Purpose of this Document

The document covers the design elements that were necessary to develop a functional financial management system for the organization. The software, written as a java application, is being implemented to replace DCH Financial Group’s current management system. Some of the goals of this project are to have the Data Handling Code be flexible and provided useful function inside of the program.

## Scope of the Project

The Scope of the Project is to replace the aging AS4000 System with a newer and faster system. To also provide better maintainability and upgradability for the Future than what the AS4000 can provide. This System is being developed for the DHC financial group.

## Definitions, Acronyms, Abbreviations

### Definitions

Deposit Account – Accounts such as Savings Accounts, Certificate of Deposit, and Money Market

Accounts, which have zero risk.

Stock – Investment account that have some risk for the customer.

Private Investment – Account with investments in a private enterprise for some return.

Portfolio – A group of assets for a particular customer.

Broker – Assigned agent to manage the customer’s portfolio. A broker can either be an Expert Broker or Junior Broker

### Abbreviations & Acronyms

XML – EXtensible Markup Language

JSON – JavaScript Object Notation

CD – Certificates of Deposit

MMA – Money Markey Account

SEC – Securities and Exchange Commission

ACID—atomicity, consistency, isolation and durability.

# Overall Design Description

The overall design principles that are in the program are OOP principles of polymorphism, abstraction, inheritance, and encapsulation. This financial management system lends itself to objects to store the different data. The three main objects being Person, Asset, and Portfolio. These are then extend and Sub-classed by Deposit, Investment, PrivateInvestment, Stock, and the Broker class under them to add or modify their implications as need. There is also the Address class to store the address’s

These are then backed up by a persistent Database following a similar style using MySQL. This is to facilitate future development in to multiple clients. The relationship is elaborated on a more detail in the 3.1 and in Figure 1 below.

## Alternative Design Options

Other design options for this project could have been

* Instead of design an inheritance system make the broker class completely separate from the person class. This would have worked but it adds a lot of redundancy in that doesn’t really provide a benefit for the added space requirements. Same could be said for the Asset class and its subclasses
* Another design option could have been to store the data in something other than a database to make it persistent, like a spreadsheet or similar file. While this would have reduced the complexity greatly down it would have cause a severer problem with allowing multiple clients to access the same data and modify it. Where the databases adds an ability to serve multiple clients and keep the ACID principles intact. Which great improve reliability.

# Detailed Component Description

## Database Design

The MySQL database schema is constructed as required within of the Order-Invoice System for DCH Financial Group. The database schema is defined with tables for each of the dependent data objects within our new financial management system, i.e. persons, assets, and portfolios. Also, tables such as ­­­­­­­­ portfolioAsset are used to establish many-to-many relationships within the schema between objects. Finally, additional tables, such as the email and mailAddress table are used to manage smaller data models involved within the Financial Management System.

*The ER diagram given in* ***Figure 1*** *below outlines the MySQL database schema.*

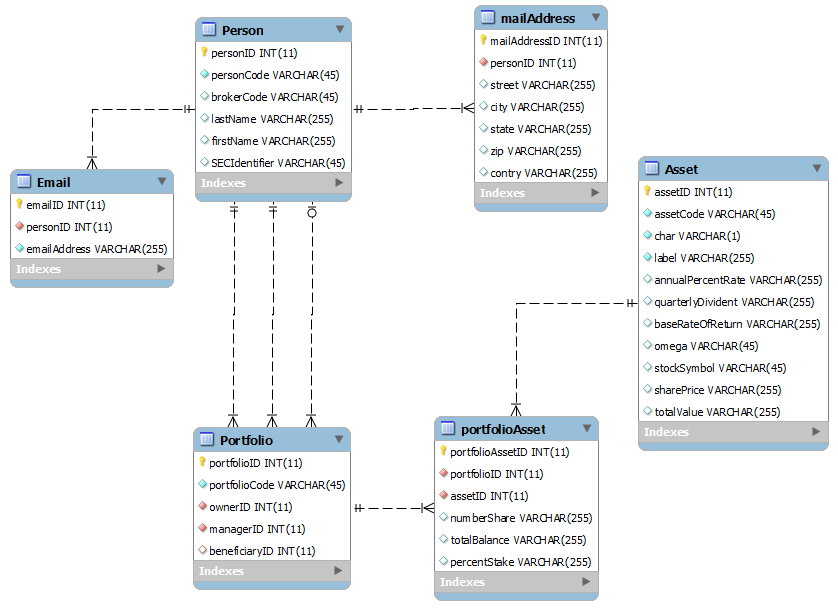


Figure Showing the Relation of Tables in the Database and what they are storing

### Component Testing Strategy

The database schema undergoes a series of test to ensure its validity as a proper model of the Financial Database Management System. More specifically, the model is tested with data from Phase I/II as well various queries upon the system. These queries attempt to test the durability of the system by attempting to insert, update, and delete data.

## Class/Entity Model

## The main class for people information is the Person class with the Broker extending it to add in the SEC and Rank for the broker. These both use the Address class. The top one for asset data is the abstract Asset Class which isn’t used directly but is extend by the Deposit class, the abstract Investments Class. Which is then extending by the PrivateInvestments class and the Stock Class.

### Component Testing Strategy

The class have undergone a series of test to ensure that it hold the information correctly and can model the information correctly. This included trying to model null data and incorrect data to test and insure that It handles it as designed.

## Database Interface

To make the data more persistent we designed a database to hold and retrieve from. This is implemented the API to handle this. This relies on another implemented separated class to handle the connecting and closing of the database connection. The driver we use the for the database was the MySQL with JDBC driver. This API takes care of

### Component Testing Strategy

We tested it

## Design & Integration of Data Structures

### Component Testing Strategy

## Changes & Refactoring

There have been no major changes or refactoring have been made as of V.2

# Additional Material

# Bibliography

[1] *Citation Styles*. (n.d.). Retrieved December 19, 2012, from [http://www.cws.illinois.edu/workshop/writers/citation/](http://www.cws.illinois.edu/workshop/writers/citation/))

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