DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

Financial Management Program

CSCE 156 – Computer Science II Project

Logan Wisniewski
Zack Boone
11/15/2013
Version 5.0

The following document describes the design and aspects of a Financial Management System developed for DHC Financial Group.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Logan Wisniewski Zack Boone	2013/09/19
2.0	Portfolio Report added (summary and detailed reports), Bibliography updated	Logan Wisniewski Zack Boone	2013/09/27
3.0	Database design added, database created, UML diagram added, ER diagram added, Bibliography updated	Logan Wisniewski Zack Boone	2013/10/11
4.0	Database integration added, Database design updated, Bibliography updated	Logan Wisniewski Zack Boone	2013/10/25
5.0	ADT design added, Bibliography updated	Logan Wisniewski Zack Boone	2013/11/08

Contents

1 Introduction

This software design description outlines the designing, testing, and implementation of an API that models a Financial Management System which manages portfolios, assets, and accounts for the DCH Financial Group. The API uses an OOP and Database paradigm to carry out necessary tasks of the project's scope. The new Financial Management System is being implemented to replace the existing system and its database used to store DCH's financial records.

1.1 Purpose of this Document

This document serves as a blue print to convey the structural ideas of the financial management API and its design. This design document provides necessary details in relation to Java about the interface; including its interactions with the user and database. This document also indicates its advantages, as well as disadvantages, with relation to design and functionality.

1.2 Scope of the Project

The Financial Management System serves as an investment manager for the DCH Financial Group. This system manages financial investments and returns based on types of assets, amount of risk, broker expenses, etc.

The details of each investment are as follows:

- Type of asset that the user is investing in
    - Private Investment
    - Deposit Account
    - Stocks
- The amount of risk that each asset contains
- The commission fees of different types of brokers
- The expected annual return of each investment

The system uses this as its primary logic when executing requested investments by accessing classes of data using the appropriate methods associated with groups of classes. The API utilizes its database to store previous instances of investments.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Comparator: A comparison function, which imposes a total ordering on some collection of objects.

Customer: the client that the company serves. The company serves three types of clients: personal, corporate, and academic.

Deposit Account: FDIC (Federal Deposit Insurance Corporation) insured accounts like Savings Accounts, CDs (Certificates of Deposit), and MMAs (Money Market Accounts).

Omega: Value between [0, 1] defining how risky an investment is; 0 being no risk and 1 being very risky.

OOP: A paradigm that uses classes that creates instances of objects with unique fields and methods.

Private Investments: Investment accounts in private enterprises (rental properties, small to medium businesses, franchises, etc.).

Stocks: An investment account that consists of a number of shares of a particular publicly traded stock.

1.3.2 Abbreviations & Acronyms

ADT: Abstract Data Type

API: Application Programming Interface

ER Diagram: Entity-Relationship Model

CRUD: Create, retrieve, update, destroy

JDBC: Java Database Connectivity

OOP: Object-Oriented Program

SQL: Structured Query Language

UML: Unified Modeling Language

XML: Extensible Markup Language

2 Overall Design Description

The Financial Management System contains classes and fields to carryout necessary JAVA methods and MySQL queries. This is done by utilizing OOP principles of abstraction, encapsulation, inheritance, and polymorphism and SQL principles of CRUD. The program creates objects such as Assets, Persons, and Portfolios. Each object contains subclasses to add functionality to different types of assets and people.

I.E. Variations of people:

- Broker
    - Expert

        - $10 per asset
        - 5% commission of portfolio value
    - Junior
        - $50 per asset
        - 2% commission of portfolio value
- Non-Broker
        - No Commissions
        - No Fees

The program also utilizes an ArrayList ADT, which carries an arbitrary number of asset, person, and portfolio instances and maintains sorting by asset label, person name, and portfolio value.

In the Financial Management System data storage is managed by a MySQL database. Each piece of data, such as Portfolio, Persons, etc., has a dedicated data table in the database. Using the JDBC API the application connects, reads and updates the MySQL database.

3 Detailed Component Description

Section 3, in general, will highlight the components of Section 2 in a more detailed manor and asses the testing methods of these design components.

3.1 Database Design

The ER Diagram in figure 1 represents the relationship between each class by labeling primary and foreign keys. The database schema was created to easily insert and retrieve information for the program.

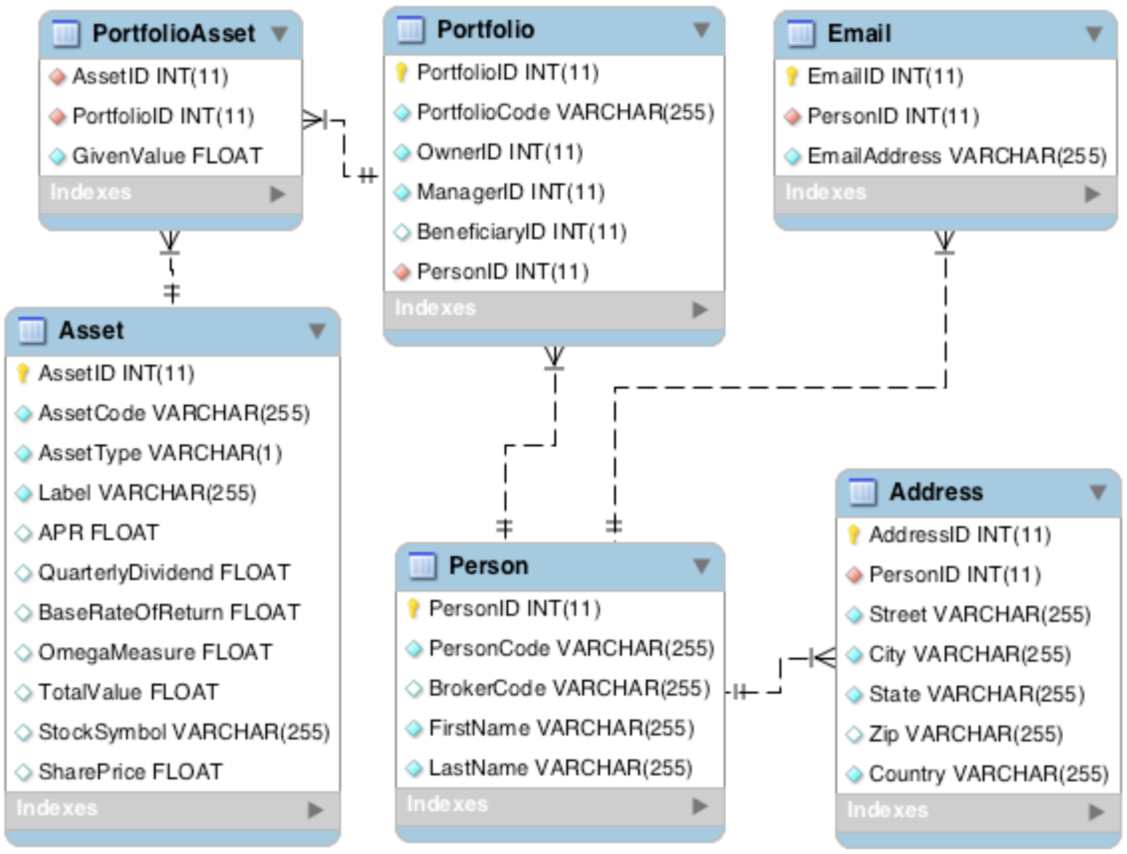

Figure 1: An ER Diagram explains the relation between data tables

3.1.1 Component Testing Strategy

In order to test the Financial Management database, the data from Phase I and II is imported into the designated tables. A series of MySQL queries are created to test that there is sufficient data in each table and that each table is accessible by the next.

3.2 Class/Entity Model

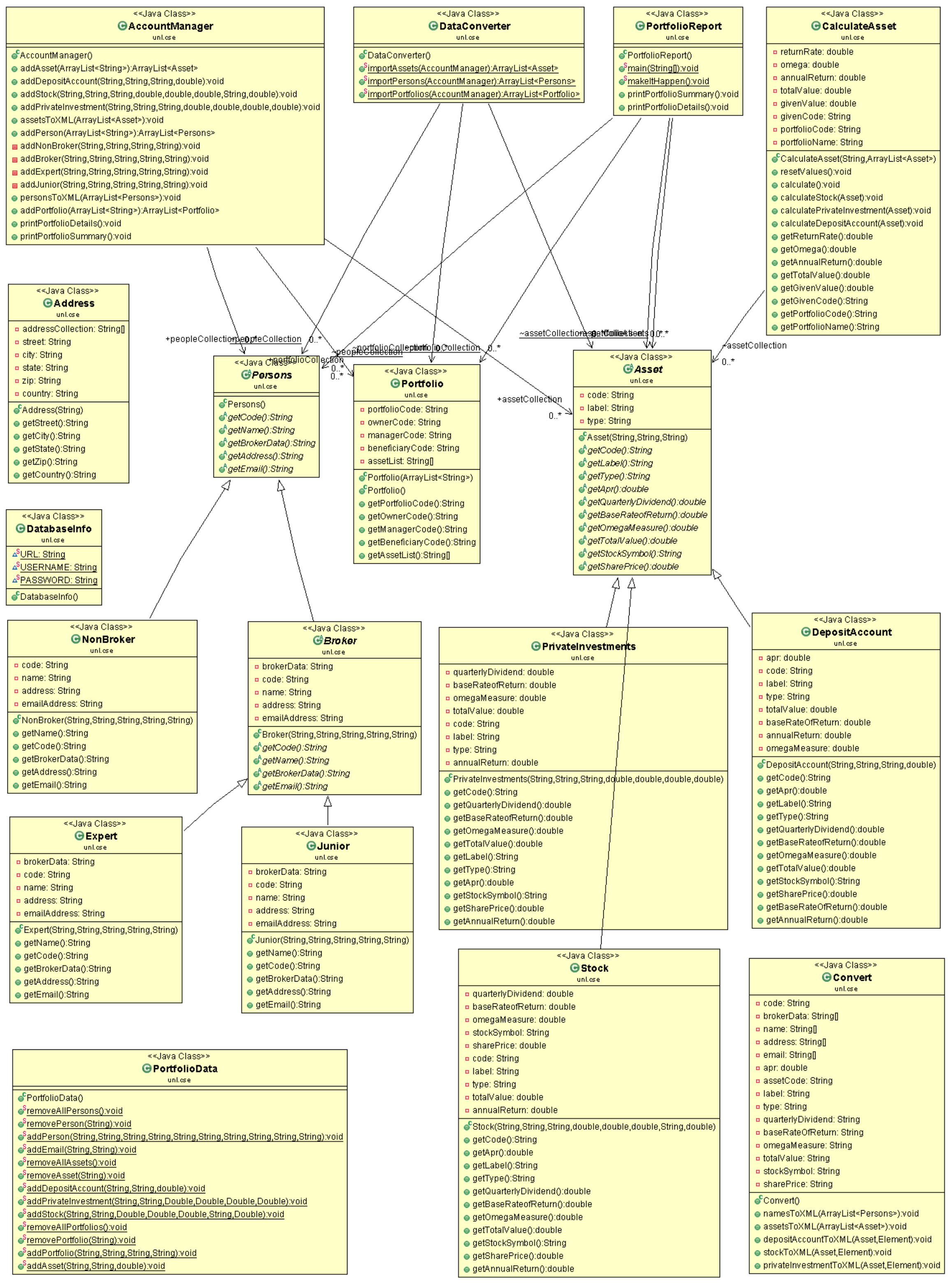

Figure 2: Financial Management System UML Diagram explains the relation between each class

3.2.1 Component Testing Strategy
Classes and entities are test by implementing test cases of non-trivial data. The data is imported to data tables and debugged using the built-in Eclipse debugger. This process insures the classes work seamlessly with the data provided. Each test case contains corner cases that tests empty portfolios, assets without values, etc.

3.3 Database Interface
The implementation of the JDBC interface with the Java program is done by using the Connections library. This library is used by assigning the database URL, username, and password. This information is stored in data containers inside their own class. The importation of information is done through JDBC by converting the import class from dat file to MySQL.

3.3.1 Component Testing Strategy
In order to test the Financial Management database, the data from the web grader test case is imported into the designated tables. A series of MySQL queries are created to test that there is sufficient data in each table and that each table is accessible by the next.

3.4 Design & Integration of Data Structures
This phase of the Financial Management System is the design and integration of the ArrayList ADT. The ArrayList of this program is designed to be a data container for an arbitrary number of objects. The ArrayList ADT consistently manages and maintains the ordering of objects. More specifically, the objects in the ArrayList can be ordered by person name, asset label, and portfolio value.

3.5 Changes & Refactoring
No major changes or refactoring have been made to this project.

4 Bibliography

Arapidis, C. (2012, September 25). How to generate UML diagrams from Java code in Eclipse. Retrieved November 14, 2013, from http://fuzz- box.blogspot.com/2012/09/how-to-generate-uml-diagrams-from- java.html

Eckel, B. (2006). Thinking in Java (4th Edition). Prentice Hall.