DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

TBF Financial System

Computer Science II Project

Natalie Ruckman & Joel Murch-Shafer
4/10/2020
Version 1.3

This document describes the design process of a software to replace the current AS400 financial management software. This software is created with Java with a SQL backed relational database.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document.	Natalie Ruckman and Joel Murch-Shafer	2/7/2020
1.1	Added description of class associations with the addition of a portfolio class.	Natalie Ruckman and Joel Murch-Shafer	2/20/2020
1.2	Added SQL relational database design.	Natalie Ruckman and Joel Murch-Shafer	3/13/2020
1.3	Added JDBC and ADT information	Natalie Ruckman and Joel Murch-Shafer	4/10/2020

Contents

1. Introduction

The AS400 green screen software has been a problem for TBF financial for the past few years. This replacement software application is being developed in Java backed by a SQL relational database to replace the AS400 system. This software stores information that models real world accounts as well as any interactions between such accounts to manage TBF's data.

1.1 Purpose of this Document

This document is aimed towards all persons that may have substantial interest in the business or technical design of this software. This document explains the development and technical functionality of this software and its components as well as its practical applications.

1.2 Scope of the Project

This project models the information held by real world assets, clients, and brokers as well as any interactions between them. This information can then be accessed and processed as needed. This software aims to manage the financial data responsibilities of TBF financial (such as creating portfolio reports and calculating risks and returns). The software is written in Java, using object-oriented programming concepts to support TBF's needs.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Class – Programming concept representing a real-world object
- XStream – A library used to serialize objects to XML.
- GSON – A library used to serialize objects to JSON.
- Log4J – A library used to create a file and log errors

1.3.2 Abbreviations & Acronyms

- ADT – Abstract Data Type
- AS400 – The previous financial system used by TBF.
- CRUD – Create, Retrieve, Update, Destroy. Basic database functionalities.
- JDBC – Java Database Connectivity
- JSON – JavaScript Object Notation
- OOP – Object oriented programming.
- SQL – Structured Query Language
- TBF – Too Big To Fail Financial Company
- XML – Extensible Markup Language

2. Overall Design Description

This software reads data from a database using JDBC and creates Java Objects to store and represent the data. Objects can be of different classes to represent the information they represent. Classes include Person, Portfolio, and Asset (subclasses Private Investment, Deposit Account, and Stock). Each class has methods that enable interaction with the object data. This information from the AS400 system will then

be stored in the SQL relational database. This software then creates reports of a list of all clients, their assets and other pertinent information. Java, as an OOP language, allows us to create objects to represent different types of information, as well as to implement inheritance through parent and child classes. Additional functionality will be expanded throughout the design process.

The MySQL database is connected to the JDBC API through the class DataLoader. The SQLFactory class is a Data Access Object used to make repetitive connections more efficient. Log4j has been implemented to log errors to a log file.

2.1 Alternative Design Options

The subclasses of Asset could have each been their own individual class, but using inheritance is better when the classes share items in common. This also allows the use of polymorphism when using a general 'asset' class is more useful.

The ADT could alternatively have been created using an array-based list. This could make it more efficient to access a piece of data at a specific location, but it would also be more taxing to have to move all of the objects backward or forward when an item is added or deleted.

3. Detailed Component Description

This software uses a dynamic class structure to model instances of assets (private investments, deposit accounts, and stocks) as well as people (owners, managers, and beneficiaries). There is an asset parent class and a person parent class, both of which pass their qualities down to their child classes (the above-mentioned subclasses.

3.1 Database Design

This database (shown below) is created in MySQL. The join table PortfolioAssets allows a many to many relationship between portfolios and assets.

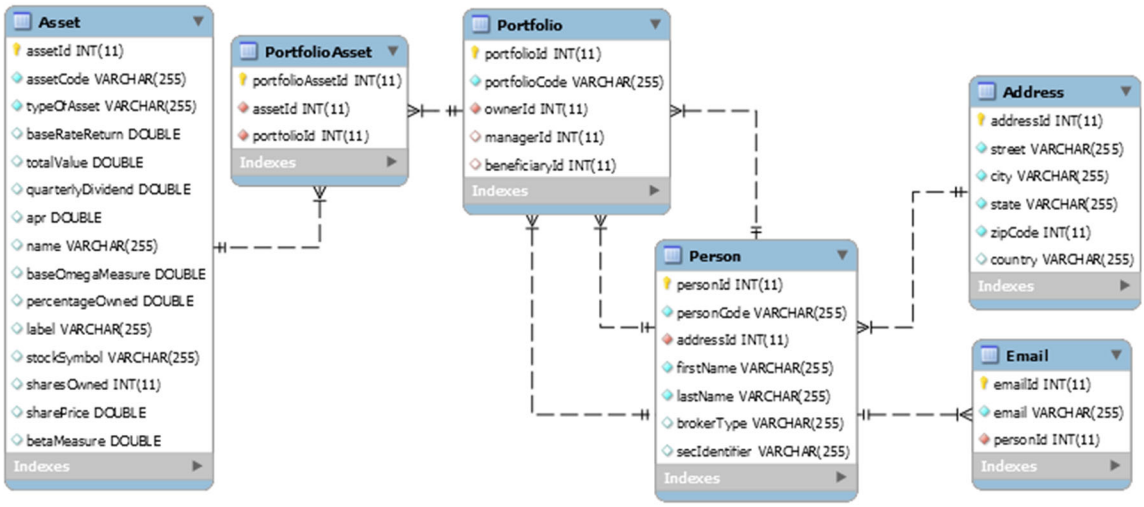

Figure 1: Shows MySQL tables and links between them.

3.1.1 Component Testing Strategy

The database function will be tested by generating edge cases and unit testing each table and interaction. Both the SQL database and the Java software is tested using select statements so that the provided database information can be manually verified to be correct.

3.2 Class/Entity Model

The new piece of software consists of 5 classes to model all real-world objects. These objects are explained below and described in more detail in *Figure 1*.

- *Person* – Holds information relating to a person and has a link to an address object.
- *Address* – Holds information to specify the address of all people
- *Assets* – Abstract class that has three subclasses.
    - *Deposit Accounts* – Holds information relating to deposit accounts stored
    - *Stocks* – Holds information needed to describe stocks.
    - *Private Investments* – Holds information needed to describe private investments.
- *Portfolio* – Stores information needed to map associations between all people and assets.

Functionality classes:

- *XMLConversion*
- *JSONConversion*
- *DataConverter*
- *DataLoader*
- *DatabaseInfo*
- *GenerateReport*
- *SQLFactory*

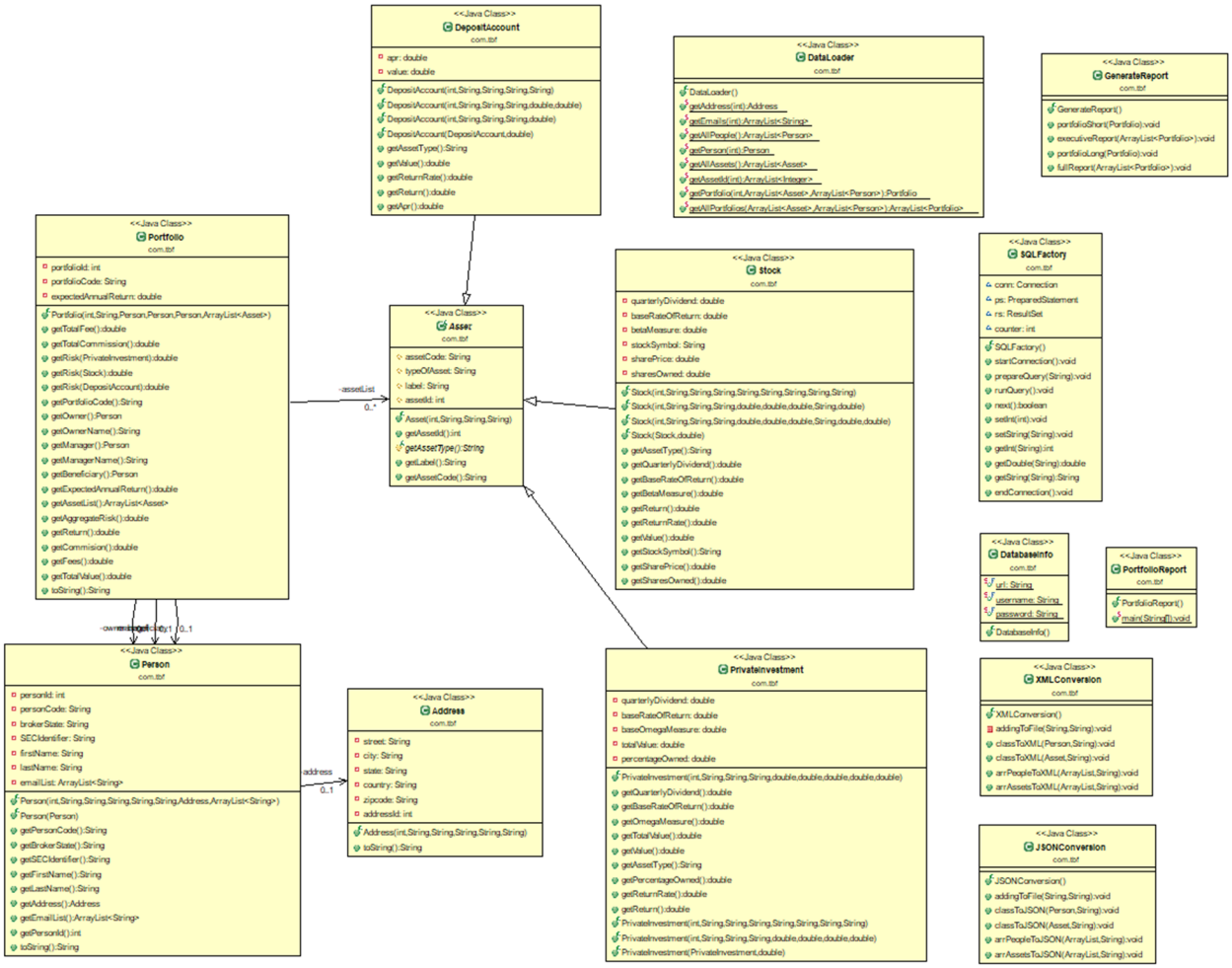

Figure 2: Shows class structures and relationships between classes.

3.2.1 Component Testing Strategy

The software will be rigorously tested with test cases designed primarily through Mockaroo, a random instance generator. These cases will then be checked by hand to ensure the correct report output.

3.3 Database Interface

JDBC, Java's API, is used to connect to the MySQL database through prepared select statements. Within the MySQL database, primary keys are auto-incremented, and all essential data is not allowed to be null. Invalid duplicate data is dealt with by the database through using uniqueness constraints.

3.3.1 Component Testing Strategy

This database will be tested using the print report functions to ensure connectivity and functionality. The MySQL database will also be tested using select statements to validate the correctness of the database.

3.4 Design & Integration of Data Structures

The ADT is implemented using an array-based list. The ADT class is constructed from database portfolio data. As the data is moved into the ADT, a comparator will sort the data based on the ordering that is

desired (by last name and first name, by total value, or by manager) and then add it to its correct place in the list. When a portfolio is added to the list (or deleted) the reference points will be moved when it is added so that it is inserted (or removed) properly into the linked list. The size of the list will be checked upon each insertion or removal, and the capacity will be changed if the size is approaching the capacity, or if the size is significantly smaller than the capacity. Order will be continuously maintained: when a node is added, it is immediately inserted into its place, and if one is removed, the references are rearranged so that order is maintained. This list could hypothetically hold class types other than portfolio, but it is intended for use on portfolios. Depending on what the alternate type is, the ordering comparisons would have to be changed.

3.4.1 Component Testing Strategy

This ADT will be tested manually using fake data to test the lists insertion and deletion capabilities and ensure proper order is maintained.

3.5 Changes & Refactoring

In update 1.1, the portfolio class was added, as well as class associations to allow the portfolio class to interact with other classes.

In update 1.2, data gathering was changed from accessing and processing a flat file, to interacting with a database.

In update 1.3, some changes were made, breaking up the responsibilities of the PortfolioReport class (such as parsing and accessing data). The Asset class was also changed to contain all of the data about any of the subclasses, with the rest of the unnecessary data that doesn't correspond to the Asset type being null. This makes it much easier to connect to the JDBC and retrieve data.

4. Bibliography

"About XStream." XStream - About XStream, x-stream.github.io/index.html.

Google. "Google/Gson." GitHub, 5 Nov. 2019, github.com/google/gson.

Mockaroo, LLC. "Random Data Generator and API Mocking Tool: JSON / CSV / SQL / Excel." Mockaroo, www.mockaroo.com.