DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

FarMart

[FMT]

[Kashish Syed]

5/9/2023
[Document Version 5.0]

This document details all the phases of making the Application for business FarMarT owned by Pablo Myers. The system will consist of different components and all the details for it will be provided in individual sections.

Revision History

[This table documents the various major changes to this document]

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Kashish Syed	2023/02/13
2.0	Added to definitions, abbreviations. More detailed UML diagram was added.	Kashish Syed	2023/03/03
3.0	Added information about the database and the EER diagram	Kashish Syed	2023/03/21
4.0	Added to Database Interface and Design and Integration of Data Structures	Kashish Syed	2023/04/04
5.0	Updated the Design and Integration of Data Structures section to add more details.	Kashish Syed	2023/04/21
6.0	Reviewed and updated the overall document.	Kashish Syed	2023/05/09

Contents

Introduction

This project is for updating all the business operations for the client, FarMarT, whose owner is Pablo Myers. This project focuses on supporting and updating the technology side of his business, which involves inventory, marketing, delivery, and sales. The new system will be responsible for keeping track of all purchases and applying taxes depending on the customers and the items they purchase.

1.1 Purpose of this Document

The purpose of this document is to provide an outline of the system for independently developing technology for the business FarMarT and the implementation of every feature inside the system. The system is written in Java and use of inheritance is done to create a more structured and efficient approach to manage the classes so that the performance and productivity is improved. Databases are made in SQL and connected to java code to make sure the data is easily retrieved and used.

1.2 Scope of the Project

The system helps FarMarT, the client, in selling their products which have a unique alphanumeric code and human-readable name to each of them. The purchasing of different items has different things associated with it like taxes, time-period, weight, and many more. When an item is purchased, various reports are produced including summary of sales and total sales, sales in each store, details of individual invoices, etc. The task is to make the reports accurate and manageable, so that it is easy to handle the sales and purchases done and this is done through SQL database through which it is easier to retrieve data.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Encapsulation – One of the key concepts in OOP and it is a technique that allows one to hide the internal implementation details of an object from the outside world.

Interface - A set of abstract methods and constants known as an interface allows classes to describe a set of behaviors without having to explain how those behaviors will be implemented.

Polymorphism - One of the key concepts in OOP which enables one to express several object types using a single interface. Method overloading and method overriding are the two strategies used in Java to provide polymorphism.

1.3.2 Abbreviations & Acronyms

Abbreviations and acronyms used in the project –

OOP – Object Oriented Programing

JSON – JavaScript Object Notation

XML – Extensible Markup Language

CSV – Comma Separated Values

SQL – Structured Query Language

FMT – FarMarT

JDBC – Java Database Connectivity

API – Application Programing Interface

ADT – Abstract Data Type

2. Overall Design Description

This application is designed in Java through OOP where number of classes/objects are defined to make the system organized. It uses encapsulation, abstraction, interface, and polymorphism to provide better functionality for sales reports. Many classes like Item, Invoice, Person, Store, Address, Equipment, Product, Service, Lease, and Purchase are made to hold the data to process the application. These classes use inheritance to create a hierarchy between classes which relates to Items to provide better functionality of code. Various classes are made to load the data from a csv file and convert it into JSON and XML format which is done through a data converter. To print out the reports for sales and purchases a class InvoiceReport is made, which organizes the input taken from the database.

Data is stored in database created in MySQL. The database is connected using a JDBC API and a connection factory so that the data can be retrieved and updated easily. JDBC allows better performance of CRUD functionality of database. The tables created are Item, Invoice, InvoiceItem, Person, Store, Address (Normalization is done for this table by splitting it into Country and State table), and Email.

2.1 Alternative Design Options

Did not make the equipment class abstract as it was hindering with the process of parsing and tokenizing the csv files. Without the abstraction it is easier to load the Invoice files and can call the methods of Equipment subclass by making new objects of it.

An InvoiceItems class was made to store the items in invoice and to connect Item and Invoice class to retrieve data. This class was removed because it was not providing any functionality and was just doing duplicate work. Instead, a list of Item was added in the Invoice class to connect the Invoice to Item class.

The database table for address was split into different components of itself and was connected to each other. This design was over complicated based on child-parent relationship and would require deleting each table in a sequence if the user wants to delete an address. To tackle this problem, this long normalization was changed, and the details are provided in section 3.1.

3. Detailed Component Description

This section gives details about the different components which help in making reports for the business and sales in various ways.

3.1 Database Design

The database is created using MySQL and represents the structure of FMT. A total of 9 database tables are made in SQL named Item, Invoice, InvoiceItem, Store, Person, Email, Address and Address table is split up into 2 more parts names Country, and State where they all have individual ID's which are primary key. Item and Invoice are connected to InvoiceItem table through one-to-many relationship as many Items can have one invoice. Similarly, Person and Invoice, Person and Email, Person and Store, and Address and Person are connected to one-to-many relationship. Address and Store have a one-to-one relationship because one store can only have one address to it. The tables made for Address are connected through one-to-many relationship and are split to do the normalization of the same so that data retrieval can be easier. More details can be seen in figure 1.

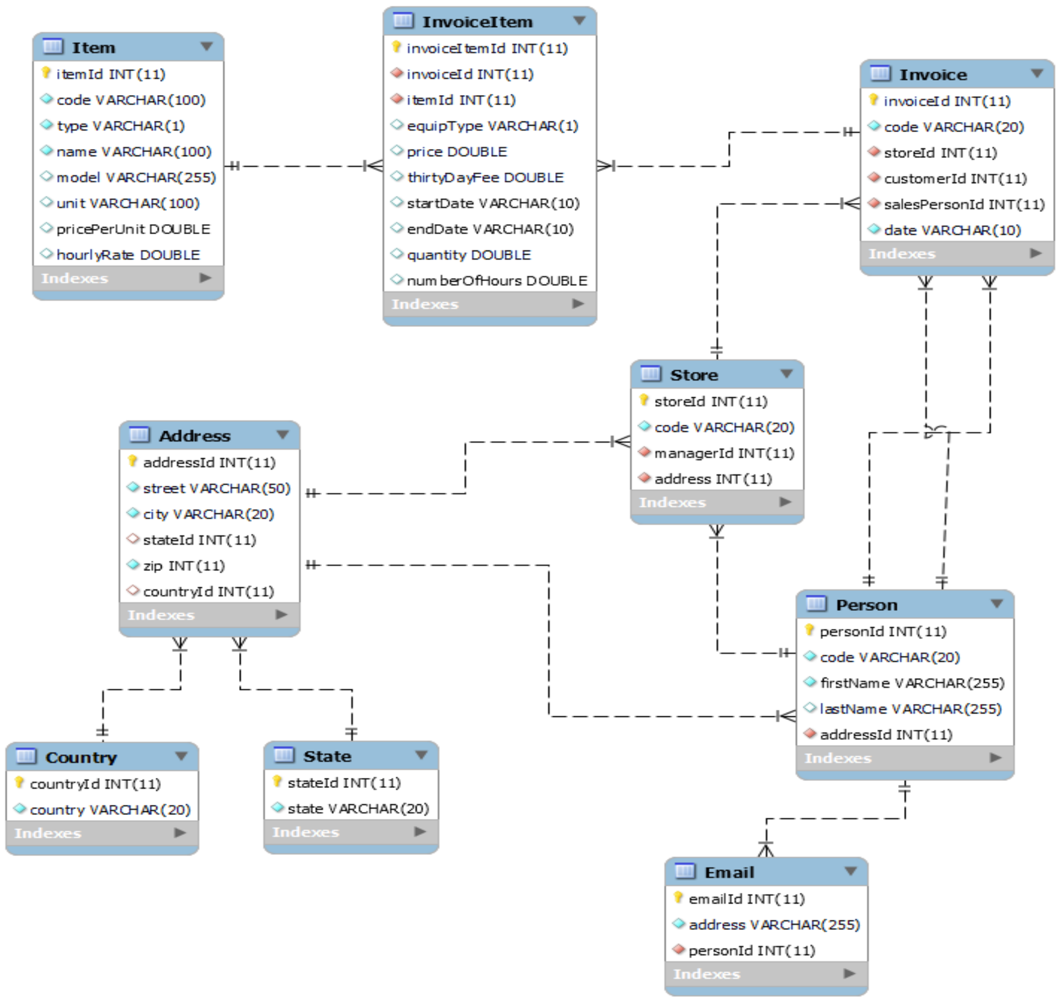

Figure 1: ER Diagram

3.1.1 Component Testing Strategy

To ensure that the program is functioning properly several rows of data have been inserted into each table, and queries have been written. This makes sure that update or retrieve queries are running efficiently.

3.2 Class/Entity Model

Several classes are made to support the application required for FMT. There are three primary classes where everything is dependent on namely Item, Person, and Store. Item is an abstract class and Equipment, Product, and Service classes extend to it. The classes Purchase and Lease are for checking if equipment is leased or purchased. Both classes extend Equipment class. Other than that, Invoice and InvoiceItem classes are to compute the sales and print out a report for same. Moreover, there are various methods to help with the application which can be seen in Figure 2.

Additionally, there are other classes:

1) InvoiceReport is the main class of the program and helps in printing the reports for FMT.
2) DatabaseLoader class is used to connect to the database and load the information from all the tables. ConnectionFactory class has all the connecting and logging parameters that are used in DatabaseLoader.
3) InvoiceData has all methods that connect to the database and are used to make changes in that through java. For example, one of the methods allows functions like adding a person's information into the database.
4) CsvDataLoader was used to read in from flat files instead of the database. This class can be helpful when csv files are provided for small data. ConvertToJson and ConvertToXml classes were helping in checking loaded data from csv files.
5) MyLinkedList is a class that implements the functionality of a linked list. It is an iterable class that has various methods. One of the methods is used to add the elements in the sorted position with the help of the comparator.

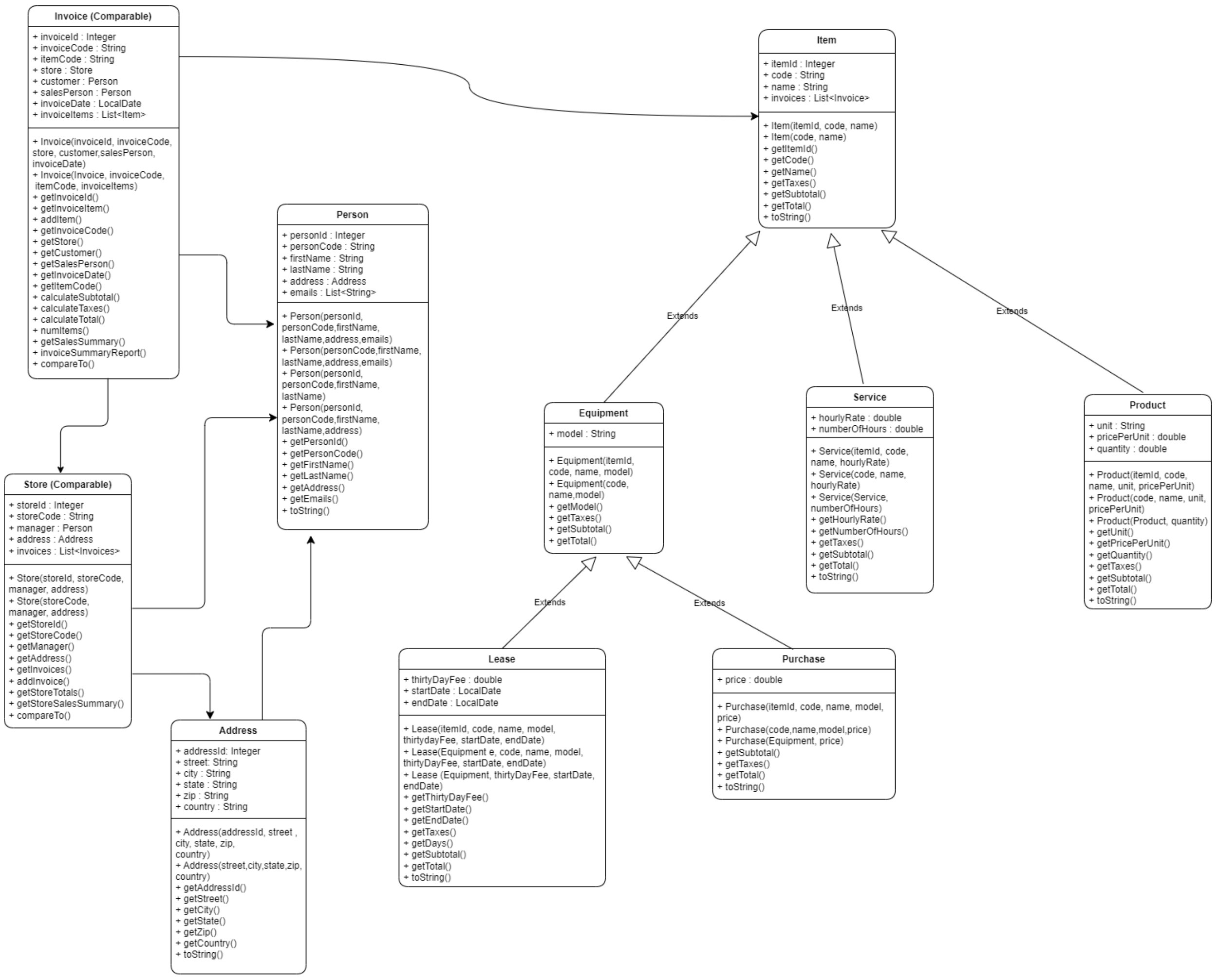

Figure 2: UML Diagram

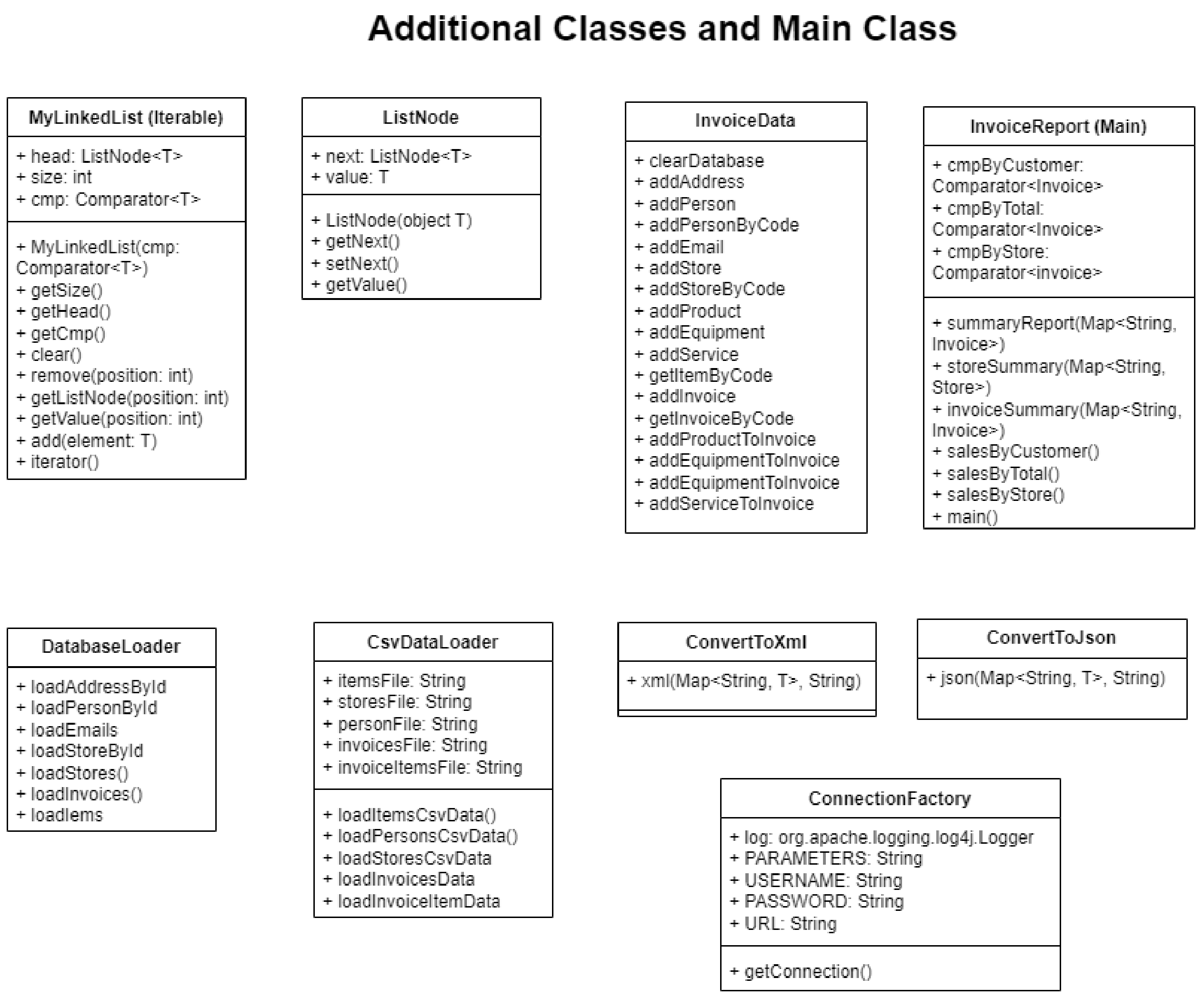

Figure 3: UML diagram for additional classes

3.2.1 Component Testing Strategy

Made hard coded test cases for Items, Persons, Stores, Invoice, and InvoiceItems in csv format to test the code for the application. The csv files are loaded through a csv loader and then converted into JSON and XML format to show how the code works. Reports were made for stores, people, and invoices for again checking the performance of the application.

3.3 Database Interface

The applications now use a class named DatabaseLoader to read in from database rather than flat files. To connect the database with the application, JDBC, a Java API, is used, which enables us to read, write, update, and retrieve that data directly from the database and no further queries are required to be written in SQL.

The database loader uses different methods for loading the data for individual objects as well as all the information on one table. The database loader first makes a connection with the database and then prepares the query provided. After the query is prepared, null or invalid values are handled accordingly.

Various constructors are also added in individual classes to make the loader function properly. Furthermore, a logger system Log4j is utilized to log any errors that may occur during the use.

3.3.1 Component Testing Strategy

A report is generated using the database connection. This report is like when the loader was loading from the csv files; however, now the data is loaded from the database. To check the correctness and performance of the program, the data in the database is updated a few times, few queries are inserted as well as deleted through the program. By doing this, it ensures that the results are desired ones, and no error or bug is seen in the application.

3.4 Design & Integration of Data Structures

The last phase of the application focused on extending the functionality of the program by implementing sorted list ADT to store arbitrary objects, mainly the invoice class objects. The ADT is responsible for ordering the objects by three criteria: 1) By the first/last name of the customer in alphabetical order 2) By total value of the invoice from highest to lowest and 3) By grouping invoice by stores and then salesperson's last/first name.

3.4.1 Component Testing Strategy

To check the integration of the ADT, the output is checked thoroughly by doing calculations by hand. The data in the database is updated randomly to make sure the program works with all three comparators and possible data.

3.5 Changes & Refactoring

- There was a class for a person's last name and first name. This class was providing similar functionality when the last/first name were added directly into Person class. Hence, deleted the Name class.
- Before the zip was an Integer type and was hindering with processing of String type zip codes such as 3456-6789 can be added as well ("-" is a character which falls into String type). Therefore, the zip in Address class was changed from integer type to String type to make sure it works with all kinds of zip codes.
- InvoiceItem class was made to add objects for all the items present in an invoice but later was removed, and the Item class was directly connected to Invoice class which proved better for the functionality of the program.