FarMarT

Invoice Management System

Peter Morales
pmorales4@huskers.unl.edu
University of Nebraska—Lincoln

2023-05-08
Version 6.0

This is a database-backed application built with Java and SQL. It is intended to be an updated system for FarMart's inventory, marketing, delivery, and sales.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Peter Morales	2023/02/14
2.0	Updated design document to reflect current progress	Peter Morales	2023/03/02
3.0	Drafted section 3.1 and added ER diagram	Peter Morales	2023/03/23
4.0	Updated all sections to better reflect completion of the project to this point	Peter Morales	2023/04/06
5.0	Updated sections to better reflect project status Update UML/ER diagrams	Peter Morales	2023/04/21
6.0	Updated sections to better reflect project status Update UML/ER diagrams Added Bibliography	Peter Morales	2023/05/08

Contents

1 Introduction

This document provides the purpose and layout of the design being implemented. The Project itself is a database-backed application written in java for FarMarT (or FMT for short) to replace their old paper filing system. Its purpose is to keep track of all invoice data and be able to produce specific sales reports by user request. Invoice data includes salesperson and customer information; products, services, and equipment that each contain their own specific data, and various calculations for sales totals.

1.1 Purpose of this Document

The purpose of this document is to give technical descriptions for a database-backed application, in the form of a design document, that keeps track of sales information, and produces reports. It will achieve this by explaining the functionality of different components and how they work together and provide a detailed example for how to create an application of this type.

1.2 Scope of the Project

The scope of this project is to store data put into the system by a user. This will be achieved by building a program that is database-backed to protect sensitive information of the customers, users, inventory. FarMart employees (users) can then use the information within the system to produce specific reports that are useful to the business.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Equipment - Includes machinery. Each piece of equipment includes an alphanumeric model number. Customers have the option of purchasing or leasing a piece of equipment. When purchased, there is a single purchase price that the customer pays that does not include a tax. However, when leased the customer signs a contract and agrees to pay a fee for every 30-day period. The lease begins and ends on specific dates and the total charge is based on the number of days between the start/end date.

Products - Includes things such as top soil, fence posts, fertilizer, and fence posts. Each product is sold in quantities by a particular unit. For example, fertilizer may be solid in liters while fence posts are sold in bundles.

Services - are services that a customer contracts with FarMart. Services are billed on a per-hour basis. When purchased, a customer pays for a certain number of hours (which can be fractional) and so the total cost is the cost per-hour times the number of hours contracted.

Invoices - a collection of items purchased by a particular customer. Each invoice includes a unique identifying alphanumeric code, the date that the invoice was created, the customer (and their info) that the invoice is for, the store that the sale was made at, and the items that were part of the invoice (which may include equipment, products, and services). Depending on the customer and items on the sale, various taxes are also applied.

MySQL - MySQL is a relational database management system.

MariaDB - MariaDB is a relational database management system.

1.3.2 Abbreviations & Acronyms

FMT FarMarT

IEEE Institute of Electrical and Electronics Engineers

EDI Electronic Data Interchange

OOP Object-Oriented Programming

JSON JavaScript Object Notation

API Application Programming Interface

ADT Abstract Data Type

SQL Structured Query Language

JDBC Java Database Connectivity

2 Overall Design Description

The design and implementation of the objects, that is the foundation for the system, is what is to be made first. This included parsers that will read data from flat files and export them to an interchange format of JSON. Next, objects are further refined to be woven into each other to generate summary reports that compile the data together. Design of a SQL database, to model the objects that will support the application, is to be created next. After completion of the SQL database, parts of the code is refactored to load the objects from a database rather than from flat files. The implementation and use of an API was created next to persist (save) data to the database. And finally, a sorted list ADT was designed, implemented, and integrated into the application.

3 Detailed Component Description

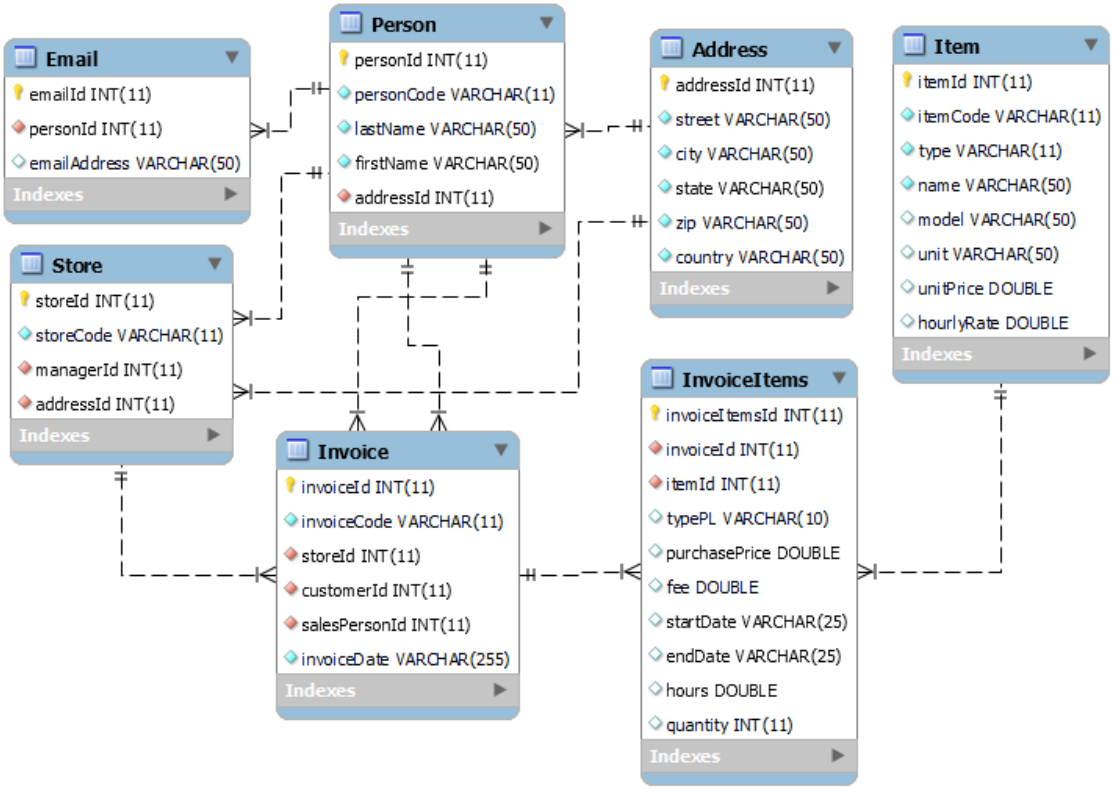
The components of this project are classes that model objects, classes that utilize inheritance, and classes for parsing files. There is also specific code that takes java objects and converts them into the JSON format. The code to convert to JSON is within the DataCoverter class.

3.1 Database Design

A SQL database is designed using MariaDB/MySQL to store data that models the java objects, as entities. The tables are designed to support the data related to the entities as well as the relationship between the entities. Primary and foreign keys are implemented in a way that appropriately relates tables to each other and each store, person, item, and invoice has a constraint that the requires their identifying code to be unique in the database. All tables include either a one-to-one, or one-to-many relationship.

3.2 Entity-Relation Diagram

Figure 3.2

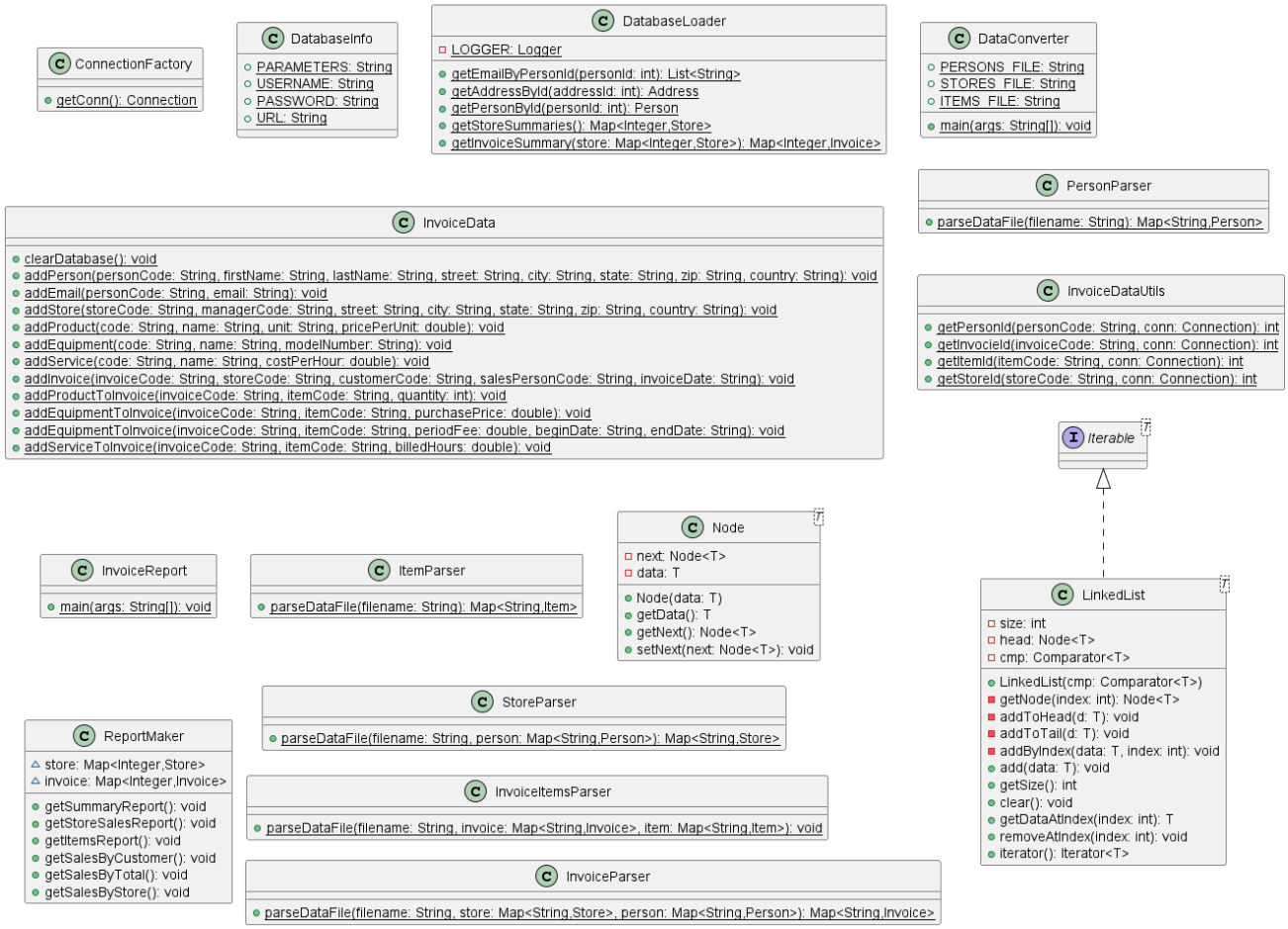


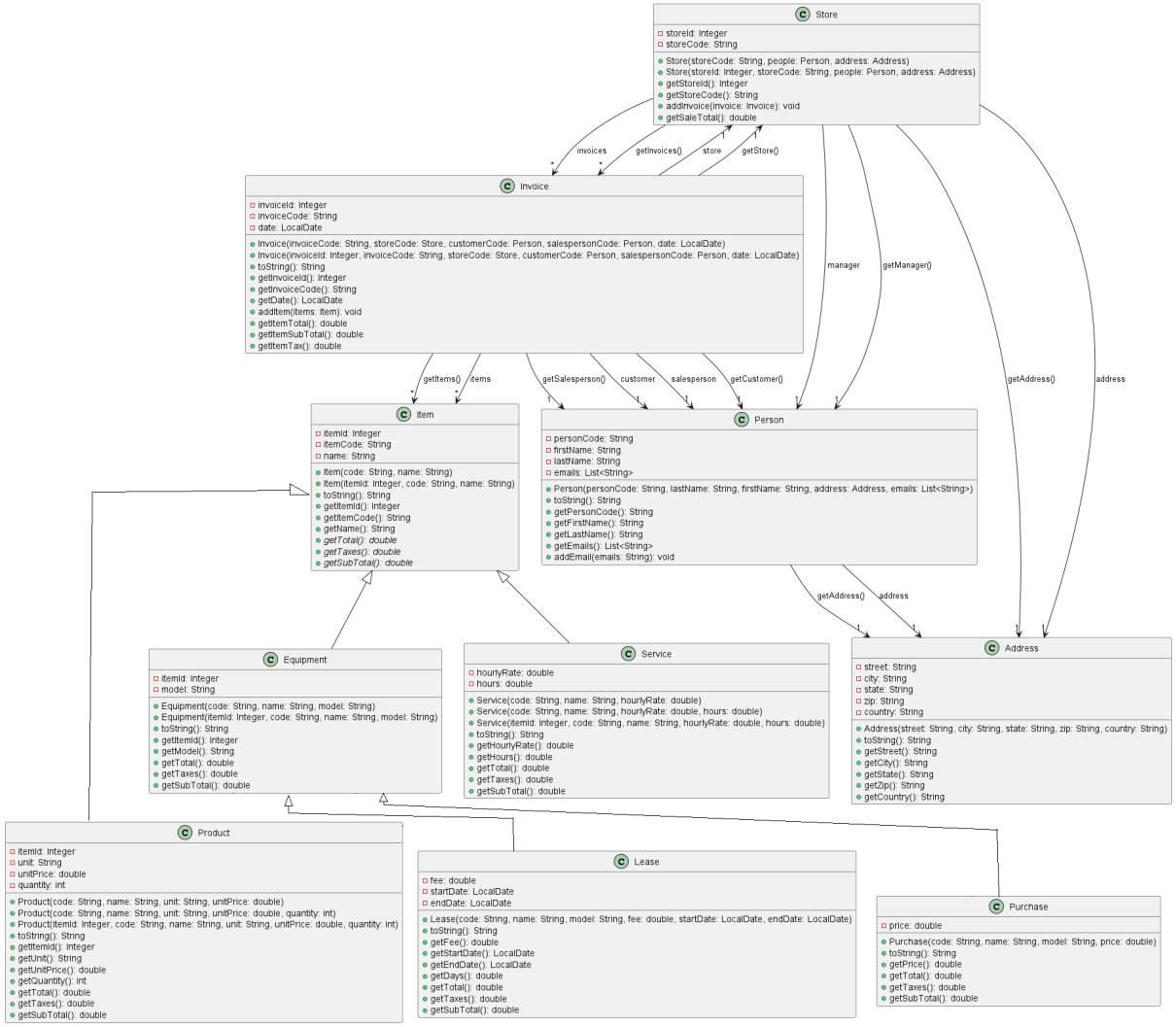
3.2.1 Component Testing Strategy

After the database is built, it is extensively tested to execute specified queries. An example of one of the queries used to test the data is a query that would detect a potential instance of fraud where an employee makes a sale to themselves (the same person is the salesperson as well as the customer). This was possible thanks to the unique key constraints which makes it easily identifiable if there is an instance of the customer code being the same as the salespersons code.

3.3 Class Model

Figure 3.3





3.3.1 Component Testing Strategy

Testing was done by using CSV files as mock data for which the program will parse and output into the desired formats. Moving forward in the design process, standard OOP design practices were used such as inheritance, abstraction, encapsulation, and polymorphism to further implement how the classes related and talked to each other.

3.4 Database Interface

To implement the database, the application is refactored so that it can load data from the database rather than from flat files. More specifically, methods are written in order to use the Java Database API (JDBC) to load the data into the java objects and produce

the same style of reports as before.

3.4.1 Component Testing Strategy

To test the database interface, various queries are implemented to retrieve data from the database tables and allow for reports to be displayed in the same manner as the flat files. If reports came back with misinformation or displayed improperly, the problem is easy to identify and locate from within the program. For example, bad data or no data indicated an error with or within the database. Improper display of reports is likely due to java code errors.

3.5 Design & Integration of Data Structures

To extend functionality to the application, a linked list data structure is implemented to maintain an ordering for each of the invoices. The three orderings provided were a sort by customer last name and then first name, total value of the invoice from highest to lowest, and by store code. The linked list is designed with an iterable interface and is able to take in a custom comparator that helps maintain the ordering of the linked list. A parameterization of the linked list is also implemented to ensure the linked list can take in all types of data.

3.5.1 Component Testing Strategy

Testing is checked by producing the reports and inspecting the outputs to ensure that the comparator and linked list are maintaining the ordering by the specified criteria. Furthermore, helper functions that added functionality are to be tested to make sure they execute properly.

3.6 Changes & Refactoring

When modeling the classes and producing reports from CSV files, initially a linear search was used in comparing manager codes in the store.csv files with the person codes in the Persons.csv files. While this initially worked, sorting people into a Map was found to be a better solution when it came to design by allow for easier access to the data when producing the reports.

When designing the database, tables were refactored by implementing unique key constraints and not allowing null values for specific columns.

4 Bibliography

Mockaroo: http://www.mockaroo.com
Generate Data: http://www.generatedata.com
log4j: https://logging.apache.org/log4j/2.x/
PlantUML