DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

FarMarT Data Technical Design Document

Ariana Shelton and Shane Thompson

5/12/2023

Document Version 6

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Ariana Shelton and Shane Thompson	2023/02/16
2.0	Description of invoice data was described	Ariana Shelton and Shane Thompson	2023/03/03
3.0	Description of SQL database and update on past descriptions	Ariana Shelton and Shane Thompson	2023/03/23
4.0	Description of JDBC and updates on SQL database	Ariana Shelton and Shane Thompson	2023/04/07
5.0	Description of InvoiceData class	Ariana Shelton and Shane Thompson	2023/04/21

Contents

1. Introduction

The newly promoted Pablo Myers has recently started updating his business's data for his family's chain of stores, FarMarT. In order to be more convenient and efficient, Pablo decided to use a new data system to update the information. The system contains models of the data for his stores, customers, and items that his customers purchase which include products, services, and equipment.

1.1 Purpose of this Document

This document details the design behind the system of converting multiple stores' data into readable and usable format using DataConverter class, InvoiceReport class, and all the other classes implemented within the DataConverter and InvoiceReport classes. Under these classes, tax/pricing calculations and product transformations are performed as well. To reference and update the business's data, the system uses a supporting database in SQL without directly affecting its foundation.

1.2 Scope of the Project

CSV files containing store info, personal info, and items must be printed to separate formatted JSON/XML files using the DataConverter class.

There are XML files that contain invoices and the items purchased in each invoice, but there is no cost of items, taxes, and total cost output. The purpose of the InvoiceReport class is to fix these problems by doing exactly that, this class loads in these XML files and calculates cost of items, taxes, and adds them to find a total cost, and outputs the three values.

This project uses several classes in the data converter to take in CSV files and print to XML and JSON files. The data converter is used for stores which allows converting personal info, store info, and items that are being stored in CSV files to a more readable format in either a JSON or XML format.

The InvoiceReport class calculates the total cost of all items, taxes, and total cost by using the given CSV file, assigning the item's data to the item, and calculating the pricings using mathematical methods.

The cost of Products is calculated by the unit price multiplied by the quantity, taxes are calculated by the cost multiplied by 0.0715.

The cost of Services is calculated by the hourly rate multiplied by hours, taxes are calculated by the cost multiplied by 0.0345.

The cost of Purchased Equipment is given by the price and there is no tax.

The cost of Leased Equipment is calculated by the monthly fee multiplied by the number of months. To calculate the taxes, the cost is inserted into a tax bracket. There is no tax for anything that costs less than $10,000, a $500 tax if the cost is less than $100,000, and a $1,500 tax for anything else.

An SQL database was created to store data more effectively in tables. These tables store the exact same data that is used in the DataConverter and InvoiceReport but can combine them into one database for more convenience.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Invoice – A list of goods sent, or services provided.

Products – Items offered by the store that can be bought in bulk.

Services – Assistance provided by the store, billed by hours.

Equipment – Machinery that can be purchased or leased.

1.3.2 Abbreviations & Acronyms

FarMarT – FMT

JSON – JavaScript Object Notation

XML - Extensible Markup Language

CSV – Comma Separated Values File

SQL – Structured Query Language

API – Application Programming Interface

2. Overall Design Description

The project converts data from CSV files into more readable strings and uses that data to calculate total costs of invoices.

2.1 Alternative Design Options

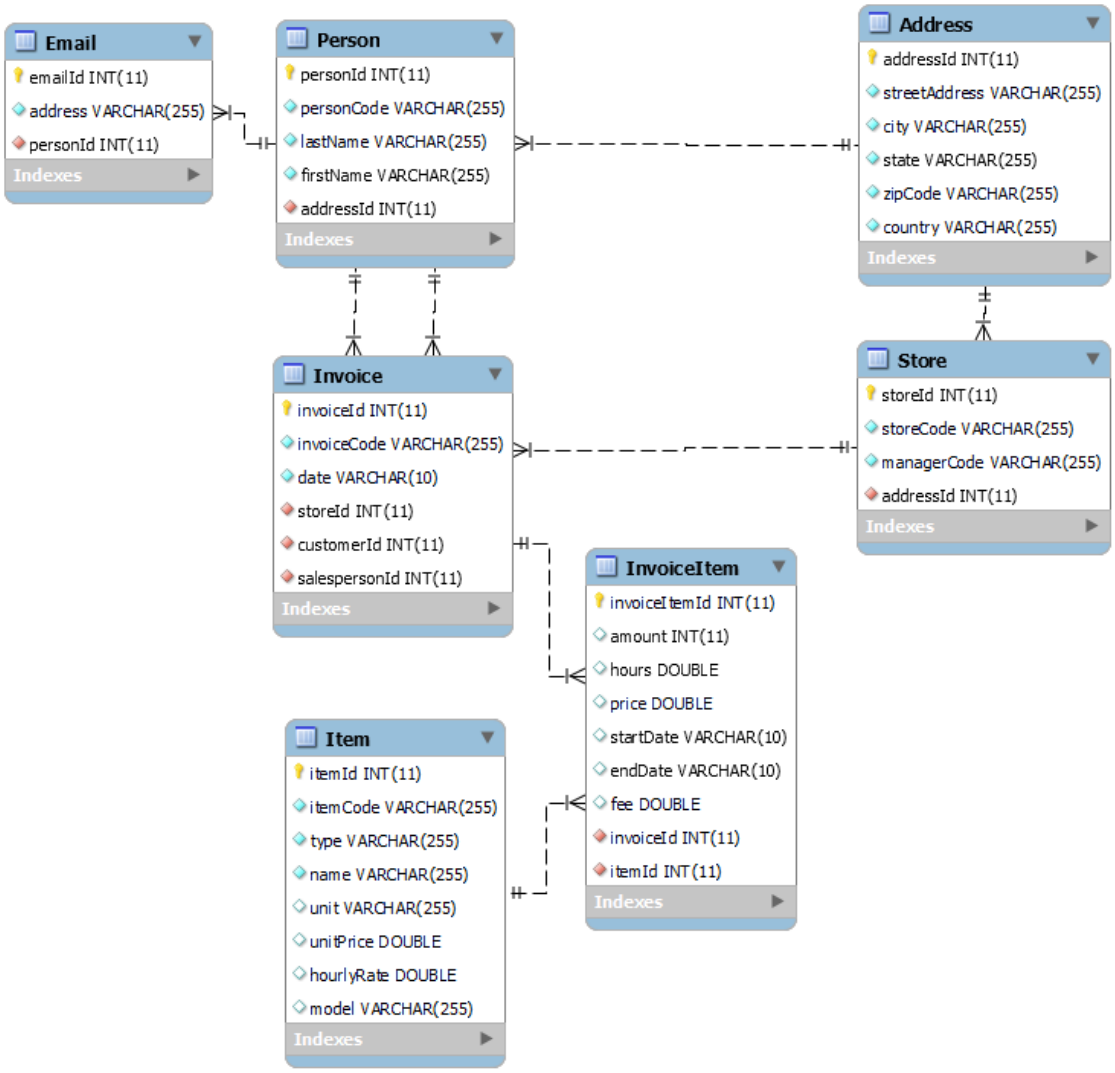
Since the items invoices have different types of data pertaining to it, instead of doing conditionals, the system can use polymorphism and create methods that will take any type of parameter (equipment, product, or service). This alternative design would make the code more convenient and compact.

Another alternative design the project can perform would be to make the Invoice class implement the Store and Person class and have the Invoice Items class implement the Items parent class. The system could do this instead of creating new instances of parameters in Invoices that are already used for the Store or Person or Items class. The parameters that could be implemented for the Invoice and Invoice items class are the store, person, and item codes. Doing so would make the code more convenient.

3. Detailed Component Description

This section details the classes and database.

3.1 Database Design



The Person and Store classes both have foreign keys that point to an address because a single address can belong to multiple people or stores.

Emails take a personId as a foreign key because people can have multiple emails.

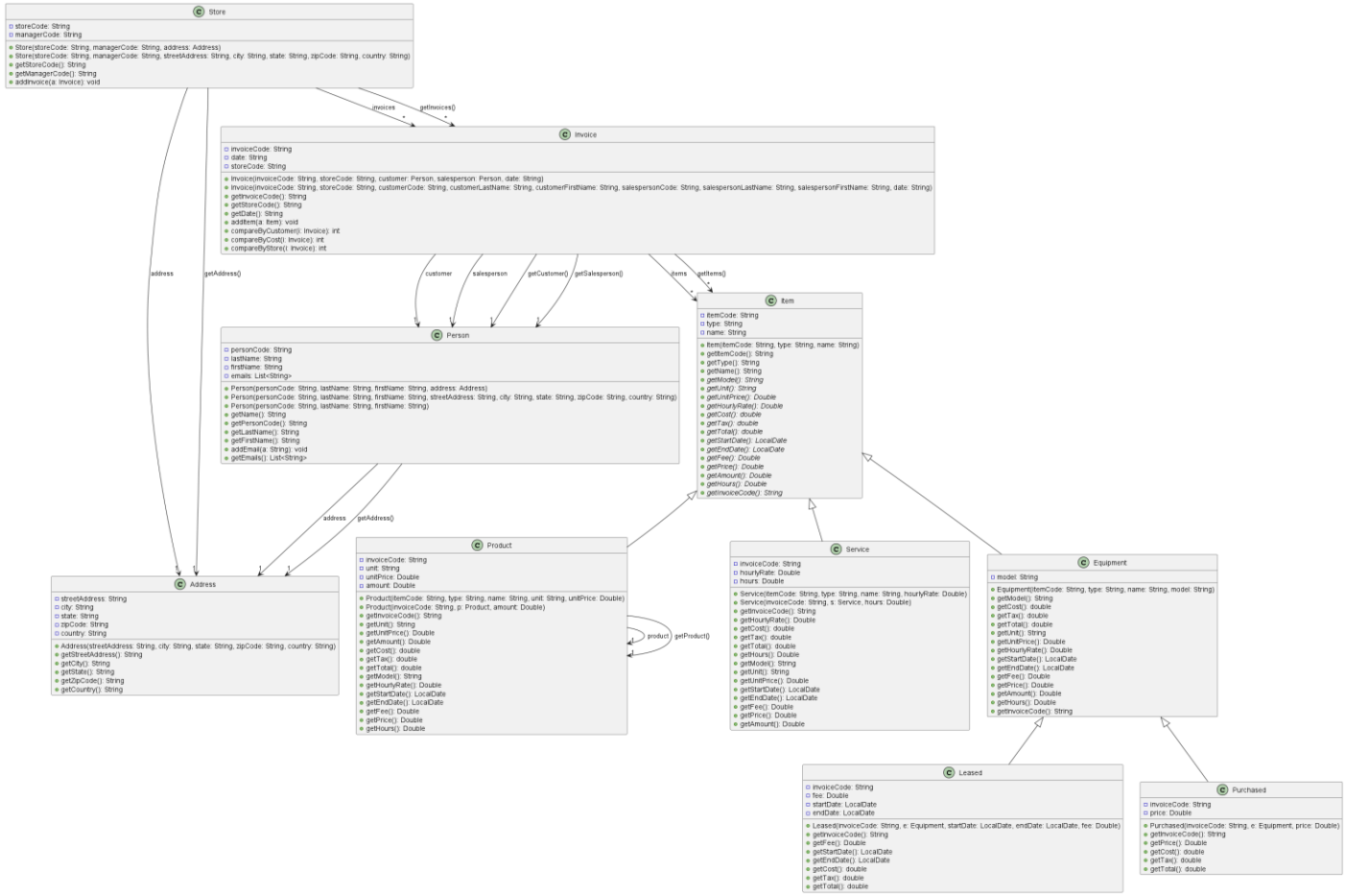
The inclusion of the InvoiceItem class was necessary due to the fact that there is a many to many relationship between Invoice and Item.

The invoice class takes 3 different foreign keys, one for a store, one for the customer, and one for the salesperson. Both the customerId and salespersonId are primary keys from the Person class.

3.1.1 Component Testing Strategy

To test the database, test data was inserted into all the tables and several queries were run to ensure that the design of the database was effective. Examples of queries include testing if new data could be inserted, old objects could be updated or destroyed without causing errors elsewhere. Many queries were also allowed to test the integrity of the database to see what the design would allow. This included testing if duplicate values could be inserted, and testing if people could sell items to themselves. An example of a query used to test this design was to delete a person's record from the database. This helped test the integrity of the database because the design should not allow for a person's record to be deleted without their email(s) also being deleted. Testing proved that this database design does not allow for situations like that to happen.

3.2 Class/Entity Model



The DataConverter class takes the given CSV files and constructs instances of the given objects for items, stores, and personal info. The project prints the data into readable output files in XML and JSON.

The InvoiceReport class takes the Invoices and InvoiceItems CSV files and outputs the total cost of items/services purchased, taxes, and a grand total.

In this model, Address and Item are abstract because they are only used to store similar data that is used by several other classes, they do not need to be constructed themselves. Address is used by Person and Store, Item is used by Product, Service, and Equipment. The three types of Items need several constructors, one that is used in the data converter, and another to be used in the invoice report when it is an invoice item. The Equipment class has two subclasses, Leased and Purchased, because there are two types of equipment that can be bought in an invoice, and they need separate constructors.

3.2.1 Component Testing Strategy

The project will use generated data from various online tools that can be prompted to give random data to test. Once generated, the system will use test classes for each of the object classes. Those classes run the data converter and, using Junit, compare the actual results to what the expected should be to test the converter's accuracy. Since the project uses specific object test classes, it is easier to find out where the faulty code is located.

The InvoiceReport class will be tested by running the class and comparing the calculated outputs to the expected outputs for cost of items, taxes, and total. The comparison will be by conditionals to test its accuracy. If they are equal, true will be returned, if not, false will be returned.

3.3 Database Interface

The interface is used to connect the SQL database to Java. Using SQL queries, the system loads in records from the database and converts them into java objects. Each type of item is added to a list, the lists created include people, stores, invoices, items and invoiceItems. Each person in the list of people will contain their code, name, address, and a list of their emails. Each store in the list of stores will contain their code, managerCode, address, and a list of invoices. Each invoice will contain a code, the customer's data, the salesperson's data, the date of the invoice, and a list of invoiceItems. Each item will contain a code, type, name, and the data indicated by the type, model for equipment, unit and unit price for products, and hourly rate for services. Each invoiceItem will include data of the corresponding item, the corresponding invoice code, and the data indicated by the type, price for purchased equipment, a monthly fee and start and end date for leased equipment, hours for services, and amount for product.

Users would be restricted to only being allowed to put in an appropriate amount of information so that they match what the classes can take in.

Everything will originally be a string, so to test this for double and integer values, those strings that should be doubles and integers will be parsed.

For the database interface, the java classes could be changed to match the tables used in the SQL database.

The API will take in data for Items up to the type first, and then by using the type, the API is able to tell which areas should be null and which to input the rest of the data into.

3.3.1 Component Testing Strategy

To test the API, the expected output was compared to the actual output and further revisions were made based on its result. The expected output is a summary report for each invoice, each store, and final a detailed report for each invoice. The summary reports output the total tax and cost for each invoice or store and give a grand total at the very end. The detailed reports will output each item in the invoice, that items cost, and then output a grand total of each invoice.

3.4 Design & Integration of Data Structures

The driver class will create lists for stores, people, invoices, items, and invoiceItems using the variables stated earlier. Using these lists and the data stored in them, the driver will generate reports for each invoice, each store, and detailed reports for each invoice. The driver loops through each invoice in the list of invoices for the summary report of invoices, it loops through each store for the summary report of each store, and it then loops through each invoice to get the detailed reports for each invoice.

The InvoiceData class contains adder methods to add instances of each type of data and a method that clears each table in the database. The adders include, addPerson, addEmail, addStore, addProduct, addService, addEquipment, addInvoice, and adders that add each type of item to an invoice.

3.4.1 Component Testing Strategy

This class was tested by adding data using each method and seeing what each method allows. The methods were adjusted to only allow for valid data to be added and not allow for any repeated data.

3.5 Changes & Refactoring

Originally, the Person and Store classes extended the Address class. This was changed to rather have the Person and Store constructors take in an Address object.

4. Additional Material

Three sorting methods are included in the invoice class. These methods sort by customer name, total cost, and store. In a separate class named Sorter, these methods are used in three separate insertion sort methods and can be used to output invoice reports in a sorted manner.

5. Bibliography

[1] *APA 6 – Citing Online Sources*. (n.d.). Retrieved March 19, 2021, from https://media.easybib.com/guides/easybib-apa-web.pdf

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

[3] "Classes and Objects in Java." *GeeksforGeeks*, GeeksforGeeks, 14 Feb. 2023, https://www.geeksforgeeks.org/classes-objects-java/.