DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

FarMarT Business Operations

CSCE 156 – Spring 2023

Matthew Schlatter

2/17/2023

1.0

This document outlines a sales subsystem for a farming company, FarMarT.

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	Matthew Schlatter	2023/02/17

Contents

1. Introduction

This design will be created for the family-owned business FarMarT (FMT), a regional chain of B2B (business-to-business) stores that sell farm equipment and supplies to farmers large and small. With new management in town, a revamp is favorable to increase profits. This part of the independently developed project is the sales subsystem, efficiently gathering and sorting different invoices, customers, and stores.

1.1 Purpose of this Document

This document should be clear and concise enough so that, if necessary, a person with enough technical knowledge will be able to reproduce this design in any coding language. It lays an outline for the entire project and is easily followed.

1.2 Scope of the Project

This project is designed to be utilized by FarMarT, specifically following their business model and procedures. It is designed to make orders easily accessible and trackable. This subsystem efficiently collects data and stores it in a database using Electronic Data Interchange where it can be gathered to create reports. This system does not deal with customer issues or marketing, only the tracking of sales.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

All necessary definitions for this document can be found here.

God-class – an object that controls far too many objects in the system and is all-seeing of the system.

1.3.2 Abbreviations & Acronyms

All necessary abbreviations and acronyms for this document can be found here.

FMT – FarMarT (Company Name)

2. Overall Design Description

This project takes in many different data points and provides different data files based on the provided information. Data files include invoices with customer orders and order information, customer data including name and contact information, as well as store information where the order was placed. Data is imported through CSV files and exported as XML and/or JSON files.

2.1 Alternative Design Options

Other alternatives would be to design the project using a god-class. However, this is disadvantageous because it is bad abstraction and encapsulation. One class would be doing too much of the heavy work, instead of splitting the work up into areas that make sense for the work. Conversely, utilizing subclasses allows common sets of data, such as a customer name or address, to be easily modeled and applicable to other higher-level classes, such as a customer's profile. Subclasses avoid unwanted god-classes.

3. Detailed Component Description

[Introduce this section here]

3.1 Database Design

[This section will be used to detail your database schema design (Phase III). In earlier phases this section may be omitted or a short note indicating that details will be provided in a subsequent revision of this document. See the full rubric for what belongs in the introduction!]

3.1.1 Component Testing Strategy

[This section will describe your approach to testing this component.]

3.2 Class/Entity Model

Outline of classes and their specific variables and methods.

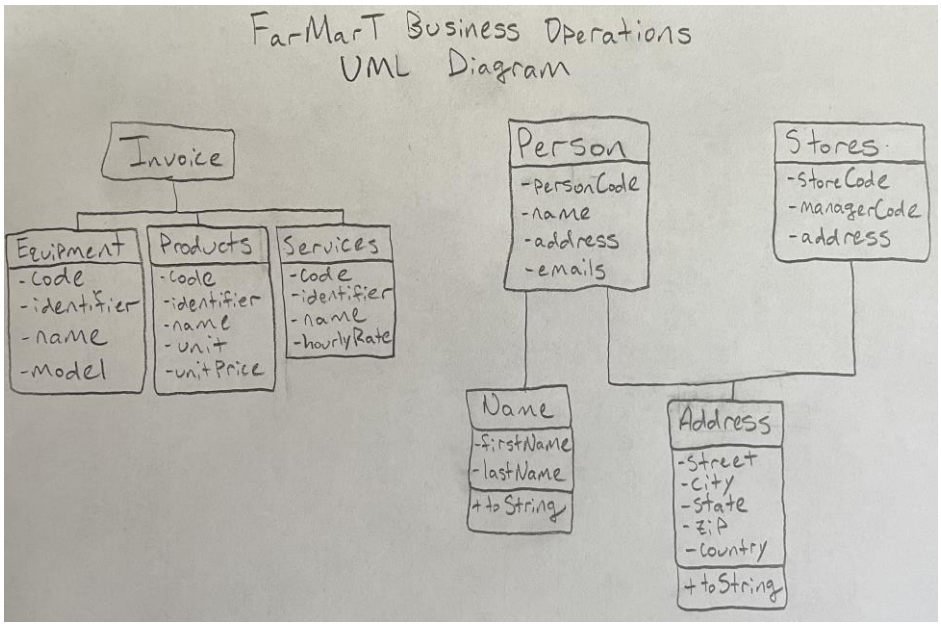

Figure 1: UML Diagram for the FarMarT operations, outlining different classes and variables

3.2.1 Component Testing Strategy

Test files with randomly generated data were input into the classes above to test the effectiveness of scanning and printing data. Data was imported through CSV files and exported to XML/JSON files. Each class passed their respective tests.

3.3 Database Interface

[This section will be used to detail phase IV where you modify your application to read from a database rather than from flat files. This section will detail the API that you designed—how it conformed to the requirements, how it worked, other tools or methods that you designed to assist, how it handles corner cases and the expectations or restrictions that you've placed on the user of the API. In earlier phases this section may be omitted or a short note indicating that details will be provided in a subsequent revision of this document. See the full rubric for what belongs in the introduction!]

Table 1: Average Performance on Assignments; on-time vs. late and individual vs partners. In general, captions for Tables should appear *above* the table.

	1	2	3	4	5	6	7
On-time	93.16% (78.46%)	88.06% (72.31%)	87.89% (67.69%)	89.37% (56.92%)	83.42% (29.23%)	88.40% (53.85%)	74.56% (75.38%)
Late	88.75% (12.31%)	85.28% (20.00%)	70.32% (15.38%)	90.40% (15.38%)	82.74% (44.62%)	94.22% (15.38%)	N/A
Diff	4.42%	2.79%	**17.57%**	1.03%	0.68%	5.82%	-
Individual	NA	88.43% (73.85%)	82.32% (33.85%)	87.22% (27.69%)	86.40% (23.08%)	82.67% (26.15%)	
Pairs	NA	83.55% (18.46%)	86.22% (49.23%)	91.00% (46.15%)	78.53% (49.23%)	92.83% (46.15%)	
Diff	NA	4.88%	3.90%	3.78%	7.87%	10.16%	

3.3.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

3.4 Design & Integration of Data Structures

[This section will be used to detail phase V where you design an original data structure and integrate it into your application. In earlier phases this section may be omitted or a short note indicating that details will be provided in a subsequent revision of this document? See the full rubric for what belongs in the introduction!]

3.4.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

3.5 Changes & Refactoring

[During the development lifecycle, designs and implementations may need to change to respond to new requirements, fix bugs or other issues, or to improve earlier poor or ill-fitted designs. Over the course of this project such changes and refactoring of implementations (to make them more efficient, more convenient, etc.) should be documented in this section. If not applicable, this section may be omitted or kept as a placeholder with a short note indicating that no major changes or refactoring have been made.]

4. Additional Material

[This is an optional section in which you may place other materials that do not necessarily fit within the organization of the other sections.]

5. Bibliography

[This section will provide a bibliography of any materials, texts, or other resources that were cited or referenced by the project and/or this document. You *must* consistently use a standard citation style such as APA [1] or MLA.

[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.