YRLess Sales

Technical Design Document

Immanuel Soh
isoh2@huskers.unl.edu
University of Nebraska—Lincoln

Spring 2024
Version 2.0

This technical design document is intended to document the design of a sales data management system for YRLess, a large cell phone and wireless company. This system processes sales and related data, then produces reports using the processed data.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Immanuel Soh	2024/02/29
2.0	Updated Abstract, Introduction, Purpose, Definitions, Design Overview, Alternative Design Options, and Detailed Component Description	Immanuel Soh	2024/03/22

Table of Contents

1. Introduction

This document shows the design of a sales data management system for YRLess, a new company formed from the aggregation of several small-to-regional cell phone companies and stores by LSP holdings.

This system is written in Java, Object-Oriented, database-backed, and supports YRLess's business model through implementation of their business rules and providing the functionality listed below.

YRLess has a business model consisting of four different types of products and services:

- Products are "items" that can be bought or leased. When a product is purchased, a customer pays for the whole cost of an item at once along with a 6.5% sales tax. However, when a product is leased, the product's original cost is marked up by 50% with no taxes. The customer then only pays the marked up cost divided by the total number of months in the lease period (thus paying only for the first month's cost up front). Leased products do not have taxes on them.
- Services are a type of "item" performed by individual employees and each have per-hour rates. The customer is charged at the hourly rate for the service multiplied by the number of hours the service is provided. A 3.5% service tax is then applied to this subtotal.
- Data plans are "items" sold by the gigabyte. The customer is charged by the cost per gigabyte multiplied by the number of gigabytes purchased. A 5.5% sales tax is then applied to the subtotal.
- Voice plans are "items" billed based on 30 day periods, with the minimum cost the cost per 30 day period. When sold, the customer can choose the number of days they want the plan, which is not restricted to 30 day periods. The customer is then charged by according to the rate for every 30 day period, with the minimum cost being the cost per 30 day period. A 6.5% sales cost is then applied to this subtotal.

The sales management system keeps track of sales, or collections of "items," and produces reports about them. Each sale includes:

- A unique identifying alphanumeric code
- The date of the sale

- The customer that made the sale and their information
- The store that the sale was made and its information
- The employee that made the sale
- The “items” that were purchased in the sale

1.1 Purpose

The purpose of this document is to outline the technical design of the sales data management system and provide an overview for the system’s implementation.

Its main purpose is to —

- Detail the functionality which will be provided by each component or group of components and show how the various components interact in the design
- Provide a basis for the sales data management system’s detailed design and development

This document is not intended to address any installation or configuration details of the actual implementation as these details are provided in technology guides produced during the course of development.

1.2 Scope

The scope of this project is the design of a sales data management system that is database-backed and supports YRLess’s business model through their rules and providing several functionalities listed below. The system is also responsible for producing several different sales reports.

The scope of this project does not include systems for marketing, inventory, billing, etc.

This is because these responsibilities have been delegated to other teams who created their own independent systems that handle each of these needs.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Gson	A CSV to JSON data formatter
MySQL	A SQL workbench
XStream	A CSV to XML data formatter

1.3.2 Abbreviations & Acronyms

2. Design Overview

The design of this sales data management system aims to support several features for YRLess, including—

- Representing stores, customers and managers, products, plans, and services in a system through classes to allow for easy data management
- Loading and converting data from flat file formats into databases using XStream, Gson, and MySQL
- Keeping track of sales, income, and other data, generating reports once all data is compiled
- Storing data about sales in a database through SQL

2.1 Alternative Design Options

When coming up with the class design for this system, a design was proposed that would include functions to handle data conversion in a main function of a program. However, this functionality ended up getting separated into different classes in order to follow the single responsibility principle, as handling data conversion directly in the main function would allow for less freedom later on in the design.

3. Detailed Component Description

In this section of the document, the different components of the system, their designs, and the testing strategies behind them are described in detail, including the database, the class structure, the database interface, and the sorted list data structure.

3.1 Database Design

[This section will be used to detail your database schema design (Phase III). For your draft, you should provide a sketch of your ER diagram (which may be replaced with a more formal one after your implementation. Identify all tables and their purpose; the columns in each table, etc. Do not include a lot of text; your ER diagram should provide enough technical details. Your text should not be redundant to your diagram. Instead, your text should *justify* the design; make reference to the introduction and to the client's business model.]

3.1.1 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. How was test data generated (if a tool was used, this is a good opportunity for a citation). How many test cases did you have; how many of each type? *Justify* why that is sufficient. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?

You may refer to the course grader system as an external testing environment "provided by the client" or "another QA/testing team".]

3.2 Class/Entity Model

[This section should detail your classes–their state, interface and how they relate to each other. Your draft should include a sketch (hand or tool generated) of your classes using a UML diagram. Figures and tables should have proper captions and be referenced in the main text just like in Figure 1. Don't have one giant UML diagram; break it up into subfigures, collecting related classes as appropriate. Your draft needs to provide enough detail that we can give feedback on your design before you submit the code for each phase. Your sketches should be replaced with formal diagrams in later drafts.

Identify which classes are responsible for each feature. Classes should follow the *Single Responsibility Principle*. This section should be updated throughout each phase as you add more classes.]

Figure 1: A UAV (Unmanned Aerial Vehicle) soars above Memorial Stadium. Figures should be numbered and properly captioned. This is just an example of how to properly include a figure.

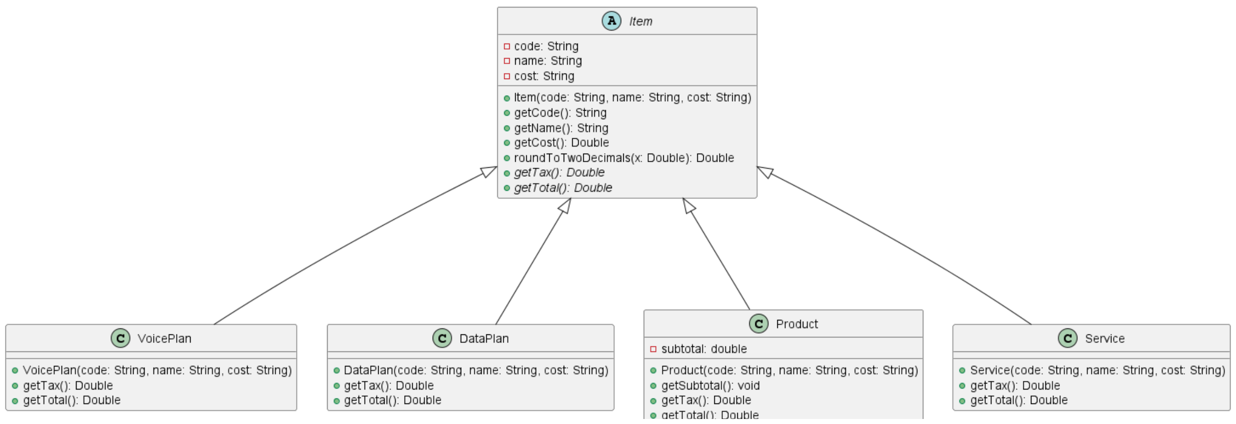

Figure 1: A section of a UML diagram of the system displaying the inheritance used to represent all the different "Items."

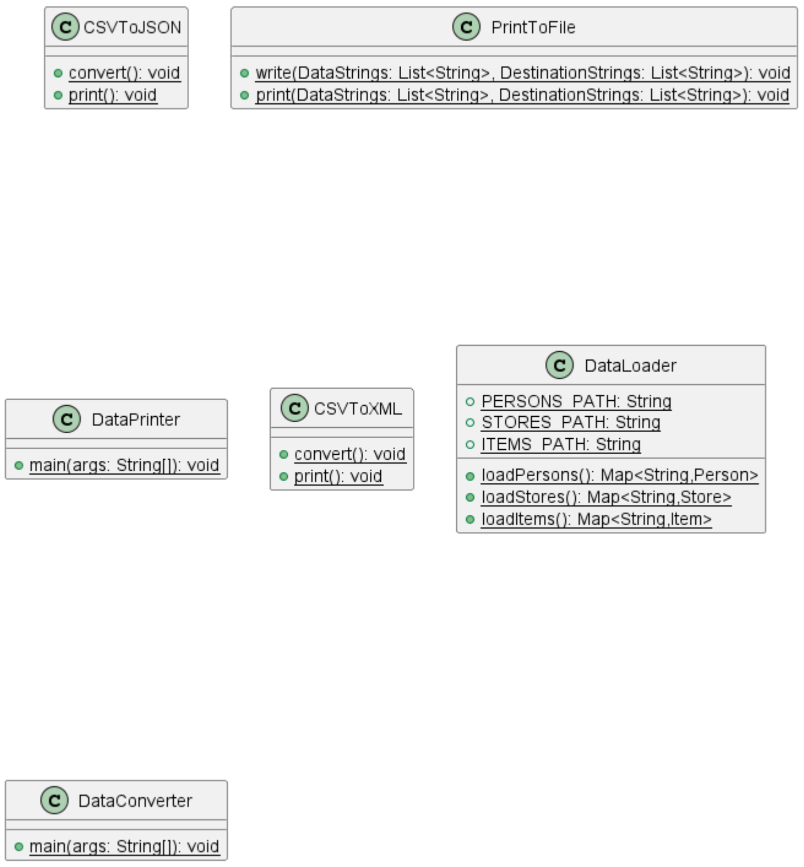
CSVToJSON
convert(): void
print(): void
PrintToFile
write(DataStrings: List<String>, DestinationStrings: List<String>): void
print(DataStrings: List<String>, DestinationStrings: List<String>): void
DataPrinter
main(args: String[]): void
CSVToXML
convert(): void
print(): void
DataLoader
PERSONS_PATH: String
STORES_PATH: String
ITEMS_PATH: String
loadPersons(): Map<String,Person>
loadStores(): Map<String,Store>
loadItems(): Map<String,Item>
DataConverter
main(args: String[]): void

Figure 2: A section of the UML diagram of the system displaying several classes created to handle several different functions for the system, like loading, converting, and printing data.

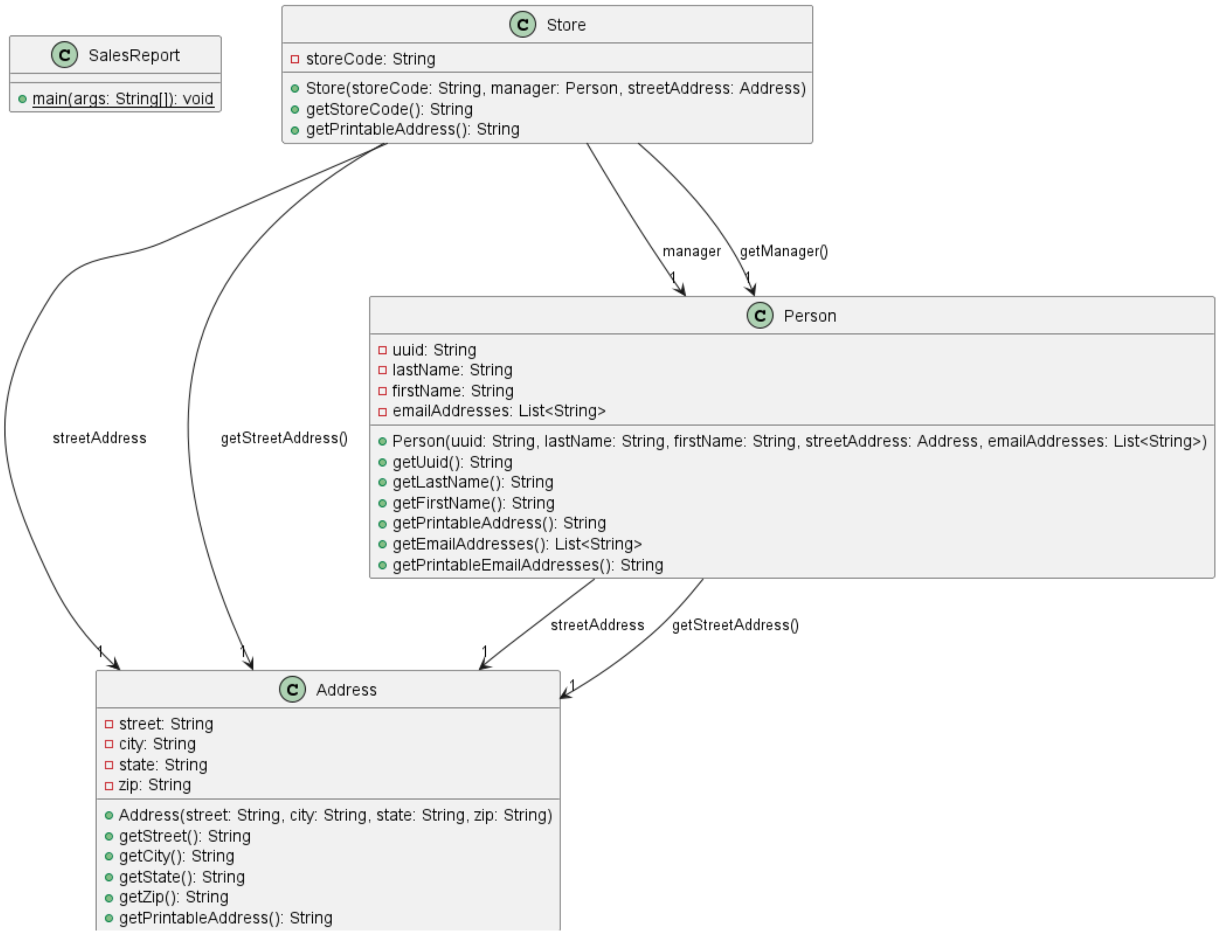
SalesReport
main(args: String[]): void
Store
storeCode: String
Store(storeCode: String, manager: Person, streetAddress: Address)
getStoreCode(): String
getPrintableAddress(): String
manager
getManager()
1
1
Person
uuid: String
lastName: String
firstName: String
emailAddresses: List<String>
Person(uuid: String, lastName: String, firstName: String, streetAddress: Address, emailAddresses: List<String>)
getUuid(): String
getLastName(): String
getFirstName(): String
getPrintableAddress(): String
getEmailAddresses(): List<String>
getPrintableEmailAddresses(): String
streetAddress
getStreetAddress()
streetAddress
getStreetAddress()
Address
street: String
city: String
state: String
zip: String
Address(street: String, city: String, state: String, zip: String)
getStreet(): String
getCity(): String
getState(): String
getZip(): String
getPrintableAddress(): String

Figure 3: A section of the UML diagram of the system displaying more classes that represent objects in the business model like stores, persons, and addresses.

3.2.1 Component Testing Strategy

During the development of this component of the system, test cases representing items, persons, and stores were developed to ensure that the class design could represent the different attributes of the different components of the business model.

An external testing environment with additional test cases was also provided by YRLess, which the test cases developed were modeled after.

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. What is "good" data and what is considered "bad" data and how does your API handle it? An example table is presented as Table 1.]

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. How was test data generated (if a tool was used, this is a good opportunity for a citation). How many test cases did you have; how many of each type? *Justify* why that is sufficient. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?

You may refer to the course grader system as an external testing environment "provided by the client" or "another QA/testing team".]

3.4 Design & Integration of a Sorted List Data Structure

[This section will be used to detail phase V where you design and implement a custom data structure and integrate it into your application. Is your list node based or array based? What is its *interface* and how does it define a sorted list? Is it generic? Why? You can/should provide another UML diagram for this list.]

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. How was test data generated (if a tool was used, this is a good opportunity for a citation). How many test cases did you have; how many of each type? *Justify* why that is sufficient. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?

You may refer to the course grader system as an external testing environment "provided by the client" or "another QA/testing team".]

4. 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.]

5. 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.]

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.