YRLess Sales

Technical Design Document

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

Jaden Miller

Jmiller144@huskers.unl.edu

University of Nebraska—Lincoln

Spring 2024
Version 5.0

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
3.0	Added ER diagram and description	Immanuel Soh	2024/04/05
4.0	Updated Abbreviations and Acronyms, Database Component testing strategy, and Detailed Component Descriptions from Versions 2.0 and 3.0	Immanuel Soh and Jaden Miller	2024/04/19
5.0	Updated Sections 2, 3.1.1, 3.2, 3.3, 3.4, and Bibliography	Immanuel Soh and Jaden Miller	2024/05/08

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
Mockaroo	A test data generator
MySQL	A SQL workbench
XStream	A CSV to XML data formatter

1.3.2 Abbreviations & Acronyms

ADT	Abstract Data Type
CRUD	Create, Retrieve, Update, Destroy
CSV	Comma Separated Values
ER	Entity Relations
JDBC	Java Database Connectivity API
JSON	JavaScript Object Notation
OOP	Object Oriented Programming
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language

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
- Using JDBC to allow the system to interact with the database
- Implementing a custom sorted list ADT to maintain ordering of data through the use of different comparators

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

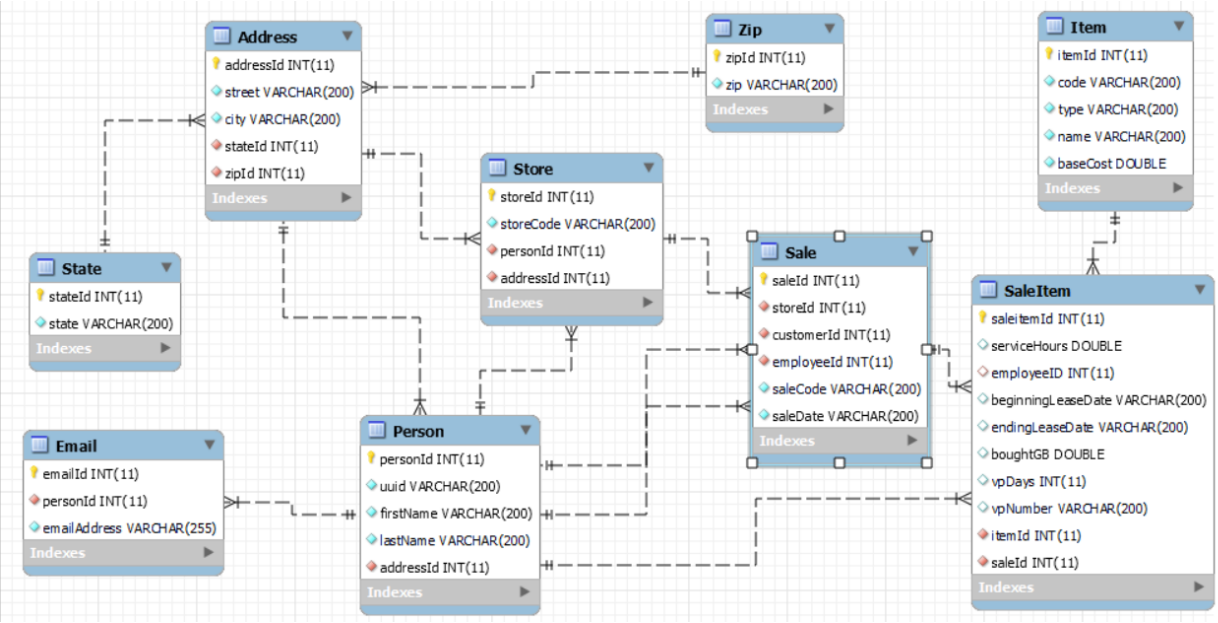

Figure 1: An ER Diagram of the database, which represents the different aspects of YRLess' business model. The Item, Zip, State, Email, and Address tables are used for better normalization of data used for the Store, Person, Sale, and SaleItem tables. The columns under all the tables are used to hold the test data used to test the code.

3.1.1 Component Testing Strategy

An external testing environment with test cases was provided by YRLess, and some new test cases were developed that were closely modeled after these provided ones.

When creating this database using MySQL, tables were created according to the relationships between the different components of YRLess' business model, and the ER diagram above was generated through the Reverse Engineer feature provided by MySQL.

The test data mentioned above representing items, persons, stores, sales, etc. was developed to ensure that the database's design could fully and accurately represent the different attributes of the different components of the business model and then inserted into this database to ensure that it could handle all the data required to be processed. Once

this data was put into the database, several queries were run to test CRUD functions on the different tables, which proved successful.

3.2 Class/Entity Model

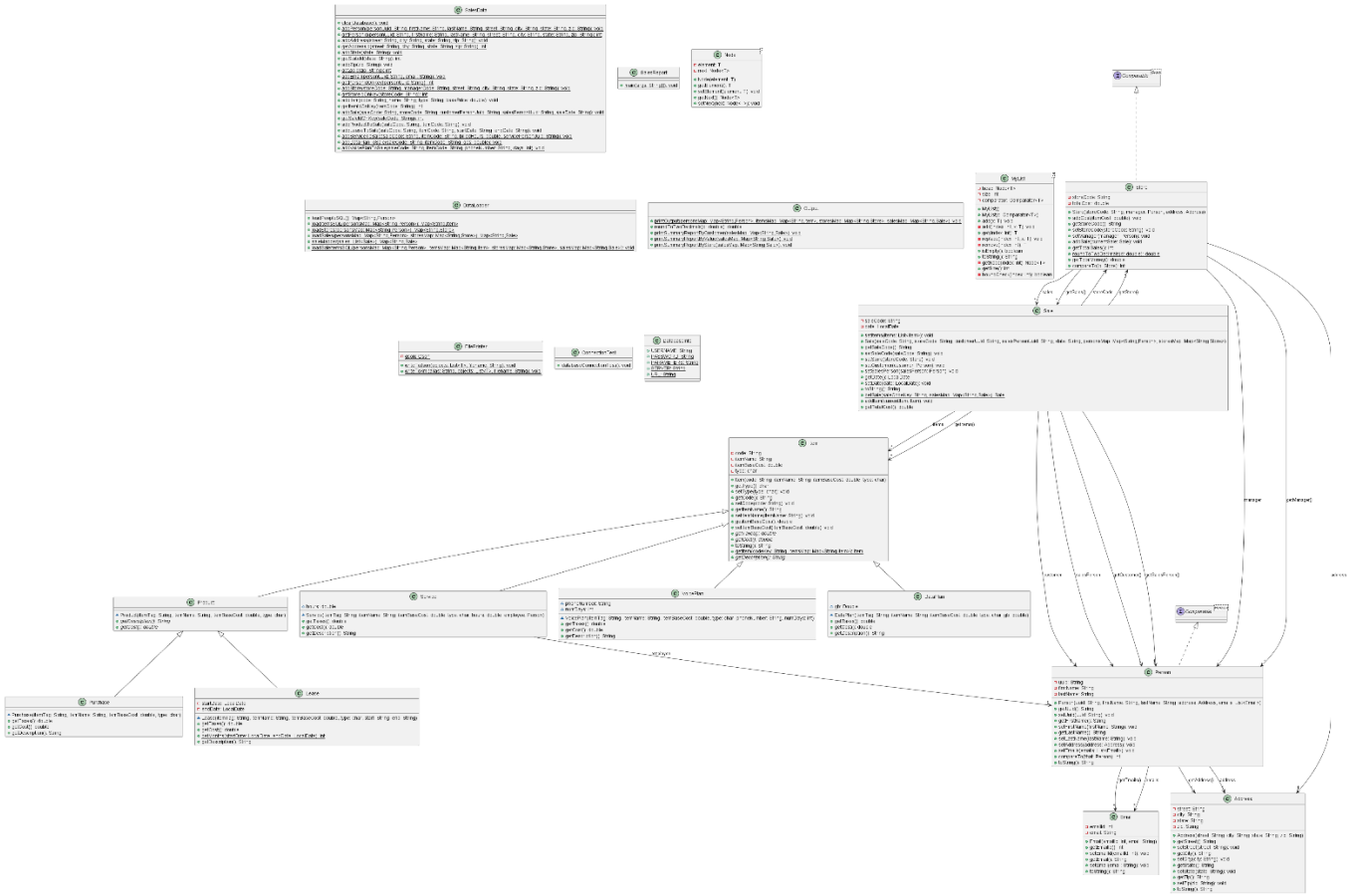

Figure 1: A "Master Diagram" of the system showing the relationships between the different classes in the final version of the diagram. The subsections will be displayed in the figures below.

Figure 2: A section of the UML diagram of the system displaying the Output, FilePrinter, and SalesReport classes. These classes were designed with SOLID principles in mind.

Figure 3: A section of the UML diagram of the system displaying the DatabaseInfo and ConnectionTest classes. These classes were designed to ensure the JDBC connection with the SQL database.

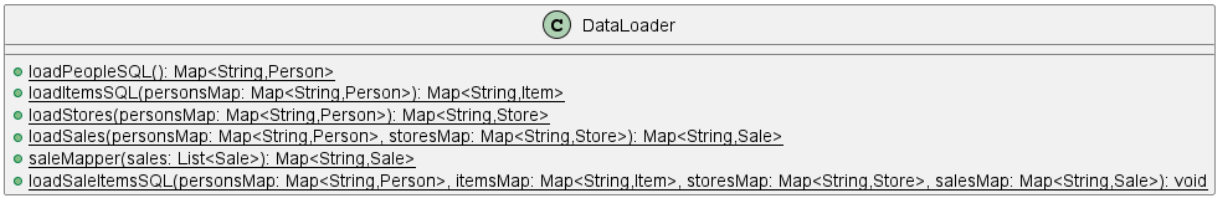

Figure 4: A section of the UML diagram of the system displaying the DataLoader class. This class was designed to handle the responsibility of loading in data from the SQL database.

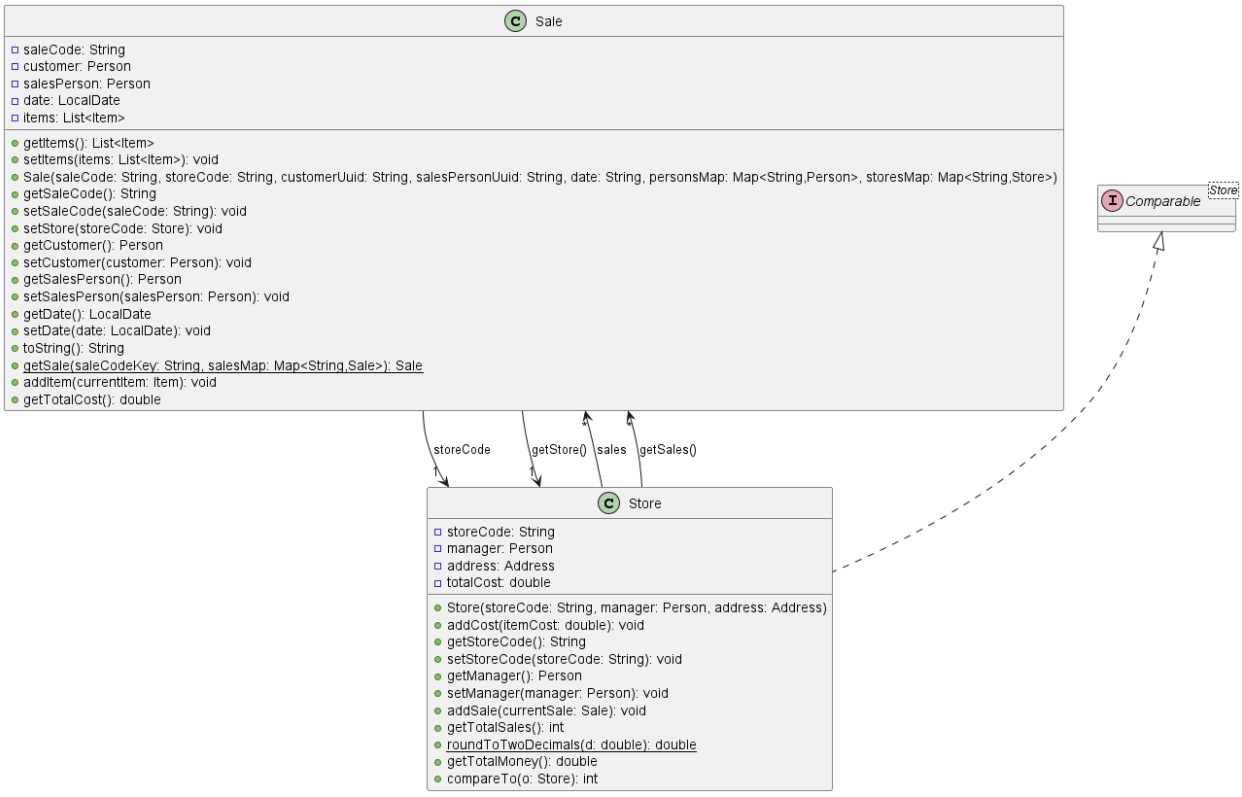

Figure 5: A section of the UML diagram of the system displaying the Store and Sale classes. These classes were designed to represent Stores and Sales as objects for the business model once loaded in from the database.

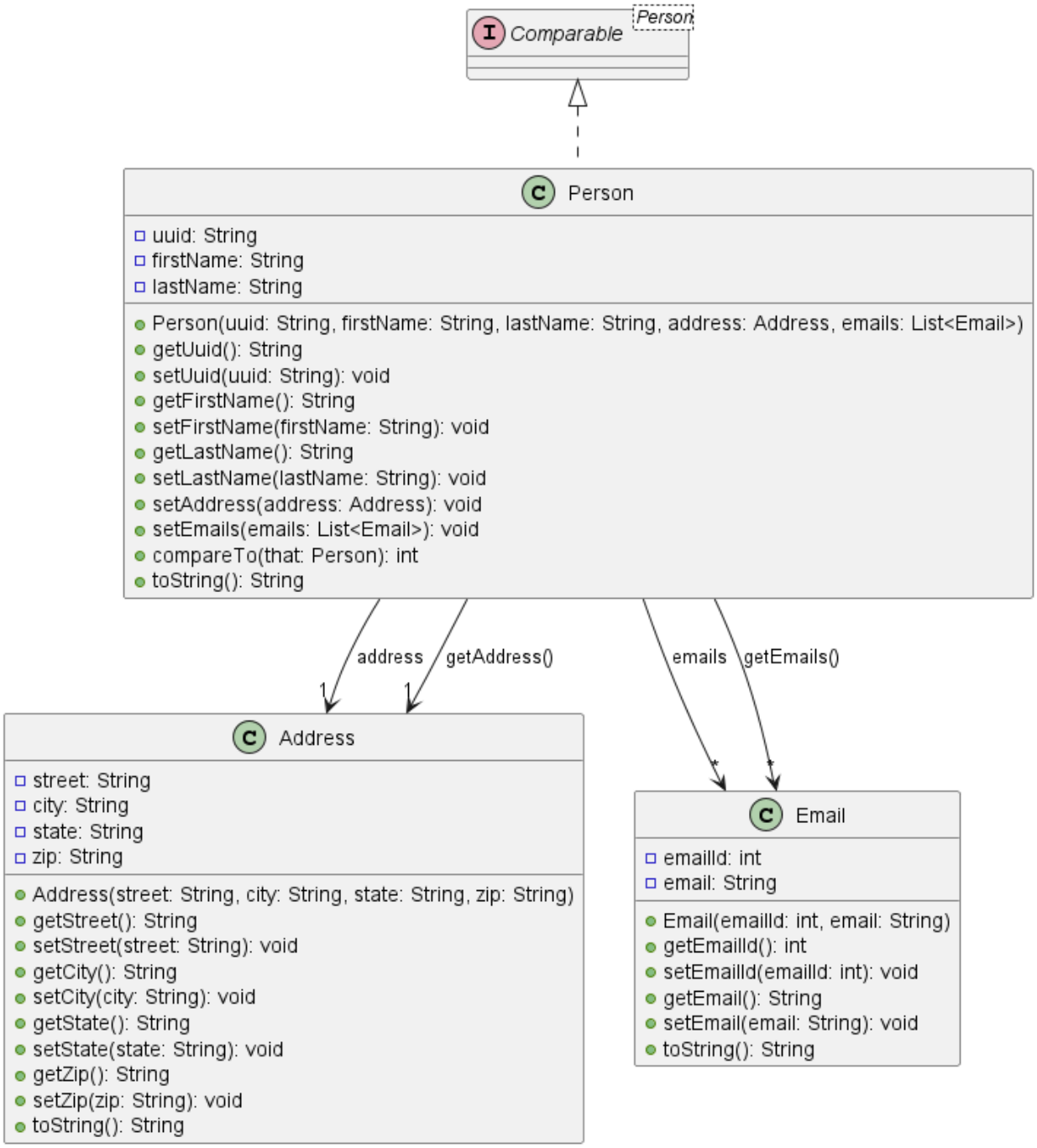

Figure 6: A section of the UML diagram of the system displaying the Person, Email, and Address classes. These classes were designed to represent Persons, Emails, and Addresses as objects for the system. The Email and Address classes were designed as helper classes for the Person and Store classes.

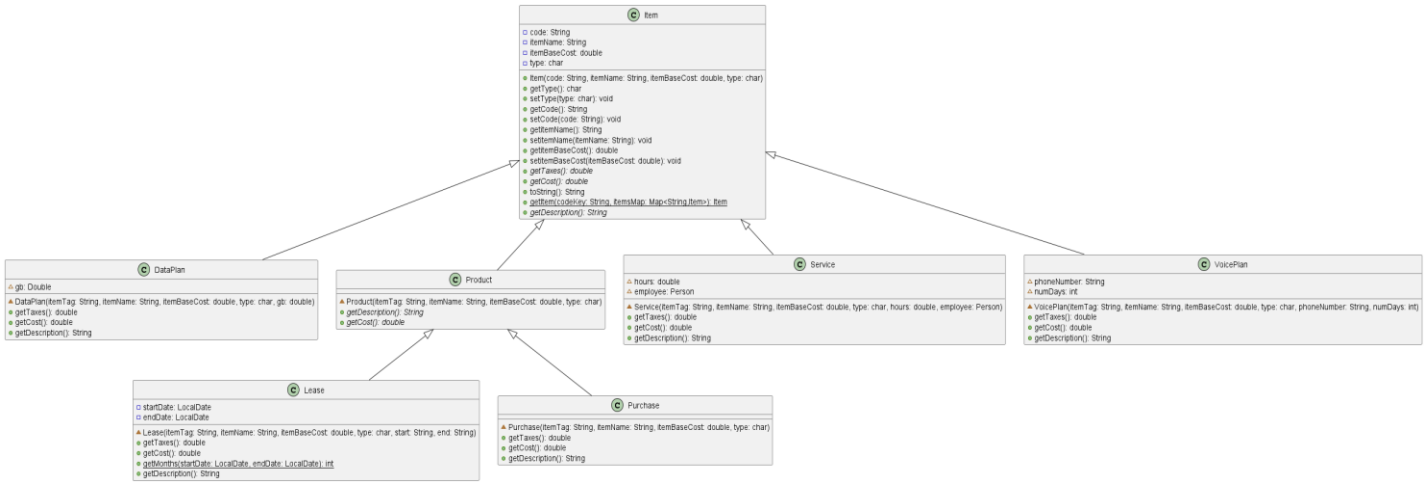

Figure 7: A section of the UML diagram of the system displaying the Item, DataPlan, VoicePlan, Product, Service, Purchase, and Lease classes. The Item class was designed with the purpose of becoming a parent class for the DataPlan, Service, VoicePlan, and Product classes. The Product class was also designed with the purpose of becoming an abstract class for the Purchase and Lease classes. All the Item subclasses were designed to represent their respective counterparts in YRLess' business model as objects in the system.

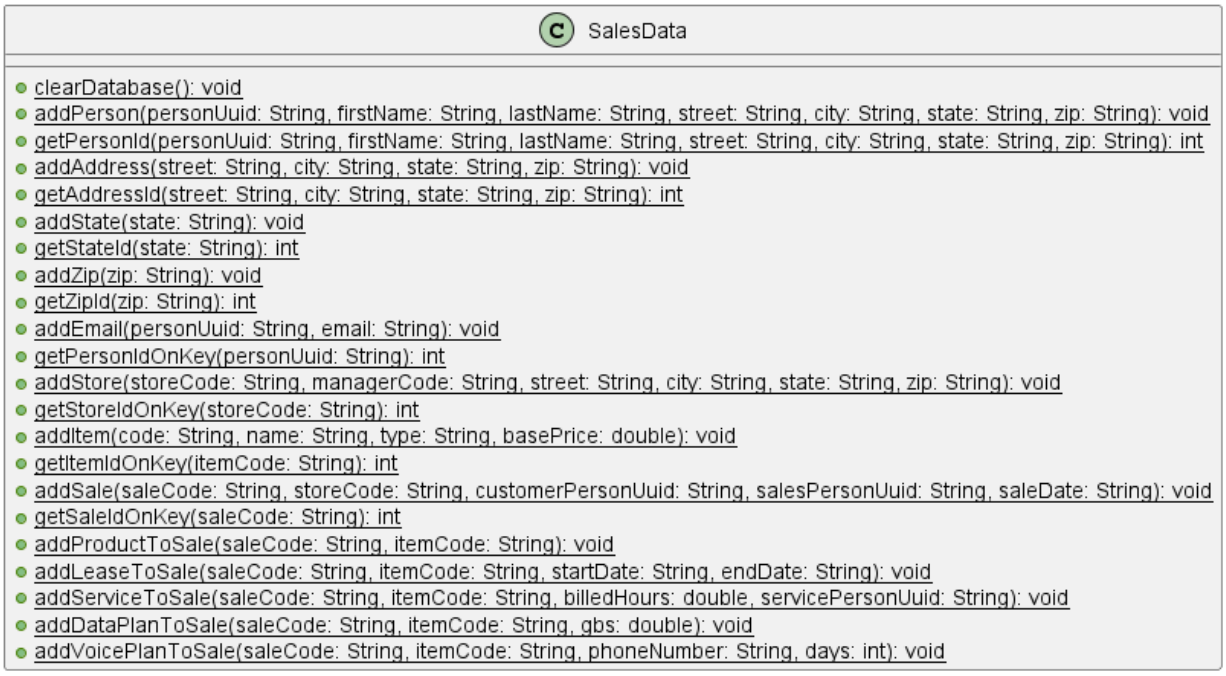

Figure 8: A section of the UML diagram of the system displaying the SalesData class. This class was designed with the purpose of persisting data to the Database using JDBC.

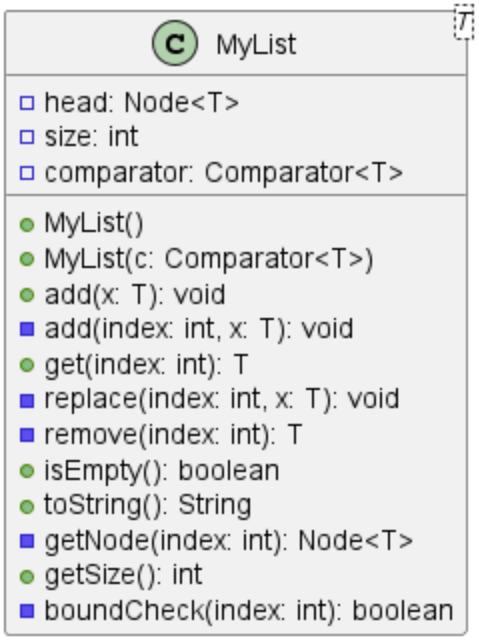

Figure 9: A section of the UML diagram of the system displaying the MyList class. This class was designed with the purpose of implementing a linked list ADT for the system.

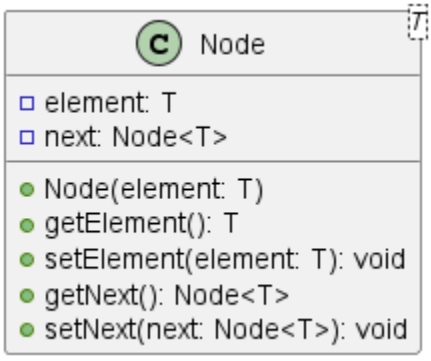

Figure 10: A section of the UML diagram of the system displaying the Node class. This class was designed with the purpose of representing nodes used in the custom list ADT for the system.

3.2.1 Component Testing Strategy

During the development of this component of the system, the test data created for testing the database was adjusted to test this component of the system. These test cases were adjusted to ensure that the class design could represent the different attributes of the different components of the business model. This data was loaded in through the form of flat CSV files then processed into a series of sale reports, which were then checked against the test cases mentioned above.

3.3 Database Interface

The database interface was created using JDBC. As JDBC essentially is a tool that allows for the execution of queries to the database, the data is loaded in through the DataLoader class using JDBC queries and stored in HashMaps that can interact with the different classes created to represent objects in the business model.

The database interface also persists data to the database using JDBC. The client provided a class, SalesData, which was implemented using JDBC to interact with the database. When inserting or deleting data from the database using this system, careful steps were taken to ensure that the queries JDBC was provided with properly handled tables based on their dependencies in order to maintain data integrity.

Additionally, since this interface allows a user to submit null data, steps were taken to ensure that it was handled based off of the degree of data submitted. For example, if a user submitted all null data, it is ignored. If a user submitted an Item that was missing data from some fields, the system ensures that no necessary fields are omitted and that it is in fact valid data.

3.3.1 Component Testing Strategy

The test cases developed and provided for the previous two components were reused for this component of the project. One key testing strategy related to this component was creating new test data with corner cases and seeing how the system handled these corner cases versus the way these cases are expected to be handled.

To ensure that the connection between JDBC and the database worked, two classes, ConnectionTest and DatabaseInfo, were made. Additionally, YRLess provided an external environment that contained test data that tested the connection between JDBC and the database and whether the queries we wrote using JDBC worked to insert/delete data in the database.

3.4 Design & Integration of a Sorted List Data Structure

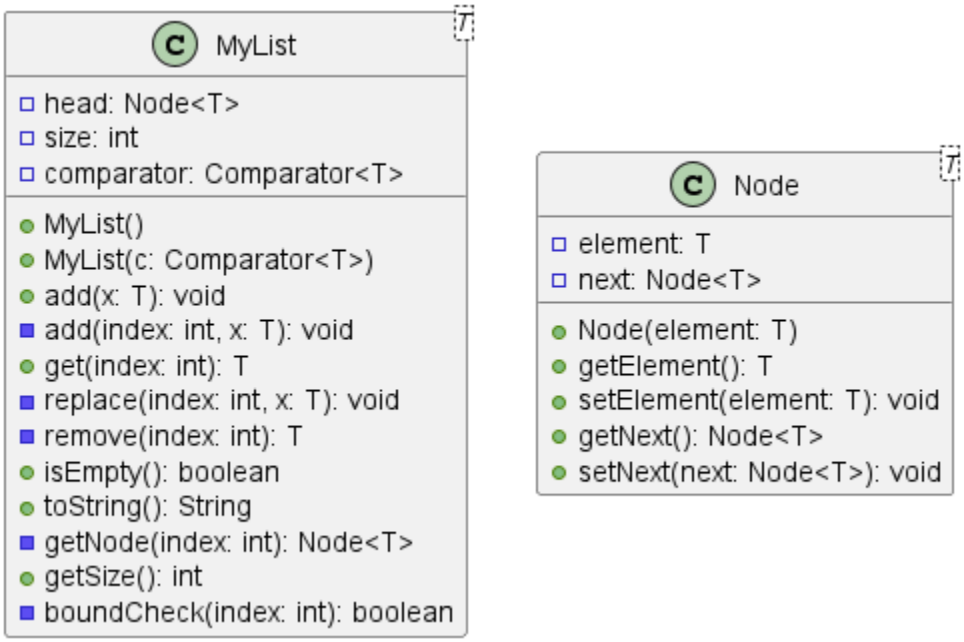

Figure 1: Section of the UML Diagram that shows the MyList and Node classes, which were designed to support a custom ordered linked list ADT implementation.

For this component of the system, it was decided that a linked list approach would be easiest to implement due to the fact that linked lists do not have indices, resulting in the development of the two classes shown in Figure 1 above.

As YRLess wanted this custom list implementation to be able to handle multiple, similarly formatted reports, these classes were made generic so that they could handle different types of data and be sorted in different ways using custom comparators.

3.4.1 Component Testing Strategy

The test cases developed in the previous components were reused for this component. Additionally, YRLess provided another external testing environment with new test cases to test the implementation of the sorted list ADTs.

Before using the external testing environment that was provided, the test cases developed for this project were loaded into the database, and the reports were displayed in the standard output. After the developed test cases passed, a JAR file was created and ran using

the external testing environment, where any necessary issues were addressed then fixed until the output matched the expected output.

4. Changes & Refactoring

Another team member was brought in during the middle of development and the Java class design was reworked from the ground up. Additionally, the way that the system displayed sales reports was fixed.

A prototype version of the system was developed to load and process data from flat CSV files and convert them into XML and JSON, but this was later changed to reading in data from the SQL database.

Near the end of the development of the phases of this project, the client requested that the sales system not only persist data to an SQL database, but also use an integrated sorted list that automatically maintained ordering of data according to different criteria.

5. Additional Material

Test data for the development of this system was developed in part with Mockaroo, and then manually edited into the proper expected format modeled by the test data already provided by the client through their external testing environment.

Ideas and concepts related to OOP were referenced from a free online textbook Computer Science II by Dr. Chris Bourke from the University of Nebraska Lincoln (See citation below).

Bibliography

[1] Bourke, C. (2021). *Computer Science II* (Version 0.3.0). Retrieved from https://bitbucket.org/chrisbourke/computerscienceii/src/master/ComputerScienceTwo.pdf