DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

CCC Simple Invoice System

Computer Science II Project

[Jin Seng Cheng & Xinyi Zhu]
4/17/2019
[Version 6.0]

[The descriptions and contents in this document describe the Object-Oriented software design for the new invoice system built for Cinco Computer Consultants.]

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	Jin Seng Cheng	2019/02/03
2.0	Updated diagrams, ER Diagram added, Database and Data Structure sections added, minor updates to the previous text.	Xinyi Zhu	2019/03/08
3.0	Altered the overall design description, updated diagrams, added data structure design section, minor revisions throughout.	Xinyi Zhu	2019/03/28
4.0	Updated design & integration of data structures	Jin Seng Cheng	2019/04/03
5.0	Minor changes to the previous parts and refactoring in other areas	Xinyi Zhu	2019/04/15
6.0	Corrected grammar and minor mistakes to the previous parts	Jin Seng Cheng	2019/04/17

Contents

1. Introduction

The project outlined in this document is the detailed software design description for the new invoice system designed for Cinco Computer Consultants (CCC). The new invoice system is an application written in the Object-Oriented Java programming language that is backed by a database to replace the aging AS400 green-screen system which is currently used by CCC. This system manages CCC's business model with the implementation of their business rules and provides functionality which includes equipment, consultations, and licenses.

1.1 Purpose of this Document

The purpose of this document is to illustrate the technical design of the new invoice system. It describes the implementation strategies of an invoice system including the testing methodology and implementation strategies. This document also outlines the MySQL database and its relationship to the newly developed Java application.

1.2 Scope of the Project

The new invoice system developed for Cinco Computer Consultants (CCC) is written in Java to replace the aging AS400 green-screen system. It is designed to manage portfolios of several products for their customers, namely equipment, consultations, and licenses. Besides, this new invoice system is also designed to assimilate CCC's business model by implementing its business rules.

The data used in the application is backed by MySQL database which enables the data to be stored, retrieved, and updated by the program. The application is also designed to generate a Summary Report and Individual Invoice Detail Reports through the generation of both XML and JSON data storage which are platform independent object declaration languages. XStream is used to generate the data in XML format while Gson is used to generate the data in JSON format.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Object-Oriented Programming – Computer programming in which not only the data type of a data structure is defined, but including the types of operations that is applicable to the data structure

Class – Templates of programming entities that are used to create objects and to define object data types and methods

Encapsulation – Concealing data fields within a class to prevent interference in these data fields by the outside world and methods in the class to manipulate these fields

Equipment – Various computer and electronic products sold by CCC to its clients as a reseller

Consultations – Services such as training and system evaluations

Licenses – Products such as software, server hosting, or third-party services

Inheritance – A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).

Polymorphism – A principle in biology in which an organism or species can have many different forms or stages.

Abstraction – A way to segregate implementation from an interface

1.3.2 Abbreviations & Acronyms

API – Application Programming Interface

ER Diagram – Entity Relationship Diagram

JDBC – Java Database Connectivity

OOP – Object Oriented Programming

SQL – Structured Query Language

UML Diagram – Unified Modeling Language Diagram

XML – Extensible Markup Language

JSON – JavaScript Object Notation

ADT – Abstract Data Type

CRUD – Create, Retrieve, Update, Destroy

Gson – Gson is a Java library that can be used to convert Java Objects into their JSON representation.

XStream – XStream is a simple library to serialize objects to XML and back again.

Apache Log4j 2 – Apache Log4j 2 is an upgrade to Log4j that provides significant improvements over its predecessor, Log4j 1.x, and provides many of the improvements available in Logback while fixing some inherent problems in Logback's architecture.

2. Overall Design Description

This application is written with Java programming language which is object-oriented. It utilizes four of the primary concepts of object-oriented programming in Java: encapsulation, abstraction, inheritance, and polymorphism. This system contains different aspects which can be considered as real-world objects. Hence, these aspects are separated into class entities. Classes created in this program are Consultations, Equipment, Licenses, Address, Customer, Person and ProductData. File input methods to read in the data from the old system are created separately in classes, such as readCustomerFile, readPersonFile, and readProductDataFile. The output of this file is generated in either XML or JSON

format. Hence, methods to convert the file input into XML and JSON format are also created separately in classes. The driver class of the application is InvoiceReport. The data fields and methods created in each class are specifically applicable to the functions designated in that class.

The CustomerData class is an abstract class and from it, two subclasses are created, which are Government and Corporate respectively. In addition, the ProductData also has three subclasses. These subclasses are Equipment, License, and Consultations.

These subclasses are developed in order to make each of the subclasses to be responsible for their own data and possess their own unique properties, such as calculations for their invoice information.

For data storage and loading, the program uses a MySQL database. Each object or class in the application is represented in the database by a table with columns for the object's specific fields with their foreign key and primary key. The connection between the MySQL database and Java program is done through the JDBC API.

Any errors that happen within the database connection will inform users by using the Apache java library called Log4j.

2.1 Alternative Design Options

Alternative design options considered in the development of this application:

- Rather than having several subclasses of Equipment, Licenses, and Consultations of ProductData and having Government and Corporate subclasses of CustomerData, those classes can be written in a God class. This would lead to some code redundancy and each type of class is unable to possess their own properties.
- Rather than interacting with MySQL database, it is also possible to load data from flat CSV file. This will cause the application to be much more inefficient because it is hard for users to add or update their data.

3. Detailed Component Description

Class entities are used in this application to represent the hierarchy of each class in the Cinco Computer Consultants (CCC). Each class possess its own constructor and contains a distinct data type from MySQL database system. All classes practice good Encapsulation to prevent these data to be accessed by public.

3.1 Database Design

The database schema of the application is shown in the ER Diagram presented in **Figure 1**. This data integrity is maintained by creating a constraint key that conceptually should not have duplicates data in invoices and product tables. This helps users prevent duplicate data in their database.

Each of the major data types including Invoice, Product, Customer, and Person are stored in the tables created. Person, Customer, and Invoice are joined to ProductOrder through the foreign key for each invoice associated with those tables. Specialized tables such as Address, email address, and Product, are

used to store the specific information for each major table in order to further normalize the database schema. Details of the database schema is shown below in **Figure 1.**

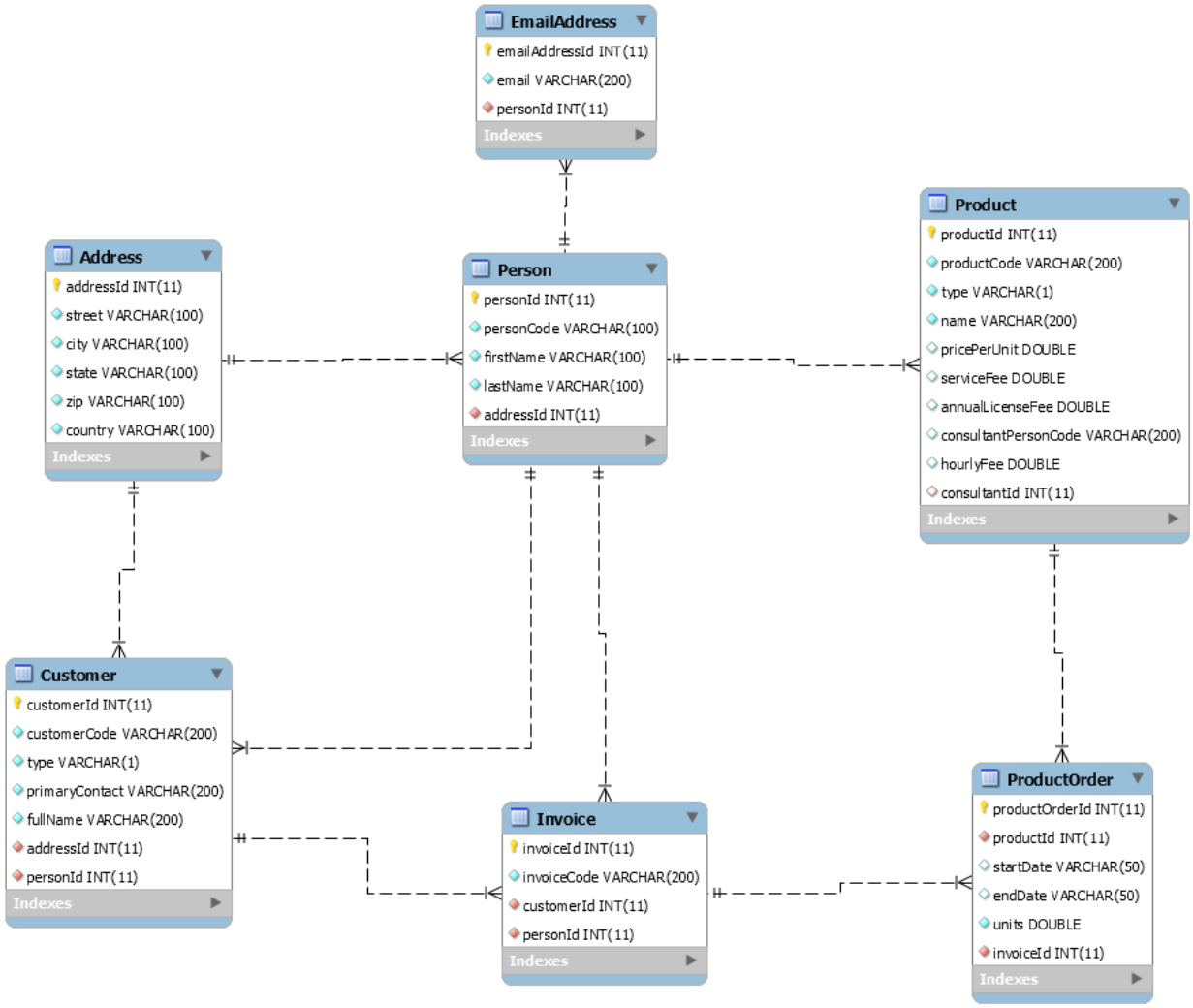

Figure 1: ER Diagram

3.1.1 Component Testing Strategy

The schema model of the database is built upon the previous Phase I/II. In order to test for the database model displayed in **Figure 1**, a series of standard and corner case queries are compiled to interact with the database. During Phase III of this project, implementation, and testing of the database such as retrieving all major fields for every person, updating a specific field of a specific data entries, and removing data, are conducted via MySQL Workbench with subsequent queries.

3.2 Class/Entity Model

The UML diagram in **Figure 2** shows the relations between and the functionality of the classes in the API. For the further description of a class, interaction refers to the overall design description in Section 2.

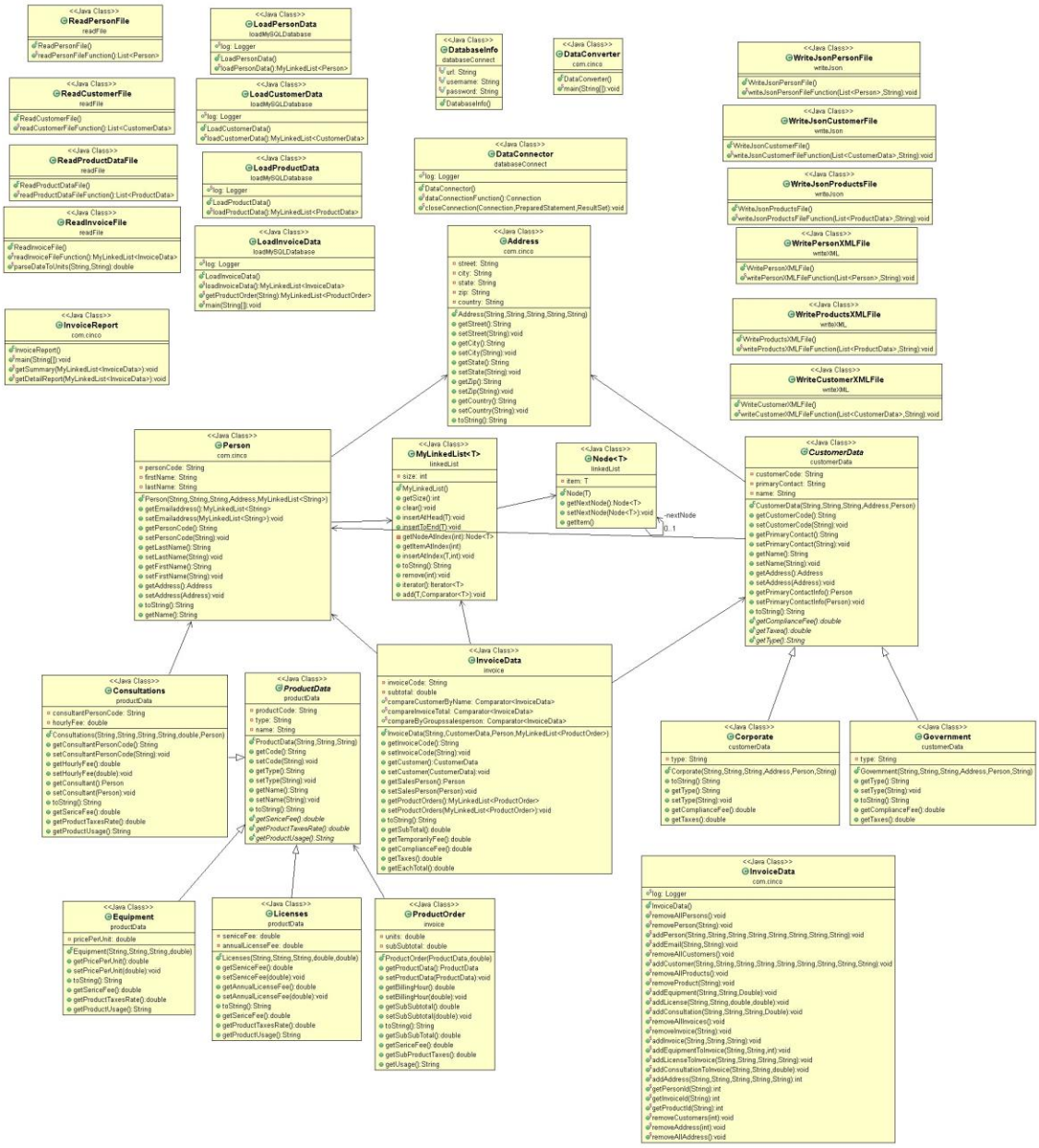

Figure 2: UML Diagram

The Invoice class uses the composition design principle which contains Person class (salesperson in each Invoice), Customer class (customer class) and ProductOrder class (the list of product customer bought).

Equipment, license, and Consultation are subclasses of abstract ProductData class. They have an "is -a" relation in this Diagram and all of them are the details for the ProductOrder class. These three classes inherit all the functionality of the ProductData class.

Government and Corporate are subclasses of abstract CustomerData. For instance, a Government object is a CustomerData object which inherits all its function and properties.

The DataConverter class is responsible for converting data from three plain .dat files into usable java instances of objects as well as generate two different types of data files—XML and JSON. Moreover, InvoiceReport is responsible for generating a summary and detailed report for all products in each invoice for the users.

3.2.1 Component Testing Strategy

Testing of the class relationship shown in **Figure 2** is conducted by using the test data in the data folder. In Phase I, the application reads through raw data files and encapsulate all data into specific classes in this application, and then generate XML and JSON formatted data files based on their input file. In Phase II, the application forms an Executive Summary Report and Individual Invoice Detail Reports for the users based on the relation in the file.

The testing process is conducted with well-formed data and intentionally broken data in all test data to test the error handling among these classes. There is also an expected output file in Phase II for a user to compare the output obtained in Phase II. With this, it shows that proper interactions between all of the classes and hierarchy systems in this application are taking place.

3.3 Database Interface

The interaction between the designed MySQL database with the Java application is achieved via the use of JDBC in Phase IV of the application development.

A driver class is created in the application to load the data from the designed MySQL database through JDBC. With this, the application can make a connection with the database and execute the CRUD principles, including creation, retrieval, updating, and deletion, of the data. The result-set stores the results from the retrieval queries to process them in the application. In the process of executing the queries, all SQL queries are sanitized so that the database and the application are protected against harmful SQL injections. The application uses prepared statements for this purpose. Besides this, the application is designed to handle null or improper data as well.

3.3.1 Component Testing Strategy

The integration of the database into the API is tested in the new driver class. Instead of loading data from the flat file, data is loaded from the database. In the new loadData class, this Java application connects to the database in order to retrieve all data for Person, CustomerData, ProductData, and Invoice. By using this data from the database, the application will generate a new invoice report. The new invoice report allows users to verify the correctness and proper functionality for this Database design.

3.4 Design & Integration of Data Structures

In the last phase in the development of the Java application, a sorted ADT list was implemented as object storage to hold an arbitrary number of objects through parameterized polymorphism - specifically invoice objects in this project which helps to preserve ordering of each invoice based on a

standard. The ADT developed in this application is the implementation of a linked list. The linked list implemented is designed in a way that can hold up to three different orderings of data, such as ordered in an alphabetical manner by last name / first name, a descending order from highest to lowest for the invoice total, or an order that groups all invoices first by the type of customer, then order by the last name or first name of the salesperson on the invoice. Comparators and generators are implemented in the linked list to support these orderings. ADT methods are implemented to facilitate adding, removing, retrieving, and iterating over the elements. However, the order of the ADT data is achieved by insertion rather than through update and impose by calling methods and is always maintained.

The list ADT implemented is a linked list. This implementation only requires the list to keep track of the head node and the pointer to the next node, and the list is automatically resized. The ordering of data in a linked list which is done by insertion as maintaining an order cost less than sorting an unsorted collection. The insertion method initially checks the position, which is also called a node, of the object to be inserted based on the specified comparator. Insertion of an object is done by changing the nodes points. For instance, if an object z is inserted between two nodes which are node a and node b respectively, z will point to node b while node a is linked to z to complete the insertion. The deletion method removes an object in a linked list. It is done by linking the previous object to the object which is placed behind the object that needs to be removed. The ADT traverses when an object is being searched in a linked list, as it always starts searching from the head and access each node until the desired object is successfully found. This happens because of the implementation of an iterable interface in ADT which allows the utilization of enhanced for-loop.

The linked list is utilized to store the objects from all the invoices. Additional fitting methods are implemented in order to create three different linked list with a different order. The printing method is implemented to generate a summary report for the different orderings of data.

3.4.1 Component Testing Strategy

Component testing strategy of the implemented ADT followed the testing strategies from Phase IV. In the previous Phase, all the instances of each Class loading from the database are stored into an array-based data structure, but all the data is stored the linked list ADT. Verifying the data structure's stead stability and correctness involves comparing the standard output and expected output in the data folder. Moreover, standard cases and corner cases were designed to test the stability and catch any illegal behavior. For example, delete the node when there is only one node in the linked list.

3.5 Changes & Refactoring

Phase I – Initial design the class and relation with no changes made.

Phase II – Redesigned CustomerData class and made two subclasses for it and added a specific abstract method for the data calculation.

Phase III – Delete the "Compliance Fee" column in Individual Invoice Detail Reports to reduce the confusion for users, put all constructors at the top of classes and reduce the spacing between the item name and the description of the transaction related to the item in Individual Invoice Detail Reports.

Phase IV – Rename the “State” to “state” in my SQL database, implement the parseDateToUnits function in ReadInvoiceFile class. Modify the parseDateToUnits so that function can take two Strings and directly give the units of date. At the end of each Individual Invoice Detail Report, put the “Compliance Fee”, “Fee” to make the report more understandable.

Phase V – Modify the DataConnector is responsible to make a connection and close connection from the database. Remove the listToHashMap function.

4. Bibliography

[1] “Java Platform, Standard Edition Tools Reference.” Javadoc, 8 Jan. 2016, docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html.

[2] Rivières, Jim des. “Evolving Java-Based APIs.” Evolving Java-Based APIs - Eclipsepedia, 2014, wiki.eclipse.org/Evolving_Java-based_APIs.

[3] MySQL Workbench Documentation “Chapter 1 General Information.” MySQL, 2018, dev.mysql.com/doc/workbench/en/wb-intro.html.

[4] S.Adamchik, Victor. “Linked Lists.” LinkedLists, 2009, www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html.

[5] Trail: Learning the Java Language.” Trail: Learning the Java Language (The Java™ Tutorials), 2017, docs.oracle.com/javase/tutorial/java/.

[6] “Trail: Learning the Java Language.” Trail: Learning the Java Language (The Java™ Tutorials), 2017, docs.oracle.com/javase/tutorial/java/.