DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

Cinco Computer Consultants Invoice System

CSCE 156H – Computer Science II Project

AUTHORS REDACTED

4/10/2014
Version 5.0

This document describes the design and aspects of a project created using an object oriented mindset. The invoice system described by this document was created using Java.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	AUTHORS	2014/02/04
2.0	Intro Finished, Title Page Completed, Added to Definitions and Abbreviations	AUTHORS	2014/02/18
3.0	Added UML Diagram, Finished Overall Design Description, Added to Definitions and Abbreviations, Added Bibliography	AUTHORS	2014/03/01
4.0	Added ER Diagram, Updated UML Diagram, Finished Detailed Component Description, Modified Alternative Design Options, Added to Definitions and Abbreviations, Updated Bibliography	AUTHORS	2014/03/24
5.0	Updated UML and ER Diagrams, Finished final sections, Revised and modified all previous sections	AUTHORS	2014/04/09

Contents

1. Introduction

This project is a replacement of the original invoice system of Cinco Computer Consultants (CCC). This invoice system is a Java application with an object oriented programming style. The invoice system manages the various services and products that CCC has to offer, while conforming to the standards of CCC.

In order to differentiate between the various products CCC offers, the system uses a unique alphanumeric product code and name for each product. The invoice system also calculates taxes correctly based on the person buying a service or product.

1.1 Purpose of this Document

The purpose of this document is to outline the various different components and applications of this invoice system. This document also explains all the connections between different elements within the project.

1.2 Scope of the Project

This invoice system will be used to organize, simplify, store, and process CCC's products data. This system will be replacing their AS400 green-screen system. The project is object oriented, written in Java, and supports the needs of CCC. These needs include separating the different products of CCC into their respective categories. Also this project will determine the different tax rates to apply to different types of products and customers.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Class – Programming entity that represents a real world entity, that contains state and behavior.

Constructor – Creates an instance of the class when called in the program.

Object Oriented Programming – Programming language model organized around objects rather than actions, and data rather than logic.

XStream – Collection of libraries used to generate XML files.

Gson – Collection of Libraries used to generate JSON files.

Log4J – Collection of Libraries used to generate a log file of errors.

Node – A place in a linked list that holds an object.

1.3.2 Abbreviations & Acronyms

CCC – Cinco Computer Consultants

API – Application Programming Interface

XML	Extensible Markup Language
JDBC	Java Database Connectivity
OOP	Object Oriented Programming
SQL	Structured Query Language
JSON	JavaScript Object Notation
ER Diagram	Entity Relationship Model
CRUD	Create, Retrieve, Update, Destroy
ADT	Abstract Data Type

2. Overall Design Description

This project is based on the four main principles behind object oriented programing: Inheritance, encapsulation, abstraction, and polymorphism. The four different aspects of OOP are used throughout the classes and project. There are several different classes that are used to store information about people, customers, invoices, and products. These classes also have many methods that allow interaction with the data. Some of these class names are Product, Invoice, Person, and Customer. The main class that drives the program is called DataConverter. DataConverter allows interaction between the database, and the java class that connects to the database.

The API that is used for the program is connected to the MySQL database through a class called InvoiceData. The relationship between different objects in the program are represented through tables with foreign key relationships. These tables also have different columns to represent the different elements in each class. The MySQL database connects to the classes through the JDBC API.

Instead of using the standard output this program the errors are logged in a log file through log4j. This enables easy debugging if errors occur.

2.1 Alternative Design Options

One alternative design option for this project was to put the specifics of what each customer ordered for each product inside of the respective product class. For example, instead of putting units in the InvoiceEquipment class it would be put in the Equipment class. This design option may make the overall layout of the project slightly simpler, but technically an equipment is only composed of a name, code, and price per unit. The actual number of units ordered is dependent on the invoice instead of the product itself.

Another design alternative would be to deal with the taxes in the Person or Customer class instead of in the Invoice class. This seems to make sense, but the customer is part of the invoice. This means that the taxes will be applied to the invoice as a whole instead of to the individual customers. Also some customers may not have an invoice attached to them, and thus should not have access to a method that would get the taxes to be applied to an invoice.

3. Detailed Component Description

The program uses the different classes to represent the different aspects of the invoice system. For example the License, Equipment, and Consultation classes all extend the product class. This is a prime example of inheritance. These three classes all have constructors, and each class has its own data that specifically pertains to the specific class, while still implementing all the functions of Product.

3.1 Database Design

MySQL was used to create this database. Uniqueness constraints have been applied where necessary, such as between ProductID and InvoiceID. This maintains data integrity by not allowing any duplicate data in fields which should not have duplicates. Each major element has its own table full of different columns of data. The Product and Invoice tables also have a join table called ProductInvoice, due to the many to many relationship between them. ProductIncovice also stores important data that pertains to the products ordered in each invoice. This allows there to be no duplicate entries for the amount of units for a certain product. Further normalization of the database is achieved through the use of a country table. A country tables prevents duplicate data entries in the country column of Address. Further details of the database can be found below in **Figure 1**.

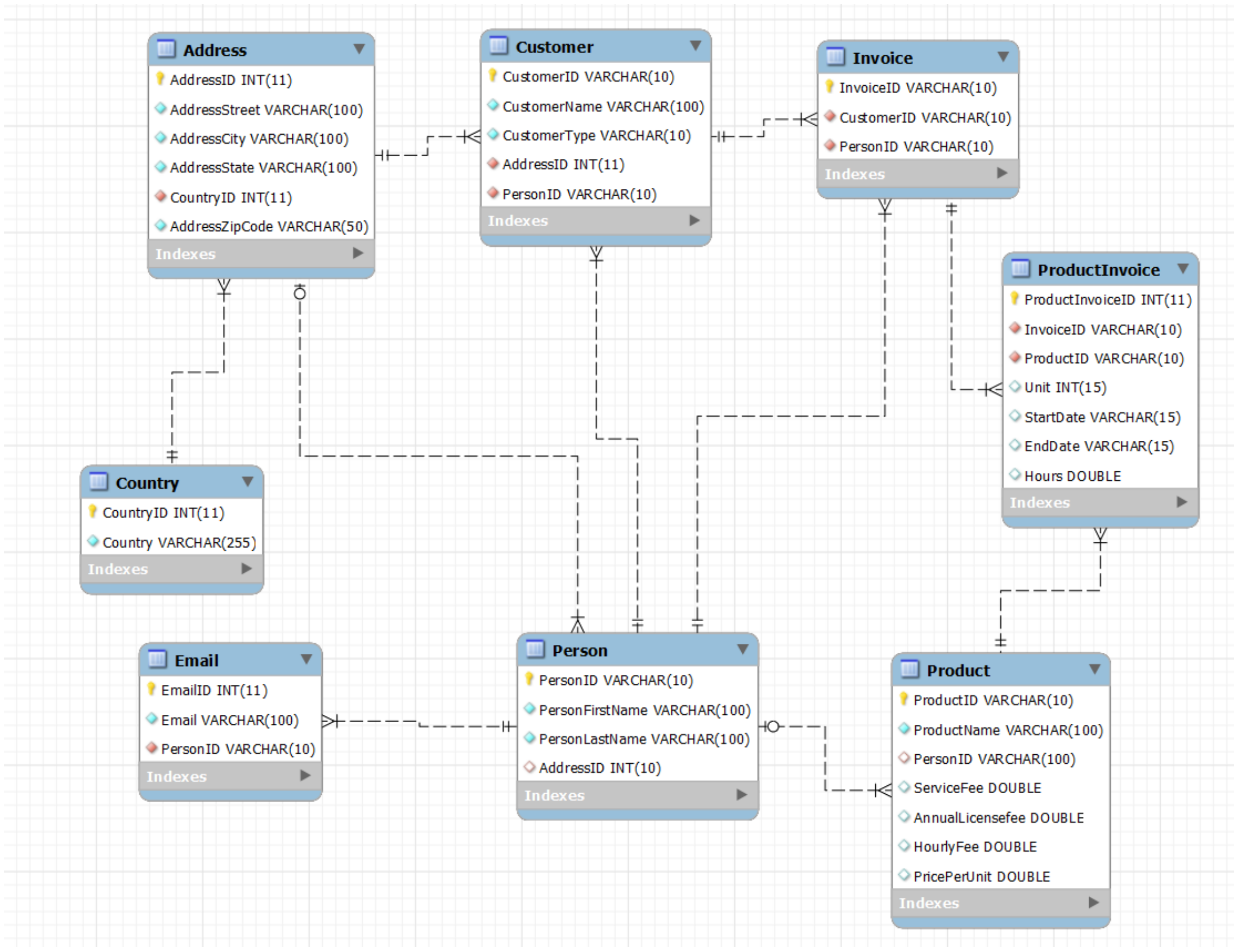

Figure 1: ER Diagram Representing Tables and Their Relationships

3.1.1 Component Testing Strategy

For testing the program, the same test data was used to test the database in MySQl and to test the program, and database in Java. Both the MySQL database, and the Java program that connects to it use delete, insert, update, and select queries to do basic CRUD. The select query is used mostly for testing the database to make sure that the table are made and formatted correctly, and to make sure that all the data is inserted, deleted, or updated correctly.

3.2 Class/Entity Model

Most of the classes stem from the Invoice class, whether it is directly, like the Product class, or indirectly like the Person and Customer classes. The Person and Customer class don't stem directly from the Invoice class, but the Invoice class has salespersonCode, and CustomerCode strings in it. The Invoice class also has get methods that retrieve information from the Person Class, along with the Customer class and all its affiliates.

Stemming from the Product class are its three "subclasses". The License class, Consultation Class, and Equipment class. These three classes are all products, therefore they are placed under the Product class.

Within the InvoiceData there are many methods that allow the program to connect to the database. All the work is actually done in the DataConverter class however. The DataConverter class is in charge of printing out the statements with the totals and invoices on them. It contains the main method that outputs all of the data. For more detailed information refer to **Figure 2.**

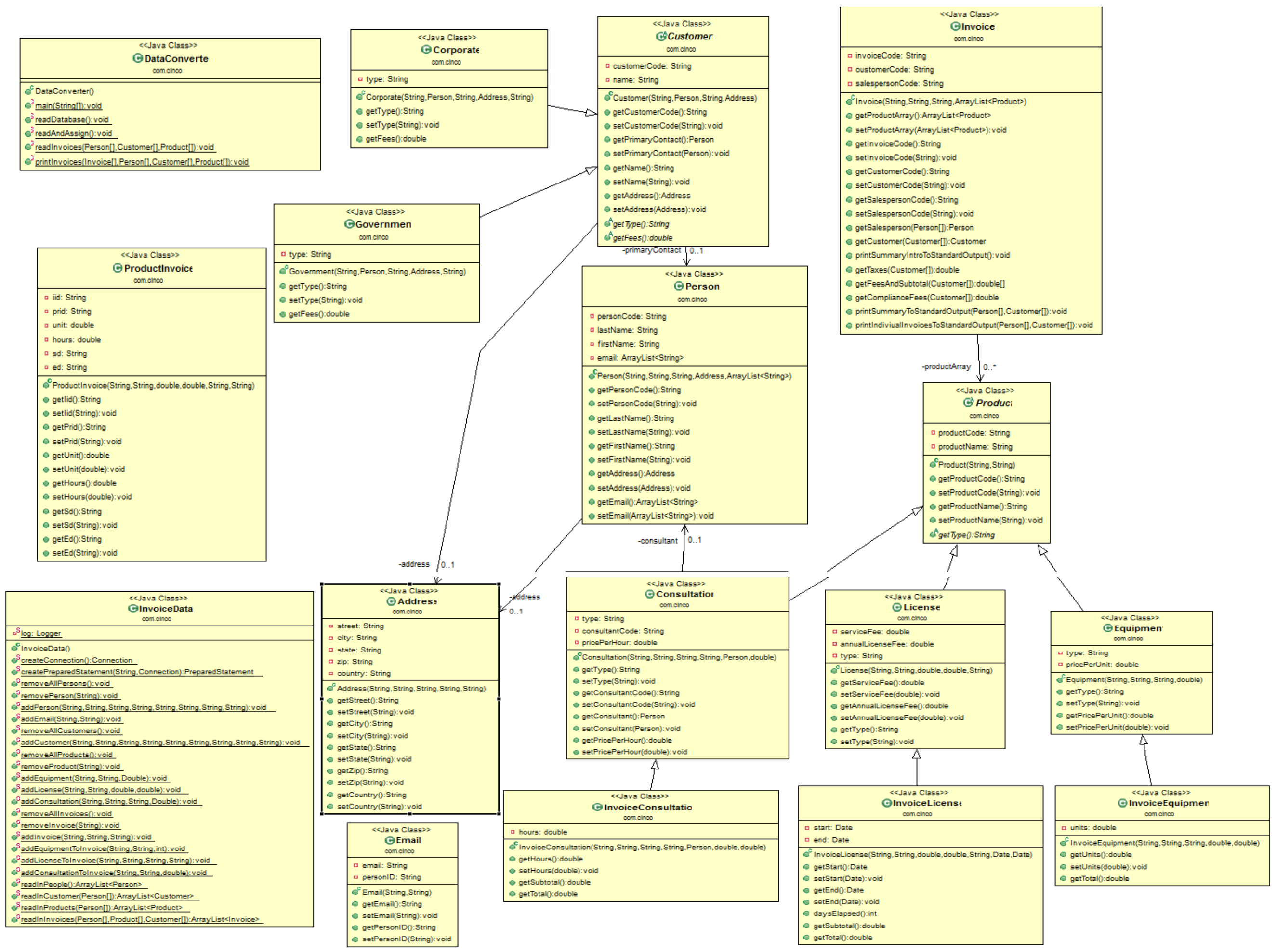

Figure 2: UML Diagram Classes and Their Relationships

3.2.1 Component Testing Strategy

The java code in this program is supposed to read in data and properly convert it to the appropriate java class. This is tested though the output of XML and JSON files. XML and JSON files contain code in XML and JSON respectively that accurately represent the data. Input data that accurately models the problem, and has many corner cases was used as input. This input was then read in through the use of the DataConverter class. The output XML and JSON files are then analyzed for correctness based on an expected correct XML and JSON output file.

This class also is responsible for printing a summary of all the invoices. The many print methods in the Invoice class are used to print the summary in a human readable format while also calculating total cost, fees, and taxes. This testing method was tested in a similar way to the XML and JSON files, because it is also compared to an expected correct output summary. This method of testing revealed many bugs in the way the taxes were handled, which allowed fixes to be quickly implemented before the final version release.

3.3 Database Interface

The Java API, JDBC, is implemented in order to connect to the MySQL database. In order to access the data, select statements must be written and executed. These select statements return a result set that can be created into Java objects as needed. The database is also able to be updated through the use of update queries. Removal is slightly harder due to the fact that many tables hold foreign key references to other tables. This requires removal to be done in a precise order, so that all data may be properly deleted.

Null values are checked through the database itself. The database has many fields that do not allow null values to be entered, but the database itself is able to take care of these exceptions. Also duplicate data is managed through the database through uniqueness constraints. These constraints prevent multiple non-unique entries.

3.3.1 Component Testing Strategy

To test data in the database basic CRUD is used. After the tables were created and the initial test data was inserted select statements were used to test the data to make sure the initial test data was inserted correctly. The test data that was inserted in this phase was the same test data that's has been used throughout the programing phases. Multiple insert, delete, select, and update queries were used in the phase to test the data, and to make sure that the tables that were used are correctly implemented. Select statements were always used after inserting, deleting, or updating tables to make sure the tables were still correct.

3.4 Design & Integration of Data Structures

The ADT for this program is implemented through the use of a sorted list. The list ADT can hold multiple different types of objects due to polymorphism, but these particular lists only hold Invoice objects. This list can be ordered in three different ways using a comparator. The first way orders the invoices based on the name of the invoice customer (Last Name/First Name). The second way to order the lists is based on the total cost of the invoice. Lastly the invoices can be ordered based on customer type, then on salesperson name (Last Name/First Name).

This ADT is a linked list. This means that only the head node needs to be kept track of. All following nodes are referenced in the previous node. This makes adding and modifying the list much easier, due to the need to only rearrange the references in each node. In order to determine the order of each element a comparator is used when adding to the list. This list also contains many methods for adding, removing, and clearing.

3.4.1 Component Testing Strategy

The ADT was tested by creating three different instances of the linked list. Adding elements to each one through the use of the comparators. The invoice summary is then printed out similarly to the testing in phase 2. After the new invoice summary is printed out, it is then compared to a proven test case to make sure that the linked list is working properly.

3.5 Changes & Refactoring

The main changes with the program came in the MySQL database section. Redesigning the database to make a join table between Product and Invoice called ProductInvoice. This join table allowed for a specific start date, end date, hours, and units to be unique to an individual product, instead of having them directly in Product. Also instead of making individual tables for each of the product's properties, the design choice was to put them all into the Product table.

4. Bibliography

[1] *Log4j 2 Guide – Apache Log4j 2.* (2014). Retrieved March 20, 2014, from Apache Logging Services:

http://logging.apache.org/log4j/2.x/index.html

[2] About XStream. (2014, January 25). Retrieved January 25, 2014, from XStream:

http://www.xstream.codehaus.org/

[3] About GSON. (2014, January 25). Retrieved January 25, 2014, from GSON:

https://code.google.com/p/google-gson/