Department of computer science & Engineering
UNIVERSITY OF NEBRASKA—LINCOLN

Movie Theater Invoice System

CSCE 156 – Computer Science II Project

[redacted]
08/11/2018
Version 1.4

The contents of this document describe the object-oriented software design for the new invoice system for Cineclark entertainment group.

Revision History

Version	Description of Change(s)	Author(s)	Date
1.0	Initial draft of this design document	[redacted]	2018/10/01
1.1	Add Database and update class design	[redacted]	2018/10/17
1.2	Add Database testing	[redacted]	2018/10/31
1.3	Update Class design and database design. Finishing	[redacted]	2018/11/07

Contents

1. Introduction

The project outlined in this document is a replacement of a movie theater invoice system, for Cineclark entertainment group. The invoice system is a Java application that uses object-oriented programming and is backed by a database. The API supports the company's business rules and creates the invoice for their product. The product includes movie tickets, season passes, parking passes and refreshments.

1.1 Purpose of this Document

The purpose of this document is to outline all aspects and implementation of the invoices system API. It provides all the components and elements of the design and explains how this system works. It also provides information about the data structure and the test case. This document provides the outline of the necessary elements for this specific API.

1.2 Scope of the Project

The Java-based invoice system developed for Cineclark entertainment group is designed to replace their old legacy invoice system. Its purpose is to generate invoices of their products including movie tickets, season passes, parking passes and refreshments for customers. The invoice includes an executive summary report and individual invoice report detail which includes the details of the personal information and the purchase. The data of this program is stored in a MySQL database; this database can be used to store and update data of the Cineclark entertainment group. There are also some methods to convert the old DAT file into XML and JSON.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Class: A blueprint or prototype that defines the methods and variables.

Constructor: A special method used to create an instance of a class.

Encapsulation: Use to wrap all the data and code into a single file.

Node: a place to hold objects in linked-list.

Object-oriented programming: A programming model for organized objects that represents real-world objects to solve problems.

1.3.2 Abbreviations & Acronyms

ADT – Abstract Data Type

API – Application programming interface

JDBC – Java Database Connectivity

OOP – Object-oriented programming

SQL – Structured Query Language

EDI – Electronic Data Interchange

XML – Extensible Markup Language

JSON – JavaScript Object Notation

2. Overall Design Description

This application is base on the object-oriented paradigm which is described as encapsulation, abstraction, inheritance, and polymorphism. Because this system is a model, this application's use of unique classes is essential. Relevant data, methods, and functionality are built into respective classes based on designing proper encapsulation enforcement. The current primary classes include InvoiceData, InvoiceReport.

The product class is an abstract class. From it, four kinds of product sub-classes are created. These sub-classes include MovieTicket, ParkingPass, Refreshment, SeasonPass. InvoiceData includes methods to control the database. DatabaseInfo is used to store database connection information and contains methods to create the connection and execute queries easily. The overall goal of the development of these classes is to promote reusability and reduction of redundant methods. This goal is reached via the implementation of the classes and sub-classes outlined above.

This application uses ADT to manage the order of the invoice that is generated. The data is stored in a MySQL database, and the connection between Java and MySQL uses the JDBC API.

2.1 Alternative Design Options

Other design options considered for development of the project include the following:

- Omitting the Address class and folding that functionality directly into the Person and customer class.
- Putting the print method directly in the driver class and directly calculating the tax in the print method.
- Separating the product type to store the product into the MySQL database.
- Using less, but more descriptive, foreign keys to connect the whole project in the database.

3. Detailed Component Description

Classes are used to represent instances of given objects. Object creation is handled by constructor methods in the respective classes via the provided data. The provided data values are encapsulated and belong to the class they were used to create. By default, all member variables of classes are set to private so that variable interfacing is handled by getter and setter methods contained with the parent classes.

3.1 Database Design

The Database is made by MySQL; The ER Diagram presented in Figure 1 represents the application's database schema. This diagram not only shows the relationship between other tables in the schema but also the primary tables with their respective fields. The database schema is separate by the Java class design. The database has carefully designed to simplify queries.

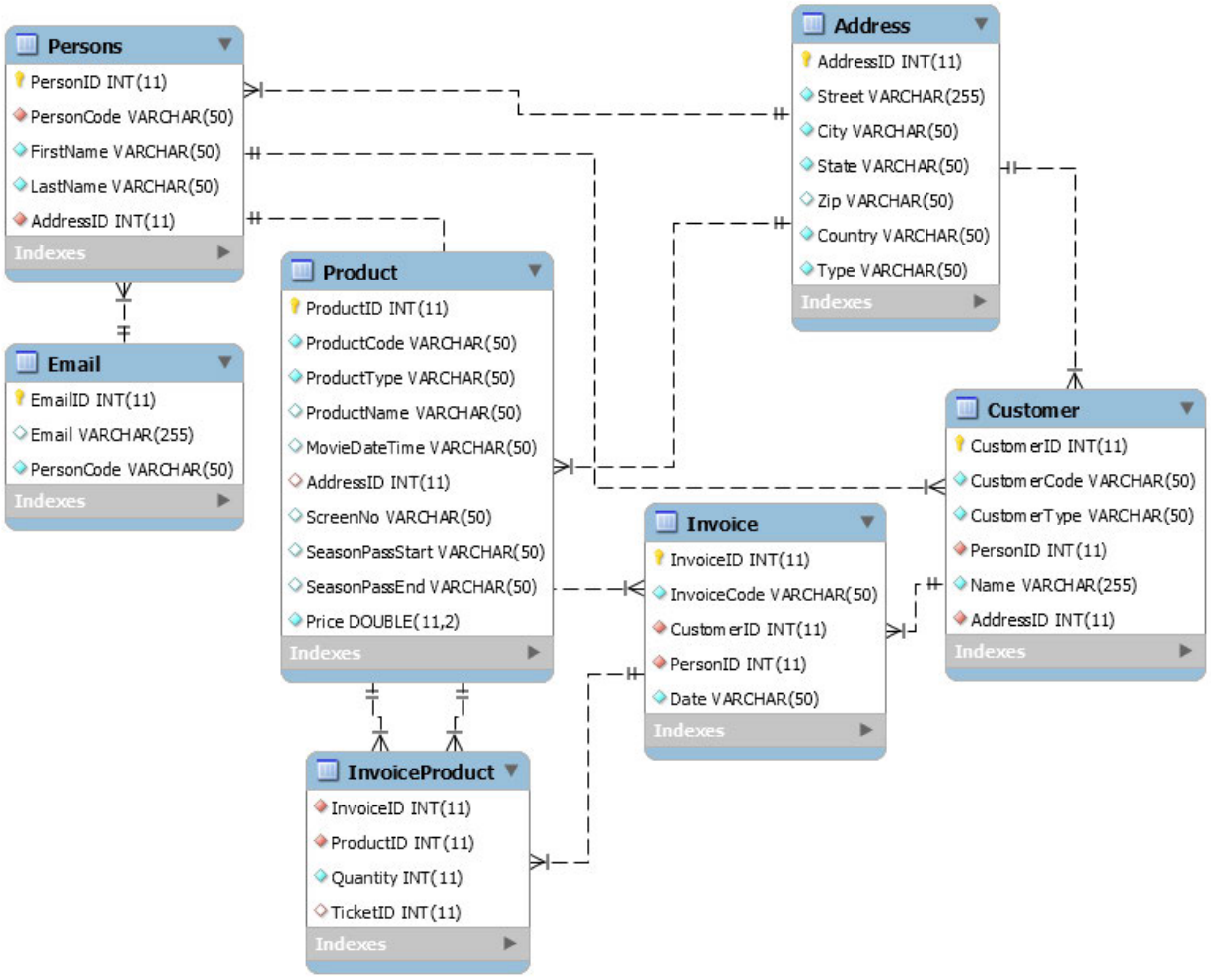

Figure 1: ER Diagram

3.1.1 Component Testing Strategy

To test the database displayed in the ER Diagram in Figure 1, some test queries were created and tested on the database schema. The test queries included: queries to find certain information using or without using a foreign key; queries to add or remove designated data with or without a foreign key; queries to update data for designated data with or without a foreign key. All the remote and testing are performed on MySQL Workbench 8.0 CE.

3.2 Class/Entity Model

The UML Diagram presented in Figure 2 represents the application's classes and their functionality. It has been completed and tested.

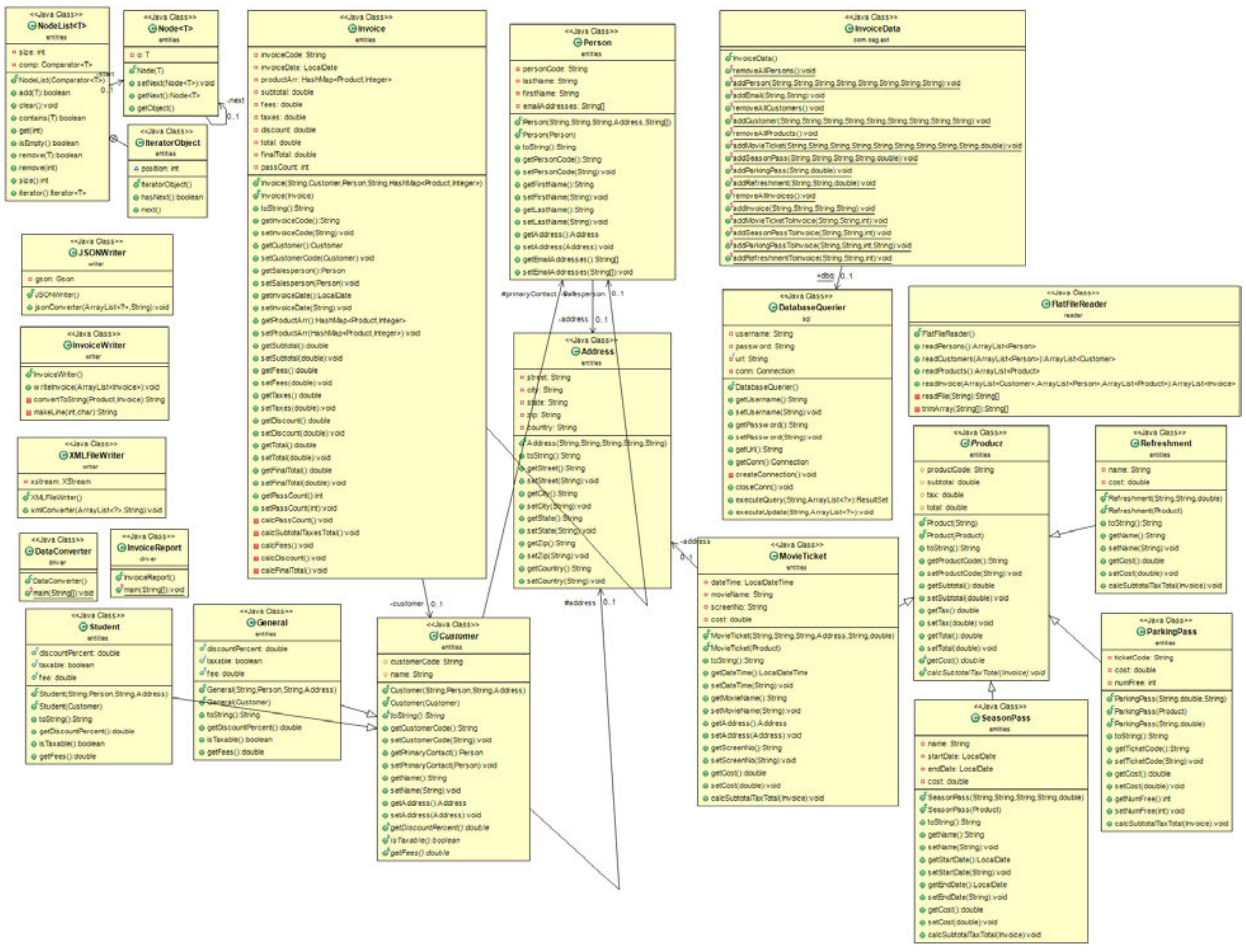

Figure 2: ER Diagram

3.2.1 Component Testing Strategy

Testing of the classes displayed in the UML Diagram from Figure 2 is conducted by testing file input processing. For the Phase I, the application must read in raw, loosely formatted data from two text files, process the data via the provided classes, and output to two XML documents.

Testing of these procedures is conducted by building well formatted and poorly formatted data files for data processing and comparing the results to the expected output. This shows that proper interactions between input, output, and classes are being conducted.

3.3 Database Interface

JDBC is used to read data from an external database and allows for the insertion of new data if necessary. This is achieved through the classes, DatabaseQuerier and InvoiceData, and the MySQL Connector library

DatabaseQuerier interacts directly with the MySQL database and contains the methods for passing queries and updates to the database in return for result sets. All queries and updates are passed using prepared statements to prevent SQL-injection attacks. InvoiceData uses the DatabaseQuerier to execute many core methods that make insertion and removal of data from the MySQL Database from inside of the java application simpler.

3.3.1 Component Testing Strategy

Testing of these methods was done by creating many test cases and checking (from MySQL Workbench) if those values corresponded to the input in the java application. This testing phase helped round out the database design as well, as the database and the java application must work in tandem to produce the desired output.

3.4 Design & Integration of Data Structures

The make future implementation of invoices easier, the application contains a self sorting ADT which is able to hold Invoices, or any other object that might be implemented in the future. The ADT consists of the NodeList and Node classes, which are abstracted to accept any type of object. NodeList is an Iterable type so it may be used in an enhanced for loop

There is currently only one Comparator type object that aides in the sorting of Invoices by total dollars spent, and many other Comparator type objects can be easily implemented whether it be for Invoices or some other object.

3.4.1 Alternative Design Options

While the ADT implements many methods that the List interface details, it does not use most of the methods detailed in the List interface. To be more in line with future ADTs, these methods should be implemented

3.4.2 Component Testing Strategy

To test the ADT, Invoices and other objects such as Persons and Customers were added to the NodeList and given temporary Comparators to go with them. Then, all of the methods and edge cases were tested and the program was modified to satisfy those edge cases.

3.5 Changes & Refactoring

Phase I - Initial design. No changes made.

Phase II - Redesign class and add methods to generate invoice report.

Phase III - Design database.

Phase IV – Design database connection and sorted list ADT.

4. Bibliography

Eclipse Platform API Specification. (2013). Retrieved November 2018, from Eclipse.org: https://help.eclipse.org/indigo/nftopic/org.eclipse.platform.doc.isv/reference/api/index.html

MySQL Workbench Documentation. (2018, November 07). Retrieved November 2018, from MySQL: https://dev.mysql.com/doc/workbench/en/index.html

Oracle. (2018). Java Platform, Standard Edition 8 API Specification. Retrieved 2018, from Oracle: https://docs.oracle.com/javase/8/docs/api/