Cinco Computer Consultants Invoice Management System

CSCE 156 – Computer Science II Project

AUTHOR
4/9/2014
Version 1.0
**Revision History**

<table>
<thead>
<tr>
<th>Version</th>
<th>Description of Change(s)</th>
<th>Author(s)</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial draft of this design document</td>
<td>AUTHOR</td>
<td>2014/04/08</td>
</tr>
</tbody>
</table>

The contents and descriptions of this document detail the design and implementation of a new invoice management system for Cinco Computer Consultants.
Contents

Revision History ................................................................................................................................................. 1

1. Introduction .................................................................................................................................................. 4
   1.1 Purpose of this Document .................................................................................................................. 4
   1.2 Scope of the Project ............................................................................................................................. 4
   1.3 Definitions, Acronyms, Abbreviations ............................................................................................... 4
       1.3.1 Definitions .................................................................................................................................... 4
       1.3.2 Abbreviations & Acronyms ....................................................................................................... 4

2. Overall Design Description ......................................................................................................................... 5
   2.1 Alternative Design Options .................................................................................................................. 5

3. Detailed Component Description ............................................................................................................... 6
   3.1 Database Design .................................................................................................................................. 6
       3.1.1 Component Testing Strategy ...................................................................................................... 7
   3.2 Class/Entity Model .............................................................................................................................. 8
       3.2.1 Component Testing Strategy ...................................................................................................... 9
   3.3 Database Interface ............................................................................................................................... 9
       3.3.1 Component Testing Strategy ...................................................................................................... 9
   3.4 Design & Integration of Data Structures ............................................................................................ 9
       3.4.1 Component Testing Strategy ................................................................................................... 10
   3.5 Changes & Refactoring ......................................................................................................................... 10

4. Bibliography .................................................................................................................................................. 10
1. Introduction
This document outlines the project responsible for creating a replacement invoice management system of Cinco Computer Consultants. The new invoice management system will be constructed using a Java application that utilizes object-oriented program principles. Persistence of data within the system will be handled by a database.

1.1 Purpose of this Document
This document provides an outline of the aspects and implementation of the new invoice management system. It provides details about how objects in the API model real world entities and how the use of object-oriented programing influenced choices throughout the design process. Finally, structure through which the Java application connects to the MySQL Database in order to manage and persist data will be discussed.

1.2 Scope of the Project
The API represents the new invoice management system designed for Cinco Computer Consultants to replace their current system, which is now out of date. The new system will manage invoices, each of which contains three primary pieces of information: the salesman, the customer, and the products associated with that invoice. Products are further broken down into three main categories: equipment, consultations, and licenses. The API will serve the purpose of allowing Cinco Computer Consultants employees to add or remove invoice records to the system, as well as print both quick and detailed reports of every invoice record in the system. The API will be supported by a MySQL database which allows the API to store, retrieve, and update invoice records for Cinco Computer Consultants.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions
Encapsulation – mechanism for restricting access to some of an object’s components and for bundling data and methods operation on that data[1]

Object-Oriented Programming – a method of programming based on a hierarchy of classes, and well-defined and cooperating objects[2]

Polymorphism – The ability for a variable, method, or object to be used in more than one form

1.3.2 Abbreviations & Acronyms
[Define all abbreviations and acronyms used in this document here. This relieves you of the need to define such things within the context of the document itself and provides an easy reference for the reader.]

ADT – Abstract Data Type

API – Application Programming Interface

ER Diagram – Entity Relationship Diagram
2. Overall Design Description

This application is designed using the object-oriented paradigm, which will be used to model invoices, products and people. The interaction of these objects will be the foundation of the Cinco Computer Consultants business model.

The Java application is designed with four primary objects: invoices, products, persons, and addresses. Invoice objects will contain data about the salesman, customers, and products purchased, along with. The products object is used to store information about each product offered by Cinco Computer Consultants, to include the product’s name, code, and pricing. It is broken down into three subclasses, each one representing a type of product offered by the company. The person object is used to represent people in every role, such as salesman, consultants, or primary contacts for a customer. A customer object was defined using the person class as a foundation, and added functionality specific to customers. The application utilizes an array-based sorted list ADT in order to manage the invoice objects. The list allows for invoices to be sorted by invoice number, type of customer, or alphabetically by the name of either the salesman or customer.

Data storage is handled by a MySQL database that the invoice management API connects to using the JDBC API. The database is designed such that each class or object within the application is represented and data integrity is maintained.

2.1 Alternative Design Options

Alternative design options included:

• One option was to remove the Products class, and allow each of its three subclasses: Equipment, Consultations, and Licenses to stand alone as unique classes. This implementation would not have provided any major benefits to the project, as all three subclasses have very similar core functionality. Additionally, by creating a Products class and subclassing the three types of products, it becomes possible to use polymorphism to handle all three types of products within a single ArrayList in the API.

• The CivilianCustomer and GovernmentCustomer subclasses of Customer originally did not exist, and a variable within the Customer class would be used to store whether the customer was a government customer or a civilian customer. However, the choice was made that each type of customer would be responsible for being aware of whether it had to pay taxes on invoice orders
or had to pay a compliance fee. By implementing these subclasses, it became possible to avoid using “if-else” decision structures with the customer class to determine what fees and taxes each customer was responsible for paying.

- The Address class could have been merged into the Persons class, as its only functionality is to hold address data. This would have had the effect of adding several new fields to the Persons class. By implementing and Address class, the OOP principle of encapsulation can be enforced, as a Person object would be aware that it has an address, but is not responsible for managing the information associated with that object.

- The ADT used to sort and store the invoice records described in section 3.4 could have been implemented using a linked-list. Due to a linked-list lacking random access abilities, it was determined that it would be better to use an ArrayList. This would allow the user to retrieve a specific invoice record.

3. Detailed Component Description
The following sections provide descriptions of the MySQL database used to store data, the Java classes used to model the data, the JDBC framework used to communicate between the invoice management API and database, and the sorted-list ADT developed to manage Invoice objects within the Java application.

3.1 Database Design
A MySQL database is used to ensure the persistence of data in the new Cinco Computer Consultants invoice management system. The database is built upon five primary tables: Addresses, Persons, Products, Invoices, and Customers. The addresses table is responsible for handling the street, city, and zip code information of all people and customers. In order to prevent the unnecessary repetition of information, the State and Country tables have been created to store state and country data respectively. These tables are referenced using foreign keys in the Addresses table. The Persons table is used to store the name and person code of every person. It also contains a foreign key referencing the Addresses table. Email information for each person is handled in a separate Emails table, which connects an email to a person by way of a foreign key. The Customers, Invoices, and Products tables are used for storing data related to those objects. Products are associated with an invoice by way of an InvoiceProducts join table. This table enforces a uniqueness constraint on the invoice-products pairing to ensure that the same product appears as a single record on an invoice rather than appearing multiple times. The ER diagram in Figure 1 provides a visual representation of the database schema.
3.1.1 Component Testing Strategy
The database was tested using the same test data as the Java application. The test data was loaded into the database using an SQL script to generate a baseline system, then testing commenced. Various queries were used to test inserting new data into the database, retrieving all the data from a table or specific records, and removing data. Following the modification of data within the database, more queries were run to ensure data integrity was maintained.
3.2 Class/Entity Model

The Java application is built upon four primary objects as previously described in Section 2. The Customer class used to represent customers is an abstract class that inherits functionality from the Persons class. This allows customer objects to utilize functionality provided in the Persons class, such as getting and setting names, addresses, and customer codes. By making the Customer class abstract, basic customer functionality could be listed without having to implement a default behavior. This choice was made to utilize the decision to create separate CivilianCustomer and GovernmentCustomer subclasses.

The Products class was also made abstract. The Equipment, License, and Consultation subclasses all inherit behavior from this Products class. This design choice made it possible to normalize core functionality across all three types of products without having to define a default behavior for these functions in the Products superclass. It also made it possible to use the OOP concept of polymorphism. Equipment, License, and Consultation all have an “is-a” relationship with Products. This makes it possible to store all three types of products in an ArrayList of type Products.

The InvoiceReport class is the main driver class for the application. It is responsible for loading information from the database into the separate Java objects. Once the Java objects have been generated, XML and JSON data files are created, and an invoice report is generated. This report provides both a quick report of all the invoices in the system that includes the customer, salesman, and amount of sales. The report also includes detailed information for each invoice, to include customer and salesman information, the entire list of products purchased, and tax and fee information. A visual representation of the invoice management API is presented in Figure 2.

[Image: UML Diagram for Java Classes, Generated using ObjectAid UML Explorer[3]]
3.2.1 Component Testing Strategy
The Java application structure was initially tested using input from flat data files. Test cases were generated to simulate product, customer, people, and invoice records. The integrity of this data was tested via visual inspection of output to the standard output. The next phase of testing involved visually inspecting the XML and JSON files generated by the application. This process was once again repeated once JDBC connectivity was established, and information was being loaded from a database rather than flat data files. The invoice report system was tested against the expected output that was calculated by hand. Errors within the application are logged using Log4J, rather than simply outputting to the standard output.

3.3 Database Interface
In order for the Java application to read data from the MySQL database, the JDBC API had to be integrated. The connection to the database was established using the MySQL driver. Once the connection is established, a series of queries are used to retrieve data from the database. This data is tested for integrity, and loaded into the appropriate Java classes. If an update query is being performed, the information to be updated is checked for integrity, and if it passes, the corresponding records are changed. Due to foreign key constraints, records must be deleted in a precise order. Any child records that depend on the record to be deleted must be deleted for the parent record can be deleted. This ensures good data integrity by preventing orphaned data from existing within the database.

In order to maintain data integrity within the database, validation has to be performed within the API. All entries into the Products, Invoices, and Customers tables are checked to make sure they comply with system specifications. The InvoiceProduct table is also checked to make sure that a particular invoice-product combination did not already exist. If it does, the current record is updated. Otherwise, a new record is created.

Certain values, such as the salesmanID and customerID values in the Invoices table, are not designed to be optional. This information is checked at the time of loading data into the database. If one of these fields is null, an exception is thrown to notify the user of the invalid information. The API ignores null values for fields that are permitted to be null, such as email records.

3.3.1 Component Testing Strategy
The JDBC integration is tested using the same test data as the Java API and database tests. The current records within the database are cleared to ensure a fresh start, and the JDBC API is tested. The first test cases involved loading data into the database. After the data has been loaded, an SQL script is run to ensure the information has been properly stored within the database. Once data integrity is ensured, stage two of testing could commence. Stage two involved running a series of queries that would retrieve data from the database and load it into the Java application. Various reports were run to ensure that the JDBC queries properly populated the Java objects.

3.4 Design & Integration of Data Structures
The invoice management system implements a sorted list ADT. This ADT is designed to hold any number of objects, and is parameterized. In this instance, the ADT is used to sort invoices. The list can be
ordered in three ways: by invoice number, by customer name, or by salesman’s last name. To achieve this sorting, the ADT implements three separate comparator methods.

The ADT is built on a framework similar to that of an ArrayList. This allows the ADT to provide random access to elements within the list. The ADT by default has a capacity of ten invoices. As invoices are added to the list, a check is performed to ensure there is more space in the list. If the ADT has reached its capacity, it is expanded by five elements. This is achieved by creating a new list that is five elements larger than the old one, and copying over the old list to the new one. When an invoice is added to the list, the comparator methods within the ADT are called to place the invoice in its proper place within the list. This maintains ordering in the list as elements are added, rather than adding elements and using further resources to sort the list. Once the place for the invoice has been determined, all elements after that point are shifted in the list, and the invoice is inserted. When an invoice is removed from the list, other elements within the list shift upward in the list in order to ensure no gaps exist in the list. The capacity of the list can be decreased to free up memory. If at any point, the current size of the list is more than five less the capacity (i.e. size of 29, a capacity of 35), the list will decrease its size by five. This decreasing in size will occur unless the list is back to its original size of ten elements.

3.4.1 Component Testing Strategy
The implementation of the list ADT is tested using data loaded into the Java application from the database. Invoice objects will be created in any order, and stored in an initial list in some random order. They will then be added one by one to the designed ADT, and the results will be printed. These results will be visually compared against the expected output. Further testing will be done to ensure that the ADT’s ability to add an invoice, remove an invoice, and clear that list have been properly implemented.

3.5 Changes & Refactoring
The following changes were made to the invoice management system during its development:

- The InvoiceProducts table originally did not have start date and end date columns. The number of days that the license was purchased for was going to be calculated when the data was loaded into the database. However, it was determined that the functionality of the InvoiceReport required the start and end dates for licenses, so the columns were added.

4. Bibliography


[3] ObjectAid LLC – ObjectAid UML Generator, April 2014