|  |
| --- |
| Department of computer science & Engineering  University of Nebraska—Lincoln |
| SCHYC Fitness Management System |
| CSCE 156 – Computer Science II Project |
|  |
| **Jonathan Trost, Nathan Pittman** |
| **2/8/2018**  **Version 1.0** |

|  |
| --- |
| This document describes the design of a gym management system for SHYCY Fitness. This system imports and processes the data of gym persons, members, and products. The processed data is exported to formatted data files. This application also uses JDBC to allow full functionality and control of an SQL database. |

# Revision History

[This table documents the various major changes to this document]

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description of Change(s) | Author(s) | Date |
| 1.0 | Initial draft of this design document | Jonathan Trost,  Nathan Pittman | 2018/02/08 |
| 2.0 | Invoice and detailed description functionality | Jonathan Trost,  Nathan Pittman | 2018/02/22 |
| 3.0 | SQL database and control functionality | Jonathan Trost,  Nathan Pittman | 2018/03/20 |
| 4.0 | Linked-List and final draft of this design document | Jonathan Trost,  Nathan Pittman | 2018/04/03 |

Contents

[Revision History 1](#_Toc349390336)

[1. Introduction 4](#_Toc349390337)

[1.1 Purpose of this Document 4](#_Toc349390338)

[1.2 Scope of the Project 4](#_Toc349390339)

[1.3 Definitions, Acronyms, Abbreviations 4](#_Toc349390340)

[1.3.1 Definitions 4](#_Toc349390341)

[1.3.2 Abbreviations & Acronyms 4](#_Toc349390342)

[2. Overall Design Description 4](#_Toc349390343)

[2.1 Alternative Design Options 4](#_Toc349390344)

[3. Detailed Component Description 4](#_Toc349390345)

[3.1 Class/Entity Model 4](#_Toc349390346)

[3.1.1 Component Testing Strategy 5](#_Toc349390347)

[3.2 Class/Entity Model 5](#_Toc349390348)

[3.2.1 Component Testing Strategy 5](#_Toc349390349)

[3.3 Database Interface 5](#_Toc349390350)

[3.3.1 Component Testing Strategy 5](#_Toc349390351)

[3.4 Design & Integration of Data Structures 5](#_Toc349390352)

[3.4.1 Component Testing Strategy 5](#_Toc349390353)

[3.5 Changes & Refactoring 5](#_Toc349390354)

[4. Additional Material 5](#_Toc349390355)

[5. Bibliography 6](#_Toc349390356)

# Introduction

The project described below is a gym management Java application that uses object-oriented programming to manage person, member, and product data. The data is formatted and manipulated to produce XML and JSON data files as output. The project also includes functionality to control and SQL database through JDBC integration in this project.

## Purpose of this Document

This document outlines the basic structure and design of the gym management system. It details the fundamental parts involved in this system.

## Scope of the Project

This system is intended to be used by SCHYC Fitness to fulfill their data management needs. Existing data can be imported to work with the new system. Imported data is organized and into logical categories for further data manipulation.

## Definitions, Acronyms, Abbreviations

### Definitions

Encapsulation: Storing data inside the relevant class that uses the data

Inheritance: Traits passed down from a parent class to a child class

Abstraction: Hiding details and only revealing essential features of an object

Polymorphism: Making methods and classes in a general format that can be used flexibly

### Abbreviations & Acronyms

OOP: Object Oriented Programming. The design philosophy that this project was built with.

XML: A formatted data storage structure.

JSON: A similar formatted data storage structure.

SQL: A database management language

JDBC: Integration functionality to allow control of an SQL database through Java

# Overall Design Description

This project uses object-oriented programming techniques such as encapsulation, inheritance, abstraction, and polymorphism. Classes used in this project consist of a person reader, a member reader, an asset reader, a data converter, various object definition classes an XML writer, and a JSON writer. Using JDBC and an SQL database this program also allows for full control of the database through the Java Application.

## Alternative Design Options

Alternate design options considered but ultimately abandoned included having only one Person class.

Several iterations of the SQL database were initially considered but the design detailed below was decided on as it was the best combination of functionality, security and minimizing the number of database accesses per item of code.

Initially a single invoice writer was used but as of 4.0, the application uses two separate invoice writers to handle the list and arrays separately.

# Detailed Component Description

This program utilizes classes for each module of the data management system. Constructors are employed to create objects used throughout the program. As well as various getter methods for object manipulation.

## Database Design

Database structure follows the Java application closely tables included in the database are: Address, Person, Member, Invoice, YearLongMembership, DayMembership, EquipmentRental, and ParkingPass. Each table represents an independent data structure within the java application and utilizes many-to-one relationships between the various services and Invoice and one-to-one relationships between Address, Person, Member and Invoice.

### Component Testing Strategy

1.0: Bad data structures and variable types were passed to ensure correct parsing and rejection of appropriate data occurred.

2.0: Improperly built invoices, bad variable types and various types of bad object arrays were passed to determine proper functionality

3.0: JDBC focused stress testing within the java application was used to ensure proper functionality, this included incorrect calls, data types and improperly formatted data. Within the SQL database, defaults and other data checks were put in place in the database design to ensure that improperly formatted data could not be passed into the database. When reading from the database, zeros and other null or empty data was tested to ensure proper building of invoice structures. Various incorrectly formatted data structures were implemented to ensure that the java application could properly handle stray data structures.

4.0: No new component testing strategies were used during this design phase.

## Class/Entity Model

The project is split into four SRCs, one containing the classes for the objects used throughout this project. These include: Address, Asset, DayMembership, EquipmentRental, Member, ParkingPass, Person and YearLongMembership. These classes contain constructors, and other pertinent functions directly associated with the objects of the same name.

The second SRC contains the reader and writer classes used to manipulate the flat data files, the arrays of objects created from the flat files and the writers necessary to write these objects to their appropriate files, including building data passed from the sql database.

The reader and writer classes which include AssetReader, DataConverter, JSONWriter, MembershipReader, PersonReader and XMLWriter, contain all the functionality needed to read and transfer data from the flat data files to the formatted file output. The DataConverter class contains the main function by which the program is executed but otherwise serves as a class from which the appropriate reader and writer functions are called. The invoiceReader, Writer, and Data classes provide functionality for reading from the SQL database and constructing all appropriate data structures. SQLFactory provides some factory SQL connection methods used throughout the program and AssetMasterList provides a list of all data contained within the flat file given to the program.

The third SRC is log4j, all this src does is provide log4j a resources location to proper execute error handling.

The final SRC is com.sf.ext, this contains InvoiceData, the class containing all methods used to write data to the database. This SRC also contains ComparatorTotal, the overwritten compare method used to compare values from the linked list. This SRC also contains InvoiceIterator, the linked list iterator and InoiceNode and InvoiceList which are used to construct and manipulate the linked list of invoices.

### Component Testing Strategy

This component was tested using a number of erroneous data types within the flat files. Examples include number data types in String data locations or completely null or empty data. This same badly constructed data was also passed into the database and read from the database to ensure proper handling of data throughout the program.

## Database Interface

The SQL database is written to with through the InvoiceData class found in com.sf.ext. This class contains all methods used to write data to the database. All data is read from the database using the InvoiceData class in readersWriters. InvoiceData also parses and constructs appropriate data structures, sorts them, and bundles them into an array for processing. All data validation is handled here as well. This class also contains functionality to round up and contain any stray data the might be in the database.

### Component Testing Strategy

This component was heavily tested using improperly formatted data as well as stray data manually added to the database.

## Design & Integration of Data Structures

The linked list was designed following standard java linked list convention, an overwritten comparator and iterator were created to allow for sorting of the list based on the total of the individual invoices and to allow iteration through the list.

Standard linked list convention can be found here:

https://docs.oracle.com/javase/9/docs/api/java/util/LinkedList.html

### Component Testing Strategy

No major component testing was required during this phase of design, all data validation was previously handled.   
Some minor testing was done to the comparator and iterator.

## Changes & Refactoring

3.0: Boolean flags were added to Invoice class to determine type of associated membership

3.0: Abstract methods added to object construction classes

4.0: A second writer was implemented to print details from the linked list

4.0: A second method was added to calculate invoice totals for the purposes of list sorting to the InvoiceWriter class.

DataConverter went through various changes during each phase of this project to allow functionality during each phase. These changes had no effect on the overall functionality of this project.

# Additional Material

No additional materials necessary with this design document.

# Bibliography

[1] *Citation Styles*. (n.d.). Retrieved December 19, 2012, from [http://www.cws.illinois.edu/workshop/writers/citation/](http://www.cws.illinois.edu/workshop/writers/citation/))

[2] Dr. Hasan M. Computer Science 156 Spring 2018 Lecture 6 [PowerPoint slides]

Retrieved January 25th and 26th 2018 from <https://canvas.unl.edu/courses/30868/pages/lecture-6-java-and-php-class?module_item_id=728917>

[3] Oracle Corporation. “Class LinkedList API” *LinkedList (Java SE 9 & JDK 9 )*, 1 Jan. 2007, docs.oracle.com/javase/9/docs/api/java/util/LinkedList.html.