Capstone Project Cover Sheet

Capstone Project Name: Java/SQL Inventory Management System

Student Name: Erik Platt

Degree Program: BS IT Software Emphasis

Student Mentor Name: Michelle Vore

Capstone Project Waiver/Release Statement Acknowledgement

It is the policy of Western Governors University ("WGU") that student Capstone projects should not be based upon, and should not include, any proprietary or classified information or material belonging to your employer or any other organization ("Restricted Information") without appropriate authorization.

Please confirm (by signing below) that you will complete (and upload into TaskStream) the IT Capstone Waiver Release form (verbiage is available in Appendix 2 of this document) indicating that your project does not include any restricted content. If you have included restricted content, please confirm that, in addition to the IT Capstone Waiver Release form, you will upload a suitable release letter giving you permission to use restricted information (A sample release letter is available in Appendix 3 of this document).

Erik Platt	12/10/2015
Student's Ink or Electronic Signature	Date Signed

Table of Contents

Capstone Project Summary

Global Document Solutions is a Document Storage, Document Destruction (shredding), and Courier provider. Global Document Solutions seeks to create a single solution that will allow order entry and tracking for all of these functions. In addition, there will eventually be integrated service accountability to both management and customers. Finally, Global Document Solutions wishes to market the solution to other service providers and thereby leverage the solution to create an additional revenue source.

In order to meet these requirements as well as allow for possible use in different vertical markets, the solution began with the most urgently needed function and will add functions in phases. The most urgent function was tracking locations of file boxes in the document storage warehouse. Previously, a spreadsheet tracked this and with thousands of boxes, it was quite cumbersome. The core functionality was abstracted in developing the solution so that only four core components form the basis of all transactions. The core components are container, location, work order and customer.

Global Document Solutions is essentially a logistics company. Moving something from one location to another is the primary work done. In the case of document storage, a file box (container) moves from the warehouse (location) to the customer’s site (location) or vice versa. In the case of document destruction, a file box (container) or shred console contents (container) are picked up from a customer location and brought to the warehouse (location) to be shredded. In the context of courier service, a package (container) is picked up from one office (location) and delivered to another office (location). By keeping these objects generic, we can also maintain the capability of utilizing this abstract conception to address the need of many other industries as well.

As mentioned earlier, functionality will be added in phases. The scope of the completed project is only the first phase. The following paragraphs detail the first phase. The second phase will be to integrate the core functionality into a web app and hosting on Amazon Web Services (AWS) utilizing Jetty. The third phase will be to allow for multi-tenancy and self-provisioning capabilities. Robust reporting capabilities make up the final phase, which will be available to users of the system as well as customers of the users. The objective is that a customer can look up a container and audit its entire life. A hospital – for example- can find out the exact date that the destruction of the contents of a specific shred console took place. Integrated dashboards with drill-down capabilities will display the information.

The initial phase is a program written in Java that tracks boxes (generically referred to as "containers" from now on) in locations in a warehouse. In an effort to limit the scope of the project to make it manageable in the context of this course, it only included writing the Java program. This includes the complete business logic and database updates. Outside of the completed scope is the web interface and additional phases.

The program allows a user to enter a customer, enter locations for containers including number of containers (capacity) of each location in the warehouse, enter a container into a location, remove a container to go to a customer, and permanently remove a container from storage. The user can also look up a container location by container number or display a report of all containers by customer ID.

Work Order functionality allows the user to pull boxes for delivery to customers. Work orders include customer delivery information as well as containers requested, and date and time the Work order is completed. Work orders print to the screen.

The user interface is a simple text-based, menu driven console interface. Options are available in the interface to perform all of the functions listed in the previous paragraphs.

All customer, container, location, and work order data is stored in a SQL database. Java Database Connectivity (JDBC) is utilized for CRUD (create, read, update, delete) functionality between the Java program and SQL database.

The development began by creating the database and database tables. Once this was complete, the actual Java coding began using the eclipse IDE. The MySQL library was added to the build path and all of the class files and class structure was created in anticipation of coding the classes. The four JavaBean classes were created first. These serve as object representations in the context of object oriented software development of a database record. Next, a couple of utility classes were created. A connection manager class was created to manage the connection to the database and an input helper class was created to interpret and parse the string input from the user so that Java could recognize it as the correct data type. The object manager classes were created next. These four classes contain the code that performs the actual manipulation of the database records represented as JavaBean objects. The work on these four classes represented the bulk of the work on the project. The main class which represents the main menu and interface to the program was written simultaneously as the four object manager classes were developed. Once this was complete, unit and integration testing were performed, and the work of the project was complete.

Review of Other Work

Even though we are in the electronic paperless age, there are still hundreds of millions of cubic feet of file boxes stored globally.

> For all the talk of a paperless office, many organizations have a very real business need for paper records. Some areas of business depend on physical authentication features such as signatures, stamps, and seals (TAB, 2015).

One of the largest commercial records management organizations in the U.S. stores 30 million cubic feet of records alone (Alston, 2015). The multitude of privacy and corporate governance regulations are contributing to the need for records management systems that can maintain records in adherence to best practices. It is so important that some organizations are elevating accountability for records management to senior level executives (Weise 2012).

The current problem that Global Document Solutions faces is organization of customer records. However, the fact that the Document Management Services market is reported to be a $6 billion market with 1,587 businesses marketing these services indicates that there is a robust market for records management systems.(IBISWorld, 2015).

In meeting these and other requirements, it is critical that an organization be able to provide evidence of reliable standards and due diligence processes (TAB, 2015).

The above quote suggests that robust audit capabilities are crucial to a records management system in order to provide the necessary value in today’s regulation heavy environment. Audit capabilities must track the entire life cycle of all records including destroy date and destruction validation such as a certificate of destruction. The system architecture must be able to support mandatory compliance to records management best practices so that the following controls cannot be circumvented:

- General records management policy including retention policy
- Detailed procedures and standardized workflows pertaining to record creation, classification, retrieval, location tracking and disposal
- Functional records classification which divides records into categories based on the different business activities that they support
- A records retention schedule which assigns standard time periods for keeping record categories based on documented statutory, regulatory and operational requirements
- Information security protocols which assign sensitivity risk levels and identify appropriate security safeguards to mitigate that risk (such as authorization levels for different employees within the organization)

It is important, then, that the unique storage and distribution needs of paper records not get overlooked amidst the justifiable excitement around electronic records retrieval (TAB, 2015).

With all of the “justifiable excitement” regarding electronic records management, it is important not to overlook some basic file management capabilities to integrate such as:

- Barcode or RF technology that help to automate the retrieval and tracking of physical records
- Web interface to facilitate pick up of new records or retrieval of records in storage
- Ability to print a label or work order to insure that records are correctly identified and located

Organizations need to develop a metadata model that will be used throughout the organization. A metadata model is a collection of approved descriptive elements that will be used to manage and retrieve information and records throughout the organization. Currently, there are government identities and businesses who are doing exactly this(Weise 2012).

Metadata must be accounted for in order to accommodate current information management best practices. In the context of physical records, metadata could be as informal as file descriptions or as formal as an exclusive list of keywords or codes

The best chance of success includes creating a solution that takes into account the current market for records management services and records management best practices.

Watermark Technologies is a leading provider of Document Management Solutions and offer a powerful system to manage and store documents electronically. Hundreds of businesses worldwide enjoy the simple and easy to use DMS that we provide as it saves them time and money (Firebird 2012).

Watermark Technologies utilizes a version of SQL in their commercial document management solution. Their continued viability and success provides justification for the utilization of a SQL based platform for document management software.

Project Rationale

Global Document Solutions had been using a spreadsheet to track the locations of customer’s file boxes as well as to maintain a count of how many boxes in storage for each customer. This solution was very prone to human error and made searching for boxes in the warehouse cumbersome. Either the entire spreadsheet had to be printed and manually searched as the employee is looking for the box or each box number would be searched at the computer on the spreadsheet then written down on a piece of scratch paper and then pulled. Often the date that the box was pulled wasn't recorded. When new boxes would come into the warehouse, they often would not be recorded on the spreadsheet until they were put on racks which might be a few days after entering the warehouse. As more and more boxes entered into the warehouse, a solution was needed.

The current available Document Management Systems are massive systems that require additional MS SQL licensing and others that are designed as electronic records management systems with physical records management as an afterthought. The only cloud-based solution built primarily for physical records was designed before cloud based SaaS delivered solutions were viable. Because it is designed as a traditional server/client application and uses MS SQL, the licensing of SQL is added in to the pricing.

Global Document Solutions could have built a small server with a MySQL database and a simple HTML/PHP interface. This would allow all of the boxes and locations to be tracked, but when reporting functions begin to be built into it, it starts to become a larger project that is better addressed with better tools. It also makes sense to use this functionality and leverage it for the document destruction and courier business units as well. Many customers utilize multiple services and it is reasonable that one system would contain all customer and service information.

Given the state of the market for systems addressing the need for physical document storage providers, the solution could be marketed to other service providers at a very reasonable price. For these reasons, Global Document Solutions build the system and is the pilot customer.

Systems Analysis and Methodology

The spreadsheet as a method of keeping track of boxes and box locations was completely inadequate. It was prone to human error and there simply were not enough checks to make sure all information was recorded on the spreadsheet and auditing was very crude if it could be called auditing. The best way to address these shortcomings is to design a new solution from scratch built on a native web platform using JavaScript and JavaScript libraries (JQuery, AngularJS) to do as much data manipulation on the client-side with only the core business logic on the (cloud-based) server. This will also allow drill-down dashboards to improve reporting well beyond anything available today. This architecture will ensure robust security with the least number of database queries to the server. For large organizations storing a million or more boxes, the difference in performance will be apparent.

The core business logic was written in Java and consists of four Java classes. One class each for the objects container, location, work order, and customer. These four objects have all of the attributes and methods necessary to form the core functionality of the program. Each instance of each object will store its state in a MySQL database through JDBC. Each instantiation of each of these classes represents a row in the database. All of the fields in the database are represented by the class variables and all of the getters and setters are included in the class as well following encapsulation best practices. The code to handle the SQL transactions are handled in four manager classes. The console interface was written as each method was written in each manager class as the primary means of reaching that method.

At this point, the code will be migrated to the Amazon cloud and an HTML/JavaScript interface created. When the interface is created, reporting capabilities can begin to be built and tested. The interface will include responsive design principles so the web app will be accessible from any smartphone or tablet. It is also possible to integrate barcode scanning capabilities in the web app so that a smartphone can replace the dedicated barcode scanners required for other solutions currently on the market.

The methodology employed will be a hybrid of the traditional waterfall model and agile. The project deliverables that resulted from the scope of the initial project followed the waterfall model. This was primarily because there was only one person performing the work and the deliverables were reasonably clear and concise. The requirements were known and unambiguous. The design was not complex because the architecture incorporates only two discreet technologies: Java and MySQL. Testing included validation that the following functions perform correctly:

- Enter a customer
- Look up a customer
- Enter locations for containers including capacity of each location in the warehouse
- Enter a container into a location
- Remove a container to go to a customer
- Permanently remove a container from storage
- Look up a container location by container number
- Create report of all containers by customer ID
- Create Work Order
- Add Customer to Work Order
- Add containers to Work Order
- Display Work Order to screen

After this initial phase was completed, the project has shifted to an agile model. The system will be tested and revised in an incremental fashion in response to feedback from users and the business. The overall process falls more in the category of agile because an effort is made to find the smallest functioning core program, build that, and then refine it until it becomes the desired solution. This process will culminate in the best result – which will then be continuously improved.

A very important piece of the continuous improvement will be adding reporting functions specifically dashboards that can be drilled down to find information about single containers such as audit trail, destruction date and time/date delivered.

By leveraging current technology to address the old problem of paper records, businesses will have dashboards close at hand, and large organizations will be able to better manage the vast number of boxes that they store.

Goals and Objectives

The primary goal of this project was to track the location of containers under the responsibility of Global Document Solutions. Tracking the containers at this point in the overall project only includes the basic capabilities of finding a box in storage and creating the order to remove a box to deliver to a customer. This goal has been met at this stage of the project. The twelve requirements outlined in the previous section have all been met and tested.

The objectives at this point in the overall project represent a small portion of the overall project in terms of hours required but also critical in terms of being the foundation of what is to become a much larger project. As previously stated, the scope of the current project includes the core business logic required to track containers in the warehouse. However, it also serves as a blueprint to add functionality with respect to courier deliveries and shred containers. As such, the current scope of the project includes three main objectives that will be discussed in detail below.

The first objective is to allow for the greatest scalability by handling many processing functions at the client browser in the future HTML interface. The core business logic will return result sets to the future browser interface where the data is further manipulated. This architecture will reduces the number of calls made to the server and queries to the database allowing for the smallest possible server footprint and in turn allowing for the greatest scalability. The architecture employed by the completed scope provides the required extensibility to be incorporated into a SaaS/Cloud model with minimal footprint on the server resources.

The second objective was to create a set of four SQL database tables that comprise the database that store the data associated with each of the objects represented by the program. These four tables are container, location, customer, and work order. This objective was met over the course of a couple of hours. The database and tables were created through a phpMyAdmin interface on a local instance of a MySQL database.

The third objective was to create a cohesive collection of Java classes that make up the core business logic. These classes are abstract representations of container, location, customer and work order. Each of these classes include the fields and getter and setter methods required to instantiate an object that represents a database row. Manager classes for each of the four core objects include the logic required for database queries through JDBC to provide functionality. Additionally, manager classes for each core class and database utility classes to manage connections and allow for use with any SQL database were created. These classes were all written within the proposed timeline.

The main java class receives input from the user and allows the user to select one of the functions:

- Enter a customer

- Look up a customer
- Enter locations for containers including capacity of each location in the warehouse
- Enter a container into a location
- Remove a container to go to a customer
- Permanently remove a container from storage
- Look up a container location by container number
- Create report of all containers by customer ID
- Create Work Order
- Add Customer to Work Order
- Add containers to Work Order
- Display Work Order to screen

To facilitate these functions, one JavaBean class per each of the following was created to include all getters and setters for each attribute:

- Container
- Location
- Customer
- Work Order

A java manager class for each of the core object representations was created to manage all the queries to the MySQL database from each of the four JavaBean classes. The manager class for each of the JavaBean class includes the queries for all functions of the program. So for example, the functionality to prompt for and accept all customer information is contained in the

CustomerManager.java class. The code to create the customer record in the database is also part of the CustomerManager.java class.

Additionally, a few utility classes were necessary. A ConnectionManager.java class maintains, opens and closes the connection to the database when necessary. The connection manager class also hides the username and password to the database behind private variables for increased security and future multi-tenant functionality. The connection manager class also provides the logic to connect to alternative database types for future extensibility.

By meeting these objectives in the creation of the classes, the goal of a fully functional system to track containers in their locations and the movement of containers in and out of locations, to customer sites and back, and permanently remove containers from storage has been achieved.

While the above objectives has met the goals in the current scope of the project, future phases of the project will include a web interface making heavy use of JavaScript, AngularJS, and JQuery to allow data manipulation to be done by the browser to reduce server-processing requirements and improve scalability. Additionally, future code refactoring and leveraging the object orientation of java can extend the functionality beyond container storage tracking to package and shred-collection container tracking.

Project Deliverables

The project deliverables for this project were the individual Java classes and the SQL database structure. These two components support the individual functions that together form the entire program. There were eleven Java classes proposed at the beginning of this project. However, it required thirteen classes to support the desired flexibility and extensibility while maintaining the required functionality. Proposed were the main class, one JavaBean class for each of the container, location, customer and work order objects, one manager class for each of the objects, a connection manager class to maintain the state of the connection to the database, and finally a class to manage and process input from the user. Two more classes were added although their functionality could have been included in the ConnectionManager.java class. The two additional classes were a DBType.java class, which just included an enumeration of different types of databases in case a different type of database was utilized at some point in the future. The other added class was a DBUtil.java class which maintained the database username and password in a separate class for security. The SQL database mirrors the object model of the program and has four tables representing the four core components of: container, location, customer and work order. In the initial scope of this project, the SQL tables have a minimal set of fields. These fields will be expanded as testing and increased functionality dictates.

The main class includes the menu of executable functions within the program. This represents the user interface and has all of the calls to the JavaBean and manager classes. Because the current scope of the project is a console application, the main class and menu looks antiquated, but it includes all of the core business logic to perform the required functions. This core business logic can now be leveraged by being hosted in the cloud and building a web interface to perform the function of the menu.

Each of the four JavaBean classes represent the state of the objects they abstract. All of the attributes of each of the four object abstractions maintain their state within the JavaBean classes. For example, the container JavaBean class maintains the fields: accessedBy, altId, containerNumber, createDate, customerId, description, destroyDate, lastAccessed, locationNumber, nextAccess, subContainers, and superContainer. Getter and setter methods for each field are also maintained within each JavaBean class. The JavaBean classes are relatively simple in that they have a simple purpose of maintaining the state of each object so private variables and getters and setters are all that is included in each of these classes.

The manager classes for each of the four objects contain the code to call the JDBC in order to query the database. All of the functions relating to the database for each of the objects are handled in these manager classes. Each menu item on the menu in the main class is supported in one of the manager classes. This includes all CRUD (create, read, update, delete) functions. For example, to create a new customer, all of the SQL to perform this task is included in the customer manager class. The work order manager class enables a user to remove a container from storage and update the database to reflect that the container is not in the warehouse but not delete the container from the database. This is accomplished by updating the container location to the work order number.

The connection manager class maintains the connection to the database and allows the object manager classes to have an open connection when a query is run. Every time an object manager class runs a query, the connection manager class is called to open the connection, maintain the connection, and then properly close the connection once the query has completed. The connection manager class also catches any errors or exceptions thrown by the interaction with the database and handles them.

The SQL database tables are container, location, customer and work order. The java program queries, updates, inserts and deletes database records. Following are the structure and fields in each of the four database tables in order to facilitate the base functionality necessary for the program to operate.

The container table has the most fields. In order to track each container the following fields were created:

- Accessed By
- Container Number (Primary Key)
- Alternate ID (customer numbering system)
- Location Number
- Customer ID
- Create Date
- Last Accessed
- Next Access
- Destroy Date
- Description
- Sub Containers
- Super Containers

The next database table required is the location table. The location table only stores two fields:

- Location Number (Primary Key)
- Quantity

The customer table stores basic customer and contact information. Later iterations of the program will allow for separate shipping and billing addresses, authorized employees by department and well as levels of access, and also common delivery addresses. The initial iteration includes a table with these fields:

- Customer ID (Primary Key)
- Company Name
- Contact
- Street Address
- City
- Zip
- Phone
- Authorized Users

The work order table stores all the data specific to a work order. A minimal number of fields are employed to initiate the first version of the work order. The purpose initially is just to enable the creation of a work order to pull a container from storage and deliver it to a customer. This will require the following fields:

- Work Order Number (Primary Key)
- Customer ID
- Date/Time WO Created
- Created By
- Priority
- Origin Location
- Destination Location
- Container ID
- Date/Time Delivered
- Delivered By

Project Plan and Timelines

Project Deliverable or Milestone	Duration	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
Create SQL database tables	*2 hrs*	*10/25/2015*	*10/27/2015*	*10/26/2015*	*10/27/2015*
Create four JavaBean Classes	*2 days*	*10/27/2015*	*10/27/2015*	*10/29/2015*	*10/28/2015*
Create Connection Manager Class	*1 day*	*10/30/2015*	*10/28/2015*	*10/31/2015*	*10/28/2015*
Create Input Helper Class	*1 day*	*10/31/2015*	*10/28/2015*	*11/1/2015*	*10/28/2015*
Create Container Manager Class	*2 days*	*11/2/2015*	*10/28/2015*	*11/4/2015*	*11/4/2015*
Create Location Manager Class	*1 day*	*11/5/2015*	*11/5/2015*	*11/6/2015*	*11/5/20105*
Create Customer Manager Class	*2 days*	*11/9/2015*	*11/6/2015*	*11/11/2015*	*11/9/2015*
Create Work Order Manager Class	*2 days*	*11/11/2015*	*11/10/2015*	*11/13/2015*	*11/10/2015*
Create Main Class	*2 days*	*11/16/2015*	*10/27/2015*	*11/18/2015*	*11/10/2015*
Testing	*4 days*	*11/19/2015*	*11/10/2015*	*12/2/2015*	*11/13/2015*

The project was scheduled to begin on October 25, but actual start date was October 27. Because creating the database tables only took two hours and the start date for the next deliverable was October 27, we were back on schedule to begin creating the JavaBean classes as soon as the database tables were completed. The JavaBean classes only took one day rather than the projected 2 days so that by the time the second task was complete, the project was one day ahead of schedule. One day was allotted in the project plan to create the connection manager class. However, since it took less than a day to create, by October 28, the project was three days ahead of schedule. The work on the input helper class was also completed on 10/28 with the connection manager class putting the project 4 days ahead of schedule. The first of the object manager classes – the container manager class – took six days instead of the projected 2 days so that at the conclusion of this task the progress of the project was as projected on schedule on 11/4. Because the location manager only took less than a day on 11/5, the project progress was now one day ahead of schedule. The customer manager class took 3 days to complete leaving the overall progress 2 days ahead of schedule. The work order manager class took the projected 2 day to complete leaving the overall progress 2 days ahead of schedule by 11/10. Because the development of the main class was actually completed as the work in the manager classes was the main class was also complete at this point in the project development. The scheduled date for the completion of this work was 11/18 putting the development work eight days ahead of schedule on 11/10. . Testing took two days so that the project was complete by 11/13. This was approximately 2 weeks ahead of schedule.

Project Development

The actual development began with creating the database and database tables. A WAMP (Windows, Apache, MySQL, PHP) server was already available and the database and tables were created through the phpMyAdmin interface (See Appendix 1). The database was created and named “docstore.” Four tables were created – one each of container, customer, location, and work_order. In each table, the fields were created to match the variables in each of the object “bean” classes (See Appendix 2). The project was scheduled to begin on October 25, but actual start date was October 27. Because creating the database tables only took two hours and the start date for the next deliverable was October 27, we were back on schedule to begin creating the JavaBean classes as soon as the database tables were completed.

Creating the JavaBean classes was also accomplished quickly. The eclipse IDE was used so all that was required to create the JavaBean classes was to create the Java variables to map to the database fields including their access modifiers (all “private”) and eclipse can create all of the associated getters and setters (or accessors and mutators) (See Appendix 3).

At this point - as per the Project Plan – a connection manager class was created. The connection manager manages the connection between the program and the database. The connection manager can be called from the object manager classes to return an instance of the database that can be queried. One day was allotted in the project plan to create this class. However, since it took less than a day to create, at the conclusion of this class, the project was three days ahead of schedule.

The next item on the project plan to be completed was the input helper class. Because the user input from the console application is all interpreted as string input, the input helper class is called to parse the string input into the appropriate data type in preparation for the database. Four methods were required for the data types used in the database. The first method is the getInput method which provides a prompt and provides a BufferedReader to read string input. Next is the getDoubleInput, which parses numeric string data into a Double data type. The third method required was the getIntegerInput. It might be reasonably obvious that this method takes numeric string input and returns an Integer. The last method was the most challenging to create. A method was required to take string input in the pattern of a date and return a SQL formatted date which could be stored in the database. However, the date had to first be parsed as a Java formatted date, then converted by calling another method to a SQL date (See Appendix 4). The work on this class was completed on 10/28 with the connection manager class putting the project 4 days ahead of schedule.

The project up to this point was primarily establishing the framework for the application. The next four deliverables were the object manager classes. In the object manager classes was where the functionality would be coded. When a user selects a menu item on the main menu, that selection is calling a method contained in one of the object manager classes. The bulk of the work was completed in these four object manager classes.

The first object manager class created was the container manager class. This class included seven methods for manipulating containers: insert (to insert a container into the database), addContainer (to prompt for all the container fields, call insert, then if successful, display the inserted container fields – See Appendix 5), getContainer (to prompt for a container number and provide limited details for that container), delete (to remove a container), displayContainersByCustomer (to display all containers for a given customer), displayContainersByAltId(to display a container by the customers numbering system), getRow (returns container information for a given container number). In a departure from the original project plan, the main menu was created as the object manager classes were being coded so that each time a method was completed to perform a specific function, the menu item was added to the menu and tested. Additional time beyond what was scheduled for this class was required for research and development. Because of this, the four days that were gained previously were lost in the development of this first object manager class. The net result was that the project was on schedule.

The second object manager class that was written was the location manager class. Although this class has as many methods as the container manager class, the development went much quicker building on what had been learned in creating the container manager class and due to the fact that the location object only has two fields (variables.) The methods in the location manager class allow the user to add a location, display all locations, select a location and edit a location. Because this class only took a portion of a day, the project progress was now one day ahead of schedule.

The next object manager class created was the customer manager class. The customer manager class included five methods to perform the functions required. Originally, there were only two functions planned for manipulating customer: adding customers and looking up customers. Two additional requirements were added. The additional requirements were the ability to remove a customer and the ability to look up a customer by any substring of the customer name. This is useful for lookups but especially in our situation where we have many customers who are law offices. There are two common scenarios that occur with law offices. The first is that the partners of law offices can change often so that Smith, Jones, Wisenheimer and Deuche can change to Jones, Wisenheimer and Quitiquit. The search will allow the user to search for “jon” and the target will be among the query results. The second common occurrence is that many law offices have names such as “Law Office of Jones and Hammer” but they are commonly just referred to as “Jones.” Having to search for “The Law Office of…” is a lot of keystrokes and is likely to return many law offices so the ability to search for “jones” is very useful. See Appendix 6 for the getCustomerLike method code. This class took 3 days to complete leaving the overall progress 2 days ahead of schedule.

The last object manager class completed was the work order manager class. This class included five methods to perform all of the required work order functions. Rather than breaking out the functions of creating a work order, adding a customer to the work order and adding containers to the work order as separate functions, they are all covered in the creation of a work order. After all, a work order is meaningless if it is missing any of this information. So the main menu only has two menu items associated with work orders. The user can create a work order or look up a work order. The requirement to print the work order to the screen is also incorporated into both of those functions. Anytime a work order is created or looked up, it is printed to the screen. The challenge associated with the work order class was in displaying the work order number when a work order was created. The work order number is automatically populated by the database and also auto-incremented. The challenge was to create the work order with the user provided information and simultaneously have the work order number returned to display to the screen. The challenge was overcome by adding the RETURN_GENERATED_KEYS statement to the end of the SQL prepared statement (See Appendix 7.)

The next item on the project plan is the main class including the main menu. As previously mentioned, the main class was developed as each method in the object manager classes was created. So by the end of the development of the object manager classes, the main menu was complete (See Appendix 8). This was completed by 11/10. The scheduled date for the completion of this work was 11/18 putting the development work eight days ahead of schedule. However, there was still testing to do.

Unit testing and integration testing was completed with no major issues. Each method was tested as part of the unit testing. Each method that made a call to another method was isolated and tested. Finally, each menu item was tested for function and for exception handling especially when incompatible data type input was given. The results were sufficient to meet the requirements, however to function as a useful tool, further exception handling features will be added. Testing took two days so that the project was complete by 11/13. This was approximately 2 weeks ahead of schedule. While it seems that being done ahead of schedule would be better than behind schedule, in this case it illustrates the inexperience with the subject matter and therefore inability to accurately estimate development timelines for deliverables.

The work completed within the scope of the initial phase of the project went very well and according to or ahead of schedule. In terms of project scope, all project requirements were met. The original scope required twelve basic functions:

1) Enter a customer
2) Look up a customer
3) Enter locations for containers including capacity of each location in the warehouse
4) Enter a container into a location
5) Remove a container to go to a customer
6) Permanently remove a container from storage
7) Look up a container location by container number
8) Create report of all containers by customer ID
9) Create Work Order
10) Add Customer to Work Order
11) Add containers to Work Order
12) Display Work Order to screen

The completed implementation included sixteen functions in all and some deviated from the original proposed capabilities. The sixteen functions (as seen on Appendix 8) are:

1) Add Location (maps to 3 above)
2) Display all Locations (added)
3) Select Location (added to view containers by location)
4) Edit Location (added to edit the quantity that a location can hold)
5) Add Container (maps to 4)
6) Permanently Remove Container (maps to 6)
7) Find Container by Number (maps to 7)
8) Containers by Customer (maps to 8)
9) Find Containers by Alt ID (added to allow the user to look up container by the customer's container number)
10) Add Customer (maps to 1)
11) Remove Customer (added to remove customers)
12) Find Customer by ID (added to lookup customer by ID)
13) Find Customer by Name (maps to 2, can lookup customer by any substring of customer name)
14) Create Work Order (maps to 9, 10, 11, 12)
15) Find Work Order by Number (added to lookup Work Orders)
16) End Program (added to end the program)

Problems Encountered

The most significant problem encountered was in creating the work order. The work order number is automatically assigned and incremented by the SQL database. At the time a work order is created, one of the most important pieces of information to have is the work order number. It was tricky to simultaneously create the work order (which triggers the database to assign the number) and display the work order number in one method. To accomplish this, it is necessary to call RETURN_GENERATED_KEYS with the prepared statement.

Another issue was mapping Java data types to SQL data types. Most have compatible types like int and varchar or string and text. But dates are a little more problematic. In the end all that was required was two methods in the input helper class. The first one prompted the user for a date which was input as a string and parsed the input to a Java date. The second method converted a Java date to a SQL date. The first method called the second method so that the end result from a user entering a date as a string was a SQL date. Only then could the date be inserted in the database.

Unanticipated Requirements

The unanticipated requirements fall into two different categories, those that could have been reasonably anticipated and those that improved the functionality of the program. In the first category is knowledge of code required to perform the desired functionality. Basically, previously unknown techniques had to be employed. The use of these was unanticipated but one could reasonably anticipate that some learning would be involved in the execution of the project. Also, reusing code used to perform one function reused and modified slightly to perform another useful function. An example of this would be the addition of the ability to remove a customer. The code existed to remove a container and was easily repurposed to remove a customer. Again, although this turned into an unanticipated requirement, it could be reasonably anticipated that something like this would occur.

The first requirement that falls into the second category of improving the functionality of the program is the ability to remove a customer. This was basically an oversight in the planning stage of the project. It is reasonable to assume that at some point a customer who discontinues service should be removed from the system. This feature was added as soon as it became apparent that it was a useful feature and the code existing to remove a location could be repurposed for this function.

The next requirement that was not stated in the planning of the project was the ability to look up a customer by any substring of the customer name. Anyone who has looked up a customer in any system can probably understand this issue. In addition to the example in relation to law offices stated previously, there are also occasions when one person may enter a customer as John Doe and another may enter that same customer as Doe, John. Sure a standard should be in place, but it is always good to know that you can make sure you can find the customer if they exist in the system.

The functions related to work orders changed considerably. However, the net result is almost the same, the requirements change pertained more to structure rather than any change to data collected or displayed. Originally, there were four planned functions of Create Work Order, Add Customer to Work Order, Add containers to Work Order, and Display Work Order to screen. Instead, all the functionality originally intended with these four functions was accomplished with one function and the additional function to look up a work order was added. The function to create a work order includes prompts to add the customer as well as the containers. The reason for this is that a work order is meaningless without either of these things. Additionally, when a work order is created, the process of confirming that the work order has been created includes printing the completed work order to the screen.

The final requirement change was that the main menu was not created at the conclusion of all other development work. Instead, it was created and added to as each function in the object manager classes was complete. This was just easier to do conceptually. It was easier as a developer – as it would be for a user – to control the application from one interface. Having a consolidated interface also assured that such an interface would ultimately be completed.

Reasons for Change

All of the changes in requirements from the proposal to the current iteration were a result of seeing things that were obviously better in the changed way versus the proposed way. In terms of the added features, as previously stated, many came as a result of reusing required code designed to perform a function on one object such as a location and using it to manipulate a different object such as a customer. The change in requirements in relation to the work order interface was also the result of being able to accomplish three items with one function.

Actual and Potential Effects

Before this solution can be implemented, at minimum a web interface needs to be completed. As such, it is not deployed yet. The potential effects the solution could have on the organization include an additional revenue source. Because future iterations will include multi-tenant and self-provisioning capabilities, the solution can be marketed to individuals or businesses maintaining their own inventory of records or even to other records management service providers.

Conclusion

In terms of project management, this project was successful in each of the three most important metrics. The scope was clear and concise so that there was no guessing as to what deliverables were required. This allowed the program to meet all of the stated requirements with minimal scope creep. The minor scope creep that did occur did not have any effect on the schedule. In terms of the schedule, the project finished three days ahead of schedule. This represents efficient use of time without being so far ahead of schedule as to call into question the estimation of time each task was supposed to take. As for budget, the estimated budget was zero and that was also the actual budget. All technologies are free. The next step which includes cloud hosting will require an actual budget. But to date, scope, schedule and budget are all met.

References

Alston, R. (2015). About Us. Retrieved September 15, 2015, from http://informationprotected.com/about-access/

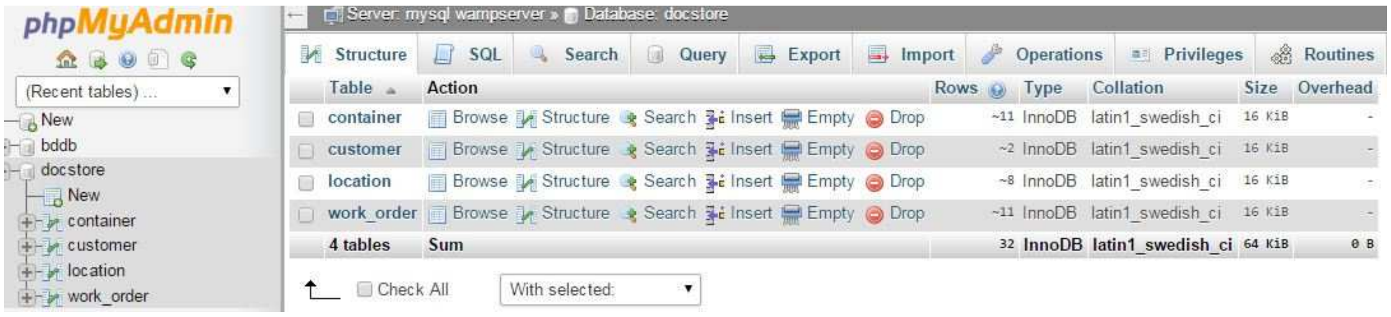
Weise, C. (2012). How to Achieve Best Practices: Records Management. Retrieved 2015, from http://www.aiim.org/Research-and-Publications/Research/How-To-Guides/Step-by-step-to-records-management-best-practices

TAB, (2015). "5 Tips for Selecting and Implementing RM Software in the Hybrid Environment." *TAB Fusion RMS*. Web. Retrieved 2015 from http://fusionrms.tab.com/resources/hybrid-records-management/guide/5-tips-selecting-implementing-rm-software-hybrid-environment/?tab_k=TAB-Fusion-GD_Selecting_Software_for_Hybrid_Environment-Web

IBISWorld, (2015) "Document Management Services in the US: Market Research Report." *IBISWorld*. IBISWorld, 1 Apr. 2015. Web. 2015. http://www.ibisworld.com/industry/document-management-services.html

Firebird SQL, (2012) “Case Studies: Document Management Solutions for Businesses” Retrieved January 10, 2016, from http://www.firebirdsql.org/en/case-studies-catalog/document-management-solutions-for-businesses-9892/

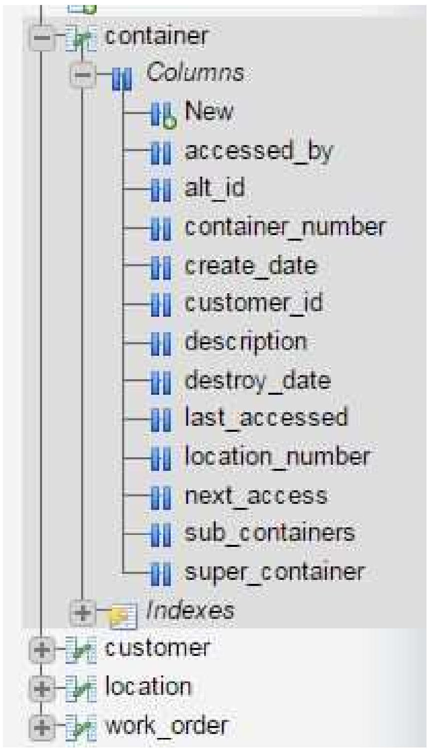
Appendix 1: Database



Server: mysql wampserver » Database: docstore

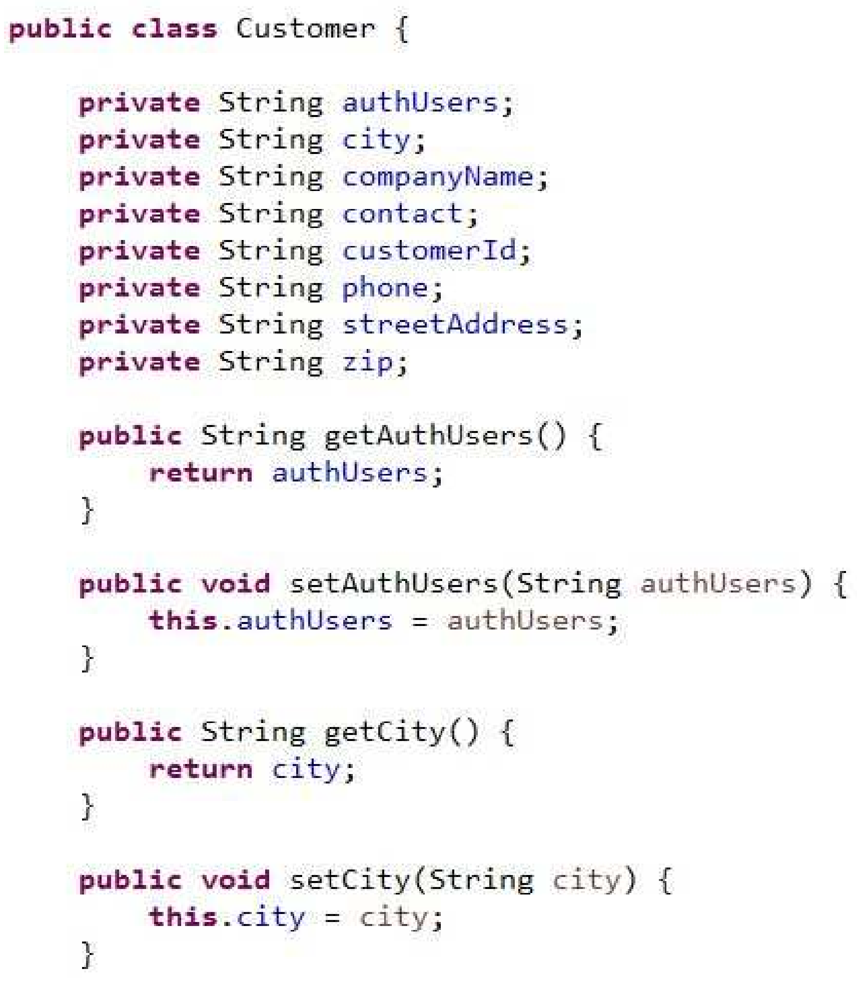
Table	Action	Rows	Type	Collation	Size	Overhead
container	Browse Structure Search Insert Empty Drop	~11	InnoDB	latin1_swedish_ci	16 KiB	-
customer	Browse Structure Search Insert Empty Drop	~2	InnoDB	latin1_swedish_ci	16 KiB	-
location	Browse Structure Search Insert Empty Drop	~8	InnoDB	latin1_swedish_ci	16 KiB	-
work_order	Browse Structure Search Insert Empty Drop	~11	InnoDB	latin1_swedish_ci	16 KiB	-
4 tables	Sum	32	InnoDB	latin1_swedish_ci	64 KiB	0 B

Appendix 2: Container DB Table


container
Columns
New
accessed_by
alt_id
container_number
create_date
customer_id
description
destroy_date
last_accessed
location_number
next_access
sub_containers
super_container
Indexes
customer
location
work_order

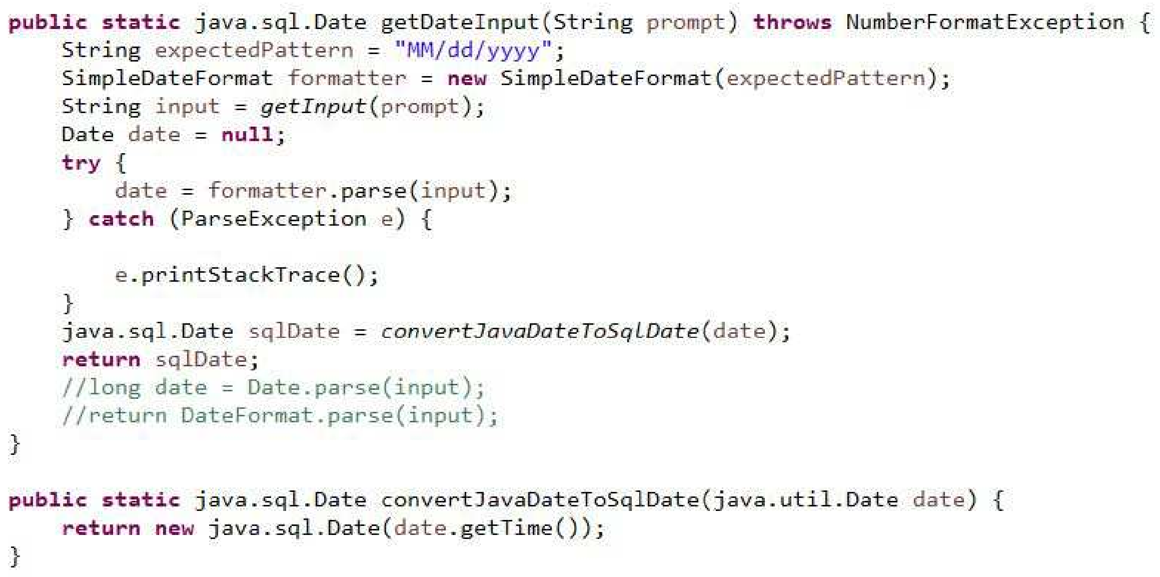
Appendix 3: Customer “bean” class

(showing first two getters and setters)



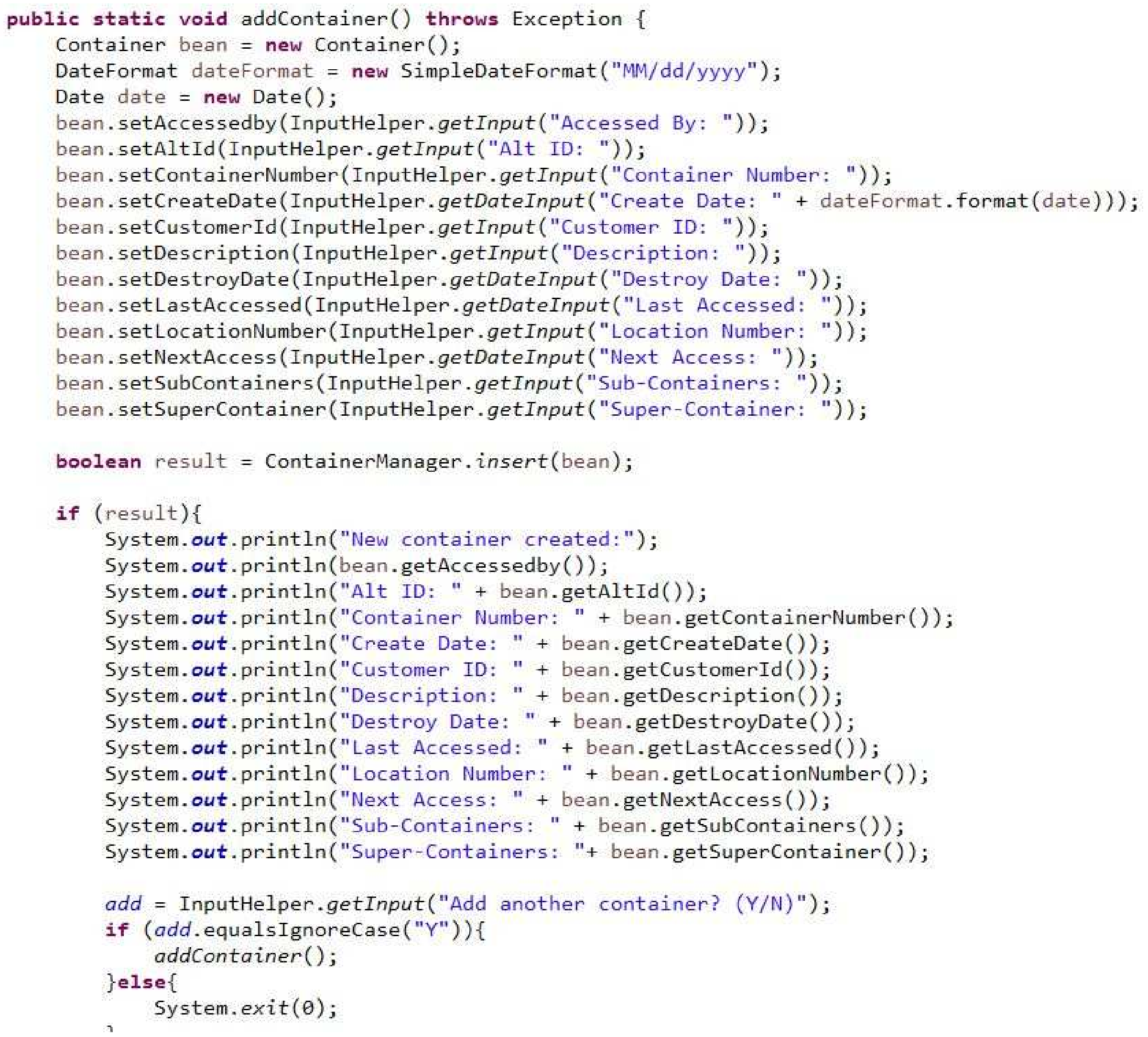
```
public class Customer {

    private String authUsers;
    private String city;
    private String companyName;
    private String contact;
    private String customerId;
    private String phone;
    private String streetAddress;
    private String zip;

    public String getAuthUsers() {
        return authUsers;
    }

    public void setAuthUsers(String authUsers) {
        this.authUsers = authUsers;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }
```

Appendix 4: Date Conversion Method



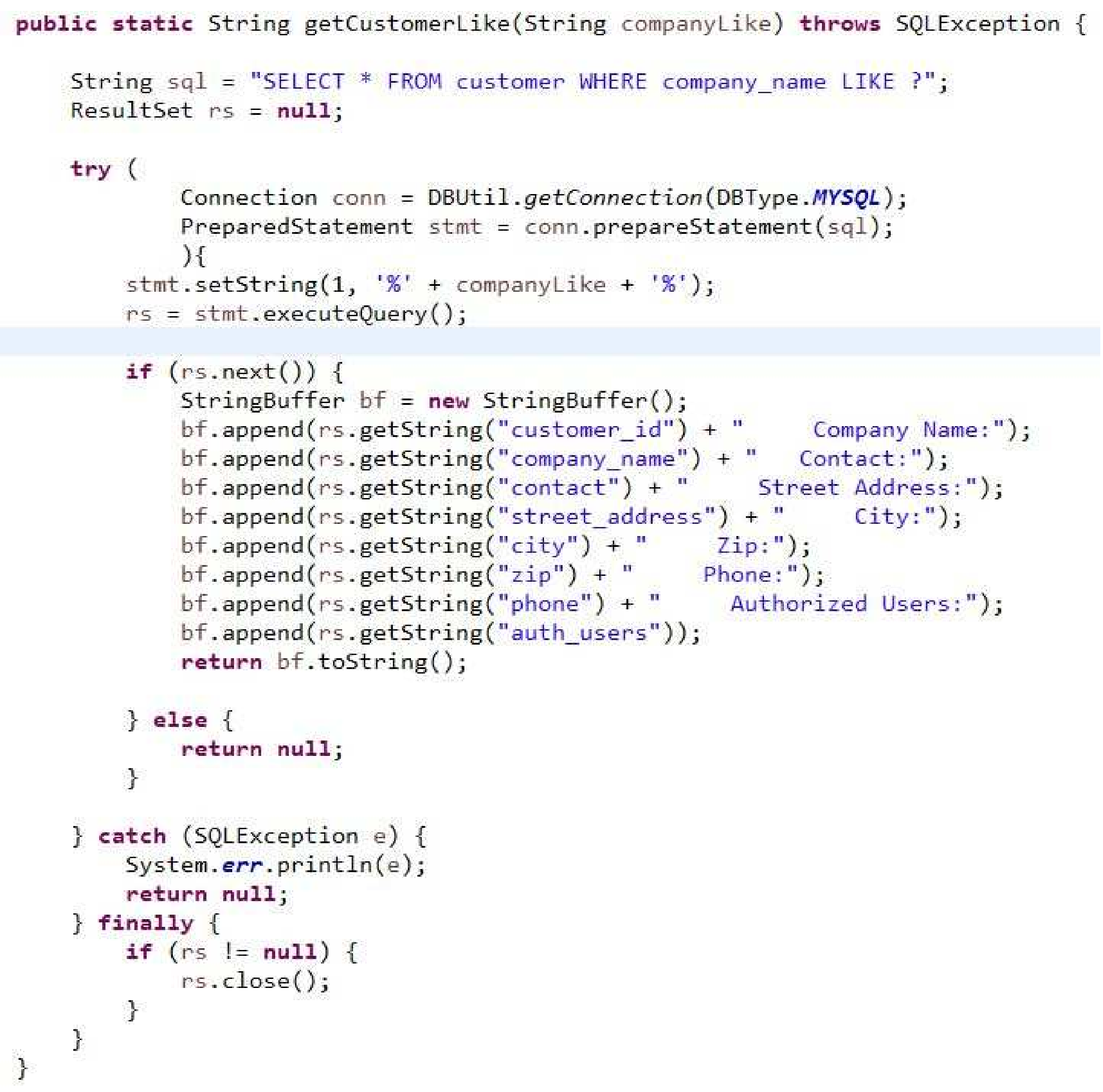
```java
public static java.sql.Date getDateInput(String prompt) throws NumberFormatException {
    String expectedPattern = "MM/dd/yyyy";
    SimpleDateFormat formatter = new SimpleDateFormat(expectedPattern);
    String input = getInput(prompt);
    Date date = null;
    try {
        date = formatter.parse(input);
    } catch (ParseException e) {

        e.printStackTrace();
    }
    java.sql.Date sqlDate = convertJavaDateToSqlDate(date);
    return sqlDate;
    //long date = Date.parse(input);
    //return DateFormat.parse(input);
}

public static java.sql.Date convertJavaDateToSqlDate(java.util.Date date) {
    return new java.sql.Date(date.getTime());
}
```

Appendix 5: addContainer Method



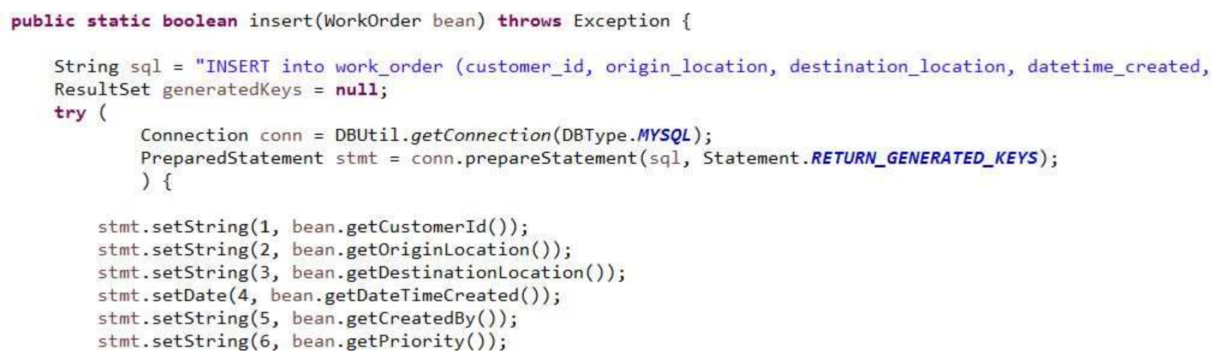
```
public static void addContainer() throws Exception {
    Container bean = new Container();
    DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");
    Date date = new Date();
    bean.setAccessedby(InputHelper.getInput("Accessed By: "));
    bean.setAltId(InputHelper.getInput("Alt ID: "));
    bean.setContainerNumber(InputHelper.getInput("Container Number: "));
    bean.setCreateDate(InputHelper.getDateInput("Create Date: " + dateFormat.format(date)));
    bean.setCustomerId(InputHelper.getInput("Customer ID: "));
    bean.setDescription(InputHelper.getInput("Description: "));
    bean.setDestroyDate(InputHelper.getDateInput("Destroy Date: "));
    bean.setLastAccessed(InputHelper.getDateInput("Last Accessed: "));
    bean.setLocationNumber(InputHelper.getInput("Location Number: "));
    bean.setNextAccess(InputHelper.getDateInput("Next Access: "));
    bean.setSubContainers(InputHelper.getInput("Sub-Containers: "));
    bean.setSuperContainer(InputHelper.getInput("Super-Container: "));

    boolean result = ContainerManager.insert(bean);

    if (result){
        System.out.println("New container created:");
        System.out.println(bean.getAccessedby());
        System.out.println("Alt ID: " + bean.getAltId());
        System.out.println("Container Number: " + bean.getContainerNumber());
        System.out.println("Create Date: " + bean.getCreateDate());
        System.out.println("Customer ID: " + bean.getCustomerId());
        System.out.println("Description: " + bean.getDescription());
        System.out.println("Destroy Date: " + bean.getDestroyDate());
        System.out.println("Last Accessed: " + bean.getLastAccessed());
        System.out.println("Location Number: " + bean.getLocationNumber());
        System.out.println("Next Access: " + bean.getNextAccess());
        System.out.println("Sub-Containers: " + bean.getSubContainers());
        System.out.println("Super-Containers: "+ bean.getSuperContainer());

        add = InputHelper.getInput("Add another container? (Y/N)");
        if (add.equalsIgnoreCase("Y")){
            addContainer();
        }else{
            System.exit(0);
```

Appendix 6: getCustomerLike Method



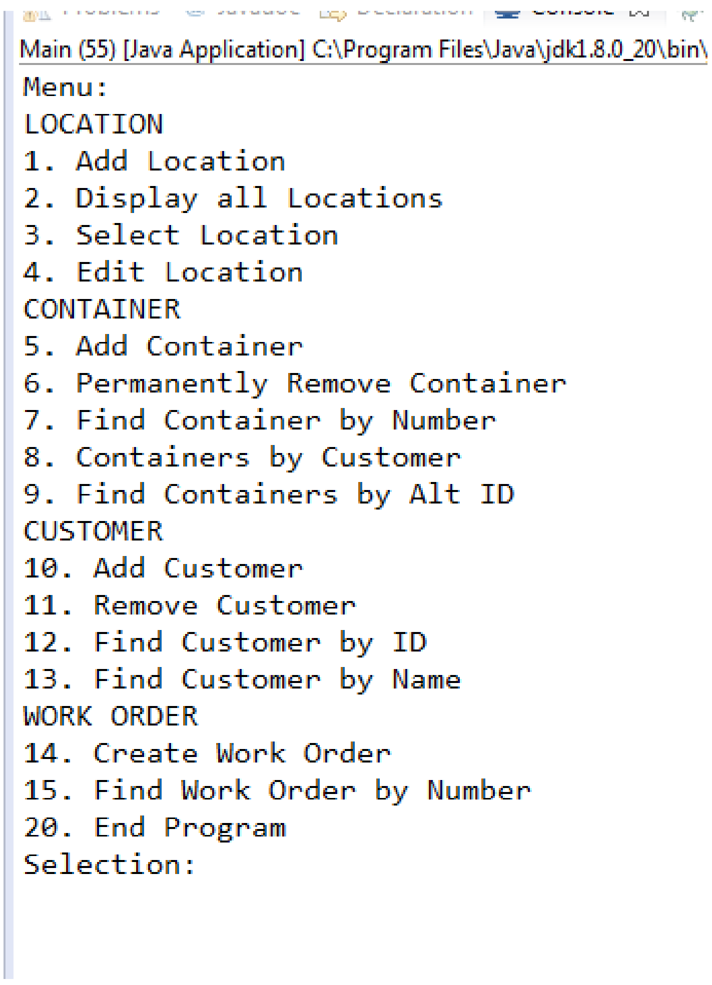
```
public static String getCustomerLike(String companyLike) throws SQLException {

    String sql = "SELECT * FROM customer WHERE company_name LIKE ?";
    ResultSet rs = null;

    try (
            Connection conn = DBUtil.getConnection(DBType.MYSQL);
            PreparedStatement stmt = conn.prepareStatement(sql);
            ){
        stmt.setString(1, '%' + companyLike + '%');
        rs = stmt.executeQuery();

        if (rs.next()) {
            StringBuffer bf = new StringBuffer();
            bf.append(rs.getString("customer_id") + "     Company Name:");
            bf.append(rs.getString("company_name") + "    Contact:");
            bf.append(rs.getString("contact") + "     Street Address:");
            bf.append(rs.getString("street_address") + "     City:");
            bf.append(rs.getString("city") + "     Zip:");
            bf.append(rs.getString("zip") + "     Phone:");
            bf.append(rs.getString("phone") + "     Authorized Users:");
            bf.append(rs.getString("auth_users"));
            return bf.toString();

        } else {
            return null;
        }

    } catch (SQLException e) {
        System.err.println(e);
        return null;
    } finally {
        if (rs != null) {
            rs.close();
        }
    }
}
```

Appendix 7: Return Generated Keys



```
public static boolean insert(WorkOrder bean) throws Exception {

    String sql = "INSERT into work_order (customer_id, origin_location, destination_location, datetime_created,
    ResultSet generatedKeys = null;
    try (
            Connection conn = DBUtil.getConnection(DBType.MYSQL);
            PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            ) {

        stmt.setString(1, bean.getCustomerId());
        stmt.setString(2, bean.getOriginLocation());
        stmt.setString(3, bean.getDestinationLocation());
        stmt.setDate(4, bean.getDateTimeCreated());
        stmt.setString(5, bean.getCreatedBy());
        stmt.setString(6, bean.getPriority());
```

Appendix 8: Main Menu



```
Main (55) [Java Application] C:\Program Files\Java\jdk1.8.0_20\bin\
Menu:
LOCATION
1. Add Location
2. Display all Locations
3. Select Location
4. Edit Location
CONTAINER
5. Add Container
6. Permanently Remove Container
7. Find Container by Number
8. Containers by Customer
9. Find Containers by Alt ID
CUSTOMER
10. Add Customer
11. Remove Customer
12. Find Customer by ID
13. Find Customer by Name
WORK ORDER
14. Create Work Order
15. Find Work Order by Number
20. End Program
Selection:
```