Project 1 Final Report

This software implements a system that tracks the actions performed by a warehouse. This warehouse contains a list of products that the warehouse carries, a list of manufacturers that supply products, and a list of clients that order products from the warehouse. Along with storing this information, a user can perform several actions to interact with the warehouse, such as placing orders with the manufacturer and accepting payment from a client.

The software is stored on the csci2 machine under the account cs430118. It is located in the directory /Desktop/Project1 and can be compiled with the java compiler with the command: “javac warehouse.java”. To run the program, enter: “java warehouse”.

The work was equally shared among all members. Tasks were distributed equally during group meetings. It was tough to meet outside of the set meeting time since we all have different time availabilities. To make up for this, we were able to set up and SVN where we could each work on the project on our own time.

The program consists of 16 files, ranging from Client, Manufacturer, and Product classes, to the User Interface and Warehouse implementations.

The Warehouse class is how the Client, Manufacturer, and Product become a part of the Warehouse system. This class was authored by Emilio Rescigno. It was the first class included in the system in the conceptual stage. The Warehouse contains a Product List, Client List, and Manufacturer List. The Warehouse initializes each list as well as initializing the Server ID’s. The methods within Warehouse include adding a product to the warehouse, adding a client to the warehouse, adding a manufacturer to the warehouse. There is a method to add Supplier Terms to the Warehouse. In this function, this is a list of manufacturers that produce a given product, and a list of products that are offered by a given manufacturer so that it is easy for a user to either search by product or search by manufacturer. There is also a Remove Terms method that will unlink a product and a manufacturer and remove them from their respective lists. A New Client Order function is used to allow a client to order a product from the warehouse. A New Manufacturer Order method is used to allow the warehouse to order a product from a manufacturer. There are Show All Clients/Client Orders/Manufacturers/Manufacturer Orders/ Products methods that display the list of all the respective entities that are a part of the Warehouse system. A Receive Shipment method is implemented to allow the warehouse to receive a shipment from a manufacturer. This function searches the system for the given shipment and updates the current stock of products within the shipment. There is also a check to see if there are any backorders. That is, if a client ordered a product that was not in stock at the time, the system will notify the user when the shipment is received and prompt the user to fulfill the back order.

The Client class is where client information is stored. This class was authored by Joseph Donabauer. It was one of the first classes included in the system in the conceptual stage. The Client is related to the Products class since the client will want to order products. A Client contains a name, address, phone number, and ID number. It also contains a list of pending orders, filled orders, and back orders which are used for tracking a client’s purchase. The methods within Client include setting and retrieving a Client’s name, phone, address, and ID number as well as functions to place an order, find pending order, remove pending order, remove back order, add filled order, and add back order to keep track of and fulfilling a client purchase.

The Client List class is a collection of Clients. This class was authored by Kunga Sherpa. It was included in the system at the same time as Client since it is directly related to Client. It contains a list of Clients and includes methods to add a client, search for client, and display the clients.

The Product class is where product information is stored. This class was authored by Joseph Donabauer. It was one of the first classes included in the system in the conceptual stage. The Product is related to the Client class since the client will want to order products and also the Manufacturer class since manufacturers produce products. A Product contains a name, description, ID number, sell price, and quantity. It also contains a list of supplier terms, since each manufacturer may supply a product to the warehouse at a different cost, and a list of Client orders. The methods within Product include setting and retrieving a Product name, description, sell price, quantity, and ID number as well as functions to add supplier terms, remove supplier terms, place a client order, update the inventory, and retrieve the list of back orders.

The Product List class is a collection of Products. This class was authored by Kunga Sherpa. It was included in the system at the same time as Product since it is directly related to Product. It contains a list of Products and includes methods to add a product, search for product, and display the products.

The Manufacturer class is where manufacturer information is stored. This class was authored by Joseph Donabauer. It was one of the first classes included in the system in the conceptual stage. The Manufacturer is related to Product class since manufacturers produce products. A Manufacturer contains a name and ID number as well as a list of supplier terms, since each manufacturer may supply a product to the warehouse at a different cost, a list of pending orders which contain a list of products the Warehouse has requested, and a list of filled orders. The methods within Manufacturer include setting and retrieving a Manufacturer name and ID number as well as functions to add supplier terms, remove supplier terms, place a manufacturer order, fill a manufacturer order, find a pending order, and remove a pending order.

The Manufacturer List class is a collection of Manufacturer. This class was authored by Kunga Sherpa. It was included in the system at the same time as Manufacturer since it is directly related to Manufacturer. It contains a list of Manufacturer and includes methods to add a manufacturer, search for manufacturer, and display the manufacturers.

The Supplier Terms class is how a Manufacturer and Product can be linked. This class was authored by Jordan Musselman. It was included in the system in the conceptual stage as a byproduct of the link between Manufacturer and Product. The Supplier Terms contain a Manufacturer, a Product, and a supply cost which is the price the Manufacturer supplies the Product to the Warehouse for. The methods within Supplier Terms include setting the Supplier Terms from a Manufacturer for a Produce at a given price, getting the product, getting the Manufacturer, and displaying the terms.

The User Interface is how the user interacts with the system. This class was authored by Jordan Musselman. It was included in the system during the implementation stage. It is not in any of our relationship diagrams, but was need to be able to use the program. The class consists of a menu with a list of options that is displayed to the user and user selects an option to carry out. Once an option is chosen, there is a function call to the relevant classes so the action can be performed.

There are 5 ID Server classes used that generate and keep track of ID numbers for their respective classes. These include the following classes: ClientIdServer, ClientOrderIdServer, ManufacturerIdServer, ManufacturerOrderIdServer, and ProductIdServer. They all function the same way and they are almost identical in code, except they are tailored to the class they are use for. These classes were authored by Emilio Rescigno. These classes were not included in the system right away. Originally, the user was asked to input an ID number for each respective class. However, we realized there was a need for a function to automatically generate an ID number for each class, so they were added in the implementation stage.

Add Client Sequence Diagram:



Add Client Use Case:

|  |  |
| --- | --- |
| 1. Receive application from a client |  |
| 1. Worker enters 1 to the system |  |
|  | 1. System asks for information on the client: name, address, etc. |
| 1. The worker fills out the requested data fields |  |
|  | 1. The system saves the data entered by the worker and generates a new unique id for that client. |

Add Client Description:

This case was authored by Joseph Donabauer. There were not really any problems encountered during implementation of our add client class, it was very similar to the add member example from the library example so most of the design choices were taken from the example and modified to fit our uses.

Place an order with a Manufacturer Sequence Diagram:



Place an order with a Manufacturer Use Case:

|  |  |
| --- | --- |
| 1. The clerk initiates the system to place an order |  |
|  | 1. The system ask for order details |
| 1. The clerk enters product details which include manufacture name, productId, orderId and quantity. |  |
|  | 1. The system then checks the order and make sure that the order placed is the right one. If the order placed is right one then the system provides again provides the details including the price to the clerk to finalize the order |
| 1. The clerk finalized the order and place the order |  |
|  | 1. The system accept the order and records the transactions |
| 1. Clerk exit the system |  |

Place an order with a Manufacturer Description:

This case was authored by Emilio Rescigno. To start, the place manufacturer order option is selected in the user interface. That in turn collects the manufacturer ID, product ID, and product quantity. With these three pieces of information, a function is called inside the warehouse class with these three parameters. This function then retrieves the corresponding Product and Manufacturer objects based on their IDs, and creates a new ManufacturerOrder object. This new object is then placed in lists inside both the corresponding manufacturer and product.

The most difficult part when implementing this function was ensuring that the lists inside both the Manufacturer and Product classes were updated properly and that data did not get out of sync. Filling manufacturer orders was many degrees more complex.

Receive an order from a Client Sequence Diagram:



Receive an order from a Client Use Case:

|  |  |
| --- | --- |
| 1. Worker receives an order from a client (customer) 2. Worker presses the “place order” button on the system |  |
|  | 1. The system asks for the client ID |
| 1. The worker enters the client ID |  |
|  | 1. The system asks for details of the order (list of products and number of each one) |
| 1. The worker enters the items and quantity of each item |  |
|  | 1. The system checks product availability and determines if the order can be fulfilled. If it can be fulfilled, subtract quantity from inventory and send a packing list to the warehouse floor.   If the request is unable to be fulfilled, subtract remaining balance from inventory, send request to the warehouse floor to be packed, and place remainder of order on backorder.  Display results of the order to the worker. |

Receive an order from a Client Description:

This case was authored by Jordan Musselman. It is very straight forward and there wasn’t many issues in implementing this case. If the user wants to perform this command, they select the option to place an order and then enter all relevant information. The system needs to use the given information to search the warehouse for a particular product and associate it with the client. There are checks to see if the products are in stock and if not, may need to be placed on backorder but this problem was not to difficult to implement.

Receive payment from a Client Sequence Diagram:



Receive payment from a Client Use Case:

|  |  |
| --- | --- |
| 1. The clerk initiates system to accept payment from the customer. |  |
|  | 1. The system ask for details |
| 1. The clerk enters the customer id and order id |  |
|  | 1. The system checks the order and shows the order detail with amount due to the clerk |
| 1. The clerk enters the amount received from the customer and the method of payment |  |
|  | 1. The system deducts the amount paid from the due amount and shows the remaining balance to the clerk. |
| 1. The clerk checks the receipt and finalize the payment |  |
|  | 1. The system records the transactions |
| 1. The clerk exits the system. |  |

Receive payment from a Description:

This case was authored by Kunga Sherpa. The use case and the sequence diagram for receiving payment from a client do not completely resemble with each other. However the core things of both are same. For the use case it is simply an interaction between the system and the users. At each level user enters certain information and the system responds to the request accordingly. I did not face any difficulty in writing the use case. For the sequence diagram every class is labeled in a rectangular box above the vertical line. The vertical line itself represents the class and the arrowhead represents the process. The vertical rectangular box represents the activeness of the class i.e. the class becomes active at the start of the rectangular box and become inactive at the end of rectangular box. Writing the exact process for each class and creating interaction among the different classes were the difficult part while creating the sequence diagram.

Individual Reports

Joseph Donabauer:

Opinion of the project – I feel like this was a good project to tackle, I have worked at a warehouse in the past so I understand that it is a real life application and different parts of the project that are required and nice to have. When the group was able to meet I feel that we worked quite well together, however the times we are available do not work very well so it is difficult to get together. I feel that I have learned a bit about java during this program and more information about objects and how they are used to organize a project more elegantly. The finished product seems to work well, there are some cases where if invalid input is entered strange things may happen but I’m not sure. I feel like I’d be able to work on this type of projects in the future.

Group Dynamic - the days that we planned to meet were: Feb 6, Feb 13, Feb 20, Feb 27, and March 6. I missed the meeting on the 27th due to a family obligation however there were several other informal meetings that I showed up to.

1. I feel that the group worked well whenever we were able to meet together
2. I feel that all members tried to do whatever they could to complete the project and I feel that Emilio R went out of his way to be an all-star on the team.
3. The communion wasn’t the greatest, were all had each other’s phone numbers but should do a better job replying to emails in a timely manner.
4. I think our project more forward just fine however it was helpful to have the due date pushed back a few days.

Kunga Sherpa:

In this project I learned a great deal of stuff about object oriented programming. The text book itself was immensely helpful in understanding the codes and the whole project. Design patterns, serialization, object collections were the ones that I learned most about the project. There were several projects that I have pursued in my previous semester in java. And turning back I felt I could have done that project more effectively and quickly had I implemented the things I learned in this project. Moreover working with a new group was also an extra experience for me. Since most of our team members do not a common schedule to meet, we kept in contact and divided the work load equally.