Diego Santosuosso - 217339987
EECS3311 Project

PROJECT REPORT

In order to implement the VideoCo Management System and provide a new computer software to allow customers to rent movies, two major design patterns were used to ensure that the system met all of its requirements. These design patterns were the Singleton and Observer patterns.

1. Singleton Design Pattern

- The Singleton pattern ensures that an object is the only one of its type providing a global access point to that instance.
- In the VideoCo system, the Singleton pattern was used in many of its classes, mainly representing databases that should only have a single instance. Some of these Singleton classes are: Inventory, CustomerDatabase, VideoCo, and Warehouse.
- In order to save memory and avoid bugs from arising from multiple instances, the databases of the system are singleton classes. This means that the VideoCo system will have only one inventory, one warehouse and one centralized customer and employee database.
- This does not mean, however, that new items cannot be added or removed from these databases. It just means that there are central and single databases that keep track of the available and unavailable movies, of the customers of the company, of the employees of the company, and of the movies that were sent to the warehouse.

2. Observer Design Pattern

- The observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This design pattern contains the two main components - the Subject and the Observer.
- In the VideoCo System, whenever an order is delivered by the delivery system, it should notify the customer and automatically update the order status. As a result, the Observer pattern can be used to model this scenario.
- Instead of the person constantly polling the delivery system and asking for an order status update, the delivery system will automatically notify the person about an update and update the status of the order.
- In this case, the Order will be the subject (because it will be constantly changing and updating its status), and the PastOrderGUI class will be the observer (because it depends on the delivery status and updates according to it). A customer views his past orders and checks the order status based on the PastOrderGUI frame.
- The concrete subject, which is the Order, implements the Observable Interface.
- Similarly, the concrete observer, which is the PastOrderGUI, implements the Observer Interface.

Please find attached a copy of my final Class Diagram in the package of this project submission. Refer to the diagram throughout this report.

You have finished an initial design for the project in the midterm exam. During the implementation of the project, what are the problems you found in your initial design?

The first step to create my VideoCo System was to create a Class Diagram. I performed this task for my EECS3311 Midterm. However, after implementing the whole system I realized that there were numerous problems with the initial class diagram design.

The first and major issue was that I did not include the GUI interfaces in the Class Diagrams. GUIs played a significant role in the implementation of this project, and since GUIs are technically classes, they should be included in a program's class diagram. I didn't foresee GUIs when I first created the class diagram for the VideoCo System.

One of the second issues is that I did not set up the Warehouse class as a Singleton, which could potentially allow the system to create multiple warehouse instances. This should not be the case as the system should only have a single main centralized warehouse.

Another issue from my initial Class Diagram is that I did not create databases for storing customers and employees of the company. This is extremely important to keep track of the users who, signed as customers or as system admins, will use the VideoCo System.

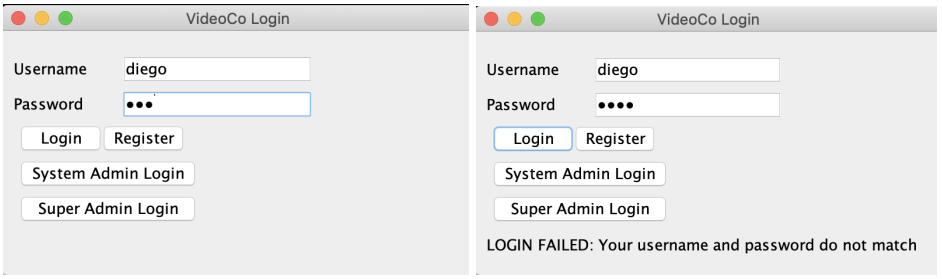
Please comment on the difference between the class diagrams of your final implementation and the initial design in the midterm (If applicable).

- There are many differences between the class diagrams of my initial and final implementation. Of course, the final implementation of the class diagram looks much more complicated and contains at least three times more classes than the initial design.
- The first difference is that the final implementation contains GUI interface classes. As was mentioned earlier, I mistakenly did not include these classes in the initial implementation of the Class Diagram.
- Secondly, there are many other Singleton Classes present - Warehouse, CustomerDatabase, VideoCo, etc.
- While I added the Singleton Pattern to many of the classes, the Observer pattern from the initial implementation stayed somewhat the same. I did change the names of the subject and observer classes, though.
- Another major change from my initial implementation is that the SearchByTitle, SearchByCategory, and SearchByAll classes do not implement the Search interface. For my final implementation, I treated each of the Search classes as separate entities. It was easier to implement this way.
- In addition, the initial implementation of my Class Diagram contained a Person interface that was implemented by the Customer and the SystemAdmin. In the actual implementation of the program, I realized that it was easier to implement them as separate entities without them implementing a common interface. As a result, I removed the Person Interface from the final implementation of my class diagram.
- The Customer, Movie and Order classes of my initial class diagram were very similar compared to my final one.
- While the initial implementation of my class diagram was extremely helpful to get me started on the project, it was altered significantly to make the program work as expected and meet all requirements.

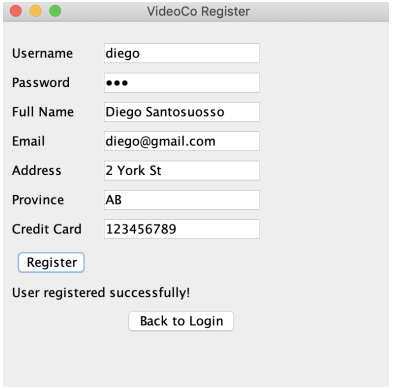
After reviewing both the initial and the final class diagrams, we can take a look at how my program meets all the software requirements. I have made a video showing how it works and I will attach screenshots to demonstrate how they work one-by-one.

REQ-1: Users must login to the system with valid and existing credentials. Invalid credentials will result in a login error.

- **IMAGE 1:** If the user clicks the "Login" button, he would login to the system (provided that he already has an account and that his login credentials are correct.
- **IMAGE 2:** If the login credentials do not match, a message will appear saying that there is a login error.

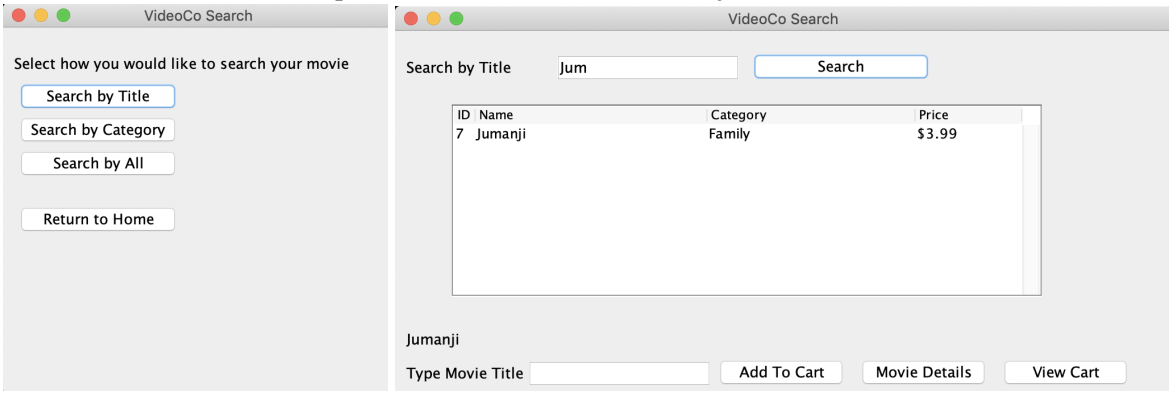


REQ-2: Users must be able to register to the system with a unique username and email. Invalid credentials will result in a registration error.



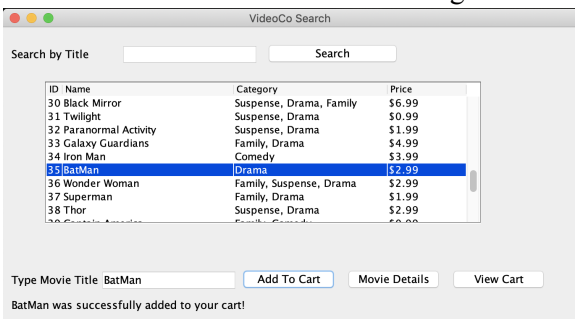
REQ-3: Users will be able to search for movies by name, category, or all movies. If no movies match the selected search, a message is displayed indicating no movies match the desired search.

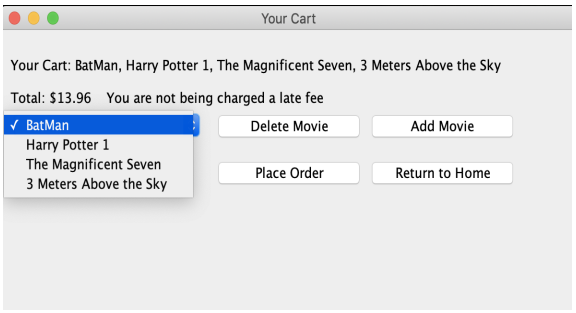
- **IMAGE 1:** User can select if he wants to search by Title, Category or all
- **IMAGE 2:** If user selects that he wants to search by Title, he can search for movies based on their names. An example is shown with movie "Jumanji", where user searched for "Jum".



REQ-4: Users must be able to create an order by adding a movie to their order. Users will not be able to add a movie which is not in stock.

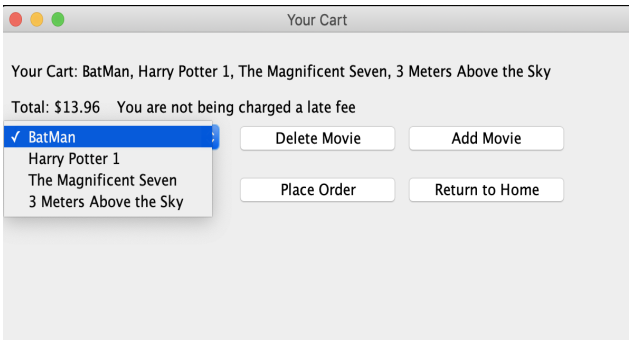
- **IMAGE 1:** Users can create an order by adding movies to their cart.
- **IMAGE 2:** User's order showing all the movies that were added to his cart.





REQ-5: Users must be able to review a tentative order, removing items from or cancelling the order as needed.

- Users can review their order, delete movies and add movies to it. They can also place their order.



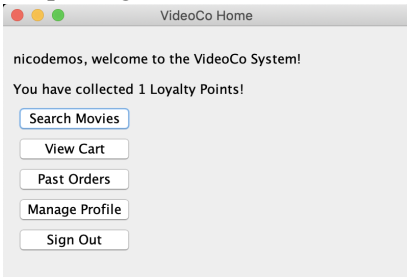
REQ-6: Users must be able to pay for an order using a payment service or loyalty points if they have enough. Invalid shipping or payment information will show the user and error respective of the invalid credential.

- In the placeOrder() function in the Customer class, if the customer has enough loyalty points (which is 10), the customer will automatically pay using his loyalty points.
- If the customer does not have enough loyalty points, then the system will automatically charge the customer's credit card.
- This requirement is also tested on the VideoCoTest file.

```
// (1.1) Charge customer (Loyalty Points)
if (this.hasEnoughLoyaltyPoints()) {
    this.loyaltyPoints = this.loyaltyPoints - 10;
    order.paidUsingLoyalty = true;
}
// (1.2) Charge customer (Credit Card)
else {
    order.paidUsingLoyalty = false;
    this.charges(order.getOrderPrice());
}
```

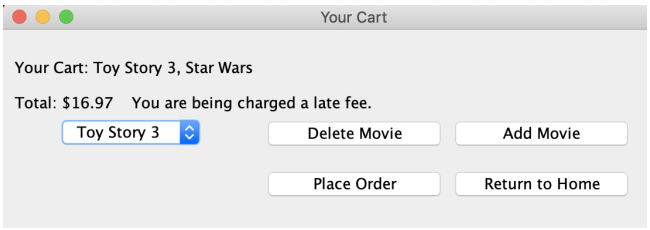
REQ-7: Users will be rewarded 1 loyalty point per order. 10 loyalty points are applicable toward a free movie rental.

- User has 1 loyalty point after placing his first order.



REQ-8: The User should be notified of a late fee based on their location. Canadian Users are charged a late fee of $9.99 if they are located outside of Ontario. The company is not global, it is Canada only.

- User is notified of a late fee because he is outside of Ontario.



REQ-9: The system will update the available movie stock according to the corresponding user action (placed order, cancelled order).

- In the deliver() method in the Warehouse class, whenever a movie is sent to a customer, it is removed from the centralized database of available movies and is placed in the out of stock movies.
- This requirement is tested in the VideoCoTest file.

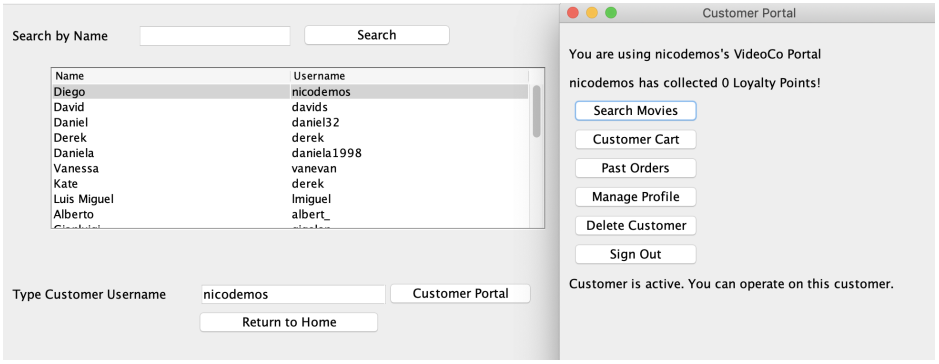
```
this.warehouse_database.remove(order);
this.delivered_database.add(order);
```

REQ-10: Users must be able to dial in an operator and place an order for their movie rentals via the operator (after providing shipping/billing address). The user will be given an unique order ID for their completed order.

REQ-11: Users must be able to dial in an operator to check the status of their order by using their unique order ID. Invalid order IDs will have no corresponding order.

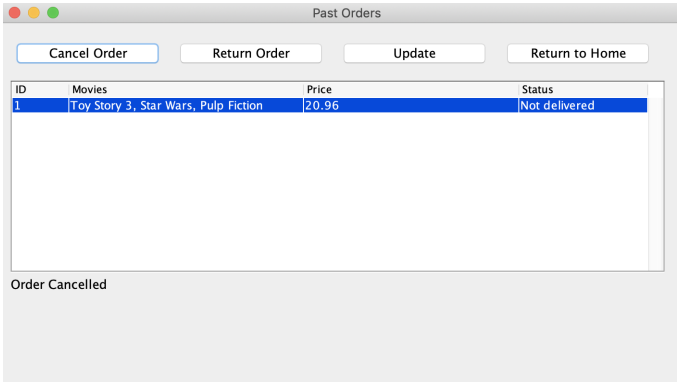
REQ-12: Users must be able to rent out available movies from the two store locations in Toronto if they choose to walk into the store.

- The System Admin is able to place orders for any customer that is registered in the VideoCo System. The image shows the SystemAdmin using the customer portal. If the system administrator were to click on Search Movies, and he added movies to a cart and placed an order, the customer would receive its order.
- The System Admin is also able to review the status of past orders for customers. If the system administrator were to click on Past Orders, he could see the customer's order history and their specific statuses.
- Similarly, if customers walk into the store, the system administrator could place orders for them by using their system administrator system features.

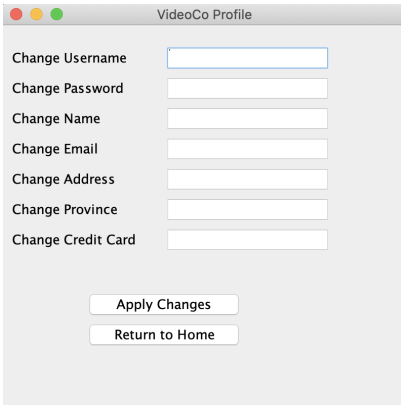


REQ-13: Users must be able to cancel their movie rental order given that the order status is not "delivered".

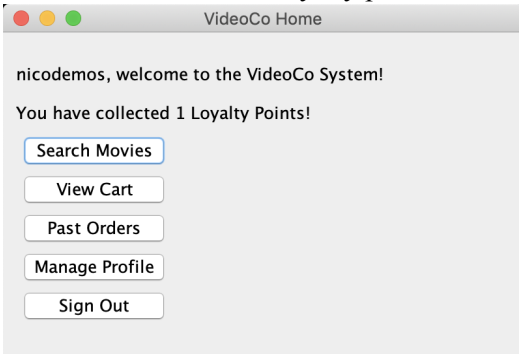
- User cancelled his order because the status was "Not delivered"



REQ-14: Users must be able to manage their account/profile. This includes changing any of the following: password, name, email.

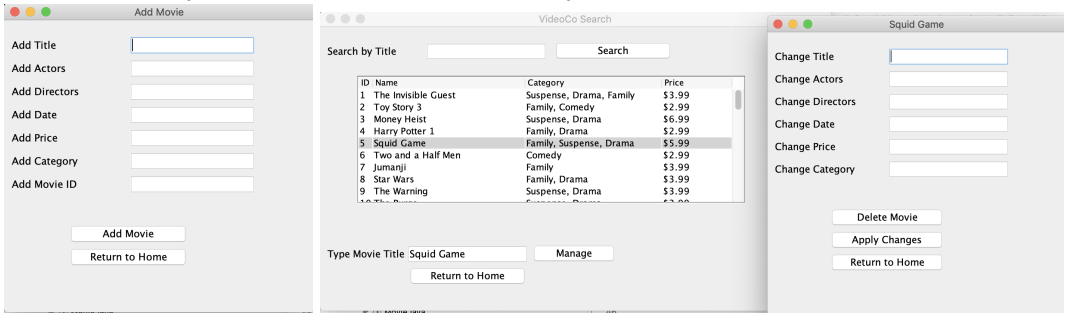


REQ-15: Users must be able to view their collected loyalty points while managing their account.



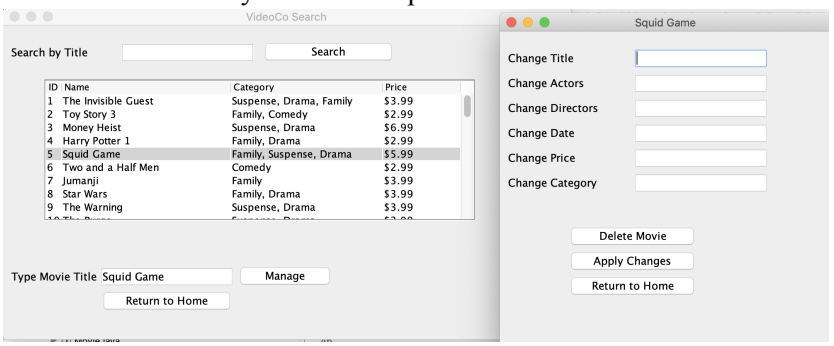
REQ-16: The System Admin is able to add/remove a movie from the system.

- **IMAGE 1:** System Admin is able to add a movie from his system admin portal.
- **IMAGE 2:** System Admin is able to select any movie and delete it.



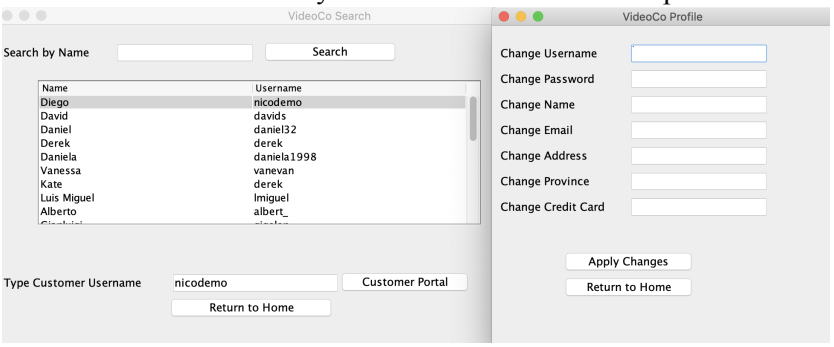
REQ-17: The System Admin is able to update movie information (title, actors, directors, date of release, description).

- System Admin can select any movie and update its information.



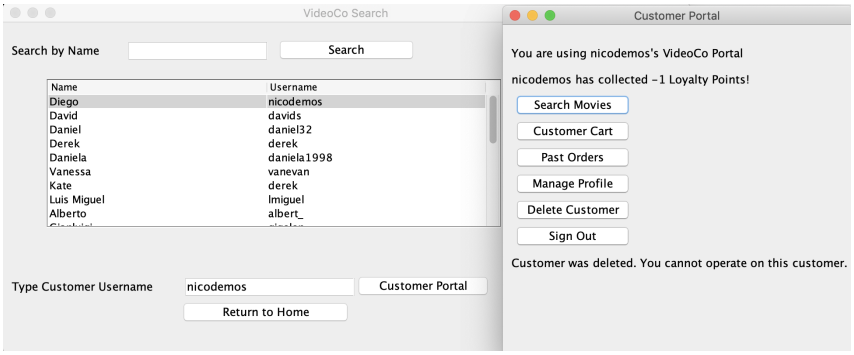
REQ-18: A system admin will be able to retrieve and update customer account information including: name, email, password, username, order information, order status.

- The System Admin can retrieve any customer information and update it accordingly.



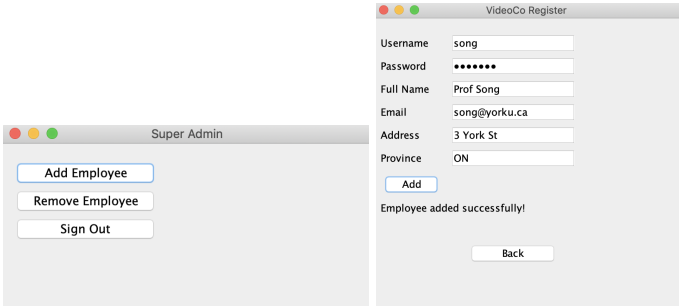
REQ-19: A system admin will be able to delete existing customer accounts.

- System Admin selected customer and deleted his account.



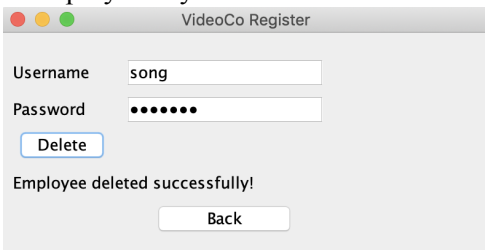
REQ-20: The system can add admin accounts to the system using the administrator's name and email address. An admin must be an employee of the company.

- SuperAdmin is able to add employees to the company using the SuperAdmin portal.



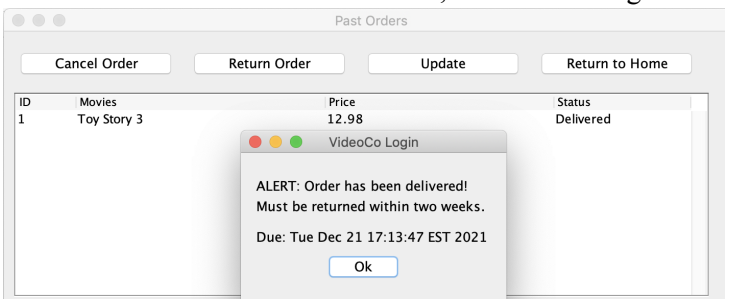
REQ-21: The system can remove admin accounts from the system.

- SuperAdmin deleted the employee's system admin account.



REQ-22: The system will be able to update the customer's order shipping status.

- System updates status. Once an order is delivered, the status changes to "Delivered".



REQ-24: The system will charge customers a late fee of $1.00 CAD per day for movies which are not returned within two weeks. The $1.00 CAD charge is a per movie charge.

- When a movie is returned, the difference between the return date and the order due date is calculated. If the difference is a negative number, this means that the customer's order is overdue and that the customer will be charged $1.00 CAD per day per movie.
- This requirement is tested in the VideoCoTest file

```
Date date1 = Calendar.getInstance().getTime();
Date date2 = selected_order.getDueDate();
double diff = date2.getTime() - date1.getTime();

Warehouse warehouse = Warehouse.getInstance();
if (selected_order != null) {
    if (warehouse.returnOrder(selected_order)) {
        cust.returnOrder(selected_order);
        if (diff >= 0) {
            success.setText("Order Returned. Thank you!");
        }
        else {

            double diff_ = Math.abs(diff);
            double diff__ = Math.floor(diff_);
            int siz = selected_order.orderList.size();

            //charge $1 per movie per late day
            cust.charges(diff__ * siz);
            success.setText("You have been charged $" + ( diff__ * siz ) + " for returning movie late");

        }
    }
    else {
        success.setText("Order cannot be returned. Contact System Administrator");
    }
}
else {
    success.setText("Please select an order.");
}
```

REQ-25: The system will send customer orders to the warehouse after they have been placed.

- In the placeOrder() function in the Customer class, once an order is placed, it will automatically be sent to the main centralized warehouse.
- This requirement is also tested on the VideoCoTest file.

```
// (2.2) Send Order to Warehouse - will deliver in 5 seconds
order.send();
order.status = "Not delivered";
```

REQ-26: The system will dispatch a list of ordered movies to be shipped to different warehouses based on the shortest geographic distance to the customer's destination address.

- In the deliver() method in the Warehouse class, whenever a movie is sent to the main warehouse, it is then sent to either WarehouseOne or WarehouseTwo, depending on which is closest to the customer's location.
- This requirement is tested in the VideoCoTest file.

```
int subwarehouse = selected_customer.getClosestWarehouse();

//sends order to closest warehouse from customer
if (subwarehouse == 1) {
    WarehouseOne warehouse1 = WarehouseOne.getInstance();
    warehouse1.addOrder(selected_order);
}
else {
    WarehouseTwo warehouse2 = WarehouseTwo.getInstance();
    warehouse2.addOrder(selected_order);
}
```

NOTE: There are two testing files for my project - VideoCoTest and SystemAdminTest.

I spent the past month working on this project and I am happy with the end result. This is my first large project on Java and I am excited about all the skills and knowledge that I have acquired. I am thankful for this opportunity and I am looking forward to applying these concepts in my future endeavours.