Author:: Najeebullah khan

Argos/Cargoss UML System design

Najeebullah khan

Contents

[*UML Assignment* 3](#_Toc76459604)

[QI 3](#_Toc76459605)

[Functional requirements 3](#_Toc76459606)

[Non-Functional requirements 4](#_Toc76459607)

[QII 5](#_Toc76459608)

[QIII 7](#_Toc76459609)

[Actors 7](#_Toc76459610)

[Customer 7](#_Toc76459611)

[Cashier 7](#_Toc76459612)

[Store person 7](#_Toc76459613)

[Collection’s receptionist 7](#_Toc76459614)

[Stock clerk 7](#_Toc76459615)

[Head office 7](#_Toc76459616)

[SMS 7](#_Toc76459617)

[Use cases 7](#_Toc76459618)

[Use case diagrams 11](#_Toc76459619)

[QIV 12](#_Toc76459620)

[Use case diagram for compile order 12](#_Toc76459621)

[Compile order use case description 12](#_Toc76459622)

[QV 13](#_Toc76459623)

[Class diagram for the entire system 13](#_Toc76459624)

[Description of classes 15](#_Toc76459625)

[QVI 17](#_Toc76459626)

[Entity classes - attributes and methods 17](#_Toc76459627)

[QVII 18](#_Toc76459628)

[Sequence diagram for compile order 18](#_Toc76459629)

[Description 19](#_Toc76459630)

[Description 19](#_Toc76459631)

[Description 20](#_Toc76459632)

[Description 21](#_Toc76459633)

[QVIII 22](#_Toc76459634)

[Compile order Communication diagram 22](#_Toc76459635)

[Description 23](#_Toc76459636)

# *UML Assignment*

# QI

## Functional requirements

Following are the functional requirements that I have managed to deduce from the Cargoss scenario:

* Update catalogue(Head office)
* Add new items
* Update items
* Remove items
* Register (customer)
* Register for corporate account
* Register for personal account
* Update account details.
* Should be able to Login (customer)
* View catalogue(customer).
* view real items
* view virtual items
* Compile order(customer)
* Add items to basket
* Remove items from basket
* Save order along with its details
* Create order
* Create order lines for items in the order
* Calculate total price for items in the order
* Update stock levels of each order item
* Process payment by credit or debit card.
* Print receipt for cash payment.

**(after payment is processed either by cashier or system)**

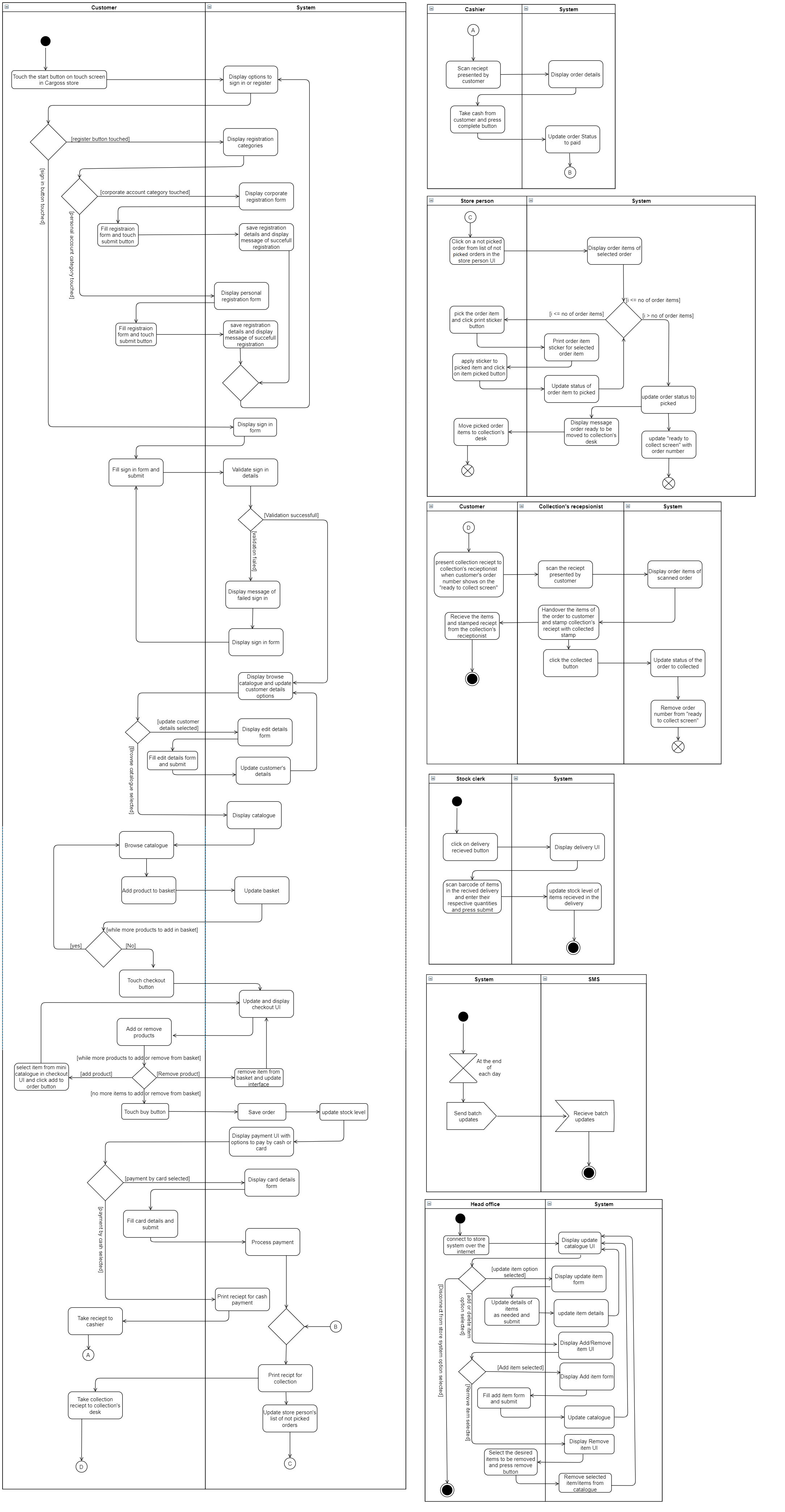
* Print receipt for collection
* Process cash payment(Cashier)
* View list of unpaid orders with payment type cash
* Register payment received from customer
* Update order status
* Manage order picking(store person)
* View list of paid orders which are not being picked by another store person
* Select order from above mention list
* View order items in selected order
* Select order items in the selected order
* Print order number sticker for selected order item
* Mark selected order item as picked
* Update status of order item to picked
* Update status of selected order to picked once all the order items are picked.
* Mark an order item as double picked
* Update stock level of double picked item
* View list of not collected orders and double picked items
* Remove the items from above mentioned list.
* Adjust stock level of removed items.
* Manage order collection(collection’s receptionist)
* View list of picked orders which are not selected by another collection’s receptionist.
* Select order from above mentioned list
* view items in selected picked order.
* Mark selected picked order as collected.
* Update selected picked order status to collected.
* Manage stock(stock clerk)
* view items and their stock levels.
* Search for item
* Update stock levels of selected items.
* Send item stock levels to SMS overnight.

## Non-Functional requirements

* Software system should be modern.
* System should be able to handle increasing scale of customer orders.
* System should provide easy to use user interface to customers without any frills.
* System should increase the customer satisfaction.
* System should computerise most of the business processes.
* System will eliminate the error of miscalculating the total price of the order by customer. System will this by calculating the order total for the customer.
* System will eliminate the 15% of cashier time correcting errors arising from customer writing a wrong product code or not realising an ordered item.
* System should reduce the errors of the customer selecting the wrong product by at least 5%
* By handling card payments system will reduce the time taken to print the collection receipt which in turn will help better manage increase customer activity by freeing the staff.
* System will reduce customer complaints by reducing the time taken to process the order.
* System should reduce queening times in the store for customers who are waiting to pay.
* Improve efficiency of order picking process.
* Minimize vagaries regarding update of item stock levels.
* Reduce the current 12% of unfulfilled orders by computerising the stock management process. So that the right stock levels are show when customer is selecting the products.
* System will eliminate most of the errors related to catalogue update by the head office.
* System should help Cargoss in marketing to its customers by storing their details and allowing them to be kept up to date by customer.
* System will better ensure availability of enough stock by computerising the update of SMS.

# QII

Following is the activity diagram I have created for the whole Cargoss system



# QIII

## Actors

Following are the actors that I have identified in Cargoss scenario along with their respective description:

### Customer

Customer can create a new personal or corporate account in the new Cargoss system along with updating their account details. Customer after login into the system can browse catalogue, add items to the basket, remove items from the basket, save order and pay for the order either by cash or by card. Once a customer have paid customer can collect the ordered products.

### Cashier

Cashier scans the payment by cash receipt from customer, accepts payment by customer and updates status of the order to paid

### Store person

Store person picks the orders paid by the customer and updates the order status to paid. Store person is also puts the uncollected and double picked items back in stock.

### Collection’s receptionist

Collection’s receptionist scans the collection’s receipt presented by customer and update the order status to collected after handing the ordered products to the customer.

### Stock clerk

Stock clerk receives the deliveries and updates the stock levels of the items received in the deliveries

### Head office

Updates the catalogue.

### SMS

Stands for stock management system. It is the old system from the Cargoss, new Cargoss system will communicate with the old SMS system over night sending it stock level updates.

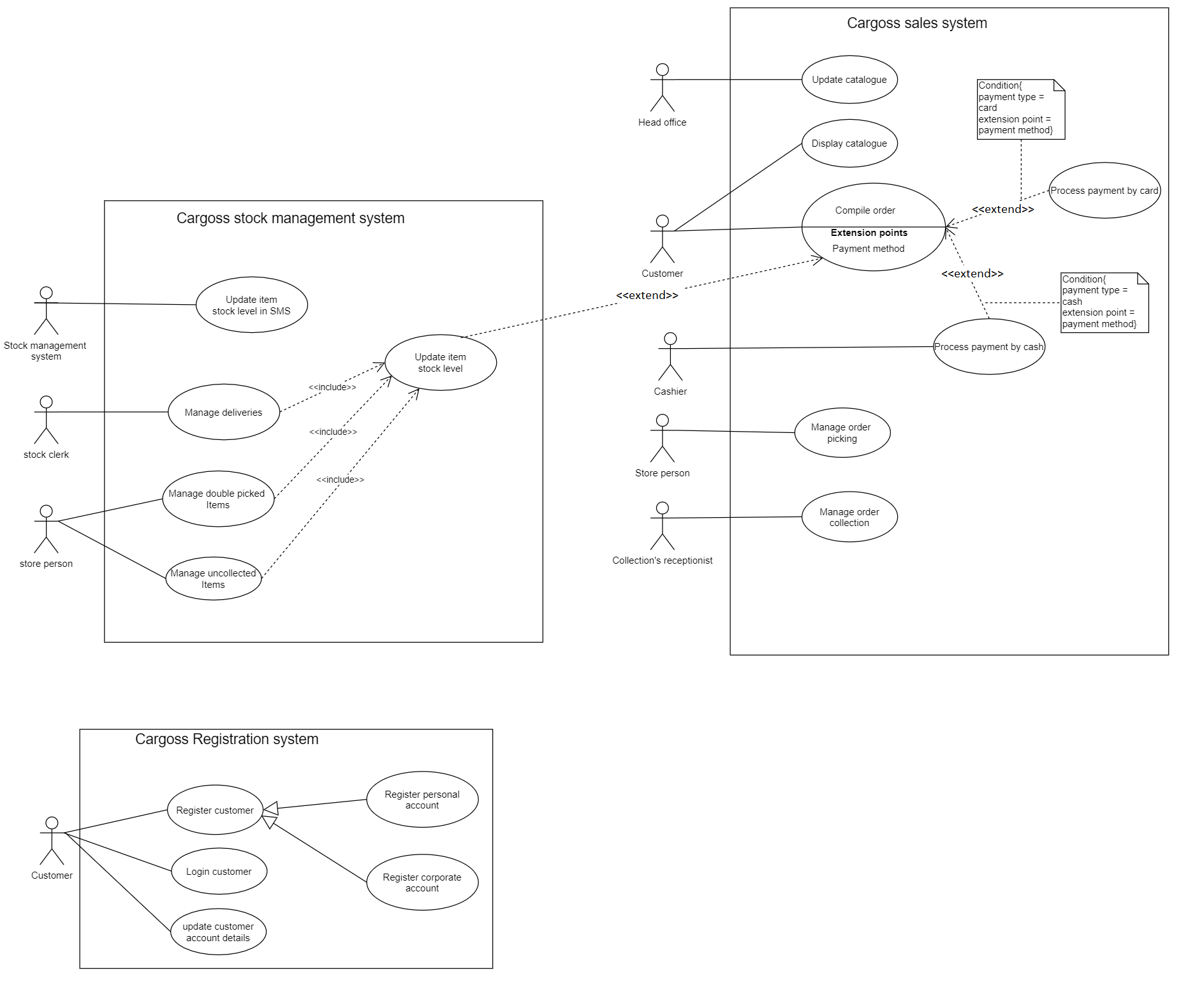
## Use cases

Following it a table that contains the use cases that I have identified in the Cargoss system along with their description:

|  |  |
| --- | --- |
| Use cases | Description |
| Register customer | **Enables the customer to create an account. only after creating an account customer can login and use the system** |
| Register personal account | **Enables the customer to register for a personal account. this use case generalizes the register customer use case. Customer selects the register personal account option and system displays personal account form. Customer then fills the form and clicks submit in response system creates a personal account for the user** |
| Register corporate account | **This use case enables customer to register for a corporate account. This use case generalizes the Register customer use case. Customer selects the corporate account option and system will display corporate account form. Customer then fills the form and clicks on the submit button system in response will create a corporate account for the customer** |
| Login customer | **Enables customer to login. Customer clicks login button and system will display login form. User then fills the form and clicks submit system in response will validate customer details. If the validation is successful system will display the customer UI. If validation fails customer system notifies customer of login failure.** |
| Update customer details | **This use case enable customer to update the account details. Customer clicks on the update details button system will display update details form. User then fills the form and clicks submit system in response will update the account details of the customer.** |
| Update catalogue | **Enables head office to update the catalogue over the internet. This use case will allow the head office to add, remove and update one or more items in the catalogue** |
| Display catalogue | **This use case enables Customer to view and browse the catalogue. Customer clicks on the browse catalogue button and the catalogue will be displayed, customer then browse the catalogue and choose the items to buy** |
| Compile order use case | **This use case enables the customer to add or remove items from the basket, save or cancel order and pay for the order either by cash or by card. A detailed description of this use case is provided in the next question.** |
| Process payment by card | **This use case allows the customer to pay for the order by card. Customer clicks on the pay by card button and system will display card details form. User then fills the form and clicks submit system will process the payment and will print receipt for collection** |
| Process payment by cash | **This use case enables customer to pay for the order by cash. Customer clicks on pay by cash button and system will print receipt for cash payment. Customer than takes the receipt to the cashier who scans the receipt. As response of the scan system will display order details belonging to the scan order. Cashier then receives the payment from the customer and updates the order status to paid. System in response updates the status of the order to paid and prints receipt for collection** |
| Manage order picking | **This use case enables the store person to pick items that are in an order and label them with that order number. This use case also allows the store person to update the order status to picked. Store person selects an order from the store person’s UI system will display order items of the selected order. User then clicks on an order item and presses print sticker button system will print the sticker of order number and update the status of the order item as picked. User upon picking and labelling of all the order items click on the order picked button the system in response will update the status of the order to picked. If an order item went missing after being updated as picked store person will click on the order item and click double pick button. System in response will update the stock level of the double picked item** |
| Manage order collection | **This use case enables the collection’s receptionist to scan the collection receipt presented by the customer and update the order status to collected. Collection’s receptionist scans the receipt presented by the customer and system displays order items of the order. Collection’s receptionist then hands over all the order items of the order to customer and after stamping the customer’s receipt with collected stamp presses the order collected button. System in response update the status of the order to collected** |
| Manage double picked items | **Enables store person to update the stock level of double picked items once it is found. Store person click on list of double picked items select the desired item and clicks on “put back to stock” button system in response will update the stock level of the item** |
| Manage uncollected items | **This use case enables the store person to update the stock level of uncollected items. Store person clicks on the list of uncollected items and system will display list of all the uncollected items. User then selects the item/items from the list that are to be put back into the stock and click “put back to stock” button in response system will update the stock levels of the select items.** |
| Update item stock level | **This use case allows the stock level of the items to be updated. This use case is always executed upon the execution of the following use case:**   * **Manage deliveries** * **Manage double picked items** * **Manage uncollected items**   **This use case extends the compile order use case.** |
| Manage deliveries | **This use case enables the stock clerk to record the received deliveries and update the stock level of the items received in the delivery.**  **Stock clerk opens the deliveries received UI and system displays user interface that allows the stock person to enter the items their quantities received in the delivery. Upon filling the required information in the user interface stock clerk clicks on the update stock level button, system in response updates the stock level of items received in the delivery** |
| Update item stock level in SMS | **This use case enables the New Cargoss system to communicate with existing stock management system(SMS) at the end of each day. This use case allows the system to send batch updates of item stock levels to SMS.** |

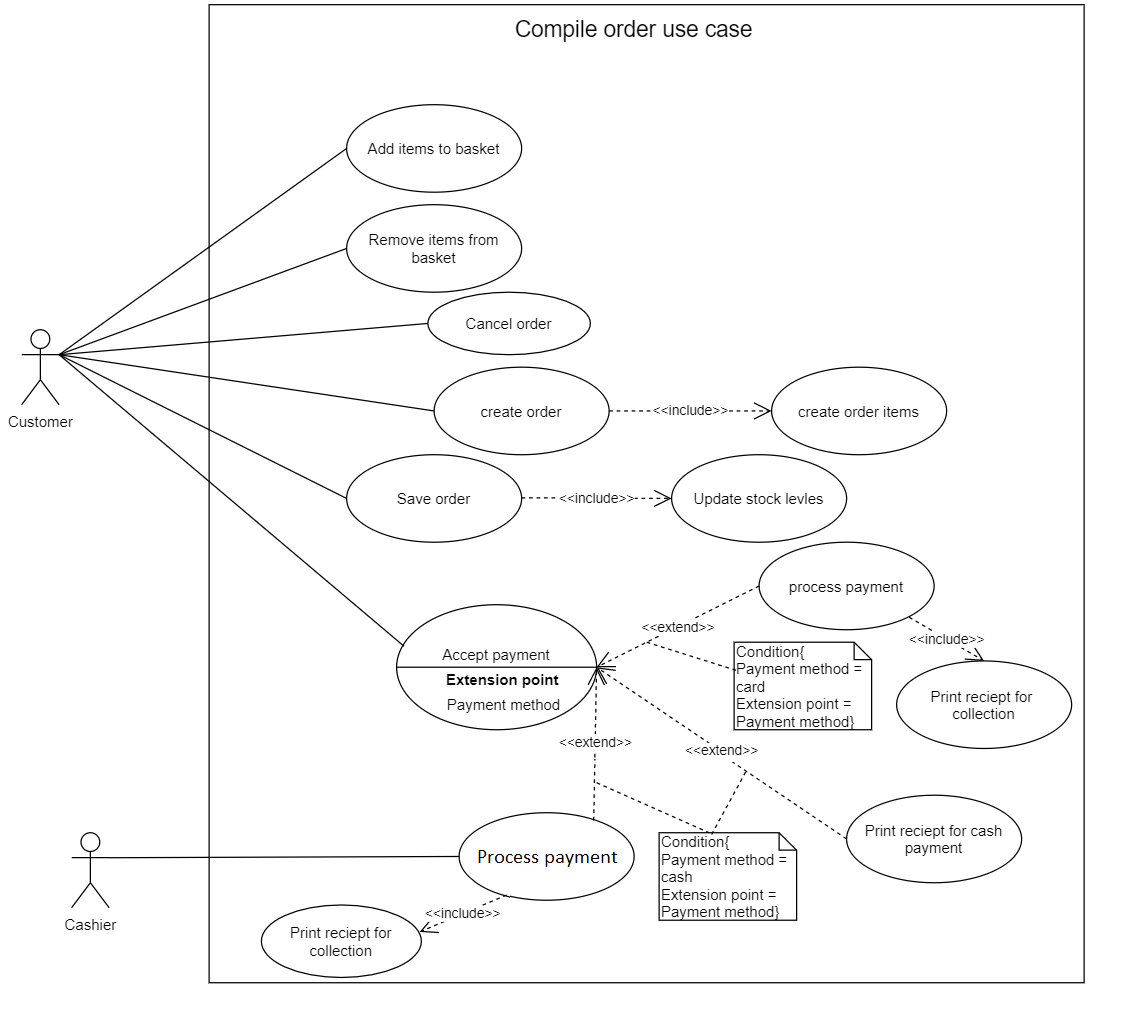
## Use case diagrams

Following are the use case diagrams that I have deduced from the Cargoss scenario. These diagrams are split into three section namely, customer registration, Cargoss sales system and stock management system :



# QIV

## Use case diagram for compile order

Following is the use case diagram decompiled into its constituent sues cases:

## Compile order use case description

Following list describes the compile order use case

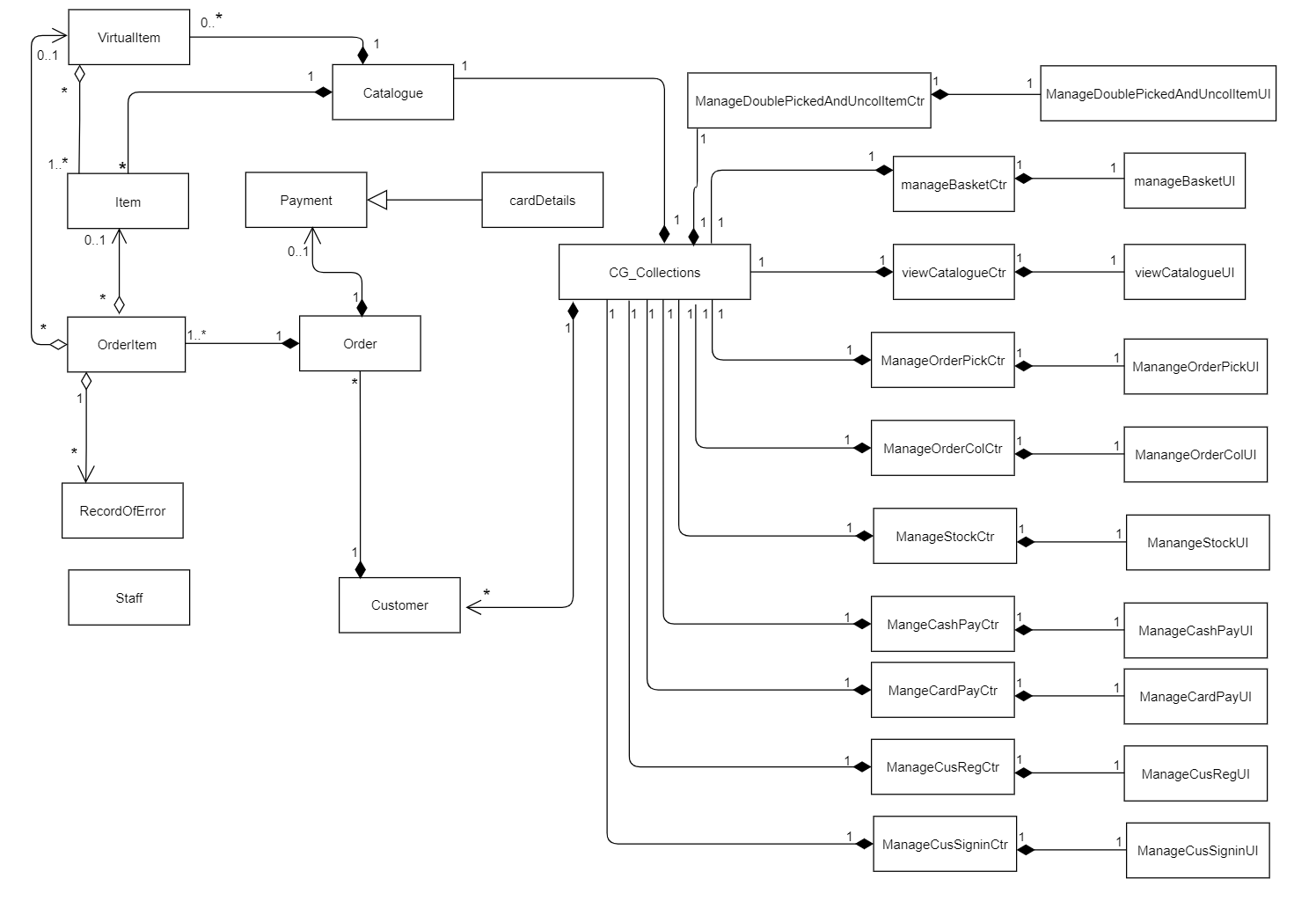
|  |
| --- |
| Customer clicks on the view basket button system will display the items in the bask of the customer along with a table of the catalogue items. |
| Customer will select an item from the catalogue items table and click on add to basket button system will update the basket and the UI accordingly. |
| Customer selects an item in basket and clicks on remove item button system will remove the item from the customer’s basket and will update the UI |
| If customer clicks on cancel order button and system will cancel the order and update the UI. |
| When customer clicks on the create order button system will create a new order and order-items for every order line in the order and assigns the order-items to the order. |
| Customer then clicks on save order button and system will update the stock level of the order-items and display the options to pay either by cash or by card |
| If customer clicks on the pay by card button then system will display card details form. User then fills the form and clicks submit, system will process the payment and will print receipt for collection |
| If customer clicks on pay by cash button then system will print receipt for cash payment. Customer than takes the receipt to the cashier who scans the receipt. As response of the scan system will display order details belonging to the scanned order. Cashier then receives the payment from the customer and updates the order status to paid. System in response updates the status of the order to paid and prints receipt for collection |

# QV

## Class diagram for the entire system

On the next page is the class diagram that I have created for the whole system.

Scroll to next page for class diagram because class diagram is to large to fit on this page



## Description of classes

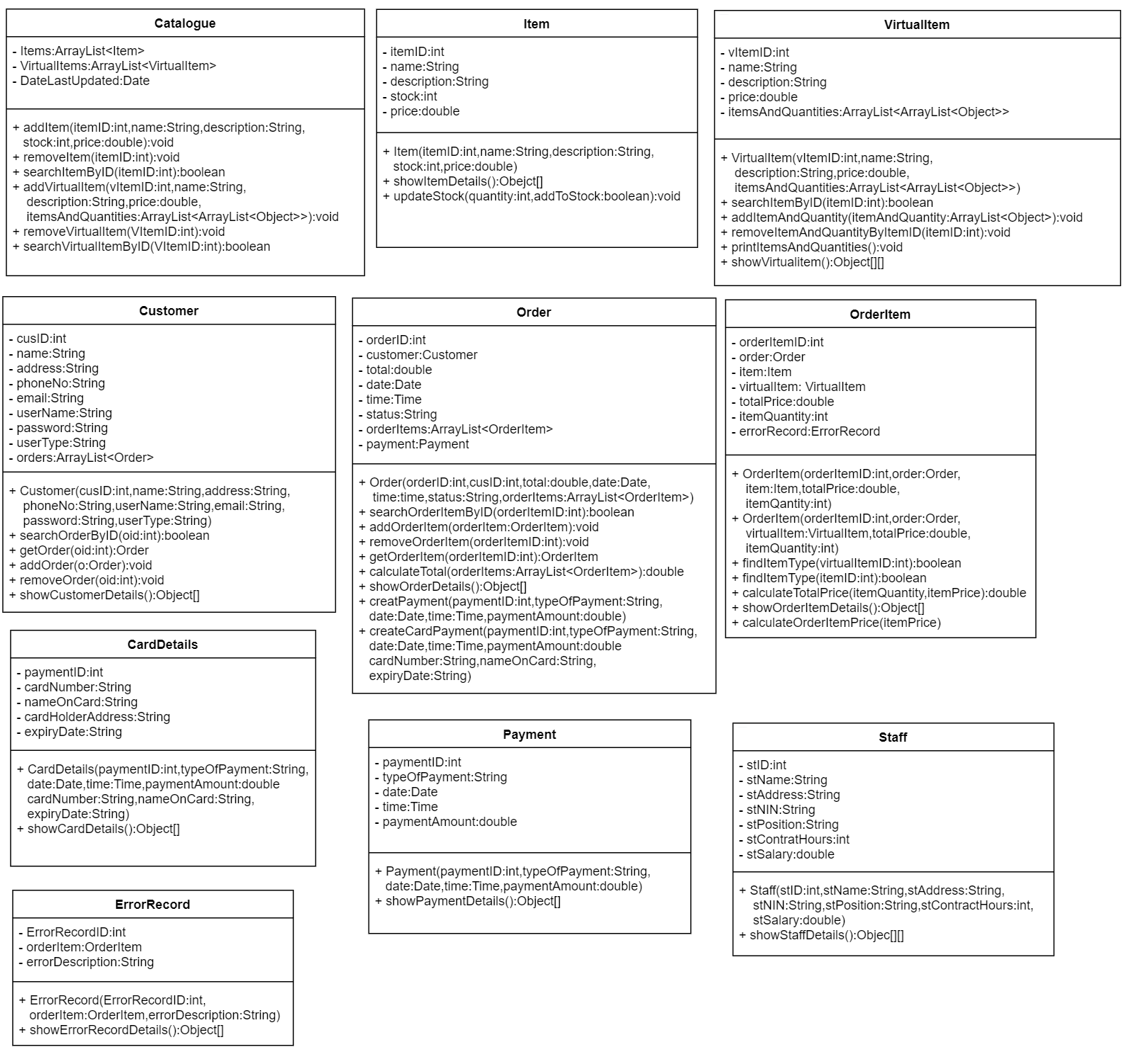
Following table contains description of the classes shown in above diagram

|  |  |
| --- | --- |
| Class name | Description |
| CG\_collections | Cargoss collections class is a collections class that holds the data of the Customer, Catalogue, Order, OrderItem, Payment, CardDetails , Item, virtual item and ErrorRecord classes. it contains methods to access that data. |
| ManageDoublePickedAndUncolUI | This class provides user interface through which store person can provide input regarding managing of double picked items and uncollected items. |
| ManageDoublePickedAndUncolCtr | This class handles the inputs from user and manages the data related to managing double picked or uncollected items. It contains all the necessary methods related to managing of double picked items and uncollected items. It also contains methods to update stock level of items |
| ManageBasketUI | This class provides user interface though which customer can provide inputs regarding basket operations. This class is also responsible for displaying the contents of the basket |
| ManageBasketCtr | This class handles data related to manage basket operations and contains methods that are related to managing basket like adding or removing items from the basket. This class provides the basket data that is then displayed by its respective UI class |
| viewCatalogueUI | This class display the catalogue and provide the functionality to scroll through the catalogue and select items. |
| viewCatalogueCtr | This class manages the data related to catalogue. It also contains methods such as add item to basket |
| ManageOrderPickUI | This class provide UI to manage the order. Using this class allows store person to input data regarding picking of the order |
| ManageOrderPickCtr | This class handles data related to managing order picking. It contains methods to update status of order item and order |
| ManageOrderColUI | This class provides UI for managing order collection. This class allows user to input data regarding collection operation of an order |
| ManageOrderColCtr | This class manages data related to collection of an order. It has methods like update order status |
| ManageStockUI | This class provide UI for managing deliveries. It allows user to input necessary data regarding the received deliveries |
| ManageStockCtr | This class handles data regarding management of stock for deliveries received. This class contains methods such as update stock levels |
| ManageCashPayUI | This class display UI for managing a cash payment. This class enables user to input data regarding processing of cash payment |
| ManageCashPayCtr | This class manages data related to processing of cash payment and it contains methods such as update order status, create payment and assign payment to order |
| ManageCardPayUI | This class displays UI for managing a card payment. This class enables user to input data necessary to process card payment. |
| ManageCardPayCtr | This class handles data related to managing a card payment it contains methods like create payment, save card details and assign payment to order |
| ManageCusRegUI | This class provides user interface related to customer account registration and allows user to input all the data necessary for registration |
| ManageCusRegCtr | This controller class manages data necessary for registration of a customer. It contains methods such as create new customer. |
| ManageCusSigninUI | This class provides UI which allows Customer to sign in. it allows user to input data such as username and password |
| ManageCusSigninCtr | This class handles data related to sign in operation. It contains methods such as validation methods which validated the input of the user against the data held in the database for that user |
| Item | Is an entity class. It is template for item object. The object of this class hold data related to a real item such as item name , item and item stock. It contains methods such as update stock level and has customized constructer |
| VirtualItem | This class creates virtual item objects which are used to hold data of a virtual item. This class has attributes such as name and price, but it also contains reference to array list of objects of Item class. Along with other methods this class contains methods for adding or removing. In easier words a virtual Item is made up of 1 or more real items at a discounted price. It is an entity class |
| Catalogue | This class contains real and virtual item objects in the form of two array lists and it contains methods to add, remove and search items of those array. It is an entity class |
| Payment | Payment class object is used to hold details of a cash payment. It has attributes like payment id, payment type and order id. It contains methods like show payment details. It is an entity class |
| CardDetails | This class inherits from the payment class and it saves details of card payment. It contains attributes like card number and card holder address. It is an entity class |
| Order | Object of this class holds data related to order. It has attributes like order id, date and total. It also contains object of payment class and an array list of OrderItem class objects. It has methods such as methods to search, add, remove order items. |
| OrderItem | This class is used to create order items object which hold data related to order line. Each object of this class holds reference to either a real item or a virtual item and it may also contain object of ErrorRecord class it contains other attributes such as quantity. It is an entity class. |
| ErrorRecord | The object of this class holds data related to an error made in picking of an order item. It has attributes as error id and error description. |
| Customer | Customer object holds data of related to customer. It has attributes such as customer id and customer name. it contains array list of order objects. It contains methods like search for order, add order and remove order. |
| Staff | This class holds data related to staff. It is an entity class and it has attributes like staff id, name, address and national insurance number |

# QVI

## Entity classes - attributes and methods

Following are the entity classes from the class diagram along with their methods and attributes:



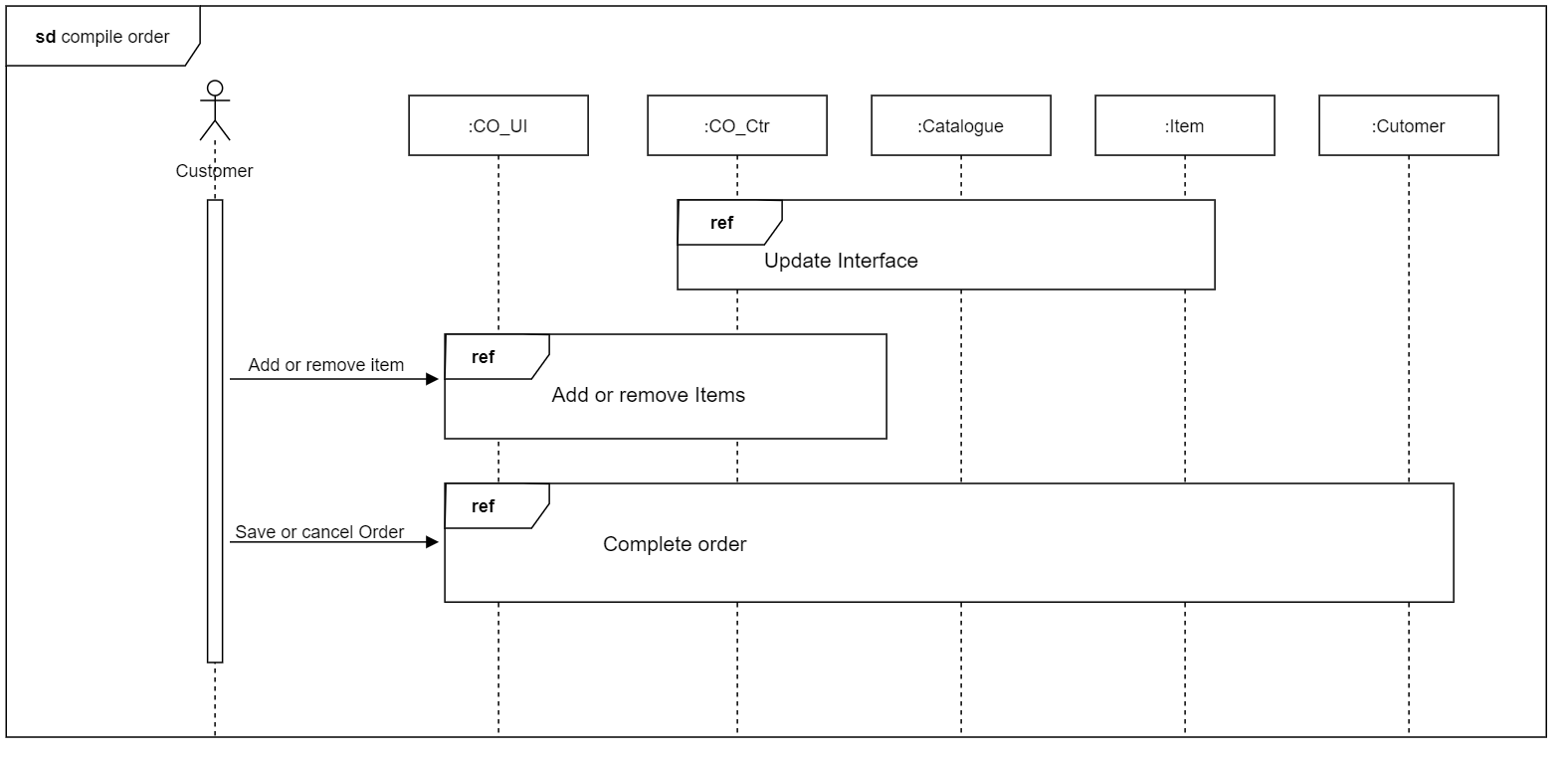
# QVII

## Sequence diagram for compile order

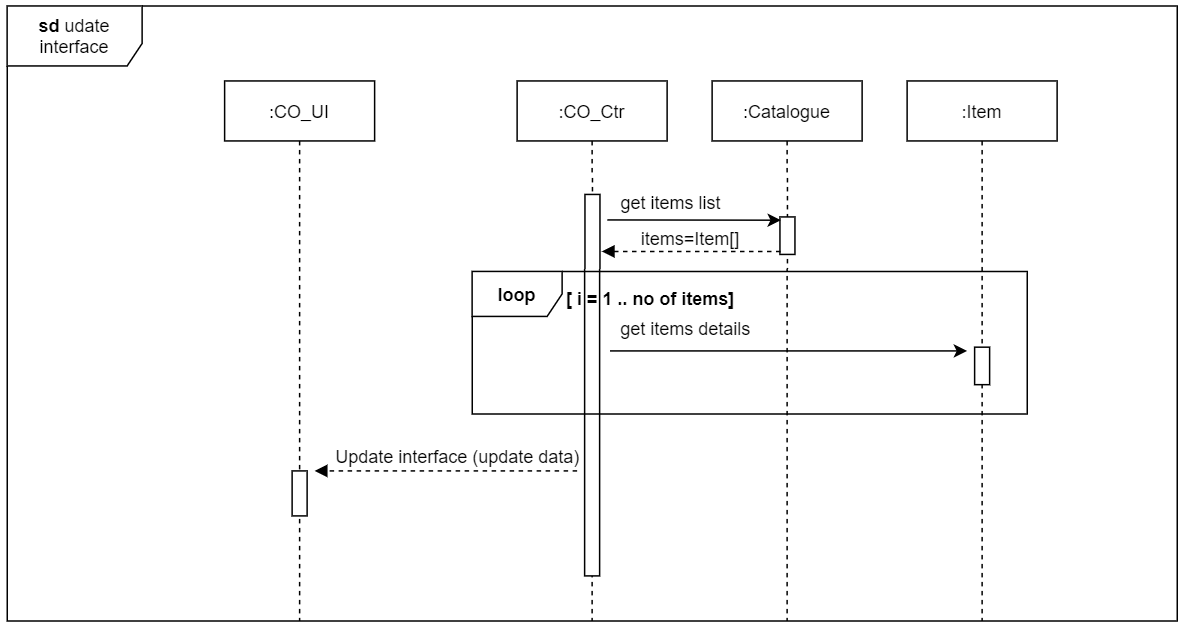
following is the compile order sequence along with its constituent parts along with assumptions:

**Assumptions:**

* Customer has clicked on the view basket button hence the diagram starts at basket user interface being updated.

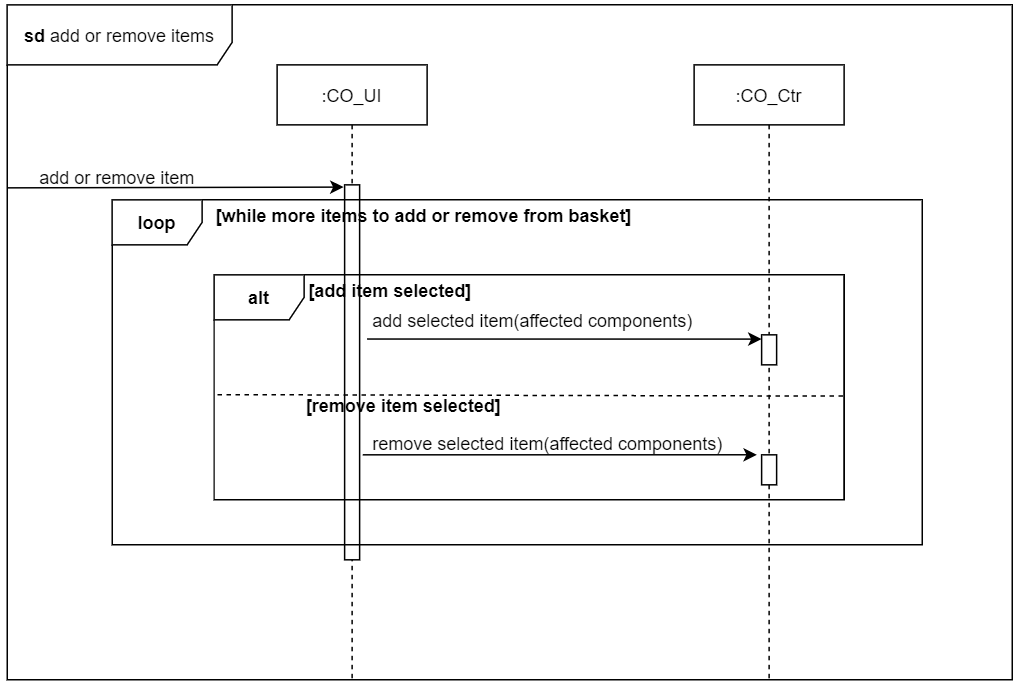


The above diagram is divided into following diagrams please see the description for each reference diagram right below it.



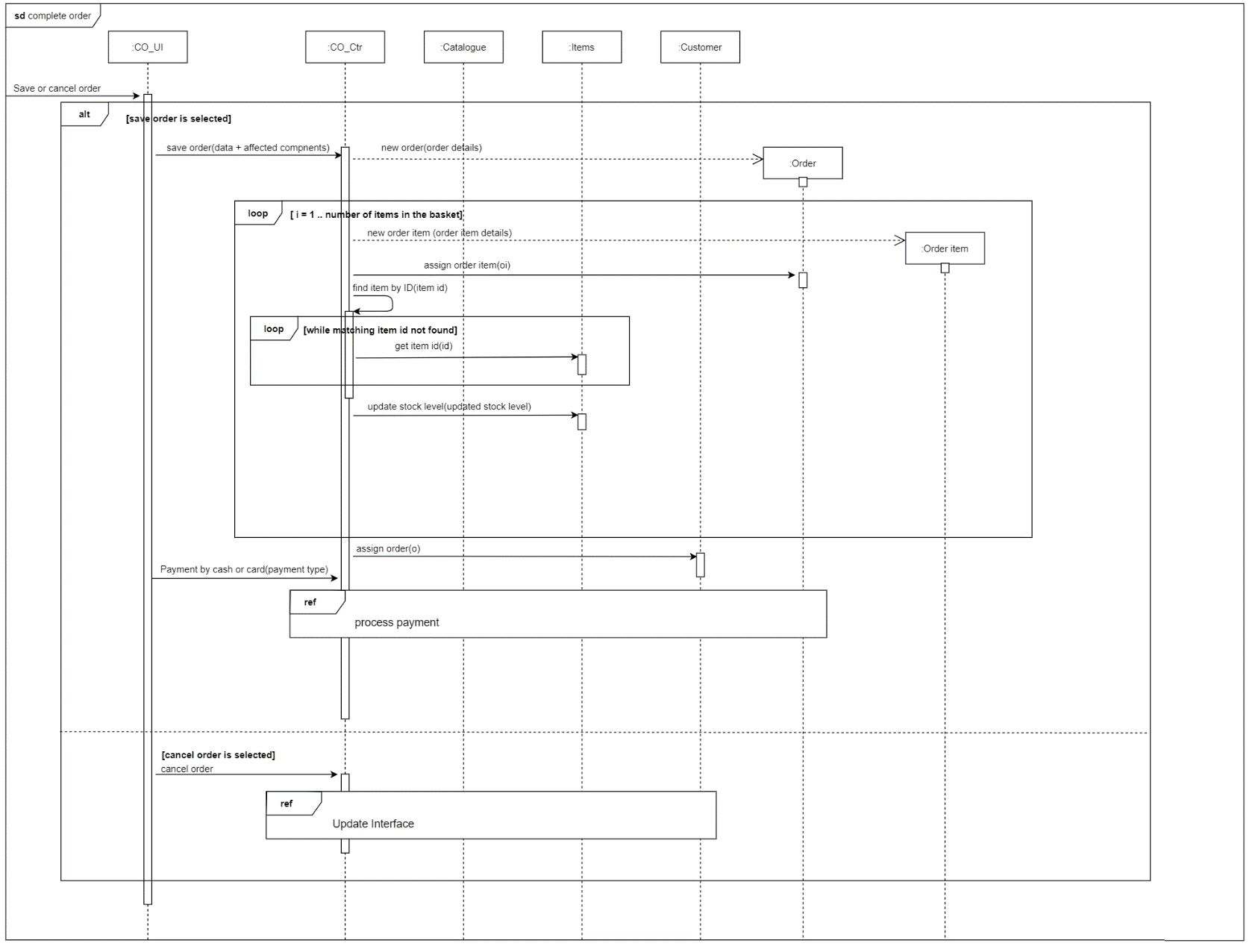
### Description

* Controller contains catalogue object
* Catalogue contains Array List of items and virtual items.
* The CO controller asks catalogue for the Array List of items and then iterates through that list, asking each item to pass its details. Those details are then stored by CO controller in an Object[]. At the end CO controller passes that Object[] to the CO interface, which displays those items



### Description

* Each time basket is updated, all the affected components that are affected by the updated are passed by CO interface in a message. After reading all the information necessary for update directly from those components, CO controller processes that information and updates the components accordingly.
* Items can always be added to the basket
* Items can only be removed from the basket if the number of items in basket is greater than zero
* Inside the above diagram the loop that handles adding or removing of the items to and from basket goes on until there are more items to be added or removed.



### Description

* Order items are contained inside orders
* Orders are contained inside the customers
* A mixture of data(for example order data) and the components(like items table, basket table) are passed by the CO interface to the CO controller in a message.
* The controllers creates a new order.
* CO Controller then loops through each item in the basket.
* For each item in the basket CO controller creates new order-item and assigns that order-item to the order. After that controller updates stock-level of the item concerned in the items table.
* Finally, the controller stops looping and assigns the order to the customer who placed it.
* If customer cancels the order the CO interface is reset. See update interface ref

See ref payment by cash or card below:

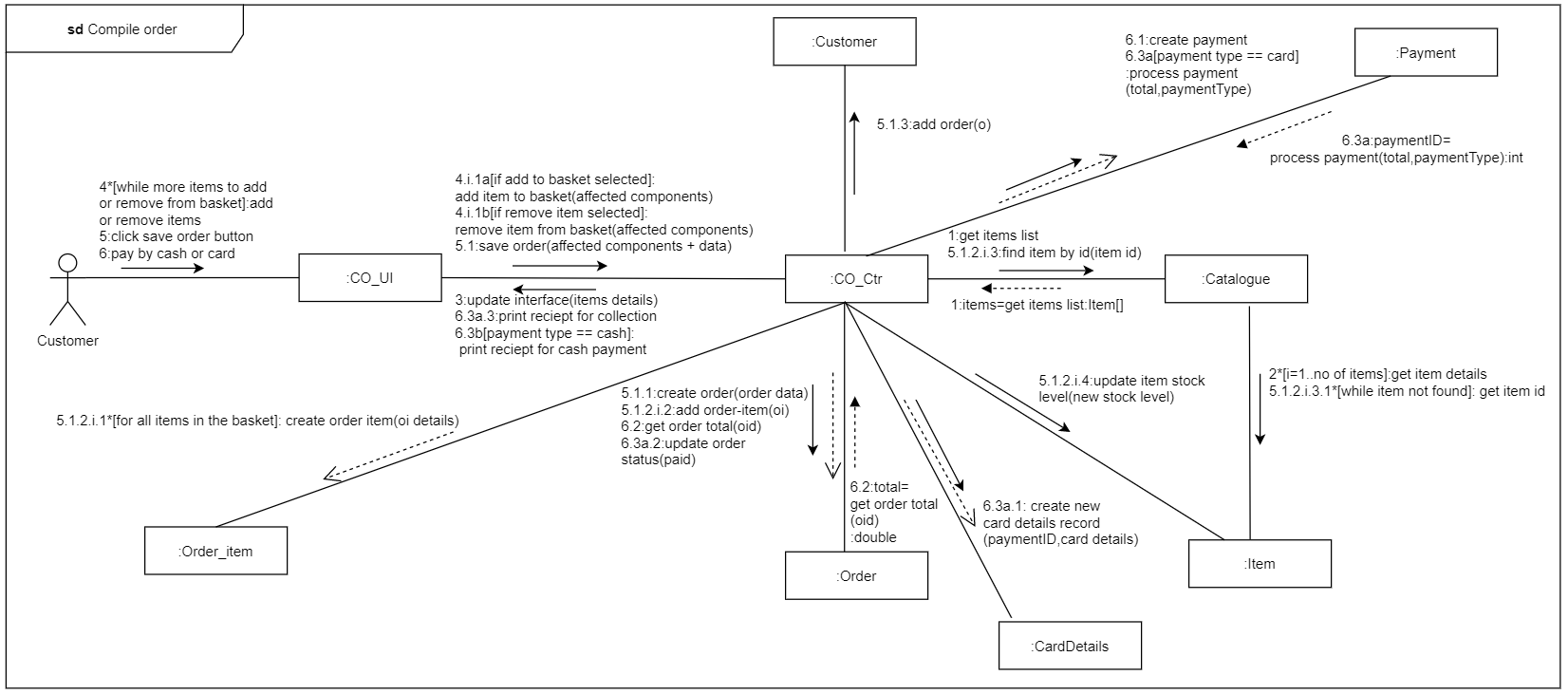
### Description

* When user clicks buy button, controller creates a payment object and then gets total price of the order from the Order object.
* If the user has selected pay by card then controller creates a payment object and passes details of payment along with order total to that object.
* At this point controller creates a new CardDetails object and passes details of the card which user has entered to that object
* Controller than updates the status of the order to paid
* Once that is done controller prints a receipt for collection
* If the customer has chosen to pay by cash controller will print a receipt for cash payment

# QVIII

## Compile order Communication diagram

Following is the communication diagram that I have draw for compile order use case followed by its description:



### Description

* In the above communication diagram when user clicks on the view basket button. Controller asks the catalogue to pass it the reference to the array containing items.
* Controller after receiving that array updates the user interface of the customer display the items.
* If customer chooses to add or remove items to and from basket controller updates the basket accordingly
* When customer clicks on save order button controller receives the affected components and data such as items in the basket and their respective quantities it creates an order object. Controller then finds the order items in the catalogue and their details. Then controller creates an order item for each item in the basket updates the stock level of those items. Once that is done controller assigns the order to the customer.
* Customer selects the payment type and fills in the details and presses the pay button controller will receive the data from UI and will create a payment object. Then controller gets the total price from the order object. Now controller check if the payment method is cash than it prints a pay by cash receipt for the customer. If pay by card option is selected controller save the payment details and receives payment id from the payment object. Controller then creates new CardDetails object and passes it payment id along with other card details entered by the customer. Once payment is complete controller updates the status of the order and prints the collection receipt for the customer.