Cover Page

Restaurant App

https://github.com/kylin1209/restaurant-app

Group#
Kylie Lin (kl4101)
Jasmine Fan (jf4654)
Ray Ochotta (amo9758)

August 13, 2025

Table of Work

(Please write x in the boxes to mention what each student achieved in this project)

	Kylie Lin	Jasmine Fan	Ray Ochotta
Project Description	X	X	X
Uses Cases Diagram(s) and discription	X	X	X
Sequence Diagrams	X	X	
Class diagram(s)			X
Implementation	X	X	
Conclusion	X	X	X

Table of Contents

- System Analysis
  - Written Project description (One Page)
    - General Description, Goals and Benefits
    - Special requirements (Performance, Interfaces, Constraints, Reliability, if any)
  - Uses Cases Diagram(s) and use cases description.
- System Design
  - Sequence Diagrams
  - Class diagram(s)
- Conclusion (work summary, difficulties, recommendations, ..)
- Optional Appendix (any related reports, questionnaires, docs..)

System Analysis:

Written Project Description:

Restaurant application is an app for a fictional restaurant with two interfaces: one for customers to order menu items from a fictional restaurant and another for deliverers to deliver said orders.

After logging in, Customers can view the menu and choose to add menu items to their cart, where they can then access from buttons on the top of the GUI. From the cart view, customers can then order their food and receive a confirmation. There will also be a hamburger menu to view their past orders and delivery status. Customers can also track the status of their order upon checkout.

After logging in, Deliverers can view a suite of different orders yet to be delivered. They can choose to "book" an order, deeming it their responsibility. They have to confirm that they will be responsible for said order through another page with the order details. After reserving the order, the Deliverer can deem it delivered through another button. A hamburger menu will also allow deliverers to view their past deliveries and log out.

Goals:

- For Customers:
    - Login
    - Sign Up
    - Order from the Menu
    - Add menu items to their cart
    - Purchase their chosen items
    - Check/ Wait for delivery
    - Check their past orders
- For Deliverers:
    - Login
    - Sign Up
    - Book an Order to deliver
    - Deliver an Order
    - See past completed deliveries
    - Earn money

Benefits:

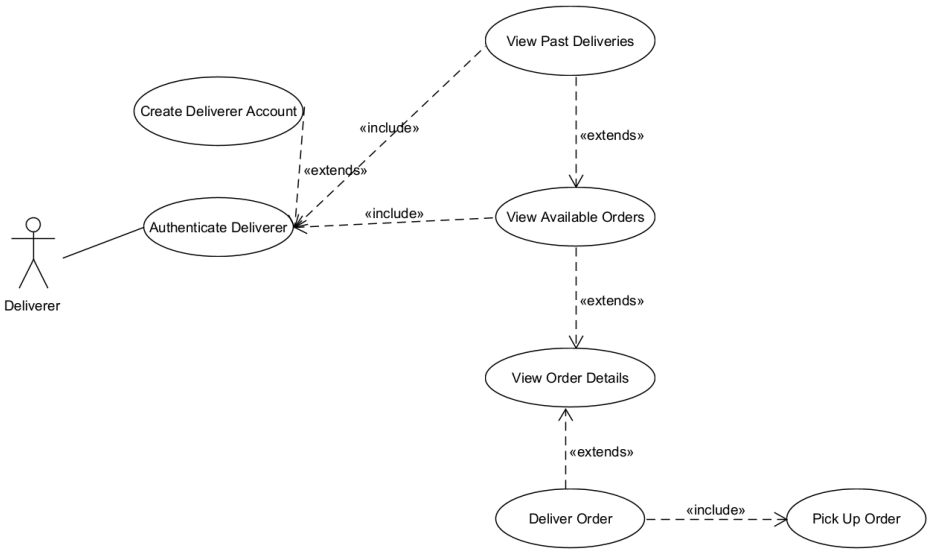
- Customers and Deliverers use one app to interact.

Special Requirements:

- Please use VS Code to run our app.

Use Case Diagrams:

Deliverer:



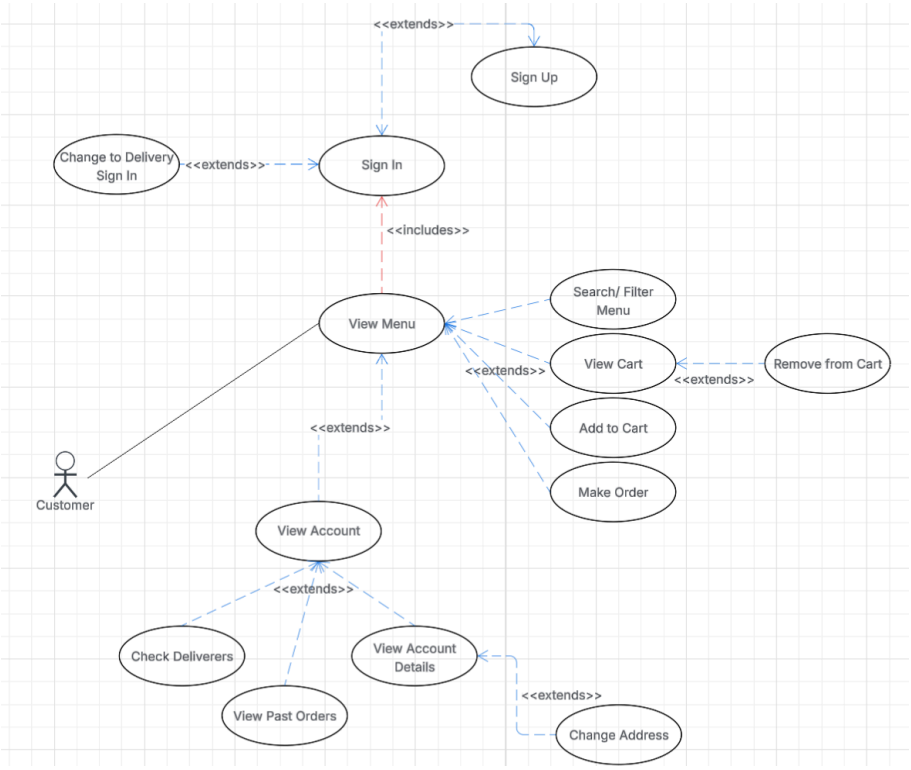
1	Authenticate Deliverer	2	Create Deliverer Account
Overview	The Deliverer logins in an existing account.	Overview	The Deliverer creates a new account, then is sent back to UC 1 to sign in to the new account.
Related Use Cases	UC 2 extends UC 1. UC 3, 4 are included in UC 1.	Related Use Cases	UC 2 extends UC 1.
Actors	Deliverer	Actors	Deliverer

3	View Past Deliveries	4	View Available Orders
Overview	The Deliverer can view their own previous deliveries.	Overview	The Deliverer views all orders that they are able to take on.
Related Use Cases	UC 3 includes UC 1. UC 3 extends UC 4.	Related Use Cases	UC 4 includes UC 1. UC 3 extends UC 4.
Actors	Deliverer	Actors	Deliverer

5	View Order Details	6	Deliver Order
Overview	The Deliverer views what to pick up, and where to deliver it.	Overview	The Deliverer selects an order to deliver, and then prepares to deliver it.
Related Use Cases	UC 5 extends UC 4, 6.	Related Use Cases	UC 5 extends UC 6. UC 7 includes UC 6.
Actors	Deliverer	Actors	Deliverer

7	Pick Up Order
Overview	The Deliverer picks up the order from the restaurant.
Related Use Cases	UC 7 includes UC 6.
Actors	Deliverer

Customer:



1	View Menu	2	Sign In
Overview	Customer opens the app to read the menu.	Overview	Customer signs in to their account.
Related Use Cases	UC 2 includes UC 1. UC 5-8 extends UC 1. UC 10 extends UC 2.	Related Use Cases	UC 2 includes UC 1. UC 2 extends UC 3, and vice versa. UC 2 extends UC 4.
Actors	Customer	Actors	Customer
3	Sign Up	4	Change to Delivery Sign In
Overview	Customer signs up for an account.	Overview	User of app switches to the Deliverer view of the app to sign in.
Related Use Cases	UC 3 extends UC 2, and vice versa.	Related Use Cases	UC 2 extends UC 4.
Actors	Customer	Actors	Any user of app or a customer

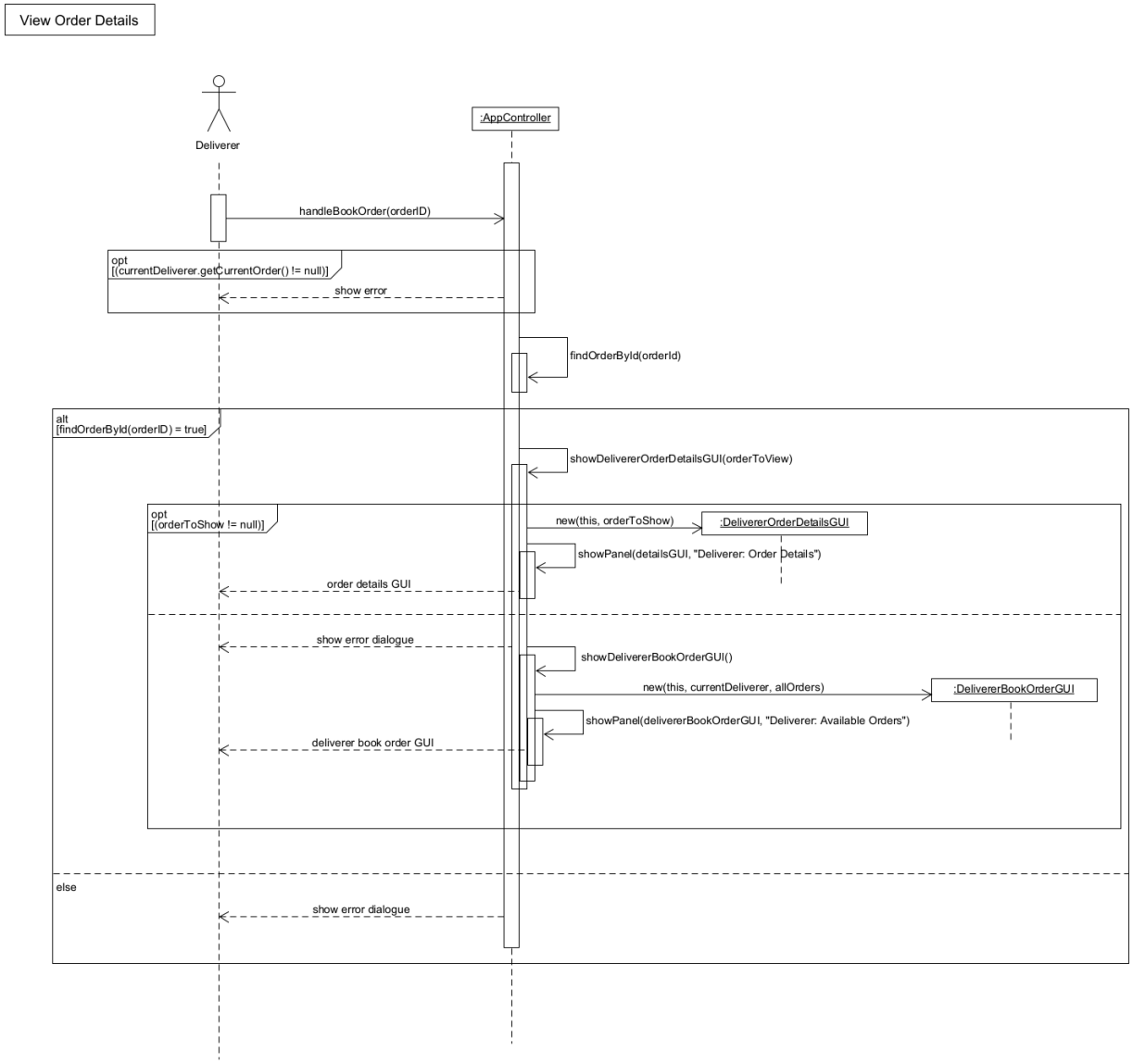
5	Search/Filter Menu	6	View Cart
Overview	The Customer looks through the menu, with several options to ease their search.	Overview	The Customer view the cart.
Related Use Cases	UC 5 extends UC 1.	Related Use Cases	UC 6 extends UC 1. UC 9 extends UC 6.
Actors	Customer	Actors	Customer
7	Add to Cart	8	Make Order
Overview	The Customer adds one item of their choosing to the cart.	Overview	The customer submits their order.
Related Use Cases	UC 7 extends UC 1.	Related Use Cases	UC 8 extends UC 1.
Actors	Customer	Actors	Customer
9	Remove from Cart	10	View Account
Overview	Customer removes a selected item from their cart.	Overview	The Customer views their account.
Related Use Cases	UC 9 extends UC 6.	Related Use Cases	UC 10 extends UC 1. UC 11-13 extends UC 10.
Actors	Customer	Actors	Customer
11	Check Deliverers	12	View Past Orders
Overview	The Customer is able to view how far along their order is.	Overview	The Customer can view all previous orders.
Related Use Cases	UC 11 extends UC 10.	Related Use Cases	UC 11 extends UC 10.
Actors	Customer	Actors	Customer
13	View Account Details	14	Change Address
Overview	The Customer views their account details.	Overview	The customer changes their address.
Related Use Cases	UC 13 extends UC 10.	Related Use Cases	UC 14 extends UC 13.

Actors	Customer	Actors	Customer

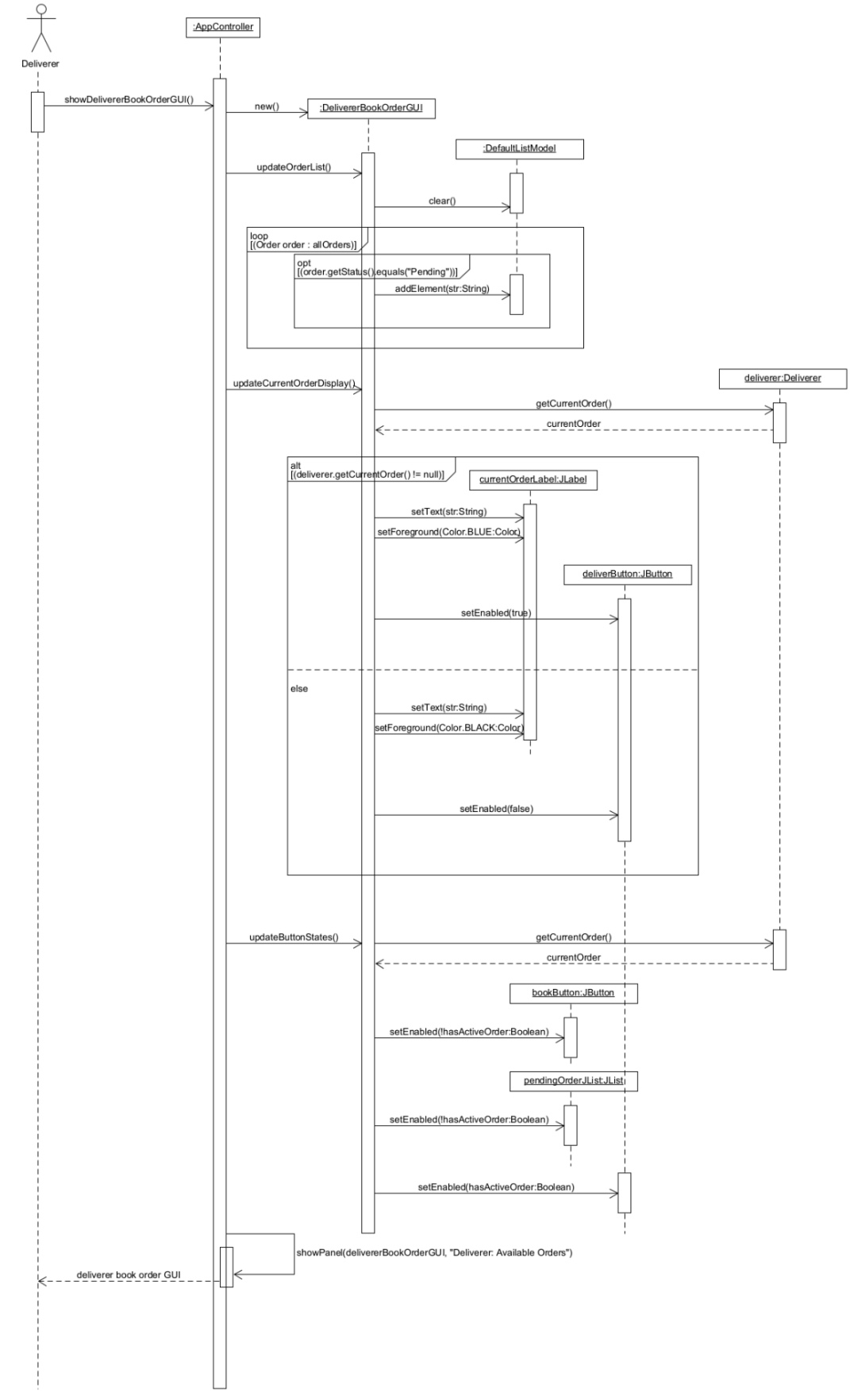
System Design:

Sequence Diagrams:

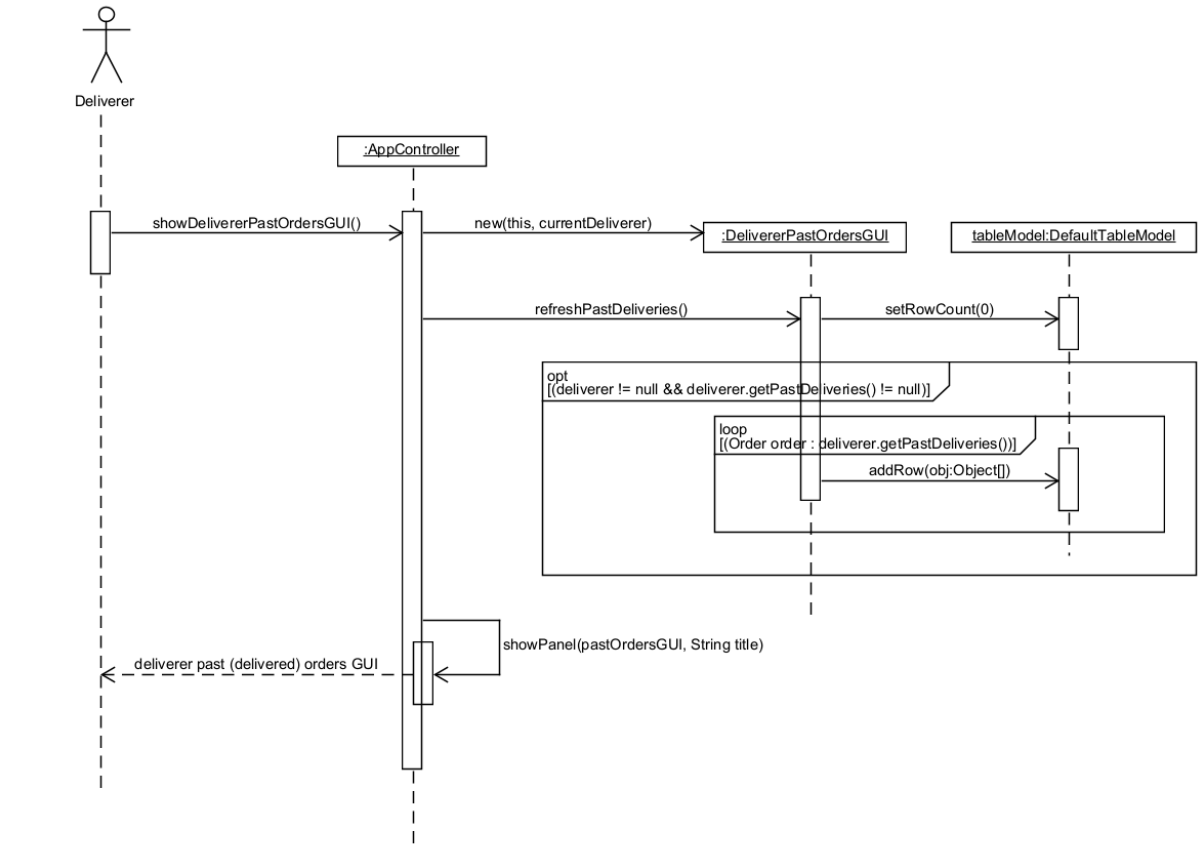
Deliverer:



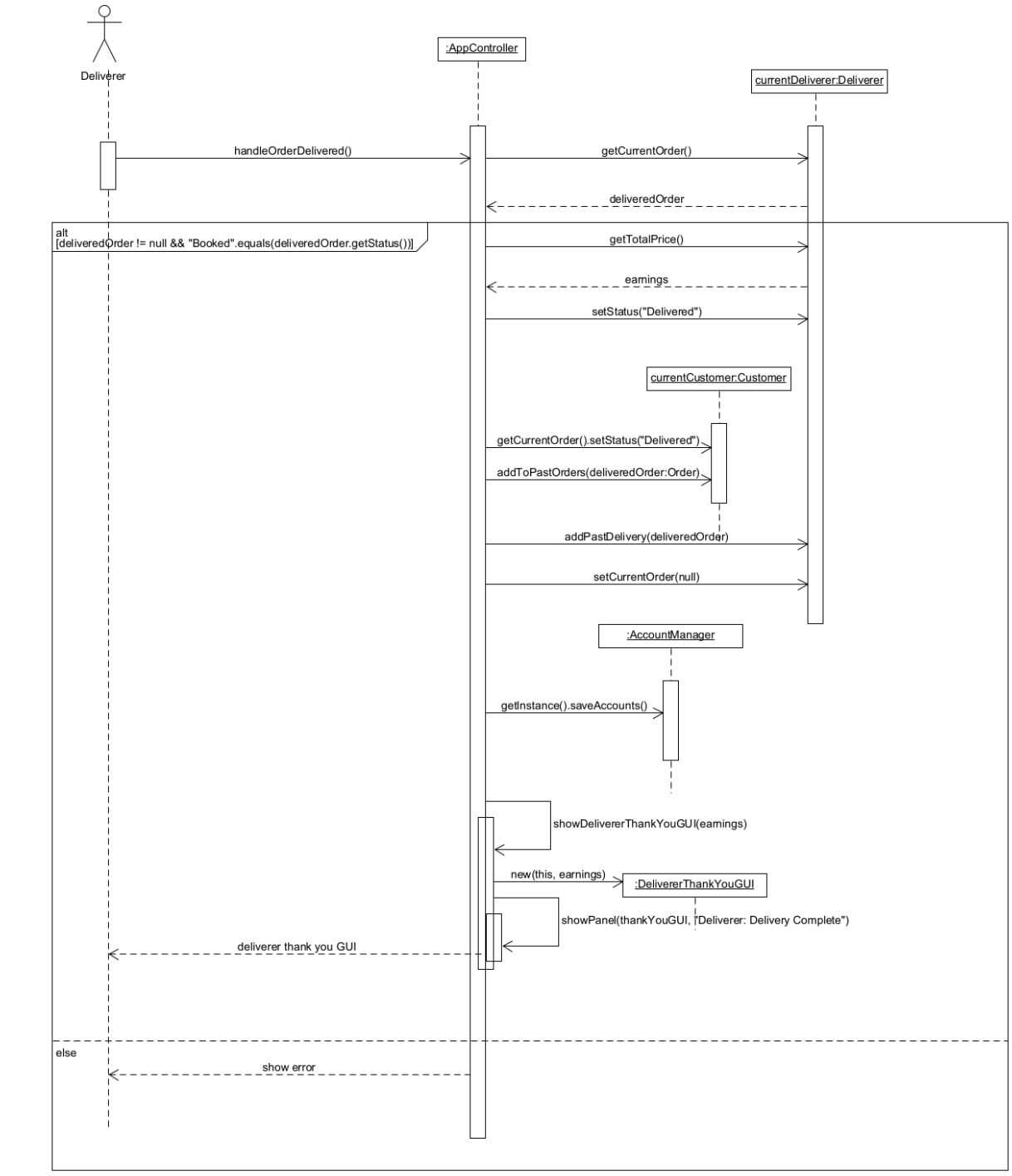
View Available Orders



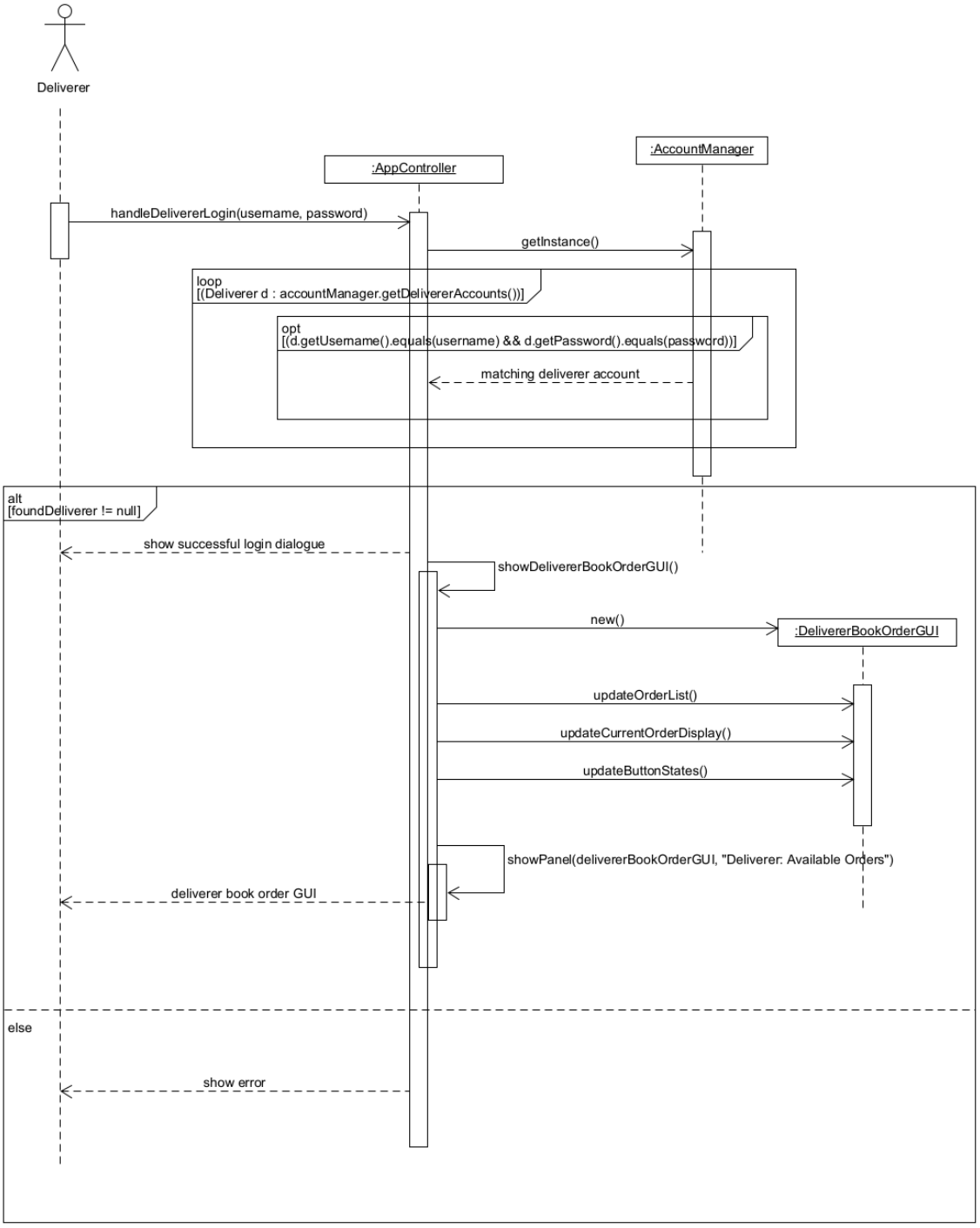
View Past Deliveries

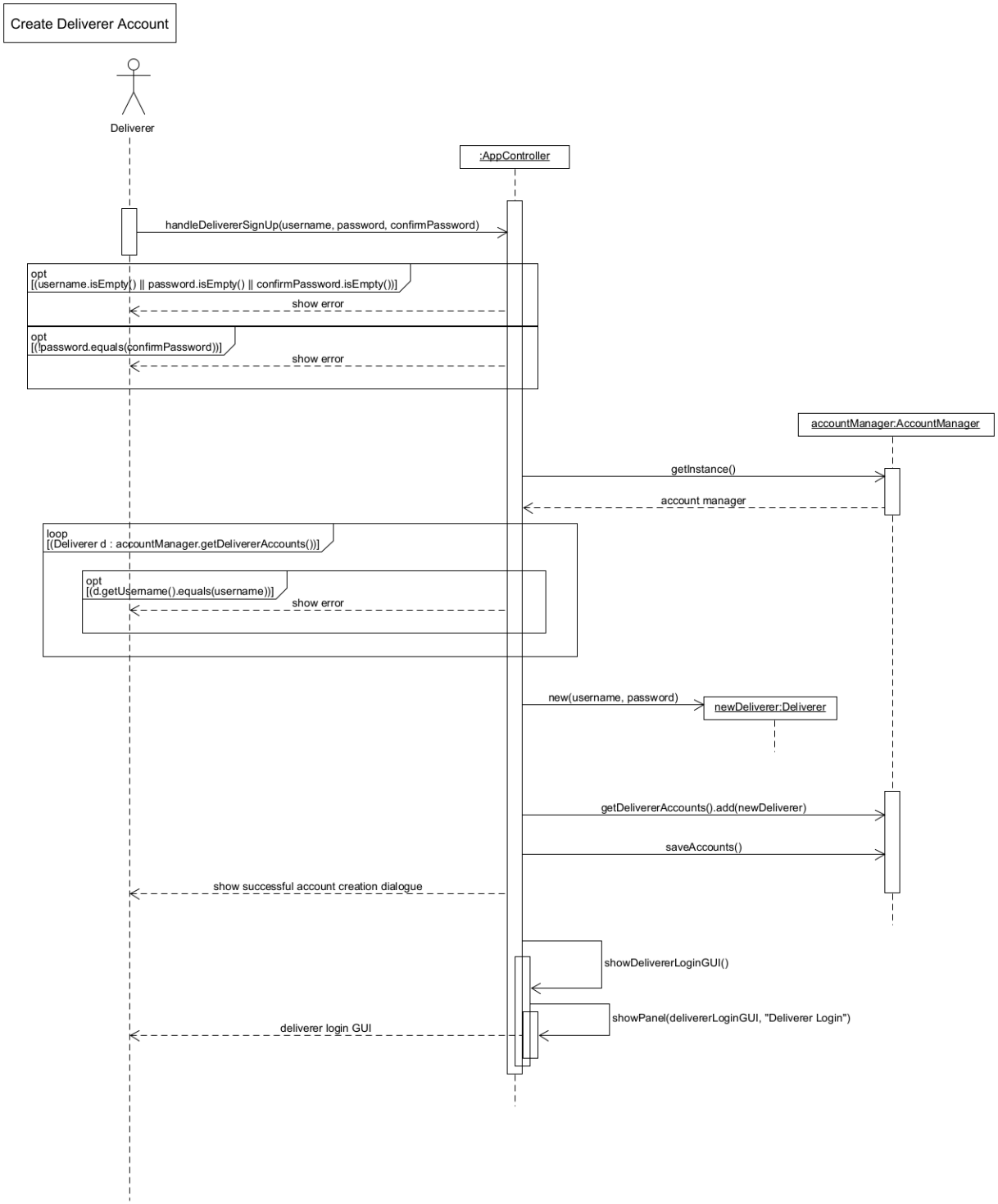


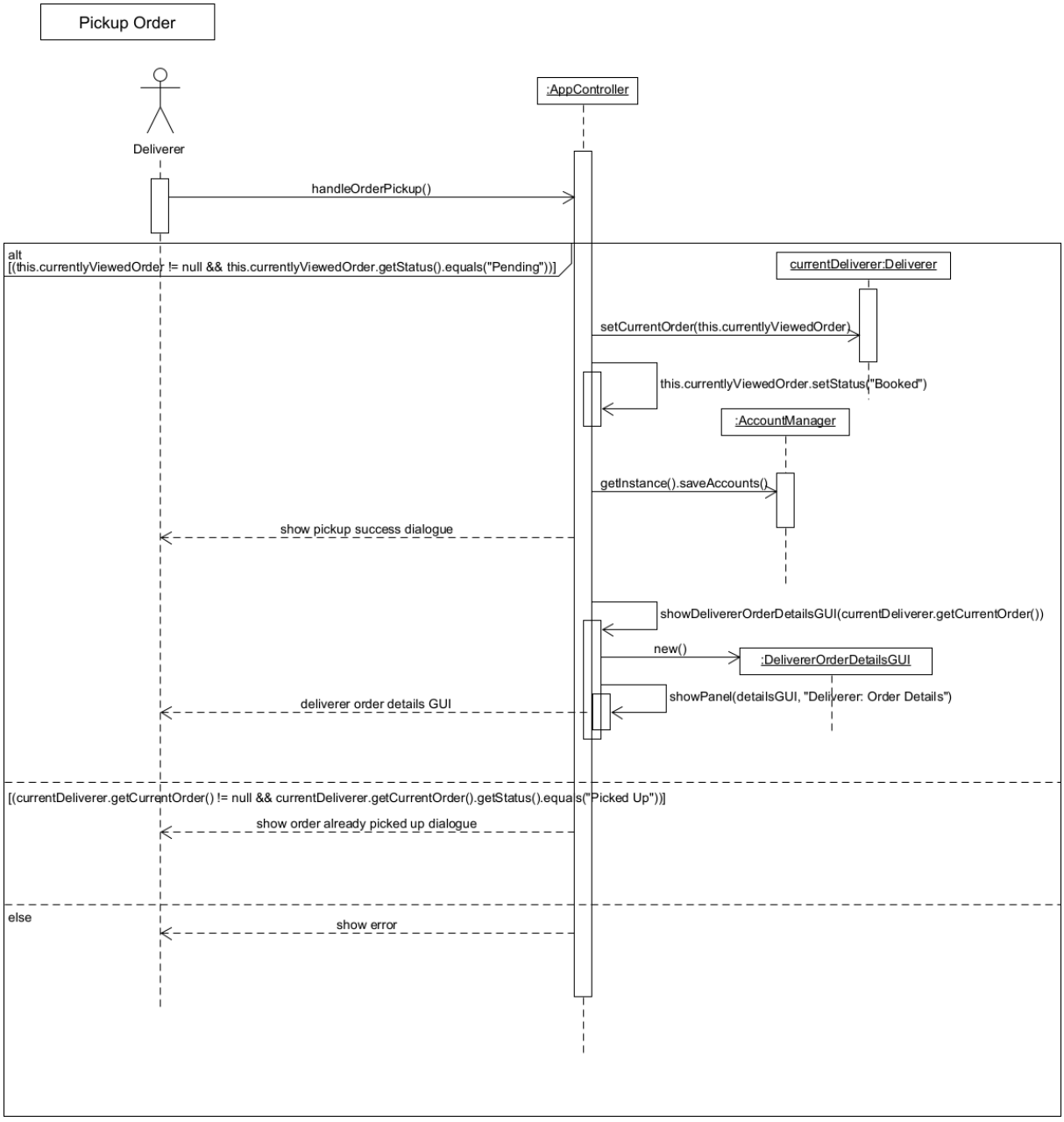
Deliver Order



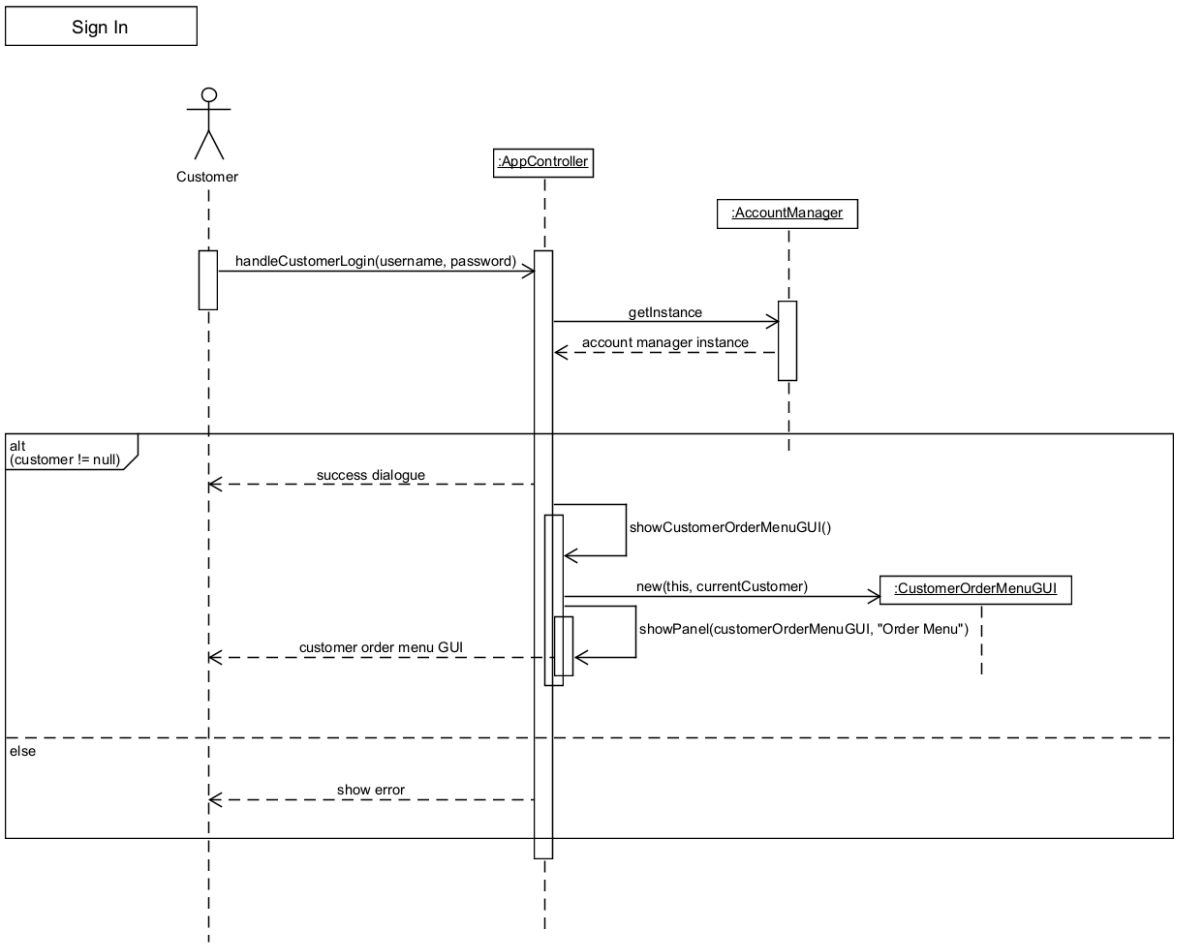
Authenticate Deliverer

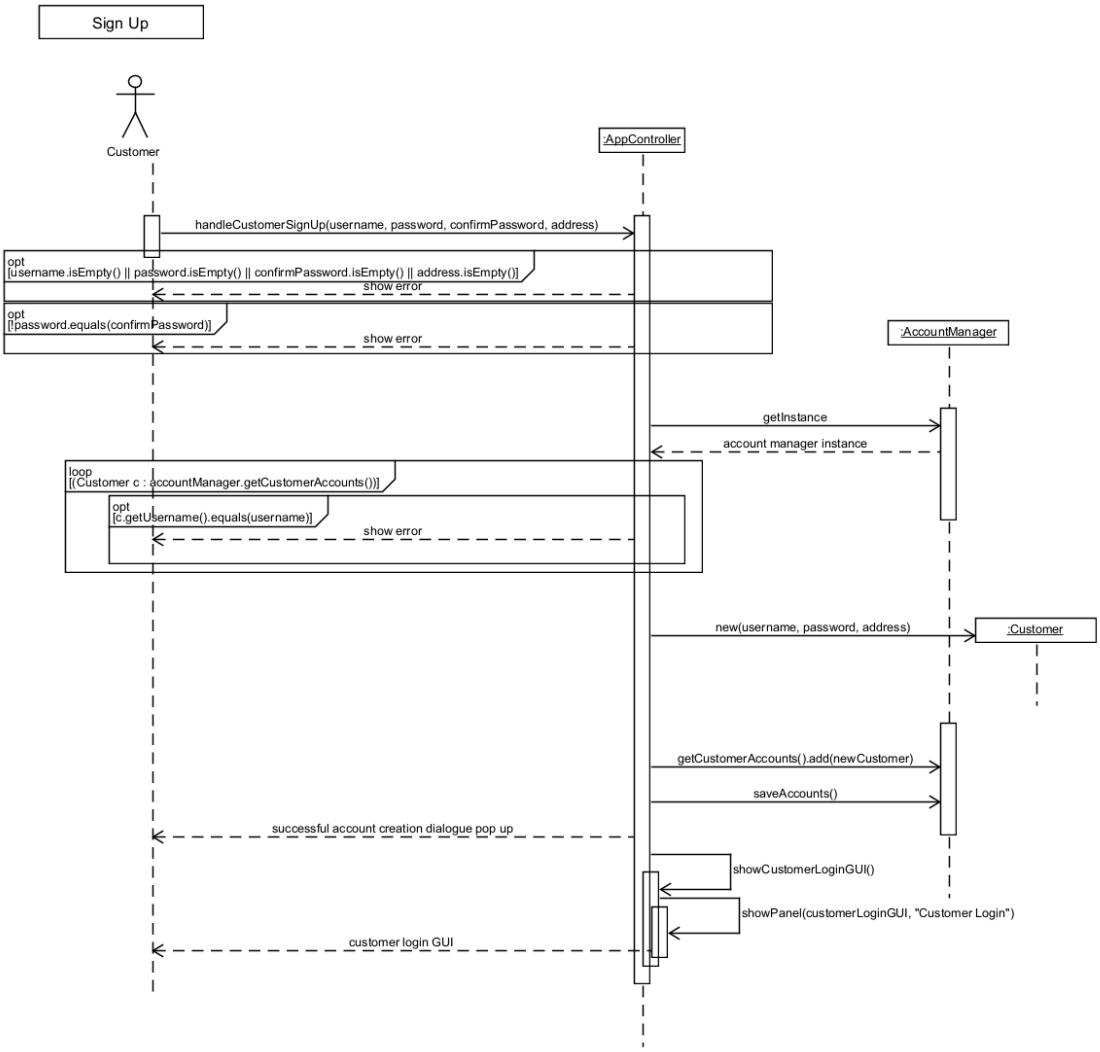



Create Deliverer Account
Deliverer
:AppController
handleDelivererSignUp(username, password, confirmPassword)
opt
[(username.isEmpty() || password.isEmpty() || confirmPassword.isEmpty())]
show error
opt
[(!password.equals(confirmPassword))]
show error
accountManager:AccountManager
getInstance()
account manager
loop
[(Deliverer d : accountManager.getDelivererAccounts())]
opt
[(d.getUsername().equals(username))]
show error
new(username, password)
newDeliverer:Deliverer
getDelivererAccounts().add(newDeliverer)
saveAccounts()
show successful account creation dialogue
showDelivererLoginGUI()
showPanel(delivererLoginGUI, "Deliverer Login")
deliverer login GUI

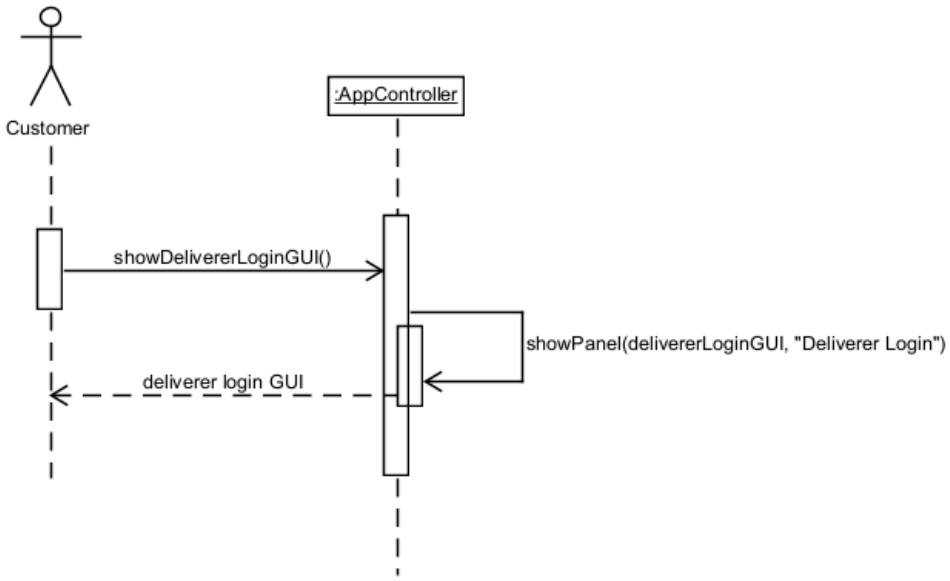

Pickup Order
Deliverer
:AppController
handleOrderPickup()
alt
[(this.currentViewedOrder != null && this.currentViewedOrder.getStatus().equals("Pending"))]
currentDeliverer:Deliverer
setCurrentOrder(this.currentViewedOrder)
this.currentViewedOrder.setStatus("Booked")
:AccountManager
getInstance().saveAccounts()
show pickup success dialogue
showDelivererOrderDetailsGUI(currentDeliverer.getCurrentOrder())
new()
:DelivererOrderDetailsGUI
showPanel(detailsGUI, "Deliverer: Order Details")
deliverer order details GUI
[(currentDeliverer.getCurrentOrder() != null && currentDeliverer.getCurrentOrder().getStatus().equals("Picked Up"))]
show order already picked up dialogue
else
show error

Customer:

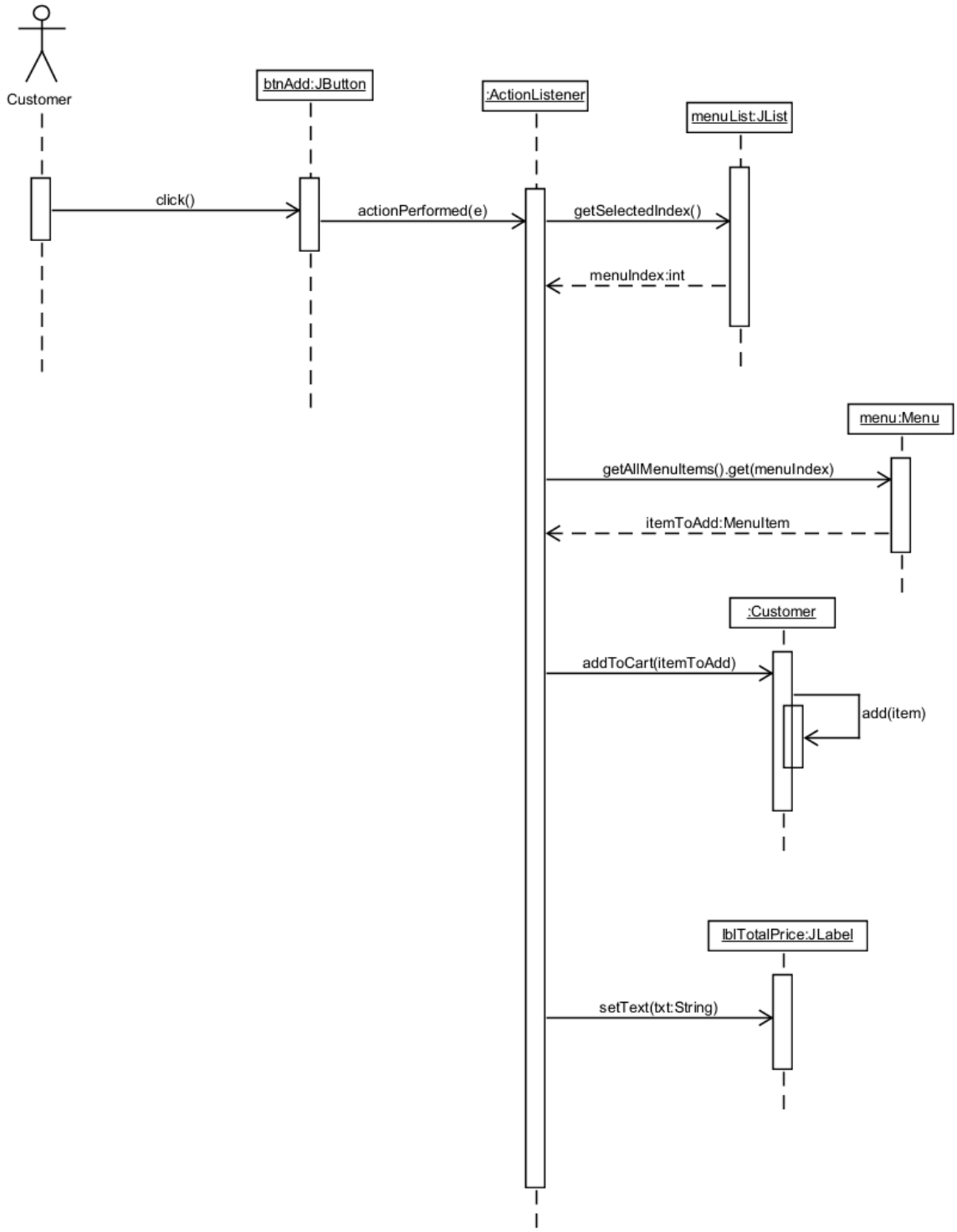

Sign In
Customer
:AppController
:AccountManager
handleCustomerLogin(username, password)
getInstance
account manager instance
alt
(customer != null)
success dialogue
showCustomerOrderMenuGUI()
new(this, currentCustomer)
:CustomerOrderMenuGUI
showPanel(customerOrderMenuGUI, "Order Menu")
customer order menu GUI
else
show error


Sign Up
Customer
:AppController
handleCustomerSignUp(username, password, confirmPassword, address)
opt
[username.isEmpty() || password.isEmpty() || confirmPassword.isEmpty() || address.isEmpty()]
show error
opt
[!password.equals(confirmPassword)]
show error
:AccountManager
getInstance
account manager instance
loop
[(Customer c : accountManager.getCustomerAccounts())]
opt
[c.getUsername().equals(username)]
show error
new(username, password, address)
:Customer
getCustomerAccounts().add(newCustomer)
saveAccounts()
successful account creation dialogue pop up
showCustomerLoginGUI()
showPanel(customerLoginGUI, "Customer Login")
customer login GUI

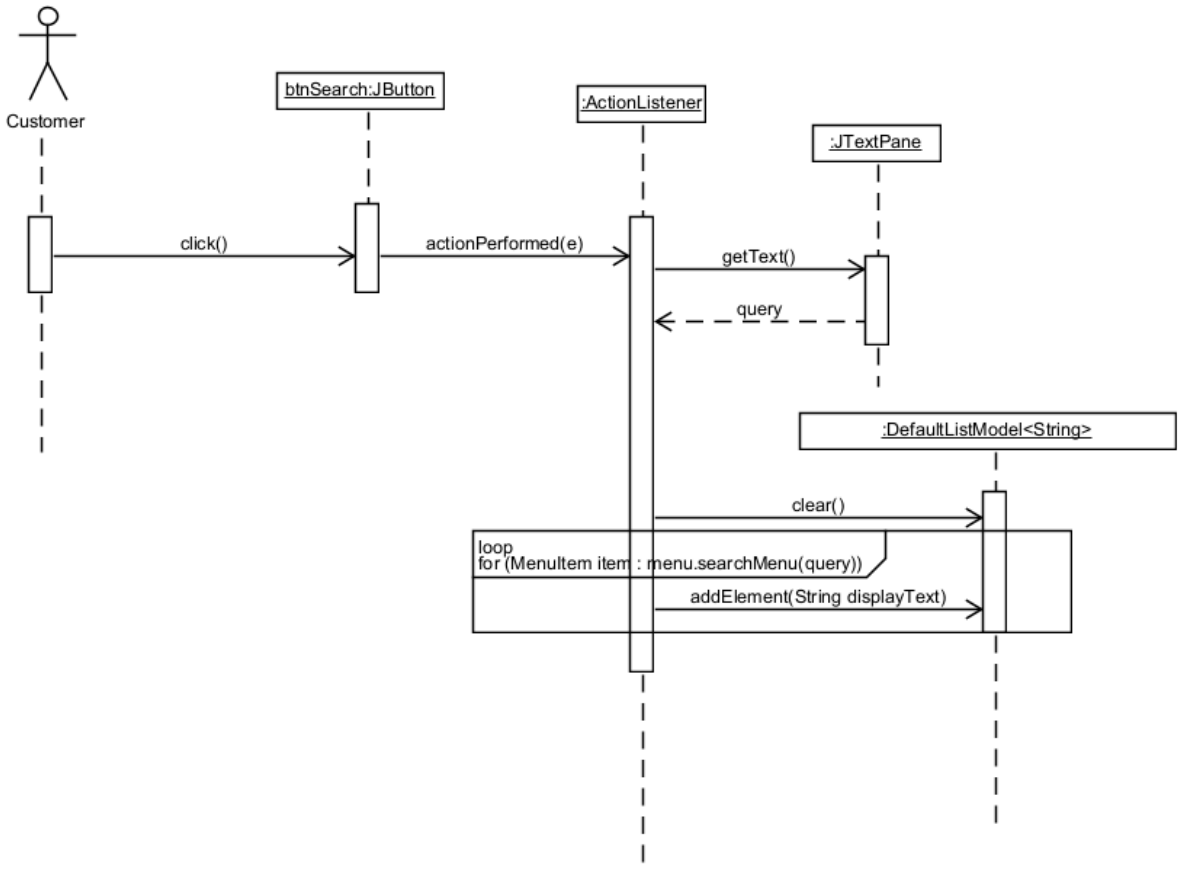
Change to Delivery Sign In



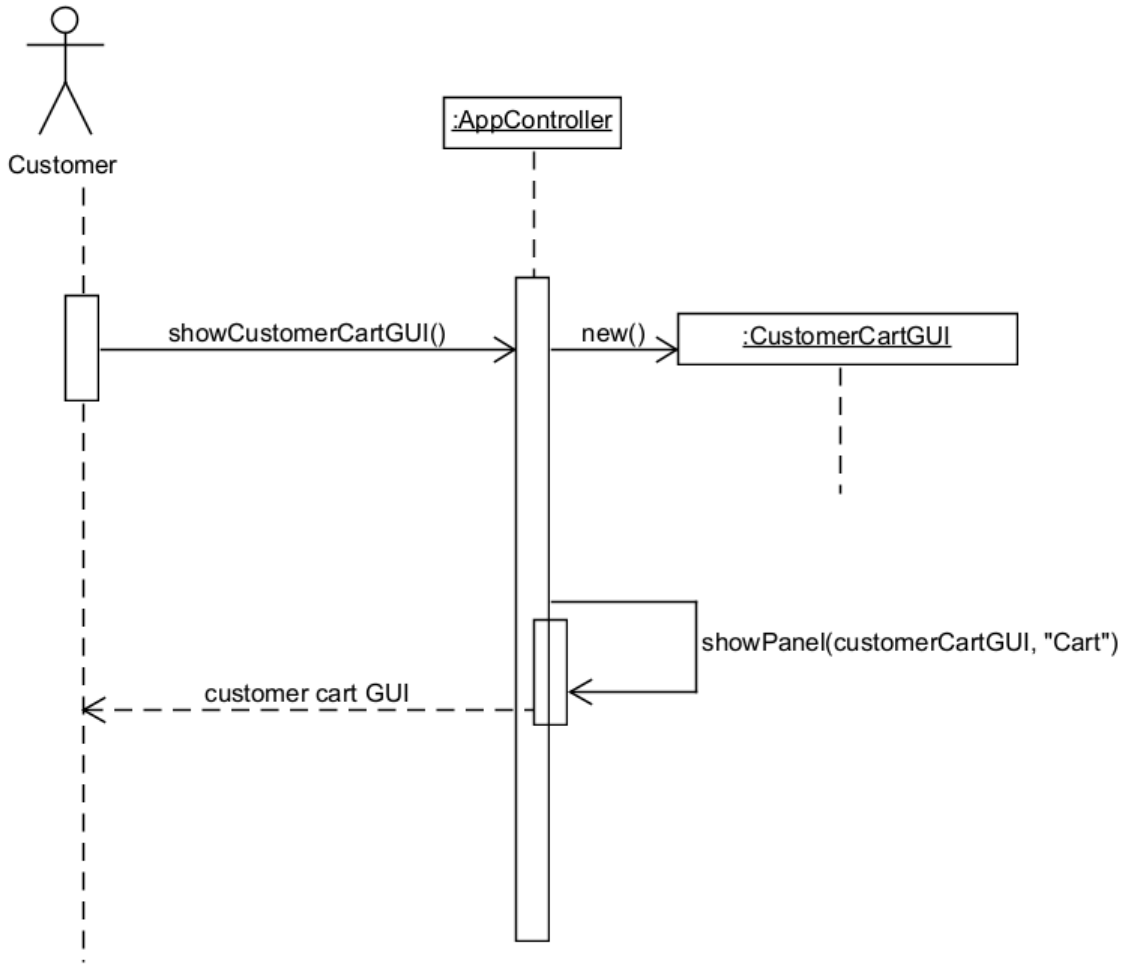
Add to Cart


Customer
btnAdd:JButton
:ActionListener
menuList:JList
click()
actionPerformed(e)
getSelectedIndex()
menuIndex:int
menu:Menu
getAllMenuItems().get(menuIndex)
itemToAdd:MenuItem
:Customer
addToCart(itemToAdd)
add(item)
lblTotalPrice:JLabel
setText(txt:String)

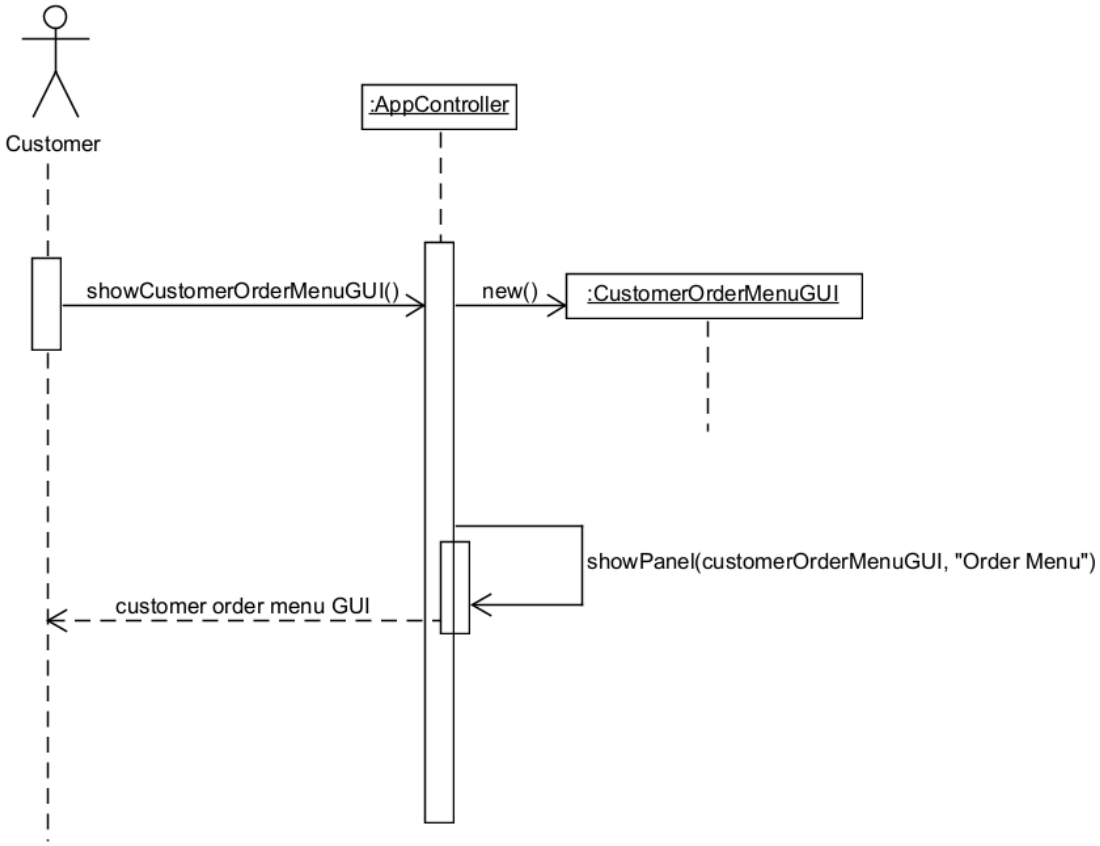
Search Menu

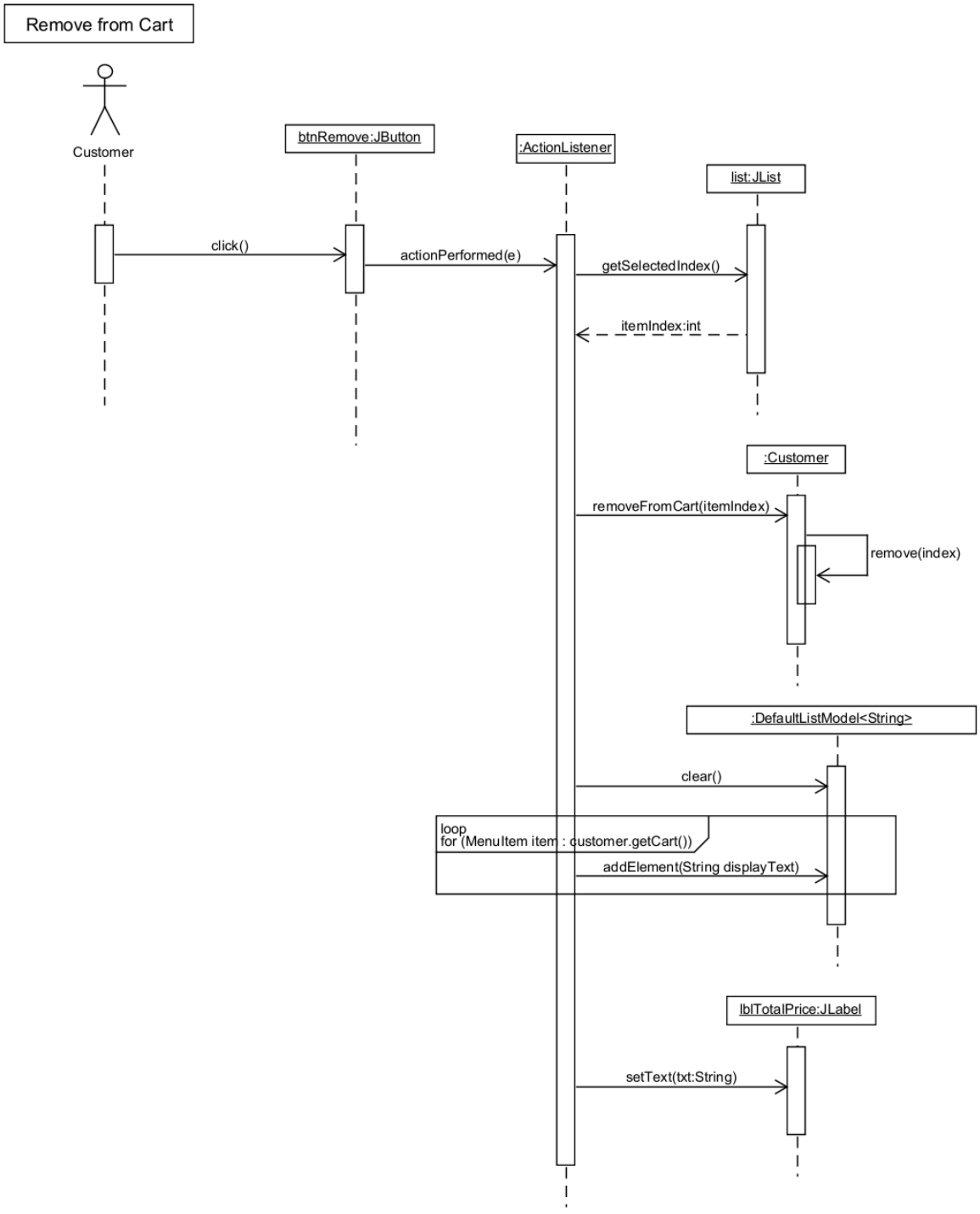

Customer
btnSearch:JButton
:ActionListener
:JTextPane
click()
actionPerformed(e)
getText()
query
:DefaultListModel<String>
clear()
loop
for (MenuItem item : menu.searchMenu(query))
addElement(String displayText)

View Cart

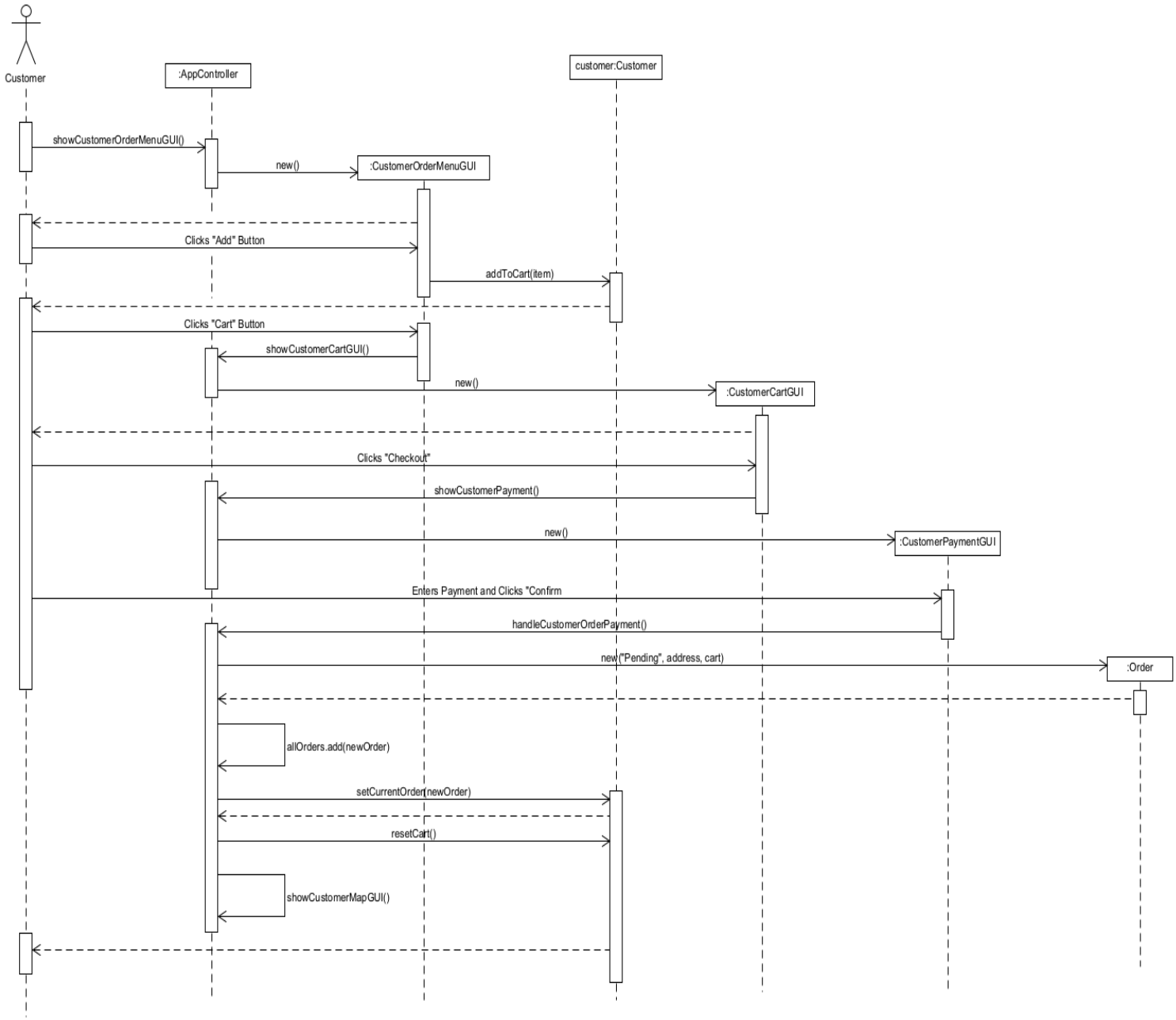

Customer
:AppController
showCustomerCartGUI()
new()
:CustomerCartGUI
showPanel(customerCartGUI, "Cart")
customer cart GUI

View Menu

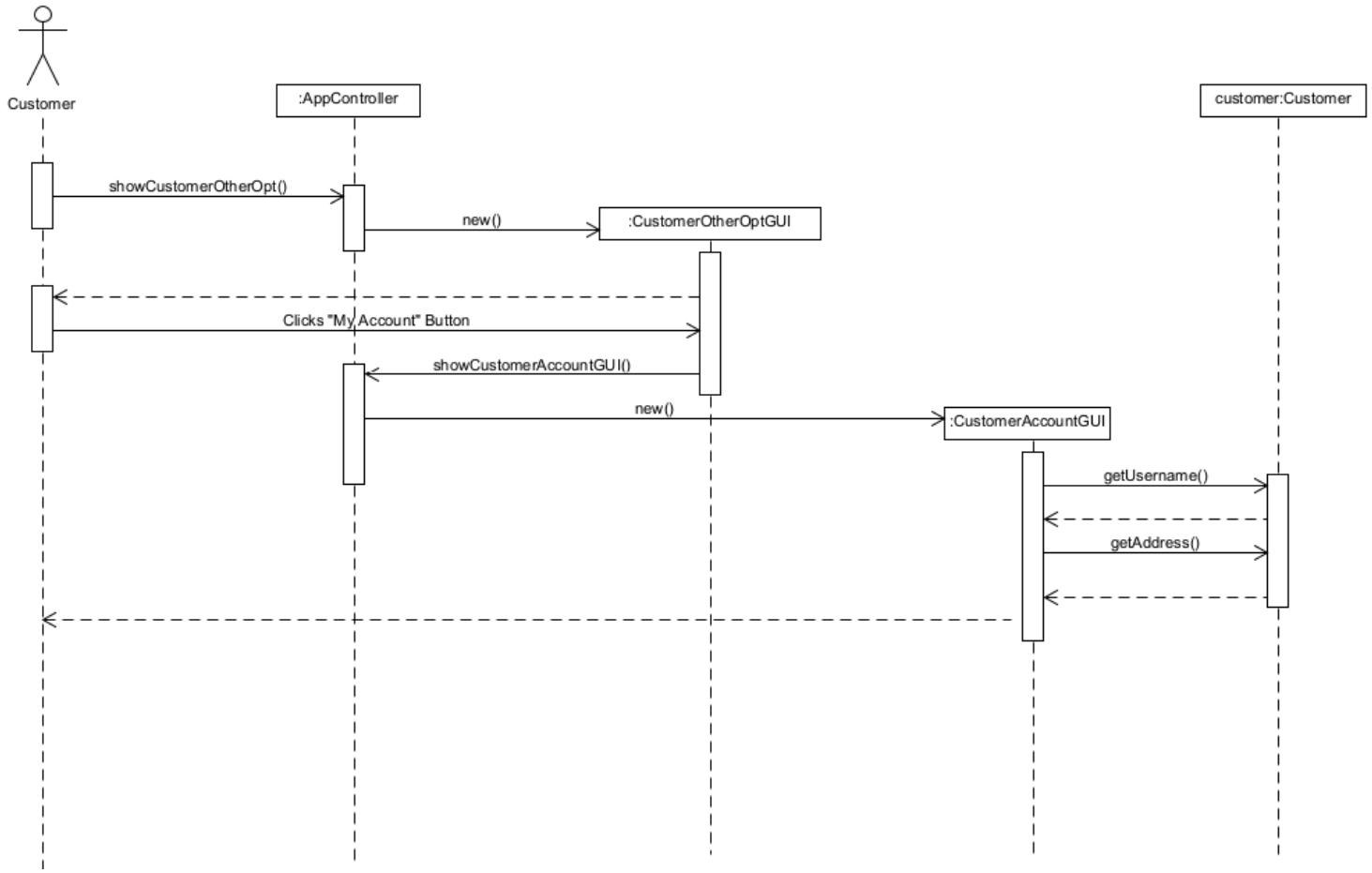

Customer
:AppController
showCustomerOrderMenuGUI()
new()
:CustomerOrderMenuGUI
showPanel(customerOrderMenuGUI, "Order Menu")
customer order menu GUI


Remove from Cart
Customer
btnRemove:JButton
:ActionListener
list:JList
click()
actionPerformed(e)
getSelectedIndex()
itemIndex:int
:Customer
removeFromCart(itemIndex)
remove(index)
:DefaultListModel<String>
clear()
loop
for (MenuItem item : customer.getCart())
addElement(String displayText)
lblTotalPrice:JLabel
setText(txt:String)

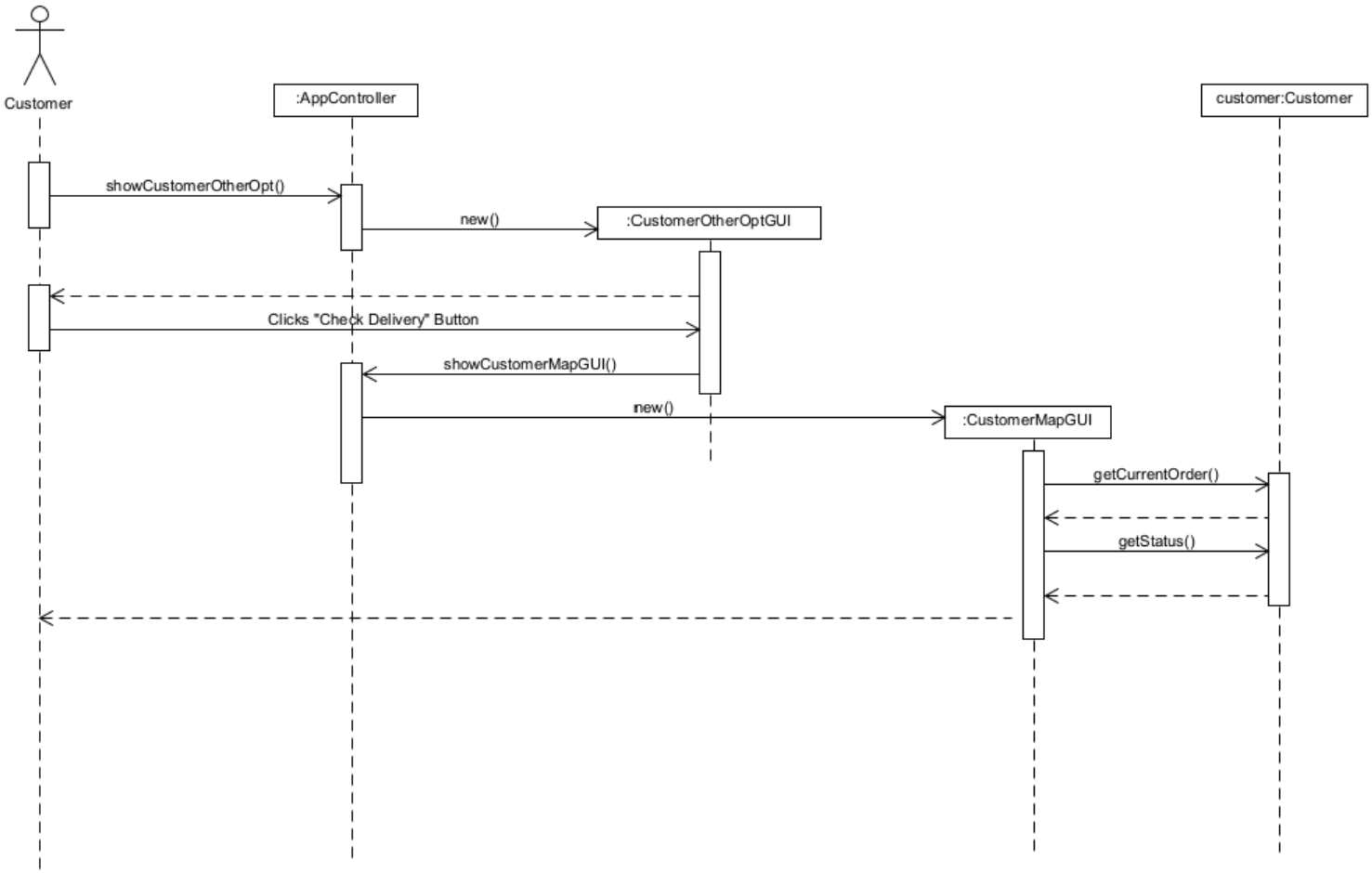
Make Order


Customer
:AppController
customer:Customer
showCustomerOrderMenuGUI()
new()
:CustomerOrderMenuGUI
Clicks "Add" Button
addToCart(item)
Clicks "Cart" Button
showCustomerCartGUI()
new()
:CustomerCartGUI
Clicks "Checkout"
showCustomerPayment()
new()
:CustomerPaymentGUI
Enters Payment and Clicks "Confirm
handleCustomerOrderPayment()
new("Pending", address, cart)
:Order
allOrders.add(newOrder)
setCurrentOrder(newOrder)
resetCart()
showCustomerMapGUI()

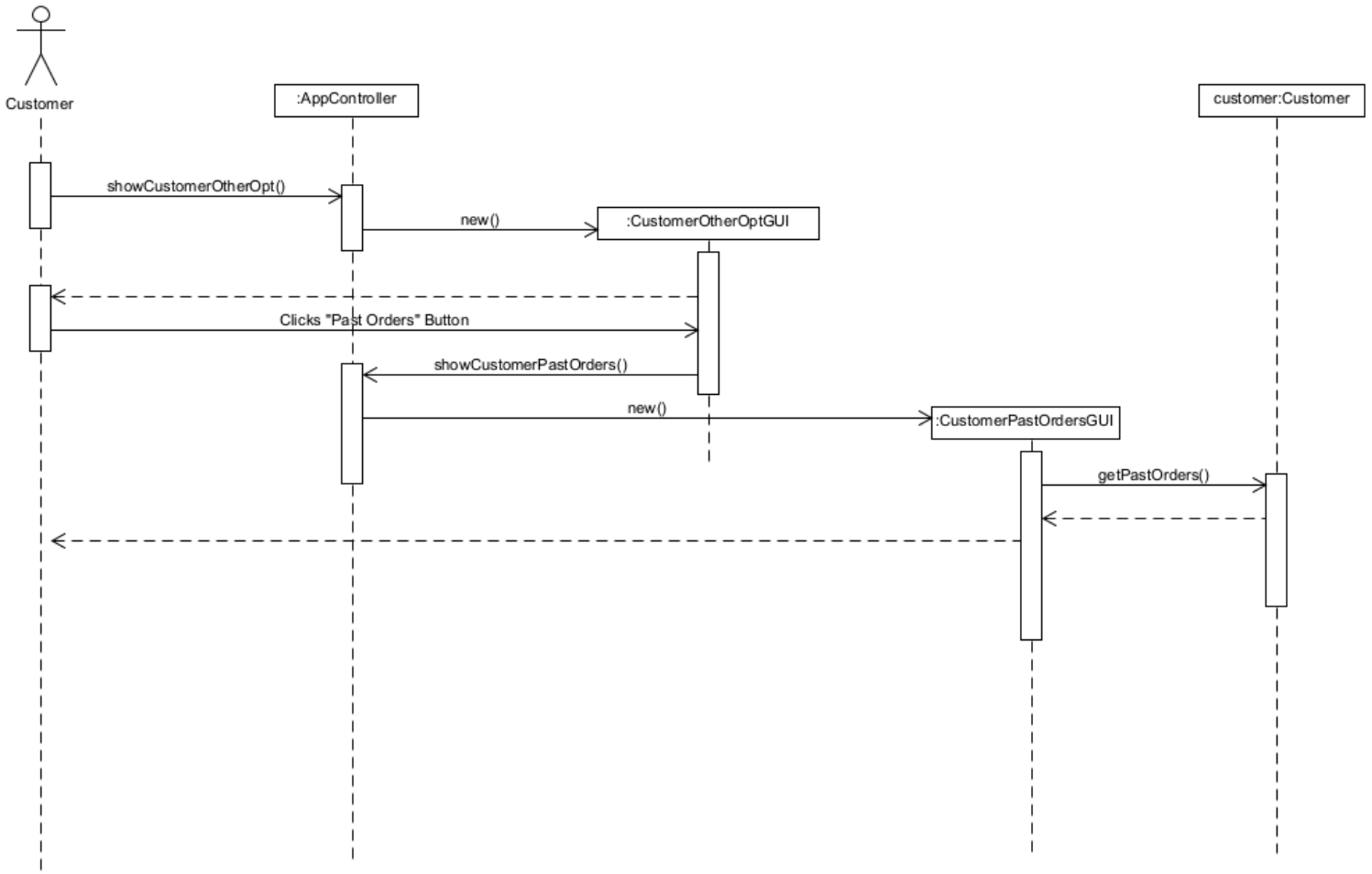
View Account



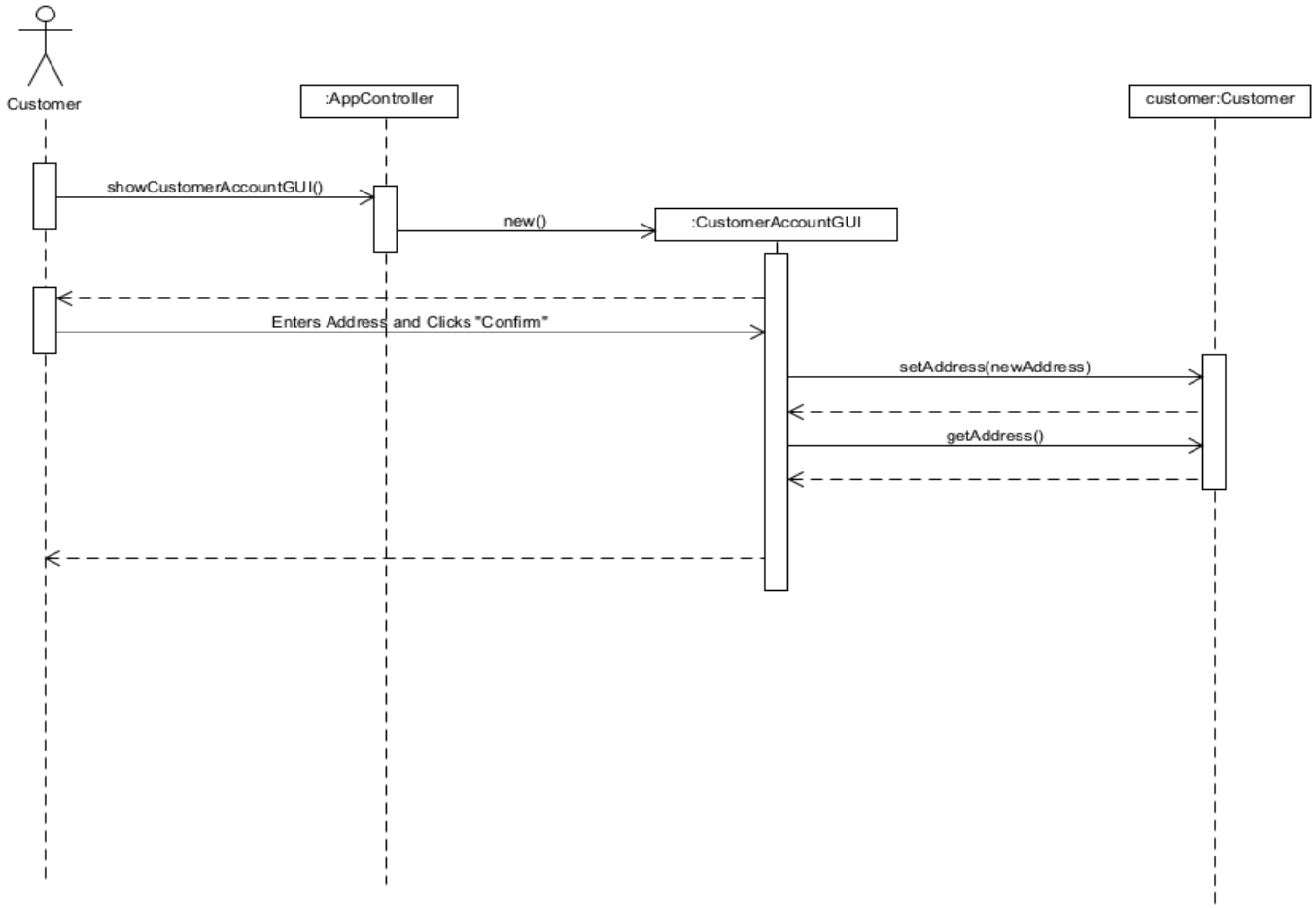
Check Deliveries



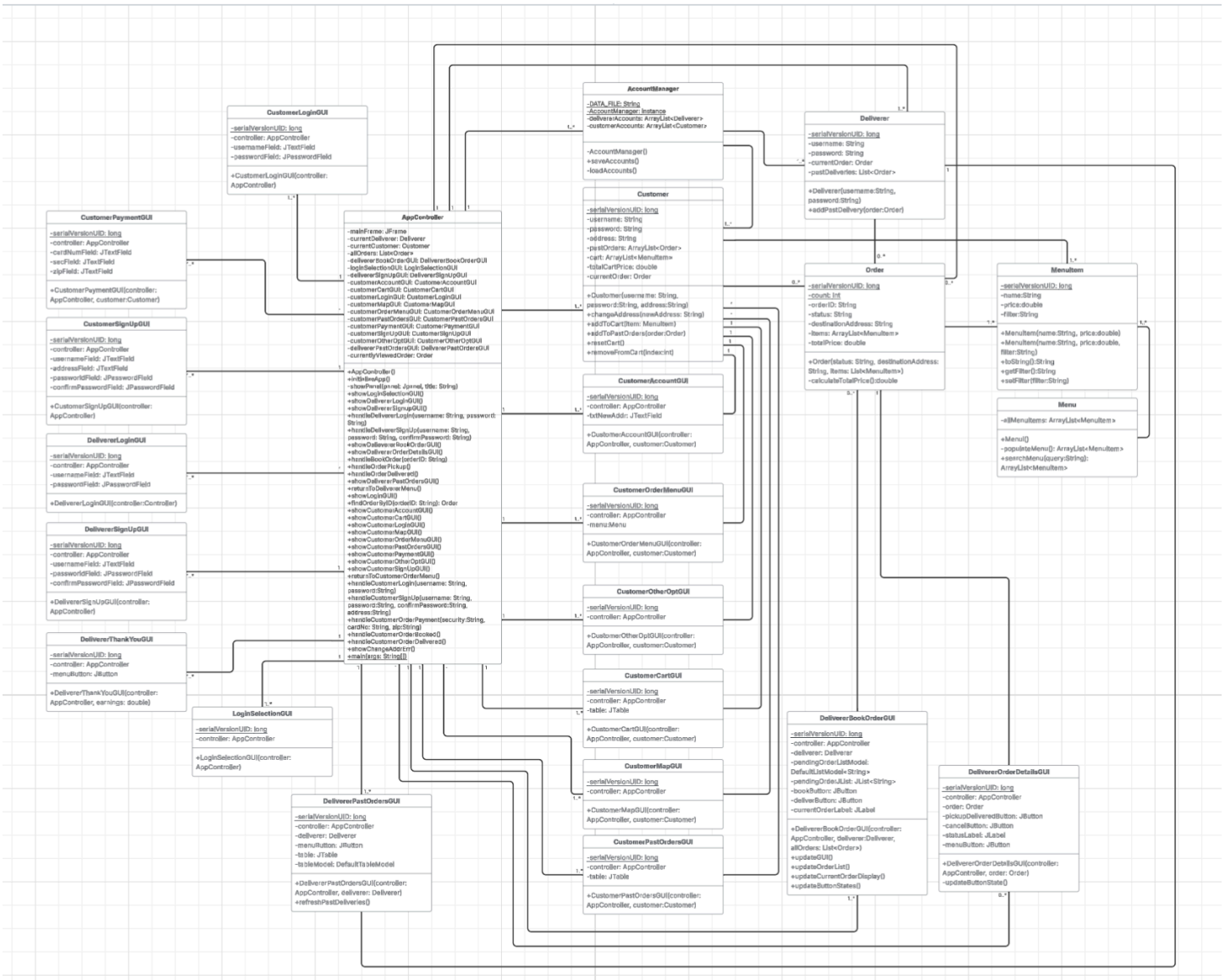
View Past Orders



Change Address



Class Diagram:
https://lucid.app/lucidchart/cbd559b6-3b72-4dc3-ae56-e77e5a291a0a/edit?viewport_loc=-5868%2C-1086%2C7993%2C3468%2CHWEp-vi-RSFO&invitationId=inv_4507711c-d80b-4d58-8d88-f3180af8a60f to full version for zooming in.



Conclusion:

For this project overall, there were no mounting difficulties faced by our team. However, it is important to note that each of our team members worked in different time zones, making it difficult for us to meet on call, though we were able to remedy that by mainly communicating our progress through Discord. Personally, I (Jasmine) had difficulties with the sheer amount of diagrams we had to make, considering we needed to create a sequence diagram for each of the use cases. The implementation had its own set of challenges, as we attempted to follow the initial diagrams we had conceived, but found it increasingly difficult during implementation, constantly causing us to revisit the diagrams and making them more complicated. Each member also had external responsibilities while working on the project, making time management difficult.

Below is a more detailed work summary:

- Kylie Lin:
  - Ideation
  - Project Proposal Write Up & Edits
  - All Deliverer side implementation & Debugging
  - Customer Sequence Diagrams: Make Order, View Account, Check Deliveries, View Past Orders, View Account Details, Change Address
  - Implemented AccountManager, AppController, LoginSelection, Order, and MenuItem
  - Final Deliverer Use Case Diagram
  - Final Customer Use Case Diagram & Edits
- Jasmine Fan:
  - Ideation
  - Project Proposal Deliverer & Customer Wireframe (Drafts)
  - All Deliverer Sequence Diagrams
  - Customer Sequence Diagrams: View Menu, Sign In, Sign Up, Change to Delivery Sign In, Search Menu, View Cart, Remove from Cart, Add to Cart
  - All Customer side implementation & Debugging
  - Implemented Menu, AccountManager, and AppController
- Ray Ochotta:
  - Ideation
  - Project Proposal Write Up & Edits
  - Class Diagram
  - Final Customer Use Case Descriptions, Diagram & Edits

Overall, some recommendations for the final project would be to…

- Have better communication (hopping on video calls, etc.). We had some problems with this because of our different time zones, but this could have been alleviated by allowing for more in-class breakout rooms. This happened in the final class, which was nice. I (Ray) would have liked one or two more of those sessions as a brief part of the class, especially during the workshops.

- Conceiving a smaller concept at first (pre-submission of the proposal), *then* expanding– rather than the other way around.
- Write more descriptive explanations as to what each screen should look like and do
- Learn UML Diagrams & OOP Design Principles long before doing the final project– the course should start with learning about these concepts before diving into coding (meaning start the semester with UML Diagrams & OOP Design Principles, then code with C++, then code with Java). Learning these crucial topics at the end of the course feels awkward when reflecting back on previous assignments. This especially feels strange because we don't even have an assignment to utilize the design principles on. Our project was ideated and locked in before we covered any design patterns in class.