Restaurant Point-of-Sale System

Seth Williams
Katie Barbaree
Zack Weld
William Jackson
Alexei Pillen

Table of Contents

Concept Statement

Design a computer system for a restaurant. The computer system must give functionality to the restaurant and make employees jobs easier. The systems main task is to handle customer orders. A server of the restaurant will take customer's order at a table. The server will make a list of the items the customers of the table request. The server will then be able to go to the computer system and log in with their employee ID. After logging in the server will have access to create a new bill. When the new bill option is selected the system opens a new window that the server will be able to enter the items the customer has ordered. While entering the items the system creates a list of the items and matches them with their price. The ingredients of the item may be out of stock so an error message should be displayed. If an item is not on the menu an error message is displayed to the server that awaits acknowledgment from the server by pressing the ok button. The server should then be able to continue entering items into the system. Once the bill is complete the server should be able to save the bill on the system. Once the save button is pressed the system calculates the total price of the bill by adding up the individual prices of the items. It then should calculate the sales tax according to the total and add that to the total. The new total value with tax included is saved separately from the total value without tax. After the totals have been calculated and added to the bill the system then displays a message to the server that the bill has been saved successfully. After acknowledging the prompt by pressing a button on the system the system returns to its main menu. If any errors occur during the calculation of the totals or the saving process an error message should be displayed to the server that prompts them of the error. After acknowledging the error by pressing a button on the system the system returns to the main menu and the bill is not saved.

When the customer is finished with their meal the employee should be able to print out the final version of the customer bill. If the customer notices an incorrect charge, like an item that was not ordered, a manager should be able to delete that item from the bill within the system and print out a correct bill. After reviewing the bill the customer should be able to write in a tip amount and then pay for the bill in a number of different ways such as: gift card, cash, credit card, debit card, coupon, or any number of combinations of these payment methods. The system should be able to handle if payment is made with part cash and part card for example by allowing the server to enter the amount paid for in cash or card. The system should then subtract that amount from the total amount and return what is left. After the bill has been paid for in the system a receipt should be printed that reflects the customer's payment method the total they have paid before tax, with tax, and with tip all on different lines of the receipt. When the receipt is signed for by the customer the bill should then be updated and marked as closed in the system by the server.

Besides handling customer payment, the system should also handle when an order has been submitted into the system. The system should update that an order has been placed and is

waiting for kitchen staff to start preparing the order. When the kitchen staff has started their preparations the system should be updated by the server that the order is being prepared. Once the order has been completed the system should be updated that the order is ready to be delivered to the customer at the table. Once the order is delivered to the customer the order in the system should be marked as complete. If the customer is not satisfied or would like to order something additional then a new order request must be submitted within the system following the same process as before.

When a customer first enters the restaurant they should be greeted by a hostess before being seated. The hostess should be able to check the system and see if there are any tables available that will satisfy the amount of seats the customer needs. If there are no tables the customer should be told to wait. The system should be able to give this information to the hostess that is interacting with the customer. The hostess should be able to see the layout of tables and view what tables are available and what tables are occupied. In addition to this, the hostess should be able to see how long the tables have been occupied in order to be able to give the customer an approximate time before a table will be available. When a table becomes available the hostess will lead the customer to the table. Once the customer has taken the table the hostess will update the system that the table is taken and the timer for how long that table is taken begins. A table is updated in the system that it is available when a busboy clears the table of any leftover food or dishes the customer left after their meal. Once the table is cleared by the busboy the server of the table should update the system that the table is available.

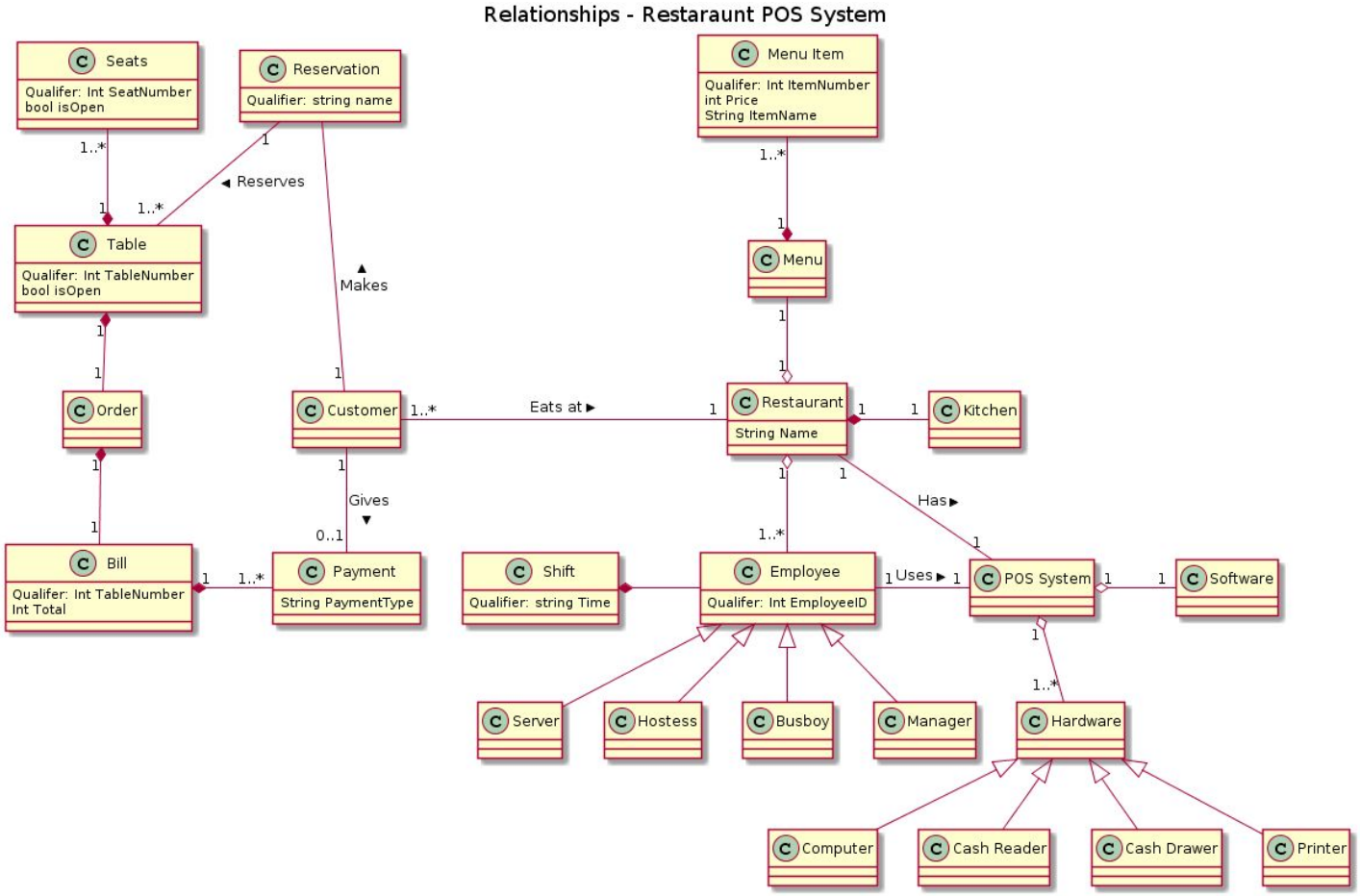
The system should also handle call in reservations. If a customer calls into the restaurant the customer should be greeted by the hostess. The customer should tell the hostess how many people will need to be seated as well as a time and date for the reservation. The hostess would be able to see in the system if there are any tables available according the criteria the customer is given. If there are no table available, the customer would be required for providing a different time or date. If a table is available, the hostess should be able to update the system that the table will be reserved at least thirty minutes prior to the customer reservation time in order to prevent the table being taken when the customer arrives. The table reservation should also be associated with the customer name given to the hostess over the phone. If the customer by that name does not show up for the table before fifteen minutes after the reservation the system should automatically update the system that the table is now available. If the customer arrives for their reservation the hostess will then update the table as being occupied the same way as when seating a walk in customer. The table editing process will then proceed as normal.

The system should also keep an updated menu. This menu can be changed by the kitchen staff. Items can be added or removed from the menu if the ingredients for the item is in stock or not in stock. New items can be added to the menu if a kitchen staff member is able to prepare that item. The system should keep all menu items in a list.

The system should also be able to keep track of employee's clock in time and clock out time. When an employee arrives to work they should be able to sign in with their employee ID number and select an option that says clock in. This starts a timer that keeps track of how long the employee has been at work. If the employee goes on lunch break or is not working, they should update the system by selecting a break option. The clock in timer is stored and put on hold while a new break timer is started. The break timer is stopped when an employee returns to work and selects the off break option within the system. The system should also handle when an employee is clocking out for the day. When an employee clocks out they should be able to take

the amount of tips they have earned for the day and select the clock out option. When the clock out option is selected the timer is stopped and stored within the system for amount of time the employee has worked that day. The timer for each employee daily work amount can be viewed by a manager. The manager can then clear the timers once they have reviewed the employee's time worked and have made a note of the amount of money they have earned for that day.

Domain Class Model



Constraints

Context: Bill
Inv: self..NumberofPayments >= 1

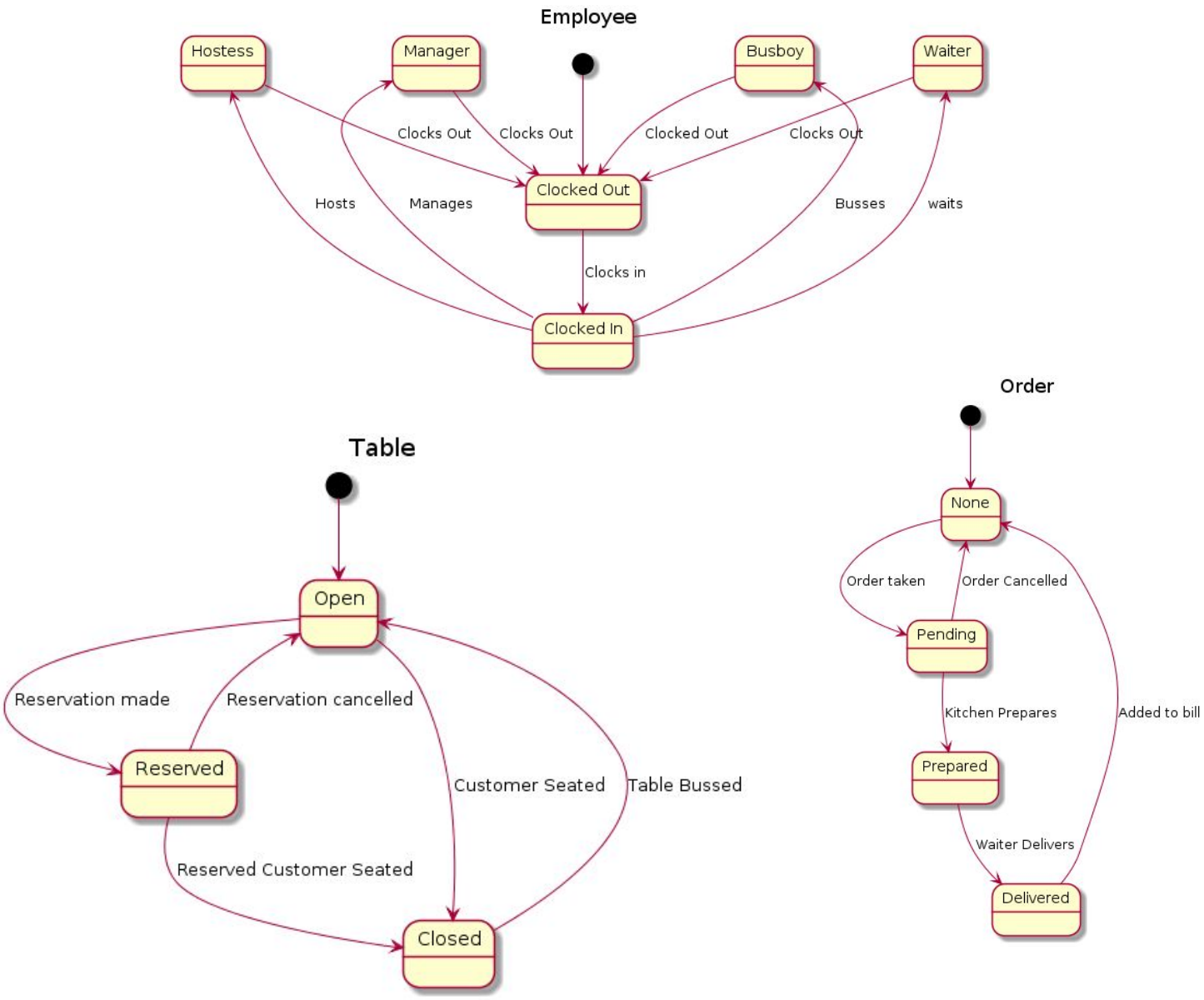
Context: Employee
Inv: self.id > 0

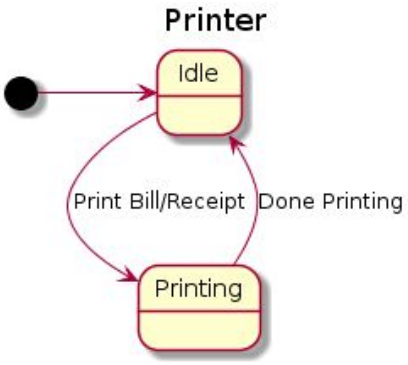
Context: Seats
Inv: self.number > 0

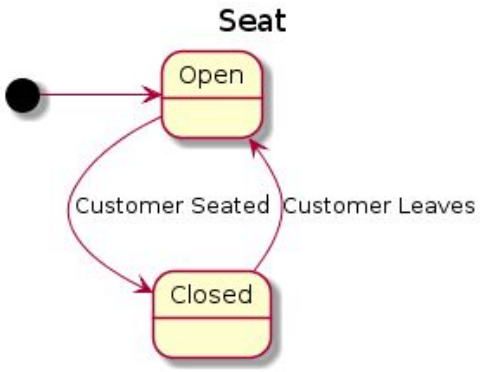
Context: Table
Inv: self.number > 0

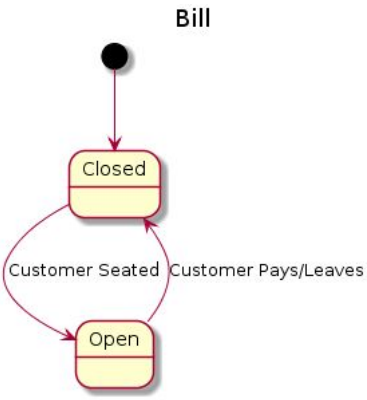
Context: Reservation
Inv: self.name != null

Domain State Models




Printer
Idle
Print Bill/Receipt
Done Printing
Printing


Seat
Open
Customer Seated
Customer Leaves
Closed


Bill
Closed
Customer Seated
Customer Pays/Leaves
Open

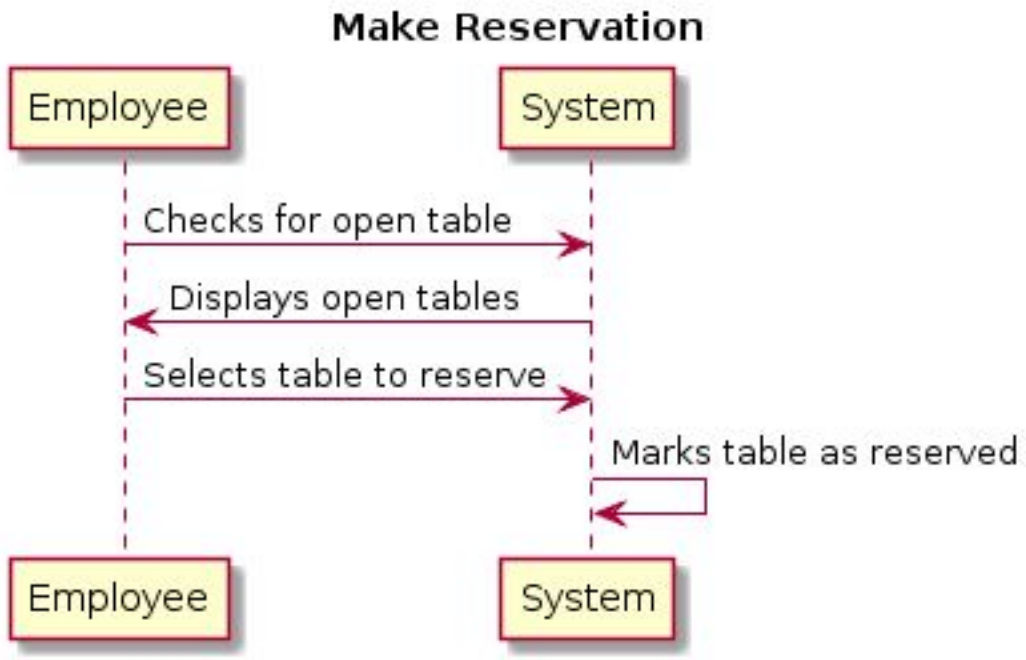
Use Cases

Make Reservation

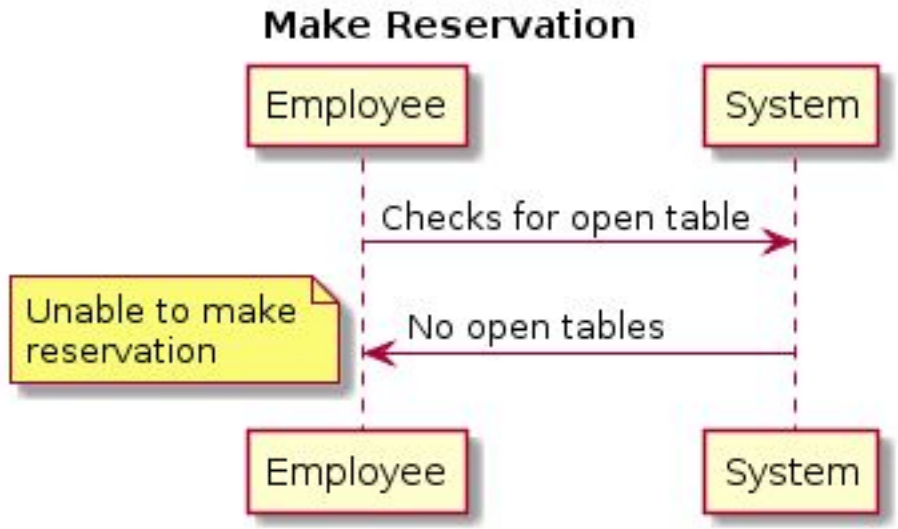
Use Case: Make Reservation
Summary: Customer makes a reservation of a table at a specific date and time.
Actors: Employee
Precondition: Restaurant is open and hostess receives reservation request

Actor Intentions	System Responsibility
1. Employee receives reservation request	
2. Employee checks POS System for an open table at requested time	3. System displays all tables marked as being available for given time slot.
4. Employee reserves a table	5. System marks table as resrved at giving time slot.

Post condition: Designated Table marked as closed
Alternative Courses:
Step 3 – All tables are full and no tables open to reserve
Step 4 – Employee unable to make reservation
Scenario:
Employee receives a request for a reservation
Employee Checks POS System for open table
POS System displays open tables at specific time
Employee reserves a table
POS System marks table as reserved



Alternative Scenario:
Hostess Receives request for reservation
Hostess checks POS System for open tables
POS System has no open tables at time to display
Hostess unable to make reservation



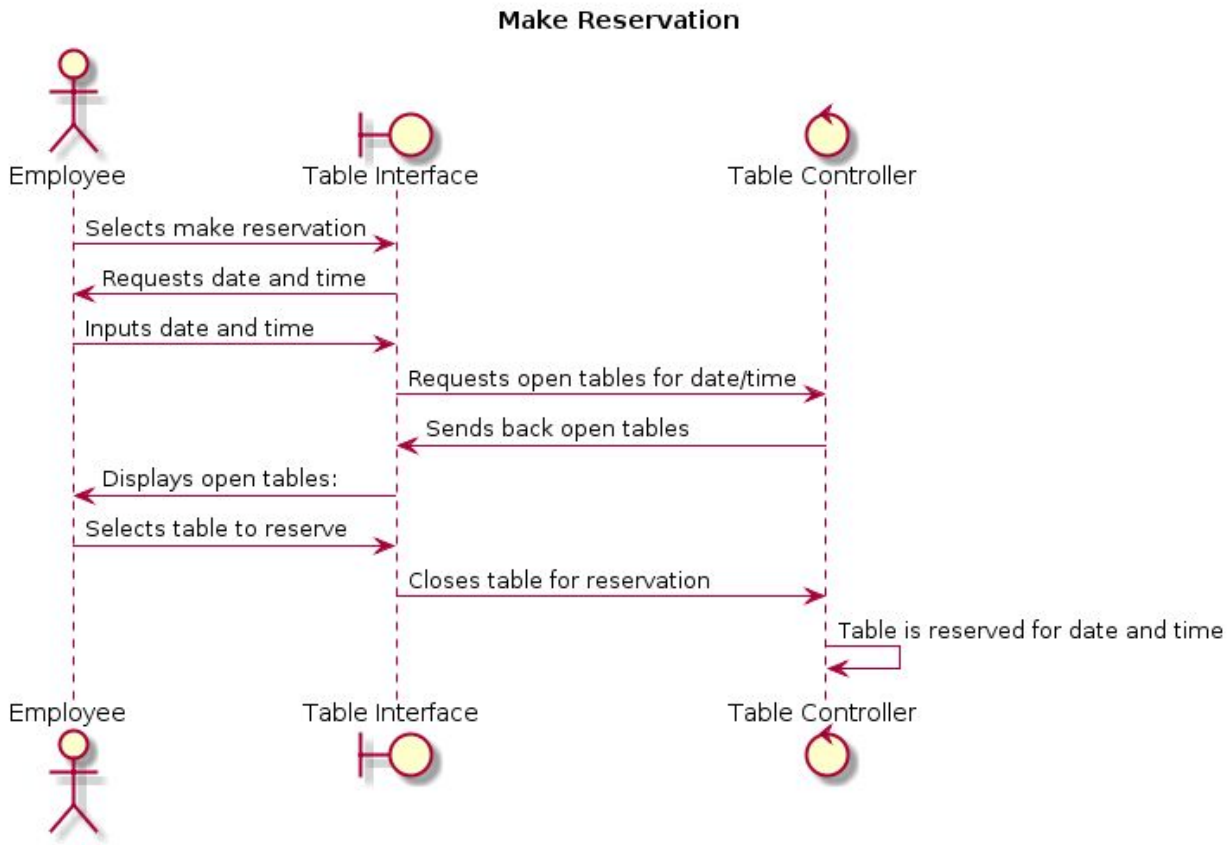
Use Case: Make Reservation

Summary: Customer makes a reservation of a table at a specific date and time.
Actors: Employee
Precondition:
Restaurant is open
Hostess receives reservation request
Reservation request is during restaurant hours of operation

Actor Intentions	System Responsibility
1. Employee receives reservation request	
2. Employee selects to make reservation	
3. Employee inputs date and time for reservation	
	4. System displays tables which are open for given date and time and their number of seats
5. Employee selects a table that was displayed for the reservation, that has enough seats to fit size of reservation party.	
	6. System receives input of table for reservation
	7. System marks table as reserved for specified date and time

Exceptions:
No open tables: After the employee inputs the date and time that was requested for the reservation on step 3, the system is unable to display available tables in step 4 because there are no tables open at given time. The Employee is then unable to make the reservation.
Party too large for single table: After the System displays the tables which are open at given date and time in step 4, the employee must select a table that is large enough for the part in step 5. If the party exceeds the size of a single table, the employee must reserve multiple tables in order to accommodate the size of the party in step 5. In step 6 and 7 the input would be for more than one table and all tables involved would be marked as closed for specific date and time.

Post condition: The table which was designated by the employee for a reservation is marked as closed for the specific date and time requested.



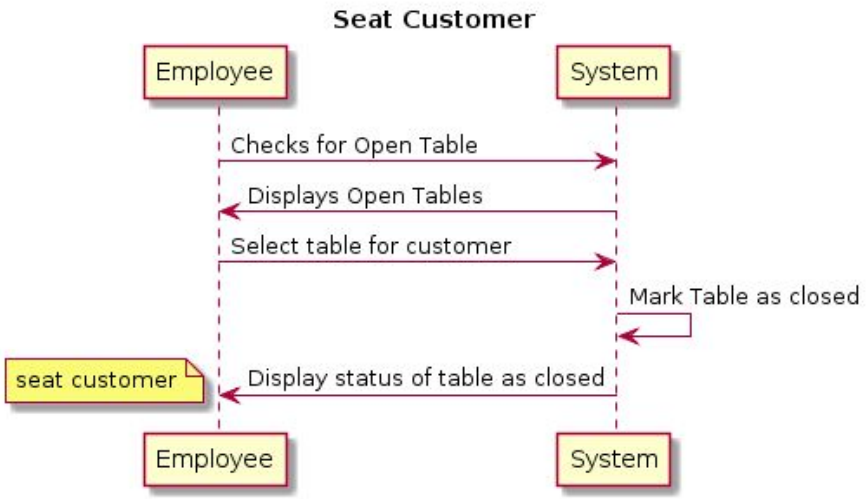
Seat Customer

Use Case: Seat Customer
Summary: Employee seats customer at a table
Actors: Employee
Precondition: Customer requests to be seated

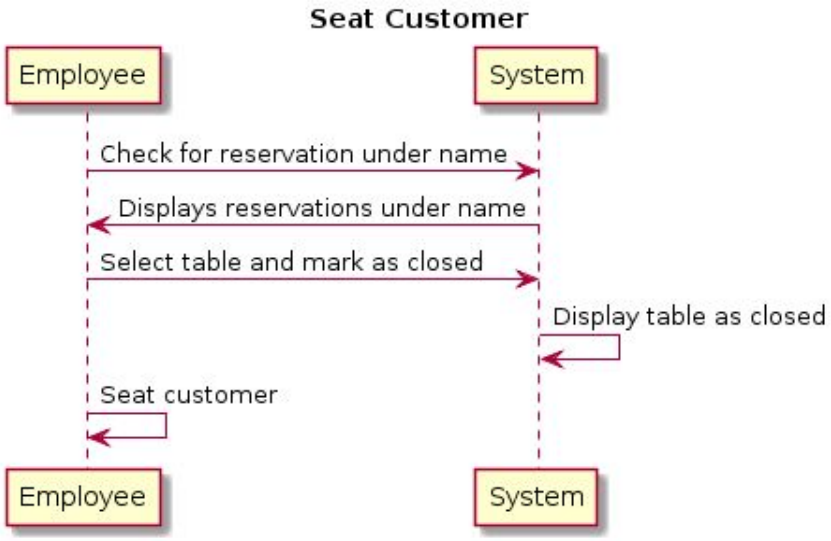
Actor Intentions	System Responsibility
1. Employee checks to find open table	
	2. System displays tables that are currently open to have a customer seated
3. Employee selects table for customer to be seated at	
	4. System marks table as closed
5. Employee seats customer at table.	

Alternative Courses:

No tables open: On step2 the system has no tables that are open for a customer to be seated.
Reservation: Step 1 Employee would search for name in system to find reservation. Step 2 would display the table that was reserved and employee would then seat the customer. Table marked as closed.
Post Condition: Table in which customer was seated is marked as closed
Scenario:
Request to be seated is made
Employee checks system for open tables
System displays tables to the Employee
Employee selects table for customer to be seated at
System marks table as closed
Employee seats customer



Alternative Scenario:
Customer requests to be seated at previously made reservation
Employee checks system for reservation under name
System displays table which was reserved under name
Employee seats customer marks table as closed.



Use Case: Seat Customer
Summary: Employee seats customer at a table
Actors: Employee

Precondition: Customer requests to be seated

Actor Intentions	System Responsibility
1. Employee selects for open tables from system	2. System displays tables that are open at current time
3. Employee selects open table from screen	4. System receives input and changes status to closed
	5. System removes tables from list of open tables
6. Employee seats customer at designated table.	

Exception:
Reservation: Customer had a previous made reservation when he requests to be seated. Step 1 Employee would select find reservation rather then for open tables, and would input name of the reservation. Step 2 system would display reservations under that name. Step 3 Employee would select the correct reservation, rather than table. Step 4 System would show for which table that reservation is for. System marks table from reserved to closed in Step 5
No open Table: Step 2 the system would be unable to display any open tables at specific time Step 3 employee would be unable to seat customer.
Multiple Tables: In either a reservation scenario or normal there may be more then one table to mark as closed once customer is seated if party size requires multiple tables.

Post Condition: Table(s) in which customer is seated are marked as closed.

Clear Table

Use Case: Clear Table

Summary: The employee clears the table.

Actors: Employee

Preconditions: The customer(s) at the table have paid their bill(s) and left.

Description:

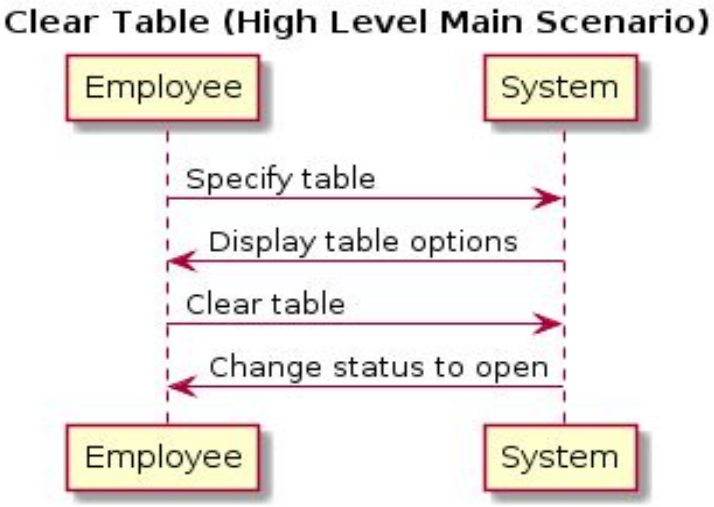
Actors	System
1) The employee specifies the table they want to clear	
	2) Displays options for that table
3) The employee selects the option to clear the table	
	4) Changes table's status to available

Alternative Flow:

Table already clear – If the employee tries to selects clear table when a table is already cleared, a message will appear to notify the employee of the error.

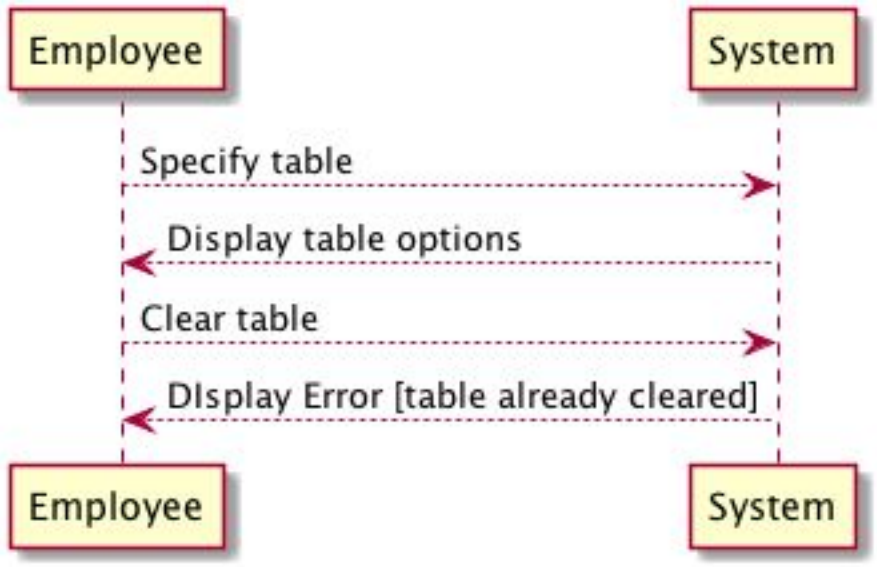
Postconditions:
The system clears the desired table.

Normal Scenario:
Employee enters table to clear.
POS System shows options for that table.
Employee selects option to clear table.
POS System marks table as available.



Alternative Scenario

Employee enters table to clear.
POS System shows options for that table.
Employee selects option to clear table.
POS System shows error stating table already clear.



Use Case: Clear Table
Summary: The employee clears the table.
Actors: Employee
Preconditions: The customer(s) at the table have paid their bill(s) and left.
Description:

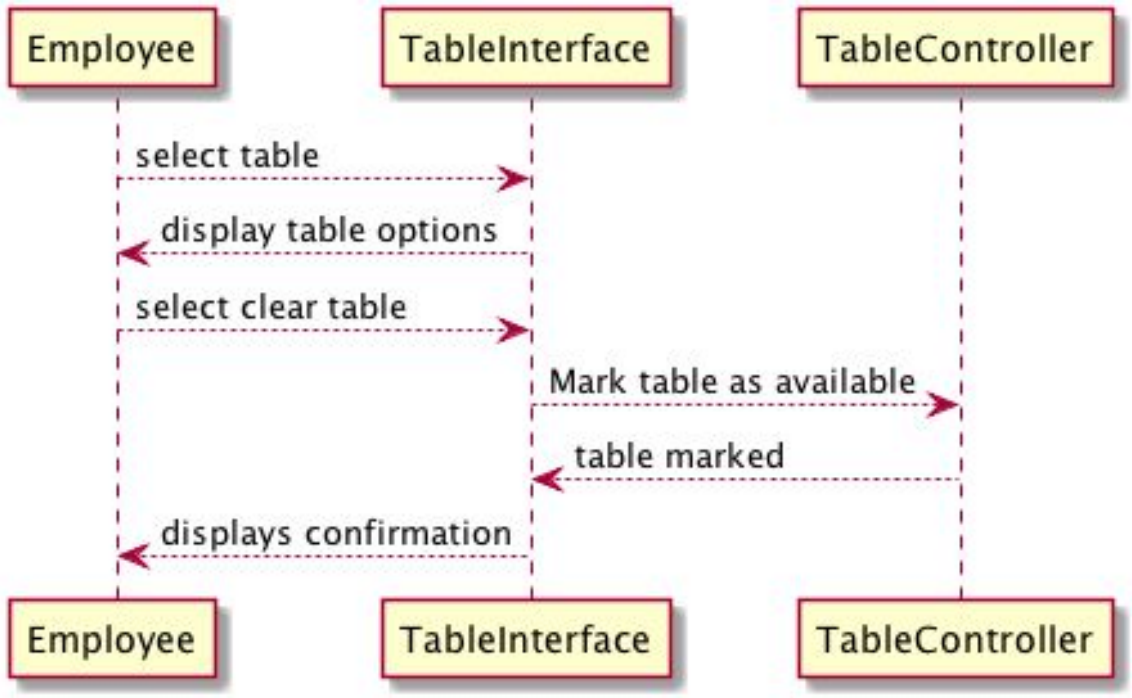
Actors	System
1) The employee selects the table where the customer is sitting.	2) Displays options for that table.
3) The employee selects option to clear table.	4) Clears any info from last customers and changes table status to available.

Alternative Flow:

Table already clear – If the employee tries to selects clear table when a table is already cleared, a message will appear to notify the employee of the error.

Postcondition:

The system clears the desired table.



Edit Menu

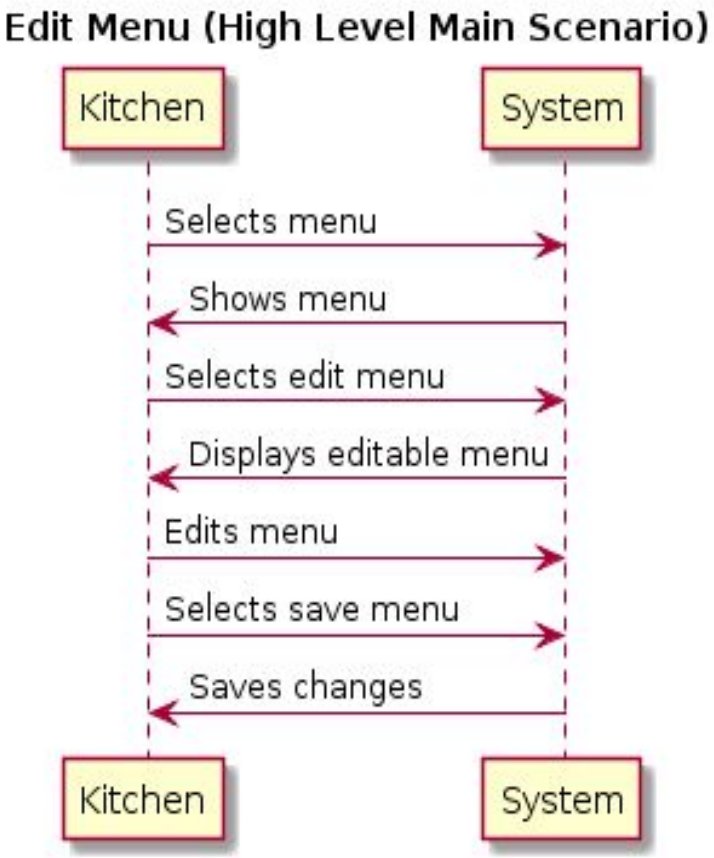
Use Case: Edit Menu
Summary: The kitchen edits the menu.
Actors: Kitchen
Preconditions: The kitchen has decided to change the menu.
Description:

Actors	System
1. Kitchen navigates to menu	
	2. Displays menu
3. Kitchen selects option to edit menu	
	4. Displays menu as editable text box
5. Kitchen edits menu in the displayed text box	
6. Kitchen selects the save menu button	
	7. Saves and reflects changes

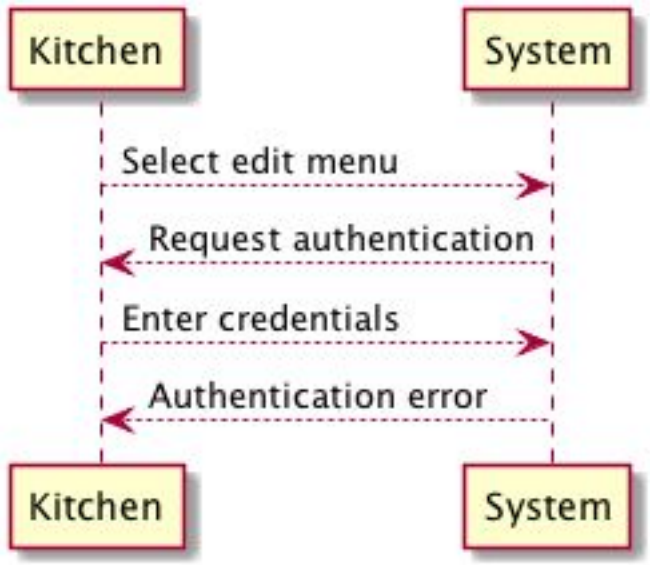
Alternative Courses:
Insufficient Permission: The employee does not have the required permission to edit the menu. An authentication error

Post conditions: The edited menu is saved.

Normal Scenario:
Kitchen selects menu.
System shows current menu.
Kitchen selects option to edit menu.
System shows menu as an editable text box.
Kitchen edits menu.
Kitchen selects option to save menu.
System saves menu and reflects changes.



Alternative Scenario:
Kitchen selects menu.
Kitchen employee enters credentials.
Kitchen employee lacks permission.
System sends an authentication error.



Use Case: Edit Menu

Summary: The kitchen edits the menu.

Actors: Kitchen

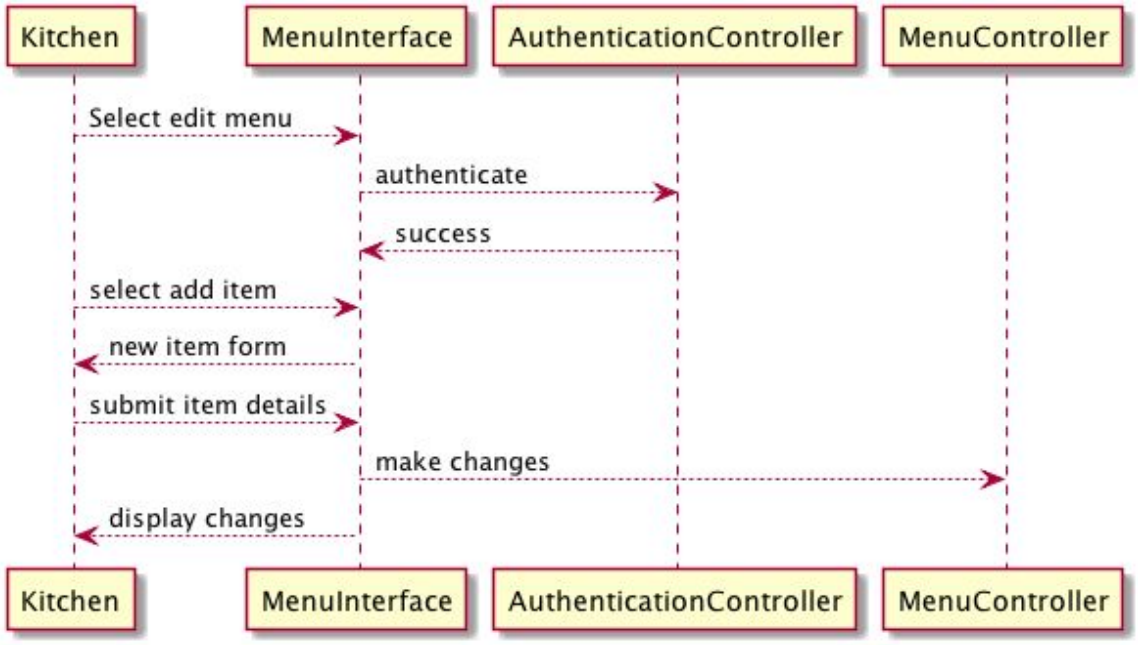
Preconditions:

Description:

Actors 1) Kitchen selects option to view menu 3) Kitchen selects option to edit menu 5) Kitchen edits menu 6) Kitchen selects save menu	System 2) Displays menu 4) Displays menu as editable text box 7) Saves new menu over old menu

Alternative Courses:
Insufficient Permission: The employee does not have the required permission to edit the menu. An authentication error

Post conditions: New edited menu is saved over old menu



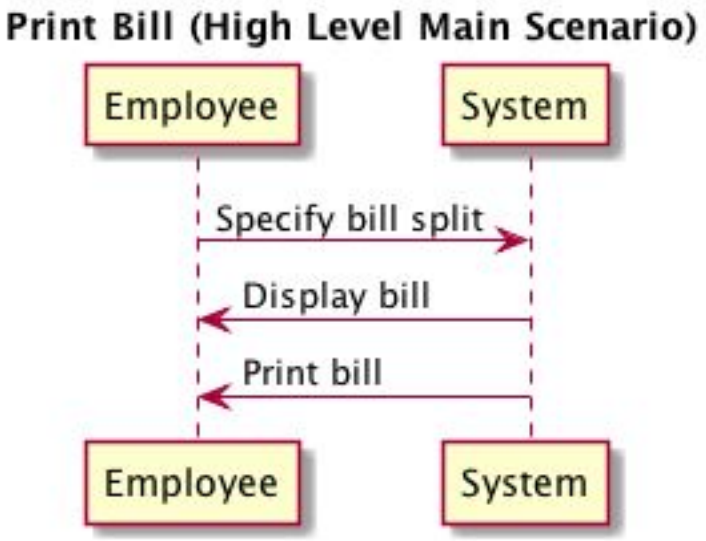
Print Bill

Use Case: Print Bill
Summary: The server prints the bill for a customer
Actors: Employee

Preconditions: Customers at a table have finished their meal have specified how they will split the bill.
Description:

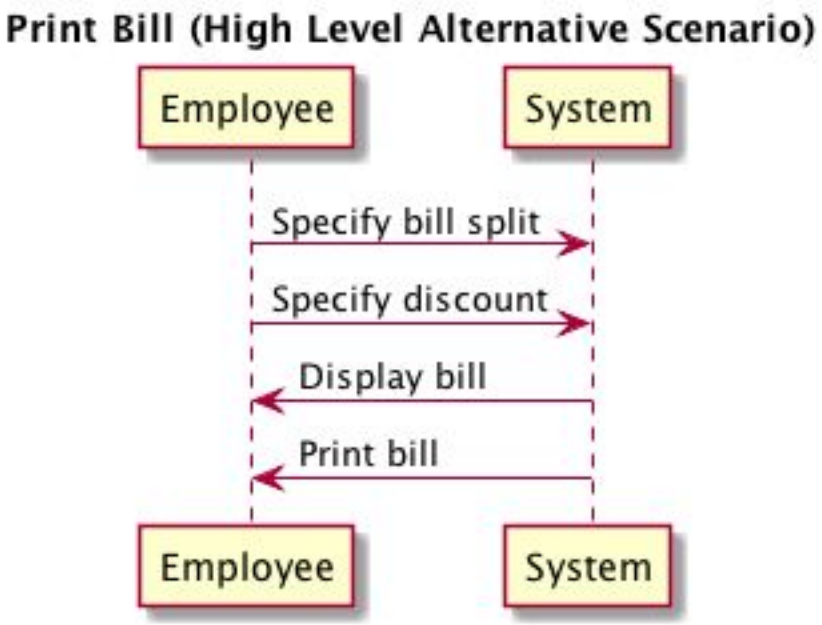
Actors	System
1. The employee specifies the table that wishes to check out. 2. The employee specifies the way the bill is to be split.	3. Each bill is calculated and printed out.

Exceptions:
Discounted Order: The customer receives a discount on the order. The discount is applied before the grand total is calculated. The name and amount of the discount is also printed on the bill.
Post conditions:
The employee prints out all bills for the table.
Normal Scenario:
Employee enters table to check out.
Employee enters how the bill will be split.
POS System displays each bill on the screen.
Each bill is printed out.
Employee brings bills to table.



Alternative Scenario:
Employee enters table to check out.
Employee specifies how the bill is split.

Employee enters discount from coupon.
POS System applies discount to specified customers bill.
POS System displays each bill on the screen.
Each bill is printed out



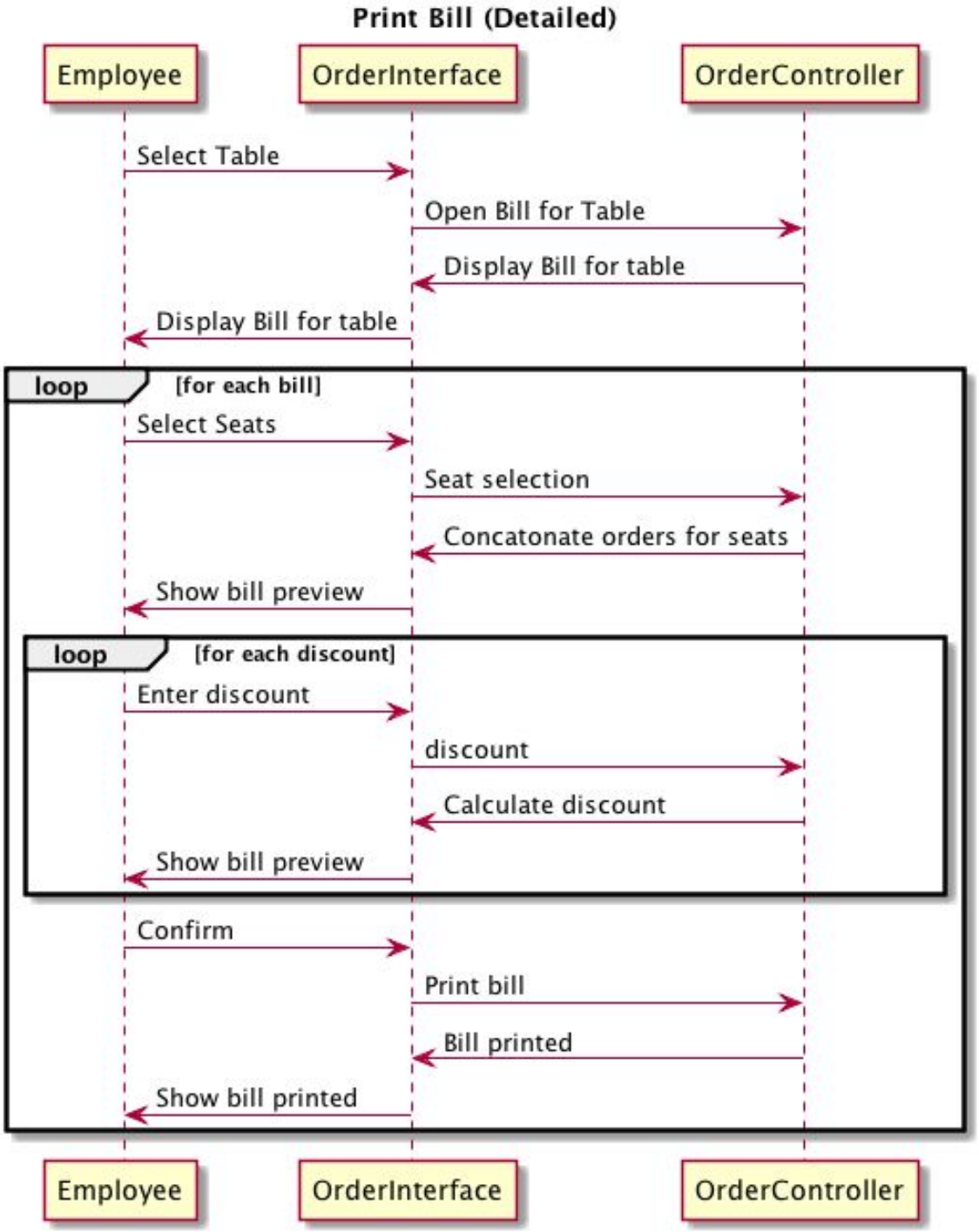
Use Case: Print Bill

Summary: The employee prints a bill for at least one seat. The bill contains all the items that were ordered and the cost.
Actors: Employee
Preconditions: The customers have ordered items and are ready to check out. The paying customer has specified which seats he will be paying for. The POS System has maintained the bill for each seat.
Description:

Actors	System
1) The employee selects table where customer is sitting.	
2) The employee selects the seats that the customer is paying for.	
	3) The bills for each seat are concatenated onto one bill.
	4) A grand total is calculated by summing all the subtotals and adding tax.
	5) A print preview is shown on the screen.
6) The employee verifies the print preview.	
	7) The bill is sent to the printer.
	8) The bill is printed out.

Exceptions:
Discounted Order: The customer receives a discount on the order. After step 3 the employee enters the discount as either a fixed amount or as a percentage on a group of items. The discount is applied before step 4. The name and amount of the discount is also printed on the bill.
Multiple bills per table: After step 8, the POS System notifies the employee that there are more bills to be printed for that table. The employee repeats the process from step 2 until all items ordered at the table have been paid for

Post conditions:
The bill has been printed and is ready to be brought to the customer for payment.


Print Bill (Detailed)
Employee
OrderInterface
OrderController
Select Table
Open Bill for Table
Display Bill for table
Display Bill for table
loop
[for each bill]
Select Seats
Seat selection
Concatonate orders for seats
Show bill preview
loop
[for each discount]
Enter discount
discount
Calculate discount
Show bill preview
Confirm
Print bill
Bill printed
Show bill printed
Employee
OrderInterface
OrderController

Take Table Order

Use Case: Take Table Order (Essential)

Summary: The server takes the order of customer(s) at a table.

Actors: Employee (Server)

Preconditions: Customer has been seated.

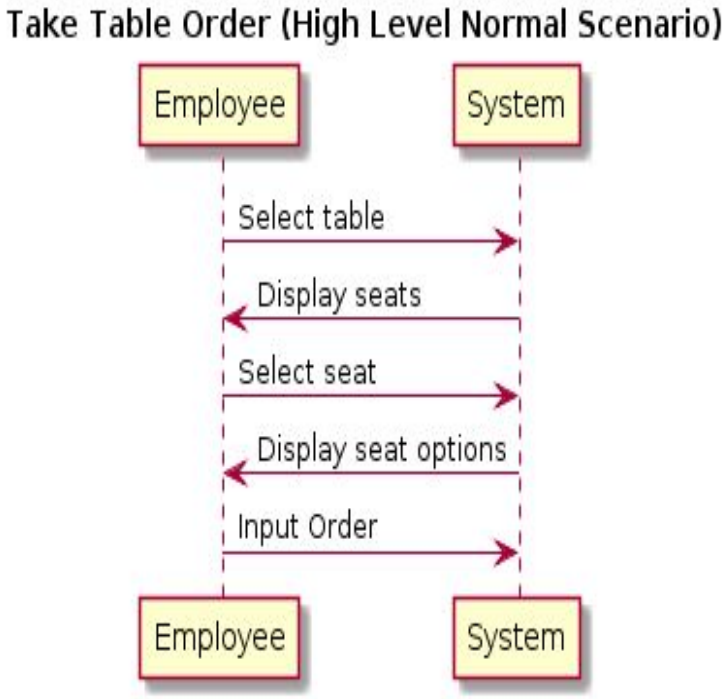
Description:

Actors	System
1) Employee takes the customer's order.	
2) Employee selects the customer's table	
	3) Shows seats at selected table.
4) Employee selects the customer's seat.	
6) Employee selects the option to create new order.	
7) Employee inputs new order	5) Shows customer's details

Exceptions:
Order for entire table: An order is being split among the entire table. Instead of selecting the customer seat the employee can enter a new order at the table level.

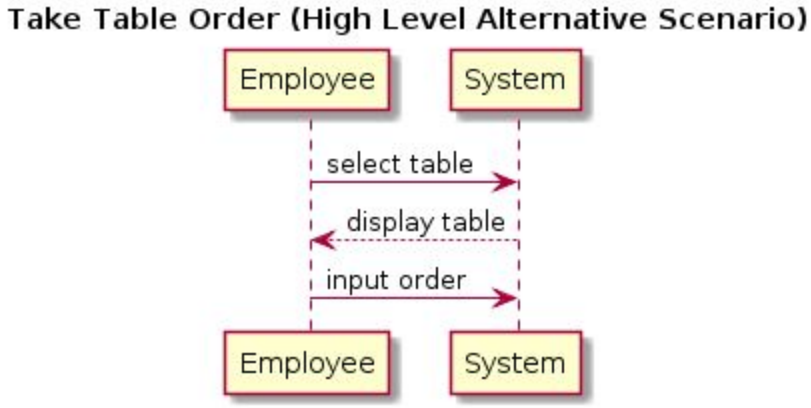
Post conditions: The customer's order has been taken and received by the kitchen.

Normal Scenario:
Employee takes the customer's order.
Employee selects the customer's table.
System shows seats at customer's table.
Employee selects customer's seat.
System shows customer's details
Employee inputs new order.



Alternative Scenario:

Employee takes the customer's order.
Employee selects the customer's table.
System shows table details
Employee inputs new order.



Use Case: Take Table Order (Concrete)

Summary: The server takes the order of customer(s) at a table.

Actors: Employee (Server)

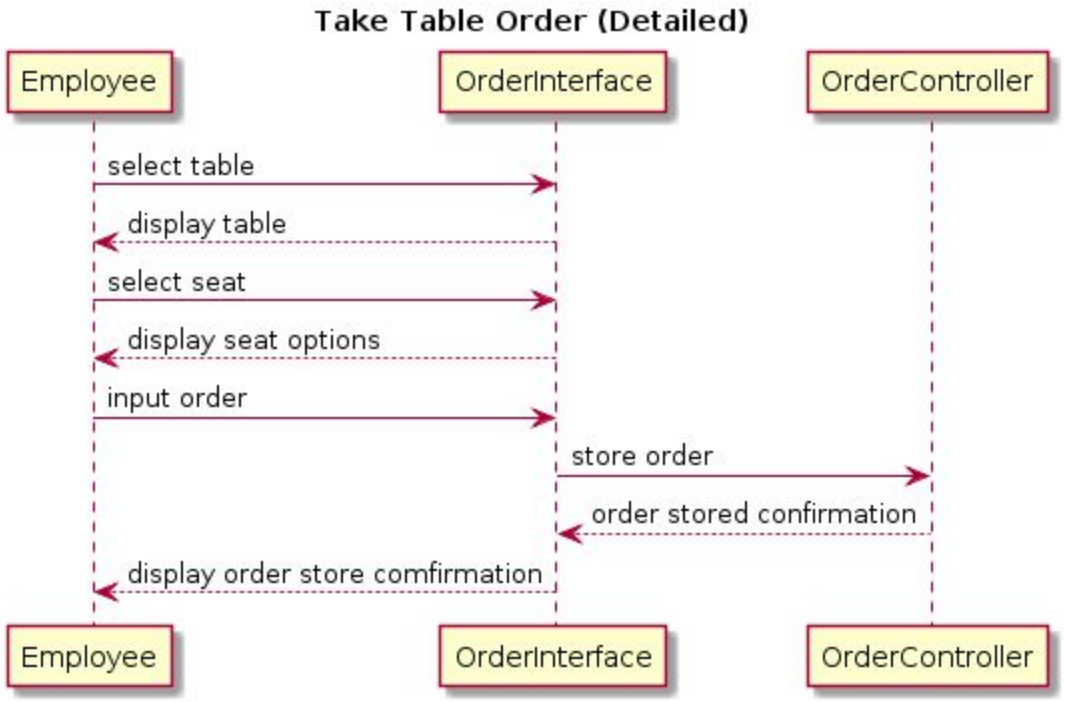
Preconditions: Customer has been seated.

Description:

Actors	System
1) Employee selects customer's table	
	2) Displays seats at selected table
3) Employee selects customer's seat	
	4) Displays options for selected seat
5) Employee selects option for new order	
6) Employee inputs order	
	7) Stores order under selected customer's account
	8) Sends order to kitchen

Exceptions:
Order for entire table: An order is being split among the entire table. Instead of selecting the customer seat the employee can enter a new order at the table level.

Postconditions: The customer's order has been taken and received by the kitchen.



Clock Out

Use Case: Clock Out (Essential)
Summary: Employee clocks out
Preconditions: Employee is currently clocked in.
Description

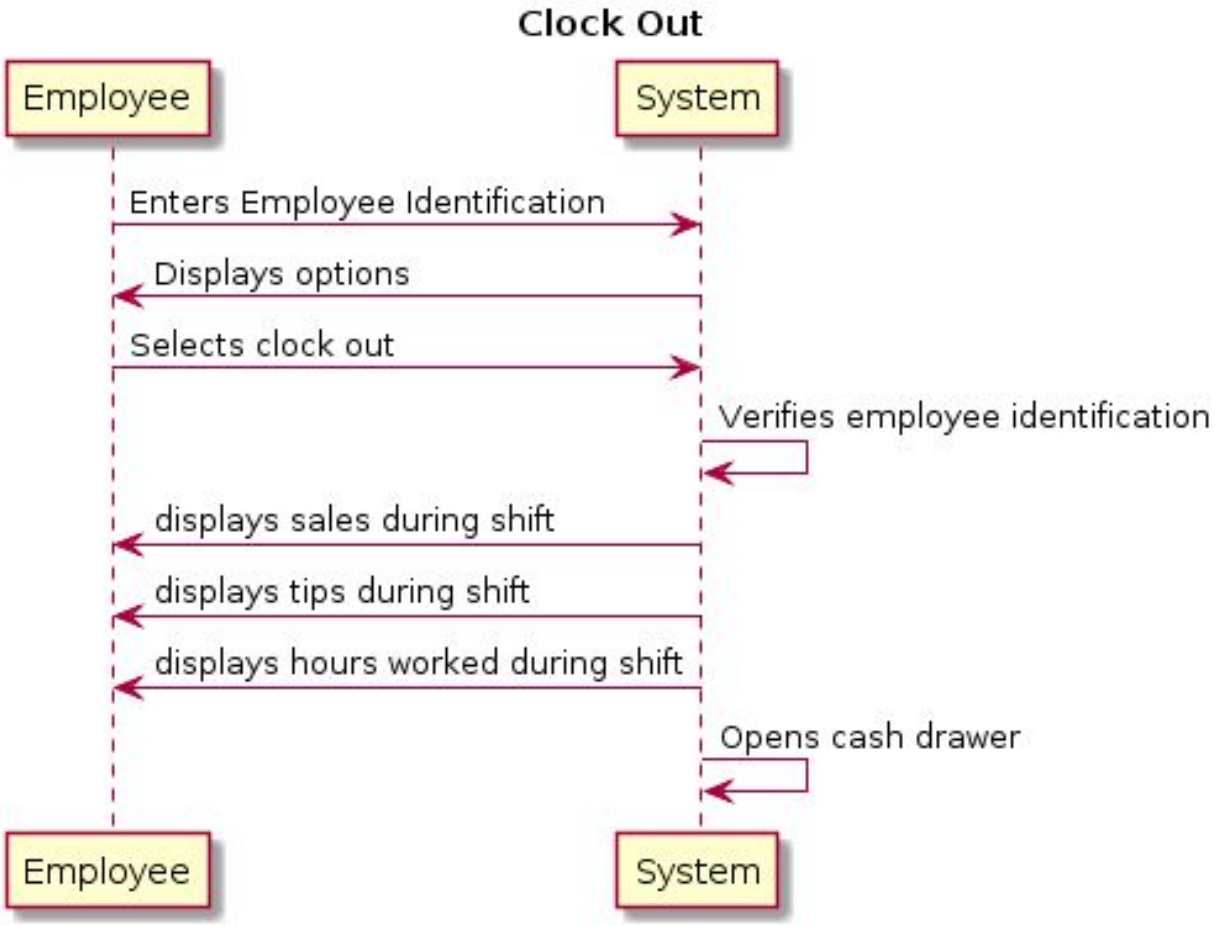
Employee	System

1. Employee enters ID 3. Employee clock out	2. Display options. Clocks out employee at current time Display employee earnings. Open cash drawer.

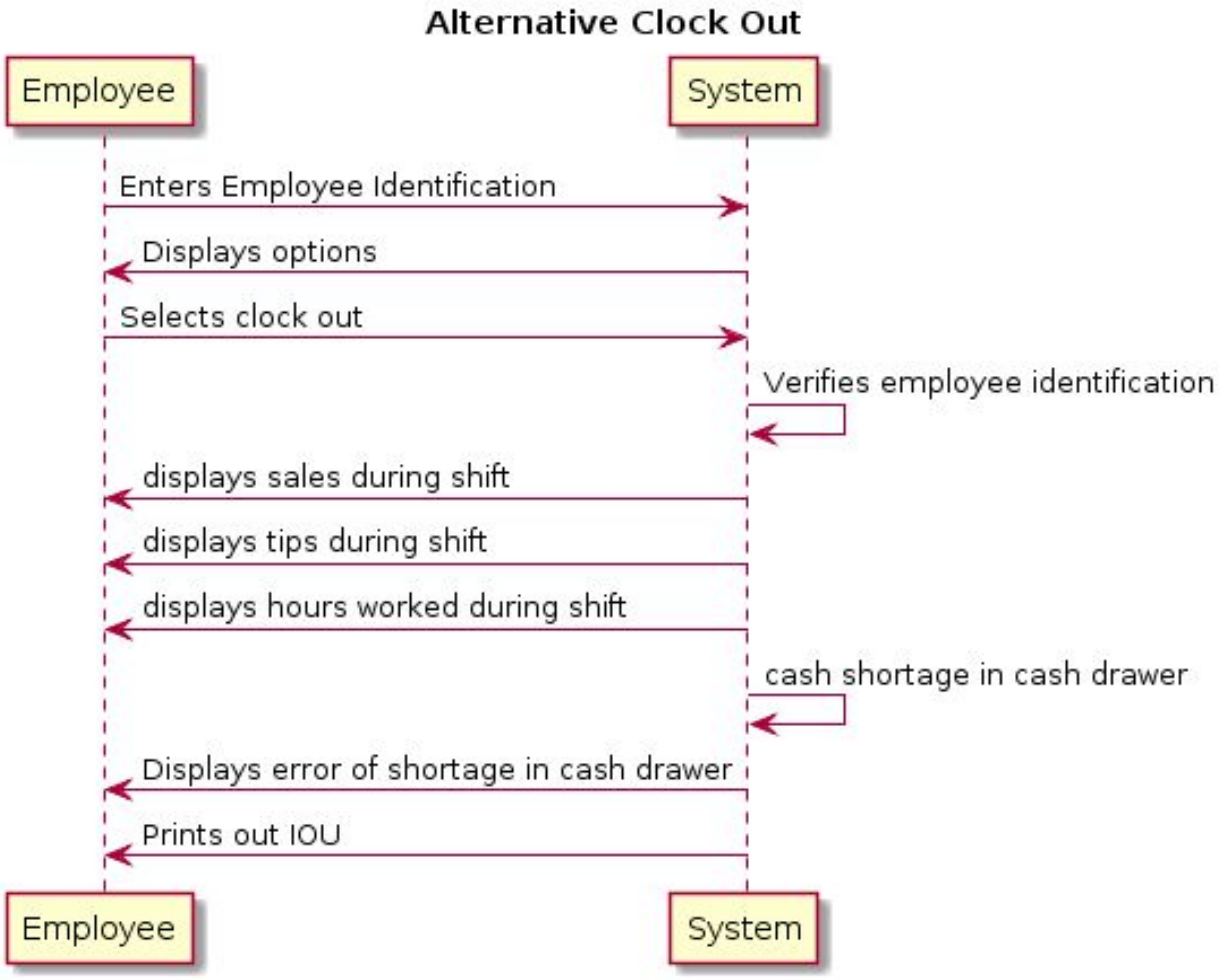
Exceptions:
Not enough cash in drawer: Manager is alerted that the employee is owed money and the amount that is owed. An IOU for the cash tips is printed out.
Wrong password: User returned to log in screen. An alert informs the employee that the wrong password was entered.

Non-existent Employee: User returned to log in screen. An alert informs the user that the specified employee doesn't exist.

Post conditions: The employee is clocked out and has either tips in cash or an IOU for that shift.
Normal Scenario:
Employee enters employee identification
Employee selects clock out
System verifies employee identification
System clocks out employee time
System displays employee sales during shift
System displays employee tips during shift
System displays employee hours worked during shift
System opens cash drawer


Clock Out
Employee
System
Enters Employee Identification
Displays options
Selects clock out
Verifies employee identification
displays sales during shift
displays tips during shift
displays hours worked during shift
Opens cash drawer
Employee
System

Alternative Scenario:
Employee enters employee identification
Employee selects clock out
System verifies employee identification
System clocks out employee time
System displays employee sales during shift
System displays employee tips during shift
System displays employee hours worked during shift
System sends error that there is not enough cash in drawer
System prints out IOU to employee



Use Case: Clock Out

Summary: Employee clocks out

Preconditions: Employee is currently clocked in.

Description

Employee	System
1. Employee selects “Clock Out” 2. Employee enters name 3. Employee enters password	1. Verifies employee credentials. 1. Clocks out employee at current time 1. Displays employee sales for shift. 1. Displays employees card tips for shift. 1. Displays number of hours worked in shift.
10) Employee removes cash of the amount of his/her earned card tips for shift.	9) Cash drawer is opened.

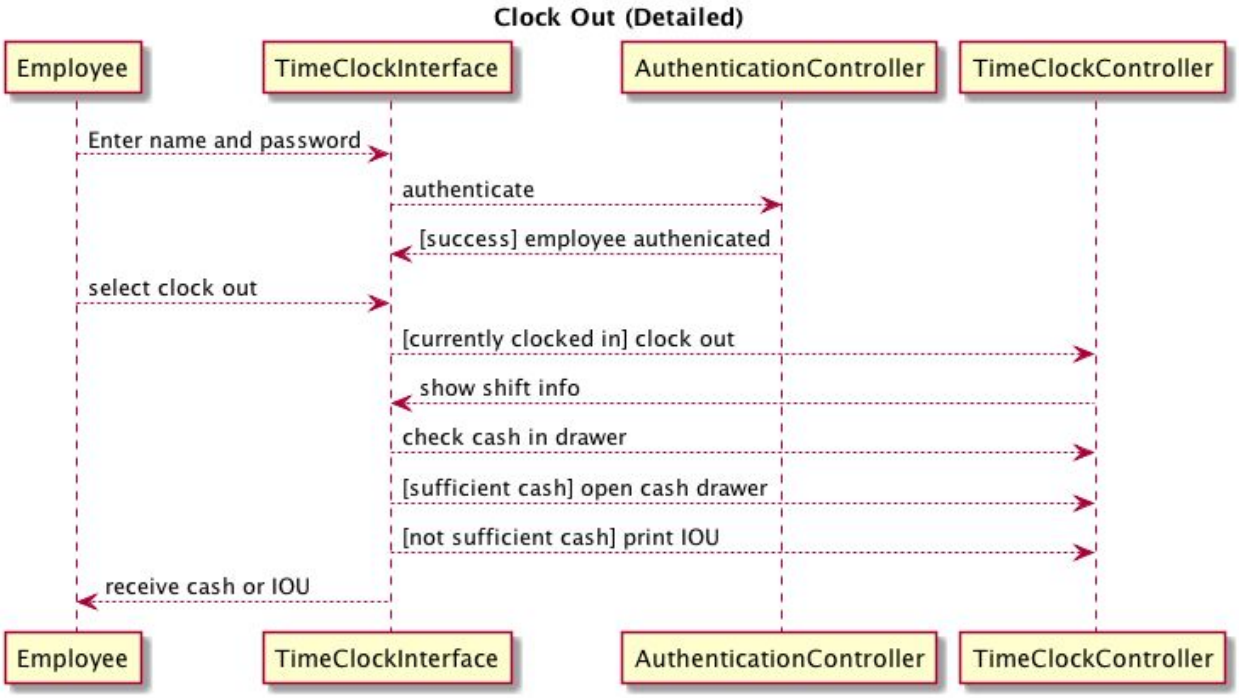
Exceptions:

Not enough cash in drawer: Manager is alerted that the employee is owed money and the amount that is owed. An IOU for the cash tips is printed out.

Wrong password: User returned to log in screen. An alert informs the employee that the wrong password was entered.

Non-existent Employee: User returned to log in screen. An alert informs the user that the specified employee doesn’t exist.

Postconditions: The employee is clocked out and has either tips in cash or an IOU for that shift.

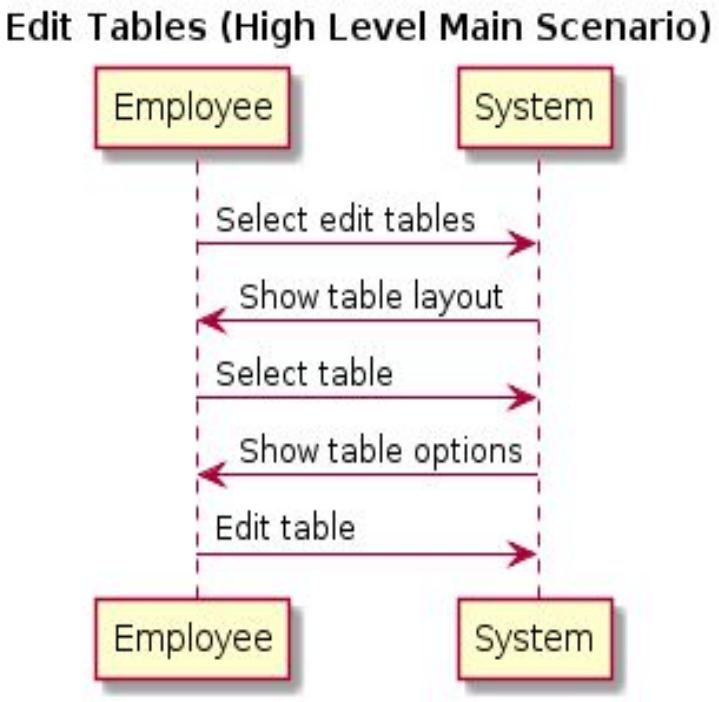

Clock Out (Detailed)
Employee
TimeClockInterface
AuthenticationController
TimeClockController
Enter name and password
authenticate
[success] employee authenicated
select clock out
[currently clocked in] clock out
show shift info
check cash in drawer
[sufficient cash] open cash drawer
[not sufficient cash] print IOU
receive cash or IOU
Employee
TimeClockInterface
AuthenticationController
TimeClockController

Edit Tables

Use Case: Edit Tables (Essential)
Summary: The employee edits the layout and details of the tables.
Actors: Employee
Preconditions: Tables are currently empty.
Description:

Actors	System
1) The employee selects option to edit tables	2) Shows layout of tables and option to add table.
3) The employee selects table they wish to edit.	4) Shows number of seats at table and gives option to delete table.
5) The employee edits the selected table.	6) Saves and reflects changes.

Post conditions:
The tables are edited as they employee intended.
Normal Scenario:
Employee selects option to edit tables.
System shows layout of tables and option to add a new table.
Employee selects the table they wish to edit.
System shows number of seats at table and gives option to delete the table.
Employee edits the selected table.
System saves and reflects changes.

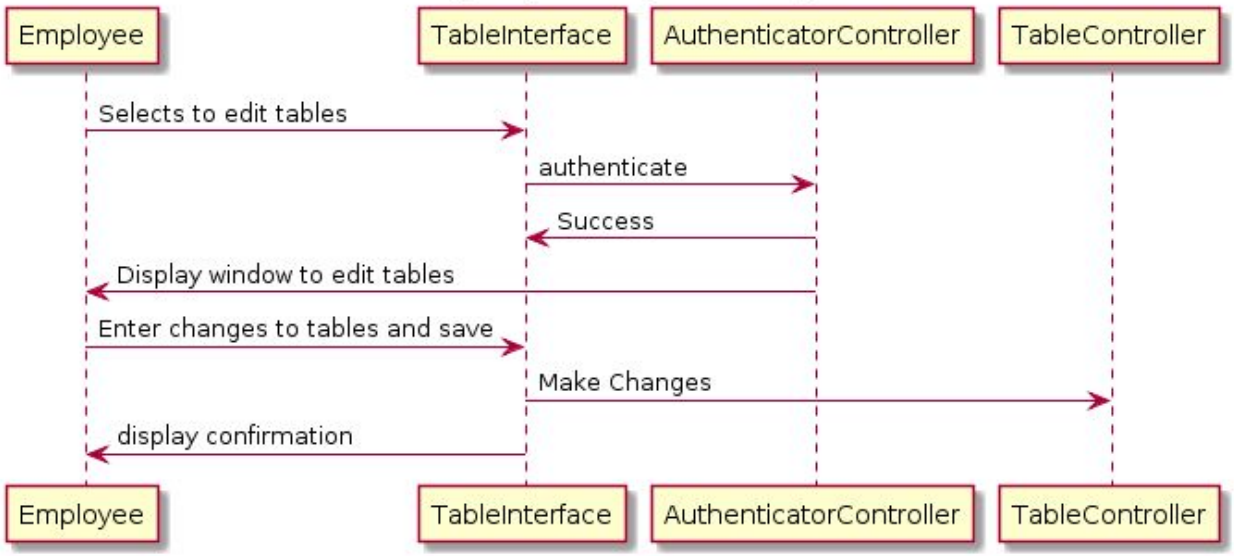


Use Case: Edit Tables (Concrete)

Summary: The employee edits the layout and details of the tables.
Actors: Employee
Preconditions: Tables are currently empty.
Description:

Actors	System
1) The employee selects option to edit tables.	2) Shows layout of tables.
3) The employee selects the table they wish to edit.	4) Shows number of seats at table and gives option to delete table.
5) The employee edits the table.	6) Saves and reflects changes.

Post conditions:
The system saves the changes.



Edit Employee

Use Case: Edit Employee (Concrete)
Summary: A manager has hired a new employee and needs to add them to the system. The manager will assign the employee an identification number and the system will add the new employee to a list of employees.
Actors: Employee (Manager)
Preconditions: There is a manager in the system that can perform the action of adding a new employee.
Description:

Actor	System
1)Manager selects option to add a new employee into the system	
	2)System is prompted of request and displays window for employee ID to be entered
3)Manager inputs his/her employee ID into the system	
	4)System verifies employee ID 5) System displays new employee window that the manager can edit

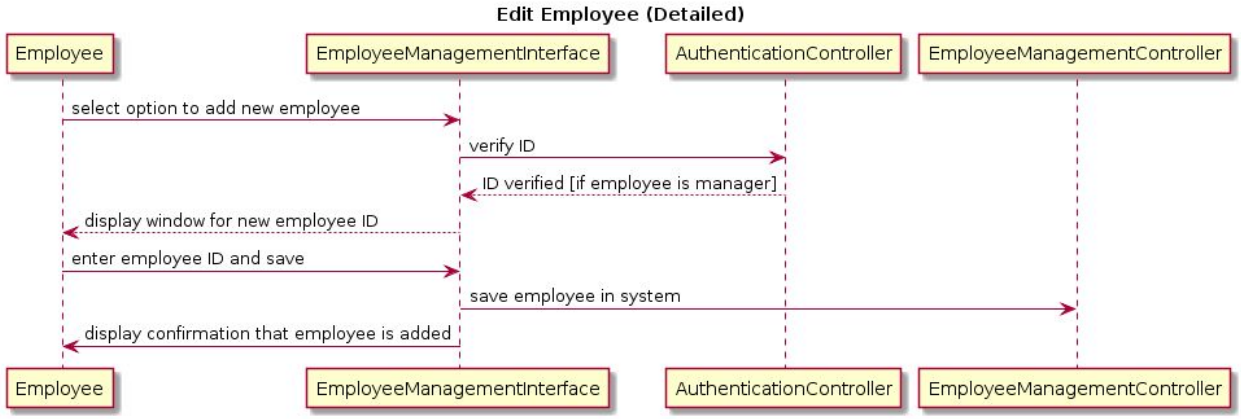
6)Manager enters a new employee ID into the window and clicks the save button 9) Manager acknowledges confirmation window by pressing the ok button.	7) The system saves the new employee ID 8)System displays a confirmation window that the employee has been added.

Exceptions:

Employee does not have manager permissions: If employee is not a manager the POS system does not display a new employee window. An invalid permissions message is displayed before step 5 and the system goes back to its main menu after the employee presses the ok button on the pop up window error message.

Employee ID invalid: Either the employee ID is already taken or does not meet the requirements of an employee ID. An error message is displayed after step 6. Once the manager acknowledges the error message by pressing the ok button the system returns to the edit employee window.

Post conditions: A new employee has been added to the system and now has employee permissions within the system.

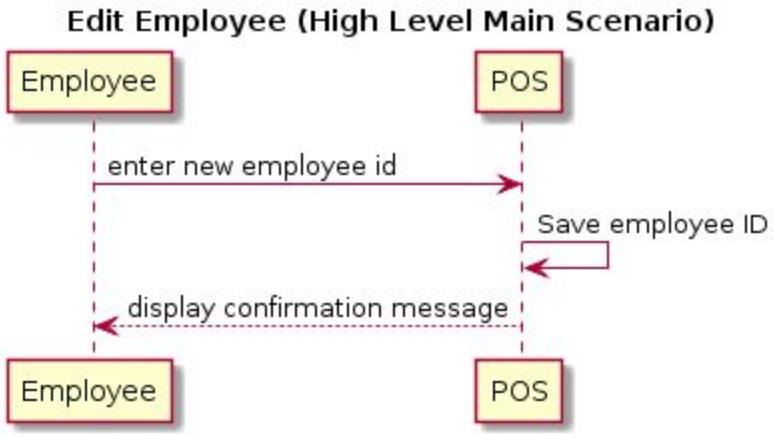


Use Case: Edit Employee (Essential)
Summary: Manager assigns an employee an ID number within the system.
Actors: Employee (Manager)
Preconditions: Employee with manager permissions is in the system
Description:

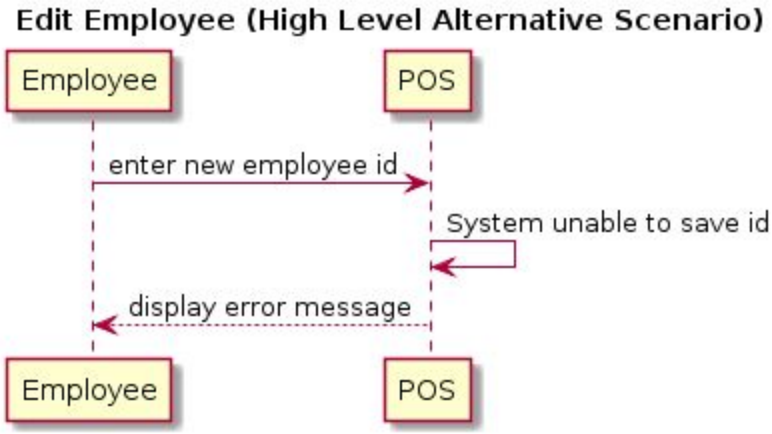
Actor	System
1) Manager inputs employee ID	2) System saves new employee ID.

Exceptions:

Employee ID invalid: Either the employee ID is already taken or does not meet the requirements of an employee ID.
Post conditions: A new employee has been added to the system
Normal Scenario:
Employee enters a new employee to be added
Employee enters employee ID
System saves new employee
System shows confirmation message



Alternative Scenario:
Employee enters a new employee to be added
Employee enters employee ID
System cannot save employee ID, ID is already in use
System shows error message



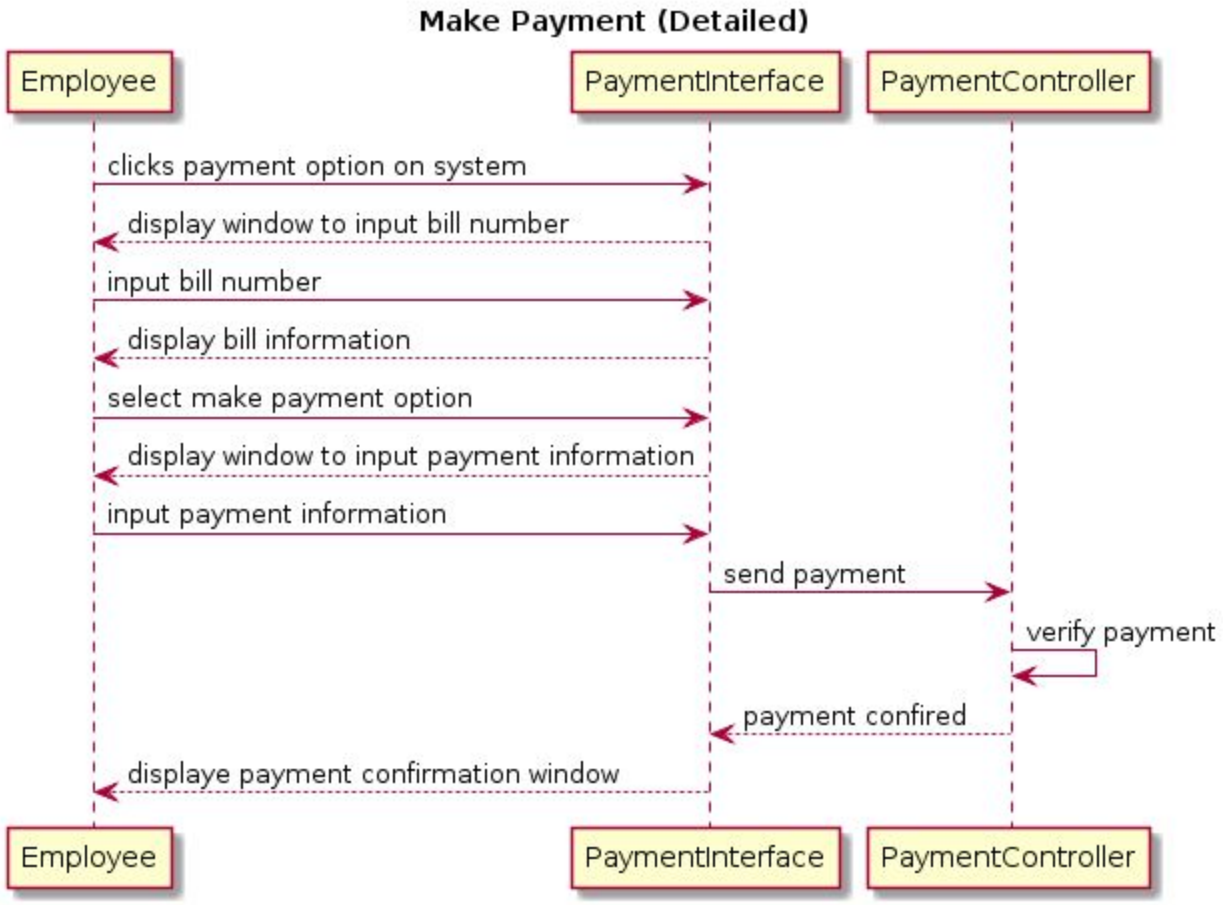
Make Payment

Use Case: Make Payment (Concrete)
Summary: Customer initiates that they would like to make the payment for their bill. The employee enters the payment into the system. The system verifies and confirms the payment of the bill.
Actors: Employee (Server)
Preconditions: Customer has already received the bill they will be paying for.
Description:

Actor	System
1) Employee clicks payment option on system	
	2) System responds with window to input bill number
3) Employee inputs the bill number into the system	
	4) System responds with window displaying information about the bill
5) Employee selects make payment option	
	6) System asks for payment information
7) Employee inputs payment information and presses confirm payment button	
	8) System confirms payment
	9) System displays bill payment confirmation window

Exceptions:
Payment declined: The customer's card is not accepted because it has expired. The transaction can not be completed and the POS terminal displays invalid payment message. The bill is not marked as paid.

Post conditions: The bill is paid for and the transaction is complete.


Make Payment (Detailed)
Employee
PaymentInterface
PaymentController
clicks payment option on system
display window to input bill number
input bill number
display bill information
select make payment option
display window to input payment information
input payment information
send payment
verify payment
payment confired
displaye payment confirmation window
Employee
PaymentInterface
PaymentController

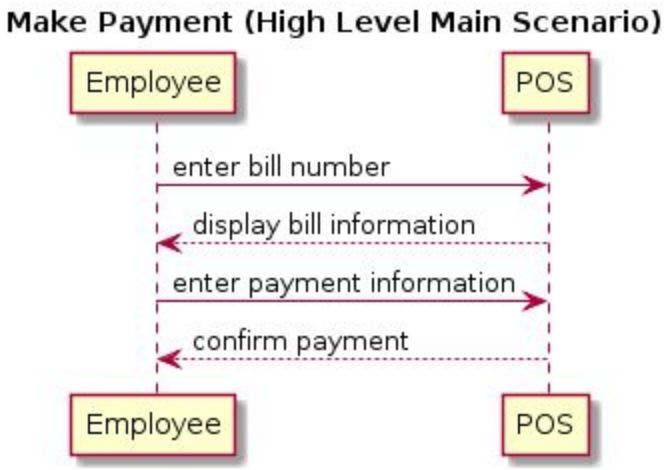
Use Case: Make Payment (Essential)
Summary: Employee enters payment information into the system. System handles customer payment.
Actors: Employee (Server)
Preconditions: Customer has already received the bill they will be paying for.
Description:

Actor	System
1) Employee inputs bill number 3) Employee inputs payment	2) System displays bill 4) System confirms payment

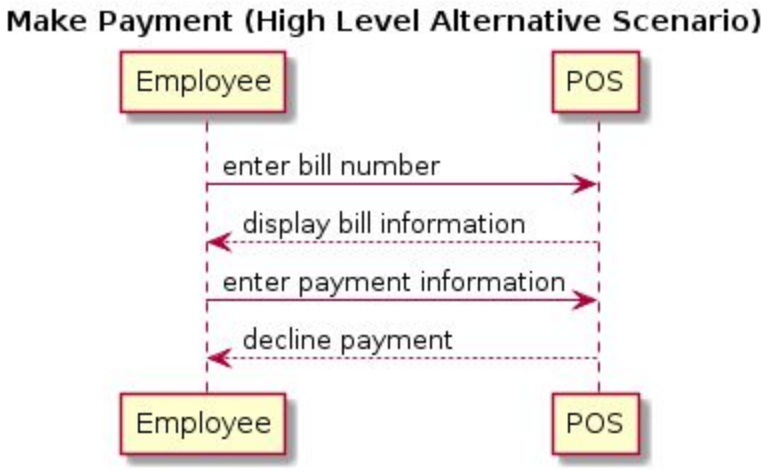
Exceptions:
Payment declined: The customer's card is not accepted because it has expired.

Post conditions: The bill is paid for and the transaction is complete.

Normal Scenario:
Employee enters bill number
System displays bill information
Employee enters payment information
System confirms payment



Alternative Scenario:
Employee enters bill number
System displays bill information
Employee enters payment information
System declines payment

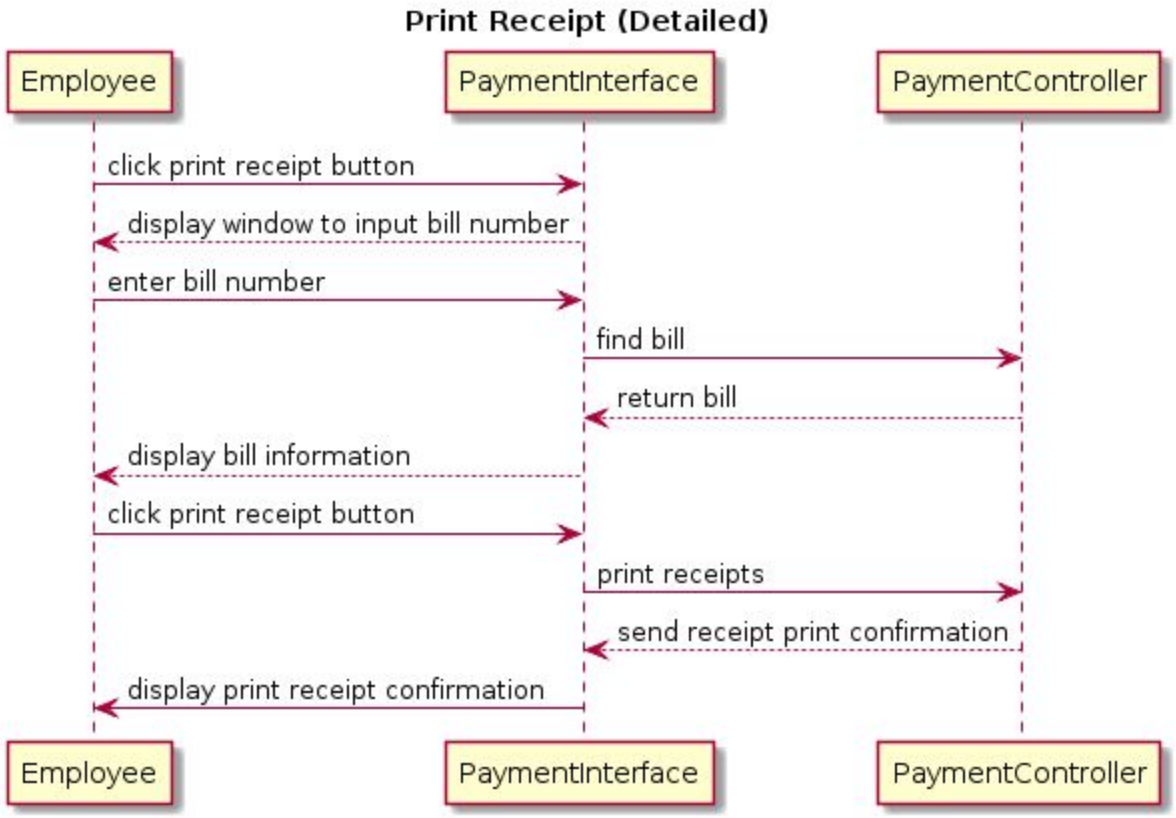


Print Receipt

Use Case: Print Receipt (Concrete)
Summary: The server prints the receipt for a customer. The receipt has the final totals of the customer order. It also shows the bill information such as what each item cost etc.
Actors: Employee
Preconditions: The bill has been fully paid for in the system and the customer is ready to leave.
Description:

Actors	System
1) Employee selects print receipt option in the system	2) System returns a window for the employee to enter bill number
3) Employee inputs bill number	
5) Employee selects print receipt option	6) System displays bill information
	7) System prints two copies of receipt (customer copy, restaurant copy)
8) Employee receives print out	

Exceptions:
No receipt: Customer does not want a receipt. The system only prints out one copy of the receipt for the restaurant.
Post conditions: The receipt is printed successfully.



Use Case: Print Receipt (Essential)
Summary: The server prints the receipt for a customer.
Actors: Employee
Preconditions: The bill has been paid for.
Description:

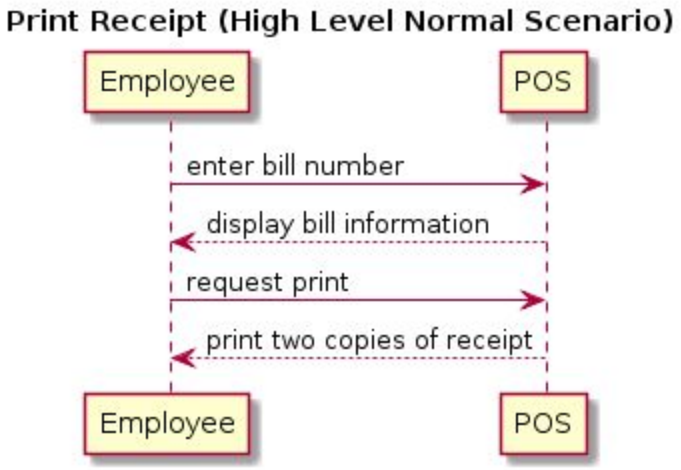
Actors	System
1) Employee inputs bill number	2) System displays bill information
3) Employee selects print receipt option	4) System prints two copies of receipt (customer copy, restaurant copy)

Exceptions:
No receipt: The system only prints out one copy of the receipt for the restaurant.

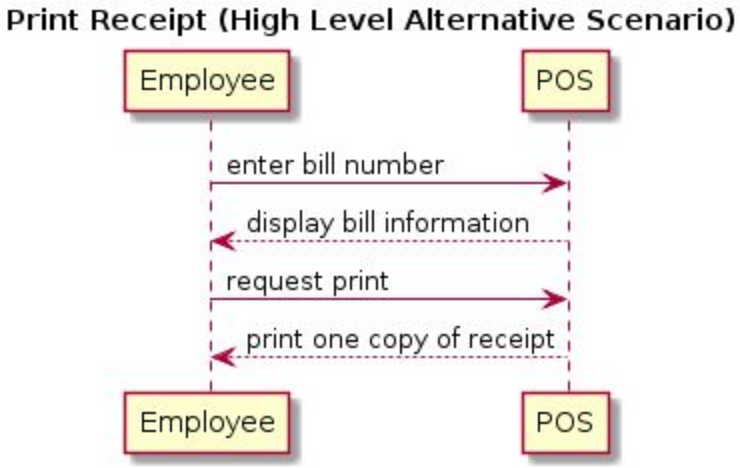
Post conditions: The receipt is printed successfully.

Normal Scenario:

Employee inputs bill number
System displays bill information
Employee selects print option
System prints two receipts



Alternative Scenario:
Employee inputs bill number
System displays bill information
Employee selects print single print option
System prints one receipt



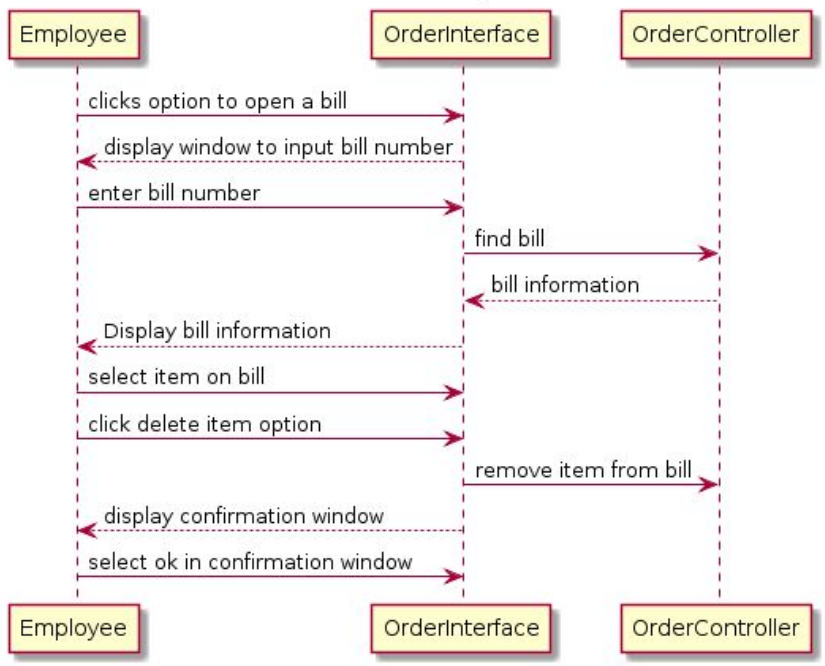
Cancel Customer Order

Use Case: Cancel Customer Order (Concrete)
Summary: Customer has issue with item they are being charged for. A manager goes into the system and cancels the charge on the bill.
Actors: Employee (Manager)
Preconditions: Customer has received bill they need to pay for.
Description:

Actor	System
1) Employee clicks option to pull open a bill	2) System displays window to input bill number
3) Employee enters bill number	4) System displays bill information
4) Employee selects item on the bill 5) Employee selects delete item option	6) System removes item from the bill and updates bill information

8) Employee selects ok option on confirmation window	7) Displays confirmation window that item was deleted

Exceptions:
Employee non manager: Employee trying to cancel the charge does not have manager permissions.
Post conditions: The charge on the bill has been removed.

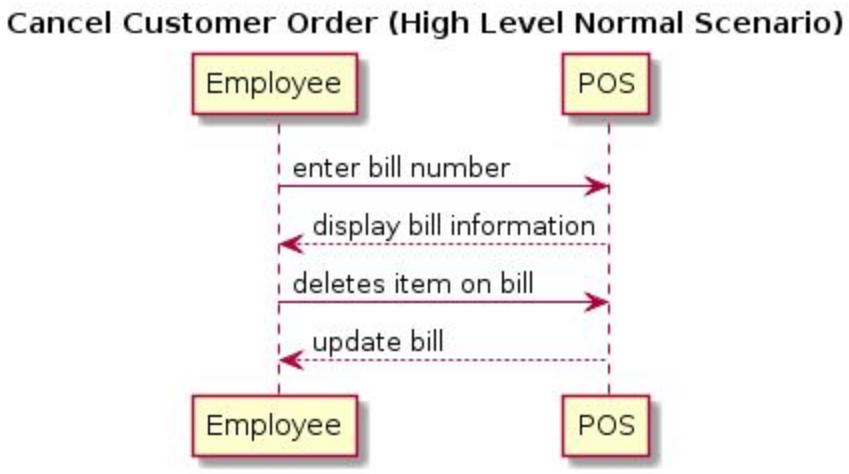


Use Case: Cancel Customer Charge (Essential)
Summary: Customer has issue with item they are being charged for. A manager goes into the system and cancels the charge on the bill.
Actors: Employee (Manager)
Preconditions: Customer has received bill they need to pay for.
Description:

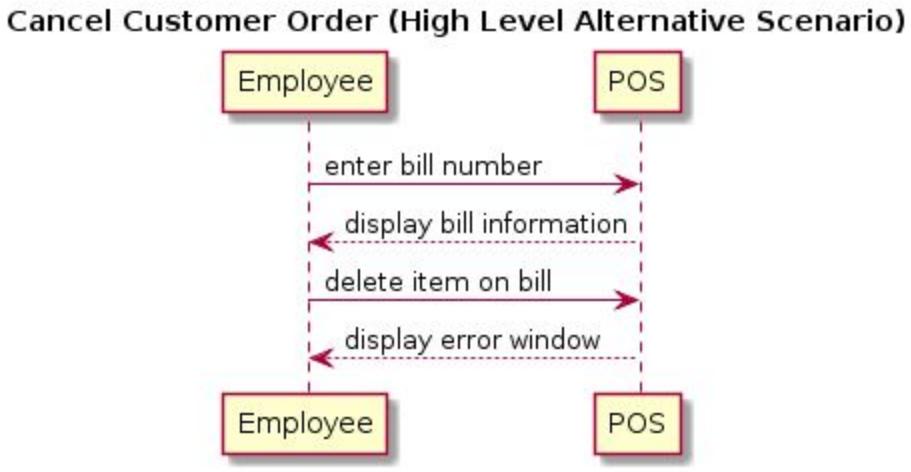
Actor	System
1) Employee inputs bill number	2) System displays bill
3) Employee deletes item	4) System updates bill

Exceptions:
Employee non manager: Employee trying to cancel the charge does not have manager permissions.
Post conditions: The charge on the bill has been removed.

Normal Scenario:
Employee enters bill number
System displays bill information
Employee deletes item on bill
System updates bill with item deleted



Alternative Scenario:
Employee enters bill number
System displays bill information
Employee attempts to delete item on bill
System declines deletion displays error message



Use Case: Prepare Order (Concrete)

Summary: Employee enters order into the system. The kitchen staff updates the system that the order is being prepared.

Actors: Employee, Kitchen

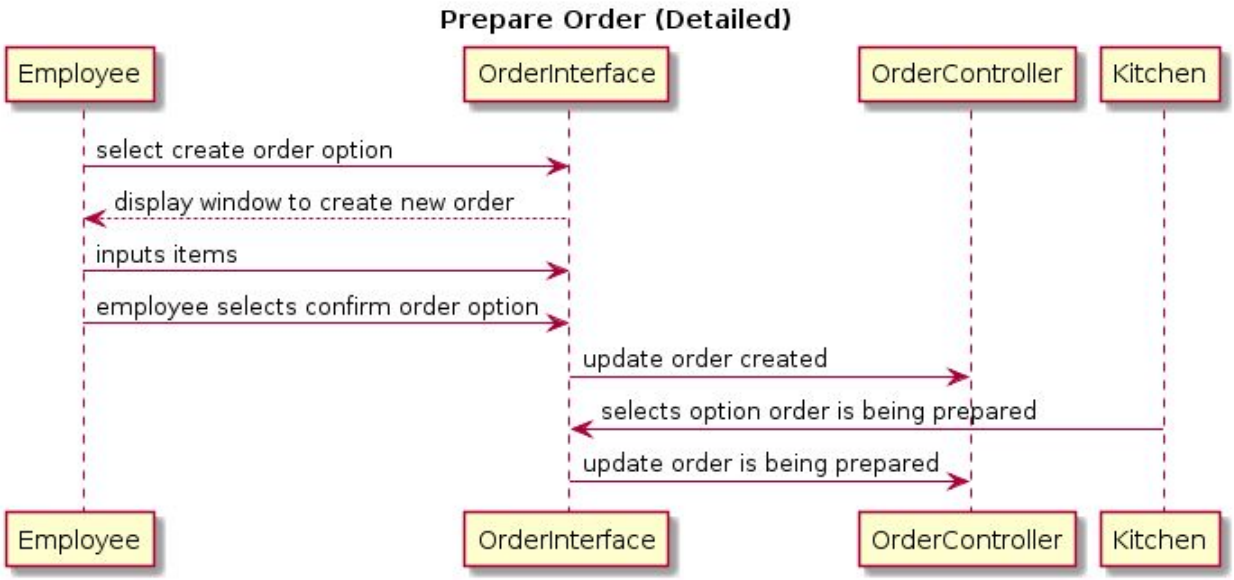
Preconditions: Customer has ordered

Description:

Actor	System
1) Employee selects create order option	
	2) System displays window to create new order
3) Employee inputs items that need to be prepared	
4) Employee selects confirm order option	
	5) System updates that an order has been created
6) Kitchen selects option that order is being prepared	
	7) System updates that order is under preparation

Exceptions:
Item out of stock: An item is out of stock in the system. System displays error message

Post conditions: Order is being prepared



Prepare Order

Use Case: Prepare Order (Essential)

Summary: Employee enters order into the system. The kitchen staff updates the system that the order is being prepared.

Actors: Employee, Kitchen

Preconditions: Customer has ordered

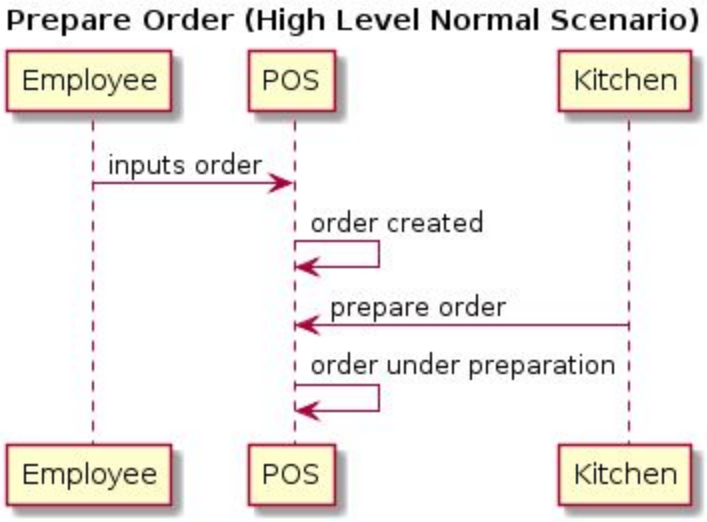
Description:

Actor	System
1) Employee inputs order	
	2) System updates that an order has been created
3) Kitchen inputs that order being prepared	
	4) System updates that order is under preparation

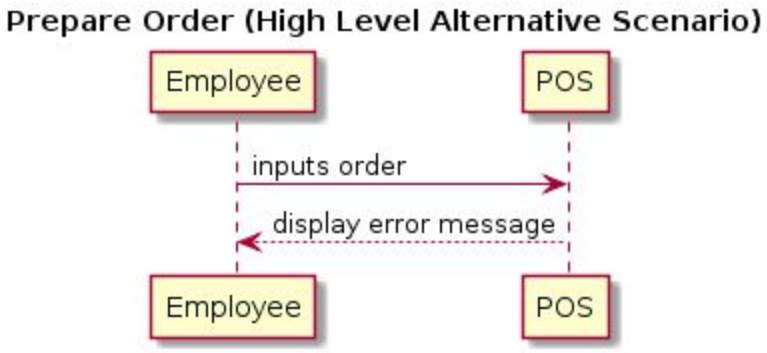
Exceptions:
Item out of stock: An item is out of stock in the system. System displays error message

Post conditions: Order is being prepared.

Normal Scenario:
Employee enters order
System updates order creation
Kitchen marks order being prepared
System updates order is being prepared



Alternative Scenario:
Employee enters order
System displays error message item out of stock



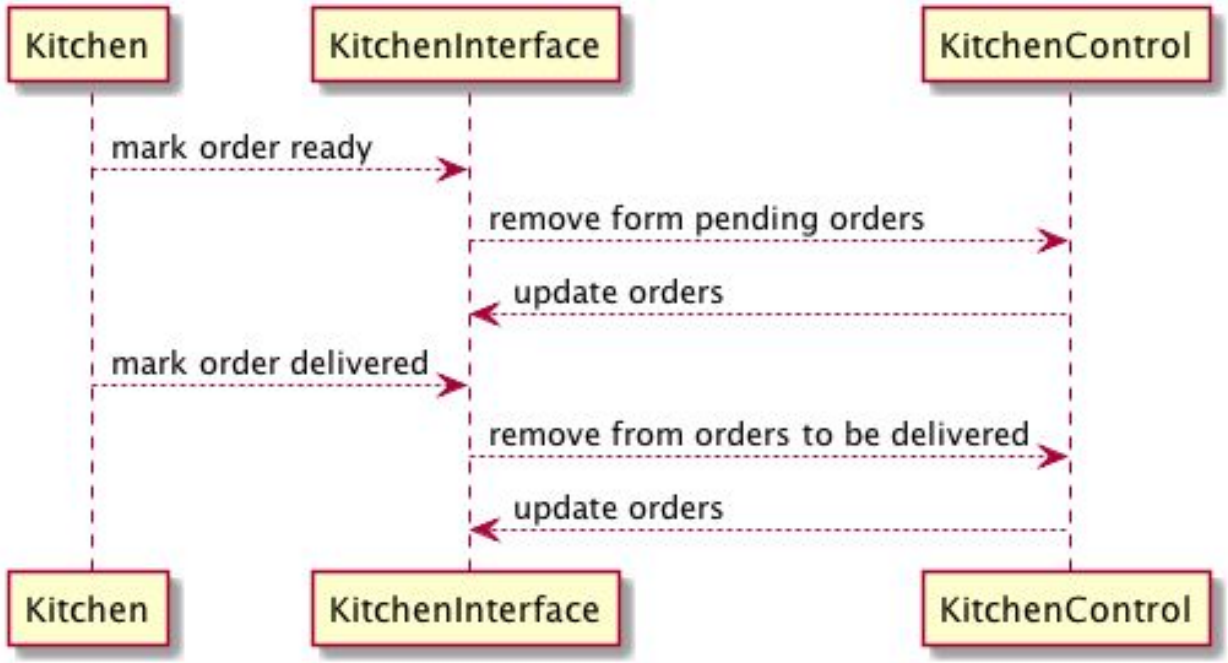
Deliver Order

Use Case: Deliver Order (Concrete)
Summary: Kitchen updates system that order is ready to be delivered. Employee updates system that order has been delivered.
Actors: Employee, Kitchen
Preconditions: Order has been prepared.
Description:

Actor	System
1) Kitchen selects order ready for delivery option	2) System is updated that order is ready to be delivered
3) Employee selects option that order is being delivered	4) System updates that order is being delivered
5) Employee selects option that order is delivered successfully	6) System updates that order is delivered

Exceptions:
Order incorrect: Order is updated in system as incorrect. A new order must then be created.

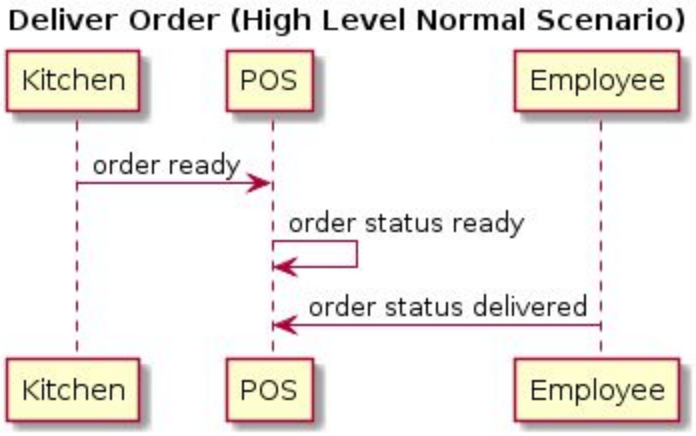
Post conditions: Order is delivered



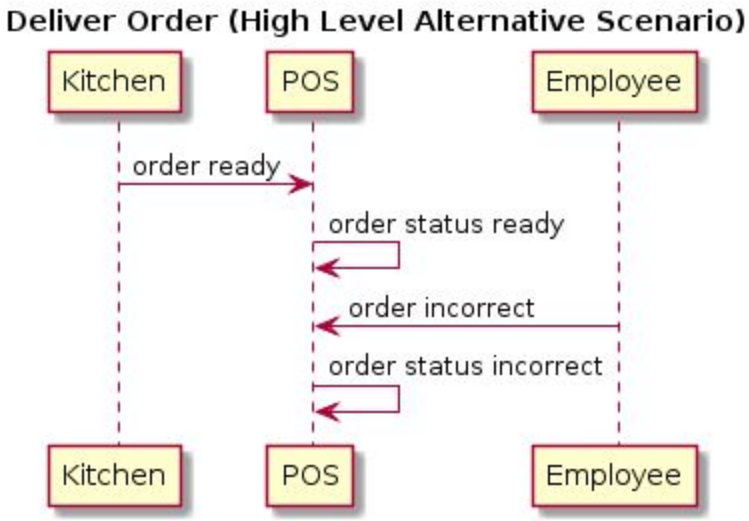
Use Case: Deliver Order (Essential)
Summary: Kitchen updates system that order is ready to be delivered. Employee updates system that order has been delivered.
Actors: Employee, Kitchen
Preconditions: Order has been prepared.
Description:

Actor	System
1) Kitchen inputs order ready 3) Employee inputs order delivered	2) Order in system updated as ready

Exceptions:
Order incorrect: Order is updated in system as incorrect. Employee updates order with correct information and then follows same flow as prepare order.
Post conditions: Order is delivered.
Normal Scenario:
Kitchen updates system order is ready
System updates that order is ready
Employee marks order delivered



Alternative Scenario:
Kitchen updates system order is ready
System updates that order is ready
Employee marks order as incorrect



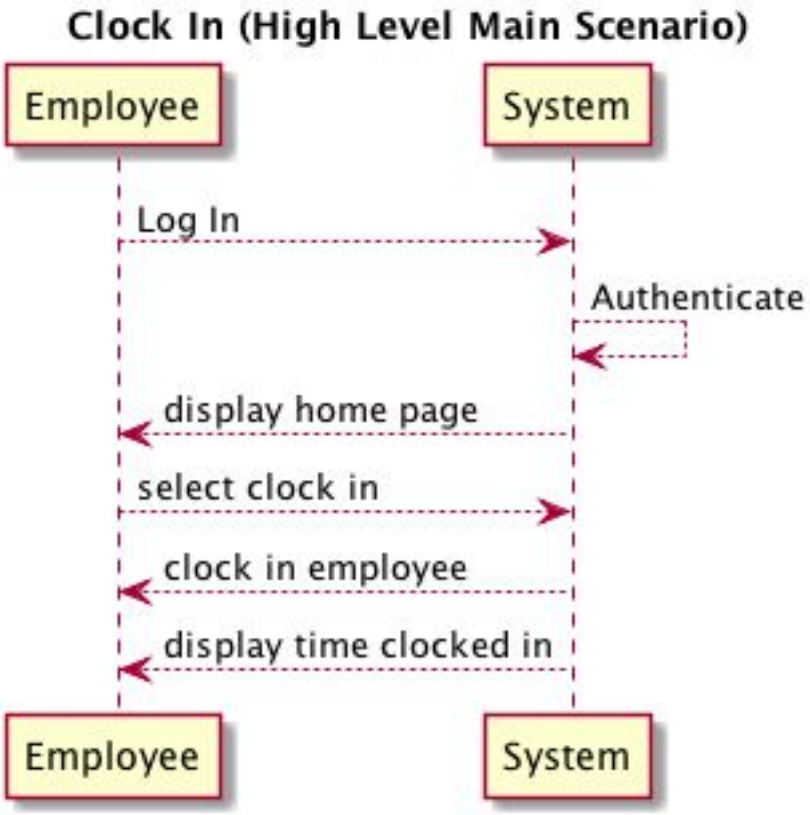
Clock In

Use Case: Clock In
Summary: Employee clocks in
Preconditions: Employee is registered in the system. Employee is not already clocked in.
Description

Employee	System
1. Employee arrives at work. 2. Employee logs in. 4. Employee clocks in.	1. Verifies employee credentials. 1. Clocks in employee at current time.

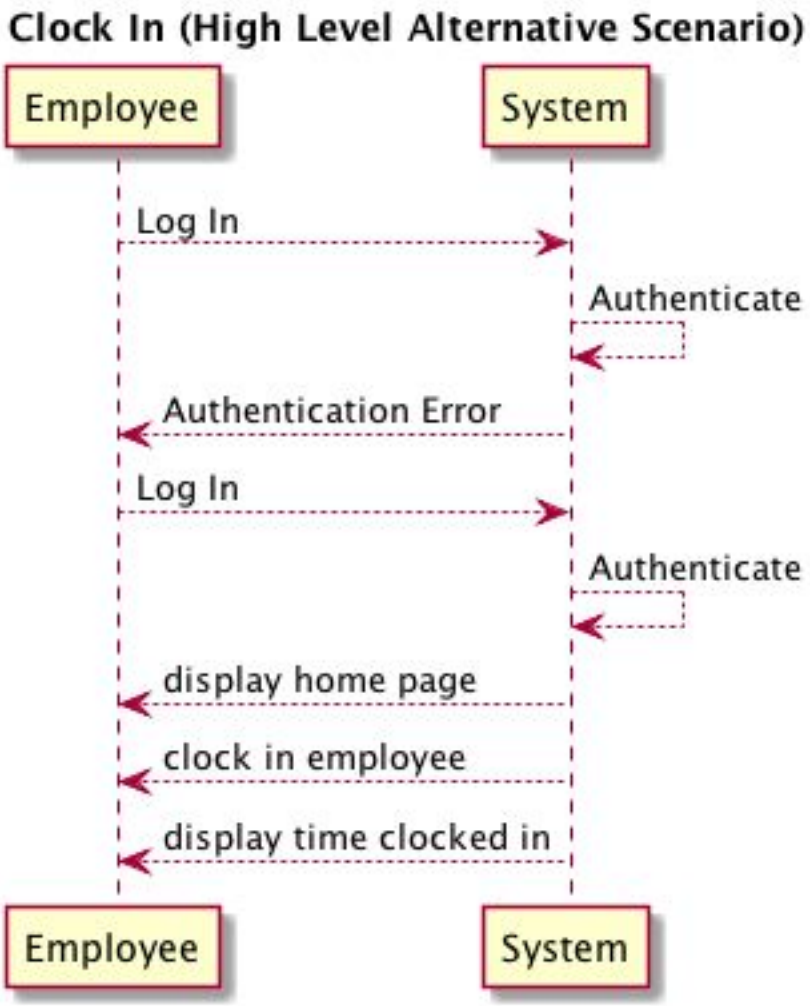
Exceptions:
Wrong password: User returned to log in screen. An alert informs the employee that the wrong password was entered.
Non-existent Employee: User returned to log in screen. An alert informs the user that the specified employee doesn't exist.
Normal Scenario

Employee logs in.
System authenticates employee.
System displays employee page.
Employee clocks in.
System clocks employee in.
System displays clock in time.


Clock In (High Level Main Scenario)
Employee
System
Log In
Authenticate
display home page
select clock in
clock in employee
display time clocked in
Employee
System

Alternative Scenario

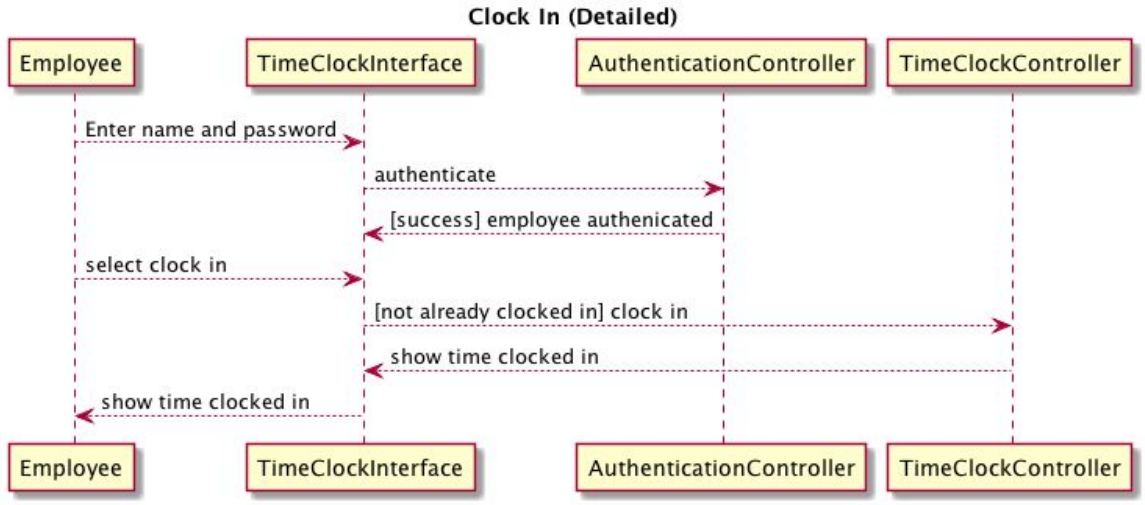
Employee logs in.
System authenticates employee.
System encounters authentication error.
Employee logs in again.
System authenticates employee.
Employee clocks in.
System displays clock in time.



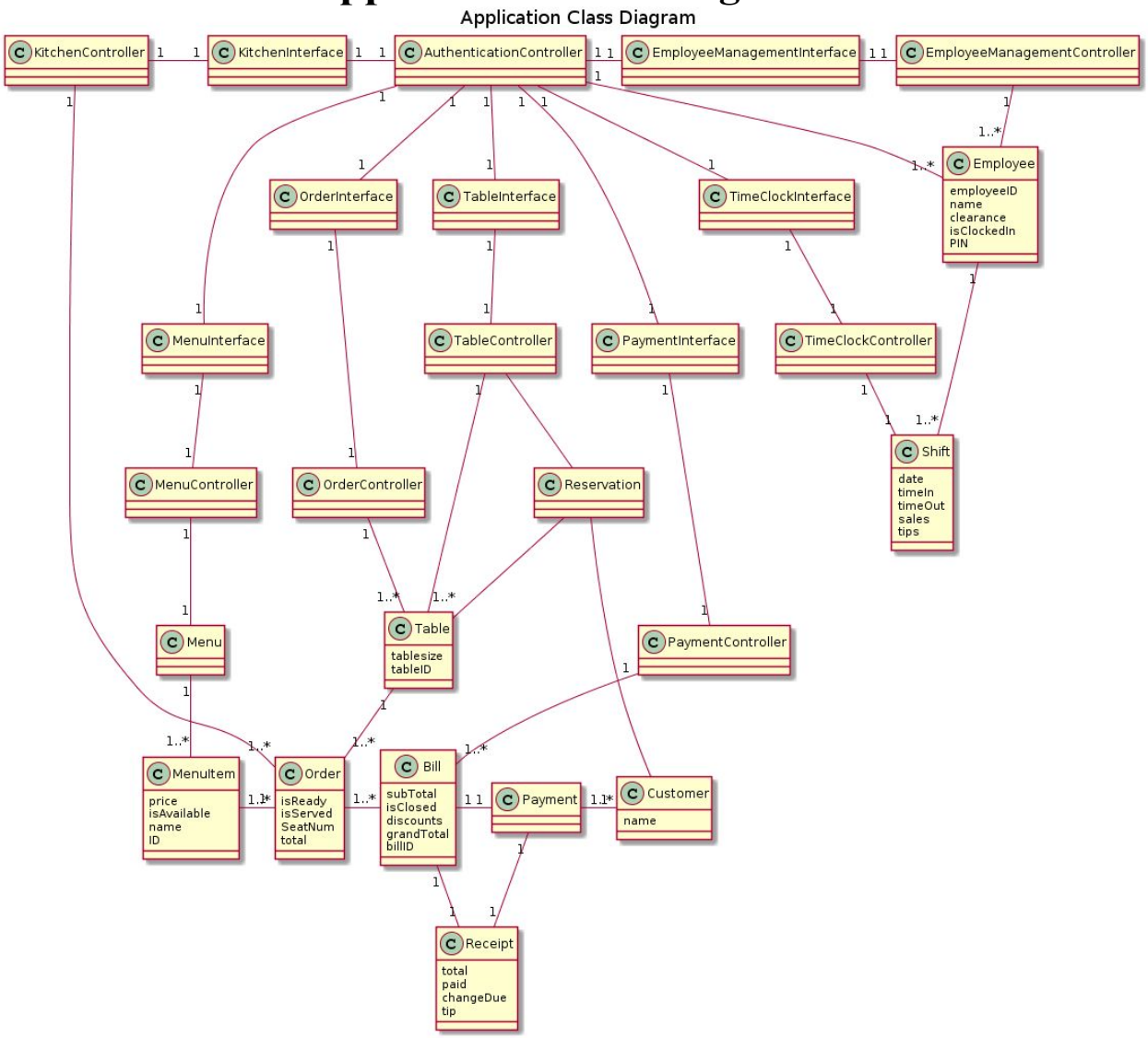
Use Case: Clock In
Summary: Employee clocks in
Preconditions: Employee is registered in the system. Employee is not already clocked in.
Description

Employee	System
1. Employee selects log in. 2. Employee enters name. 3. Employee enters password. 1. Employee selects clock in. 8) Selects Log Out.	1. Verifies employee credentials. 1. Clocks in employee at current time. 1. Displays time clocked in.

Exceptions:
Wrong password: User returned to log in screen. An alert informs the employee that the wrong password was entered.
Non-existent Employee: User returned to log in screen. An alert informs the user that the specified employee doesn't exist.



Application Class Diagram



Constraints

Context: ClockOut
Pre: employee.clockedIn = True

Context: ClockIn
Pre: employee.clockedIn = False

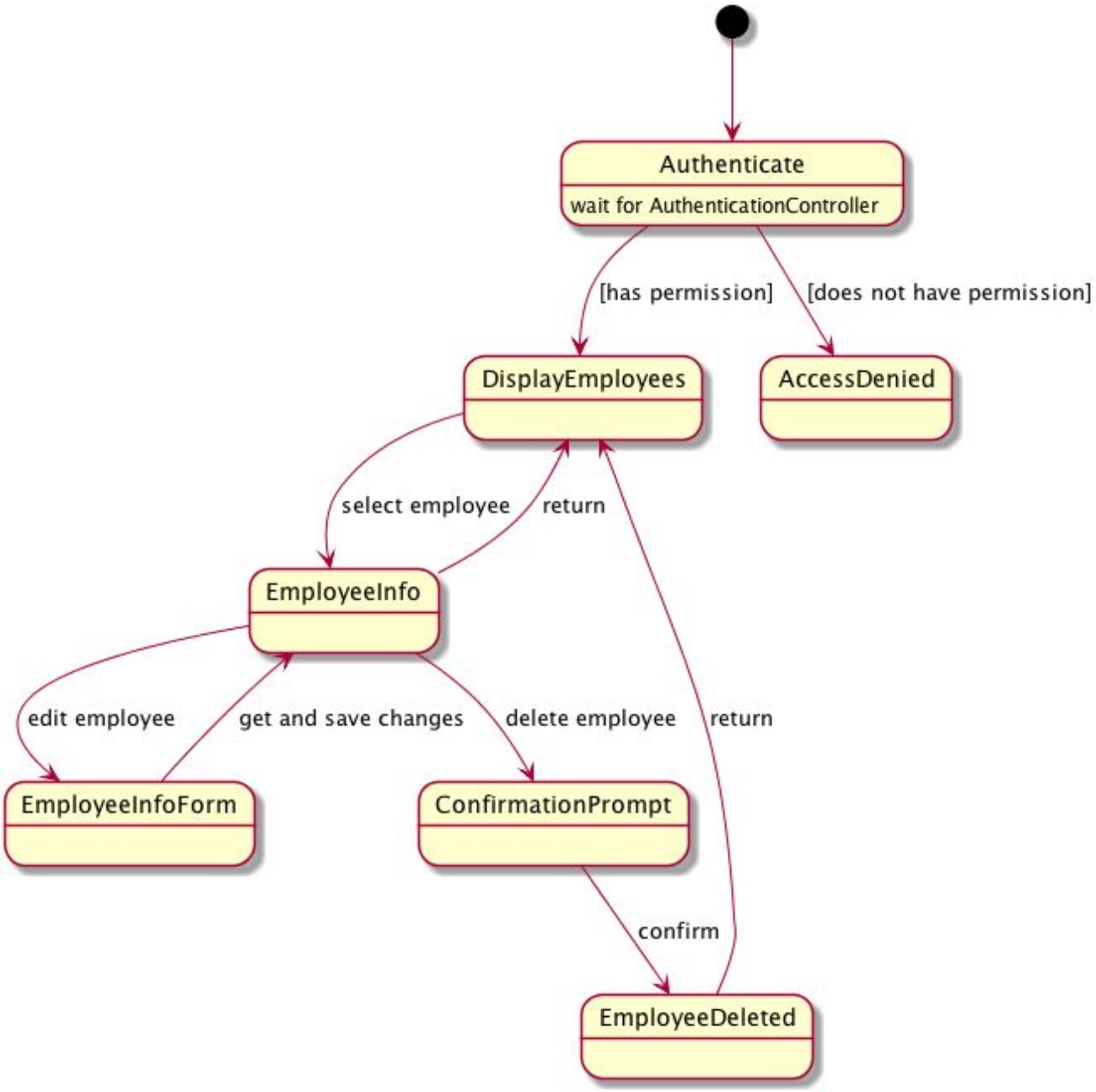
Context: Order.isServed
Pre: Order.isReady = True

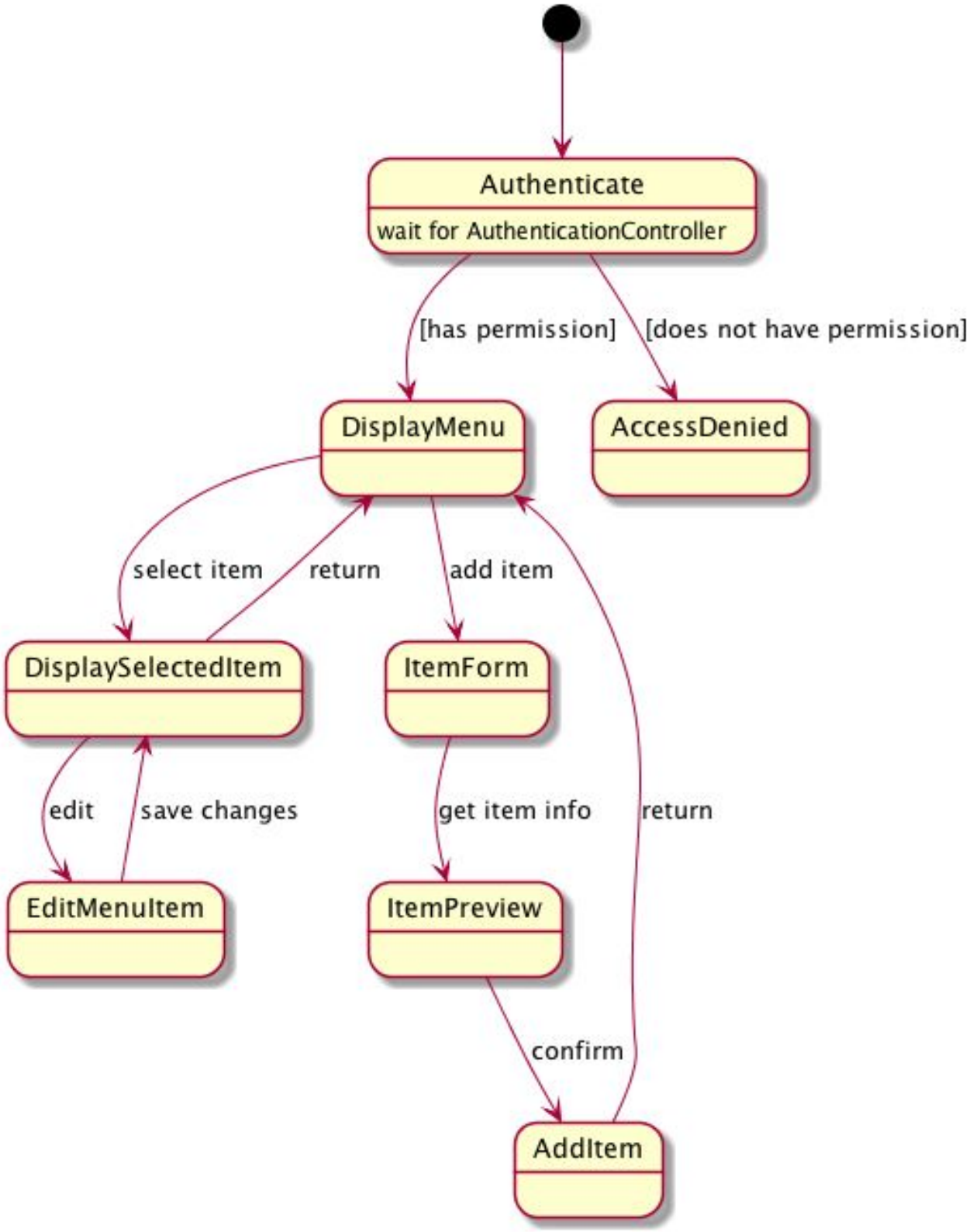
Context: Menu Item
Inv: Self.Price > 0

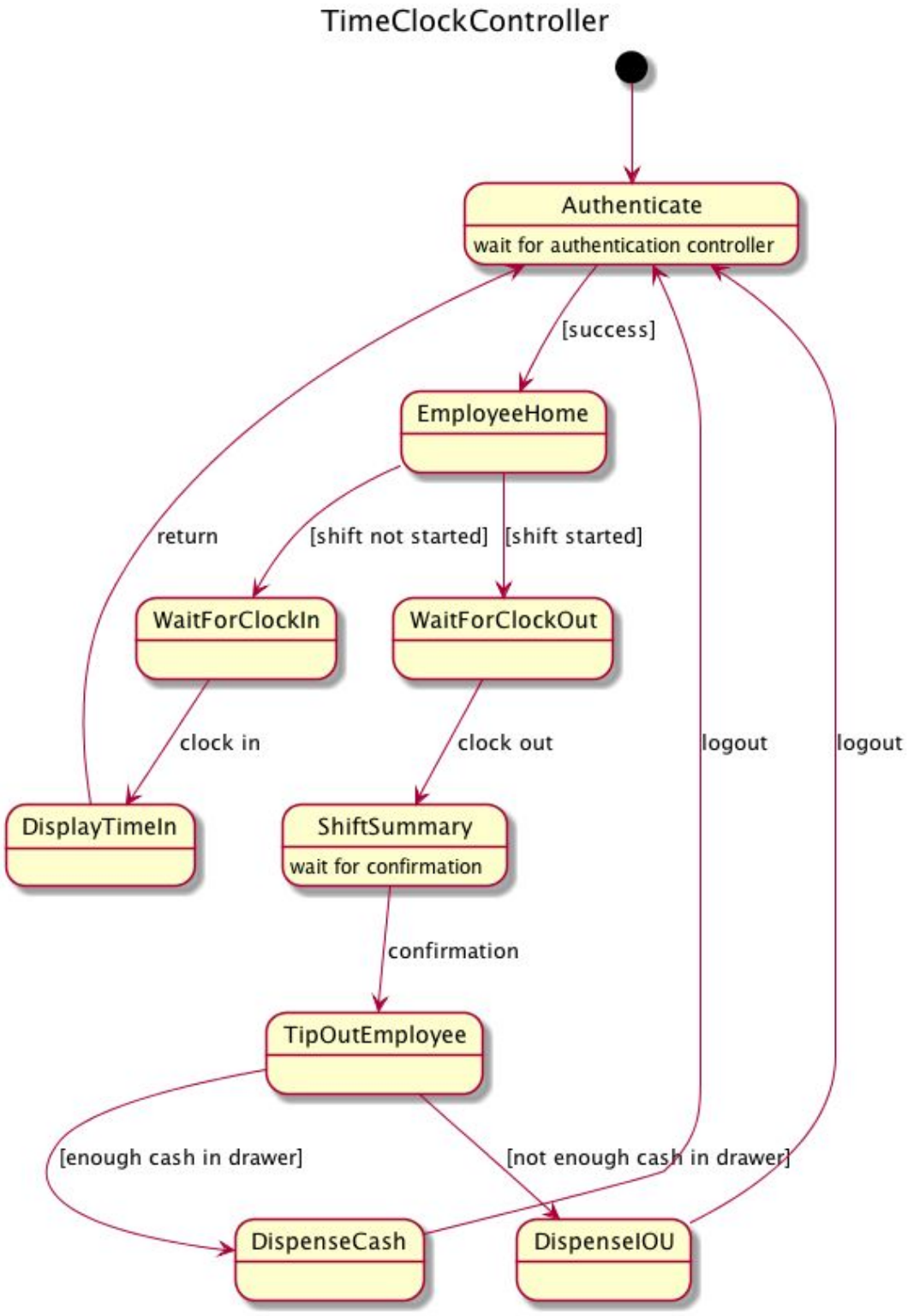
Context: Receipt
Pre: Bill.isClosed = True

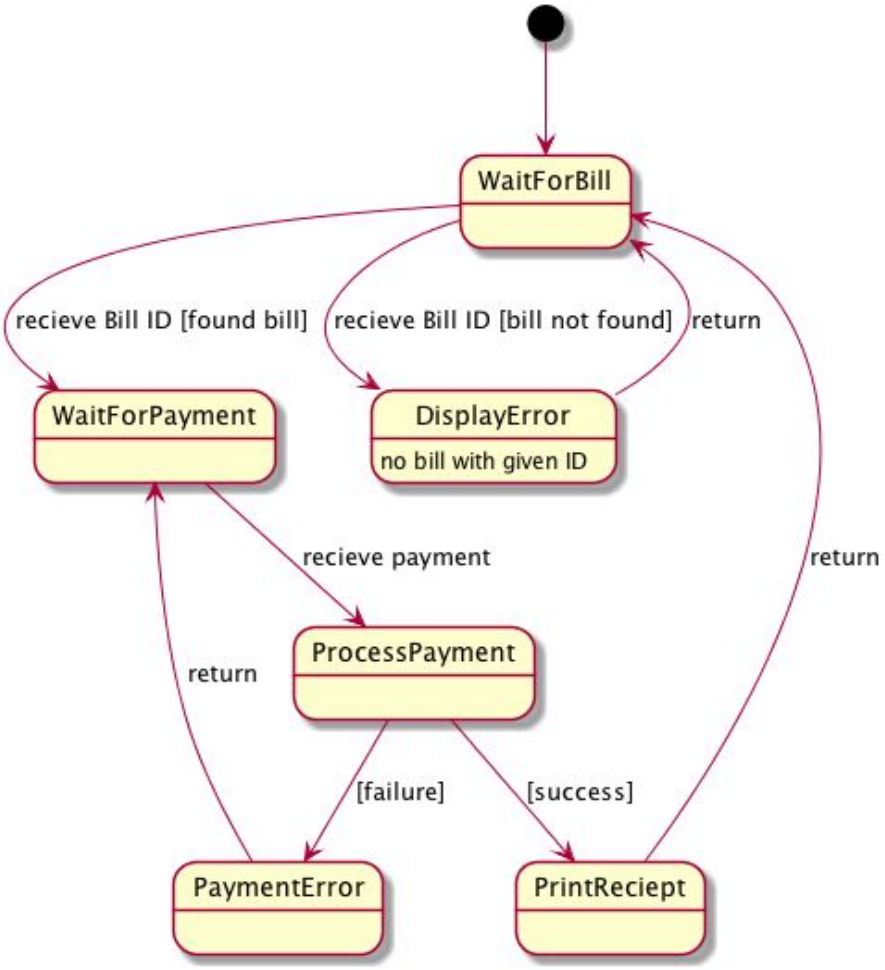
Context: changeDue
Pre: Payment.Amount > Bill.total

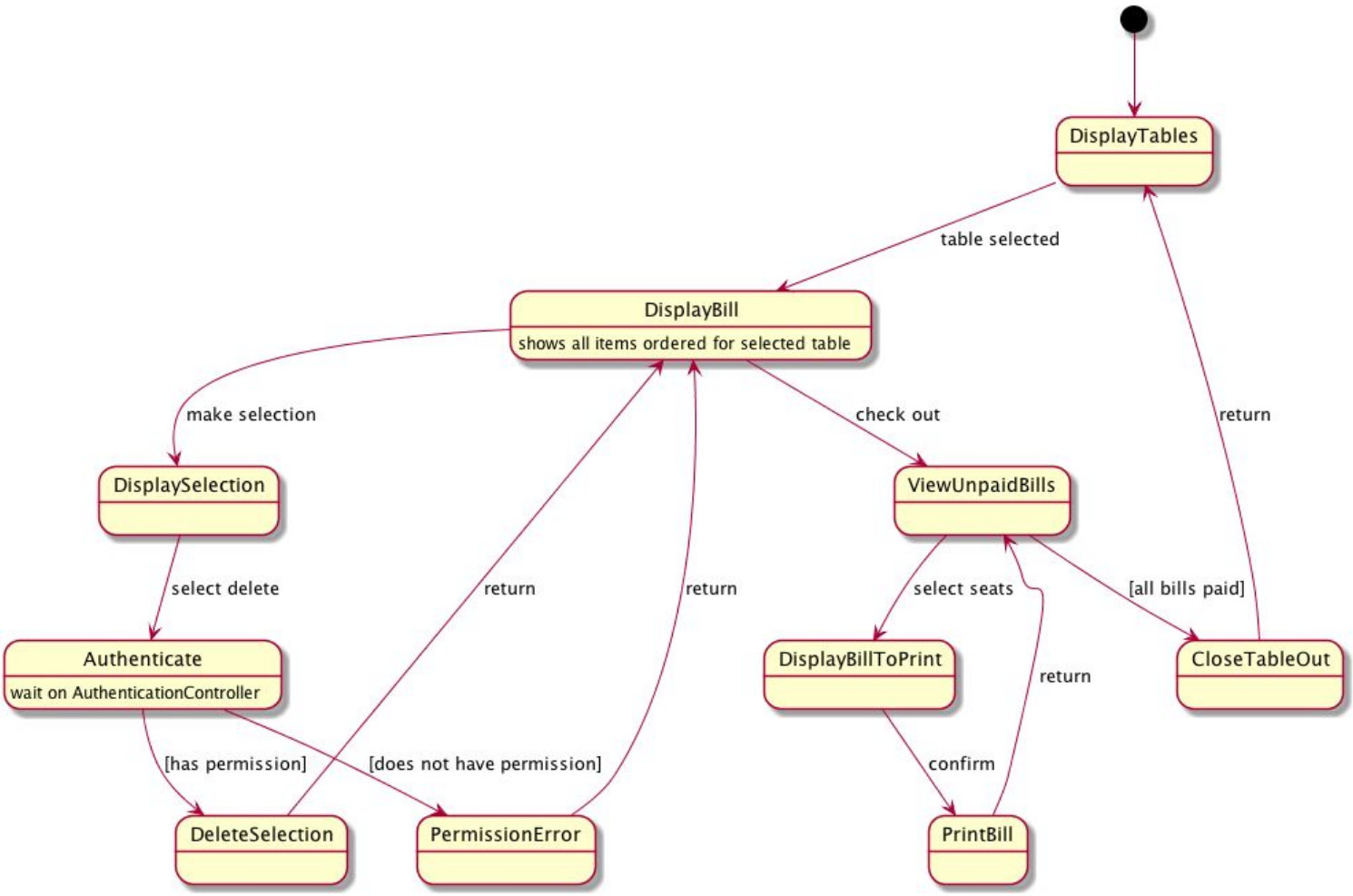
Application State Diagrams

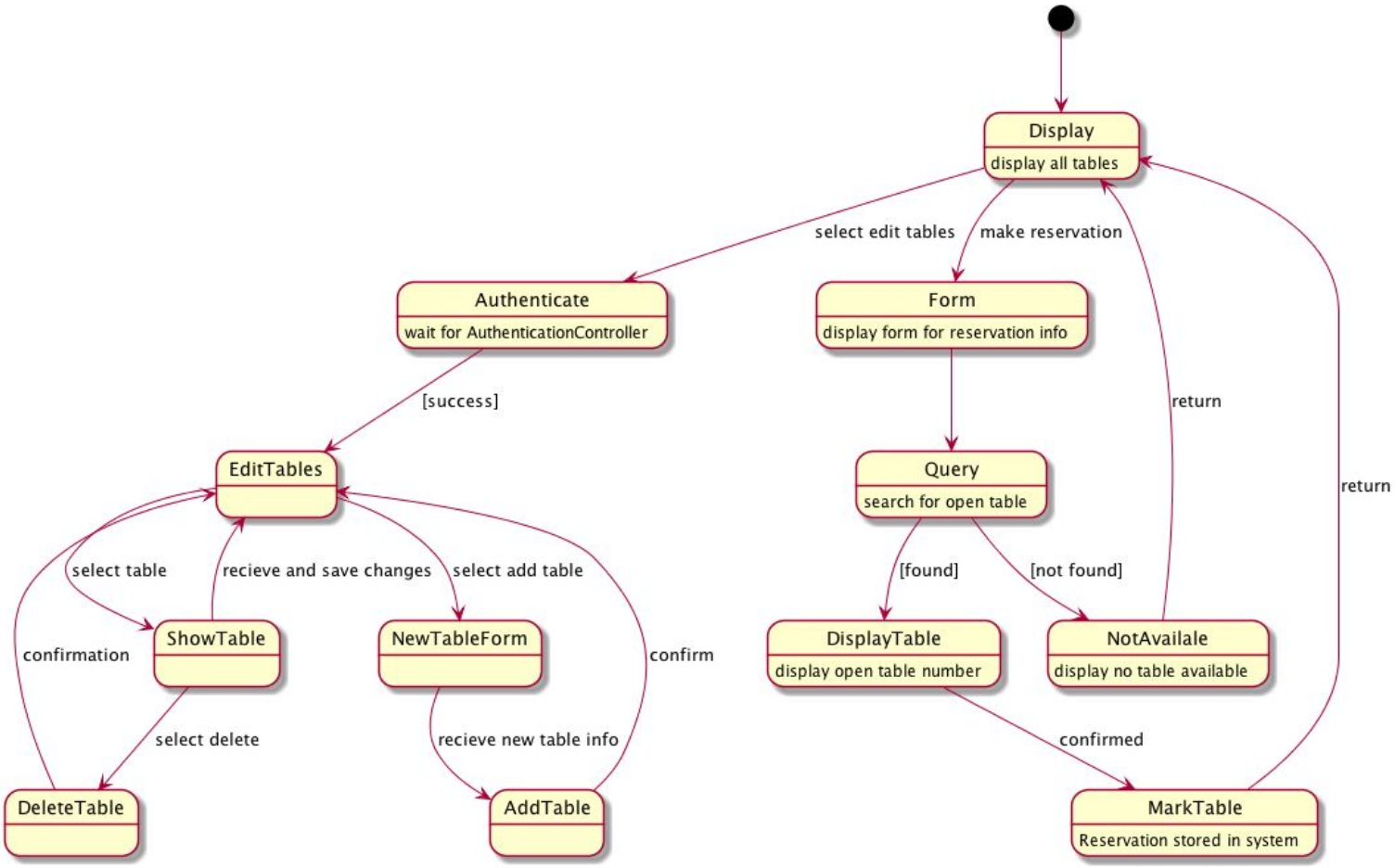


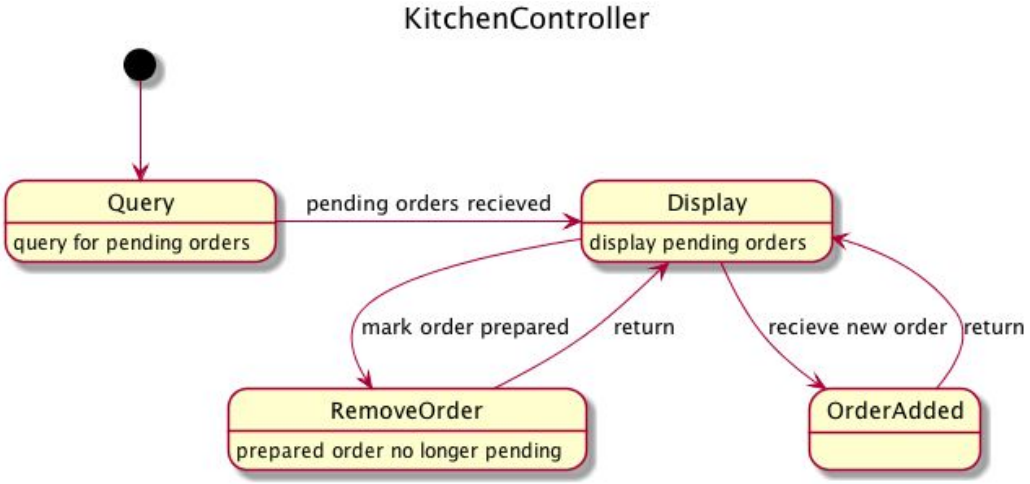

Authenticate
wait for AuthenticationController
[has permission]
[does not have permission]
DisplayMenu
AccessDenied
select item
return
add item
DisplaySelectedItem
ItemForm
edit
save changes
get item info
return
EditMenuItem
ItemPreview
confirm
AddItem


TimeClockController
Authenticate
wait for authentication controller
[success]
EmployeeHome
return
[shift not started]
[shift started]
WaitForClockIn
WaitForClockOut
clock in
clock out
logout
logout
DisplayTimeIn
ShiftSummary
wait for confirmation
confirmation
TipOutEmployee
[enough cash in drawer]
[not enough cash in drawer]
DispenseCash
DispenseIOU

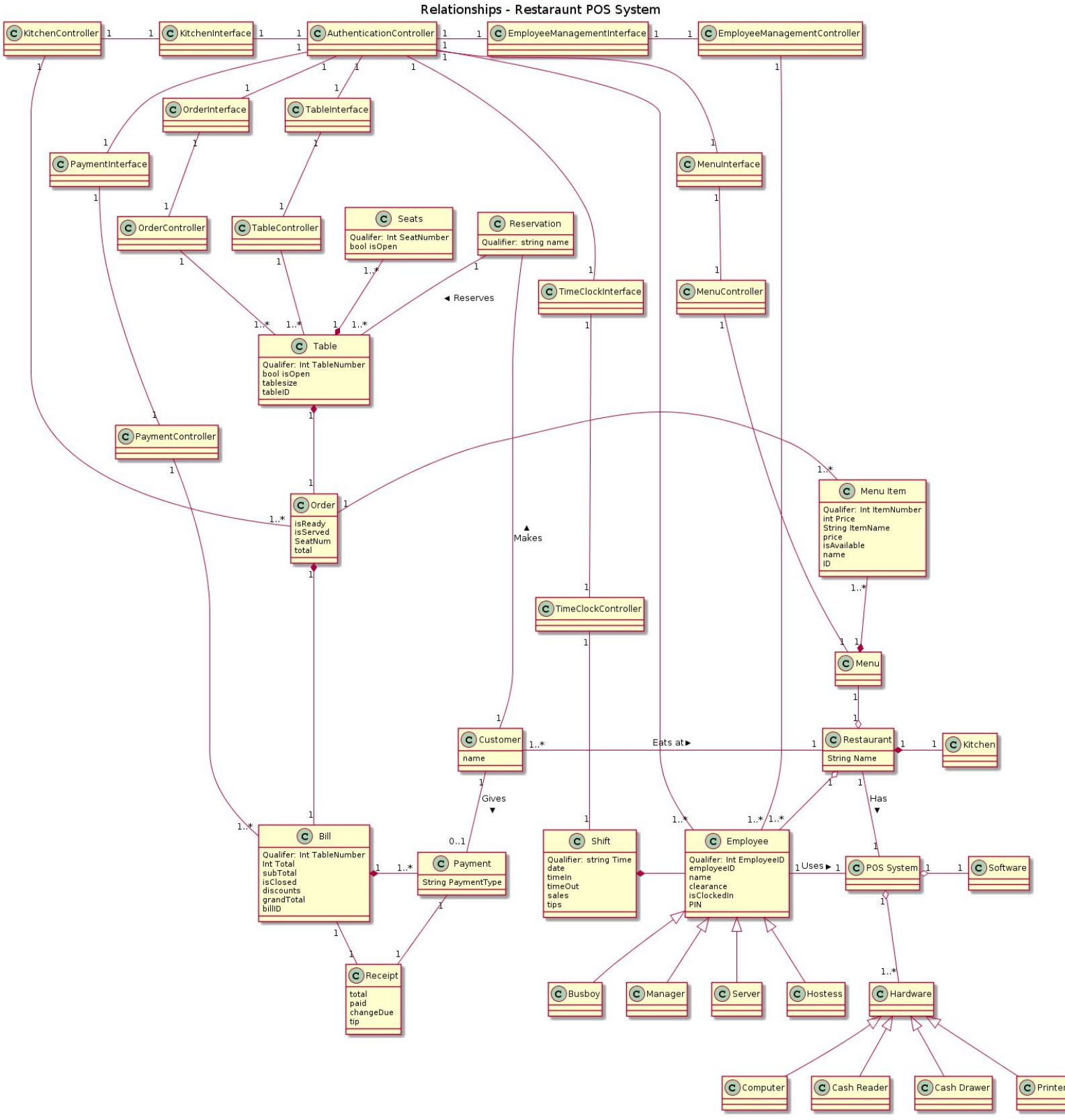

WaitForBill
recieve Bill ID [found bill]
recieve Bill ID [bill not found]
return
WaitForPayment
DisplayError
no bill with given ID
recieve payment
return
return
ProcessPayment
[failure]
[success]
PaymentError
PrintReciept


DisplayTables
table selected
DisplayBill
shows all items ordered for selected table
make selection
check out
return
DisplaySelection
ViewUnpaidBills
select delete
return
return
select seats
[all bills paid]
Authenticate
wait on AuthenticationController
DisplayBillToPrint
CloseTableOut
return
[has permission]
[does not have permission]
confirm
DeleteSelection
PermissionError
PrintBill


Display
display all tables
select edit tables
make reservation
Authenticate
wait for AuthenticationController
Form
display form for reservation info
[success]
return
EditTables
Query
search for open table
return
select table
recieve and save changes
select add table
[found]
[not found]
confirmation
ShowTable
NewTableForm
confirm
DisplayTable
display open table number
NotAvailale
display no table available
select delete
recieve new table info
confirmed
DeleteTable
AddTable
MarkTable
Reservation stored in system


KitchenController
Query
query for pending orders
pending orders recieved
Display
display pending orders
mark order prepared
return
recieve new order
return
RemoveOrder
prepared order no longer pending
OrderAdded

Consolidated Class Model



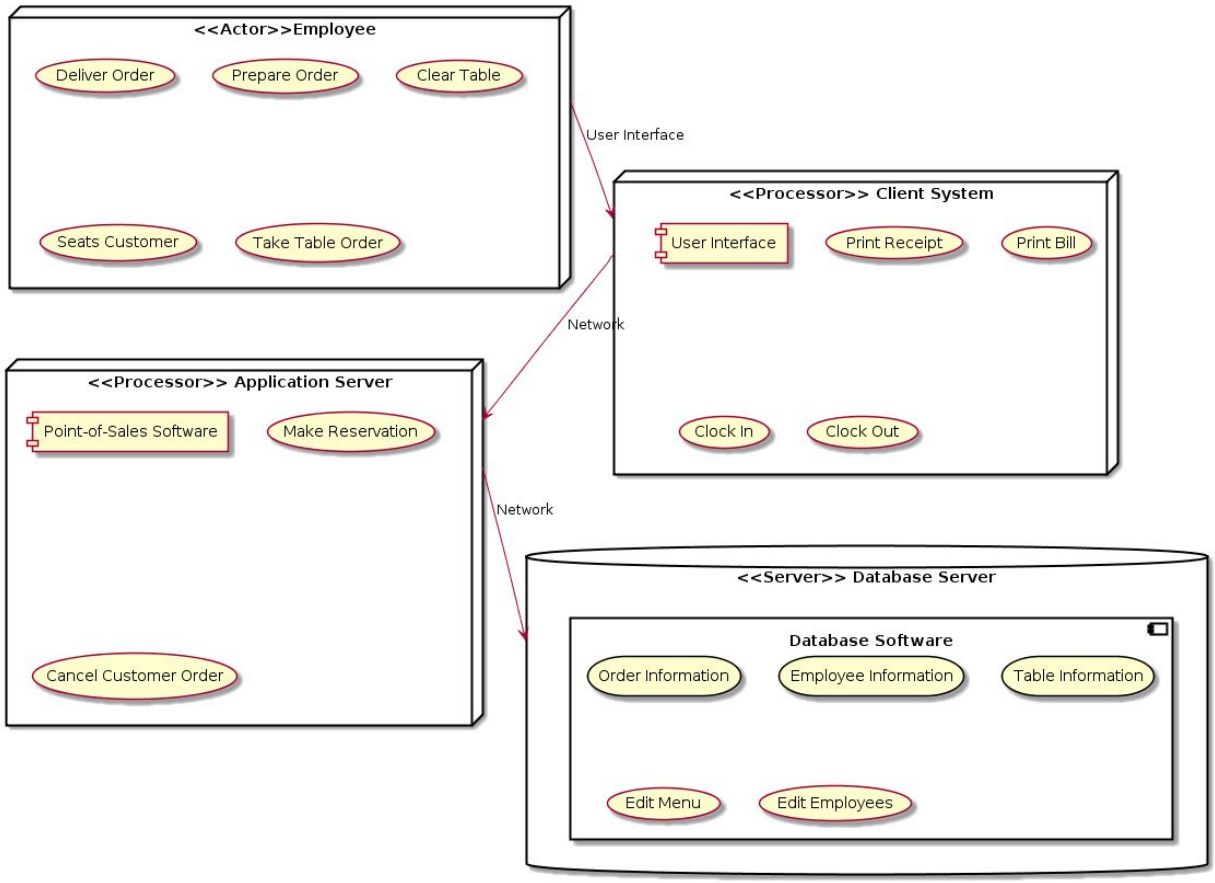
Model Review

The models created strive to meet the criteria for completeness, consistency, and correctness but fall short in some cases and excel in others. The class model addresses everything within the constraints of the concept statement. It also outlines the multiplicity as well as the relationships between classes. The class model is able to meet the criteria for correctness, completeness, and consistency. The domain state models also hold up well under the criteria for completeness, consistency, and correctness. The domain state models accurately show the transitions between states within the system and remains consistent with the domain class model. The application class diagram correctly shows the interactions between interfaces, controllers, and classes within the system. The application class diagram also remains consistent with the class diagram as it does not pull in any new classes because it only adds in how the classes from the class diagram interact with the system. Once issue with the application class diagram is that it is very complex. This leads to it being very messy and somewhat confusing. The application state diagrams also hold up well under the criteria for completeness, correctness, and consistency. The application state diagrams show the transitions between states from the application state diagram. It is able to expand upon the classes shown within the application class diagram and show the different states that occur within the system of these classes. The area where the diagrams do not hold up are within the use case diagrams. The work was divided among the group members when creating the use case diagrams. Each member had their own way of formatting and naming conventions for their use cases. Some uses cases address more exceptions within the use case while others fall

short. A few use cases fall short of meeting the criteria of completeness and correctness because the diagrams do not fully depict how the use case should function. The biggest issue is not keeping consistency between use cases. This is extremely hard when different group members have different ideas of how the use cases interact with the system.

Phase 2 Start

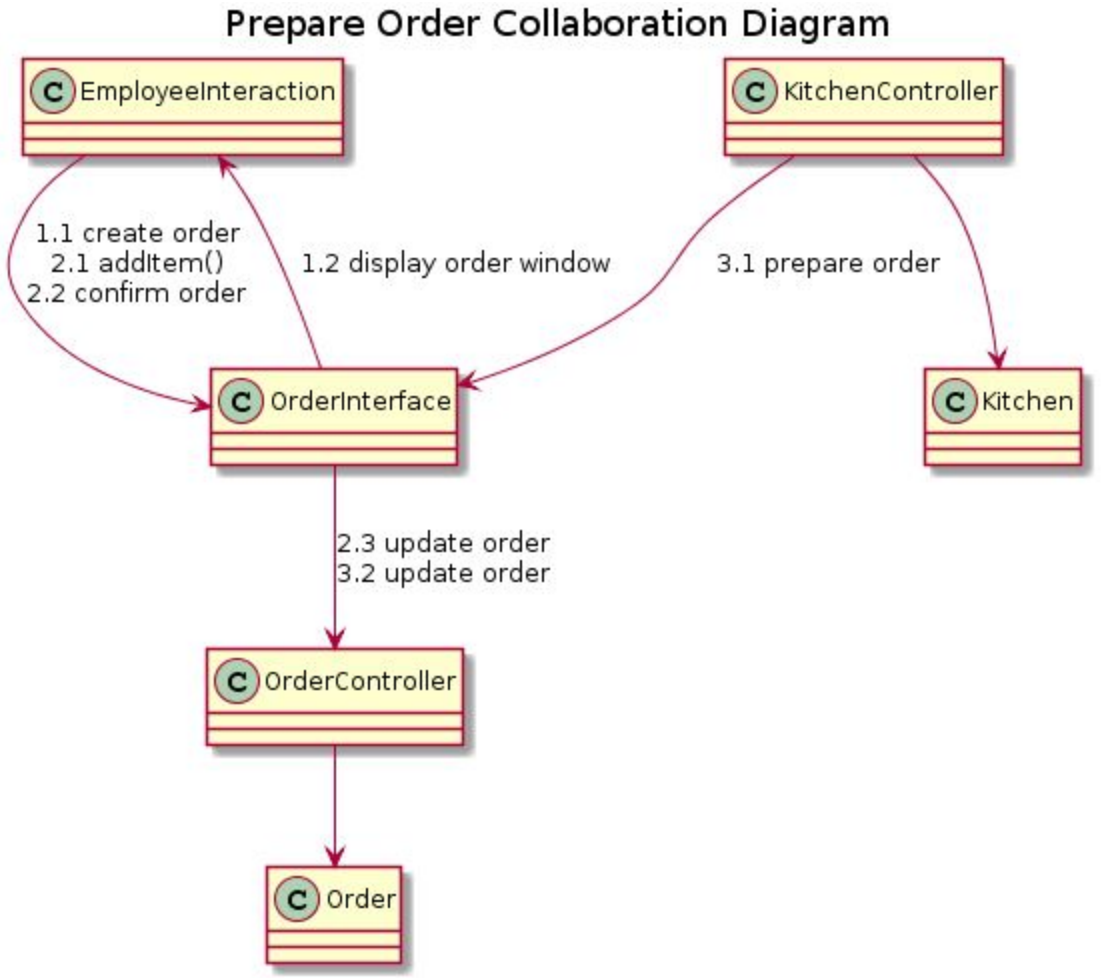
Deployment Diagram



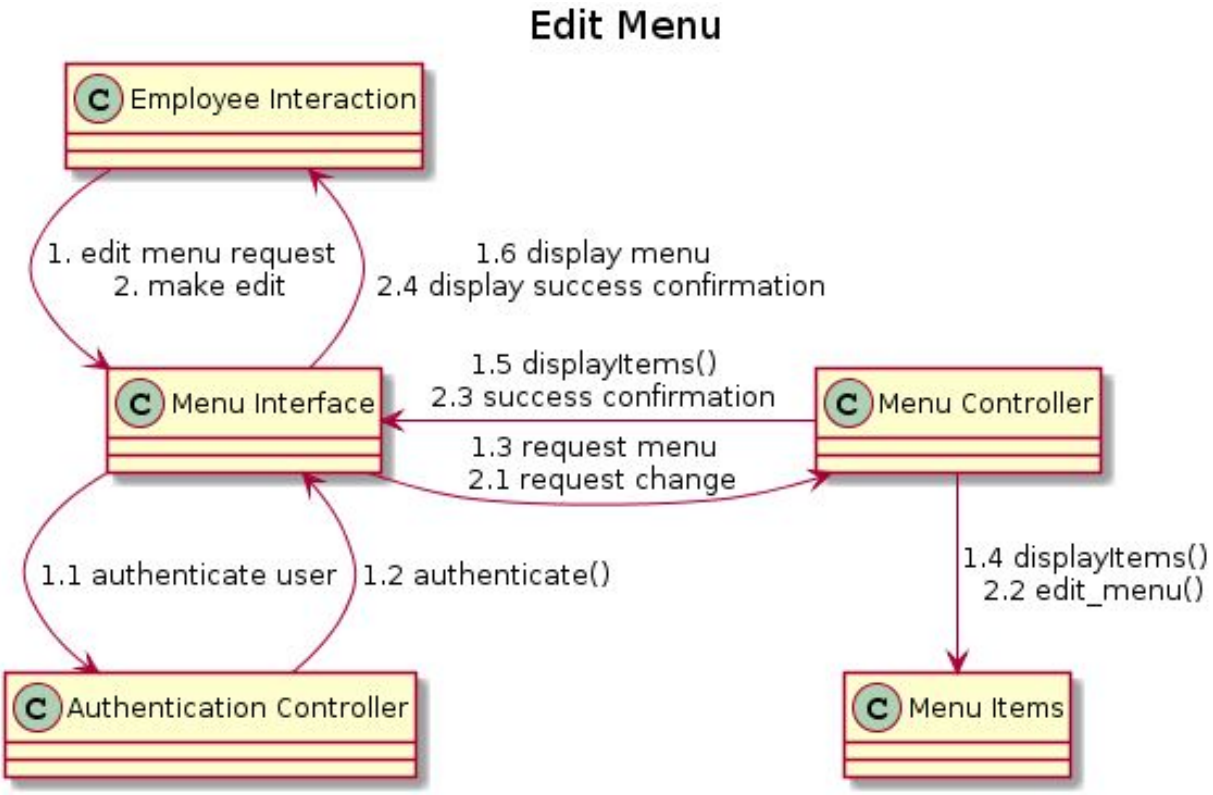
The style of the architecture that we chose is a three-tier client server. The rational for using this style of architecture was that we had three main components to our system: Client, Application server and the Database. This style was superior to the repository style due to the fact that there is no need for the client to communicate directly with the database and in fact could be problematic if it did. We want the client to only modify the database *through* the application server because it has the point-of-sales software on it. The software on the application server will control when changes are made the the database which will ensure that the data stored stays correct and consistent. This is important because making sure the customers orders are correct along with billing of the customer is correct are the top priority of the system. This style also has the benefit of having a central database which means the information retrieved from the database by each client should be consistent regardless of the client. The problems with using this style is the vulnerability with only having a single database, which can cause a complete failure of the system if there is a problem with the database, or since each system does ***not*** keep its own database a problem with simply the connection of the network could prevent any sales from being made. There could also be collisions in accessing/writing

data that should be handled because they could create inconsistencies in orders since all the clients are accessing the same database.

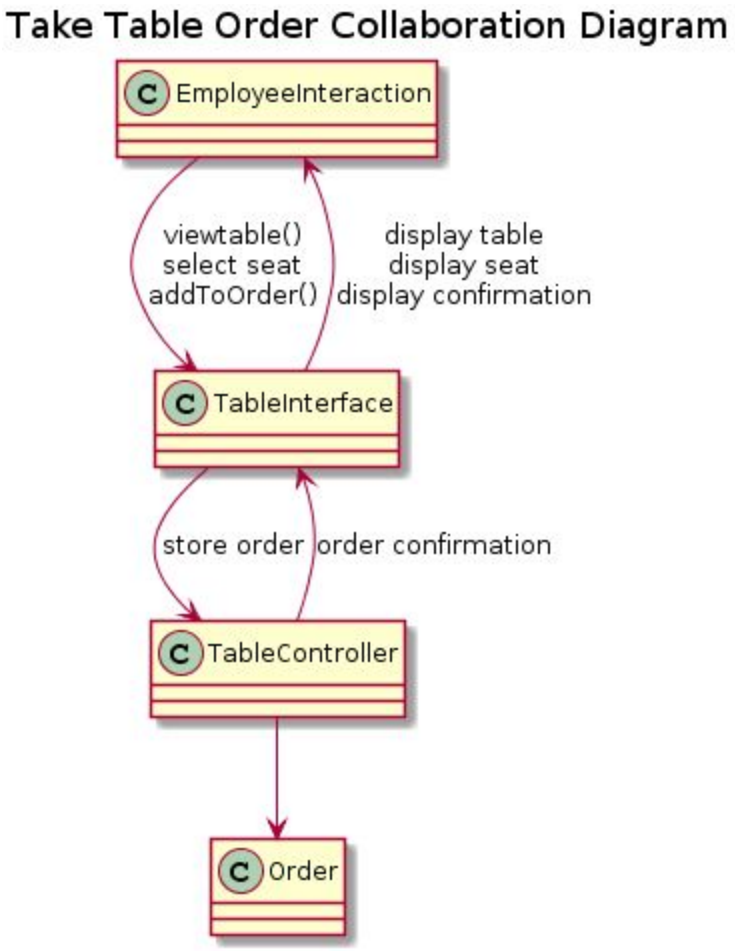
Collaboration Diagrams



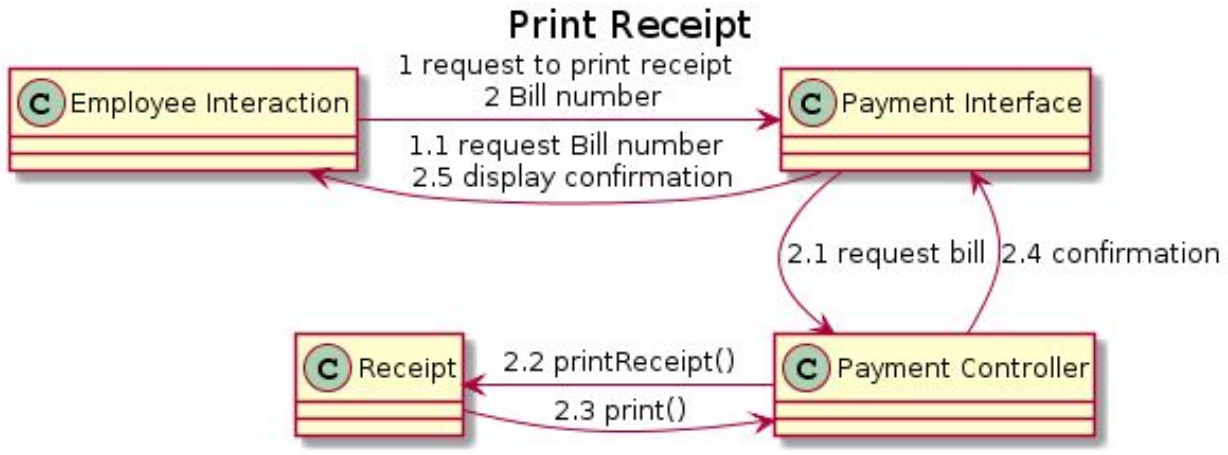
Prepare Order collaboration diagram makes use of the following GRASP patterns: Controller, Creator, Polymorphism, High Cohesion, and Low Coupling. The Controller GRASP pattern can be applied to the KitchenController and the OrderController because they control the system operation of update order and prepare order. Creator GRASP pattern can be applied because the OrderController creates an order in the system. Polymorphism GRASP patter can be applied because when the OrderController updates order it can either update by creating an order, deleting an order, or adding to an order. The High Cohesion and Low Coupling GRASP patterns can be applied because the actions performed within Prepare Order are broken down into separate classes that have specific responsibilities.



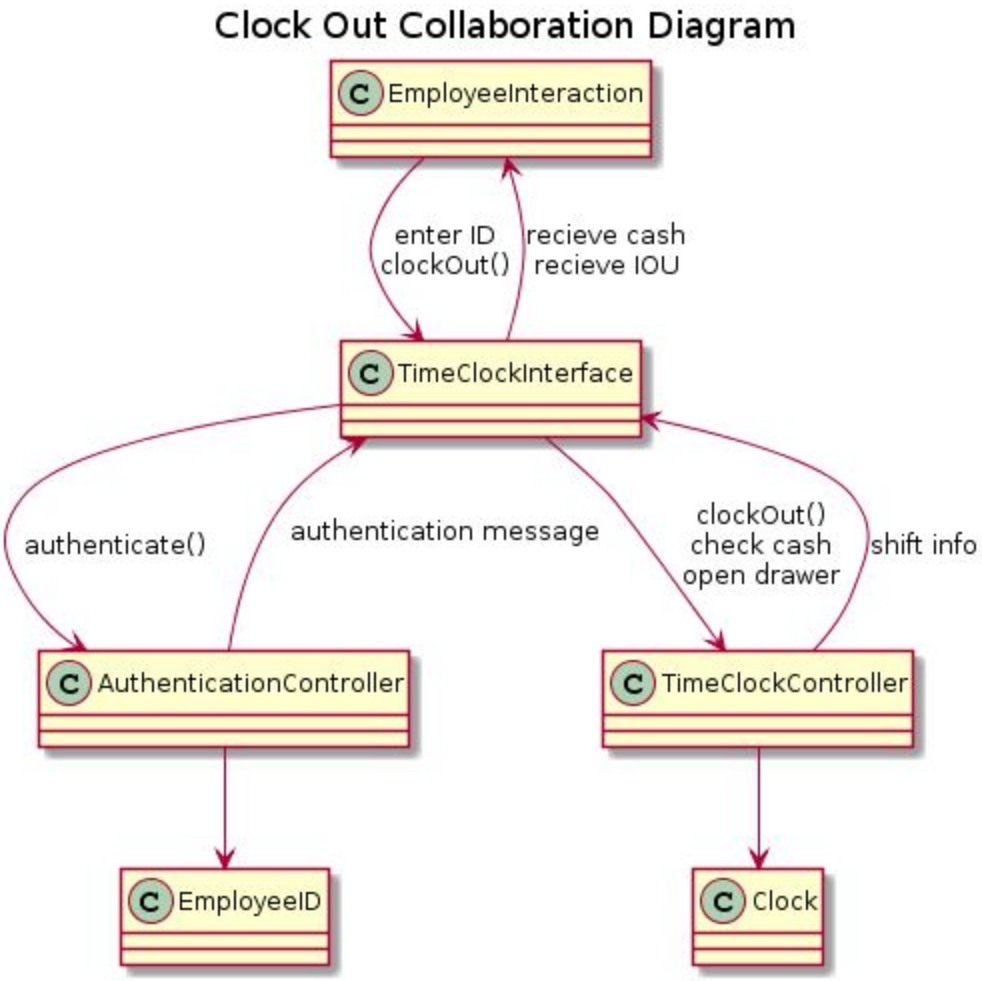
For Edit Menu's collaboration diagram overall it has both high cohesion and low coupling, the reasoning is because all of the elements have a very specific purpose (controller controls the access and manipulation of the menu and menu items, authentication controller only authenticates users, and so on) and other than employee interaction with the system everything has a low reliance on any other part even though they do interact. The controller as the name would suggest is obviously a controller as well as a creator since it creates and handles menu item objects. I would also say that it uses Expert due to the fact that within itself it is delegating information and responsibility but it is also delegating the information to other systems Lastly the interface uses polymorphism dealing with various kinds of input and interpreting it and handling it as needed.



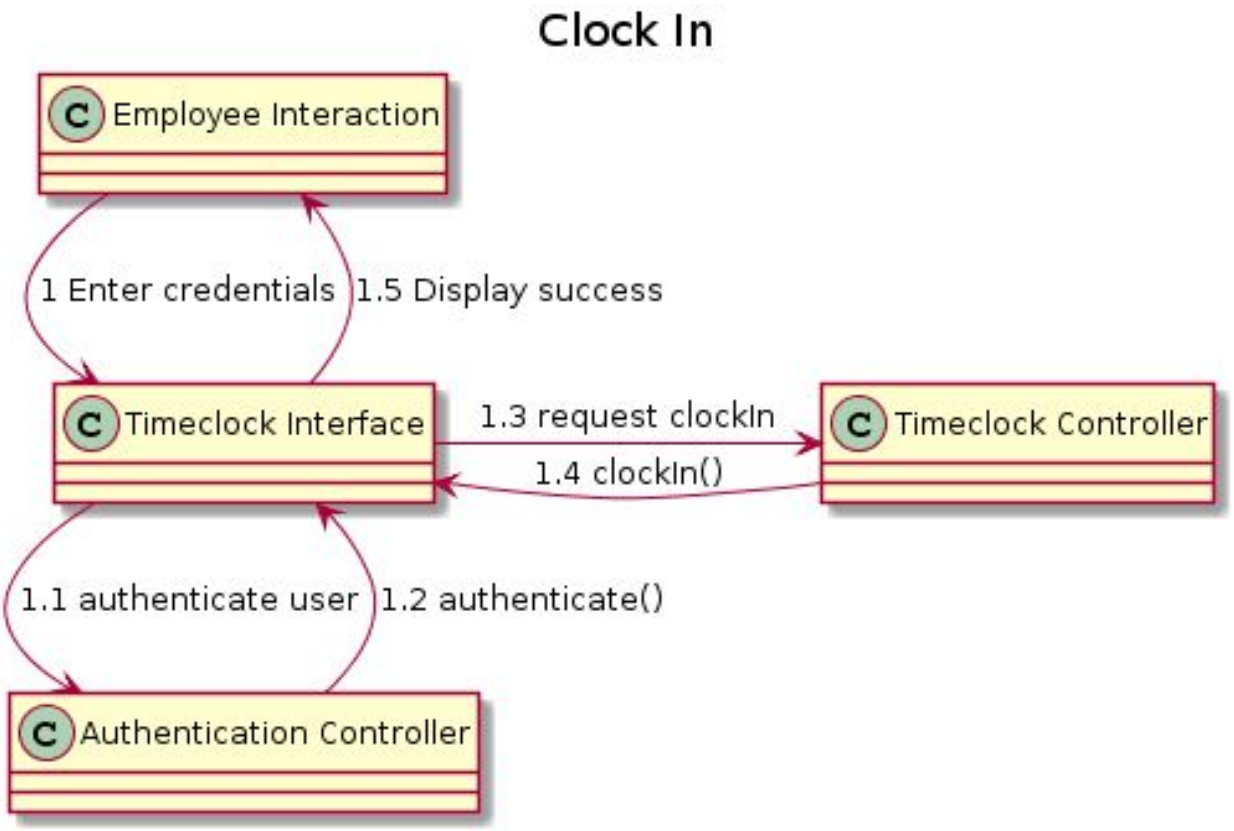
Take Table Order collaboration diagram makes use of the following GRASP patterns: Controller, Polymorphism, High Cohesion, and Low Coupling. The Controller GRASP pattern can be applied to the TableController because it controls the system operation of update order and prepare order. Polymorphism GRASP patter can be applied because the TableInterface because it handles various behaviors that are sent to it by the Employee when the employee interacts with the interface. The High Cohesion and Low Coupling GRASP patterns can be applied because the actions performed within Take Table Order are broken down into separate classes that have specific responsibilities. The interface handles the interaction between the customer and the system and displays different pages depending on what the customer has selected. The controller handles the storing of the order information and sending back a confirmation to the interface that can then be displayed to the Employee.



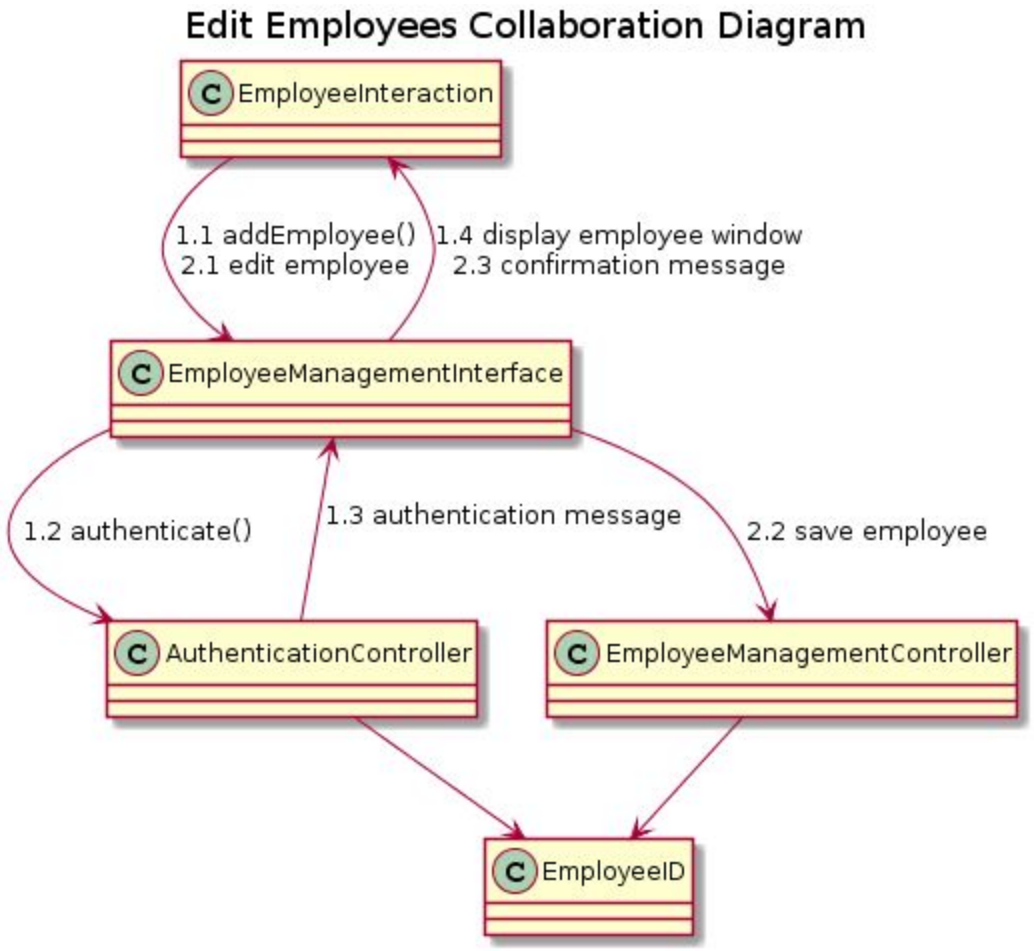
For print receipt's collaboration diagram overall it has both high cohesion and low coupling, the reasoning is because all of the elements have a very specific purpose, controller controls the access and actions with receipts, authentication controller only authenticates users, and so on). Other than employee interaction with the system everything has a low reliance on any other part even though they do interact. The controller as the name would suggest is obviously a controller as well as a creator since it creates and handles receipt objects. Lastly the interface uses polymorphism dealing with various kinds of input and interpreting it and handling it as needed.



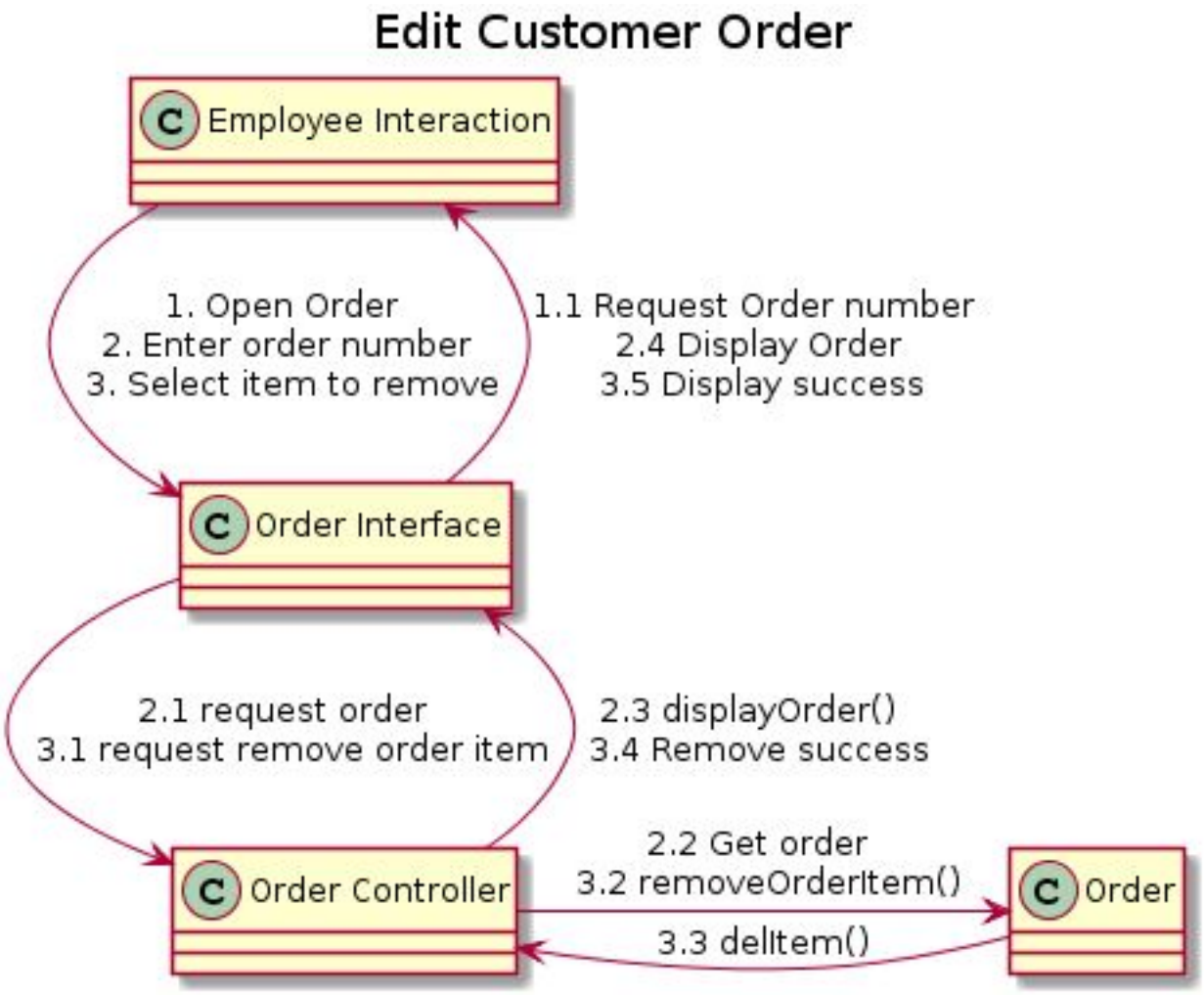
Clock Out collaboration diagram makes use of the following GRASP patterns: Controller, Polymorphism, High Cohesion, and Low Coupling. The Controller GRASP pattern can be applied to the AuthenticationController and the TimeClockController because they control the operations of authentication of an employee ID, employee clocking out, and checking for money. Polymorphism GRASP pattern can be applied because when the AuthenticationController handles authentication of employee ID it can be checking if the employee has permission to clock in or edit employees in the system. In this case the authenticate method is being used to check the employee ID before clocking out. The High Cohesion and Low Coupling GRASP patterns can be applied because the actions performed within Clock out are broken down into separate classes that have specific responsibilities. The authentication of the employee is handled through the controller and does not depend on how the actual clocking out of the employee is handled within the TimeClockController.



For clock in's collaboration diagram overall it has both high cohesion and low coupling, the reasoning is because all of the elements have a very specific purpose, controller controls the access and actions with receipts, authentication controller only authenticates users, and so on). Other than employee interaction with the system everything has a low reliance on any other part even though they do interact. The controller as the name would suggest is obviously a controller. Lastly the interface uses polymorphism dealing with various kinds of input and interpreting it and handling it as needed.

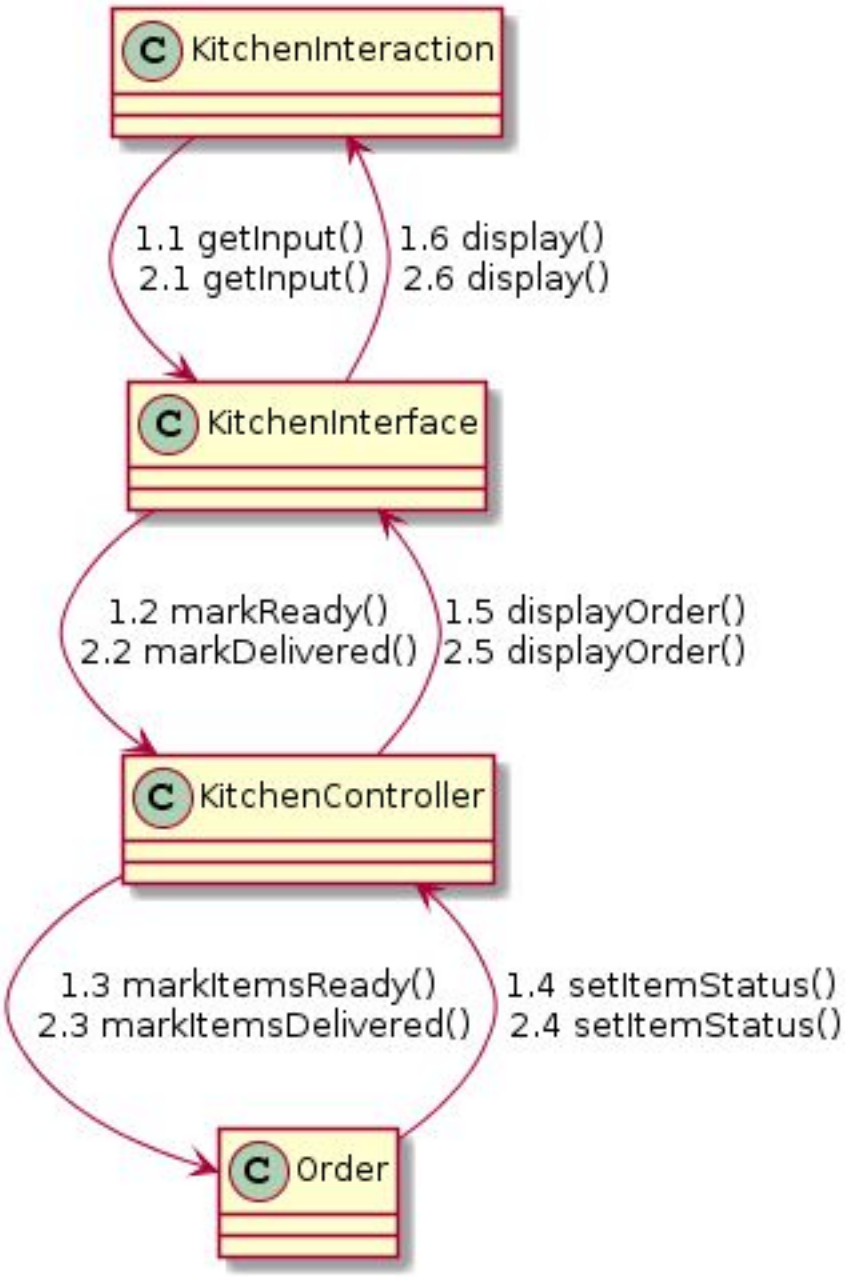


Edit Employees collaboration diagram makes use of the following GRASP patterns: Controller, Creator, Polymorphism, High Cohesion, and Low Coupling. The Controller GRASP pattern can be applied to the AuthenticationController and the EmployeeManagementController because they control the system operation of authenticating an employee ID and saving an employee's information. Creator GRASP pattern can be applied because the EmployeeManagementController creates a new employee ID within the system Polymorphism GRASP pattern can be applied because when the AuthenticationController handles authentication of employee ID it can be checking if the employee has permission to clock in or edit employees in the system. In this case the authenticate method is being used to check the employee ID before it gives permission to allow the employee to edit employees in the system. The High Cohesion and Low Coupling GRASP patterns can be applied because the actions performed within Prepare Order are broken down into separate classes that have specific responsibilities. For example, the AuthenticationContoller has the responsibility of authenticating the employee's ID and the EmployeeManagmentController has the responsibility of updating an employee's information in the system and are not directly connected to one another.



For edit customer order's collaboration diagram overall it has both high cohesion and low coupling, the reasoning is because all of the elements have a very specific purpose, controller controls the access and actions with receipts, authentication controller only authenticates users, and so on). Other than employee interaction with the system everything has a low reliance on any other part even though they do interact. The controller as the name would suggest is obviously a controller as well as a creator since it creates and handles order objects. I would also say that overall it is an expert because it its controlling the information for the menu which is used by most systems. Lastly the interface uses polymorphism dealing with various kinds of input and interpreting it and handling it as needed.

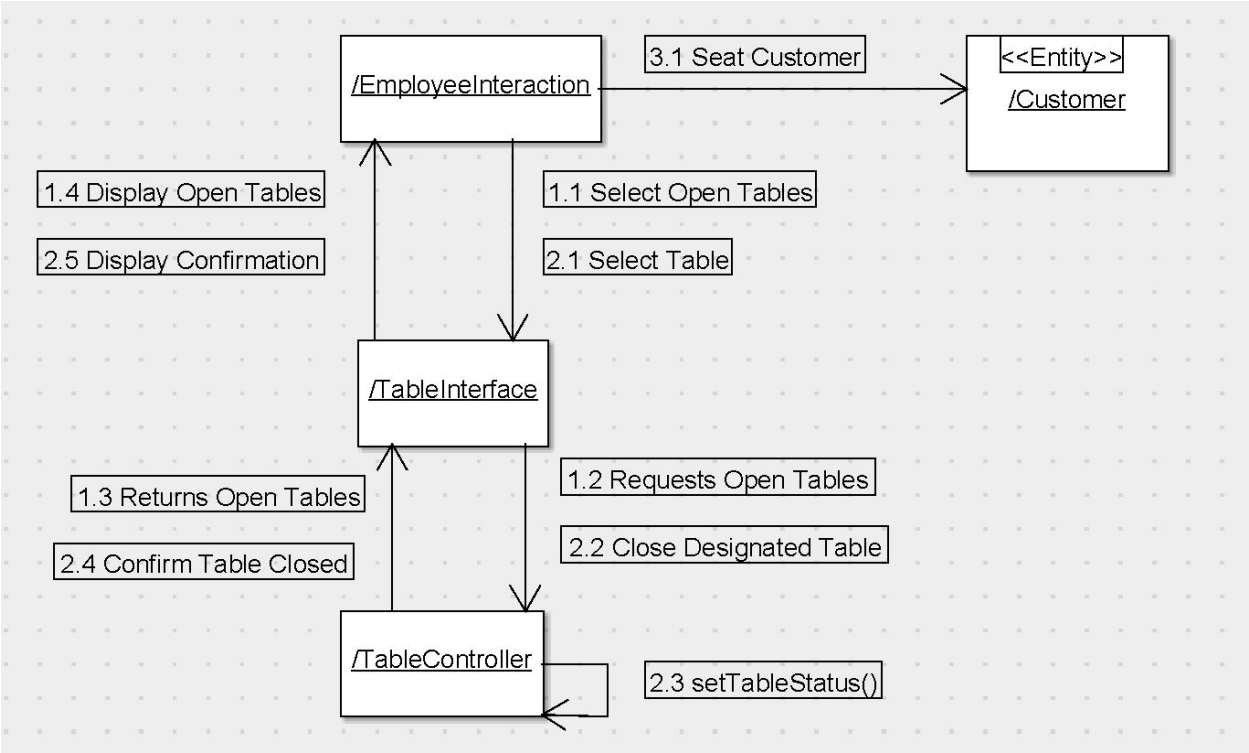
Deliver Order



For deliver order's collaboration diagram overall it has both high cohesion and low coupling, the reasoning is because all of the elements have a very specific purpose, controller controls the access and actions with orders, and so on. Other than employee interaction with the system

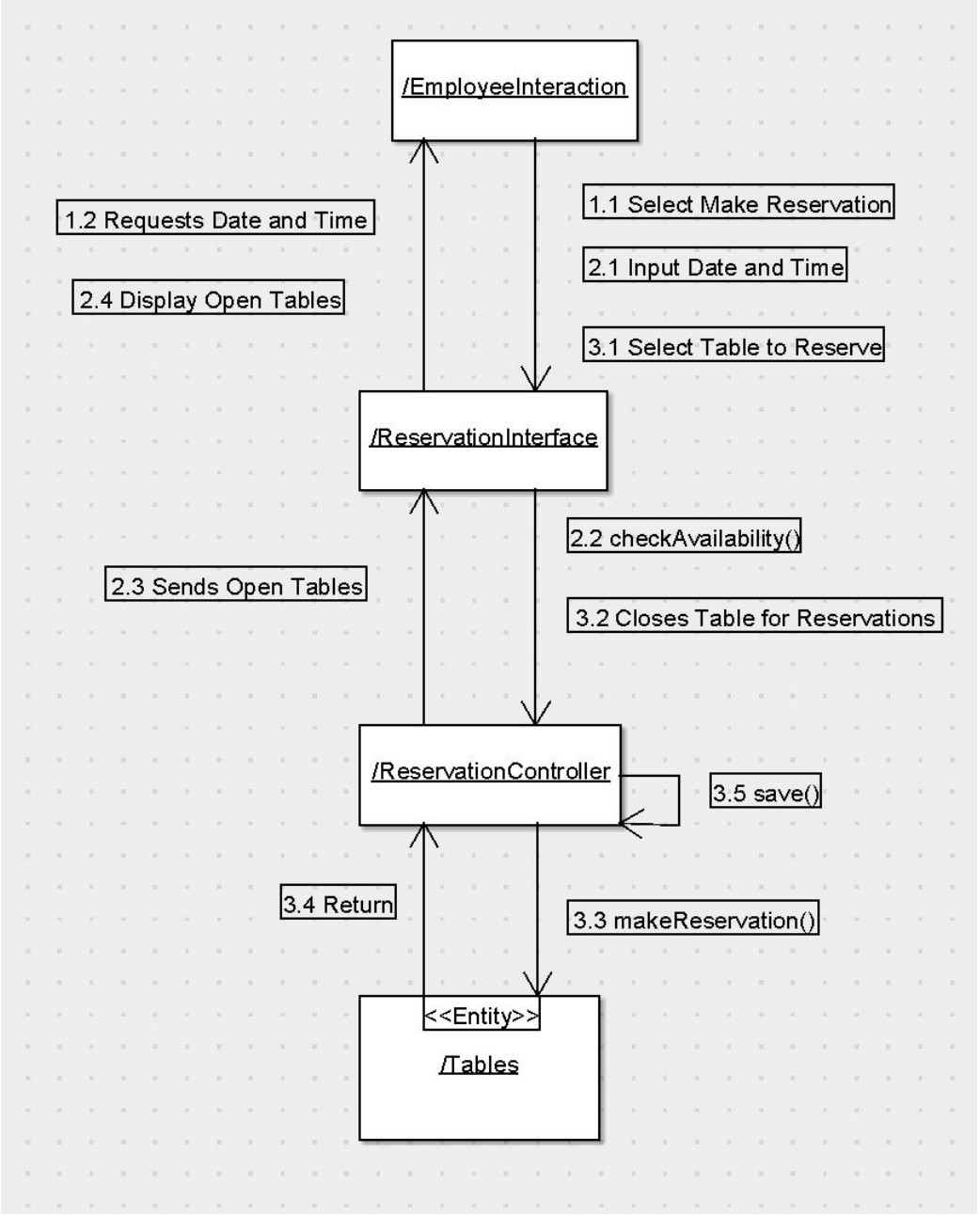
everything has a low reliance on any other part even though they do interact. The controller as the name would suggest is obviously a controller, but also a creator because it has the ability to create and delete order objects. Lastly the interface uses polymorphism dealing with various kinds of input and interpreting it and handling it as needed.

Seat Customer



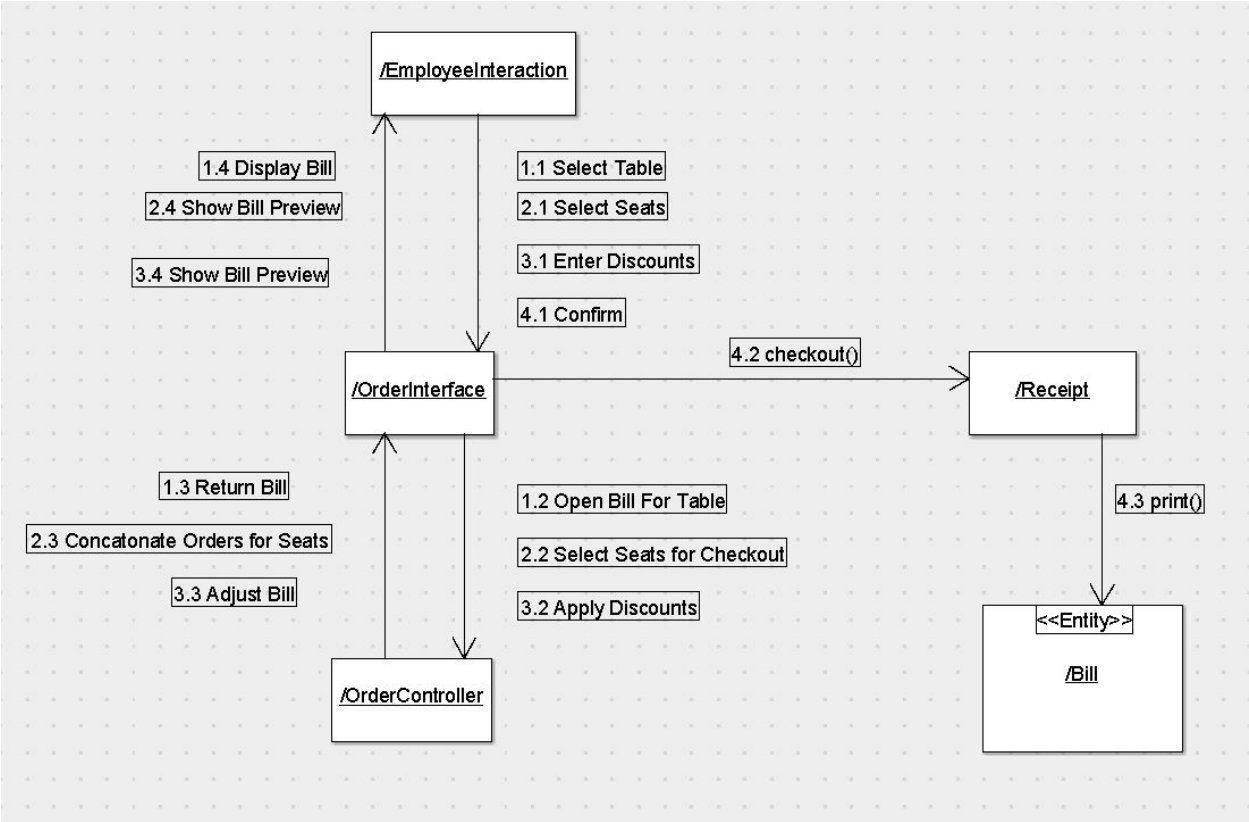
The Expert pattern is used by the TableController, because it knows the only information required to seat a customer. It adjusts that information with the Controller pattern that sets the status of the table. The coupling is low since the TableController is only in charge of interactions involving the tables. It is not required to perform a wide variety of actions. The TableInterface is responsible for the different user inputs which uses the Polymorphism pattern.

Make Reservation



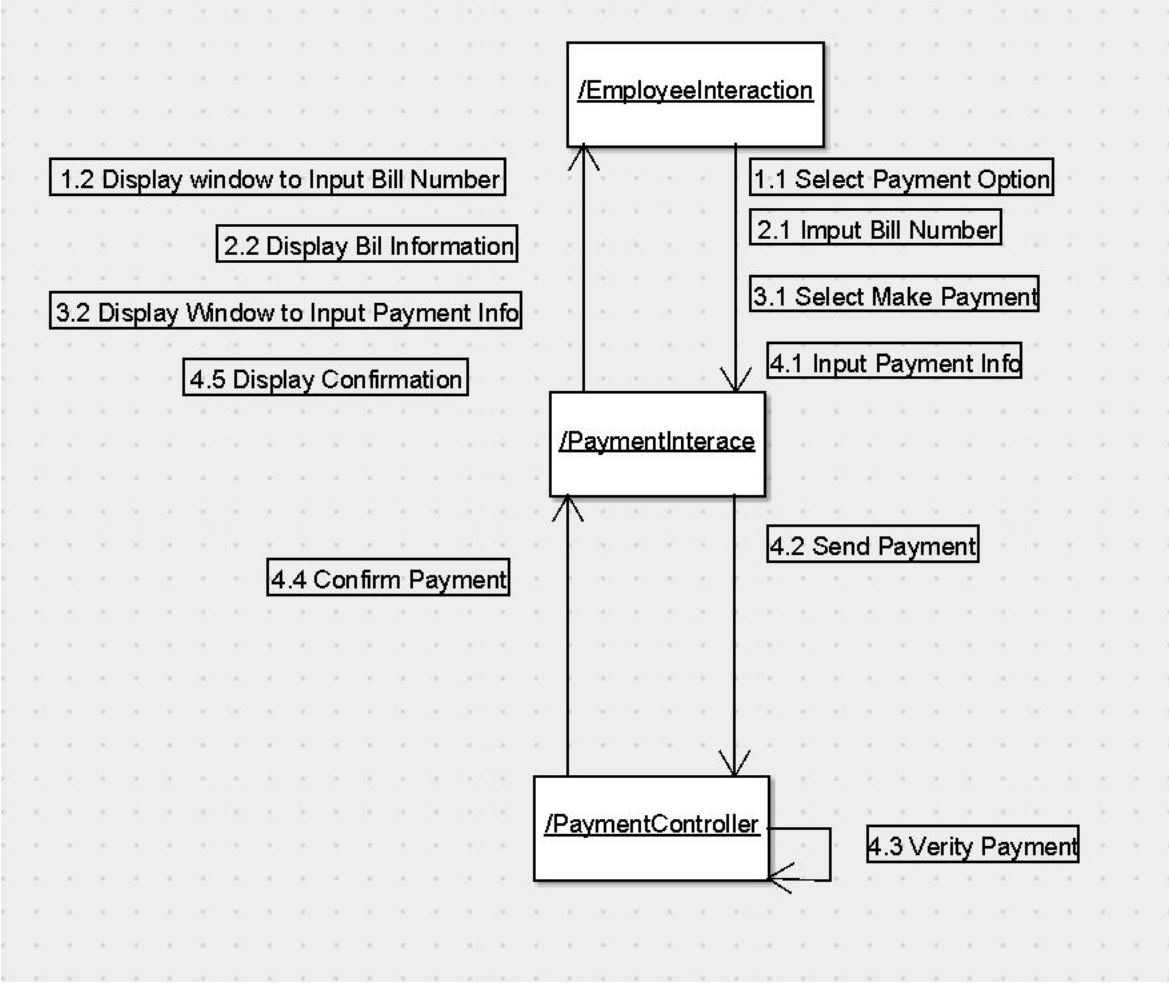
The ReservationController accounts for three different GRASP patterns. It uses the pattern Expert as it knows all of the required information to know what reservations are available at which times. It does not require outside help to obtain the information. Controller is used because it is responsible for receiving and handling system events. Creator is used when it physically creates the reservation. Since this controller only objective is to handle reservations, coupling remains low. The user interface uses the Polymorphism pattern since it is in charge of handling multiple different inputs depending on what action if being performed. High Cohesion is used along side low coupling. This is achieved because the responsibilities of the controller are only focused on one simple area, instead of being in charge of multiple different types of actions.

Print Bill



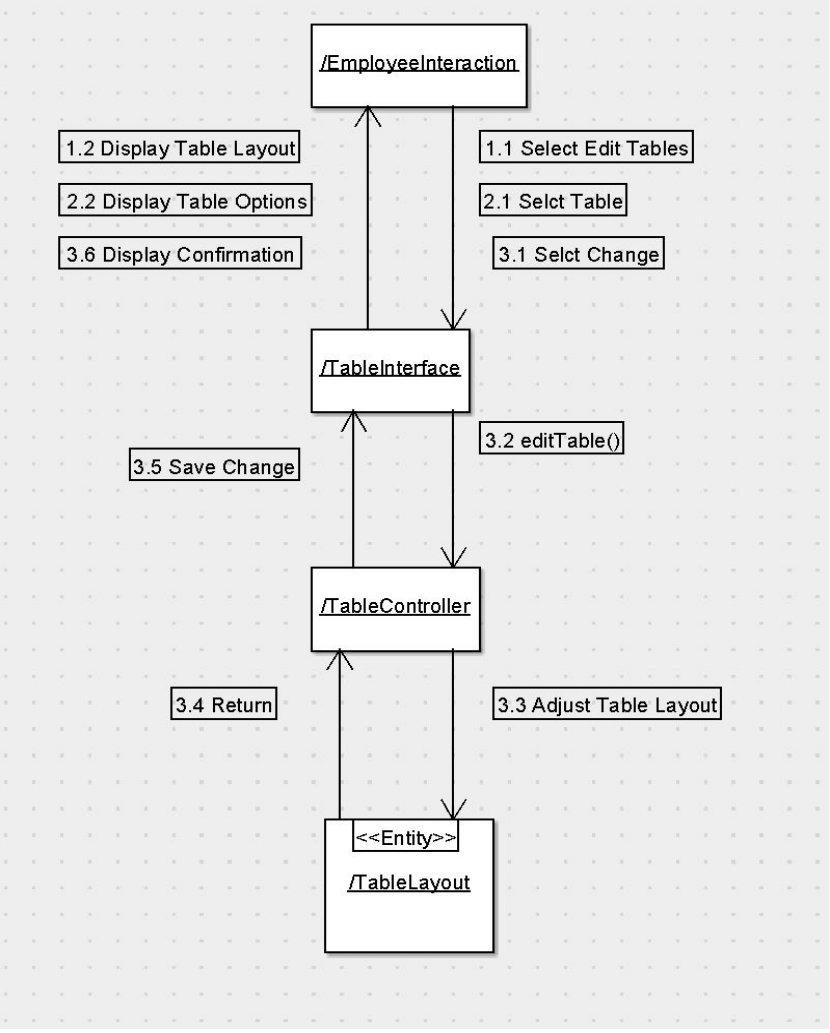
The Creator pattern is used as the final bill is created and printed. The OrderController uses the Controller pattern to receive and handle the system events that it is in charge of. It also uses the Expert pattern since the OrderController contains all of the information needed to format and print the bill. The cohesion for the OrderController is not very high since it is in charge of a multitude of tasks, but the coupling will be low since all the tasks for this controller revolve around the orders, and nothing else. Finally, the OrderInterface is responsible for the various user inputs uses the Polymorphism

Make Payment



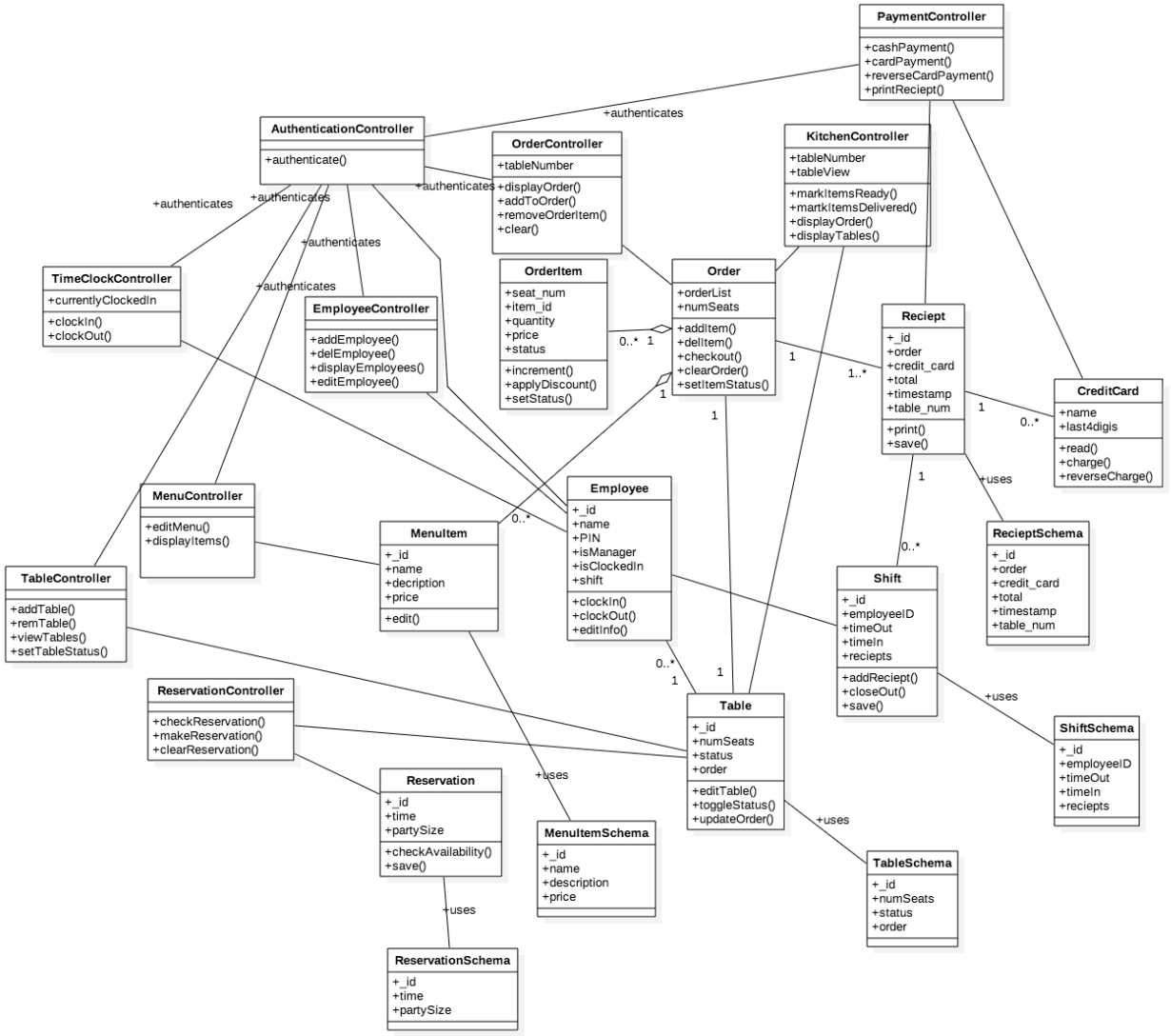
The PaymentController uses the Polymorphism pattern through the user interface, and through the action of accepting multiple different types of payments. The controller pattern is used to verify and complete the actions required throughout the payment process. There is Low Coupling since the PaymentController's only job is to handle payments. This also results in high cohesion, since it only has to perform a few simple actions.

Edit Table



The TableController uses the Controller pattern to perform the actions required while editing the layout of the tables. It uses the creator pattern to build the layout for the tables. Since it is responsible for designing the table layout, it also has all of the information required to edit the current tables. This uses the Expert pattern. The TableController does a variety of different actions which lowers the cohesion, but the coupling remains low since editing the tables is not affected by other methods, and the TableController is only in charge of the tables.

Design Class Diagram

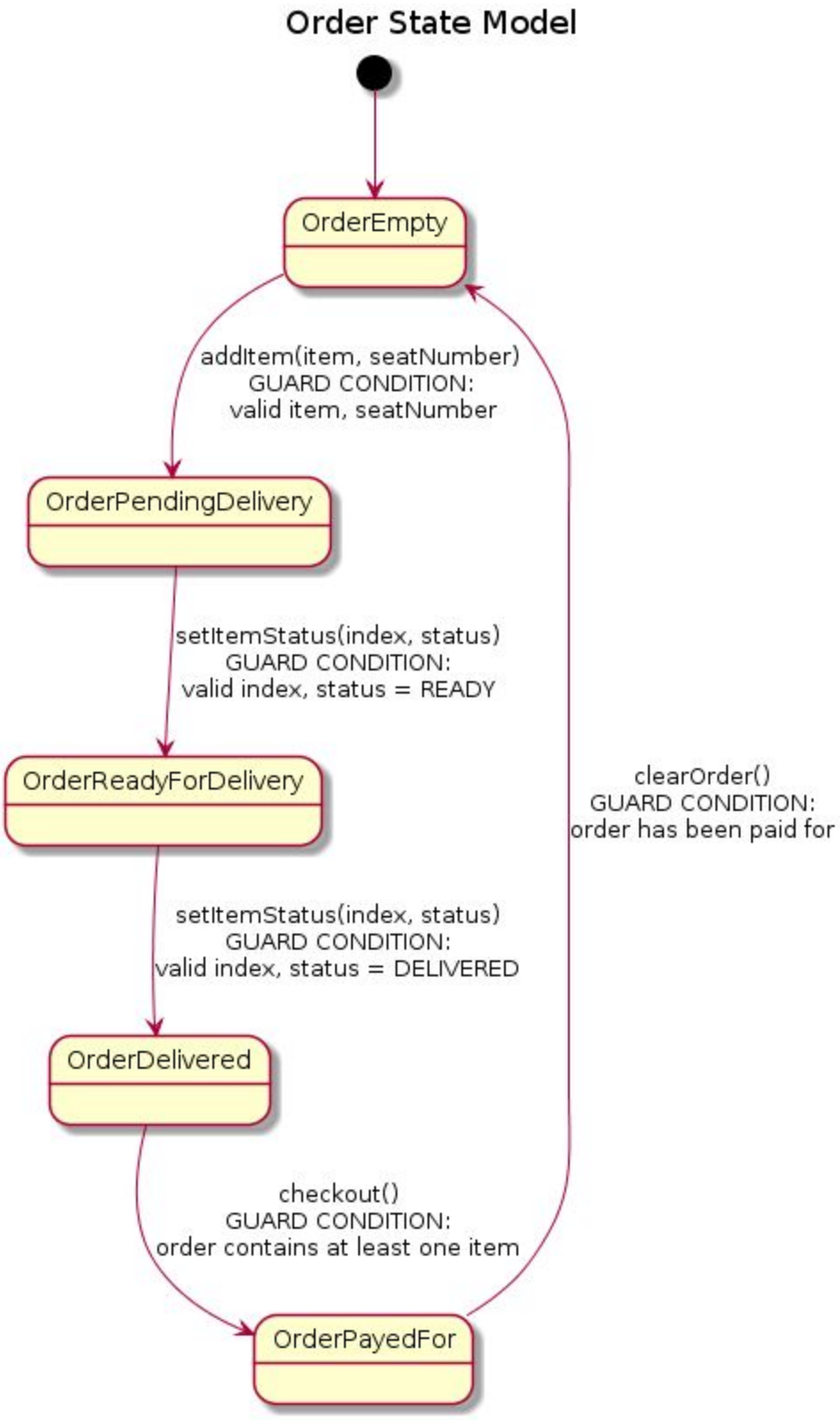


Class Design Patterns

Every time a user interacts with the system he/she must do it through one of the controller objects. These controller objects are examples of the facade design pattern. The facade design pattern provides an interface for the user to interact with the other objects in the system. For example, the TableController class is a facade for interacting with Table objects.

Objects in the database exist as an instance of one of the schema objects. Instead of manipulating these objects directly in the database, we maintain proxy objects that we can edit and then forward to the real objects in the database. Every schema object in our class design has a corresponding proxy object. For example, the Table class is a proxy for TableSchema.

Object Design

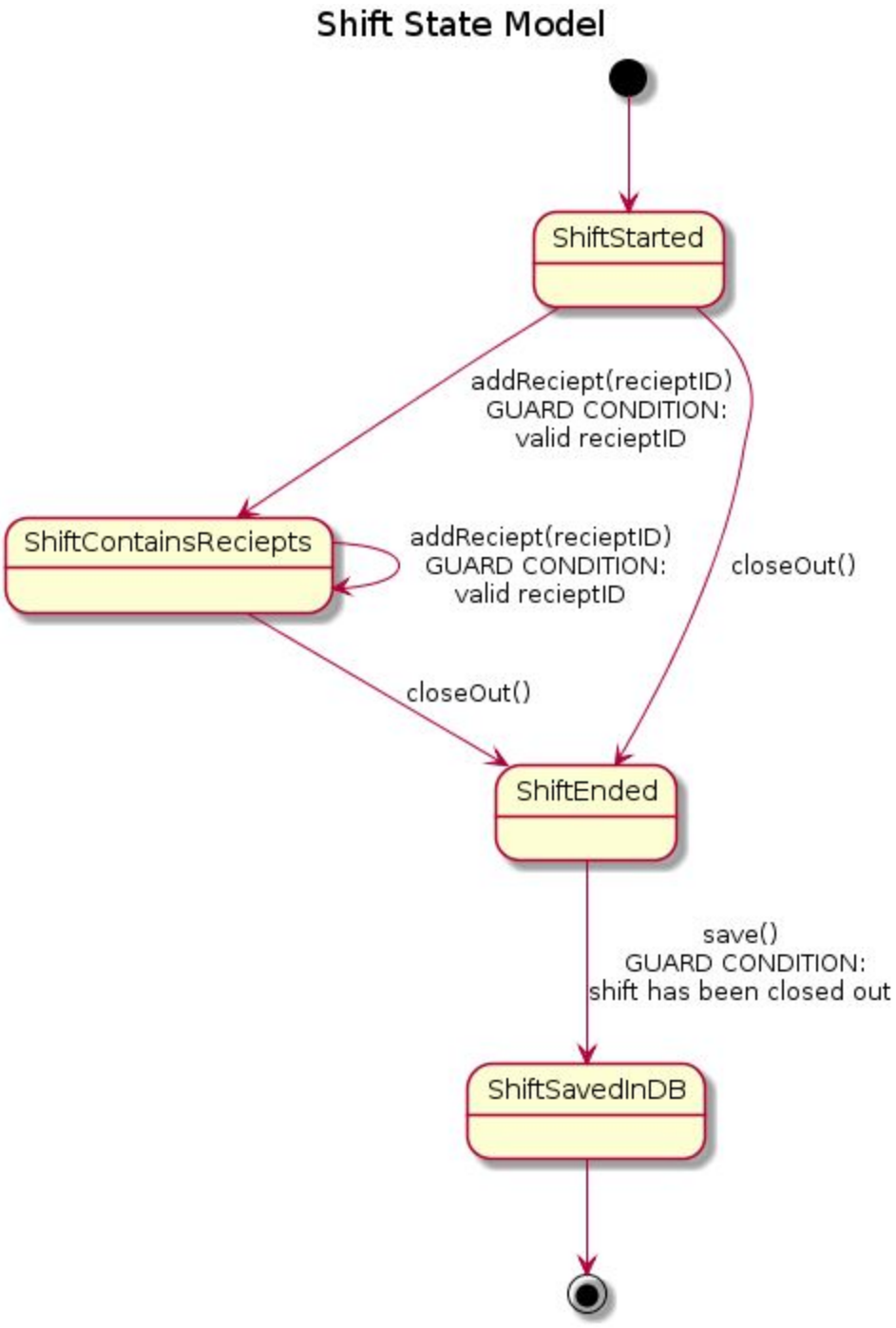


The Order class contains the following methods:

- addItems(itemID, seatNum)
    - Preconditions:
      The order has not been paid for.
      itemID points to a valid seat and seatNum < this.numSeats.
    - Postconditions:
      The item has been added to the order under the given seat number
- delItems(index)
    - Preconditions:
      The order contains at least one item and index is a valid index in this.itemList
    - Postconditions:
      The orderItem residing at index has been deleted.
- checkOut()
    - Preconditions:
      The order contains at least one item and all items have status DELIVERED.
    - Postconditions:
      The order has been paid for.
- clearOrder()
    - Preconditions:
      The order has been paid for,
    - Postconditions:
      The order is cleared, this.itemList is now empty.
- setItemStatus(index, status)
    - Preconditions:
      The order contains at least one item, and index is a valid index in this.itemList.
      Status is one of DELIVERED, READY, or PENDING

addItems(itemID, seatNum) control flow:

- Assert seatNum < this.numSeats
- Item = get Item with itemID
- If (this.itemlist[seatNum] contains Item)
    - orderItem = orderItem containing Item
    - orderItem.increment()
- Else
    - orderItem = new OrderItem(Item)
    - this.itemList[seatNumber].add(orderItem)

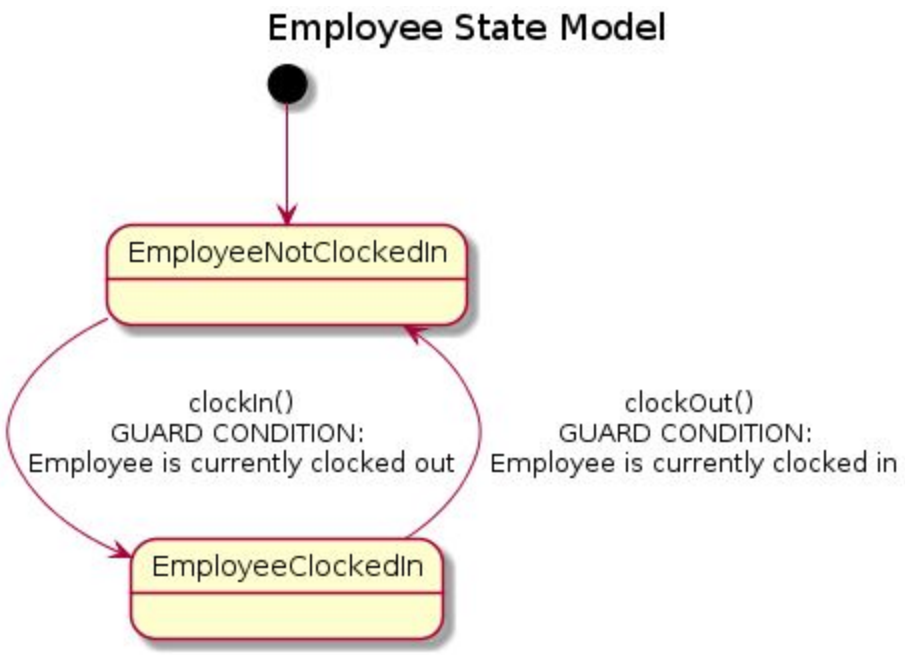

Shift State Model
ShiftStarted
addReciept(recieptID)
GUARD CONDITION:
valid recieptID
ShiftContainsReciepts
addReciept(recieptID)
GUARD CONDITION:
valid recieptID
closeOut()
closeOut()
ShiftEnded
save()
GUARD CONDITION:
shift has been closed out
ShiftSavedInDB

The Shift class has the following methods:

- addReceipt(receiptID)
    - Preconditions:
      Shift has been initialized and closeOut() has not been called yet.
      receiptID points to a valid receipt in the database.
    - Postconditions
      Receipt with receiptID is add to this.receipts.
- closeOut()
    - Preconditions:
      Shift has been initialized.
    - Postconditions:
      this.timeOut is set to current time.
- save()
    - Preconditions:
      this.closeOut() has been called (this.timeOut is set).
    - Postconditions:
      The shift is saved in the database.

addReceipt(receiptID) control flow:

- Assert receiptID points to valid receipt in database.
- Assert this.receipts does not already contain receiptID
- this.receipts.add(receiptID)

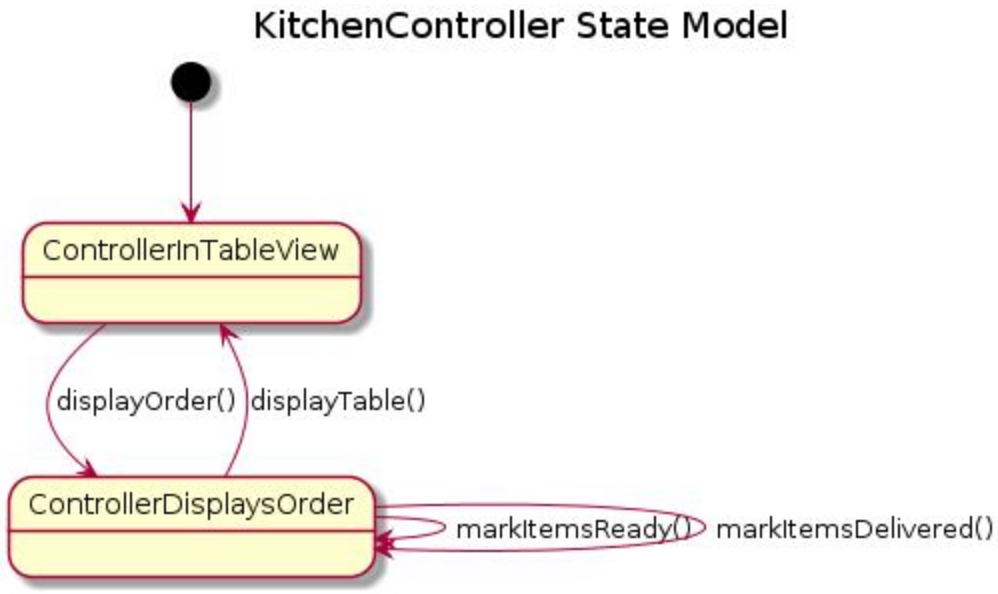


The Employee class contains the following methods:

- clockIn()
  - Preconditions:
    isClockedIn = false, employee shift attribute is null
  - Postconditions:
    isClockedIn = true, employee shift attribute is initialized
- clockOut()
  - Preconditions:
    isClockedIn = true, employee shift attribute has been initialized
  - Postconditions:
    isClockedIn = false, employees shift is finalized and saved
- editInfo()
  - Preconditions:
    This employee object has been initialized
  - Postconditions
    This employee objects details have been altered and saved

clockIn() control flow:

- Assert this.isClockedIn is false
- Set this.isClockedIn to true
- Initialize this.shift with this.shift.timeIn as current time



The KitchenController class contains the following methods:

- markItemsReady(indexList)
    - Preconditions:
      The KitchenController is in orderView and the current order contains items pending delivery. The Kitchen has finished preparing the item.
    - Postconditions
      Items have been marked as ready for delivery.
- markItemsDelivered(indexList)
    - Preconditions:
      The KitchenController is in orderView and the current order contains items ready for delivery. A server has come to deliver the item(s).
    - Postconditions:
      The items are markes as delivered.
- displayOrder()
    - Preconditions:
      The KitchenController is in orderView and this.table is a valid table
    - Postconditions:
      The order is displayed on the screen

markItemsReady(indexList) control flow:

- itemsList = this.table.order.itemsList
- For (index in indexList):
    - Assert itemsList[index].status is PENDING
    - Set itemsList[index].status to READY

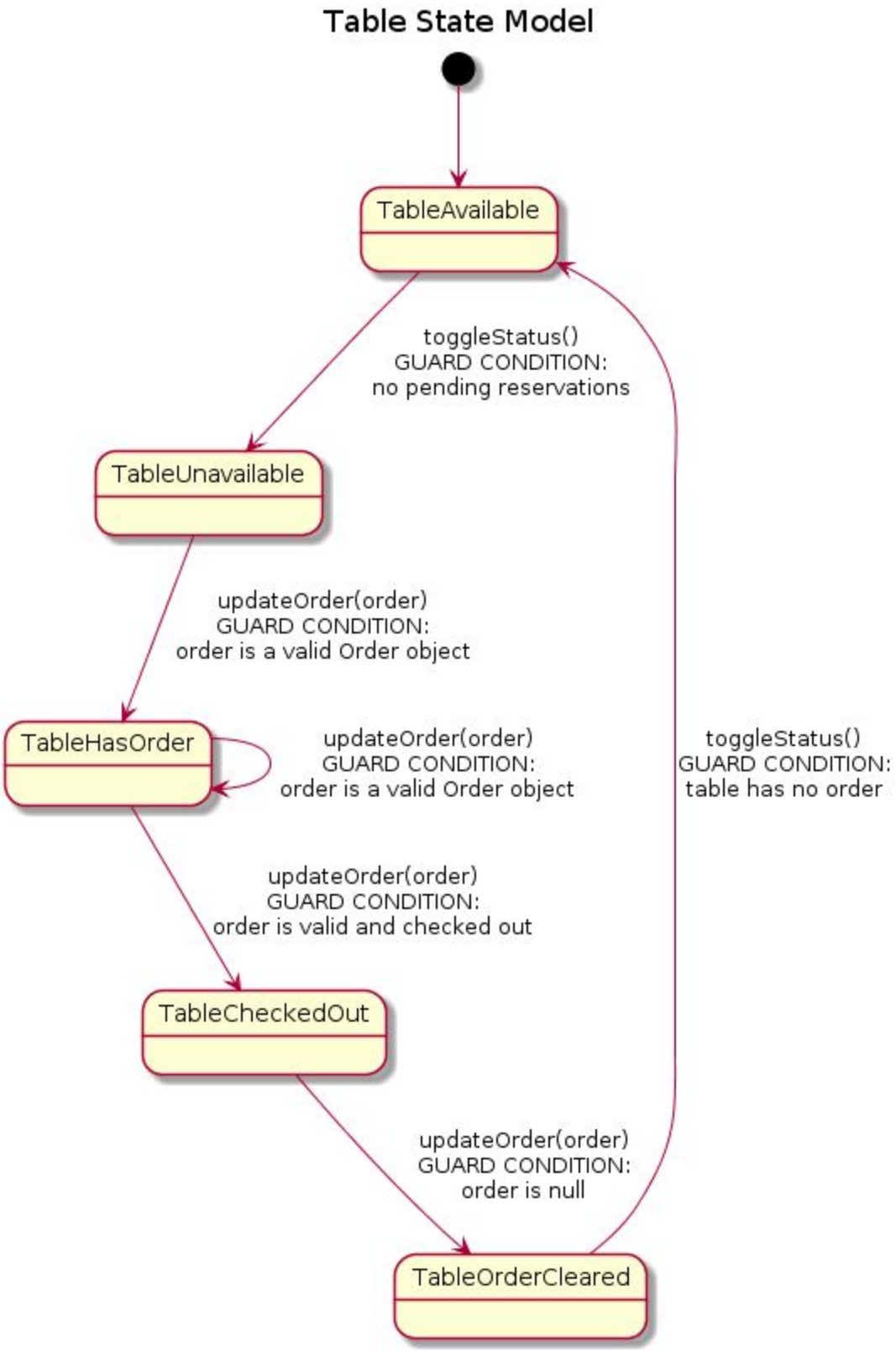


Table State Model
TableAvailable
toggleStatus()
GUARD CONDITION:
no pending reservations
TableUnavailable
updateOrder(order)
GUARD CONDITION:
order is a valid Order object
TableHasOrder
updateOrder(order)
GUARD CONDITION:
order is a valid Order object
toggleStatus()
GUARD CONDITION:
table has no order
updateOrder(order)
GUARD CONDITION:
order is valid and checked out
TableCheckedOut
updateOrder(order)
GUARD CONDITION:
order is null
TableOrderCleared

The Table class has the following methods:

- editTable()
    - Precondition: Table exists and has status AVAILABLE.
    - Postcondition: Table number of seats has been edited.
- toggleStatus()
    - Precondition: Table status is either AVAILABLE or TAKEN.
    - Postcondition: Table status has been switched.
- editOrder(order)
    - Precondition: order is a valid order with order.numSeats <= this.numSeats
    - Postcondition: this.order = order

toggleStatus() control flow:

```
If status is AVAILABLE
        Set status to TAKEN
Else if status is TAKEN
        Set status to AVAILABLE
return
```

Model Review

The models and diagrams created in the detailed design model continue to uphold the requirements for completeness, consistency, and quality by expanding upon the analysis artifacts produced while maintaining the core components the analysis artifacts described. The first artifact that upholds the criteria for completeness, consistency or transformational accuracy, and quality is the deployment diagram that demonstrates the subsystems and their interconnections. The deployment diagram is able to accurately show the architecture style we chose (three-tier client server) and break down where the use cases fall within our design architecture. This leads into the deployment diagram meeting the criteria for consistency. The diagram is consistent with the use cases established in the analysis artifacts that were created in phase I of this project. It does not add any unnecessary information nor does it add any information that cannot be found in the diagrams that it draws its information from. The deployment diagram also meets the criteria for completeness. The diagram is able to display the architectural style that was chosen and illustrates the allocation of use cases to the nodes.

The next set of artifacts are the UML Collaboration Diagrams. These diagrams are consistent with the DSSDs that they are derived from because they use the same objects or classes that are described in the DSSDs. The Collaboration Diagrams also show the different actions, information, and methods that create the interaction between the employee, interface, and controller outlined in the DSSDs. This leads into the evaluation of completeness. Most of the Collaboration Diagrams meet the criteria of completeness because they show the interactions between the employee, interface, and controllers within the system. They also add on entity objects that are lacking in the DSSDs. However, these diagrams fall short in meeting the criteria

for quality. As a group we were stretched thin by the end of this project because two group members had to drop the course. This left a six person project down to four people so the collaboration diagram quality was not fully able to be achieved. The GRASP patterns that are stated for each diagram are fully explained and detailed. Some patterns may have been over looked but overall the patterns mentioned are complete and accurate with excellent justification.

The next artifact is the Design Class Diagram. This diagram excels in upholding the criteria for completeness. The diagram successfully describes each class thoroughly and depicts how those classes are connected. It also completely describes the attributes within the classes. The Design Class Diagram is also able to achieve consistency because it takes the classes and interactions found in the collaborations and DSSD diagrams to depict a comprehensive design class diagram without adding information that it not supposed to depict or that cannot be found within the collaboration or DSSD diagrams. The quality of this diagram, however, does fall a little short. The quality is not lacking because the time or effort was not put into creating this diagram but because of the complexity of the system the diagram needed to depict. With all of the various connections among classes the connections are somewhat complicated to follow because there are so many. However, this is the best quality we could achieve that would fit within the constraints of a small picture for this project.

The final artifacts that need to be evaluated are the UML State charts of the five objects chosen from the DCD. These state charts are consistent with the state charts seen in the analysis artifacts. They are also able to achieve the criteria for completeness and quality because they are

able to depict the various states that these objects can be in with detail on the transitions between these states.

Overall the artifacts produced have greatly achieved the criteria for completeness, consistency, and quality.