Informatics 2C Software Engineering

Coursework 2

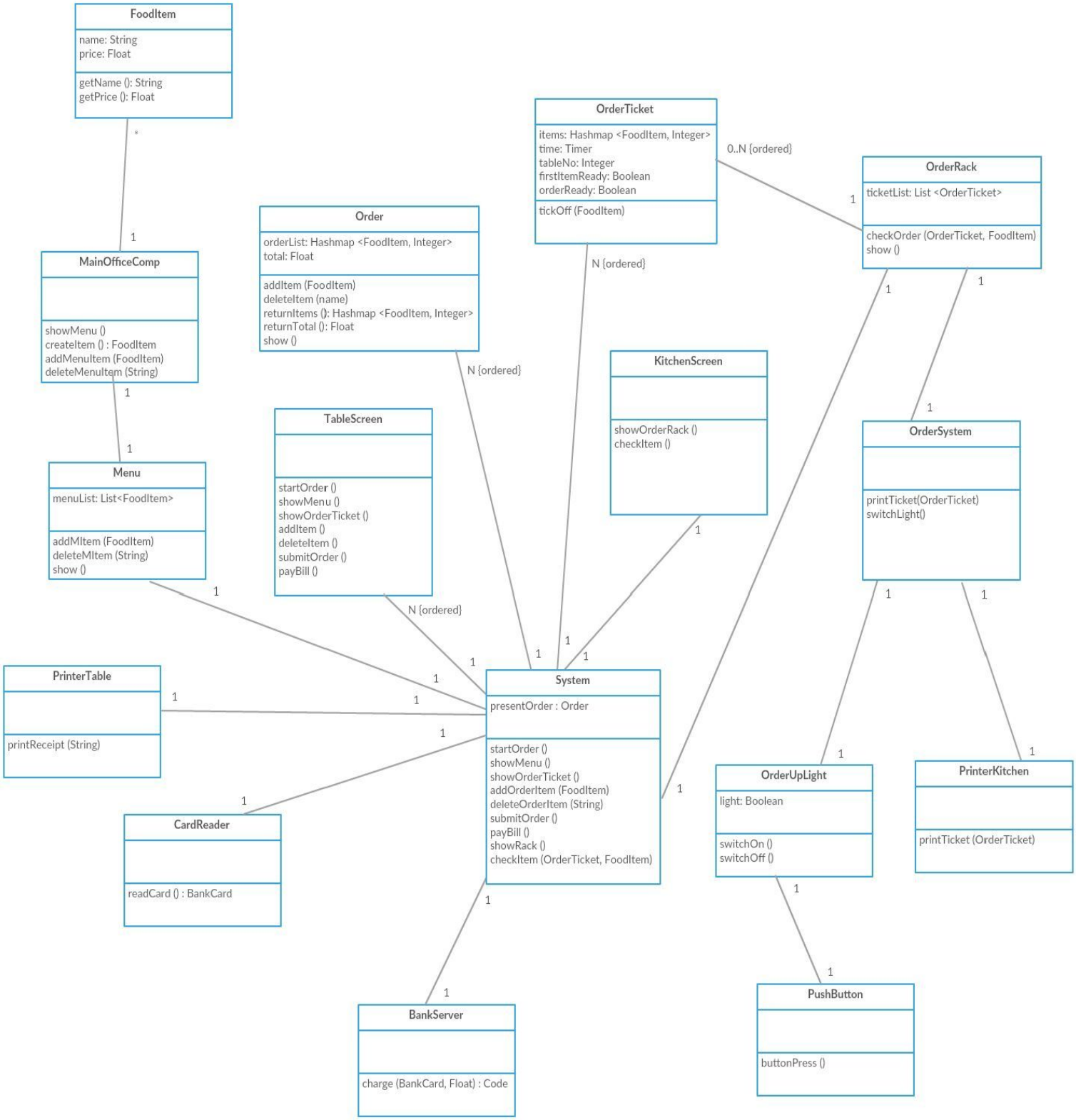
Akshay Chandiramani - s1558717
Hamza Latif - s1503416

2.1 Introduction

The system allows customers to place orders and pay for them from their tables via touch screen displays and the restaurant to deliver these orders with the kitchen staff and waiters using a large touch screen display and other input/output devices. It also allows the head chef to manage the restaurant menu. This document gives a description of the classes and objects included in the system. It mentions the functions and purpose of each class and explains the communication occurring in the system to carry out its purpose. Please refer to this document and the previous one (Course Work 1) for a complete understanding of the system.

2.2 Static Model

2.2.1 UML Class Model



2.2.2 High-Level Description

- The design that we have chosen ensures low coupling between an increased number of classes through mediator classes, high cohesion between functionalities in a class through specific class functions and an appropriate representation of the application domain and use cases through the various classes.

- The customer interacts with the system through the ***TableScreen*** class and the chef and clock interact with the system through the ***KitchenScreen*** class.

- The order rack interacts with the order ready light and the ticket printer through the ***OrderSystem*** class.

- The design uses a ***FoodItem*** class with the variables name and price to refer to the various menu items instead of using MenuID as the type of identifiers for various objects of the class Menu (which consists of food items). This way, the head chef can easily refer to an item while adding or deleting food items from the menu and the customer can access a food item similarly while building the order.

- We have implemented the order using a HashMap as it helps the system to map the unique food item in the order list with the quantity ordered by the customer. We did not consider using a List as we could not find any other way to associate the ***FoodItem*** object with the quantity the customer orders.

- The creation of separate ***Order*** and ***OrderTicket*** classes enables the system to create the final order ticket only after the customer is done with his order and presses submit.

- The *checkItem()* function in the ***KitchenScreen*** class calls the *checkItem()* function in the ***System*** class, which calls the *checkOrder()* function in the ***OrderRack*** class which in turn calls the *tickOff()* function in the ***OrderTicket*** class. These are the most complicated functions in this model encapsulated to enable abstraction of key functionalities. It is assumed that order tickets are removed from the kitchen display when the order is complete. We also assume that the chef starts with the first item of the order so that the order ticket is printed immediately after the first item is ticked off.

- We use the ***MainOfficeComp*** class through which the head chef can create new items and add them to the menu, or delete food items from the menu, by calling the functions in the Menu class. This is clearer than having the ***Menu*** class deal with the creation of individual items before adding them to the list (the Menu class only stores the list of food items). We assume that the system does not show the menu when the customer starts the order.

- The *payBill()* function in the ***TableScreen*** class calls *payBill()* in the ***System*** class, which in turn calls *readCard()* in the CardReader class and then calls charge() in the BankServer class using the bank card details returned by *readCard()* and the total amount of the present order. *charge()* returns an authorisation code of type ***Code***. *payBill()* in the ***System*** class finally calls *printReceipt()* in the ***PrinterTable*** using the present order details as a string argument. This separates out the classes for the card reader, the bank server and the receipt printer which don't have anything to do with each other (thus enabling low coupling). These classes have specific well defined functionalities (enabling high cohesion).

- We first thought of implementing the system without mediator classes, but we ended up with a system with high coupling. We then introduced the first mediator class ***System*** but because of the the flow of control in our UML sequence diagram, we chose to introduce another mediator class ***OrderSystem.***

2.2.3 Further Class Documentation

1) FoodItem

    - This class defines objects for food items that are stored as a list in our Menu class and as a HashMap in the Order and OrderTicket classes.

2) Menu

    - This class defines the menu as a list of food item objects, and enables adding and deleting items from the list using the functions addMItem() (use case 4(b)) and deleteMItem() (use case 4(c)). It also allows the current menu to be displayed by using showMenu() (use case 4(a)) which calls show() in the Menu class.

3) Order

    - This class defines an order list of type HashMap and stores the total amount of the order in a float variable.
    - addItem() - takes an argument of type FoodItem and either adds it to the order list, or increments that key's value (quantity ordered) by 1 if it is already present in the HashMap (order). Use case 1(d).
    - deleteItem() - deletes the key (item) from the HashMap (order), or decrements its value by 1 if more than one of the items are present in the order. Use case 1(e).
    - returnItems() - returns the final order.
    - returnTotal() - returns the total amount of the order.

4) OrderTicket

- This class defines the items on the order ticket and their quantities using a HashMap that is initialised using the returnItems() function of the Order class (when the user submits the order).
- The time of submission and information about the table are recorded by attributes of type Timer and Integer respectively.
- The firstItemReady and orderReady attributes are used for indicating when the first item is ready (to print the order ticket) and when the order is ready (to switch on the order ready-up light).
- tickOff() - takes an argument of type FoodItem and deletes this FoodItem key from the HashMap (indicating the item is ready). Use case 2(b).

5) OrderRack

- This class defines a list of the current order tickets ordered by submission time.
- checkOrder() - takes two arguments of type OrderTicket and FoodItem. Calls the tickOff() function of the argument of type OrderTicket passed to it, using the argument of type FoodItem passed to it. Checks if the first item is ready and calls printTicket() from the OrderSystem class to print the ticket if it is. Also checks if the order is ready and calls switchLight() from the OrderSystem class to turn the ready-up light on. Use case 2(b).
- show() - shows the current orders (list of order tickets). Use case 2(a).

6) TableScreen - *models touchscreen display at each table*

- This class is an interface class that models the customers interaction with the touchscreen display. The various functions call their counterparts with the same or slightly different names in the System class. These functions are trigger input events for use case 1.

7) KitchenScreen - *models large touchscreen display in the kitchen*

- This class is an interface class that models the clock and chefs interaction with the large touchscreen display. showOrderRack() calls showRack() in the System class, and checkItem() calls checkItem() in the System class that takes an order ticket and a food item as arguments. These functions are trigger input events for use case 2.

8) System

- This class contains an attribute of type Order which contains the current order.
- startOrder() - creates a new object of type Order. Use case 1(a).

- showMenu() - calls the show() function in the Menu class. Displays the menu to the customer. Use case 1(b).
- showOrderTicket() - calls the show() function in the Order class. Displays the current order to the customer. Use case 1(c).
- addOrderItem() - takes a FoodItem argument and calls addItem() in the Order class to add the item to the order or increase its ordered quantity by 1. Use case 1(d).
- deleteOrderItem() - takes the name of an item to delete and calls deleteItem() in the Order class to remove the item from the order or decrease its ordered quantity by 1. Use case 1(e).
- submitOrder() - creates an OrderTicket object using the final order and adds it to the order rack (ticketList). Use case 1(f).
- payBill() - calls readCard() in the CardReader class and then calls charge() in the BankServer class using the bank card details returned by readCard() and the total amount of the present order as arguments. It finally calls printReceipt() in the PrinterTable using the present order details as a String argument to print the receipt, if the customer requests for a receipt. Use case 1(g).
- showRack() - calls show() in OrderRack to display the current orders. Use case 2(a).
- checkItem() - calls checkOrder() in OrderRack using the arguments of type OrderTicket and FoodItem to check that item off a specific ticket. Use case 2(b).

9) OrderSystem

- printTicket() - calls printTicket() in PrinterKitchen to print the ticket using the argument of type OrderTicket. Use case 2(b).
- switchLight() - calls switchOn() in OrderUpLight to indicate the order is ready. Use case 2(b).

10) PrinterTable - *models receipt printer*

- This class has one function printReceipt() that takes a String argument of the details of the order and prints them. Use case 1(g).

11) CardReader - *models contactless card reader*

- This class has one function readCard() that returns some bank card details of type BankCard when the user requests the bill and taps his card. Use case 1(g).

12) BankServer

- This class has one function charge() that takes the bank card details and the total amount as arguments and returns the authorisation code of type Code. Use case 1(g).

13) OrderUpLight - *models Order ready-up light*

- This class has one attribute of type Boolean that stores the current state of the light (on is True, off is False).
- switchOn() - changes the attribute to true (on). Use case 2(b).
- switchOff() - changes the attribute to false (off). Use case 3(a).

14) PrinterKitchen - *models Ticket printer (Output)*

- This class has one function printTicket() that takes an argument of OrderTicket and prints the ticket out. Use case 2(b).

15) PushButton - *models press button to cancel light (Input)*

- This class has one function buttonPress() that calls switchOff() in the OrderUpLight class. This models the waiter cancelling the ready-up light. Trigger input event for use case 3(a).

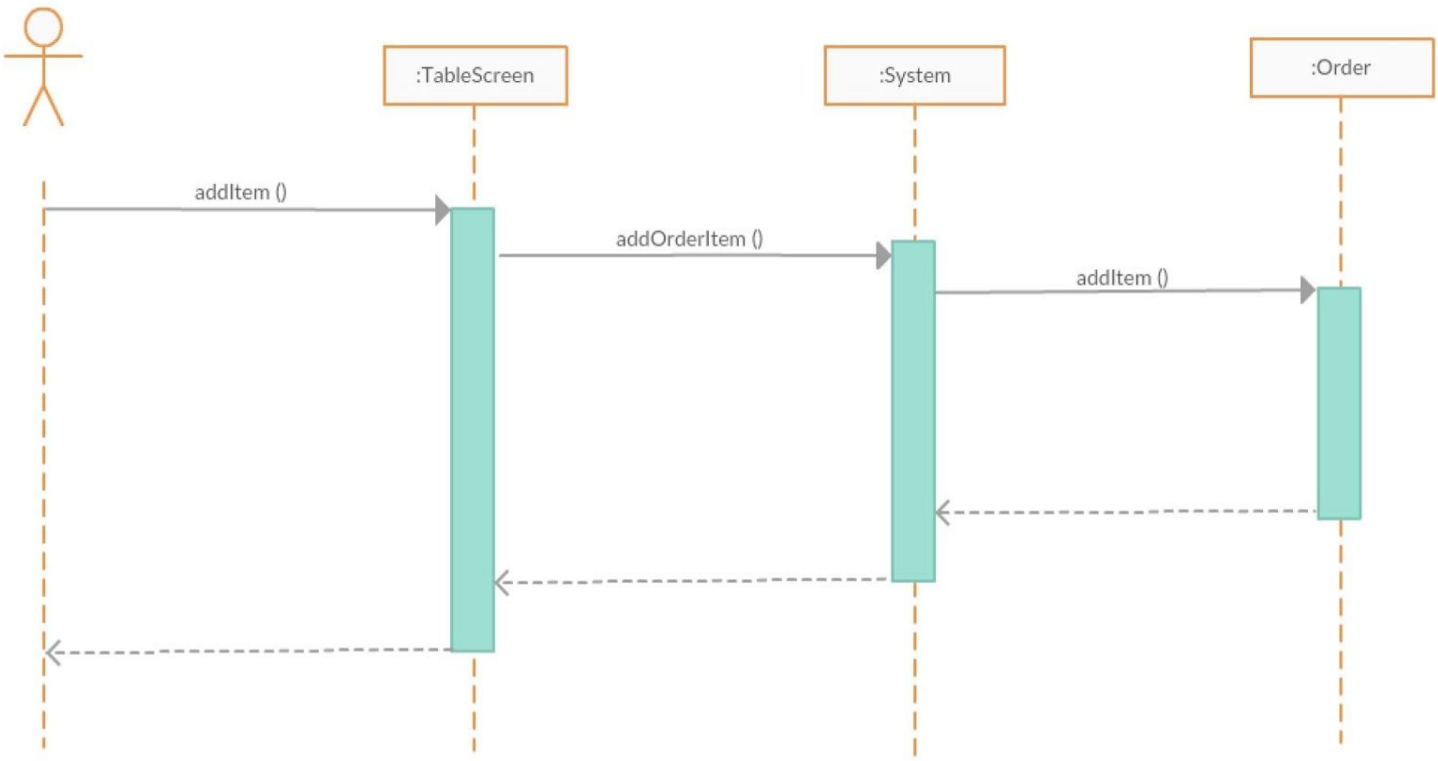
16) MainOfficeComp - *models the Office computer (Input/Output)*

- This class enables adding and deleting food items from the menu.
- createItem() - creates an object of type FoodItem and returns it.
- showMenu() - calls show() from the Menu class, which displays the menu. Trigger input event for use case 4(a).
- addMenuItem() - takes the created FoodItem object as an argument and calls addMItem in the Menu class to add an item to the menu. Trigger input event for use case 4(b).
- deleteMenuItem() - takes the name of the item to delete as an argument and calls deleteMItem in the Menu class that deletes the item from the menu. Trigger input event for use case 4(c).

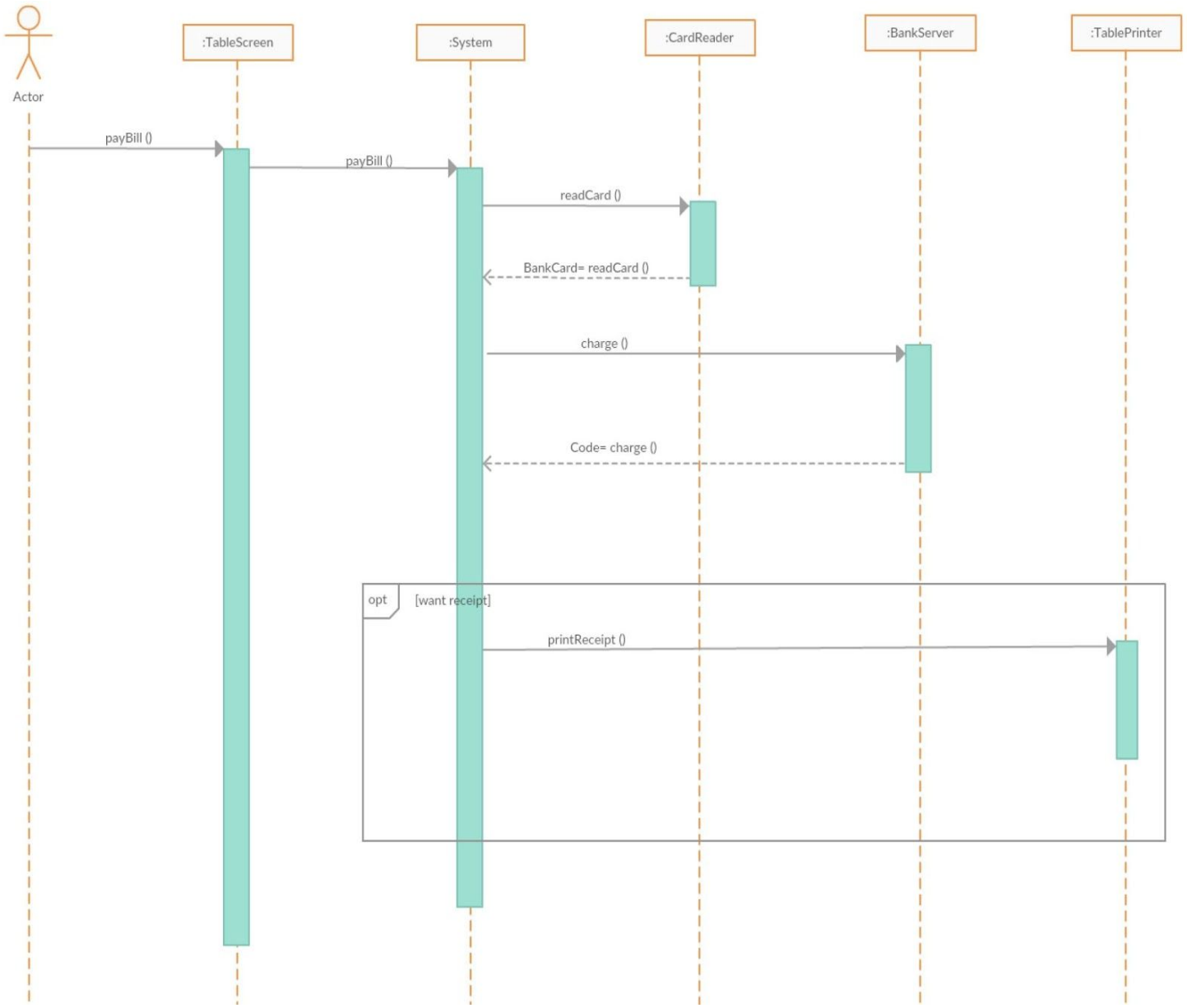
2.3 Dynamic Models

2.3.1 UML Sequence diagrams

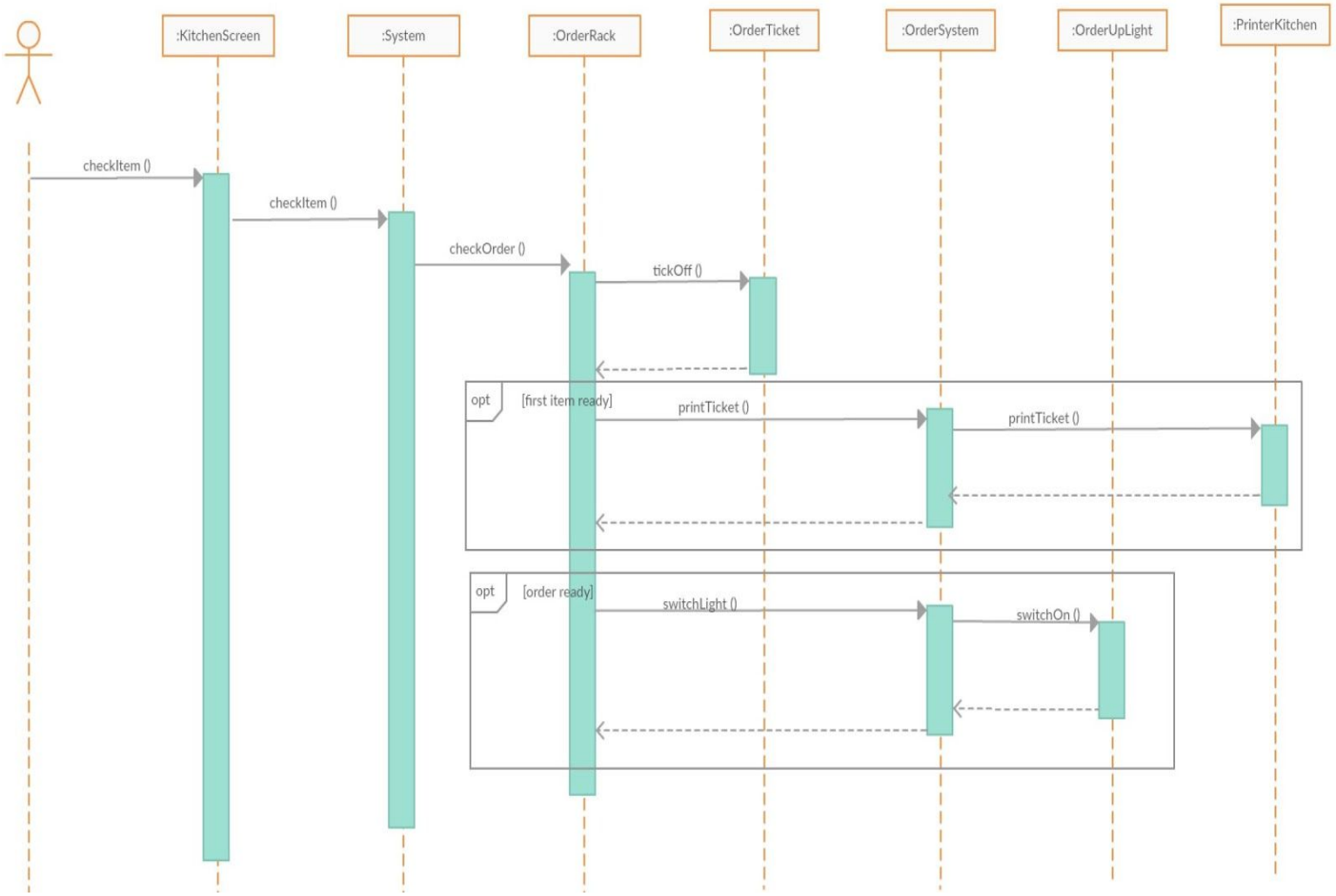
Use case 1(d)



Use case 1(g)


Actor
:TableScreen
:System
:CardReader
:BankServer
:TablePrinter
payBill ()
payBill ()
readCard ()
BankCard= readCard ()
charge ()
Code= charge ()
opt
[want receipt]
printReceipt ()

Use case 2(b)



2.3.2 Behaviour Descriptions

1. 1(b) - The input event is the customer (actor) calling showMenu() in the TableScreen class. This showMenu() calls the showMenu() of the System class, which in turn calls the show() function in the Menu class which displays the menu to the customer.

2. 1(f) - The input event is the customer (actor) calling submitOrder() in the TableScreen class. This submitOrder() calls the submitOrder() of the System class, which in turn creates an OrderTicket object using the final order and adds it to the order rack.

3. 2(a) - The input event is the chef (actor) calling showOrderRack() in the TableScreen class. This showOrderRack() calls the showRack() of the System class, which in turn calls show() in the OrderRack class that displays the current order tickets to the chef.

4. 3(a) - The input event is the waiter (actor) calling buttonPress() in the PushButton class. buttonPress() then calls switchOff() in the OrderUpLight class, which changes the boolean attribute in the class to false (turns the light off).

5. 4(c) - The input event is the head chef (actor) calling deleteMenuItem() with the name string of the item passed as the argument, in the MainOfficeComp class. This deleteMenuItem() calls deleteMItem() using the same name string, in the Menu class, which deletes the FoodItem object.