*Akshay Chandiramani*
**s1558717**

*Agnibho Chattarji*
**s1552184**

# Inf2C Software Engineering 2016-17

## Coursework 3:

## Report

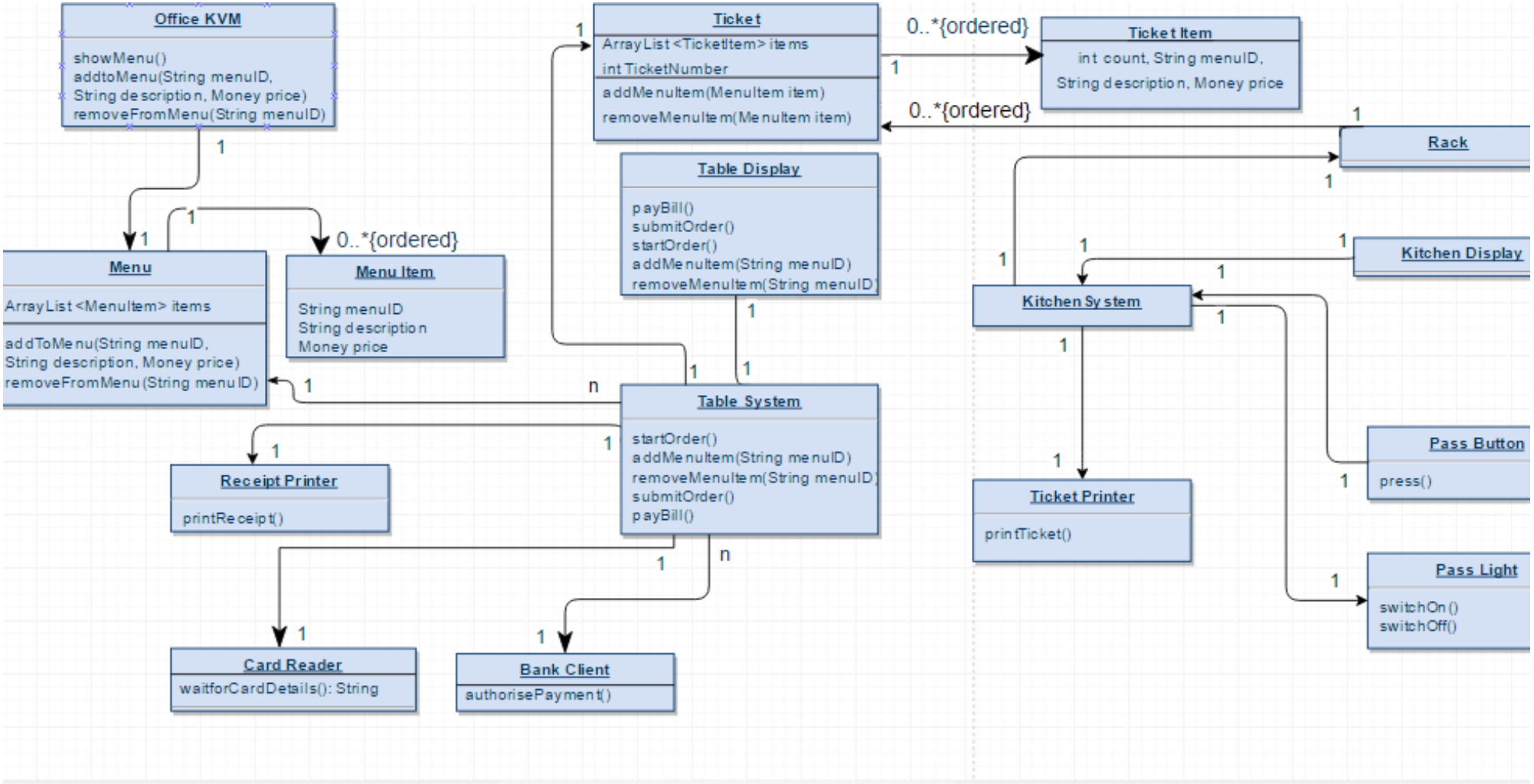
## 1. *Work Completed*

We have completed working codes for Phase I and Phase II and would like to be marked on these phases. All parts of the phases I and II have been completed. We do not want to be assessed on Phase III.

The code compiles and works completely according to our tests and understanding. The code passes the tests we have implemented in our SystemTest class and we believe it should pass all other automarking tests.

Even though the rack class has been displayed in the UML Class model we have **NOT** attempted Phase III.

## 2. UML Class Diagram



## 3. *High Level Description*

The design that we have chosen ensures low coupling between an increased number of classes through a mediator class, 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. (*as in the second coursework).*

- Every ***TableSystem*** interacts with exactly one ***TableDisplay, RecieptPrinter and CardReader.***
- All ***TableDisplays*** interact with exactly 1 ***BankClient*** and ***Menu.***
- The ***TableDisplay*** starts the order, adds and removes items from the order (ticket), submits the order and even pays for the order through the mediator class ***TableDisplay.***
- Submitting the order causes the ticket to get assigned a ticket number. The bill payment is made using the details returned by ***CardReader*** and the total amount, and the receipt is printed using the total amount and the authorisation code returned by ***BankClient.***
- The single ***OfficeKVM*** class interacts with the only ***Menu*** class and can add or remove items from the menu. We assume that the system does not show the menu when the order is stared.
- ***SystemCore*** has been replaced by a central ***KitchenSystem.***

The same reference to the ***Menu*** is passed to both the ***TableSystem*** and the ***OfficeKVM*** classes since there is only one ***Menu*** class. We first thought of implementing the system without mediator classes, but we ended up with a system with high coupling. We then introduced the mediator class **TableSystem**. We also thought of a way to implement use case events in the Kitchen using ***KitchenSystem*** mediator class.

This design differs from our original design because we have used two mediator classes instead of one core mediator class and another mediator class for just handling Orders (doesn't include the

Kitchen). Instead of using separate classes for the Order and the final OrderTicket, we have used one single Ticket class which is passed to the rack when the order is submitted.

## 4. *Implementation decisions*

The ***Menu*** has been implemented as an ArrayList of ***MenuItem*** objects, because it is easier to store the ID, description and price of the item in the menu in a separate class. It also enables us easily add or remove items from the menu by creating a new **MenuItem** object using the parameters or deleting the item with the given parameter menuID respectively.

The ***Ticket*** has also been implemented as an ArrayList of ***TicketItem*** objects similarly to enable easy addition and removal of items from the ticket. A ***ticketItem*** object also keeps track of the count of the item apart from the ID, description and price. This is used in keeping track of the number of times an item is ordered, and helps in calculating the total final amount.

All other implementation decisions have been briefly covered in the high level description.

## 5. *Tests*

| **JUnit Test** | **Use Case** |
|---|---|
| addShowMenu | 4(a), 4(b) |
| deleteNotinMenu | 4(a), 4(b), 4(c) and assert |
| deleteInMenu | 4(a), 4(b), 4(c) |

| | |
|---|---|
| addShowTicket | 1(a), 1(d) |
| removeFromTicket | 1(a), 1(c), 1(d), 1(e) |
| addTicketPayBill | 1(a), 1(c), 1(d), 1(f), 1(g) |