Software Engineering

Books or notes are **not** allowed.
Write only on these sheets. **Concise** and **readable** answers please.

Surname, name, matricola _____

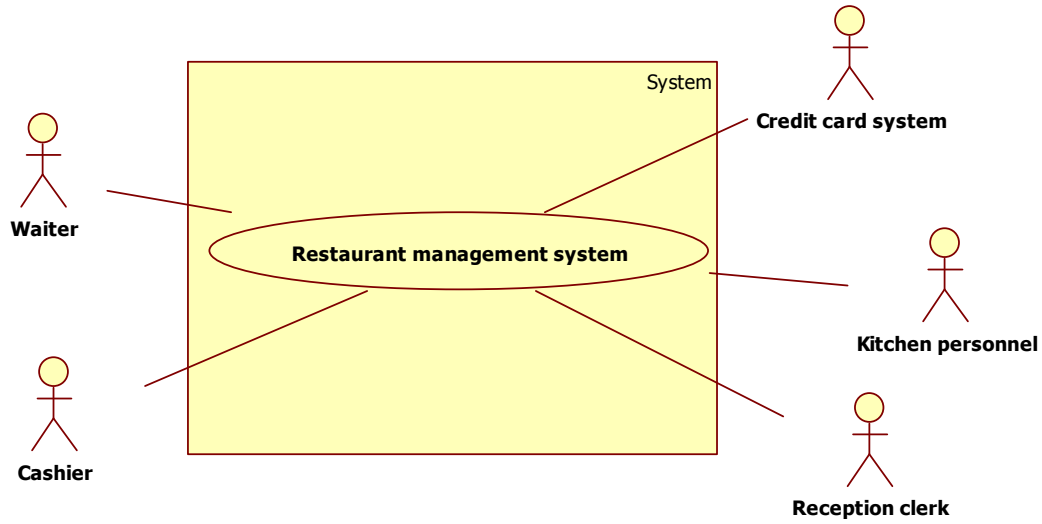
Restaurant management - wireless
In the past restaurants were managed with paper, except for payment, where often a register was used. Now more and more restaurants are managed using wireless devices. Each waiter has a wireless device (ex a modified PDA with wi-fi connection), the kitchen has one or more PCs, the acceptance desk has a PC too, all are connected.

Key functions to be considered are:

- Order from waiter to kitchen: the waiter takes orders from a table and sends them to the kitchen
- Modify order from waiter to kitchen: the waiter takes changes from a table and sends them to the kitchen
- Inquiry: the waiter asks about the status of an order
- Dish from kitchen to waiter: the waiter should be alarmed to collect a dish ready to be dispatched to the right table
- Checkout: compute the amount due by a table, manage payment, issue receipt. Payment by the customer could be made at the table (the waiter handles the credit card or cash and the receipt) or at a register.

Consider also that usually restaurants, unless very small, are divided in zones allocated to one or more waiters. Besides, large restaurants could have many exits and many registers for payment.
In the following you should analyze and model the application that supports a 'wireless' restaurant.

1 (14 points) – a. Define the context diagram (including relevant interfaces)



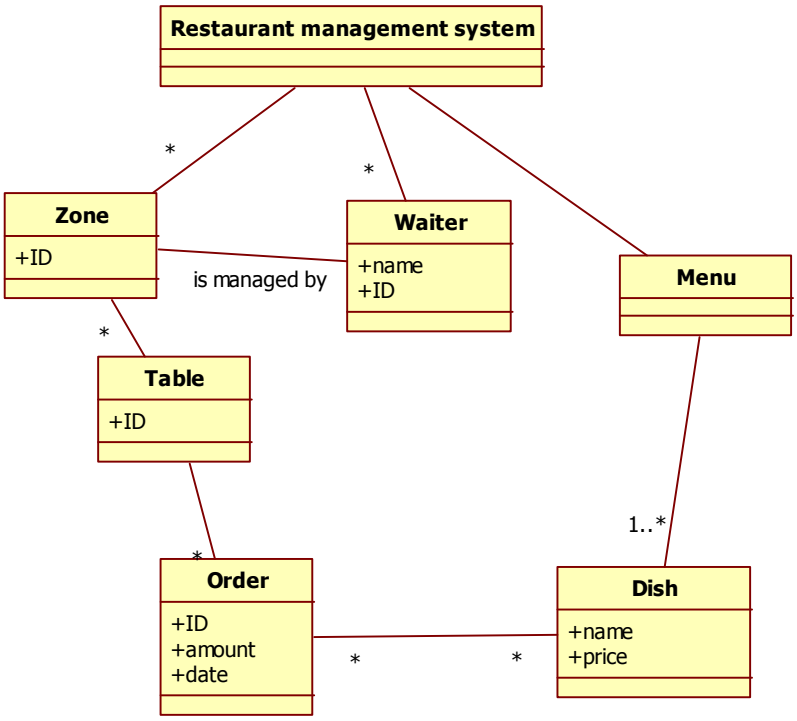
Interfaces
With Cashier, Kitchen personnel, Reception clerk: GUI on PC
With Waiter: GUI on PDA
With credit card system: internet connection and secure protocol

List the requirements in tabular form

ID	Type (Functional Non Functional)	Description
1	F	Create order, associate order to table
2	F	add dish to order
3	F	Cancel dish from order
4	F	Send order to kitchen
5	F	Communicate dish for order is ready, alert waiter
6	F	Manage payment for order
7	F	Issue receipt for payment
8	F	Show status of tables (busy, free)
9	F	Change status of table (free to busy, busy to free)
10	F	Show status of dishes of order
11	NF	The connection with the credit card system should be secure
12	NF	The restaurant wi-fi network should be protected from unauthorized access
13	NF	The waiter should be able to use the system after max 2 hours on the job training

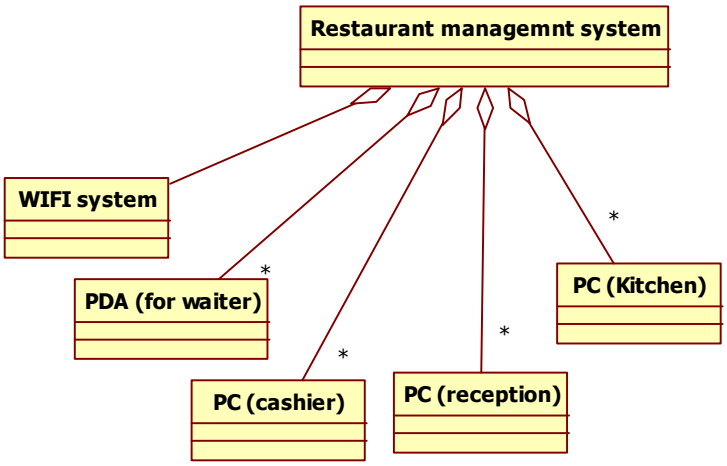
The requirements should be for the management system (software + hardware), not for the restaurant (so requirements like *The waiter takes an order* is not appropriate, the requirement should be like R1 (the system can *create an order* ..)

Define the application model (UML class diagram) for the application



This model shows the key concepts in the problem (glossary) that typically become classes in the software. The Order (associated to a table, and listing a number of dishes) is *essential* (traditionally a piece of paper with the dishes ordered and a table number).

Define the system design model (UML class diagram)



Define one scenario describing an order from table to kitchen
Precondition: no order O for table T
Postcondition: order O for table T received by kitchen

Step	Description	Req ID
1	Waiter creates order O and associates to table T	1
2	Waiter adds dish to order	2
3	Waiter adds dish to order	2
4	Waiter adds dish to order	2
5	Waiter cancels dish from order	3
6	Waiter sends order O to kitchen	4

The scenario was only about ‘order from table to kitchen’ (not also what follows afterwards). Doing it, it becomes clear, once more, the need for classes Order, Dish, and Table. The scenario must be consistent with requirements (right column). And must contain one action per line (not many).
This means that a requirement like ‘take order’ (that includes R1 and R2 and R3) is too big.

2 (8 points) -Define black box tests for the following function

The function returns the discount to be applied to customers of the restaurant in function of how much they spend and how many times they have been to the restaurant in the week.

int computeDiscount(int amount, int nTimesInWeek);

applying the following rules
if amount is > 100, discount rate = 10%
if amount is > 200, discount rate = 20%
if amount is > 100, and nTimesInWeek >1, discount rate = 15%
if amount is > 200, and nTimesInWeek >2, discount rate = 25%

ex. computeDiscount(150, 0) → 15
computeDiscount(250, 0) → 50
computeDiscount(120, 2) → 18

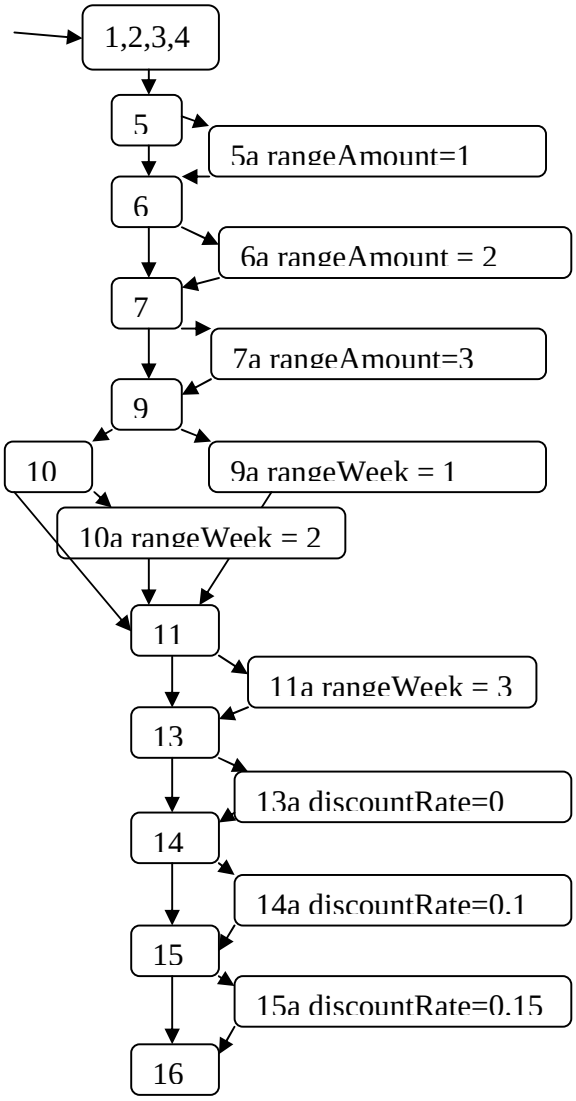
Criterion		Valid/in valid	Test case	Boundary condition
Amount	nTimesInWeek			
[minint, 0[	-	Invalid	T(-10, 1; err)	Amount = minint, 0, 1
[0, 101[	[Minint, 0[	Invalid	T(10, -10; err)	Amount = 100, 101; nTimesInWeek = minint, 0,1,2,3,maxint
	[0, 1]	Valid	T(10, 0; 0)	
	]1, 3[	Valid	T(10, 2; 0)	
	[3, Maxint]	Valid	T(10, 4; 0)	
[101 , 201[	[Minint, 0[	Invalid	T(150, -10; err)	Amount = 200, 201 nTimesInWeek = minint, 0,1,2,3,maxint
	[0, 1]	Valid	T(150, 0; 15)	
	]1, 3[	Valid	T(150, 2; 15)	
	[3, Maxint]	Valid	T(150, 4; 22.5)	
[201 , maxint]	[Minint, 0[	Invalid	T(1000, -10; err)	Amount = maxint nTimesInWeek = minint, 0,1,2,3,maxint
	[0, 1]	Valid	T(1000, 0; 200)	
	]1, 3[	Valid	T(1000, 2; 200)	
	[3, Maxint]	Valid	T(1000, 4; 250)	

3 (6 points) – For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage.
For the test cases, **write only the input value**.

```
1     double computeDiscount(float amount, int nTimesInWeek){
2           double discountRate=0;
3           int rangeAmount =0;
4           int rangeWeek =0;
5           if (amount <= 100) rangeAmount =1;
6           if (amount <= 200) rangeAmount =2;
7           if (amount > 200) rangeAmount =3;
8
9           if (nTimesInWeek <= 1) rangeWeek =1;
10          else if (nTimesInWeek <= 2) rangeWeek =2;
11          if (nTimesInWeek > 2) rangeWeek =3;
12
13          if (rangeAmount == 1 && rangeWeek == 1) discountRate =0;
14          if (rangeAmount == 2 && rangeWeek == 1) discountRate =0.1;
15          if (rangeAmount == 2 && rangeWeek == 2) discountRate =0.15;
16          return amount * discountRate;
      }
```

Coverage type	Feasibility (Y/N)	Coverage obtained (%) and test cases
Node	Y, considering the graph below there are 20 nodes to cover	3 test cases to cover 5a, 6a, 7a ex (50, -) (150,-) (250 -) 3 test cases to cover 9a 10a 11a ex (-, 1) (-, 2) (-, 3) so with these test cases T1(50,1) T2(150, 2) T3(250, 3) We cover all nodes until 11 T1 covers 13, 13a 14 15 T2 covers 13 14 15, 15a T3 covers 13 14 15 We need a test to cover 14a, T4(150,1) overall with these 4 test cases it is possible to obtain 100% node coverage
Edge	Y there are 28 edges to cover	T1 T2 T3 T4 cover all edges
Multiple condition (line 13)	Y, 4 test cases	T1 covers T,T T2 covers F,F T3 covers F,F T4 covers F T We need to cover TF T5(50,1)
Loop	No loops	
Path	Partially	There are 8 (ifs until line 8)x3(if line 9 to 10) x16(ifs line 11 to 15) 384 = paths. So in a finite time it is conceivable to write 384 test cases. However many of these paths are unfeasible because of dependencies between conditions in ifs.

Flow graph:



4 (1 points) – Items A and B are under configuration in subversion. User U1 performs a commit on A with revision 121. Next user U2 performs a commit on B. What are the revision numbers?

Revision number for A: __122__ Revision number for B:_____122___
The commit on B changes revision number to ALL items in the repository

5 (1 points) – Describe how to implement a baseline in Subversion
Using tags

6 (1 points) – In the context of software design, mention an example of tradeoff between non functional properties of a system.
Efficiency (speed) tipically requires a tradeoff with security.

7 (1 points) – What is the difference between fault and failure?
A failure is external evidence for the user of a malfunction. A fault is the internal cause of the failure. Not all faults cause failures.