*In general, try to put the supertypes above the subtype when possible. It makes the model far easier to read.*

*I think your definitions need to be read aloud to someone not on the team. Doing that test will help you see where you need to rephrase/rethink some of your classes.*

**Shak**

* Your class MenuCategories can go away, and MeatEntrees will then relate directly to ManuItems. The way that you treat this means that each meat entree will appear in the menu 6 times: one for each meat type. Another way to handle that matter is to put the meat on the association class that you still need between MenuItems and Order.
* Think of it this way, the Order is just like an Order in the Customer Orders database that we used for our first example of a many to many without history. The Order represents a grouping of menuItems that get ordered together. The MenuItems are independent of the Order, they don't depend upon an order. Given the multiplicity on your association from Order to MenuItems, Order would be the parent, which tells me that a given MenuItem can never show up on more than one order.
* And let's talk about that many to many between Order and MenuItems some more. It really should be a many to many between Order and MenuPrices. The reason that I say that is that MenuItems doesn't know what menu it's on, so there's no way to price it, which means that there's no way for you to calculate the total for your orders.
* You might want to make your menuUsedDays even more flexible than it is. You could have an enumeration for the days of the week, and make a many to many between days of the week and Menus. Just a thought.
* The spice level of a given entree is decided on at the time that it's ordered, so you need to put that into the Order Items association class (that's missing from your model).

**Sooyoung**

* ~~Done~~
  + ~~Order does not need an OrderType. Remember the Categories lab? We were able to deduce which category an instance of the generic supertype fell into by looking at the subtype tables to see which one of them had a row with the same primary key.~~
* Done?
  + Associating your EatIn orders with the tables is a great thing. But the CustNumbers[1..\*] isn't quite what you want, for a couple of reasons. For one, those customer numbers really should come from the Customer class via a migrating foreign key. But there's another problem here because not all customers will ever get a row in the Customer table because they elect to remain anonymous. Instead, you really need to concern yourself with individual seats around those tables, so that you can keep track of what food a given seat ordered (notice the wait staff really doesn't care who is in that seat, just that the right food has to show up in front of the right person). Another problem that I see with CustNumbers is that the Table is totally independent of any particular party that sits there. Over time, there will be lots and lots of customers (anonymous or not) sitting at that table, so you don't want to couple your table that tightly to the customers.
* Confused
  + What I would suggest that you do is to relate that missing association class between MenuPrices and order, let's call it Order Item, to one of the seats at the table for a given EatIn order. I'll probably need to walk through this with you in a meeting, let me know when your team can get together tomorrow or Friday.
    - I tried lol
  + ~~Your association there between Takeout, Online, and Phone is not part of the UML class model lexicon. I have no idea what that means. It might just be a typo and that you meant for it to be a categorization. I hope so.~~
    - Done

**Jon**

* Good job doing a many to many between Line Cook and Station. Now, realizing that a line cook is assigned to one or more stations during a shift, how would you capture that? Even better, how would you capture that and make sure that it's only line cooks who get assigned to stations?
* ~~Once again, that mysterious graphic construct shows up between chef, head chef, and line Cook and sous Chef.~~
* ~~Your sousChef needs little work. Everything you have there is good, you just need to tweak it a little. A given SousChef has mastered many menu items, and a given menu item has been mastered by many sous chefs. That screams many to many without history between SousChef and MenuItems. Your UML doesn't specifically say that there is an association between sous chef and the Menu Items class, just between Sous Chef and a separate table that has a list of all of the menu items in it, totally unrelated to your Menu Items class that you've defined elsewhere in the model. Then, that association class that's between sous Chef and Menu item has an employee number and the Menu Item Number, and you can use that to capture the mentor and the Menu Item that the mentor is teaching about. Then all you need is to bring the other Sous Chef into Mentorship and you're done.~~
* ~~That construct that you have going into roles is another invention of yours. I have no idea what it means. A given employee is a Maitre d' regardless of the shift that they work, or the time. That's their job and they stick to it. So I'm not sure~~
* ~~There should be some sort of association between recipes and Menu Items. Otherwise, there's really no point in the head chef creating the recipes if they are never going to get prepared.~~
* ~~The manager Name in Shift is totally unregulated. A much better way to do that is to run an association from empShift in to Shift and role name the Employee ID that comes from Employees to empShift and then into Shift as Manager ID. Then run a second association from Manager into Shift and role name the Employee ID to Manager ID in that relationship as well, which then forces you to only designate an Employee who is a manager and who works that shift as the manager for that shift.~~
* ~~Same thing for the head cook for the shift.~~

**Omar**

* Your Payer attribute in Bill should be the result of a migrating foreign key from Customer, if the Bill is paid by a Customer. You need to categorize Bill into those that are paid anonymously, and those paid by a Customer of Record.
* I don't see how you track the use of Miming's Money.
* I think your definitions need to be read aloud to someone not on the team. Doing that test will help you see where you need to rephrase/rethink some of your classes. Take OrderInstance for instance. What causes a row to get added to that table? It is in a 1..1 to 1..1 relationship with Order, so it really should be merged with Order. It has an OrderNumber, which migrates in from Order??? If so, we would not show that in the UML model because it's a migrating foreign key.

**Jon**

* Association descriptions are just like what we did when we first started talking about one to many associations. An example would be "Customer places Orders". Very simple, very brief, very much tied to the verb phrases (which you still need) and the class names in the model.
* Please alphabetize your definitions by the class name. That makes it loads easier for me to find things.

**Omar & Shak**

* Your denormalization isn't really a denormalization. I want to see you create a subkey of some sort, and then figure out how you're going to make sure that you don't get contradictory data in your tables because you've relaxed your normalization.