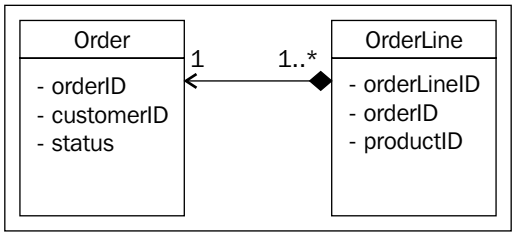
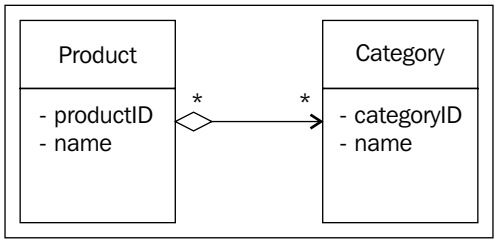
The relationship between Order and OrderLine objects is the same. An order can have multiple products; each product will be shown in a separate line (called as OrderLine) in the Order. So there can be one or more order lines for a single order, as shown here:



The above diagram confirms that for each order, there will be one or more order lines. We can't use 0..* here in place of 1..* because each order will have atleast one product in it (as one order line item).

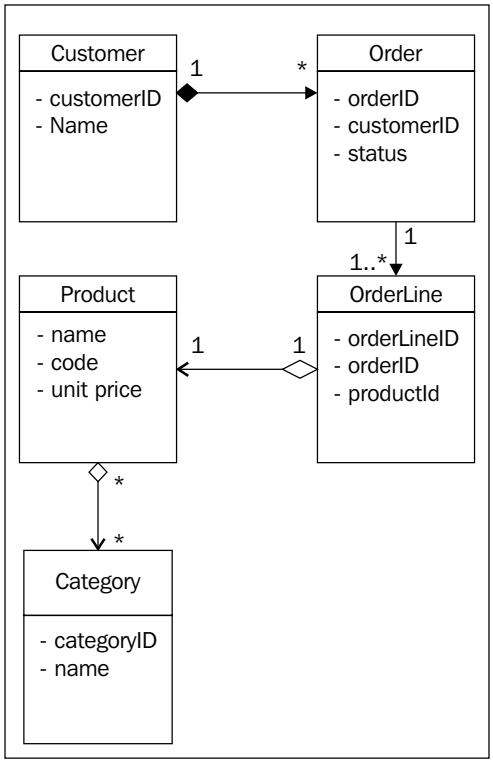
Also, if an order gets cancelled (destroyed), then all order lines will be destroyed. It doesn't make sense to have order lines that are not a part of any order—hence the composition.

- **Many-to-many**: A Product can belong to multiple Categories, and a Category object can include multiple Product objects. To depict such many-to-many relationships, we use asterisk at both ends of the relationship arrow, as shown here:



Also note the aggregation relationship between the Product and the Category, because both can exist independently of each other.

So, now, we can combine all of the above diagrams and create a simple class diagram with all of the relationships and multiplicities for our OMS. Here is the combined UML class diagram for our sample application:



So we have a very simple domain model of a simple Order Management System. Now, based on the above classes, let's look at how we can convert this domain model to code by creating a 1-tier 3-layer architecture based web application.

1-tier 3-layer Architecture using a Domain Model

Based on the above class diagram, we will create a new simple 3-layered application using the entities defined in the above domain model. We will create a new ASP.NET Web Project in VS. This time, you should create two new folders inside your root web folder (using the **Add New Folder** option in VS):

- **BL**: This folder will contain all of the business logic domain classes
- **DAL**: This folder will contain the data access code files (for each entity)