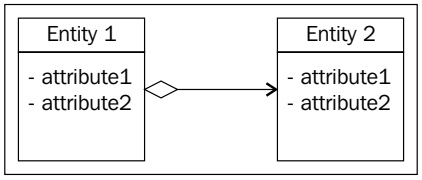
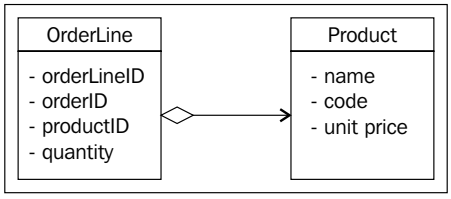


So, in our example, Entity2 is a part of (or subordinate to) Entity 1. If you destroy the parent class (Entity 1) in an aggregation (weak) relationship, the child class (Entity 2) can survive on its own.

Let's understand aggregations by using our example of the Order Management System. Consider the OrderLine and Product classes. An OrderLine can have multiple quantities of one Product. If an OrderLine is destroyed, it does not mean that we delete the Product as well. A Product can exist independently of the OrderLine object. Here is the relationship diagram between OrderLine and Product classes:

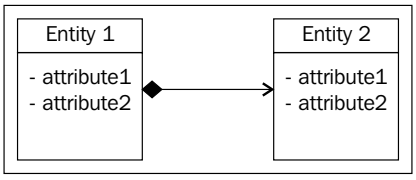


In the diagram, we can see an Aggregation relationship between OrderLine and Product classes. Put simply, the above diagram states that if an order is cancelled, all of the products will not be destroyed; they will only be "de-associated" from that particular order.

Composition

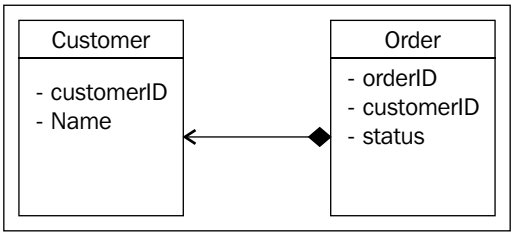
A **Composition** is exactly like Aggregation except that the lifetime of the 'part' is controlled by the 'whole'. For example: You have a 'student' who has a 'schedule'. If you destroy the student, the schedule will cease to exist.

In this case, the associated entity is destroyed when the parent entity goes out of scope. Composition is represented by a straight arrow with a solid diamond at the tail, as shown below.



In our case, Entity-2 is controlled by Entity-1. If Entity 1 is destroyed in a composition (strong) relationship, Entity-2 is destroyed as well.

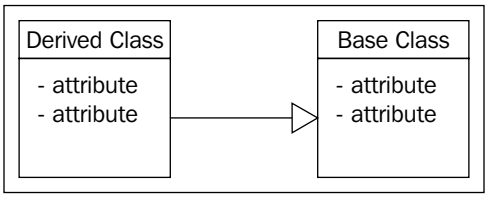
Let's understand compositions by using our example of the Order Management System. Consider the Customer and Order classes. A Customer can have one or more orders, and an Order can have one or more Products (in order lines). An Order object cannot exist on its own without a Customer. So the following Composition indicates that if a Customer object goes out of scope, the Orders associated with that Customer go out of scope too.



Generalization Relationship

Inheritance is a very widely known and common feature of OOP. In UML, inheritance is depicted using generalization relationships, depicted by a straight arrow with a hollow arrowhead (triangle) at one end. A generalization relationship (also known as a "is-a" relationship) implies that a specialized (child) class is based on a general (parent) class.

Here is a diagram illustrating this:



Here, we can see that the arrow points in the direction of the base class. In our Order Management System, we can have a base class for all customers; we can call it Person class so that we have other classes derived from it, such as Customer, CSR (Customer Sales Representative), and so on.