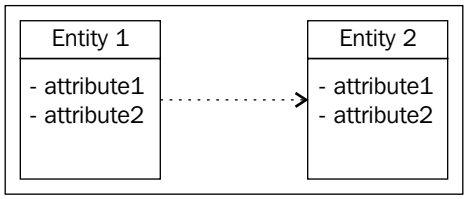
UML Relationships

In an ER diagram for two entities A and B, we can show only one type of relationship–a Relational relationship–which means that entity A is somehow related to entity B. But in a class diagram, the relationships can be further divided on the basis of object-oriented principles such as inheritance, association, and so on. The following sections describe the main UML relationships used in a class diagram.

Dependency Relationship

A **Dependency** exists between two elements if changes to one element will affect the other. A dependency relationship is the simplest relationship of all, and means that Entity 1 depends on Entity 2 in such a way that any change in entity 2 might break entity 1. This is a one-way relationship only—changes in entity 1 will not affect entity 2 in any manner. Dependency relationships are represented by a broken (dashed) line with an "empty" arrow (--->). The direction of this arrow flows to the entity that is dependent on the entity that the arrow flows from.



A simple example would be:

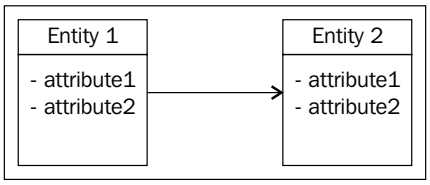
```
public class Entity1
{
    public void MethodX (Entity2 en)
    {
        // . . .
    }
    public void MethodY ()
    {
        Entity2 en2 = new Entity2();
        // . . .
    }
}
```

In the above pseudo code, Entity2 is used in Entity1 first as a method parameter and secondly inside `MethodY()` as a local variable. Both of these cases show a dependency relationship.

An important point about dependency relationships relates to the state of the objects involved. The state of Entity2 is not related to the state of Entity-1, as Entity2 is not a member of the Entity1 class. It is only used in the methods as local variable.

Association Relationship

An **Association** shows the relationship between instances of classes. An association is similar to a dependency except that it specifies the relationship in a stricter form. An association means that instead of Entity2 being used in Entity1 as a local variable, it would be a global variable instead, which can be a private, public, or protected class member. In its basic form, it is represented by a solid arrow (instead of dashed as in a dependency relationship).




```
public class Order  //entity1
{
  private Customer _customer;
  public Customer GetCustomer ()
  {
    return _customer;
  }
}
```

In the above pseudo code, the Customer object is a part of the Order class—a private member in this case. This means that the Customer object forms a part of the state of the Order. If you have an Order object, you can easily identify the Customer associated with it. This is not possible in a dependency. Hence, the association is a stronger form of dependency.

An Association relationship can be divided further into two separate relationships, based on the state of the aggregated object in the dependent class.

Aggregation

An **Aggregation** relationship depicts a classifier as a part of, or as subordinate to, another classifier. For example, if Entity1 goes out of scope, it does not mean that Entity2 has to go out of scope too. That is, the lifetime of Entity2 is not necessarily controlled by Entity1. An aggregation is represented by a straight arrow, with an empty diamond at the tail, as shown in the following figure: