An instance of an association is called a link, as discussed in Chapter 16.

**Multiplicty** An association represents a structural relationship among objects. In many modeling situations, it’s important for you to state how many objects may be connected across an instance of an association. This “how many” is called the multiplicity of an association’s role. It represents a range of integers specifying the possible size of the set of related objects. It is written as an expression with a minimum and maximum value, which may be the same; two dots are used to separate the minimum and maximum values. When you state a multiplicity at the far end of an association, you are specifying that, for each object of the class at the near end, how many objects at the near end may exist. The number of objects must be in the given range. You can show a multiplicity of exactly one (1), zero or one (0 .. 1), many (0 .. *), or one or more (1 .. *). You can give an integer range (such as 2 .. 5). You can even state an exact number (for example, 3, which is equivalent to 3 .. 3).

For example, in Figure 5-6, each company object has as employee one or more person objects (multiplicity 1 .. *); each person object has as employer zero or more company objects (multiplicity *, which is equivalent to 0 .. *).

**Aggregation** A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other. Sometimes you will want to model a “whole/part” relationship, in which one class represents a larger thing (the “whole”), which consists of smaller things (the “parts”). This kind of relationship is called aggregation, which represents a “has-a” relationship, meaning that an object of the whole has objects of the part. Aggregation is really just a special kind of association and is specified by adorning a plain association with an unfilled diamond at the whole end, as shown in Figure 5-7.
Advanced relationship concepts are discussed in Chapter 10.

Class diagrams are discussed in Chapter 8.

Links are discussed in Chapter 16.

Note: The meaning of this simple form of aggregation is entirely conceptual. The open diamond distinguishes the "whole" from the "part," no more, no less. This means that simple aggregation does not change the meaning of navigation across the association between the whole and its parts, nor does it link the lifetimes of the whole and its parts. See the section on composition in Chapter 10 for a tighter form of aggregation.

Other Features

Plain, unadorned dependencies, generalizations, and associations with names, multiplicities, and roles are the most common features you'll need when creating abstractions. In fact, for most of the models you build, the basic form of these three relationships will be all you need to convey the most important semantics of your relationships. Sometimes, however, you'll need to visualize or specify other features, such as composite aggregation, navigation, discriminants, association classes, and special kinds of dependencies and generalizations. These and many other features can be expressed in the UML, but they are treated as advanced concepts.

Dependencies, generalizations, and associations are all static things defined at the level of classes. In the UML, these relationships are usually visualized in class diagrams.

When you start modeling at the object level, and especially when you start working with dynamic collaborations of these objects, you'll encounter links, which are instances of associations representing connections among objects across which messages may be sent.
Attributes are discussed in Chapters 4 and 9.

Qualification In the context of an association, one of the most common modeling idioms you’ll encounter is the problem of lookup. Given an object at one end of an association, how do you identify an object or set of objects at the other end? For example, consider the problem of modeling a work desk at a manufacturing site at which returned items are processed to be fixed. As Figure 10-5 shows, you’d model an association between two classes, WorkDesk and ReturnedItem. In the context of the WorkDesk, you’d have a jobId that would identify a particular ReturnedItem. In that sense, jobId is an attribute of the association. It’s not a feature of ReturnedItem because items really have no knowledge of things like repairs or jobs. Then, given an object of WorkDesk and given a particular value for jobId, you can navigate to zero or one objects of ReturnedItem. In the UML, you’d model this idiom using a qualifier, which is an association attribute whose values identify a subset of objects (usually a single object) related to an object across an association. You render a qualifier as a small rectangle attached to the end of an association, placing the attributes in the rectangle, as the figure shows. The source object, together with the values of the qualifier’s attributes, yield a target object (if the target multiplicity is at most one) or a set of objects (if the target multiplicity is many).

Composition Aggregation turns out to be a simple concept with some fairly deep semantics. Simple aggregation is entirely conceptual and does nothing more than distinguish a “whole” from a “part.” Simple aggregation does not change the meaning of navigation across the association between the whole and its parts, nor does it link the lifetimes of the whole and its parts.
An attribute is essentially a shorthand for composition: attributes are discussed in Chapters 4 and 9.

However, there is a variation of simple aggregation—composition—that does add some important semantics. Composition is a form of aggregation, with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it. Such parts can also be explicitly removed before the death of the composite.

This means that, in a composite aggregation, an object may be a part of only one composite at a time. For example, in a windowing system, a Frame belongs to exactly one Window. This is in contrast to simple aggregation, in which a part may be shared by several wholes. For example, in the model of a house, a Wall may be a part of one or more Room objects.

In addition, in a composite aggregation the whole is responsible for the disposition of its parts, which means that the composite must manage the creation and destruction of its parts. For example, when you create a Frame in a windowing system, you must attach it to an enclosing Window. Similarly, when you destroy the Window, the Window object must in turn destroy its Frame parts.

As Figure 10-6 shows, composition is really just a special kind of association and is specified by adorning a plain association with a filled diamond at the whole end.

![Figure 10-6: Composition](image)

Note: Alternately, you can show composition by using a structured class and nesting the symbols of the parts within the symbol of the composite. This form is most useful when you want to emphasize the relationships among the parts that apply only in the context of the whole.

Association Classes In an association between two classes, the association itself might have properties. For example, in an employer/employee relationship between a Company and a Person, there is a Job that represents the properties of that relationship that apply to exactly one pairing of the Person and Company. It wouldn’t be appropriate to model this situation with a Company to Job association together with a Job to Person association.