Class Diagram

Relationships between Classes:

In a class diagram, obviously you can't have classes just floating around; you need to see the relationship between them. There are following types of relationships amongst classes.

- Association
- Aggregation
- Generalization

Association Relationship

An Association is a connection between classes, showing how one class relates to another. The association is shown as a line drawn between the classes. An Association Name can be given to the association to clarify the association between two classes. The name is written on the association line, with an arrow indicating the direction that the association name is to be read, but the association is navigable in both directions. The following example shows the relationship between a CD Player and a CD. The Track information is deliberately omitted as this is a special relationship.

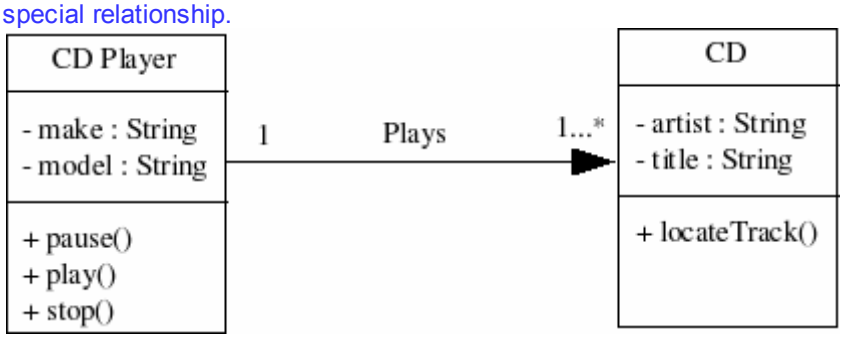

Figure 5.7 An association relationship between two classes

In figure 5.7 shows the relationship is between "CD Player" and "CD". The name of the relationship is, "Plays", and there's an arrow on the association from "CD Player" to "CD" indicating unidirectional association.
Association represent conceptual relationships between the types involved. Associations may be bi-directional (can be navigated in either direction) or unidirectional (can be navigated in one direction only).
Bidirectional association is depicted as solid line without arrowhead while unidirectional association is shown as solid line with filled triangular arrowhead. Conceptually all associations can be thought of as bi-directional, but unidirectional associations are important for specification and implementation models. For specification models bi-directional associations give more flexibility in navigation but incur greater coupling. In implementation models a bi-directional association implies coupled sets of pointers, which many designers find difficult to deal with.

Association Adornments

The following adornments may be added to the diagram to add extra information about the relationships.
Number of Instances in a Relationship: One of the key aspects of association is the cardinality (multiplicity). Multiplicity is used to show how many instances of one class may be associated with another class. The multiplicity ranges in are as shown in table 5.1.

Table 5.1: Multiplicity

Multiplicity	Adornment
Optional	0 1
Zero or more	0 ..... * or *
At least one	1 .... *
Specified	A number, or range of numbers. (e. g., 2, or 1, 4, 9..12)
Exactly one	1 or left blank

Role Names : An association can have a role name to add extra information about the role of the class in the association, providing a context of the class and its objects. The role name is placed near the end of the

association next to the class to which it applies.

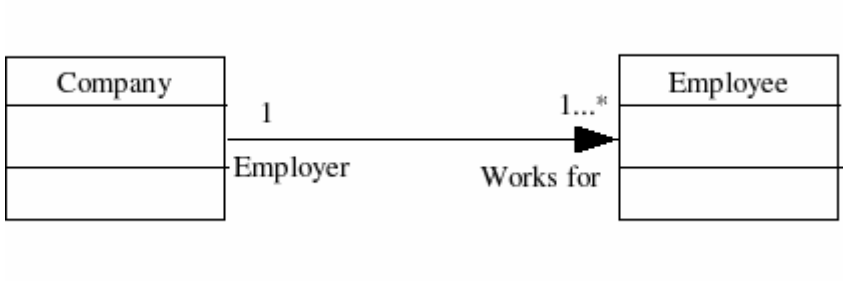

Figure 5.8 An association relationship between two classes with role names

Recursive or Reflexive Associations : A recursive association is where a class is associated to itself. This is the type of association used with Linked Lists. When a class is associated with itself, this does not mean that a class's instance is related to itself, but that instance of the class is related to another instance of the class.

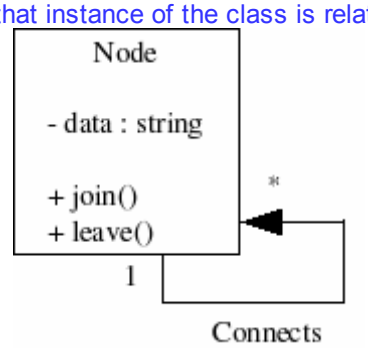

Figure 5.9 Recursive relationship

The above relationship specifies that a node is related to zero or more nodes. The class is called node, and has an association with itself.

Or-Associations : Sometimes, it is not possible to have instances of all classes in a relationship. For example, a decoder may receive either Cable or Satellite Transmissions, but not both. has an Or association that specifies that instances of a class may only participate in one of the associations at a time. The Or-Association is depicted by a dashed line between the associations with { or } written above the dashed line.

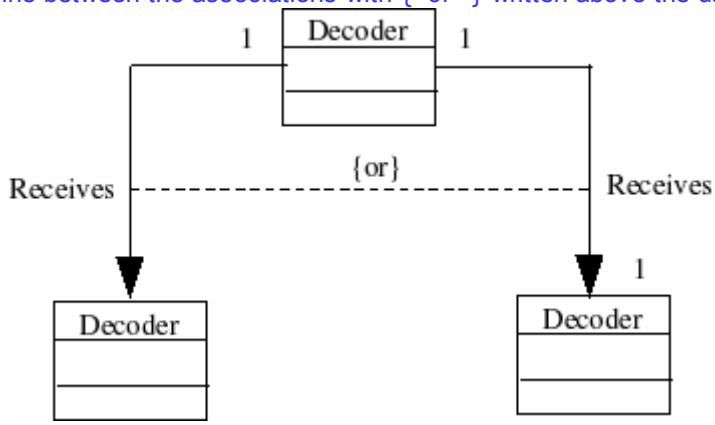

Figure 5.10 Or relationship

A "Decoder" class has an association with either a "Cable", or "Satellite". A dashed line is drawn between the two associations to show it is an "Or" relationship.

Ordered Associations : An instance of an Association is called a Link. If an explicit order is required between the links, it is shown by putting { ordered } next to the association line. The default is unordered.

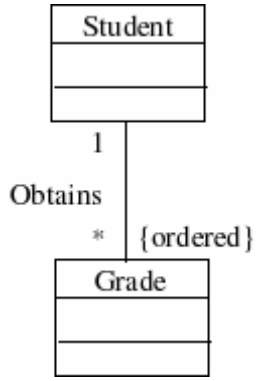

Figure 5.11 Ordered association

The relationship is a 1 to many between "Student", and a "Grade". The association has an {ordered} adornment to show the "Grades" should be ordered.

Aggregation Relationship

Aggregation is a special case of association used to model a “Whole to its Parts” relationship. An Aggregate relationship can be thought of as a "Consists Of" relationship, where the Whole consists of the Parts. For example, we can think of Car as a whole entity and Car Wheel as part of overall Car. The wheel can be created weeks ahead of time, and it can sit in warehouse before being placed on a car during assembly. In this example, the Wheel class's instance clearly lives independently of the Car class's instance. However, there are times when the part class's life cycle is not independent from that of the whole class – this is called composite aggregation. Consider, for example, the relationship of a Company and its Departments are modeled as classes, and a department can't exist before a company exists. Here the department class's instance is dependent upon the existence of the Company class's instance.
There are three types of aggregation associations.

- Basic Aggregation
- Shared Aggregation
- Composite Aggregation

Basic Aggregation: An association with an aggregation relationship indicates that one class is part of another class. In an aggregation relationship, the child class instance can outlive its parent class.

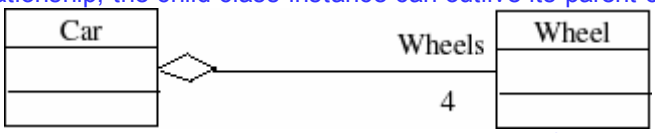

Figure 5.12 Basic Aggregation relationship between class Car and class Wheel

The aggregation is depicted as solid line form the parent class to the part class, and draw an unfilled diamond shape on the parent class's association end. Figure 5.12 shows an example of an aggregation relationship between a Car and a Wheel.
Shared Aggregation : Shared aggregation occurs where the Part in any of the instances of a Whole. This is indicated with the multiplicity of the Whole being depicted by something other than one. For example, a Tutor may teach at more than one Institution. The following shared aggregation example shows that an Institution consists one or more tutors, and a Tutor may teach at one or more Institutions.

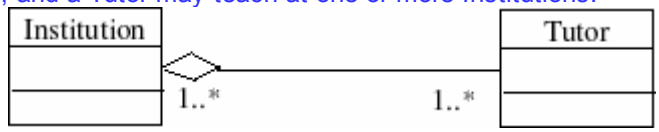

Figure 5.13 Shared Aggregation relationship between class Institution and class Tutor

Composition Aggregation : The composition aggregation relationship is just another form of the aggregation relationship, but the child class's instance lifecycle is dependent on the parent's instance lifecycle. Figure 5.14 shows a composition relationship between a Company class and a Department class. The composition relationship is shown like aggregation relationship, but the diamond shape is filled.

Figure 5.14 Composition relationship between class Company and class Department

In the relationship modeled in figure 5.14, a Company class will have always have at least one Department class instance. Because the relationship is a composition relationship, when the Company class's instance is removed/destroyed, the Department class's instance is automatically removed/destroyed. Another important feature of composition aggregation is that the part class can only be related to one instance of class.

Generalization Relationship

The term **generalization** is used to specify the classification relationship between a general element and a more specific element. In fact, the term 'generalization' specifies a viewpoint focused on a classification hierarchy. For example, an animal is a more general concept than a cat, a dog, or a raccoon. Conversely, a cat is a more specialized concept than an animal.

The more specific element may contain information that is particular to it, as long as it remains completely consistent with the description of the more general element. In the case of classes, the generalization relationship expresses the fact that the elements of one class are also described by another class (in fact, by the *type* of another class). The generalization relationship signifies 'is a' or 'is a kind of'. A cat *is an* animal; that has to do with generalization.

The generalization relationship is represented by an arrow that points from the more specialized class to the more general class. The tip of the arrow is an empty triangle, which allows it to be distinguished from the open arrow that symbolizes the navigation property of associations. In the following example, the **Animal** class is an abstraction of the classes **Cat**, **Dog** and **Raccoon**.

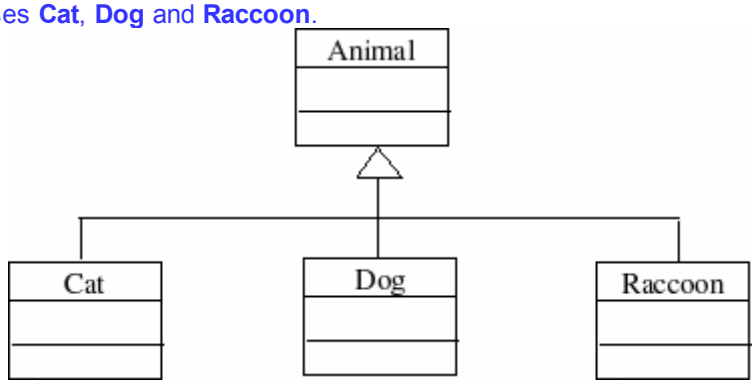

Figure 5.15 Generalization between class Animal and classes Cat, Dog and Raccoon

Realization

The realization relationship relationship specifies a contract between two entities in which one entity defines a contract that another entity guarantees to carry out. For example, an interface defines a set of functionalities as a contract and a class “realizes” the contract by implementing the functionality defined in the contract.

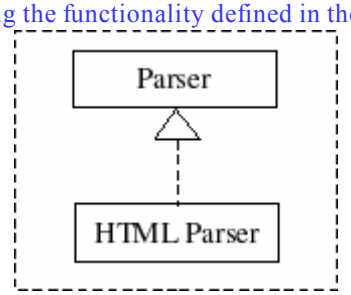

Figure 5.16 Realization

Dependencies

When a class uses another class, perhaps as a member variable or a parameter, and so "depends" on that class, a Dependency relationship is formed. Thus, a dependency exists between two elements if changes to one will affect the other. If for example, a class calls an operation in another class, then dependency exists

between the two. If you change the operation, then the dependent class will have to change as well. A Dependency relationship is indicated by a dotted arrow.
Dependency represents weak relationship between two classes. Dependencies are not implemented with member variables at all, rather than they might be implemented as member function arguments. Consider, for example, the Draw function of the Shape class as shown in figure 5.17. Suppose this function takes an argument of type DrawingContext. This relationship simply means that Shape somehow depends upon DrawingContext. In C++ this almost results in #include.

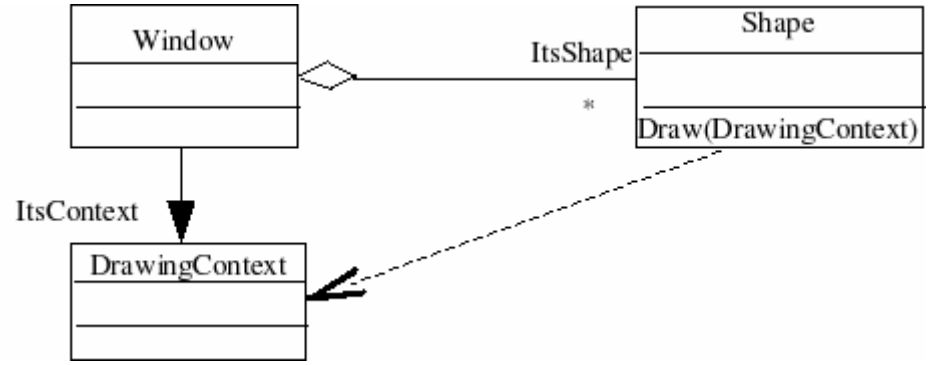

Figure 5.17 An example of Dependency

Association Class

Oftentimes when modeling how classes associate with one another, the association itself specifies a lot of information about the relationship. When such is the case, you may model the association as an 'association class' -- literally specifying that the association has class-like properties, such as attributes, operations, and other associations.
In the UML Class diagram, the association class is shown as a class symbol attached to the association path by a dashed line. Logically, the association class and the association represent the same underlying model element, which has a single name; however, they are graphically distinct. The name may be placed on the path, in the class symbol, or on both.
For example, you are building an application for Human Resources. In your application you want to specify that a class **Person** may have a job with the class **Company**. A person is paid a salary, and so you could model salary as an attribute of the class Person. But what if the person has more than one job (different companies), or even more than one job at your company (they might have a 9-5 job but also work freelance for another department). For a number of reasons, it makes sense to make the association contain job information such as salary.

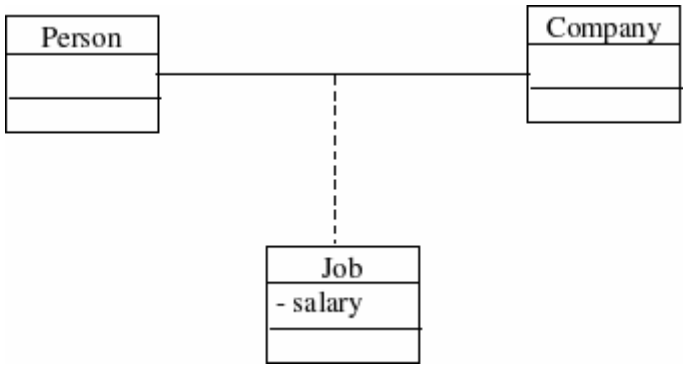

Figure 5.18 An example of Association Class

Previous Content Next