Class relationships

Class relationships

There are four main types of class relationships: association, aggregation, composition, and generalization.

Association

The simplest form of a relationship between two classes is that of an **association**. It is drawn as a solid line (usually straight, but arcs and curves are allowed) between the two classes in the UML diagram. There may be more than one association between the two classes, and a class may be associated with itself. Association lines may cross each other, but an optional "line hop" may be used to help avoid ambiguity.

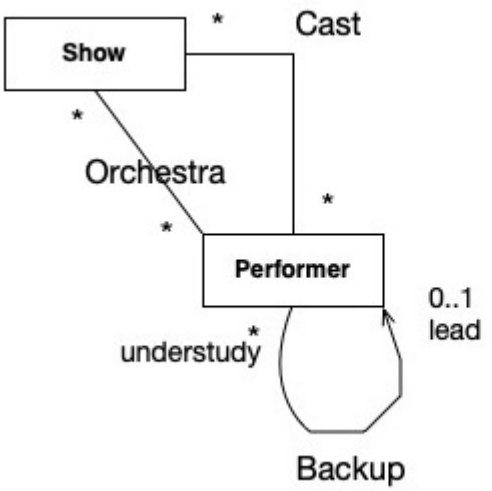
The association itself may have a name, but that is optional. The name should not be placed too close to either end in order to avoid confusing it with the association end name if one is present. The name may be decorated with a solid triangle to help indicate in which direction the naming relationship is to be read. An example of that is shown in the diagram below, where the triangle tells us that the association name is to be read as: *Person works for Company*.

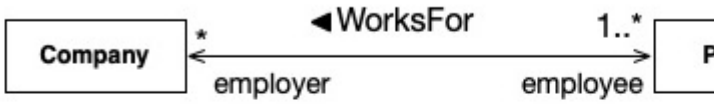
Association ends

Each end of the association may have:

- a name, which is equivalent to declaring an attribute in the class (sometimes called a *role name*) e.g. employee, understudy
- a navigation arrow to indicate that it is the holder of the information (the arrowhead must be "open" i.e. a >-shape) e.g. the understudy Performer "knows about"/"has" a lead Performer
- an associated multiplicity e.g. a Person may have 0 or more employers, a

- an associated multiplicity e.g. a Person may have 0 or more employers, a Company has at least one Employee
- constraints, such as order





Aggregation

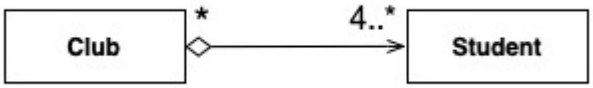
Aggregation is a form of association that describes a "whole-part" relationship where one class, the **aggregate** or whole, is made up of the constituent part class. The end of the association with the aggregate is marked with a special symbol, an open/hollow/white diamond. Only one end of the association may be marked as an aggregate. It is possible to decorate either end of the association with a navigation arrow, or multiplicities, or constraints.

It is possible that the aggregation is shared i.e. a part is shared between one or more aggregates. These parts may also exist independently of the aggregate. An aggregate may or may not be responsible for destroying the parts.

This is often described as a **"Has-A" relationship**, where if *A* has a *B*, **typically**:

- *B* exists independently outside of *A*
- if *A* is destroyed, *B* lives on
- If *A* is copied, then *B* is not deep copied i.e. perform a shallow copy and *B* is shared

For example, a student may belong to zero or more clubs and exists independently of the clubs. A club recognized by the UW Federation of Students needs at least four students to be a part of it, one each to take the roles of president, vice president, secretary, and treasurer.



This is usually implemented via pointers or reference fields.

Composition

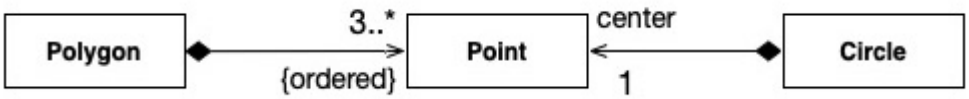
Composition is a stricter form of aggregation. In particular, once a "part" is joined to a "whole", **it may not be shared with any other object**. The whole is also responsible for destroying all of its component parts when it is destroyed. It may also be responsible for creating its components. Whether the components exist independently beforehand or are created by the owner is something to be specified in the design.

This is often described as an **"Owns-A" relationship**, where if *A* owns a *B*, **typically**:

- *B* has no identity or independent existence outside of *A*
- if *A* is destroyed, *B* is destroyed
- If *A* is copied, then *B* is copied i.e. perform a deep copy

Since the owner has an implicit multiplicity of 1, we often don't bother to specify it (although it is not a problem to write the number 1 anyway). Note that the "owner" end of the association is marked with a solid, black diamond.

An example of a composition relationship is that of a Point to a Polygon or a Circle. An individual Point object may be part of an object of either type, but not part of both simultaneously.



This is usually implemented by a composition of classes. For example, an instance of the class `Basis` is composed of two `Vec` objects:

```
class Vec {
    int x, y;
  public:
    Vec( int x = 0, int y = 0 );
    int getX();
    int getY();
    ...
```

```
};

class Basis {
    Vec v1, v2;
  public:
    Basis( Vec v1, Vec v2 );
    ...
};
```

Generalization/specialization

The **generalization** (or **specialization**) association between the parent/superclass and the child/subclass is indicated by putting a triangular arrowhead on the association end that joins the parent. By definition, there is no multiplicity or navigation arrowhead, though constraints may be added