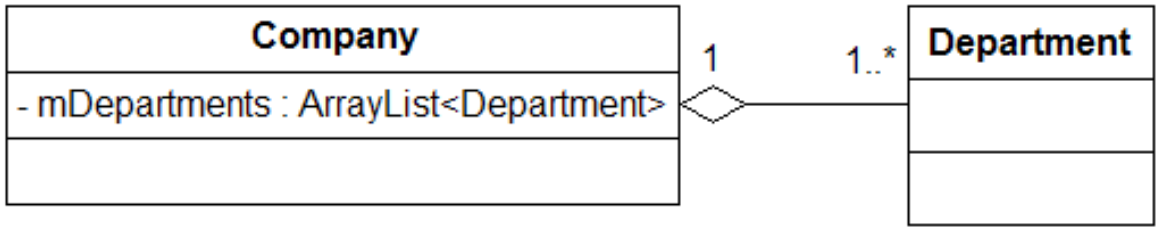
3. Object Oriented Design :: Multiplicities in Aggregation Relationships

Multiplicities may be drawn at one or both ends of the line in an **aggregation relationship** to denote how many objects are aggregated. Common multiplicities are:

0	Zero
1	One
0..1	Zero or one
1..*	One or more
*	Any number

For example, to more explicitly denote that a *Company* consists of one or more *Departments*:



3. Object Oriented Design :: Composition Relationships

A **composition relationship** is similar to an aggregation relationship. Both are forms of a "has a" relationship with the primary difference being that **composition is a stronger form** of "has a" in the sense that the composite object has sole responsibility for the creation and destruction of the components.

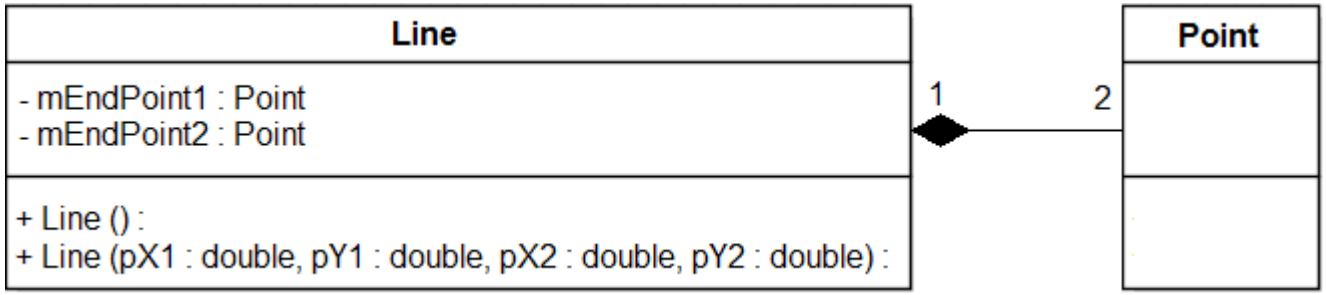
For example, consider a *Line* class where a *Line* has two data members representing the end points of the line; each of these instance variables is a *Point* object. *Line* is called the **composite class** because it is **composed** of *Point*s. *Point* is referred to as the **component class** because *Point*s are **components** of *Line*s.

```
public class Line {
  private Point mEndPoint1;  // This is composition.
  private Point mEndPoint2;  // And so is this.
  public Line(double pX1, double pX2, double pY1, double pY2) {
    mEndPoint1 = new Point(pX1, pY1);
    mEndPoint2 = new Point(pX2, pY2);
  }
}
```

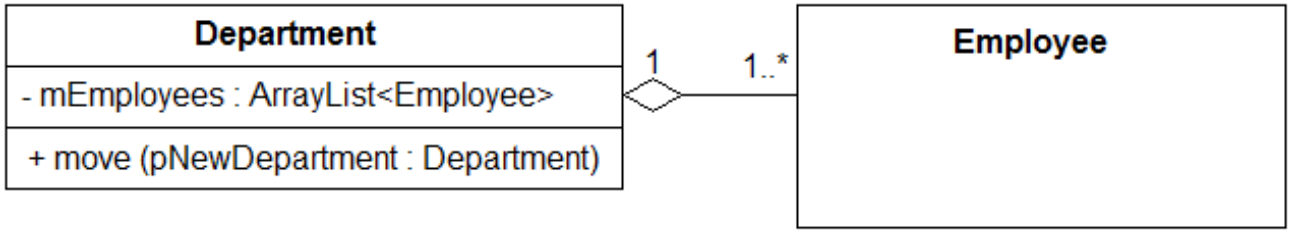
The two *Point* objects are created in the *Line* constructor and when a *Line* object is destroyed, the *Point* objects will also be destroyed, making this a **composition relationship**.

3. Object Oriented Design :: Composition Relationships (continued)

The composition relationship between *Line* and *Point* is represented as a solid line connecting the classes with a filled diamond shape on the composite class end (multiplicities are optional).



Aggregation forms a weaker "has a" relationship in the sense that the component may outlive the aggregate. For example, suppose we have classes *Department* and *Employee* where a *Department* has one or more *Employee*s. If *Employee*s outlive *Department*s, i.e., when a *Department* is eliminated the *Employee*s are moved to another *Department*, then it would be appropriate to model this relationship using aggregation:



Before a *Department* is eliminated, the *Employee*s in the *Department* are moved to a new *Department* by calling the *move*() method.