LAB # 10

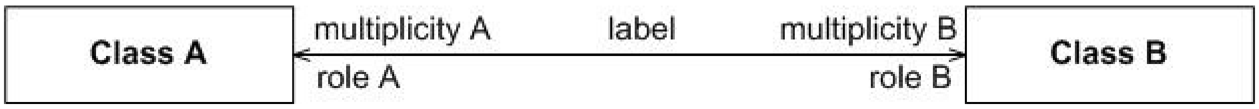
Associations and Inheritance in Class Diagram

Associations

Objects are often associated with, or related to, other objects. For example, as you see in Figure 2 several associations exist: Students are ON WAITING LIST for seminars, professors INSTRUCT seminars, seminars are an OFFERING OF courses, a professor LIVES AT an address, and so on. Associations are modeled as lines connecting the two classes whose instances (objects) are involved in the relationship.

When you model associations in UML class diagrams, you show them as a thin line connecting two classes, as you see in Figure 6. Associations can become quite complex; consequently, you can depict some things about them on your diagrams. The label, which is optional, although highly recommended, is typically one or two words describing the association. For example, professors instruct seminars.

Figure a. Notation for associations.



It is not enough simply to know professors instruct seminars. How many seminars do professors instruct? None, one, or several? Furthermore, associations are often two-way streets: not only do professors instruct seminars, but also seminars are instructed by professors. This leads to questions like: how many professors can instruct any given seminar and is it possible to have a seminar with no one instructing it? The implication is you also need to identify the multiplicity of an association. The multiplicity of the association is labeled on either end of the line, one multiplicity indicator for each direction (Table 1 summarizes the potential multiplicity indicators you can use).

Table 1. Multiplicity Indicators.

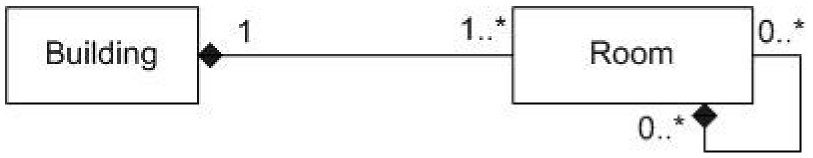
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Another option for associations is to indicate the direction in which the label should be read. This is depicted using a filled triangle, called a direction indicator, an example of which is shown on the *offering of* association between the *Seminar* and *Course* classes of Figure a. This symbol indicates the association should be read "a seminar is an offering of a course," instead of "a course is an offering of a seminar." Direction indicators should be used whenever it isn't clear which way a label should be read. My advice, however, is if your label is not clear, then you should consider rewording it.The arrowheads on the end of the line indicate the directionality of the association. A line with one arrowhead is uni-directional whereas a line with either zero or two arrowheads is bidirectional. Officially you should include both arrowheads for bi-directional assocations, however, common practice is to drop them (as you can see, I prefer to drop them).At each end of the association, the role, the context an object takes within the association, may also be indicated. My style is to model the role only when the information adds value, for example, knowing the role of the *Student* class is enrolled student in the enrolled in association doesn't add anything to the model. I follow the AM practice Depict Models Simply and indicate roles when it isn't clear from the association label what the roles are, if there is a recursive association, or if there are several associations between two classes.

Composition Associations

Sometimes an object is made up of other objects. For example, an airplane is made up of a fuselage, wings, engines, landing gear, flaps, and so on. Figure 8 presents an example using composition, modeling the fact that a building is composed of one or more rooms, and then, in turn, that a room may be composed of several subrooms (you can have recursive composition). In UML 2, aggregation would be shown with an open diamond.

Figure 8. Modeling composition.



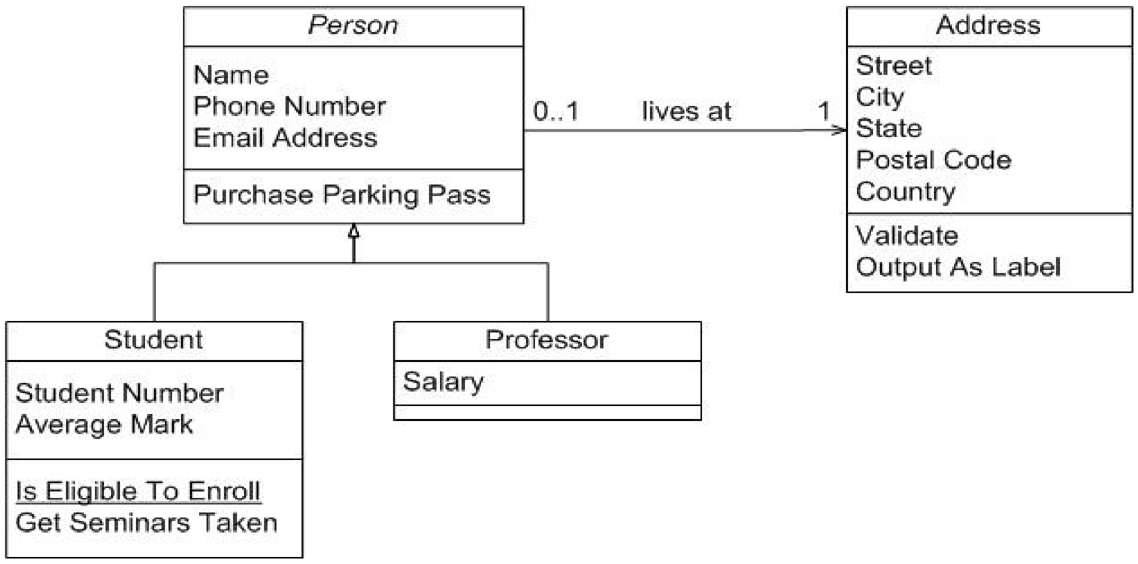
I'm a firm believer in the "part of" sentence rule -- if it makes sense to say that something is part of something else then there's a good chance that composition makes sense. For example it makes sense to say that a room is part of a building, it doesn't make sense to say that an address is part of a person. Another good indication that composition makes sense is when the lifecycle of the part is managed by the whole -- for example a plane manages the activities of an engine. When deciding whether to use composition over association, Craig Larman (2002) says it best: If in doubt, leave it out. Unfortunately many modelers will agonize over when to use composition when the reality is little difference exists among association and composition at the coding level.

Inheritance Relationships

Similarities often exist between different classes. Very often two or more classes will share the same attributes and/or the same methods. Because you don't want to have to write the same code repeatedly, you want a mechanism that takes advantage of these similarities. Inheritance is that mechanism. Inheritance models "is a" and "is like" relationships, enabling you to reuse existing data and code easily. When *A* inherits from *B,* we say *A* is the subclass of *B* and *B* is the superclass of *A*. Furthermore, we say we have "pure inheritance" when *A* inherits all the attributes and methods of *B*. The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.

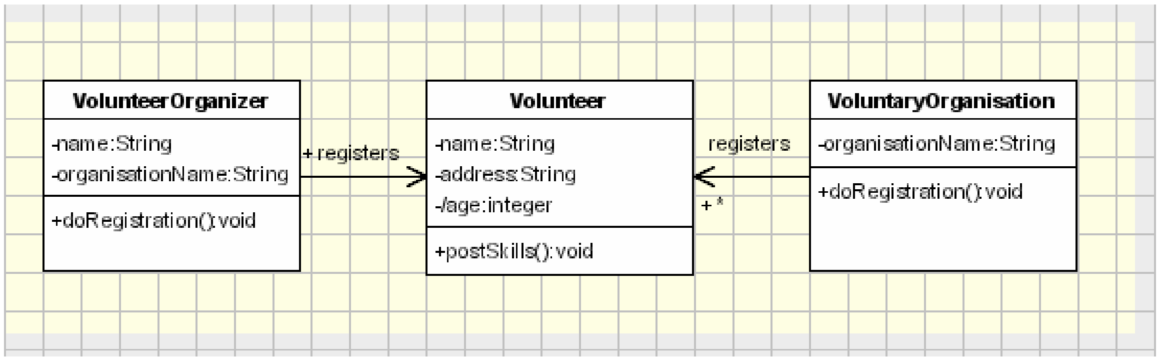
Many similarities occur between the *Student* and *Professor* classes of Figure b. Not only do they have similar attributes, but they also have similar methods. To take advantage of these similarities, I created a new class called *Person* and had both *Student* and *Professor* inherit from it, as you see in Figure b. This structure would be called the *Person* inheritance hierarchy because *Person* is its root class. The *Person* class is abstract: objects are not created directly from it, and it captures the similarities between the students and professors. Abstract classes are modeled with their names in italics, as opposed to concrete classes, classes from which objects are instantiated, whose names are in normal text. Both classes had a name, e-mail address, and phone number, so these attributes were moved into *Person*. The *Purchase Parking Pass* method is also common between the two classes, something we discovered after Figure b was drawn, so that was also moved into the parent class. By introducing this inheritance relationship to the model, I reduced the amount of work to be performed. Instead of implementing these responsibilities twice, they are implemented once, in the *Person* class, and reused by *Student* and *Professor*.

Figure b. Inheritance hierarchy.



Lab Work and Assignment:

Q: 4.6 **(Chapter Number 4 Supplementary Problem)**


VolunteerOrganizer
-name:String
-organisationName:String
+doRegistration():void
+registers
Volunteer
-name:String
-address:String
-/age:integer
+postSkills():void
registers
+ *
VoluntaryOrganisation
-organisationName:String
+doRegistration():void