[L3P1]

Object-Oriented Programing: Basics

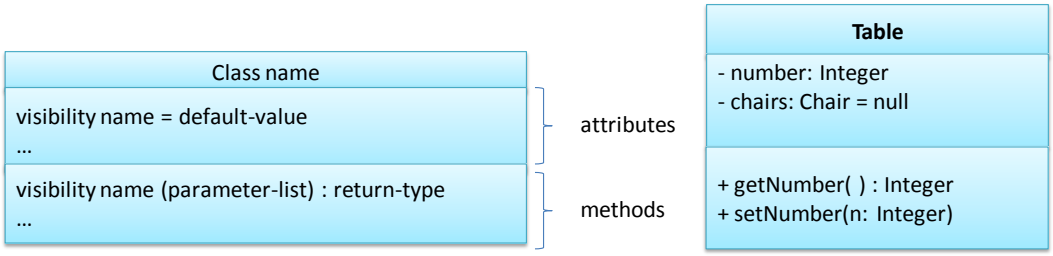
Introduction to Object-Orientation

Instead of writing our own handout, we refer you to the document *Object-Oriented Programming with Objective-C* released by Apple Inc. This is compulsory reading for those who are new to Object Oriented (OO) Programming. In spite of the title, the document is mostly programming language independent; you may ignore any references to Objective C.

Creating an OO solution

Class diagrams

UML *class diagrams* describe the structure (but not the behavior) of an OO solution. These are possibly the most often used diagrams in the industry and an indispensable tool for an OO programmer.



The above illustrates the basic notations of a class diagram. In particular, attributes represent the data of the class, as opposed to methods that represent the operation (or behavior).

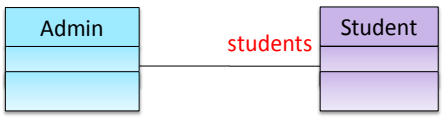
Visibility

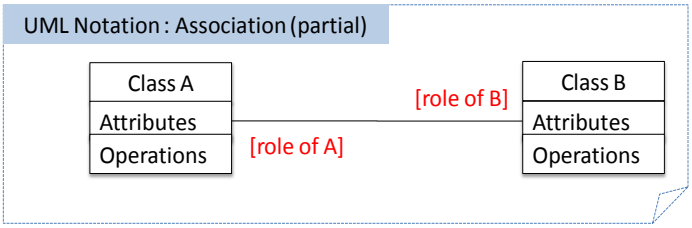
The *visibility* of attributes and operations is used to indicate the level of access allowed for each attribute or operation. The types of visibility (and their exact meaning) depend on the programming language used. Here are some common visibilities and how they are indicated in a class diagram:

+ public - private # protected ~ package

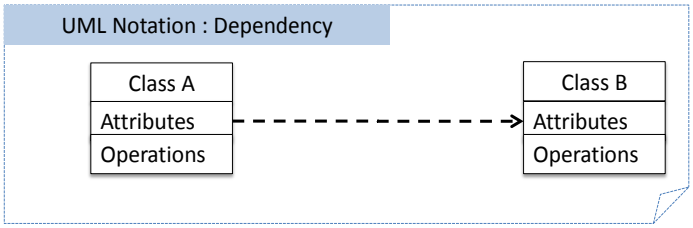
Associations

Within an OO solution, objects collaborate with one another. These collaborations are represented as connections (or *associations*). Each association is denoted as a line between two classes in a UML class diagram. As an example, the Admin object needs to be associated to Student objects, so as to access information. The association is depicted as follows.

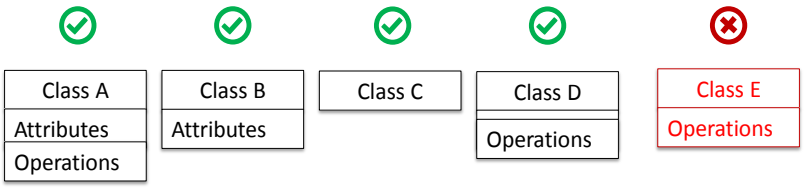




Note that an association link in a class diagram denotes a permanent or semi-permanent structural link between objects of two classes. Temporary contacts between objects (e.g. ClassA uses a constant defined in ClassB) are not shown as associations. Such temporary contacts are shown as *dependencies* instead (denoted as a dashed arrows).

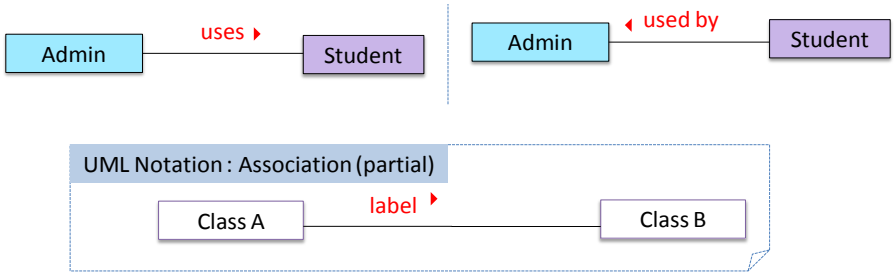


The 'Operations' compartment (or even both 'Attributes' and 'Operations' compartments) may be omitted if such details are not important for the task at hand. For simplicity, these two compartments are omitted from most of the subsequent class diagrams.



'Role' labels are optionally used to indicate the role played by the classes in the relationship. For example, the association given below represents a marriage between a Man object and a Woman object. The respective roles played by objects of these two classes are 'husband' and 'wife'.

Association labels are used to describe the association. In the diagram below, "an Admin object uses Student objects" or "Student objects are used by an Admin object". The arrow head indicates the "direction" in which the label is to be read.

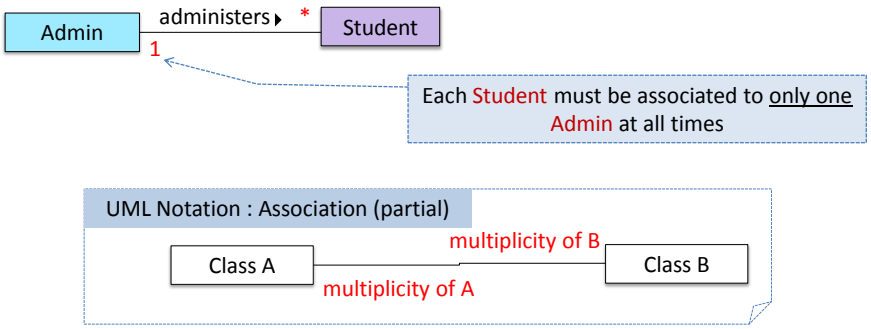


A class diagram can also indicate the *multiplicity* of an association. Multiplicity is the number of objects of a class that participate in the association.

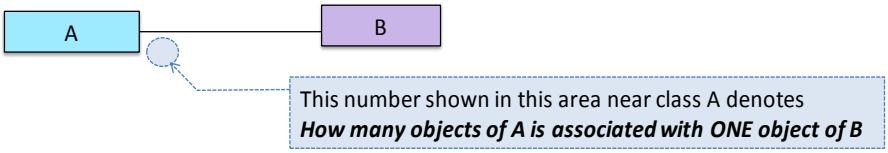
Commonly used multiplicities:

- 0..1 : optional, can be linked to 0 or 1 objects
- 1 : compulsory, must be linked to one object at all times.
- * : can be linked to 0 or more objects.
- n..m : the number of linked objects must be n to m inclusive

In the diagram below, an Admin object administers (in charge of) any number of students but a Student object must always be under the charge of exactly one Admin object.



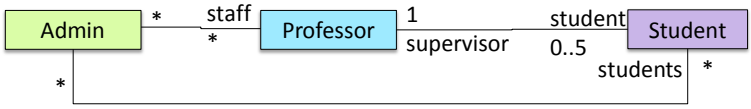
This proper use of multiplicity is shown below:



As an exercise, suppose the following additional details are included:

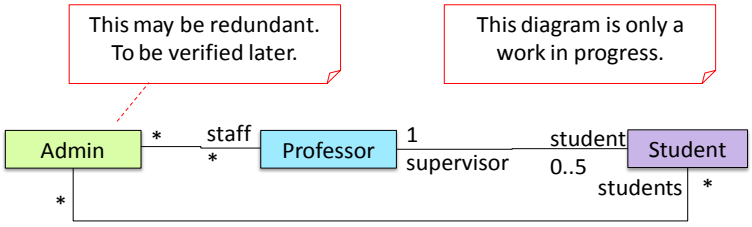
- Each Student must be supervised by a Professor.
- Students have matriculation numbers. A Professor cannot supervise more than 5 students.
- Admin staff handles Professors as well.

Here is the updated class diagram:

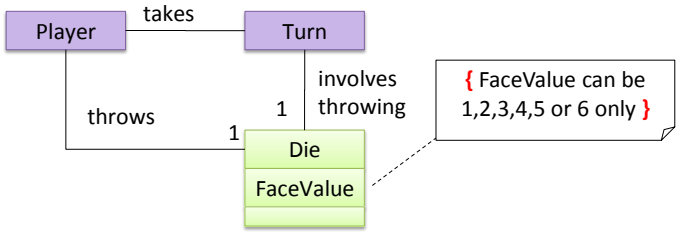


UML notes and constraints

UML notes are used to augment a UML diagram with additional information. These notes can be shown connected to a particular element in the diagram or can be shown without a connection. The diagram below shows examples of both.

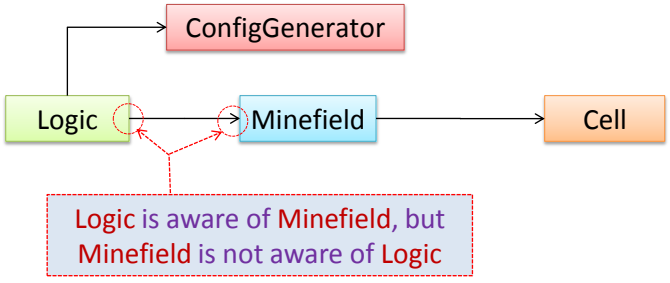


A note can be used to specify a *constraint* within curly brackets. Natural language or a formal notation such as OCL (Object Constraint Language) may be used to specify constraints. OCL is beyond the scope of this handout.

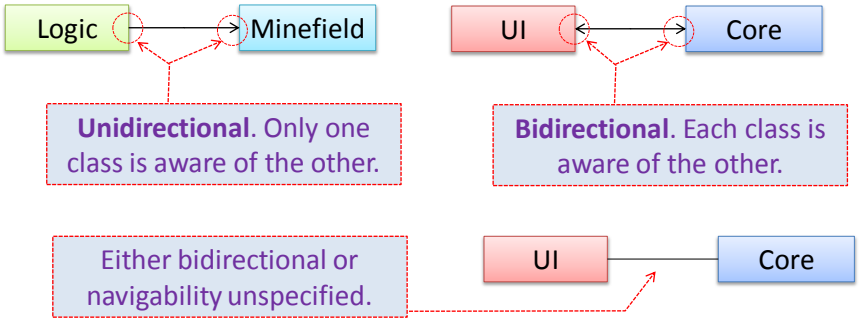


Navigability

Navigability is another detail that can be shown in class diagrams. Navigability (denoted as an arrowhead in the association link) indicates whether a class involved in an association is "aware" of the other participating class.

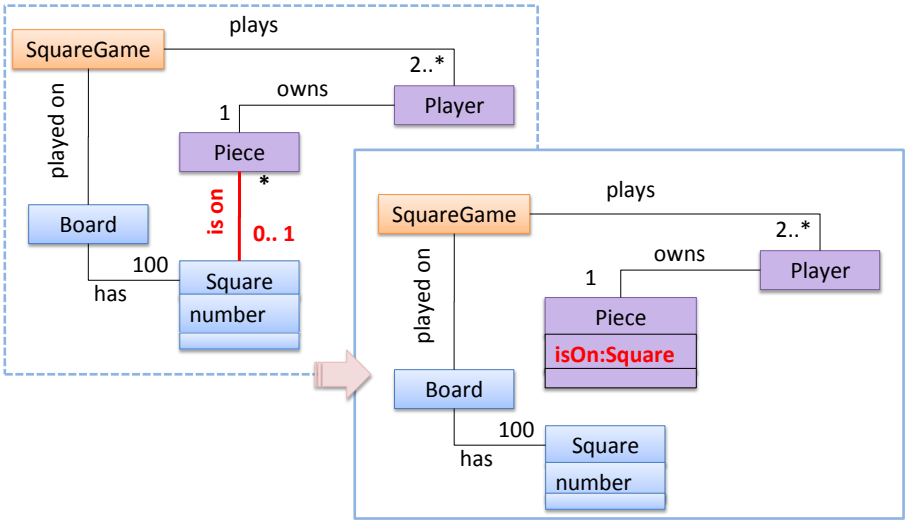


At the level of program code, navigability implies how objects are referenced. In the example above, a Logic object holds a reference to a ConfigGenerator object, but not the other way around. Navigability can be *unidirectional* or *bidirectional*. For convenience, a line without arrow heads can be used to indicate a bidirectional link. However, if none of the associations in a class diagram has navigability arrows, it usually means that navigability is "unspecified".



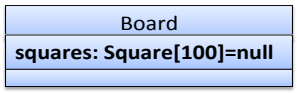
Associations as attributes

An attribute is sometimes used to represent an association. The diagram below depicts a multiplayer *Square Game* being played on a board comprising of one hundred squares. Each of the squares may be occupied with any number of Pieces, each belonging to a certain player. A piece may or may not be on a square. Note how the association 'is on' can be replaced by an isOn attribute of the Piece class. In order to realize the 0..1 multiplicity, the isOn attribute can either be null or hold a reference to a Square object.



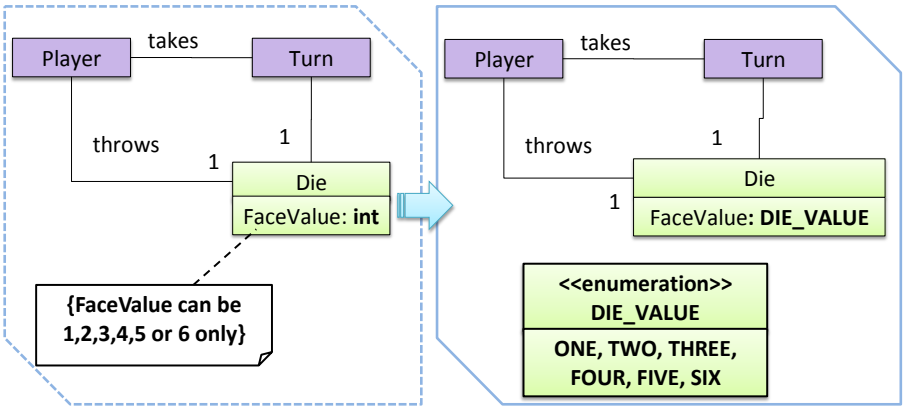
In addition, association multiplicities of two or more can be made as part of the attribute using

name: type [multiplicity] = default value



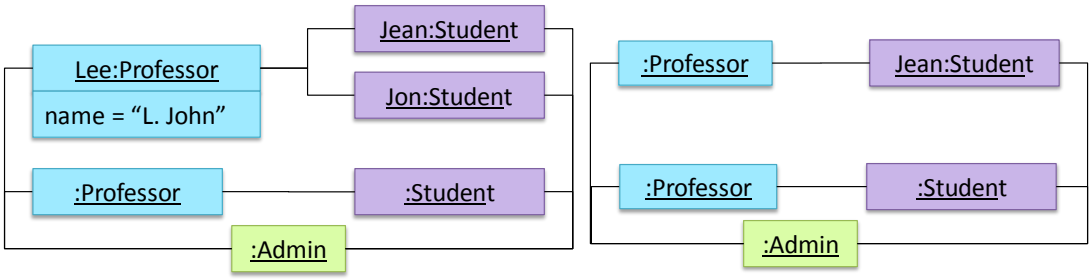
Enumerations

An <<enumeration>> is used to indicate that an attribute can only take on a fixed set of values.



Object diagrams

Sometimes it is useful to show objects instead of classes. These are *object diagrams*. An object diagram shows an object structure at a given point of time while a class diagram represents the general situation. Given below is an example.



Object names are depicted differently from class names. Firstly, object names are underlined. Secondly, each object may be given an 'instance name', in addition to the class name, using the format instance name: class name. An example is Jean:Student where Jean is the object instance name and Student is the class name. Also note the omission of association roles, most attributes, and some object names, as they do not add value to the diagram. It also does not make sense to show methods in an object diagrams.

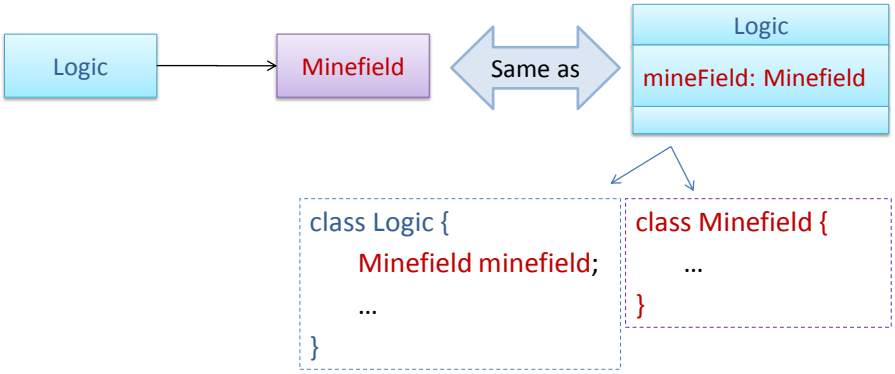
Both object diagrams are derived from the same class diagram shown earlier. In other words, each of these object diagrams shows 'an instance of' the same class diagram.

Implementing basic class structures

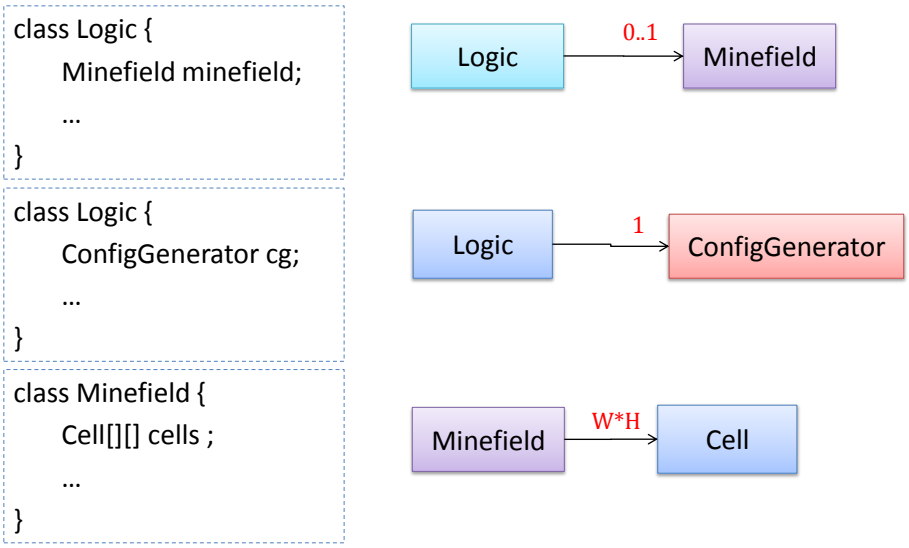
Most OO languages have direct ways of implementing the class structures.

Java	Classes → Java classes Attributes → Java variables Operations → Java methods
C++	Classes → C++ classes (and header files) Attributes → C++ variables Operations → C++ functions

Note that reference variables are used to implement associations, but not primitive variables such as int.

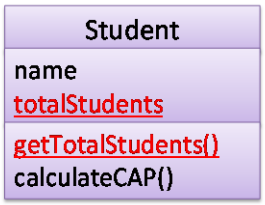


A variable gives us a 0..1 multiplicity (also called *optional associations*) because a variable can hold a reference to a single object or null. A variable can be used to implement a 1-multiplicity too (also called *compulsory associations*). To implement other multiplicities, choose a suitable data structure such as Arrays, ArrayLists, HashMaps, Sets, etc.



Class-level members

Most OO languages allow defining class-level (also called static) attributes and operations. They are like 'global' variables/functions but attached to a particular class. For example, the variable totalStudents of the Student class can be declared static because it should be shared by all Student objects. However, the variable name should not be static as each Student should have its own name. Similarly, getTotalStudent() can be a static operation. In UML class diagrams, underlines are used to denote class-level attributes and variables.



Single Responsibility Principle

The Single Responsibility Principle (SRP) states,

> A class should have one, and only one, reason to change.

If a class has only one responsibility, it needs to change only when there is a change to that responsibility. A counter example is a TextUi class that does parsing of the user commands as well as interacting with the user. That class needs to change when the formatting of the UI changes as well as when the syntax of the user command changes. Hence, such a class does not follow the SRP. Refer the following article for a longer write up on SRP:

> http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Encapsulation in OOP

An object is an *encapsulation* some data and related functions. i.e.,

1. It *packages* those data and related functions together into one entity.
2. The data is hidden from the outside world (we call this concept *information hiding*) and are only accessible using the functions.

References

[1] Object-Oriented Programming with Objective-C , A document by Apple inc., retrieved from http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf

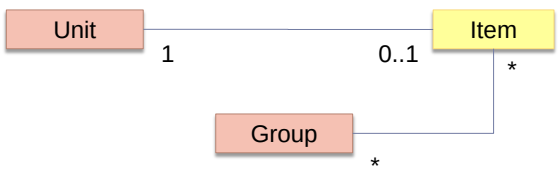
Worked examples

[Q1]

(a) Which of the following class diagrams match with the object diagram below? For example, class diagram (1) matches with the object diagram because the object diagram could be an instance of the class diagram given.

(1)

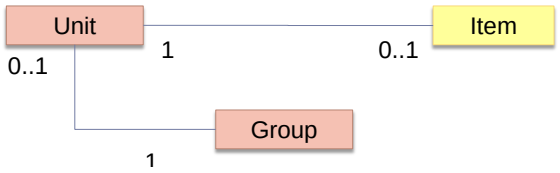
(2)



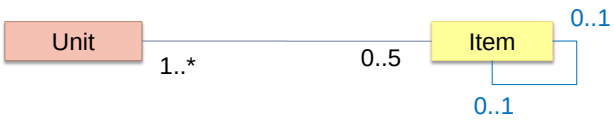
(3)

(4)

(5)



(b) Which of the following object diagrams are allowed by the class diagram below?



(1)

(2)

(3)

(4)

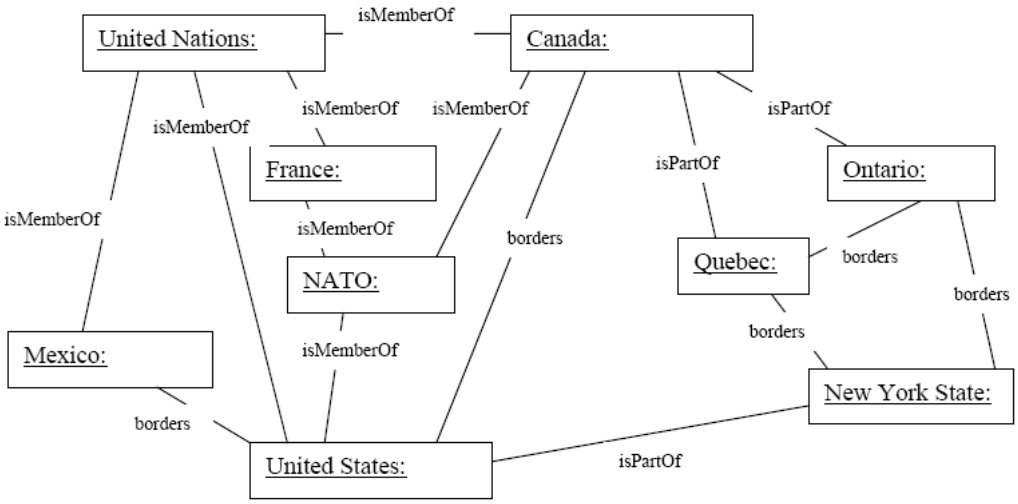
[A1]
(a)

(1) matches
(2) matches
(3) does not match – According to this model, there should be at least 2 Items per Unit.
(4) matches
(5) does not match – According to this model, a Unit must have a link to a Group.

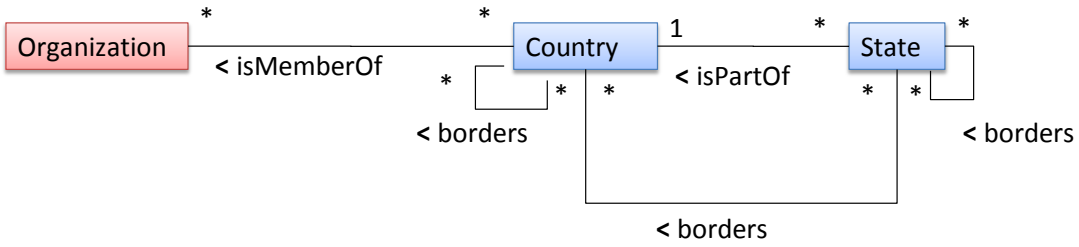
(b)

(1) allowed
(2) not allowed. One item is not linked to a Unit
(3) Not allowed. An Item cannot be linked to more than one other Item.
(4) Allowed. A unit can exist without an Item.

[Q2]
Infer the class names that are missing in the following object diagram. Draw a class diagram that could possibly generate this object diagram.



[A2]



Note: This answer does not use inheritance as it was not covered in this handout.

[Q3]
First, draw an object diagram to represent the following description. Then, use the object diagram to draw a class diagram. Be sure to indicate the multiplicity and label the associations.

Sichuan, Jiangsu, and Guandong are all provinces in China. Singapore, Malaysia, and China are all countries in Asia. Beijing is the capital of China. Kuala Lumpur is the capital of Malaysia.

[A3]