

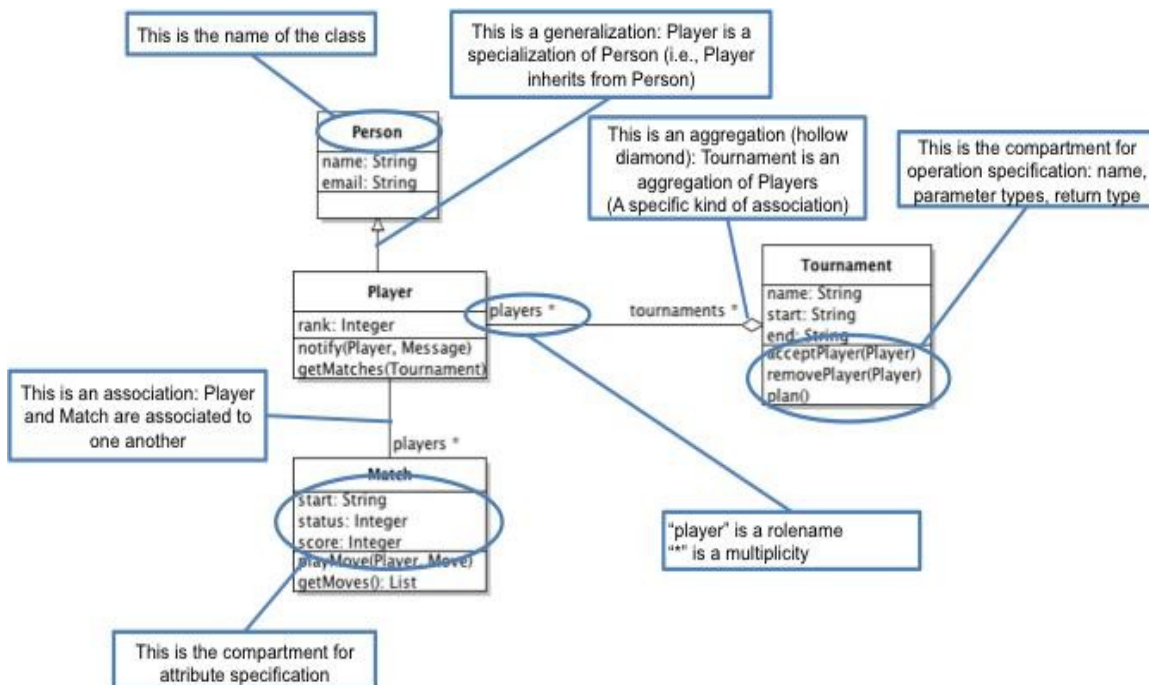
In this lab we will be working on making a class diagram for a given problem. For making a class diagram you will be required to identify classes in a system, relationships among the classes and multiplicities between them.

### Attendance/Demo

To receive credit for this lab, you must make reasonable progress towards completing the exercises. When you have finished all the exercises, call your TA, who will review your work. For those who don't finish early, the TA will ask you to show whatever diagrams you have completed, starting at about 15 minutes before the end of the lab period. Finish any exercises that you don't complete on your own time.

### Basics

A Sample Class diagram is shown below, followed by a description.



### **A little bit on the semantics of the UML class diagram notation.**

A rectangle with three compartments is used to specify a class: the top compartment shows the class name (if the name is written in italic, this means the class is abstract), the middle compartment shows the attribute specifications, and the bottom compartment shows the operation specifications.

#### **Attribute specification:**

An attribute is specified with a name and a type.

Attributes can only have primitives types (Boolean, Integer, Real, String), or user-defined types.

Attribute that refer to other classes are not indicated: instead, the attribute name is the role name shown on the association.

Additionally, an attribute can have a visibility (similarly to what you know in programming languages) as indicated by the symbol that appears before the attribute name:

+ name: type	this is a public attribute
- name: type	this is a private attribute
# name: type	this is a protected attribute
~ name: type	this is a package-visible attribute

If an attribute specification is underlined, this means the scope of the attribute is the class (as opposed to the object/instance). In other words, the attribute is static (programming concept).

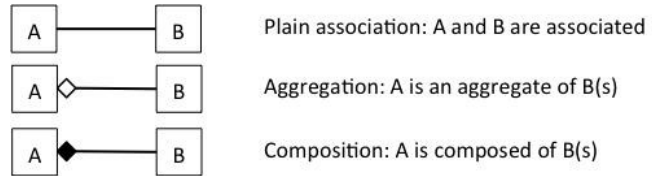
#### **Operation specification**

An operation is specified with a name, a list of parameters, a visibility symbol, and a return type (optional). The return type can be one of the primitive types or a class in the class diagram. Each parameter can be specified with an optional direction (in/out/inout), an optional parameter name, and a type. Similarly to return types, parameter types can be primitive types or classes in the class diagram. An *in* parameter is passed by the caller, an *inout* parameter is passed by the caller and possibly modified by this operation, and an *out* parameter is not passed by the caller but returned by this operation.

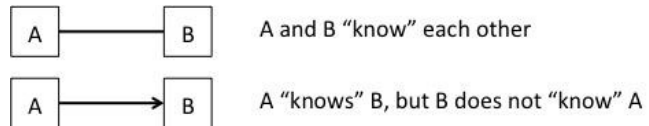
If an operation specification is underlined, this means the scope of the operation is the class (as opposed to the object/instance). In other words, the operation is static (programming concept).

## Association specification

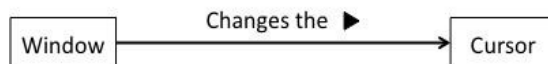
An association is a plain association (plain line), an aggregation (hollow diamond), or a composition (full diamond).



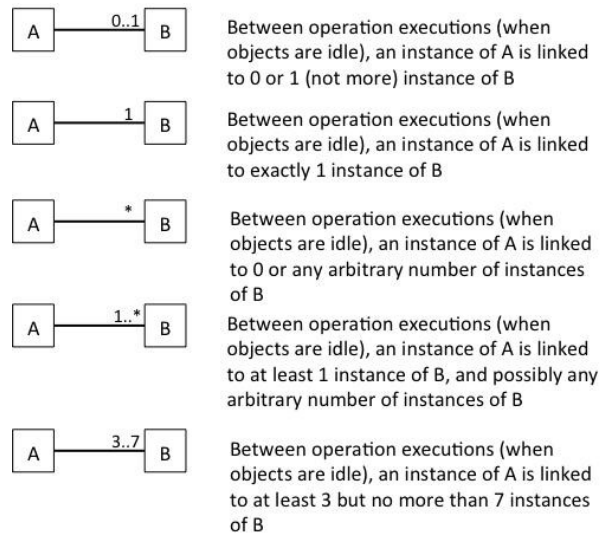
An association can have a direction.



An association can have a name. A name is optional. When using a name, an indication (triangle) can be added to help "read" the association. The association name appears in the middle between the two classes.



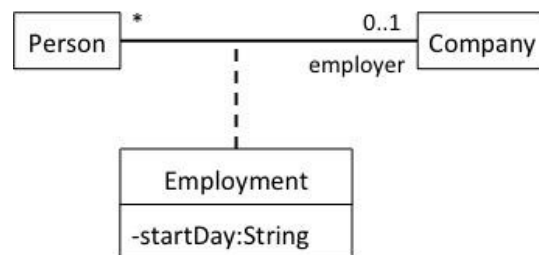
An association can have multiplicities at both ends. Multiplicities are strongly recommended! They specify the number of instances of the classes at each end of the association that can be linked at runtime.



The default (omitted) multiplicity is 1.

An association can carry data and behavior, in which case it is at the same time an association and a class: this is called an association class.

An association class is used when we have an association between two classes and there exist data (and possibly behavior) that do not belong to any of the two associated classes but instead to the relation between those two classes<sup>1</sup>.



## Exercise

For a project planning system (something like Basecamp!), we want to model the domain of project management.

A project has a project manager, and any number of other participants. People can participate in multiple projects.

A project can be decomposed into activities. Activities are related where one predecessor activity may have any number of successor activities, and one successor may have at most one predecessor activity.

An activity involves any number of work products (which can be documents or software packages): the work products can be inputs or outputs to the activity. Work products are the result of a single activity but may be inputs to several other activities.

An activity can be decomposed into tasks, that can be assigned to (one or several) participants. Tasks are related where one predecessor task may have any number of successor tasks, and one successor may have at most one predecessor tasks.

Draw a class diagram to model these concepts, showing classes, their relationships (generalization, association, aggregation, composition) as appropriate, roles and multiplicities on associations. Include 1 or 2 attributes that make sense for each class. You can leave out operations.

---

<sup>1</sup> Important note: Violet does not support association classes! This is one inconvenient of using a free software: the whole UML notation is not supported. If you want to specify an association class, you need a work-around. I suggest the following: add a note, add two note connectors, one between the note and the class that should be the association class and one between the note and the association that should be the association class, and write in the note that the class and the association connected by the notes are one single association class. However, in this exercise you don't necessarily need association classes.