The Unified Modeling Language (UML)

Instructor: Dr. Metin Özkan

M. Ozkan, 02.2012, Ver. 1.0

The Unified Modeling Language - UML

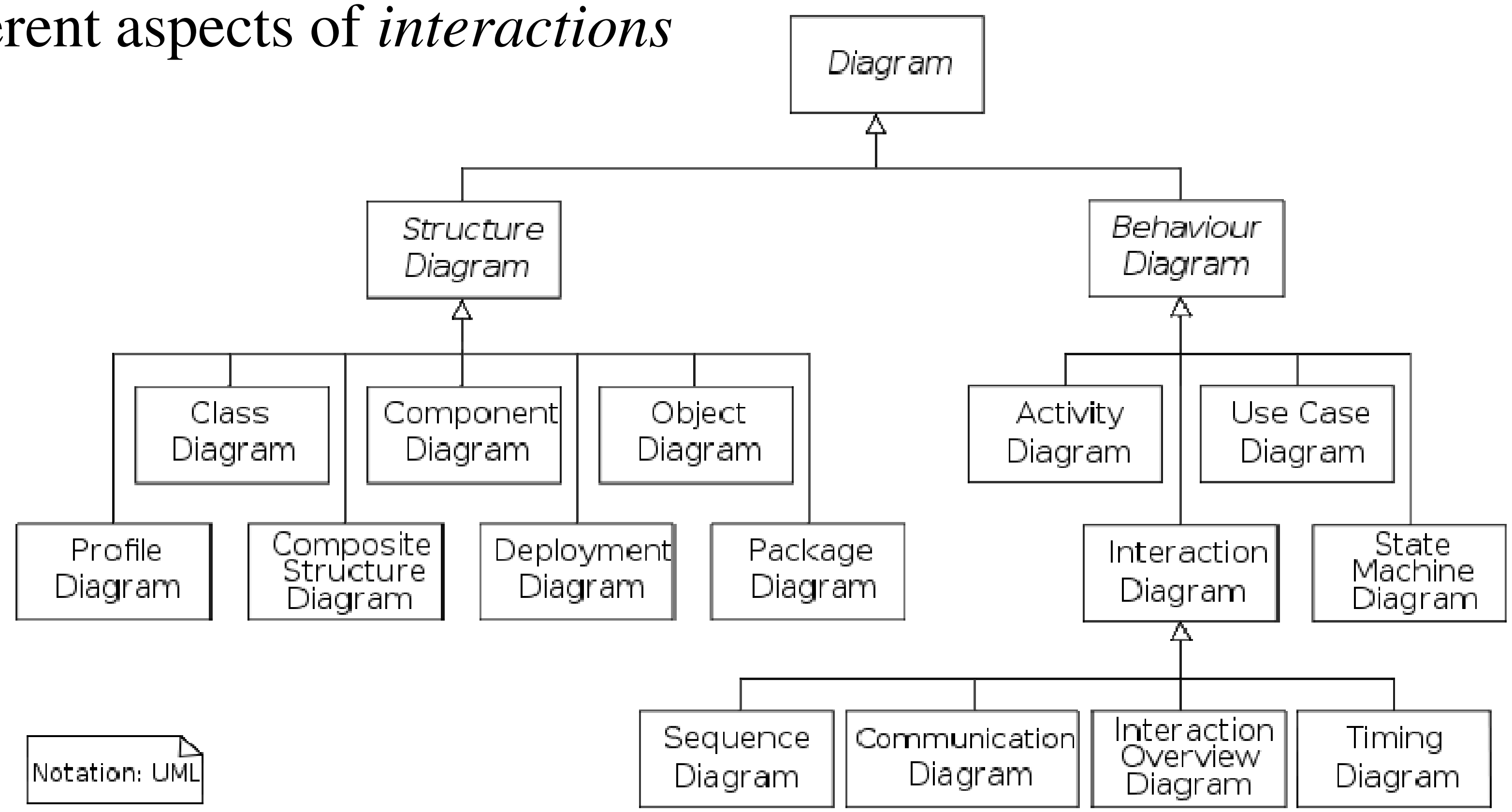
- The UML is a visual language for specifying, constructing, and documenting the artifacts of a software.
- The UML is not a method to design systems, it is used to visualize the anaysis and design.
- It makes easier and to understand and document software systems.
- It supports teamwork because UML diagrams are more understandable than the program code.
- There are different kinds of UML diagrams, which are used in design and coding levels

The Unified Modeling Language - UML

- The current specification of the UML is available in the web site of the Obect Management Group (OMG). http://www.omg.org

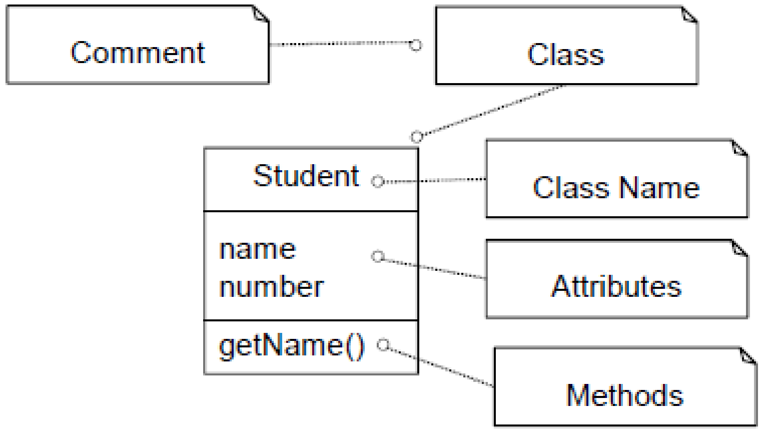
The Unified Modeling Language - UML

- UML 2.2 has 14 types of diagrams divided into two categories: Seven diagram types represent *structural* information, and the other seven represent general types of *behavior*, including four that represent different aspects of *interactions*

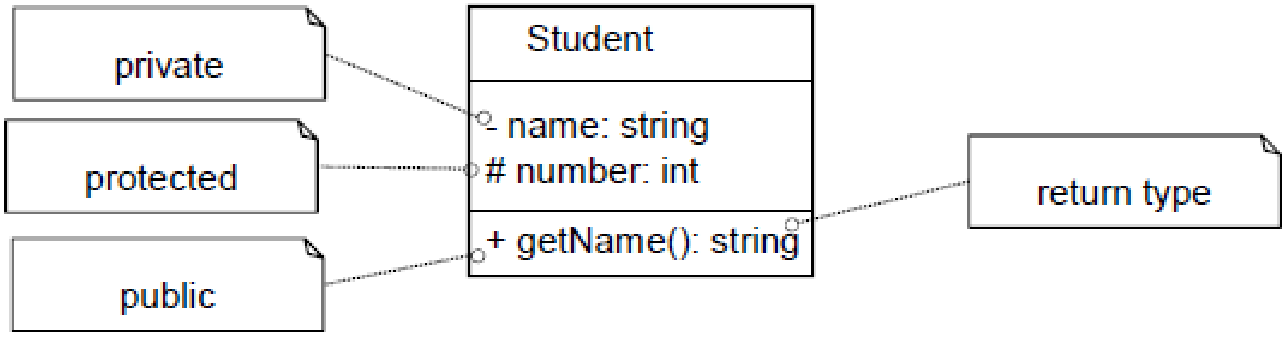


Class Diagrams

- A class diagram shows the structure of the classes and the relationships between them.



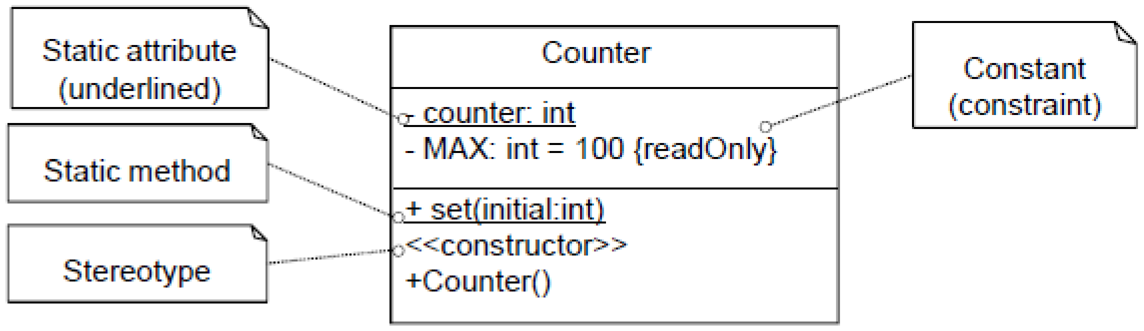
- If necessary, access modes and data types may be also shown.



Ref: http://web.itu.edu.tr/buzluca/

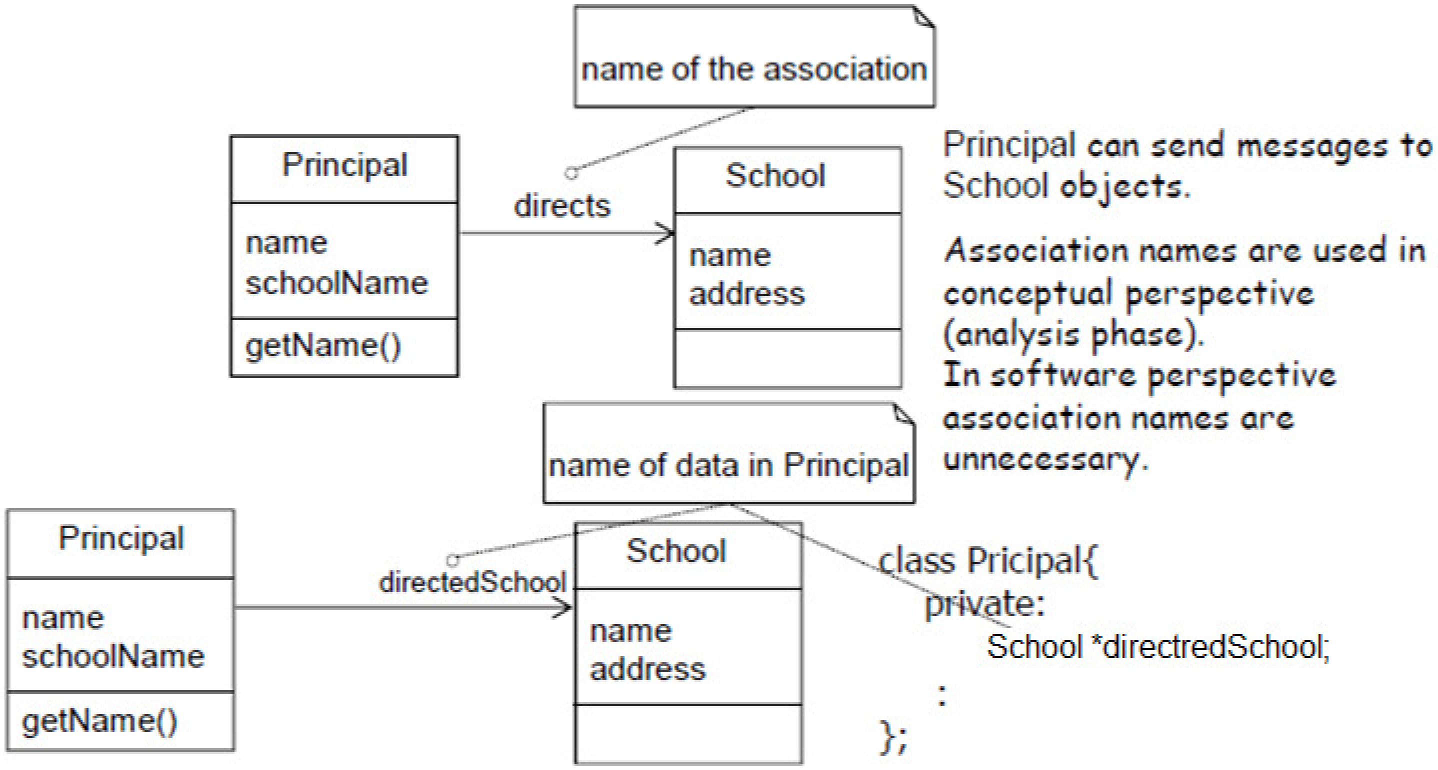
Class Diagrams

- Comments: are placed in dog eared rectangles.
  - *You can use comments to put anything you want in a diagram. You can use comments to add application and program specific details.*
- Stereotypes: is a way of extending the UML in a uniform way, and remaining within the standart.
  - *You indicate a stereotype using: <<stereotype name>>*
- Constraints: is a text string in a curly braces ({usually language specific}) . The UML defines a language (Object Constrain Language - OCL) that you can use for writing constraints.



Class Diagrams (Relationship)

- A class diagram also shows the relationship between classes such as association, aggregation, composition, and inheritance
- Association: A general type of relationship. Objects of a class can send messages to objects of another class.



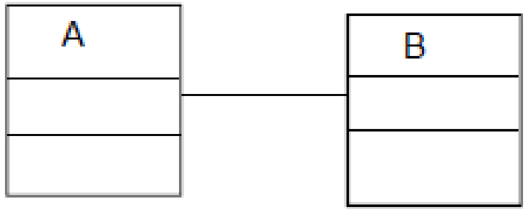
Principal can send messages to School objects.

Association names are used in conceptual perspective (analysis phase).
In software perspective association names are unnecessary.

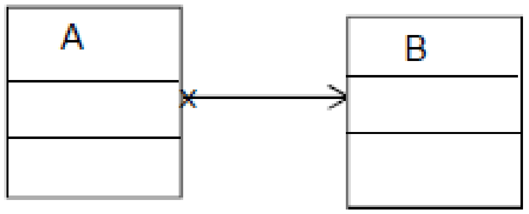
```
class Pricipal{
   private:
      School *directredSchool;
   :
};
```

Class Diagrams (Relationship)

- Direction of the message flow:

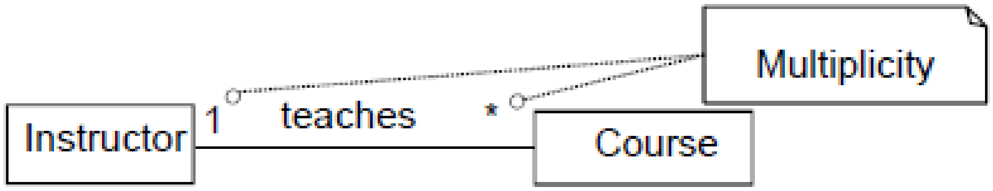


Direction of messages is unspecified. Both may send messages to each other.



A can send messages to B.
A get a service from B.
B can not send messages to A.

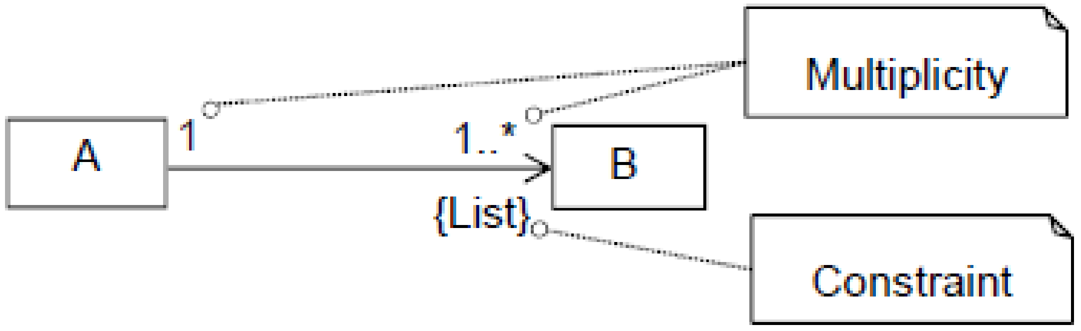
- Multiplicity: indicates the number of any possible combination of objects of one class associated with objects from another class.
  - *In other words, it shows the number of objects from that class, that can be linked at runtime with one instance of the class at the other end of the association line*

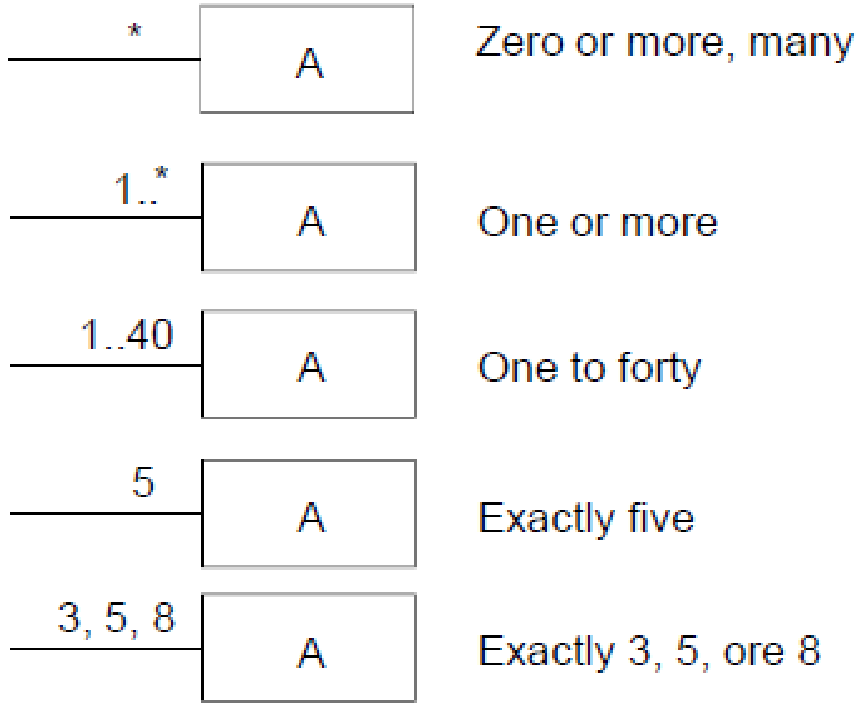


  - *An instructor teaches zero or more courses. An association may also read in reverse order. A course is given exactly by one instructor.*

Class Diagrams (Relationship)

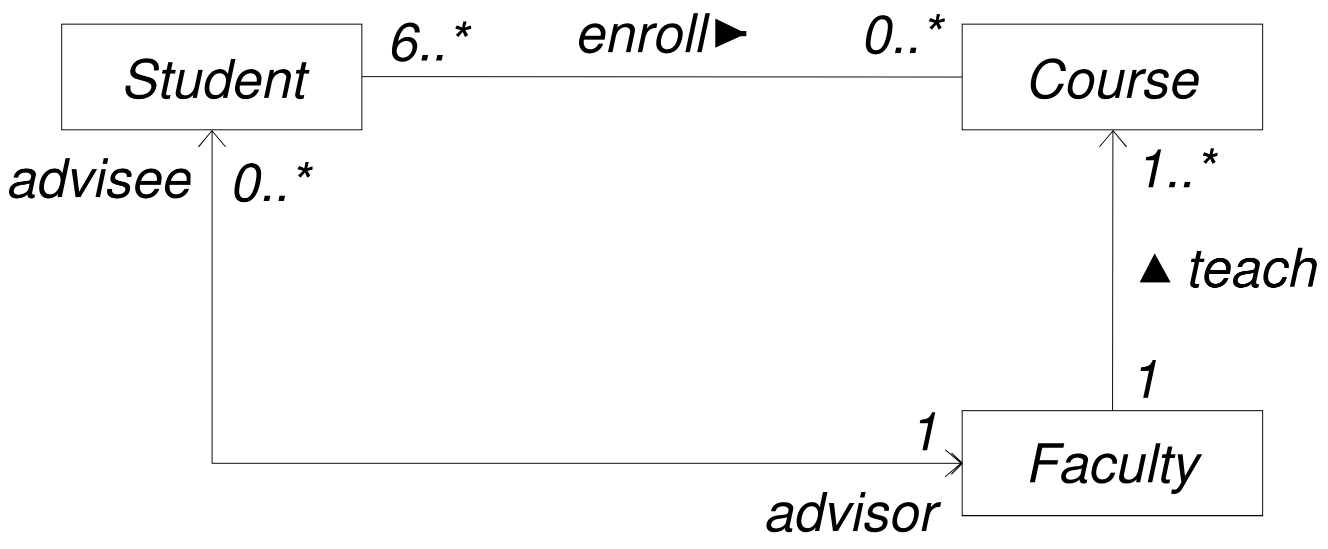
- Example: One object of class A is associated with one or more objects of class B at a time.
- Class A includes a list that can contain one or more objects of class B.





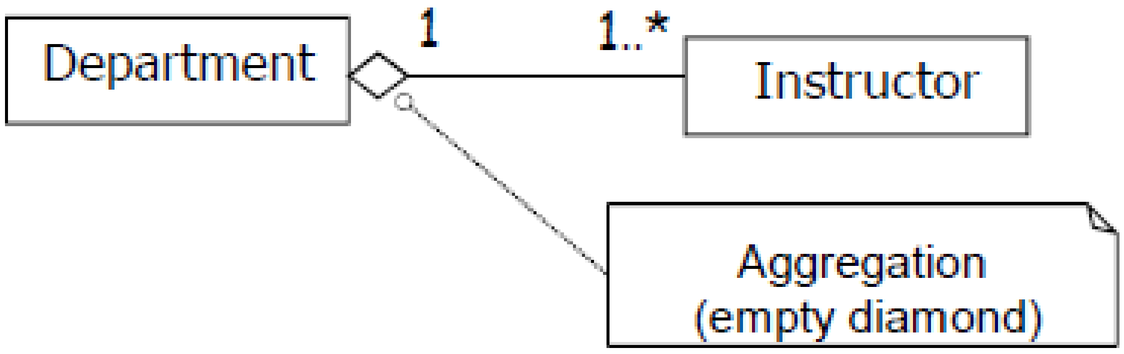
Class Diagrams (Relationship)

- Example



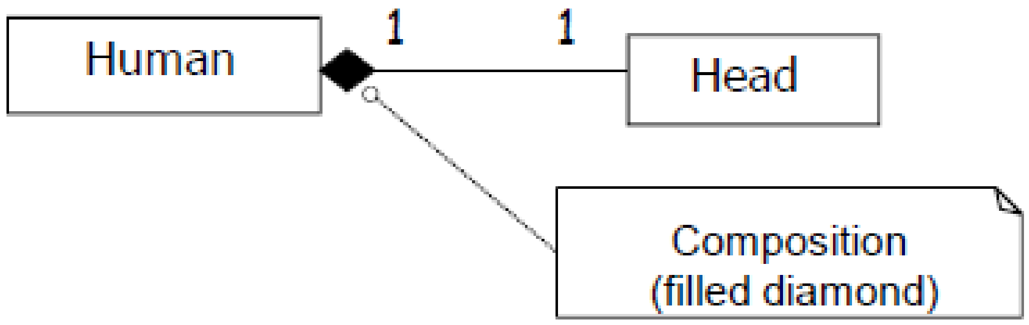
Class Diagrams (Relationship)

- Aggregation ("is part of"), composition ("is entirely made of"): Both are a type of association. They are qualified by a "has a" relationship.
- There is a small difference between them.
- Aggregation: It indicates a "Whole/part" relationship.
  - *A department of the faculty has instructors.*
  - *Parts (instructors) can still exist even if the whole (the department) does not exist.*
  - *The same part-object can belong to more than one object at a time.*



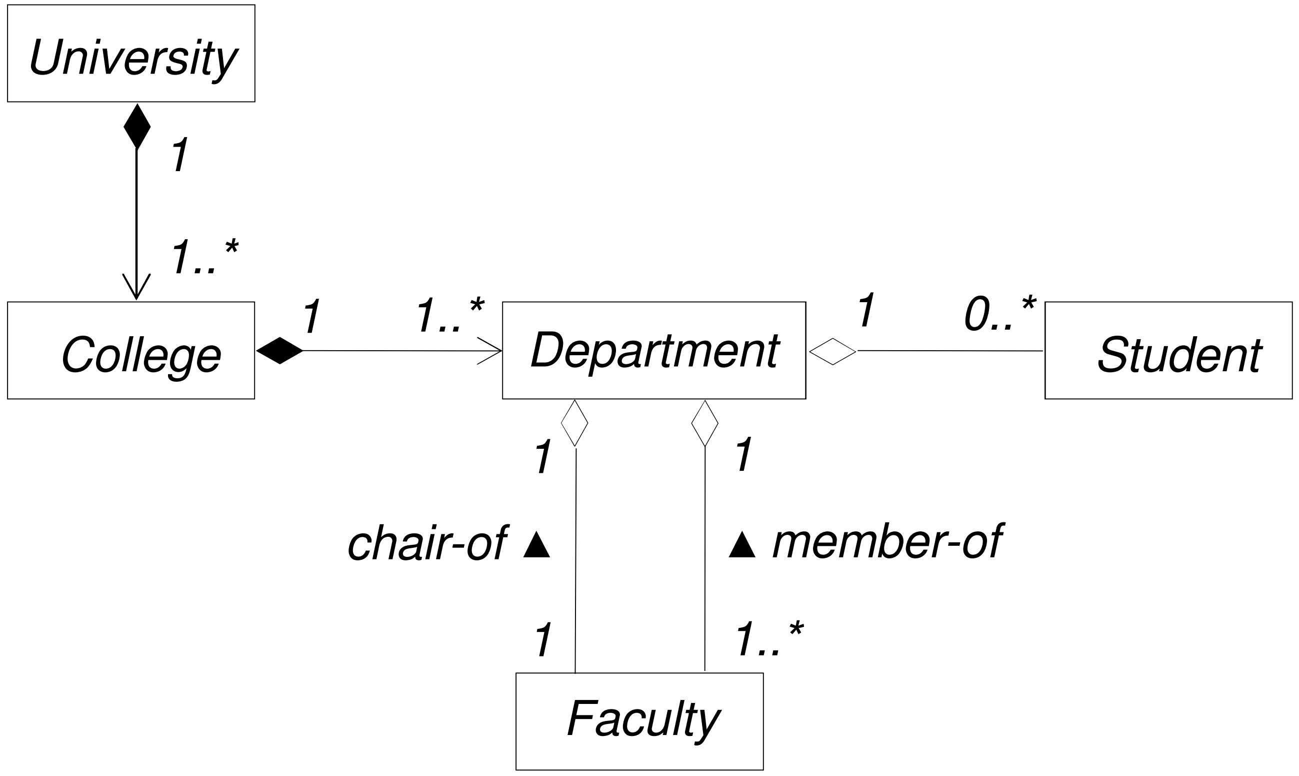
Class Diagrams (Relationship)

- Composition: is a strong kind of aggregation where the parts cannot exist independently of the "whole" object.
  - *Examples: A human has a head. A car has an engine.*
- A composition relation implies that:
  - *An instance of the part belongs to only one composite objects.*
  - *An instance of the part must belong to one composite object. It can not exist without the whole-object.*
  - *The composite is responsible for the creation and deletion of its part. If the composite is destroyed its parts must either be destroyed.*



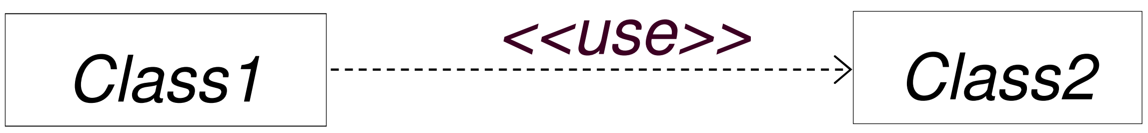
Class Diagrams (Relationship)

- Example:



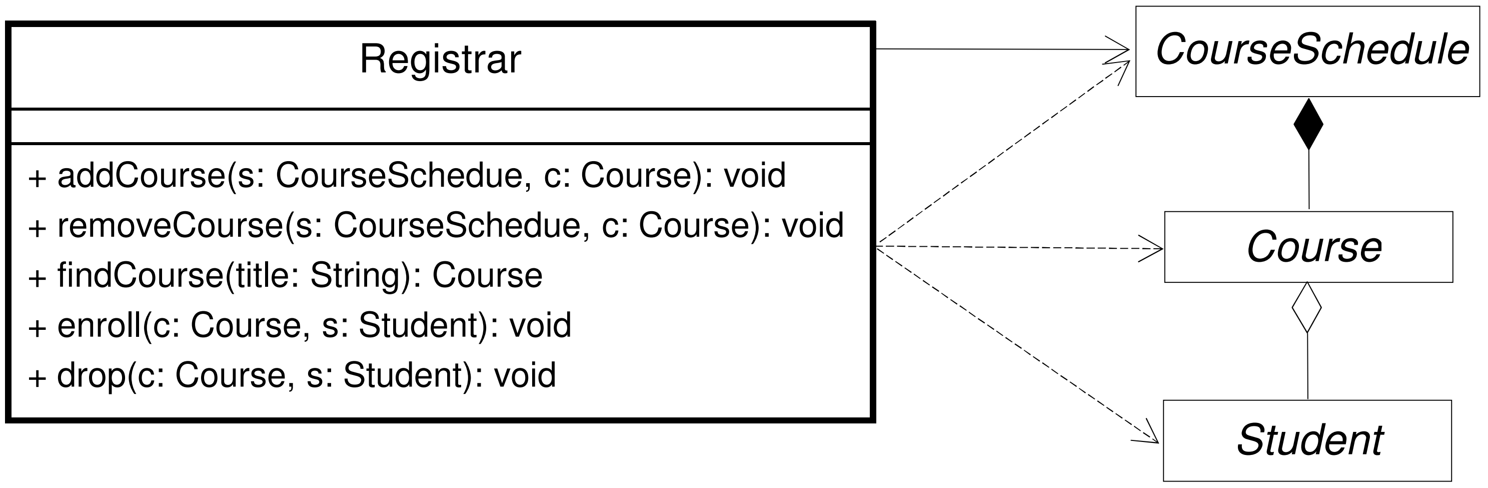
Class Diagrams (Relationship)

- Dependency: Relationship between the entities such that the proper operation of one entity depends on the presence of the other entity, and changes in one entity would affect the other entity.
- The common form of dependency is the *use* relation among classes.



Class Diagrams (Relationship)

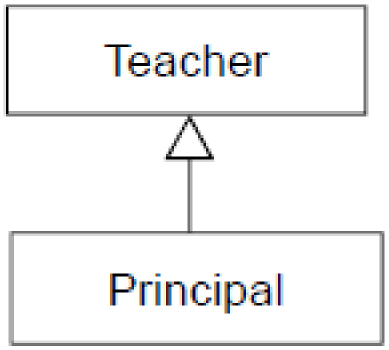
- Example


Registrar
+ addCourse(s: CourseSchedue, c: Course): void
+ removeCourse(s: CourseSchedue, c: Course): void
+ findCourse(title: String): Course
+ enroll(c: Course, s: Student): void
+ drop(c: Course, s: Student): void
CourseSchedule
Course
Student

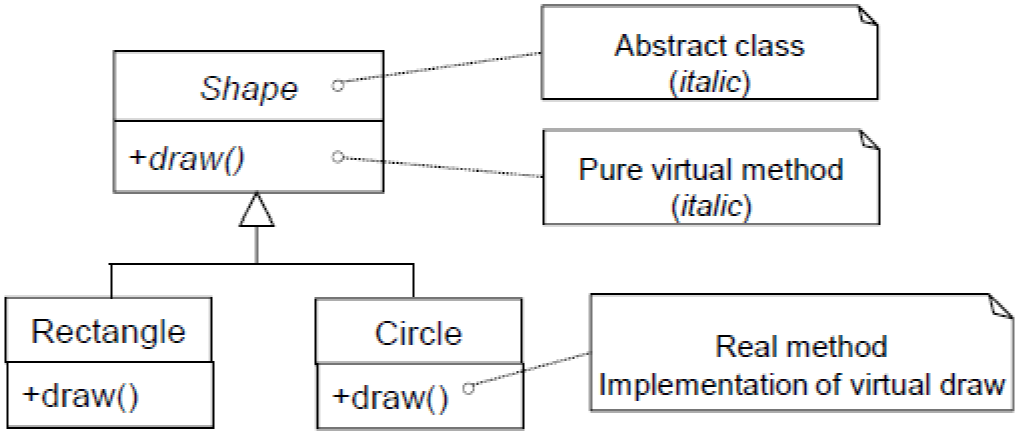
Dependencies are most often omitted from the diagram unless they convey some significant information.

Class Diagrams (Relationship)

- Inheritance: The white triangular arrow point towards the class being extended.
- The arrow should point upwards. This is not a rule of UML, but it feels more logical and easier to read in this form.

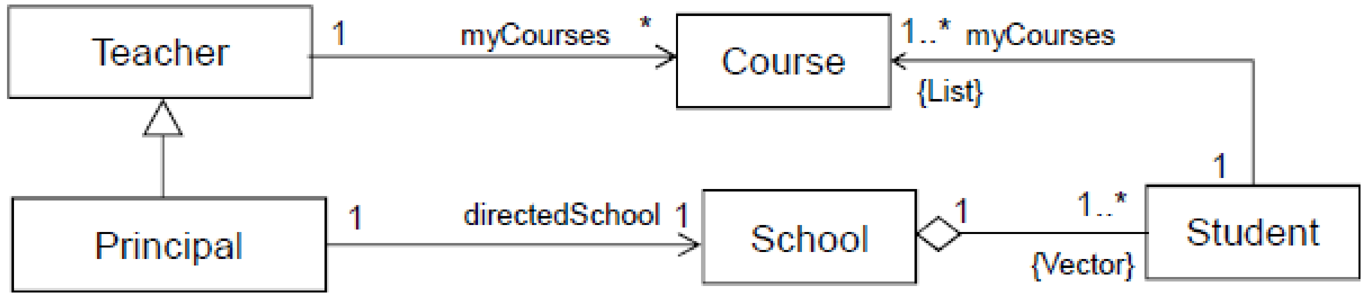


- Abstract classes and pure virtual (abstract) methods are written with italic fonts.


Shape
+draw()
Abstract class (*italic*)
Pure virtual method (*italic*)
Rectangle
+draw()
Circle
+draw()
Real method Implementation of virtual draw

Class Diagrams (Relationship)

- Example: Partial class diagram of a part of a system.

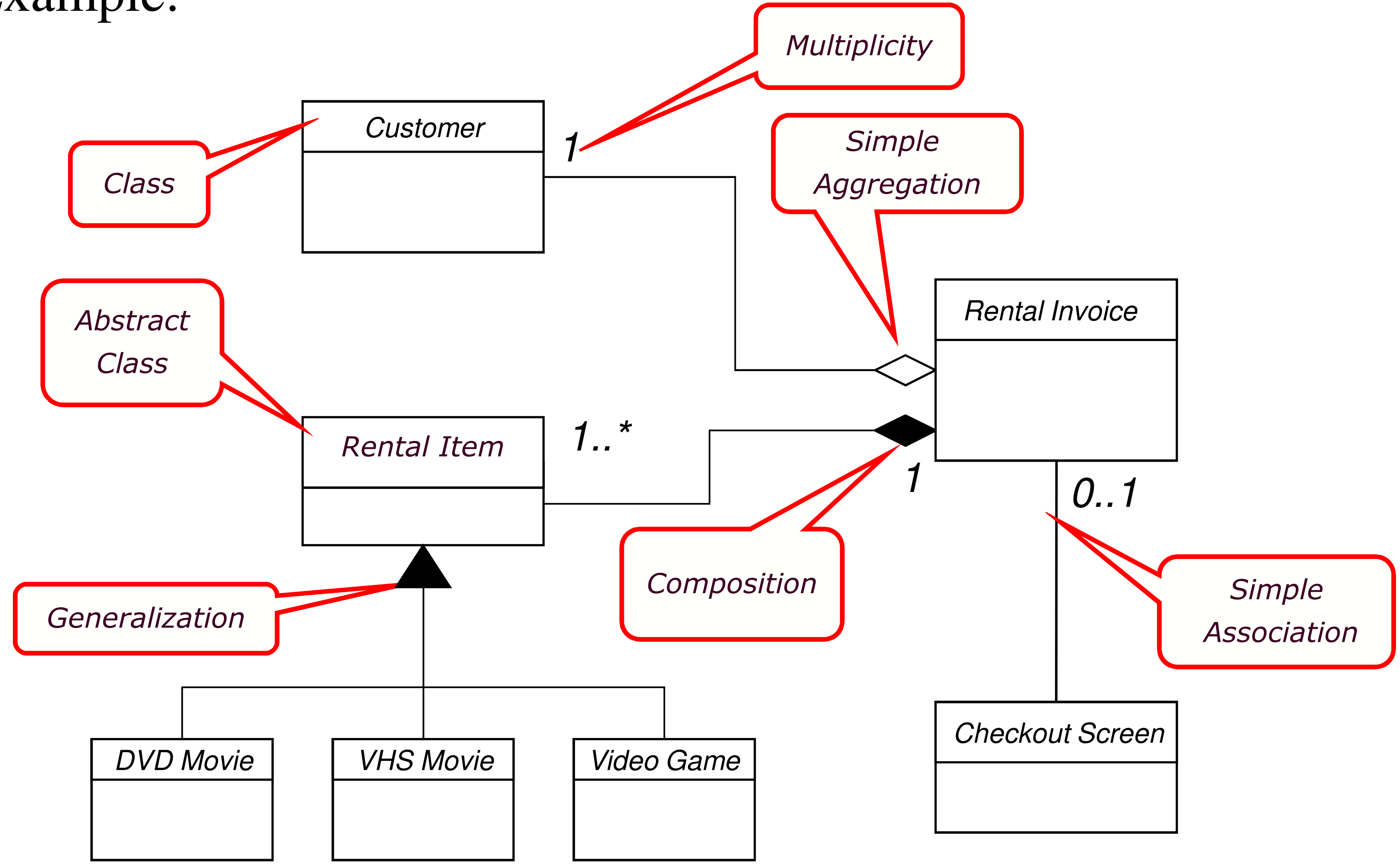


```
Class Teacher {
    private:
      Course * myCourses; //may be a linked list
    :
};

class Principal:public Teacher{
  private:
     School directedSchool;
     // or
     School *directedSchool;
};
```

```
Class School {
    private:
        vector<Student*> students;
      :
};

class Student{
  private:
      list<Course*> myCourses;
       :
};
```

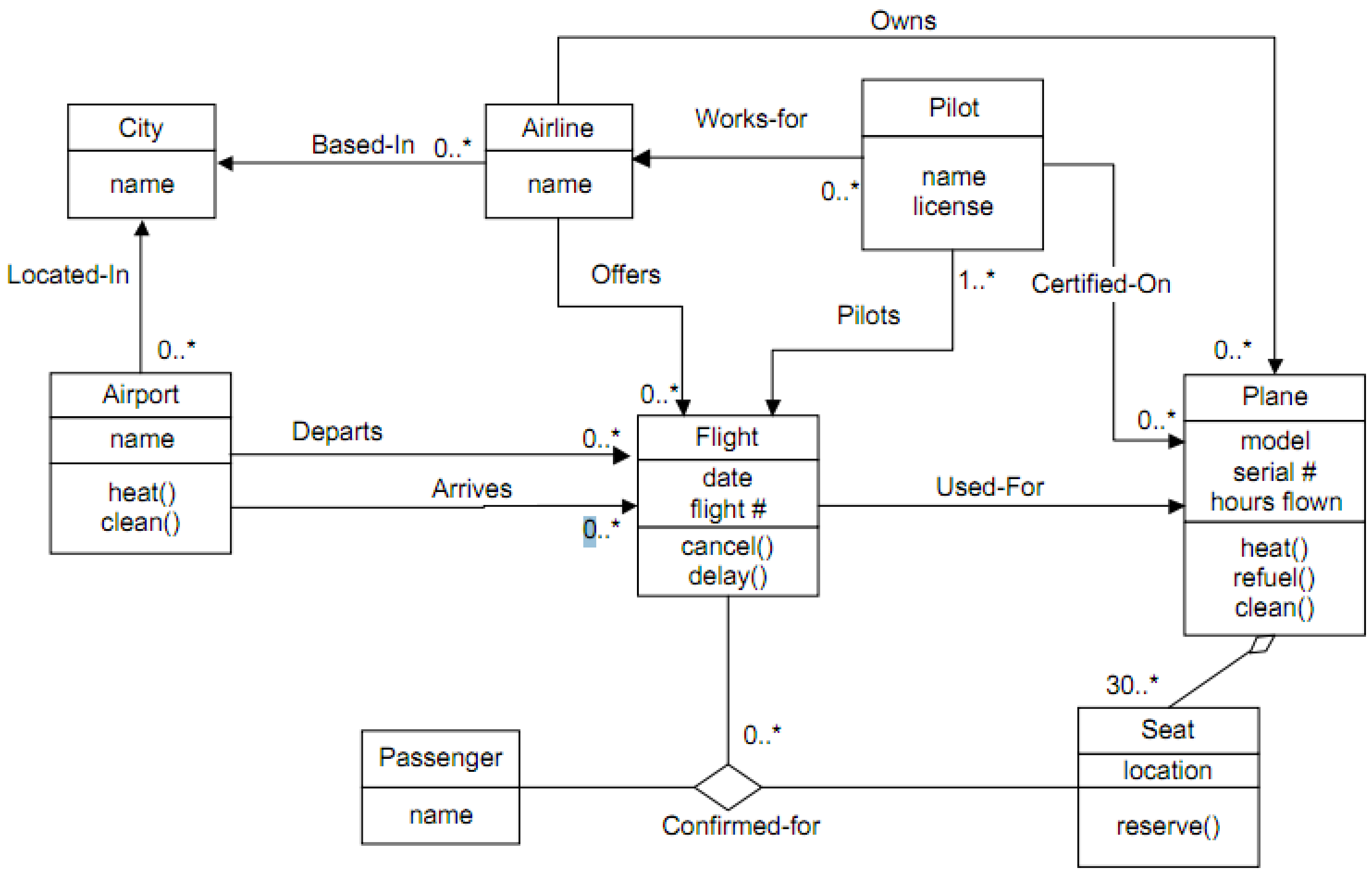
Class Diagrams (Relationship)

- Example:



Class Diagrams (Relationship)

- Example:


Owns
City
name
Based-In
0..*
Airline
name
Works-for
Pilot
name
license
0..*
Located-In
Offers
1..*
Certified-On
Pilots
0..*
0..*
Airport
name
heat()
clean()
0..*
Departs
0..*
Flight
date
flight #
cancel()
delay()
0..*
Plane
model
serial #
hours flown
heat()
refuel()
clean()
Arrives
0..*
Used-For
30..*
Seat
location
reserve()
0..*
Passenger
name
Confirmed-for