More on Designing with UML
UML2 Diagram Types

- Diagram
  - Structure Diagram
    - Class Diagram
      - Composite Structure Diagram
    - Component Diagram
    - Object Diagram
      - Deployment Diagram
      - Package Diagram
  - Behaviour Diagram
    - Activity Diagram
    - Use Case Diagram
    - State Machine Diagram
      - Interaction Diagram
        - Sequence Diagram
        - Interaction Overview Diagram
        - Communication Diagram
        - Timing Diagram
Review:
Classes are shown by boxes

Student

name

Course

*Classes*, not actual objects

(Objects can also be shown by boxes;
For objects, names are always *underlined.*)
Review:
Attributes may be listed

<table>
<thead>
<tr>
<th>Student</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Date of birth</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course Offering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Instructor</td>
</tr>
</tbody>
</table>
Operations (methods) may be listed

<table>
<thead>
<tr>
<th>Student</th>
<th>Course Offering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Number</td>
</tr>
<tr>
<td>Address</td>
<td>Title</td>
</tr>
<tr>
<td>Date of birth</td>
<td>Instructor</td>
</tr>
<tr>
<td>takeCourse</td>
<td>enroll</td>
</tr>
<tr>
<td>graduate</td>
<td>drop</td>
</tr>
<tr>
<td>operations</td>
<td>assignGrade</td>
</tr>
</tbody>
</table>

*Operations* (more detail can be given, such as argument and result types, and visibility)
Review:
*Associations are shown by lines*

```
<table>
<thead>
<tr>
<th>Student</th>
<th>Course offering</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
```

*association*

*Generally this means that there are 0 or more pairings of students with course offerings.*
Many Possible Implementations of Associations

- Recall implementations of undirected graphs:
  - List of pairs
  - Arrays or list of references (or pointers) to other objects
  - Fixed reference or pointer variables
  - Implied associations
Directionality of Associations

- By default, associations allow "bi-directional" navigation:
  From an object in either class, one can get to the associated objects in the other class.

- Adding an open arrow-head restricts navigation to be one-way, in the direction of the arrow.
Here a Course Offering knows about its Textbooks but not vice-versa.

This is sometimes called a “navigation arrow”.

If absent, then navigation is assumed to be bi-directional.
Directionality

- Directionality is a “design detail” that need not be of concern in initial passes of the design.

- It will impact the choice of implementation techniques and performance.
Review: Ordered Reading of Association Names

Arrowhead shows direction of reading the name of the association,
e.g. “A Course Offering uses a Textbook”. 
Ordered vs. Directional

- Ordered involves the **reading** interpretation of the association only.

- Directional determines the navigability.

- The two are totally *independent*. 
Review: Associations may have a multiplicity

Multiplicity: says that each Time Interval has two Times (such as a start time and an end time).
Association multiplicities

- The default multiplicity is 1.
- m..n means m through n (m and n fixed numbers).
- m..* means m or more.
- * means the same as 0..* (0 or more).
- a, b, c, ... means one of a, b, c ...
- 0,1 or 0..1 is a way of saying optional.
Note on Multiplicities

- Multiplicity should be the one that you wish the **software application** to address, rather than what *might* be the case in nature.

- For example, a major of a given *name* may exist in several colleges, suggesting *:* association.

- However, *:1* association might be wanted (one college has multiple majors), but a given major belongs to a college.
Roles in Associations

Roles go with the object, not the subject.

Roles: indicate what role a Time plays with respect to Time Interval.

(Since this is a class diagram and not an object diagram, it is not implied that start and end are the same Time.)
Roles in Associations

Here both associations have role names on their respective ends.

Roles are also called “Association Ends”.

<table>
<thead>
<tr>
<th>Time</th>
<th>Time Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
Corresponding Object Diagrams

: Time
start

Name of object’s class

: Time Interval
end
Object Diagrams with Objects Named

Start1 : Time
start

End1 : Time
end

Interval1: Time Interval
Name of object
Objects and Class in One Diagram

Shape

«instance of»

myShape : Shape

Class

Object
In addition to object diagrams, the object notation is used in:
- collaboration diagrams
- sequence diagrams
- and others

which will be described later.
We may wish to emphasize that an association may *itself* take the form of an object relating two or more other objects together.

Here an Enrollment is an association object relating a Student and a Course Offering.
Multi-Way Association Classes

Associations classes aren’t limited to 2-way.

![Diagram showing a 3-way association class for marriage with entities Man, Woman, and Authority. The groom and bride are connected to the marriage, and an agent is connected to the marriage.]
Aggregation and Composition

- These are both specialized forms of Association.

- They suggest whole/part relationships.

- They add certain kinds of constraints.
An aggregation is a special form of association in which a collection of objects, each having an independent existence, is associated with a single object.

Unfilled diamond means "aggregation": components exist independent of container.
An object can be in multiple distinct aggregations.
Composition

Filled diamond means "composition": components are inseparable, non-shareable, part of container.

The container is composed of the components (and possibly others).

In some sense, the container "controls" the components.

Multiplicity 1 is thus implied.
Question

- Can an object be in an aggregation and a composition simultaneously?

- Is it advisable to do this?
Possible C++ comparison

- **Aggregation**

```cpp
class College {
    list<Student*> students;

public:
    void addStudent(Student* s) {
        students.add(s);
    }
    ...
};
```

- **Composition**

```cpp
class College {
    list<Building*> buildings;

public:
    void buildBuilding(string name) {
        buildings.add(new Building[n]);
    }
    ...
};
```

Students exist outside of the college.

Construct inside; assuming buildings don’t exist outside of the college.
C++ Destruction Note

- With **composition**, contained objects are always created and known only “on the inside”.

- With **aggregation**, aggregate objects are created and destroyed independent of the aggregating object.
Exercise: Identify Likely Aggregations and Compositions
Qualified Association

An attribute indicating how to locate the associated object.

College

Student number

1

Student

0..1
Comparison:
Qualified vs. Unqualified Association

without Qualified Association

<table>
<thead>
<tr>
<th>College</th>
<th>1</th>
<th>Student</th>
</tr>
</thead>
</table>

| Student number | 0..1 |

with Qualified Association

<table>
<thead>
<tr>
<th>College</th>
<th>Student</th>
</tr>
</thead>
</table>

| Student number | 1 | 0..1 |
Exercise: Identify Opportunities for Qualified Association

- Room
- Instructor
- Course Offering
- Time Slot
- Meeting
- Textbook
- Department
- Semester
- Course
- Enrollment
- Author
- College
- Major
- Building
- Student
Inheritance and Generalization
Inheritance/Generalization

In this form of inheritance, a member of the derived class **is-a** member of the base class, as far as behavior is concerned.
Usually there will be multiple derived classes if there is any.
This notation is equivalent to that on the preceding slide.
"Multiple Inheritance" is possible, although should be avoided since not all implementation languages support it well.
"Interface Inheritance" alternative

In Java, `Interface` is a part of the language definition; in C++ it is a matter of interpretation.

Dashed line denotes Interface inheritance.

- «interface» Administrator
- Faculty Member
- Department Chair
Alternative Notation for Interface

- Faculty Member
- Department Chair
- Interface Symbol
- «interface» Administrator
Recursive Structure

Use inheritance to articulate recursive structures.

"An item can be either an atomic item or a container. A container contains 0 or more items."

OK (One arrow is aggregation, the other inheritance.)
Corresponding Object Diagram

Objects
(all are Items)

:Container

:Container

:Container

:Atomic Item

:Atomic Item

:Atomic Item