Relationships Between Objects

· In the real world, there are relationships between objects

Examples:

- Students enroll in courses
- o Classes have classrooms
- Professors have a list that contains the courses they offer.
- The university consists of faculties, and faculties consist of departments.
- The dean of the faculty is a professor.
- o A Ph.D. student is a kind of student.
- · The objects can cooperate (interact with each other) to perform a specific task, Examples
 - A professor can get the list of the students from the course object.

 - A student can get her grades from the related course objects.

 A university can send an announcement to all faculties, and faculties can distribute this announcement to their departments.

1999 - 2025 Feza BUZLUCA

Relationships Between Objects (cont'd)

- In object-oriented design (OOD), we try to lower the representational gap between real-world objects and the software components.
- This makes it easier to understand what the code is doing.
- · To represent real-world relationships, we also create relationships between software objects.

Types of relationships in object-oriented design (OOD):

- There are two general types of relationships, i.e., association and inheritance.
- Association is also called a "has-a" ("uses") relationship.
 Inheritance is known as an "is-a" relationship.

- o A course has a classroom,
- The dean of the faculty is a professor.
- In this section, we will cover association, aggregation, and composition.

 While association itself is a general "uses-a" relationship, its subtypes, **aggregation** and **composition** are forms of the has-a relationship.
- Inheritance ("is-a" relationship) will be covered in the coming sections

@ 0 0 0 1999 - 2025 Feza BUZLUCA

Association means instances of class A ${\it can}$ use services given by class B .

- Instances of A know instances of B.
- Programming: Class A has pointers (or references) to objects of class B.
- The relationship may be unidirectional or bidirectional (where the two objects are aware of each other). If the relationship is bidirectional, class B also has pointers (or references) to objects of class A
- Instances of A and B can communicate with each other.
- Instances of class A can send messages to instances of another class $\boldsymbol{B}_{\!\scriptscriptstyle L}$
- Programming: Objects of class A can call methods of objects of class B. There may be one-to-one, one-to-many, or many-to-many associations between objects
- The objects that are part of the association relationship can be created and destroyed independently, Each of these objects has its own life cycle.
- Programming: The constructor of a class does not have to call the constructor of the other class, The destructor of a class does not have to call the destructor of the other class
- There is no "owner".

@ ● ● 1999 - 2025 Feza BUZLUCA

6.3

Association (cont'd):

Students register for courses.

Real World:

- A student can enroll in multiple courses.
- A course can have multiple students enrolled in it, and students can enroll in several courses (bidirections
- A student is associated with multiple courses. At the same time, one course is associated with multiple students (many-to-many).
- Students can get their grades from the course
- Courses also can access some information about students, such as their IDs.
- Each of these objects has its own life cycle.

The department can create new courses. In this case, new students are not created. When a course is removed from the department's plan, the students are not destroyed. Students can add or drop courses.

(a) 0 (b) 1999 - 2025 Feza BUZLUCA

6.4

Example (cont'd): Students register for courses

- The Student class can have a collection (e.g., array, list) of Course objects.
 A Course class can also have a collection of the Student objects enrolled in that course (bidirectional).
- A Student object can call methods of course classes, for example, to get the grade.

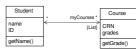
 If there is a bidirectional relation, the Course class can also call the methods of the Student class.
- Each of these objects has its own life cycle. The Student class does not have to create or destroy Course objects
- The Course class does not have to create or destroy Student objects

(C) 1999 - 2025 Feza BUZLUCA

Example: Association between students and courses

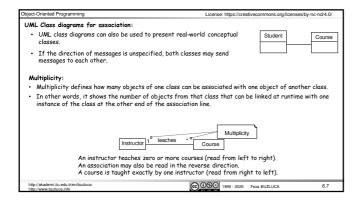
UML Notation:

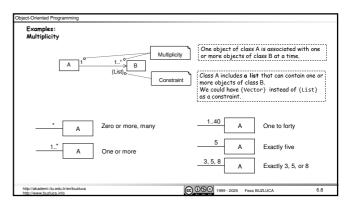
Software class diagram

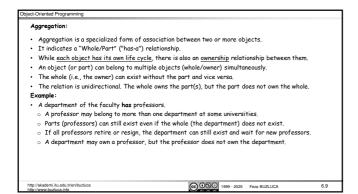


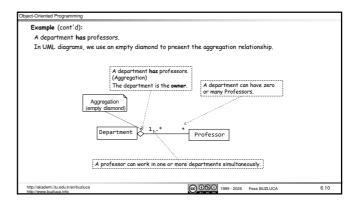
An association is a weak "uses-a" relationship between two or more objects in which the objects have their own lifetimes, and **there is no owner**.

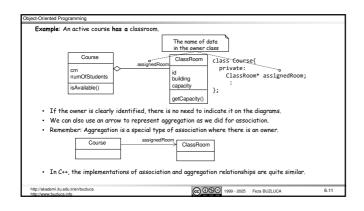
@ 000 1999 - 2025 Feza BUZLUCA











```
Composition:

The Composition is also a specialized form of association and a specialized form of aggregation. Composition is a strong kind of "has-a" relationship.

It is also called a "part-of" or "belongs-to" relationship.

There is an owner.

The objects' lifecycles are tied.

The part object (e.g., room) cannot exist without the owner/whole (e.g., house).

The whole and part objects are created together.

Constructors in C++ will ensure the creation of the parts when the owner is created.

When the owner object is deleted, the part objects are also deleted.

The relation is unidirectional.

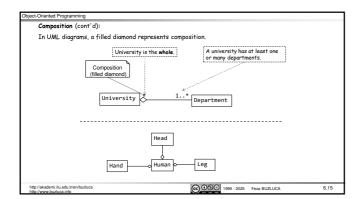
Examples:

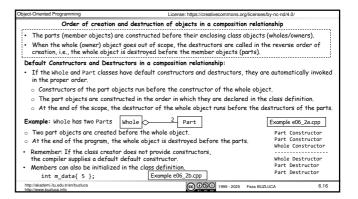
A university is composed of departments, or departments are parts of a university.

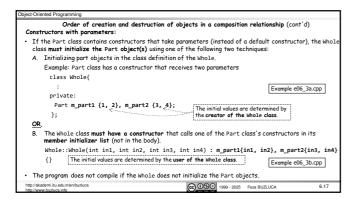
A rectangle is composed of four points.
```

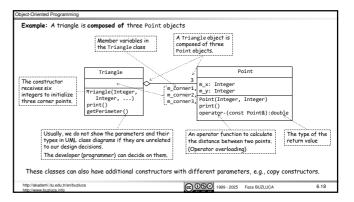
@ 000 1999 - 2025 Feza BUZLUCA

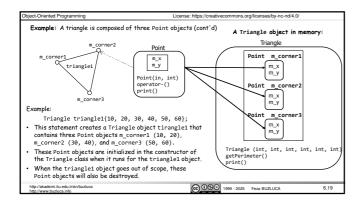
6 14











```
Example: A triangle is composed of three Point objects (cont'd)

The creator of the Triangle class calls the constructors of the Point class to initialize Point objects

The constructor of the Triangle class must call one of these constructors in the member initializer list
(not in the body).

Constructor of Triangle with the coordinates of three corners

Triangle::Triangle(int corner1_x, int corner1_y, int corner2_x, int corner3_y)

: m_corner1_x (corner1_x, corner1_y), m_corner2_x, corner2_y),

m_corner1_x (corner1_x, corner1_y), m_corner2_x (corner2_x, corner2_y),

The constructor of the Point is called three images.

This constructor takes the x and x coardinates of three corner points (six integers) and calls the constructor of the Point class three times; once for each corner point.

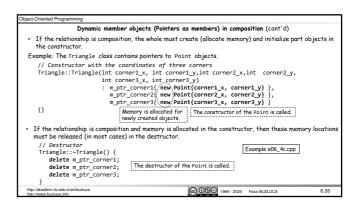
Int main() {

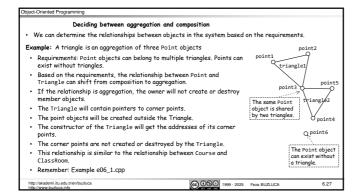
Triangle triangle1[10, 20, 30, 40, 50, 60]; // The points are created before the triangle riangle1 object are destroyed automatically.

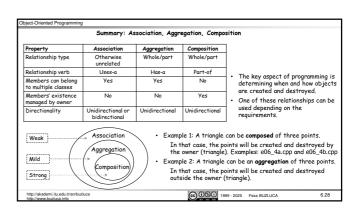
The triangle is destroyed before the points

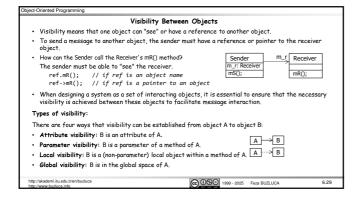
When triangle1 object are destroyed automatically.

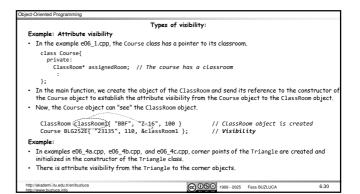
The points are created before the points of the points are corner1, m_corner2, and m_corner3) and the triangle1 object are destroyed automatically.
```

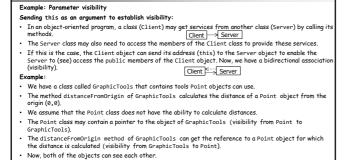












@ 000 1999 - 2025 Feza BUZLUCA

