

CISC 3115

# Relationships of Classes: Part I

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

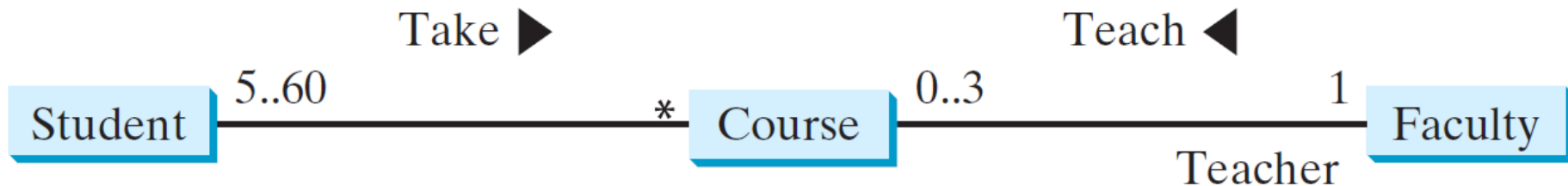
- Discussed
  - Concepts of two programming paradigms
    - Procedural and Object-Oriented
  - Design classes for problem solving
    - Think in terms of class
- Discover relationship of classes
  - Association
  - Aggregation
  - Composition (to be revisited in Chapter 13)
  - Inheritance (to be discussed in Chapter 11)

# Relationship of Classes

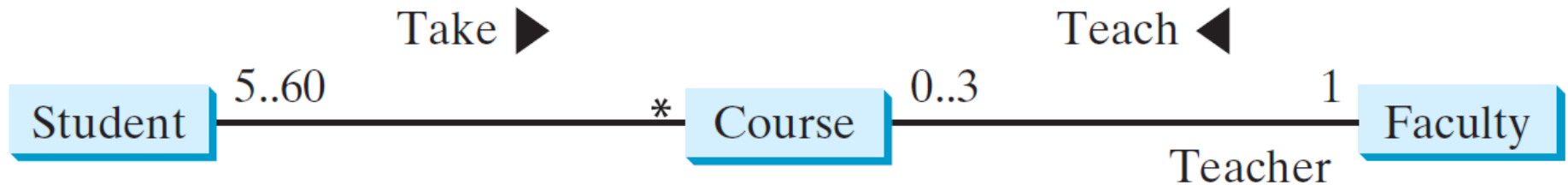
- To analyze the problem and design classes, we need to explore the relationships among classes (and objects of the classes).
  - Association
  - Aggregation
  - Composition (to be revisited in Chapter 13)
  - Inheritance (to be discussed in Chapter 11)

# Association

- A general binary relationship that describes an activity between two classes
- UML diagram
  - Consider 3 classes, Student, Course, and Faculty



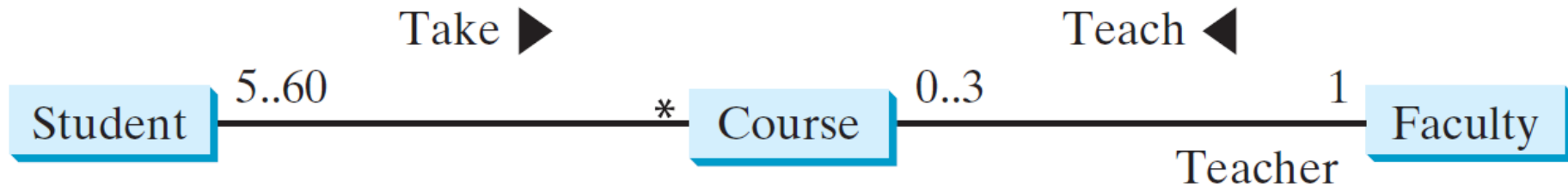
# Association: UML notation



- Role
  - Take, Teach; arrow indicates “subject” & “object” in English
- Multiplicity
  - A course has 5 ~ 60 students (5..60)
  - A student takes any number of courses (\*)
  - A faculty teaches 0 ~ 3 courses (0..3)
  - A course has 1 faculty (1)

# Class Representation: Association

- Using data fields and methods



```
public class Student {
    private Course[] courseList;
    public void addCourse(Course c) {
    }
}
```

```
public class Course {
    private Student[] studentList;
    private Faculty faculty;
    public void addStudent(Student s) {
    }
    public void setFaculty(Faculty f) {
    }
}
```

```
public class Faculty {
    private Course[] courseList;
    public void addCourse(Course c) {
    }
}
```

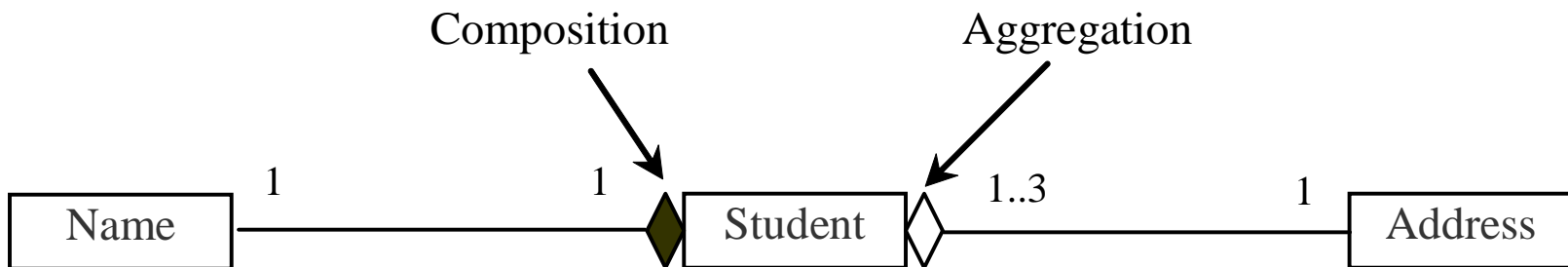
# Aggregation

- A special form of association that represents an ownership relationship between two objects
  - It models a has-a relationship
  - Owner object/class: aggregating object/class
  - Subject object/class: aggregated object/class
- UML diagram
  - Consider 2 classes, Student and Address



# Composition

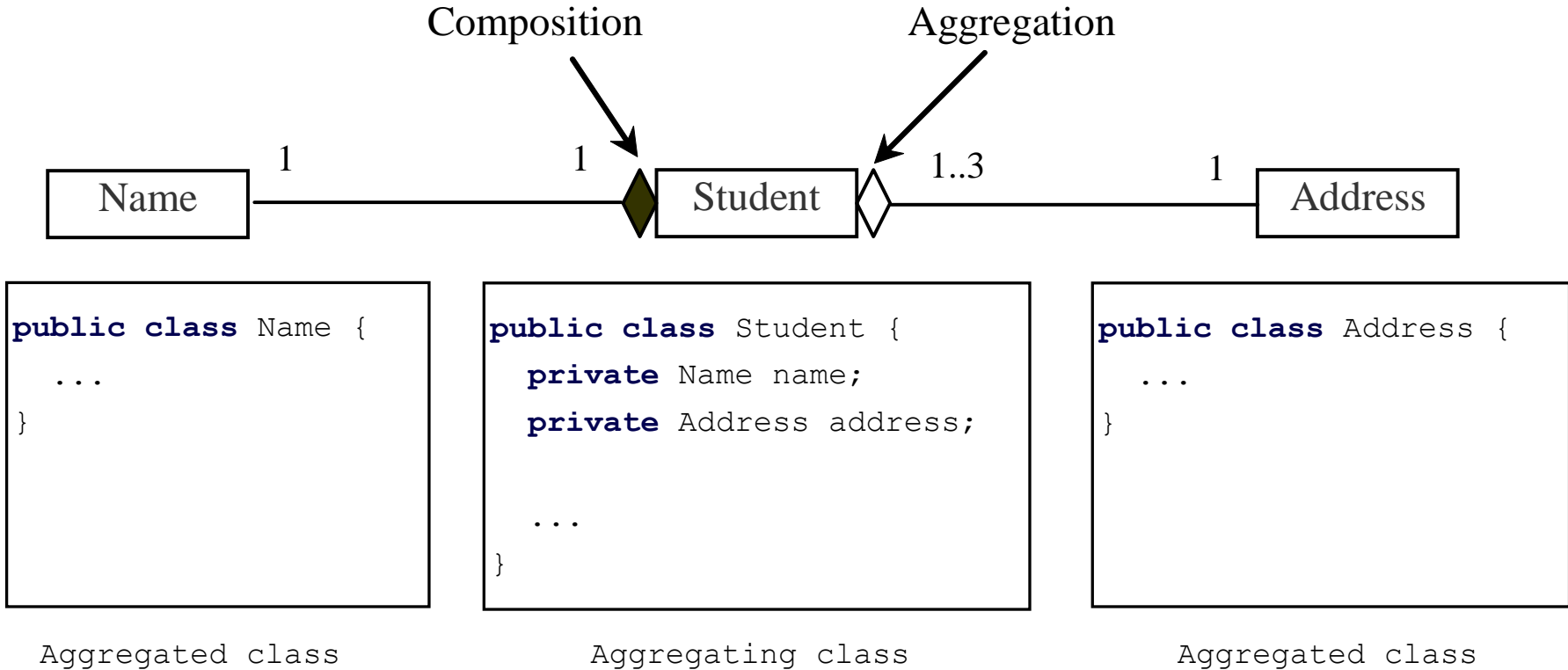
- A special case of the aggregation relationship where the existence of the aggregated object is dependent on the aggregating object (i.e., aggregated object does not exist by itself)
- UML diagram
  - Consider 3 classes, Name, Student, and Address





# Class Representation: Aggregation and Composition

- An aggregation relationship is usually represented as a data field in the aggregating class.



# Aggregation or Composition

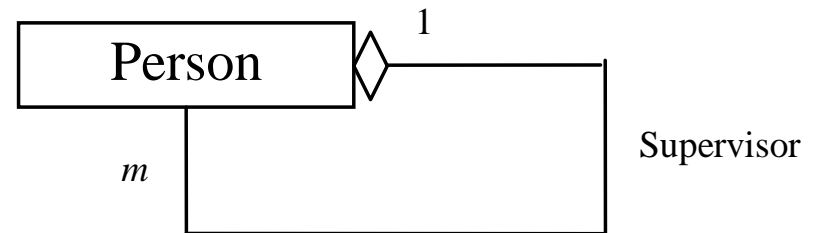
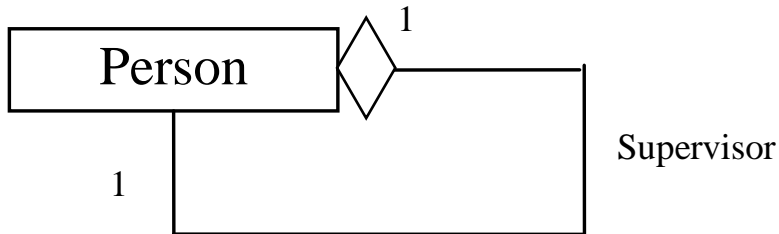
- Aggregation and composition relationships are represented using classes in similar ways, many texts do not differentiate them and call both compositions.

# Aggregation Between Same Class

- Aggregation may exist between objects of the same class.
- Example
  - A person may have a supervisor who is also a person.

# Self-Aggregation: UML Diagram and Class Representation

- UML diagram



- Class representation

```
public class Person {  
    // The type for the data is the class itself  
    private Person supervisor;  
    ...  
}
```

```
public class Person {  
    // The type for the data is the class itself  
    private Person[] supervisors;  
    ...  
}
```

# Example: The Course Class

Course
<pre>-courseName: String -students: String[] -numberOfStudents: int</pre>
<pre>+Course(courseName: String) +getCourseName(): String +addStudent(student: String): void +dropStudent(student: String): void +getStudents(): String[] +getNumberOfStudents(): int</pre>

The name of the course.

An array to store the students for the course.

The number of students (default: 0).

Creates a course with the specified name.

Returns the course name.

Adds a new student to the course.

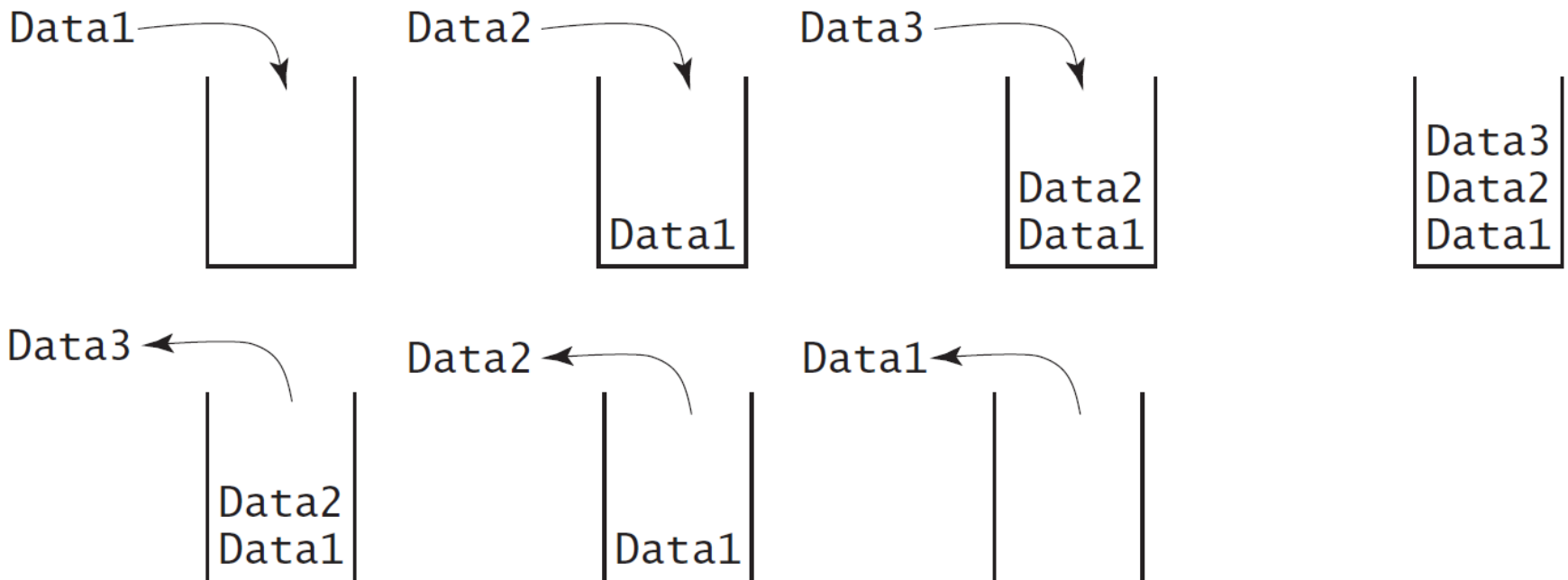
Drops a student from the course.

Returns the students in the course.

Returns the number of students in the course.

# Example: Designing The StackOfInteger Class

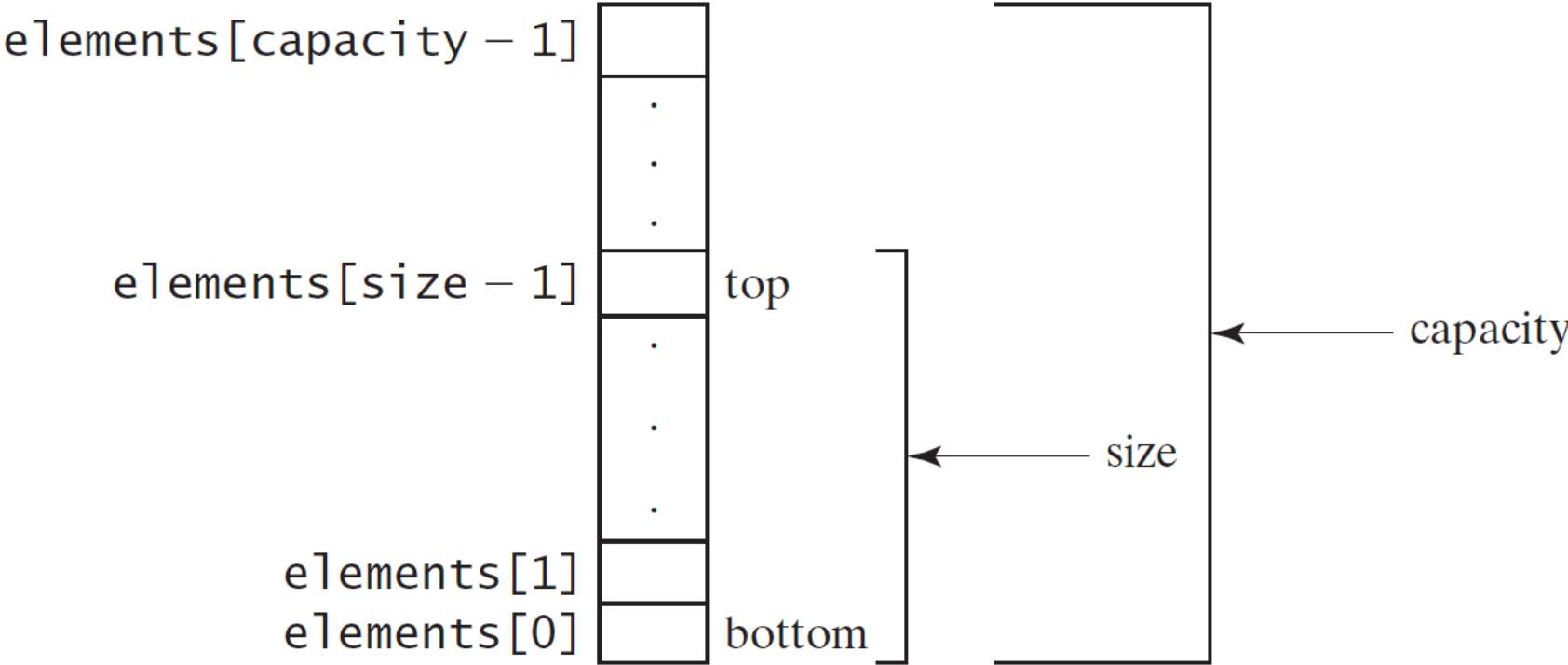
- A stack is a data structure that holds data in a last-in, first-out fashion



# Example: The StackOfInteger Class

StackOfIntegers	
-elements: int[]	An array to store integers in the stack.
-size: int	The number of integers in the stack.
+StackOfIntegers()	Constructs an empty stack with a default capacity of 16.
+StackOfIntegers(capacity: int)	Constructs an empty stack with a specified capacity.
+empty(): boolean	Returns true if the stack is empty.
+peek(): int	Returns the integer at the top of the stack without removing it from the stack.
+push(value: int): int	Stores an integer into the top of the stack.
+pop(): int	Removes the integer at the top of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.

# Example: Implementing the StackOfInteger Class





# Exercise

- Implement the StackOfInteger class.

# Questions?

- Relationship among classes
  - Association
  - Aggregation
  - Composition (to be revisited in Chapter 13)
  - Inheritance (to be discussed in Chapter 11)
  - How to represent the relationship using classes/objects?