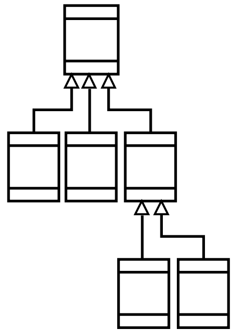
BCS2143: OBJECT ORIENTED PROGRAMMING

Chapter 4

- CLASS DIAGRAM

Modeling Notations - UML

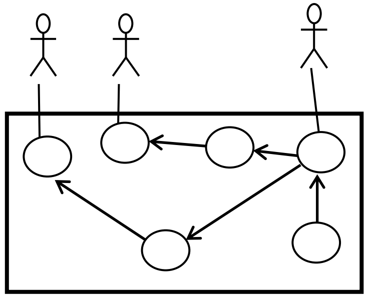


UML Class Diagrams

information structure

relationships between data items

modular structure for the system

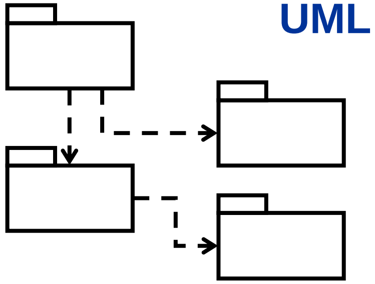


Use Cases

user's view

Lists functions

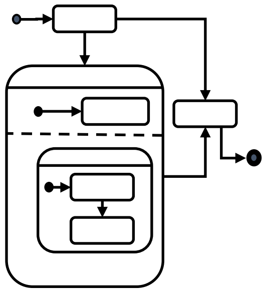
visual overview of the main requirements



UML Package Diagrams

Overall architecture

Dependencies between components

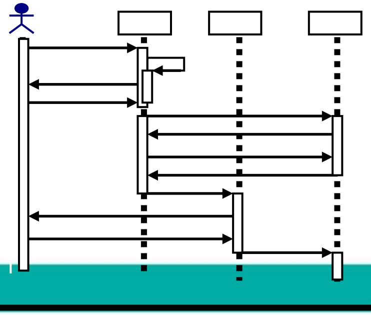


(UML) Statecharts

responses to events

dynamic behavior

event ordering, reachability, deadlock, etc

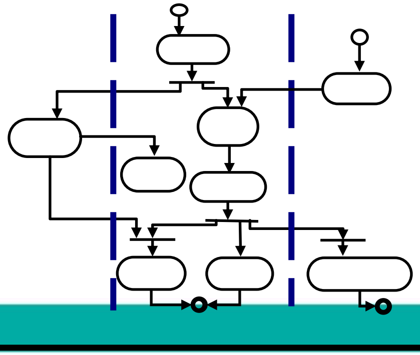


UML Sequence Diagrams

individual scenario

interactions between users and system

Sequence of messages



Activity diagrams

business processes;

concurrency and synchronization;

dependencies between tasks;

Design phase

- **Design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation (coding).
  - What **classes** will we need to implement a system that **meets our requirements**?
  - What **fields** and **methods** will each class have?
  - How will the classes **interact** with each other?

Introduction to UML & Class Diagram

- **Unified Modeling Language (UML) :** pictures of an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- **What is a UML class diagram?**
  - A picture of the **classes** in an OO system, their **fields** and **methods**, and **connections** between the classes that interact or inherit from each other.
- **What are some things that are not represented in a UML class diagram?**
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

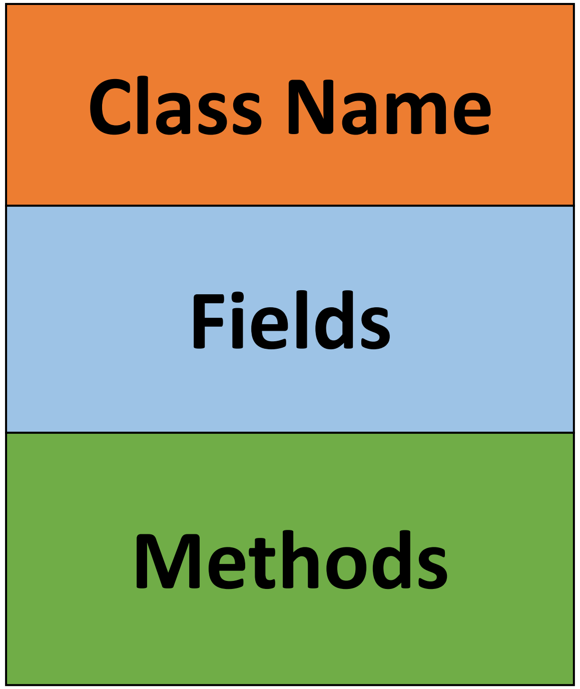
Uses for UML

- **As a sketch:** to communicate aspects of system
  - **forward design:** doing UML before coding
  - **backward design:** doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas
- **As a blueprint:** a complete design to be implemented
- **As a programming language:** with the right tools, code can be auto-generated and executed from UML
  - only good if this is faster than coding in a "real" language

What are Classes?

- A **class** describes a **group of objects** with
  - similar **properties** (fields),
  - common **behaviors** (methods),
  - common **relationships** to other objects,
  - and common **meaning** (“semantics”).

- Graphically, a class is rendered as a rectangle, usually including its **class name**, **fields**, and **methods** in separate, designated compartments.



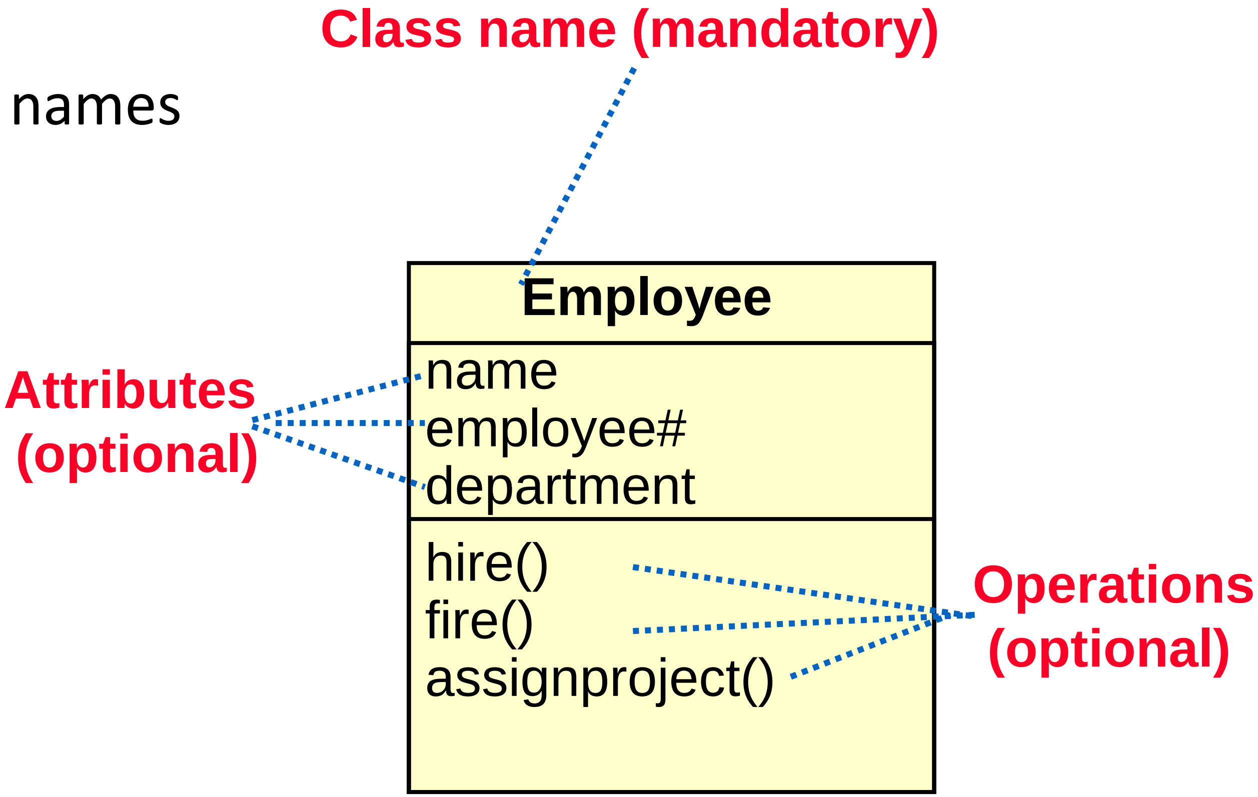
Class Name

Fields

Methods

Diagram of Class

- **Class name in top of box**
  - write **<<interface>>** on top of interfaces' names
  - use *italics* for an ***abstract class*** name
- **Attributes / fields (optional)**
  - should include all fields of the object
- **Operations / methods (optional)**
  - may omit trivial (get/set) methods
  - should not include inherited methods



Class Attributes

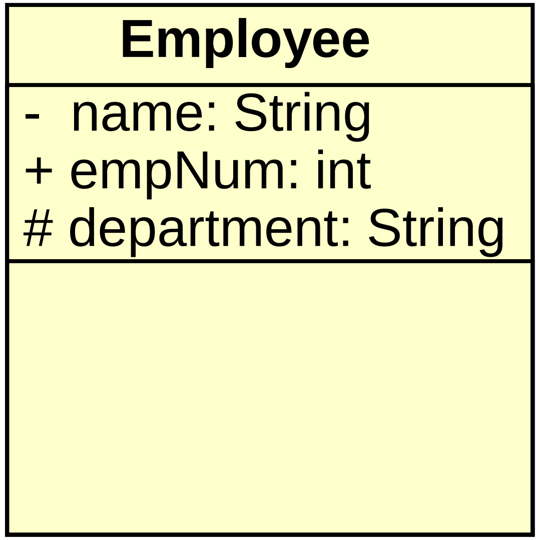
- **Attributes (fields, instance variables)**

  - **visibility:**

+	public
#	protected
-	private
~	package (default)

  - underline **static attributes**

  - example: **- balance : double = 0.00**



Employee

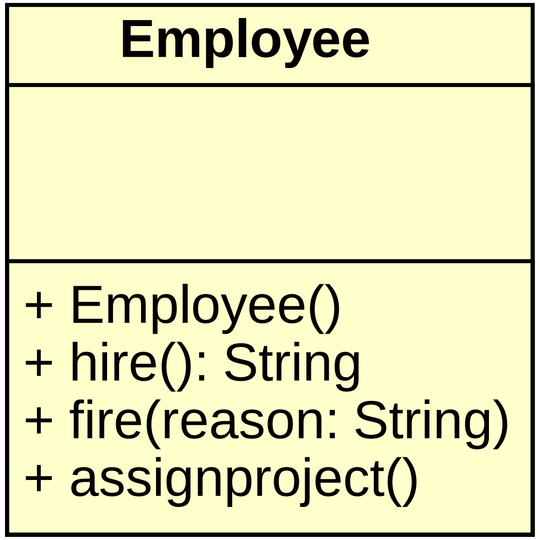
- name: String
+ empNum: int
# department: String

Class Operations

- **Operations / methods**
  - **Format** *visibility name* (*parameters*) : *return_type*
  - **visibility:**

+	public
#	protected
-	private
~	package (default)

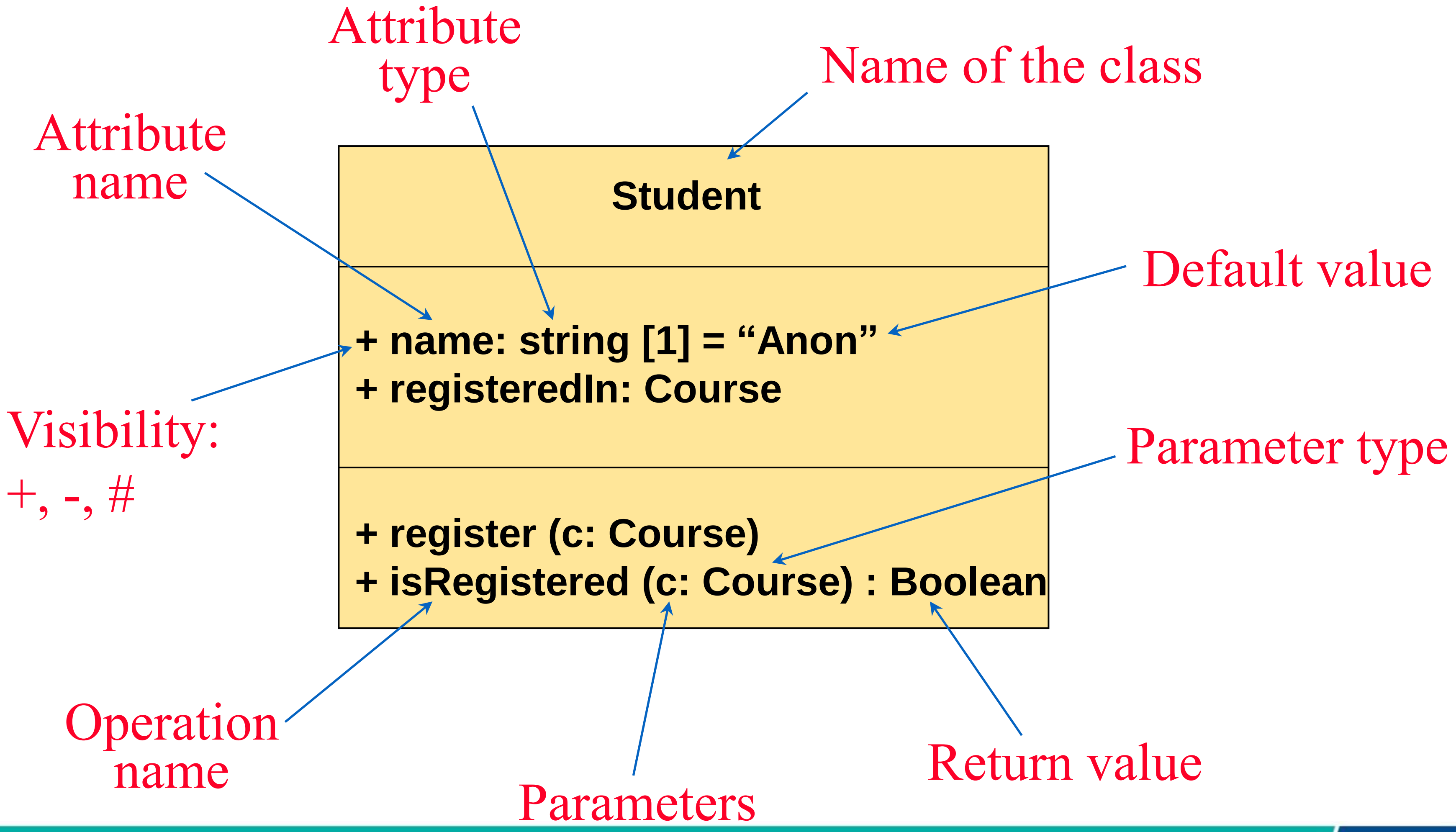
  - underline static methods
  - parameter types listed as (name: type)
  - example:
    + distance(p1: double, p2: double): double



Employee

+ Employee()
+ hire(): String
+ fire(reason: String)
+ assignproject()

Full Class Notation



Attribute type

Name of the class

Attribute name

Student

+ name: string [1] = “Anon”
+ registeredIn: Course

Default value

Visibility:
+, -, #

Parameter type

+ register (c: Course)
+ isRegistered (c: Course) : Boolean

Operation name

Parameters

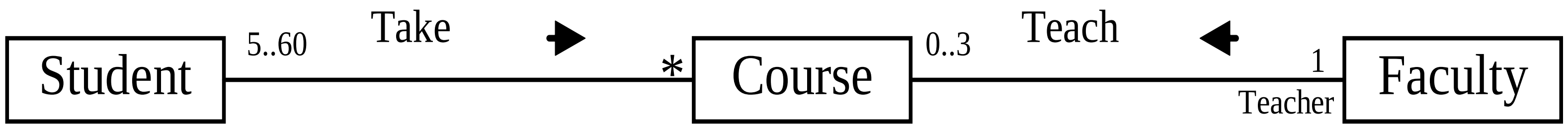
Return value

Relationships Between Classes

- Classes do **not exist in isolation** from one another
  - A **relationship** represents a connection among things.
  - E.g. Employee class is **associated** with the Person class.
- Class diagrams show classes and their relationships
- In UML, there are different types of relationships:
  - Association
  - Aggregation and Composition
  - Dependency
  - Generalization

Association

Association represents a general binary relationship that describes an activity between two classes.



```
public class Student {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

```
public class Course {
  /** Data fields */
  private Student[]
    classList;
  private Faculty faculty

  /** Constructors */
  /** Methods */
}
```

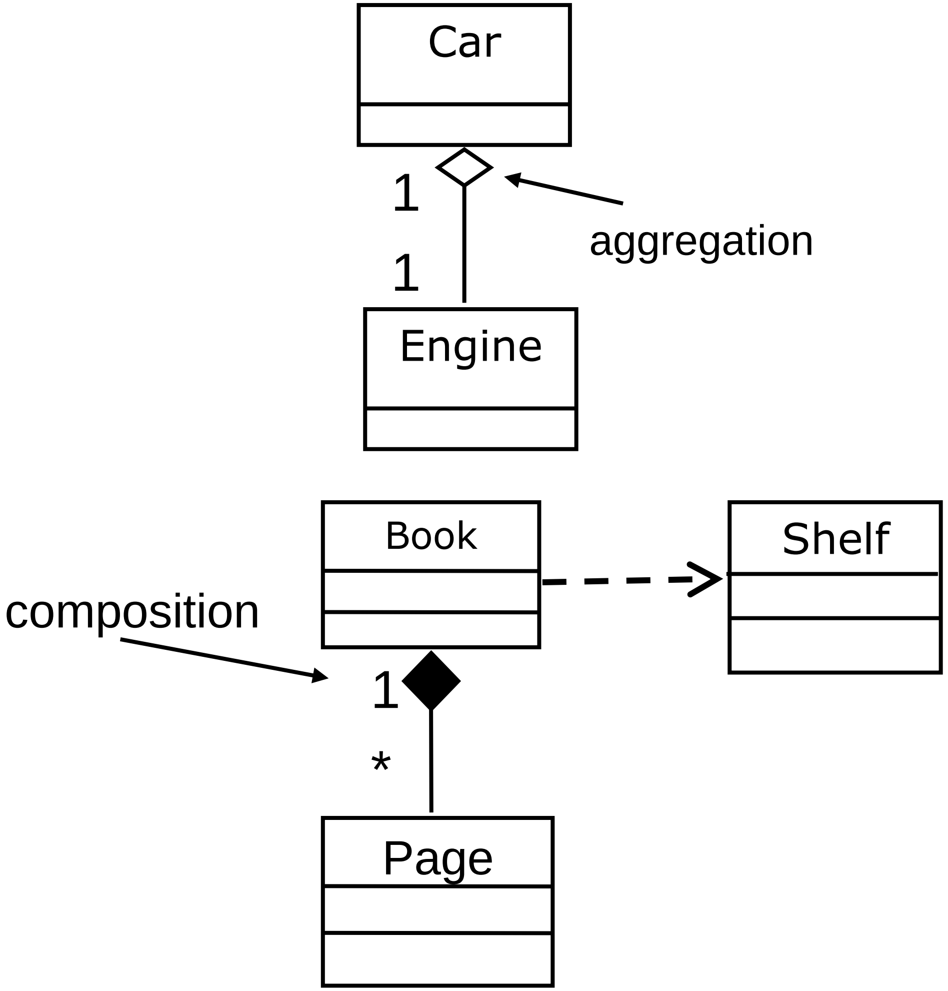
```
public class Faculty {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

An association is usually represented as a data field in the class.

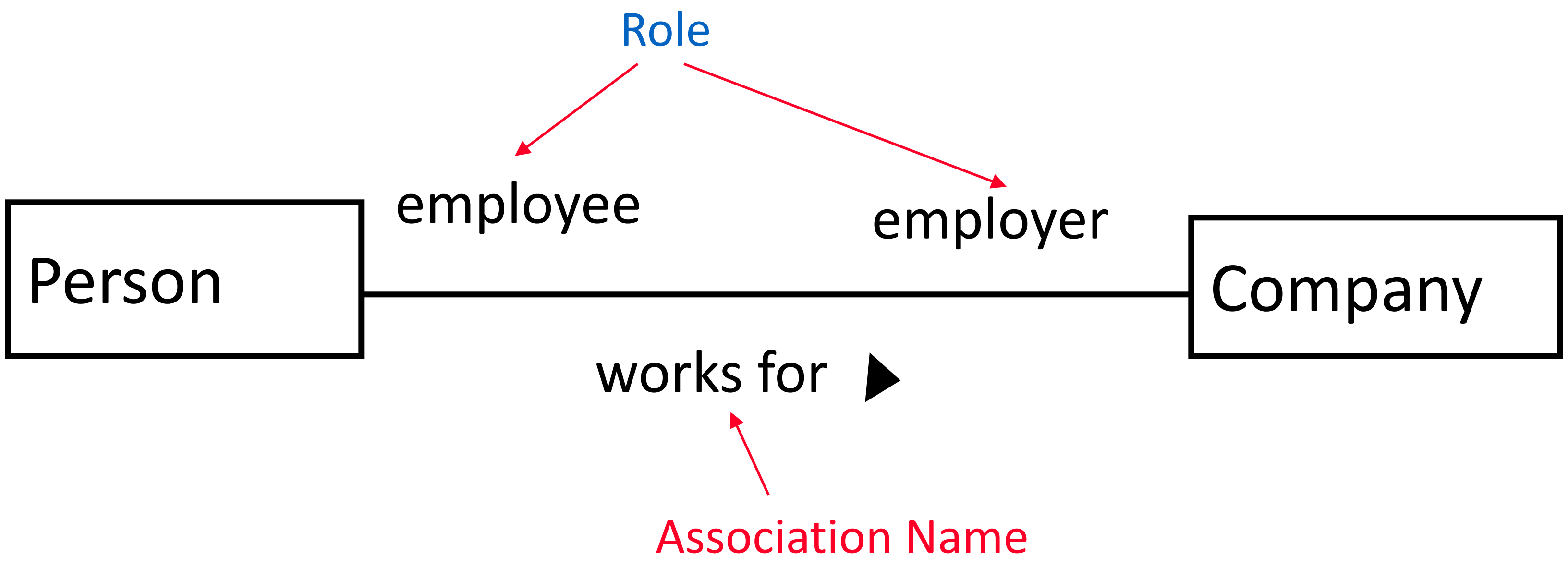
Association types

- **Aggregation**: "is part of"
  - symbolized by a clear white diamond
- **Composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond
- **Dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



Association - Example

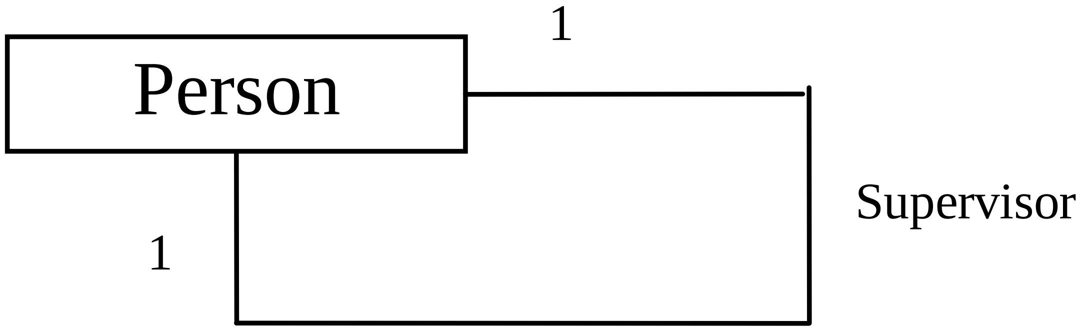
- A Person works for a Company.



Association Between Same Class

Association may exist between objects of the same class.

For example, a person may have a supervisor.



Multiplicity

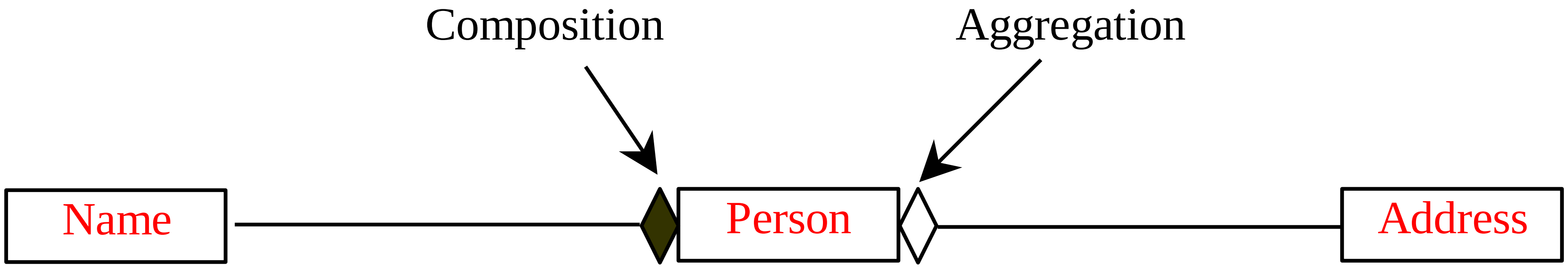
- One class can be related to another in a
  - One-to-one
  - One-to-many
  - One-to-one or more
  - One-to-zero or one
  - One-to-a bounded interval (one-to-two through twenty)
  - One-to-exactly n
  - One-to-a set of choices (one-to-five or eight)

Multiplicity

- Multiplicity can be expressed as,
  - Exactly one - 1
  - Zero or one - 0..1
  - Many - 0..* or *
  - One or more - 1..*
  - Exact Number - e.g. 3..4 or 6
  - Or a complex relationship – e.g. 0..1, 3..4, 6..* would mean any number of objects other than 2 or 5

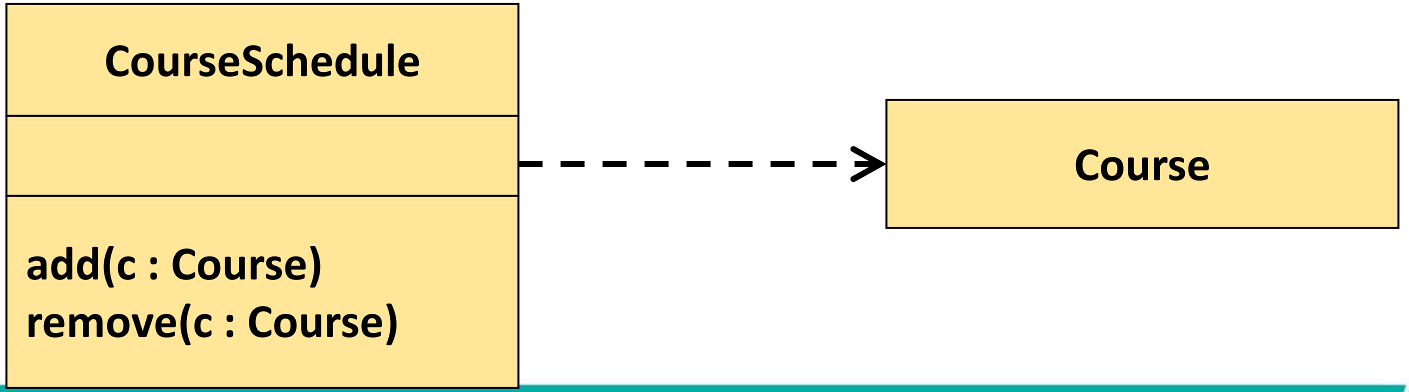
Association: Aggregation & Composition

- ***Aggregation:*** special form of association
  - Represents ownership relationship
  - Aggregation models the **has-a** relationship.
- ***Composition:*** special form of aggregation
  - object exclusively owned by aggregated object



Dependency Relationships

- **Dependency:** relationship between two classes where one (**called client**) uses the other (**called supplier**).
- In UML, draw a dashed line with an arrow from the client class to the supplier class.
- The dependency from CourseSchedule to Course exists because Course is used in both the add and remove operations of CourseSchedule.

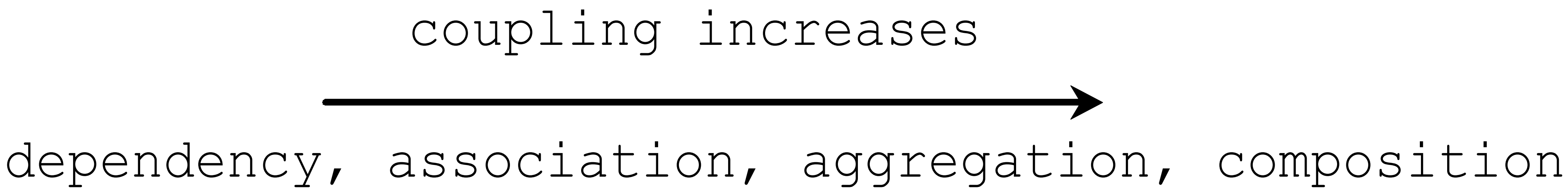


Dependency vs. Association

- Association is stronger than dependency. In association, the state of the object changes when its associated object changes.
- In dependency, the client object and the supplier object are loosely coupled.
- Implementation in programming
  - Association is implemented using data **fields and methods**. There is a strong connection between two classes.
  - Dependency is implemented using **methods**.

Coupling

- Dependency, association, aggregation, and composition all describe **coupling** relationships between two classes.



coupling increases

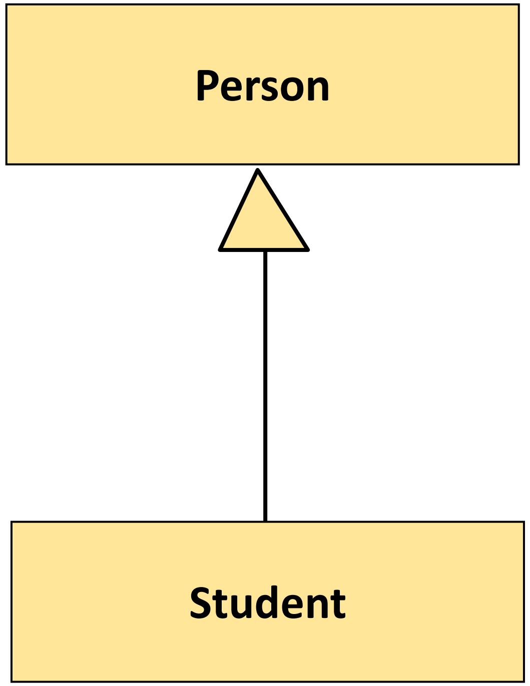
dependency, association, aggregation, composition

Generalization

- Generalization models the **is-an-extension-of** relationship between two classes: **is – a** relationship.
- hierarchies drawn top-down with arrows pointing upward to parent
- line/arrow styles differ, based on whether parent is a(n):
  - class:
    solid line, white arrow
  - abstract class:
    solid line, white arrow
  - interface:
    dashed line, white arrow

Generalization

- A **generalization** connects a **subclass** to its **superclass**. It denotes an **inheritance of attributes and behavior** from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

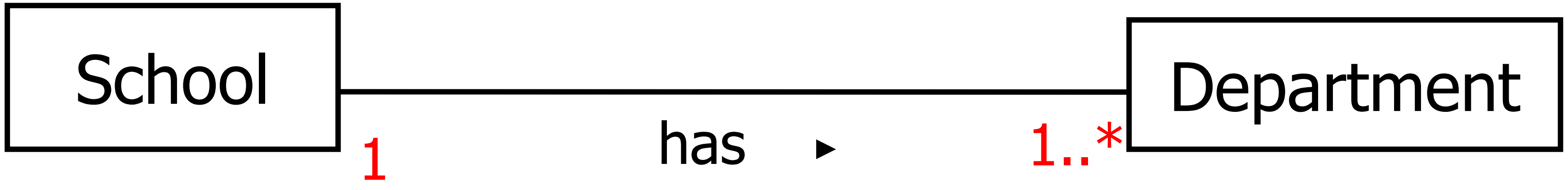


Class Diagram - Example

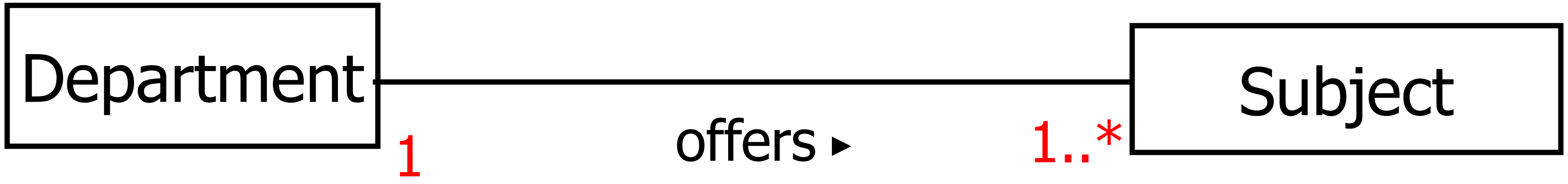
- Draw a class diagram for an information modeling system for a school.
  - School has one or more Departments.
  - Department offers one or more Subjects.
  - A particular subject will be offered by only one department.
  - Department has instructors and instructors can work for one or more departments.
  - Student can enrol in upto 5 subjects in a School.
  - Instructors can teach upto 3 subjects.
  - The same subject can be taught by different instructors.
  - Students can be enrolled in more than one school.

Class Diagram - Example

- School has one or more Departments.

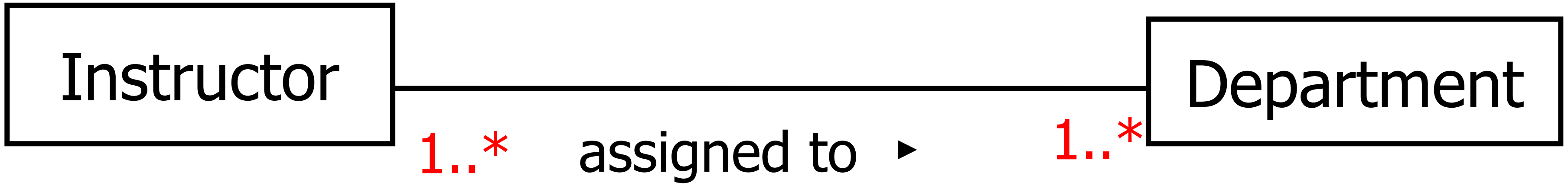


- Department offers one or more Subjects.
- A particular subject will be offered by only one department.

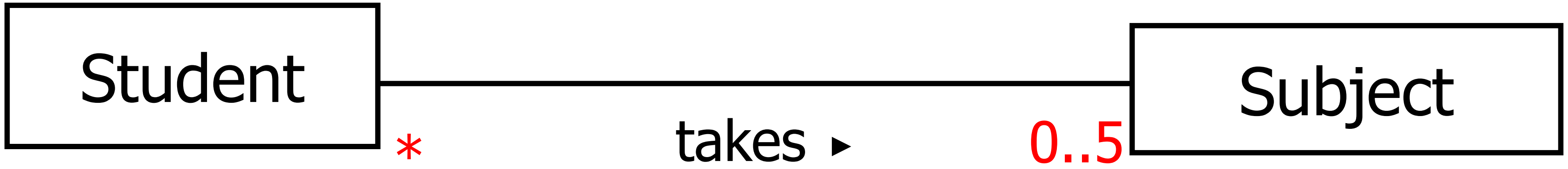


Class Diagram - Example

- Department has Instructors and instructors can work for one or more departments.

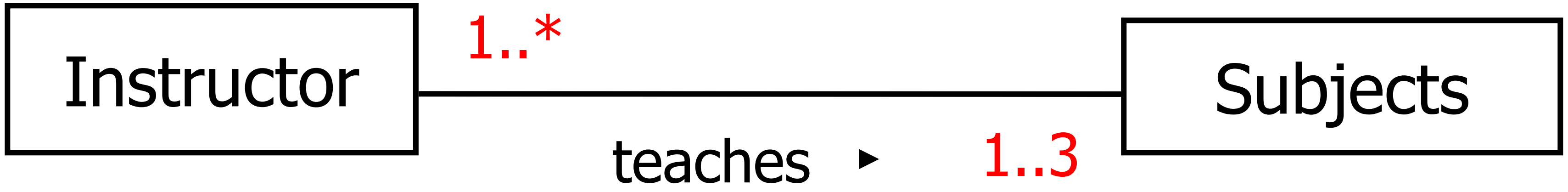


- Student can enrol in upto 5 Subjects.



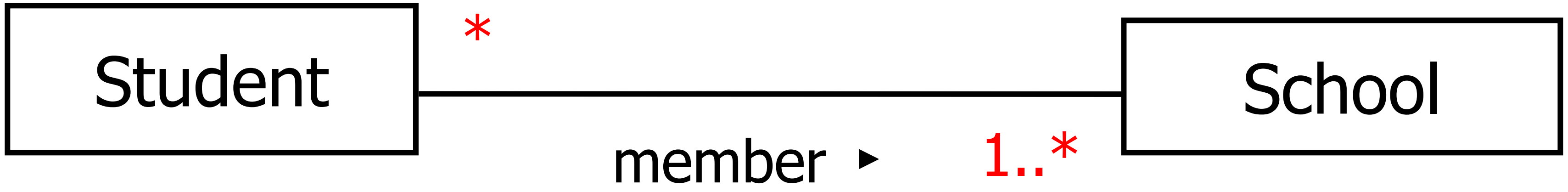
Class Diagram - Example

- Instructors can teach up to 3 subjects.
- The same subject can be taught by different instructors.

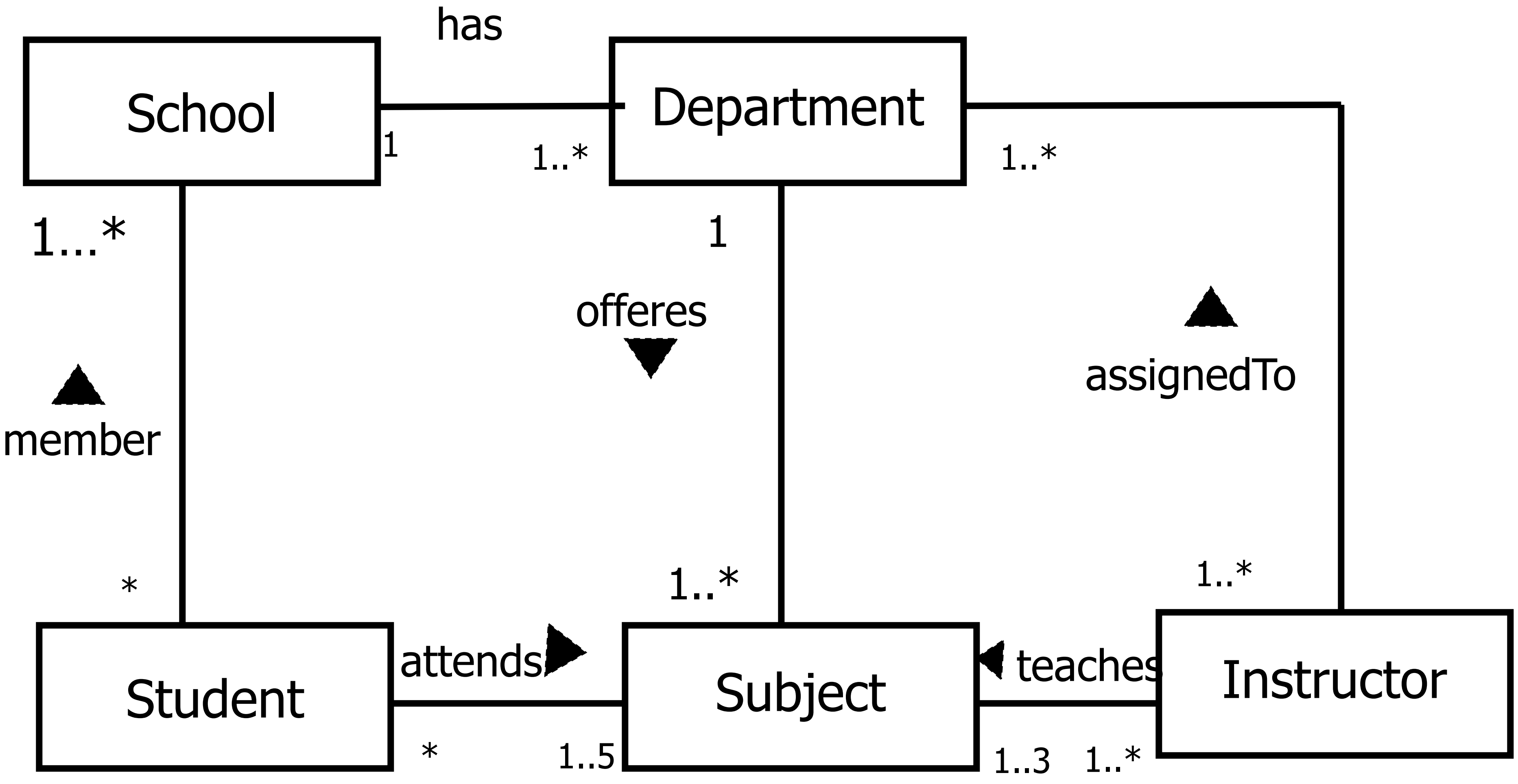


Class Diagram - Example

- Students can be enrolled in more than one school.



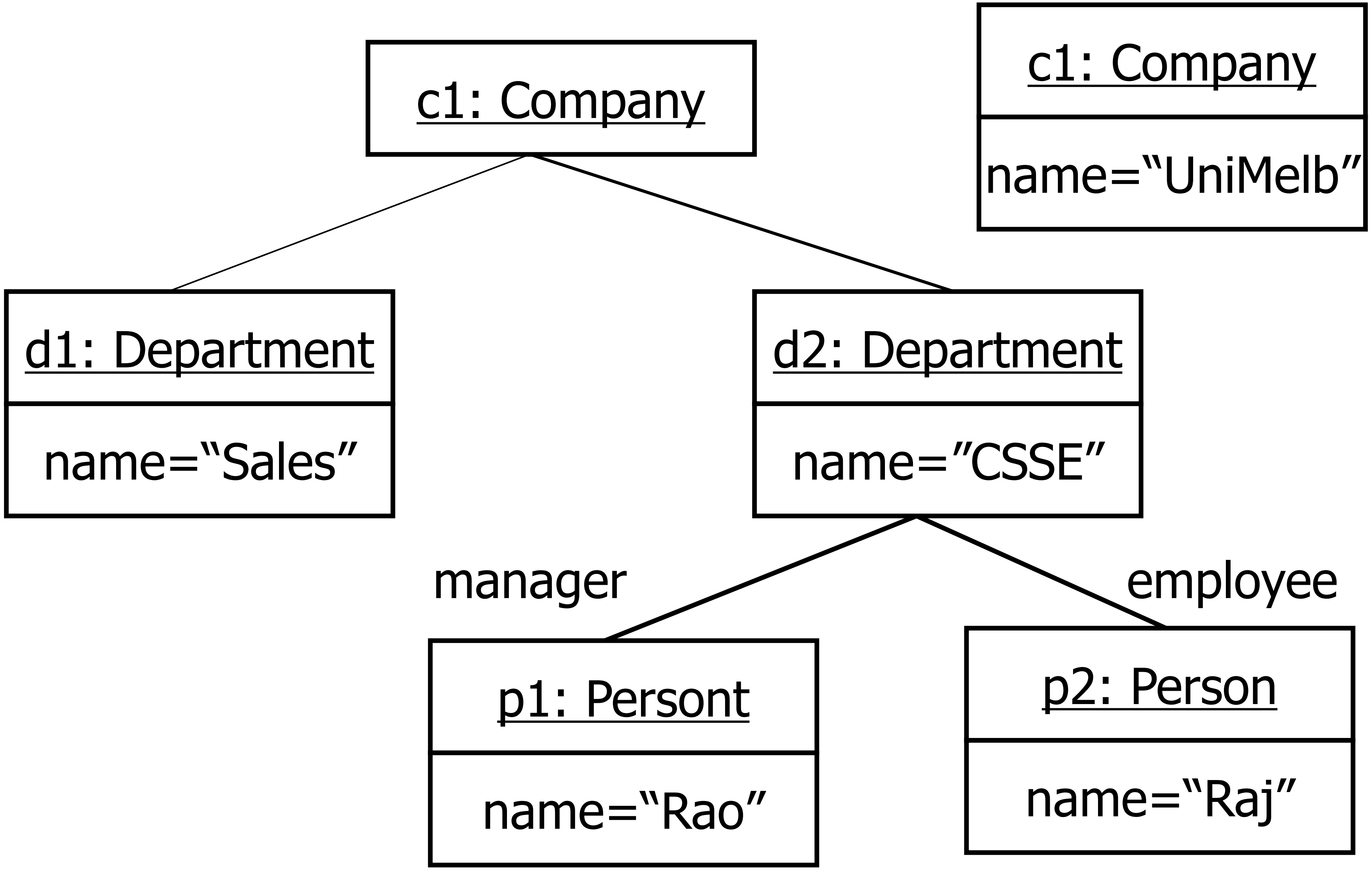
Class Diagram Example



Object Diagram

- Object Diagram shows the relationship between objects.
- Unlike classes objects have a state.

Object Diagram - Example



c1: Company
c1: Company
name="UniMelb"
d1: Department
name="Sales"
d2: Department
name="CSSE"
manager
employee
p1: Persont
name="Rao"
p2: Person
name="Raj"