Modeling using a Class diagram

Objectives

- Describe the static view of the system and show how to capture it in a model.
- Demonstrate how to read and interpret a class diagram.
- Model an association and aggregation and show how to model it in a class diagram.
- Model generalization on a class diagram.

Categories of diagrams

- Structural Diagrams
  - Class diagram
  - Object diagram
  - Component diagram
  - Deployment diagram
- Behavioral Diagrams
  - Use case diagram
  - Sequence diagram
  - Collaboration diagram
  - Statechart diagram
  - Activity diagram

How is an Object model built?

- Object model or class model is constructed with the help of **class diagrams and object diagrams**
- It defines the various attributes carried by every class and the operations that each class performs.
- Subsequently objects also inherit the same when classes are realised

Class Diagram Usage

- When modeling the static view of a system, class diagrams are typically used in one of three ways, to model:
  - The vocabulary of a system
  - Collaborations
  - A logical database schema

Class diagram

- Main static analysis diagram
- The static view of a system primarily supports the functional requirements of a system.
- Class diagrams show the static structure of the model, in particular, the things that exist such as classes, their internal structure, and their relationships to other classes.
- Class diagrams do not show temporal information.
- A collection of classes and their relationships connected as a graph to each other and to their contents.

When is a Class diagram constructed?

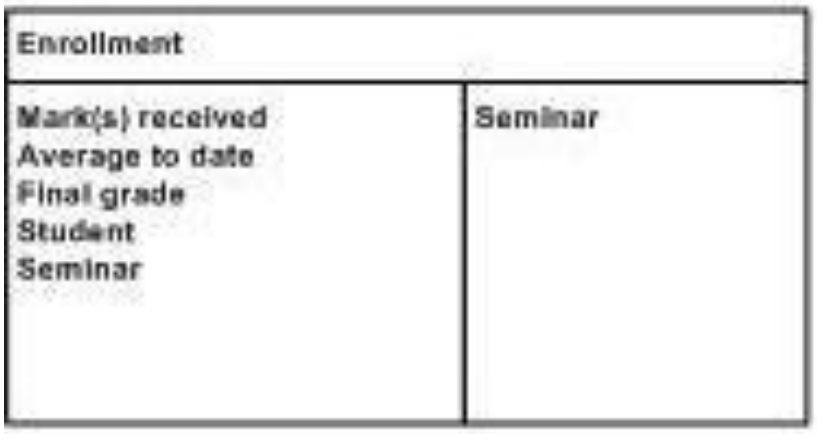
- **During Design phase in order to** specify the structure of how a software system will be written and function, without actually writing the complete implementation
- It is a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?

What should a Class diagram contain and what it should not contain?

- **UML class diagram**: a picture of
  - the classes in an OO system
  - their fields and methods
  - connections between the classes
    - that interact or inherit from each other
- Not represented in a UML class diagram:
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

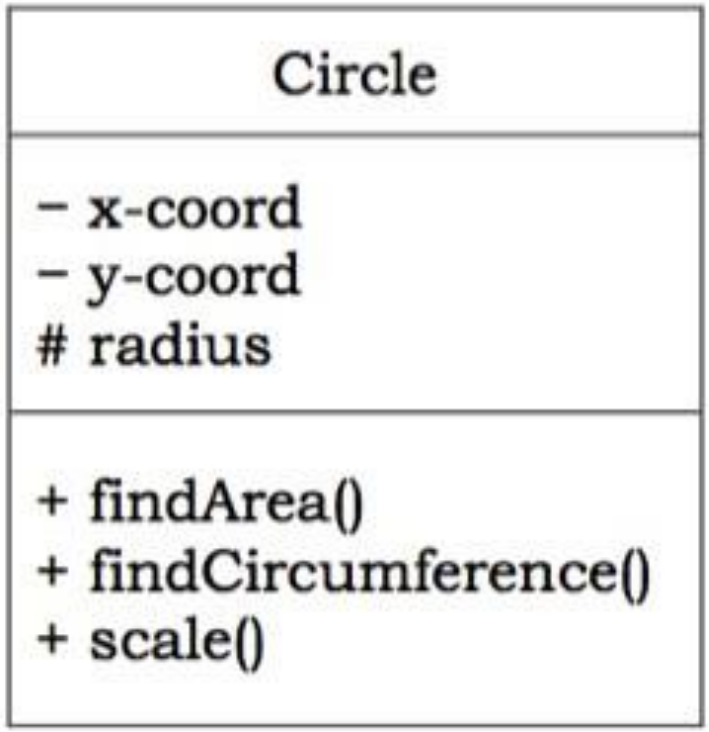
The four alternative approaches for identifying classes

- The noun phrase approach.
- The common class patterns approach.
- The use-case driven, sequence/collaboration modeling approach.
- The classes, responsibilities and collaborators (CRC) approach.



Class notation

- A class is drawn as a rectangle with three compartments separated by horizontal lines.
- The top compartment holds the class name, middle compartment holds attributes and their properties and the last compartment holds a list of operations.
- Either or both the attribute and operations compartments may be suppressed. A separator line is not drawn in such a case




```
class Circle
{
    private:
      double x_coord;
      double y_coord;
    protected:
      double radius;
    public:
      double findArea();
      double findCircumference();
      void scale();
};
```

Diagram of one class

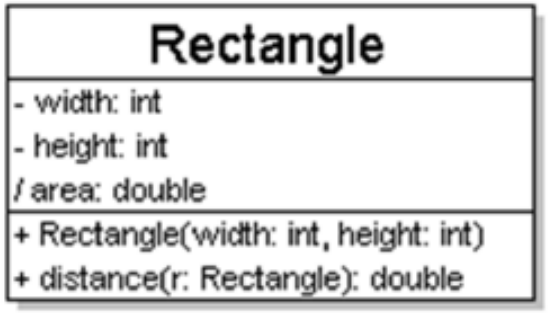
- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods

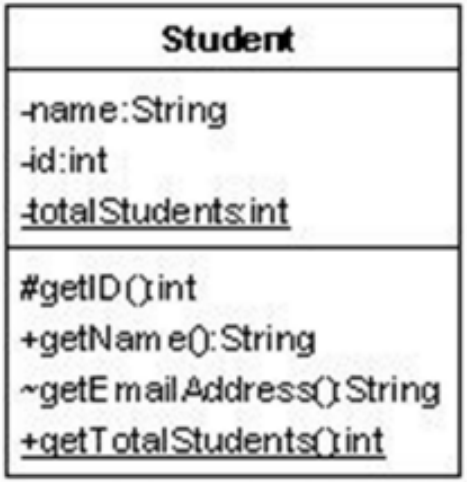
Class attributes /fields

- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:

+	public
#	protected
-	private
~	package (default)
/	derived

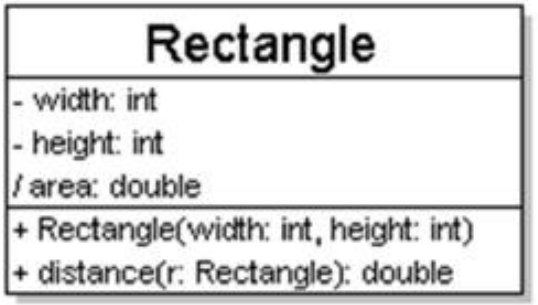
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
    - "specification fields " from CSE 331
  - attribute example:
    - balance : double = 0.00

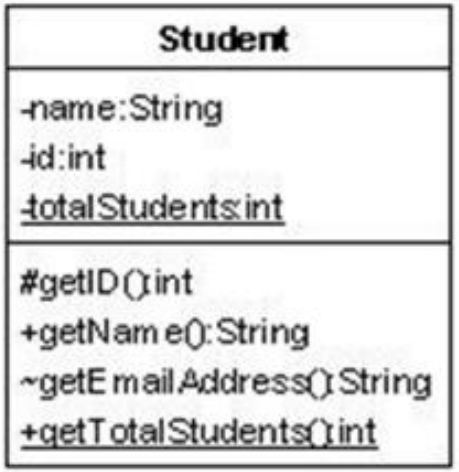




Class operations / methods

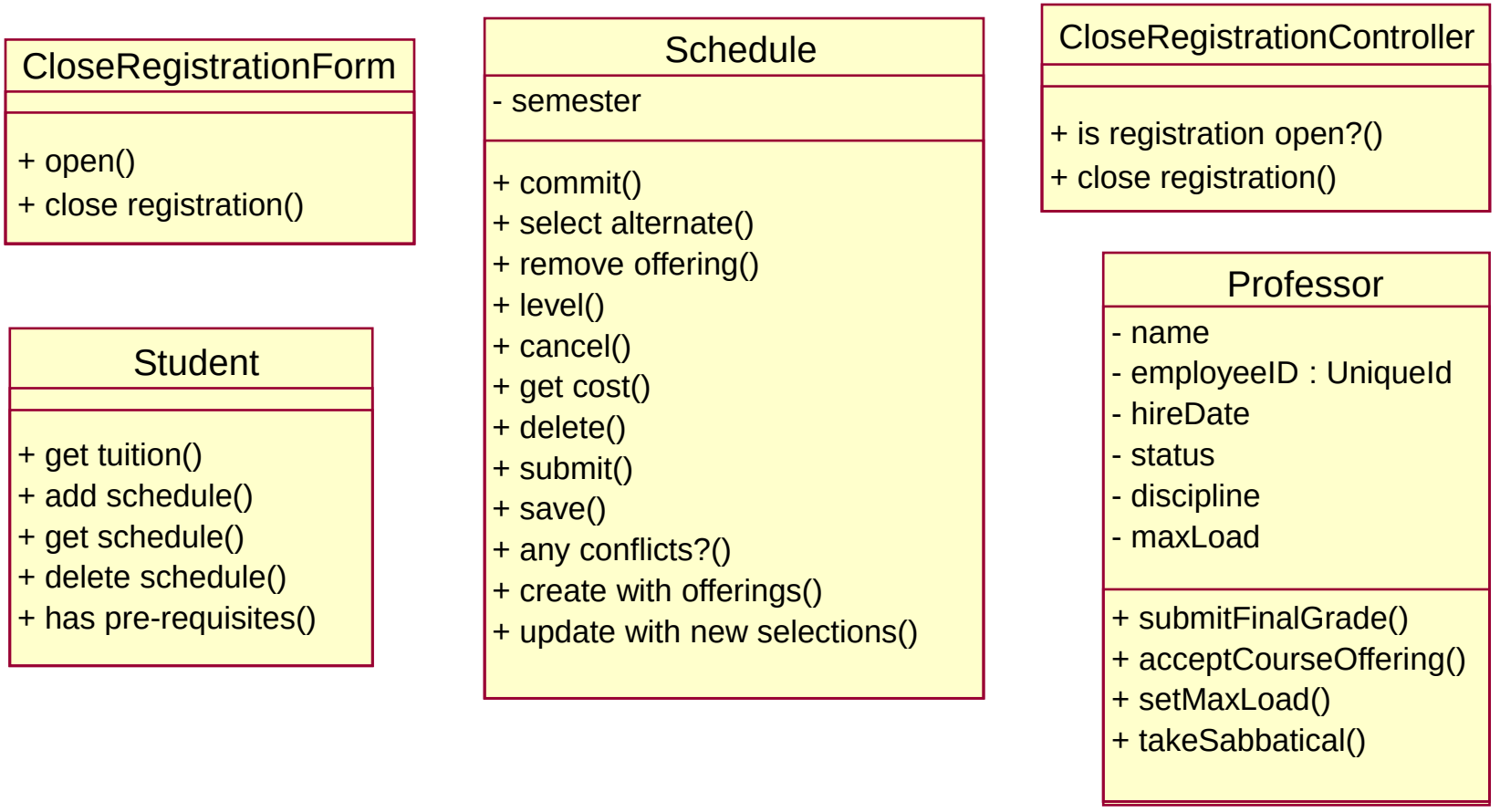
- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility:
    - \+ public
    - \# protected
    - \- private
    - ~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    + distance(p1: Point, p2: Point): double





What Is a Class Diagram?

- Static view of a system



Relationships in class diagrams

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

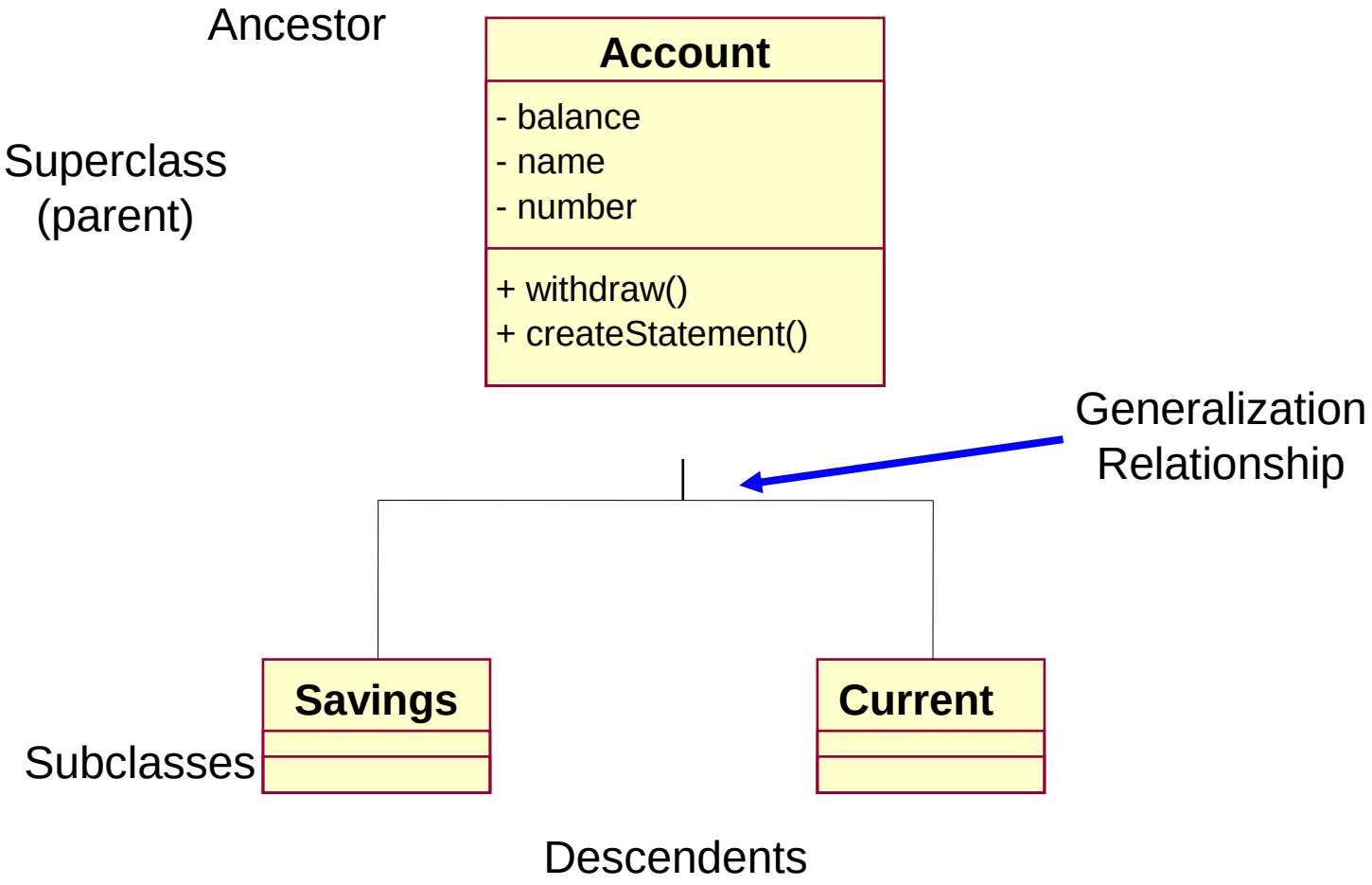
- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

What Is Generalization?

- A relationship among classes where one class shares the structure and/or behavior of one or more classes.
- Defines a hierarchy of abstractions where a subclass inherits from one or more superclasses.
  - Single inheritance
  - Multiple inheritance
- Is an “is a kind of” relationship.

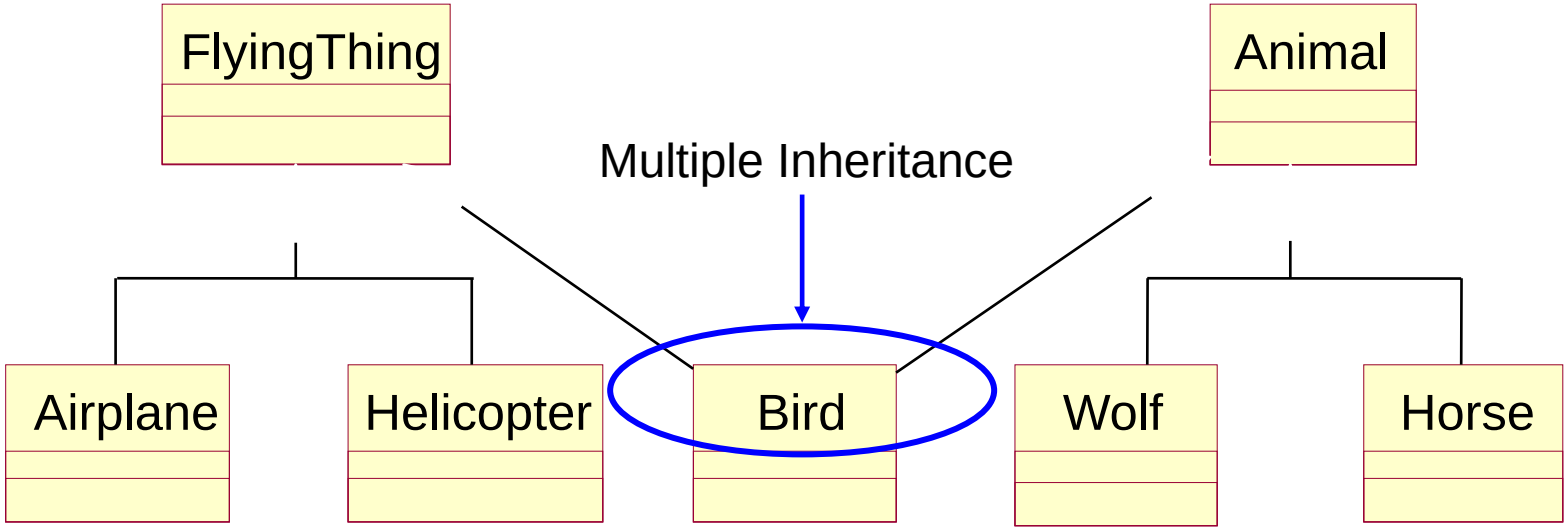
Example: Single Inheritance

- One class inherits from another.



Example: Multiple Inheritance

- A class can inherit from several other classes.

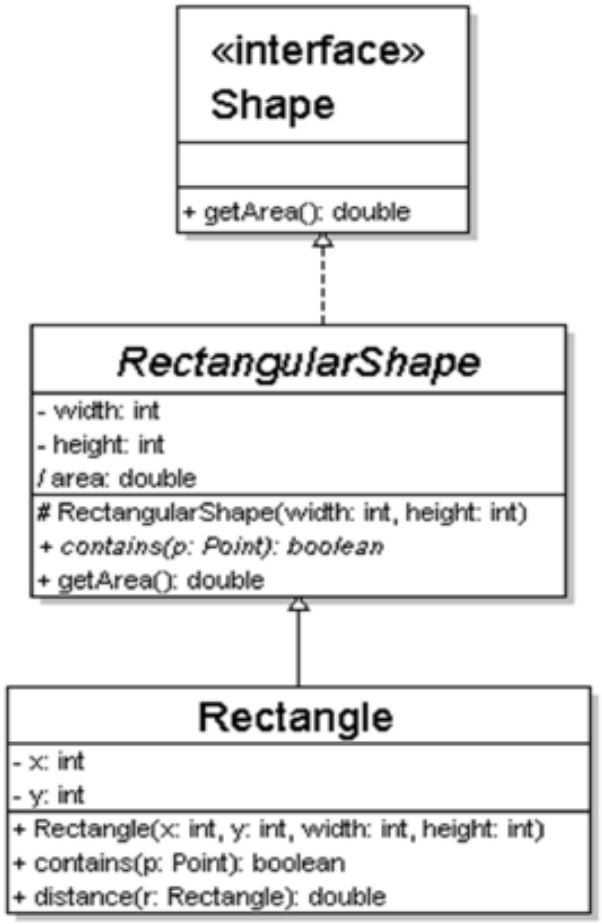


Use multiple inheritance only when needed and always with caution!

Generalization (inheritance) relationships

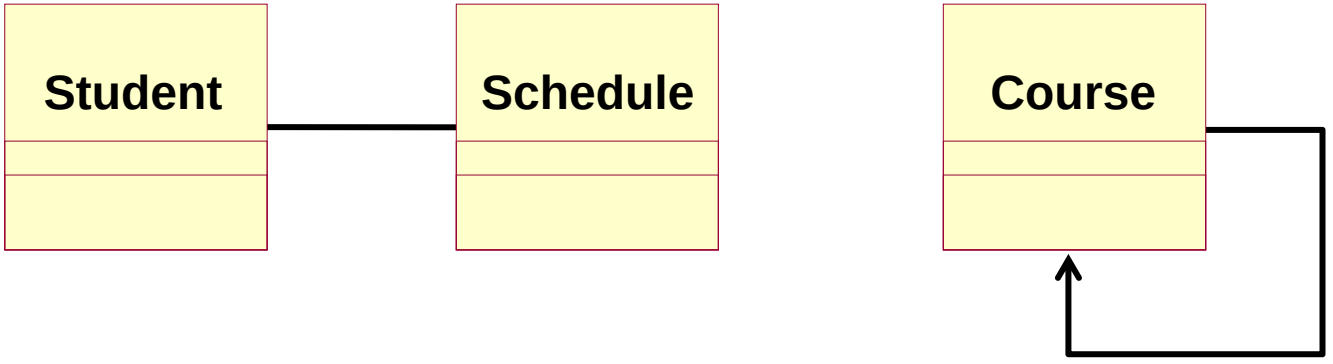
- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
  - class:
    solid line, black arrow
  - abstract class:
    solid line, white arrow
  - interface:
    dashed line, white arrow

- often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



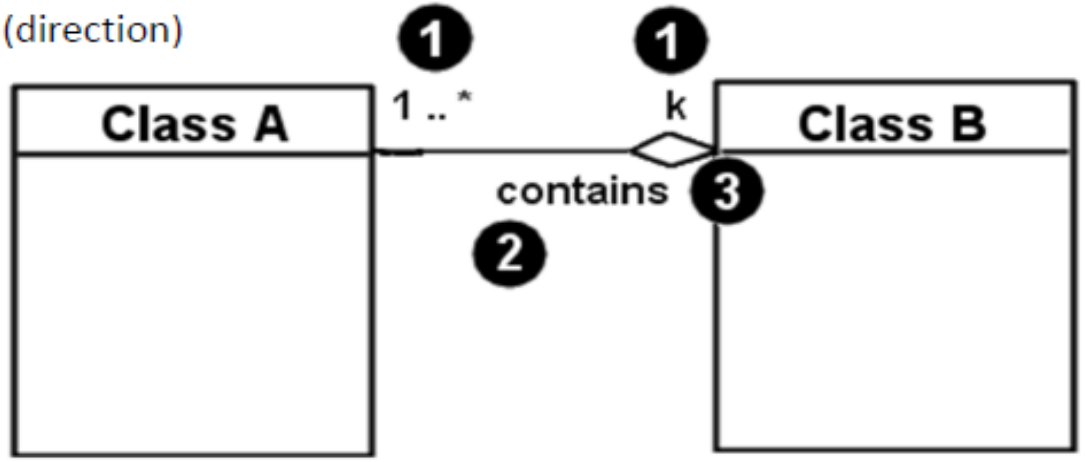
What Is an Association?

- The semantic relationship between two or more classifiers that specifies connections among their instances.
- In other words, an association is a structural relationship that specifies that objects (instances of classes) of one thing are connected to objects of another thing
- A structural relationship specifying that objects of one thing are connected to objects of another thing.



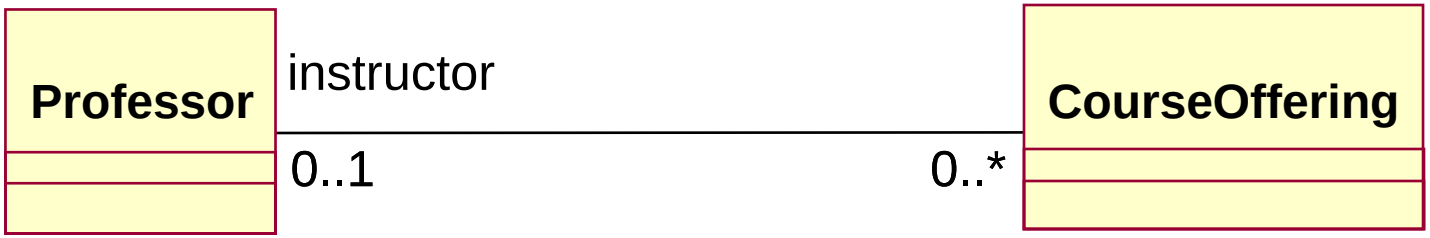
Association

- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as "3..")
  2. name (what relationship the objects have)
  3. navigability (direction)



What Is Multiplicity?

- Multiplicity is the number of instances one class relates to instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.

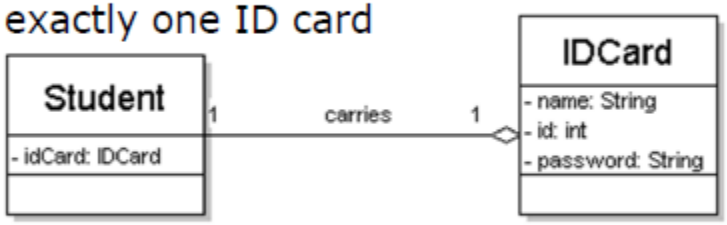


Multiplicity Indicators

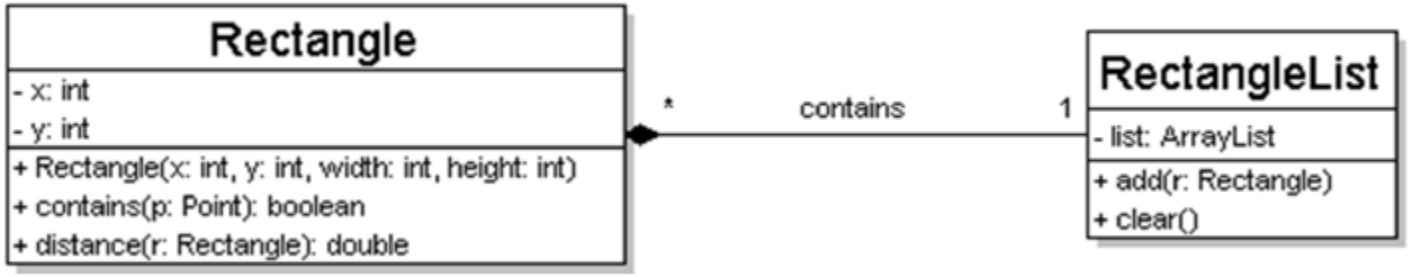
Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional scalar role)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

Multiplicity of Associations

- one-to-one
  - each student must carry exactly one ID card

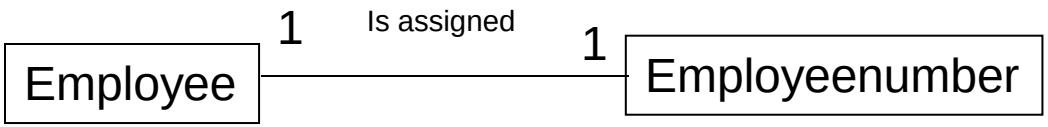


- one-to-many
  - one rectangle list can contain many rectangles

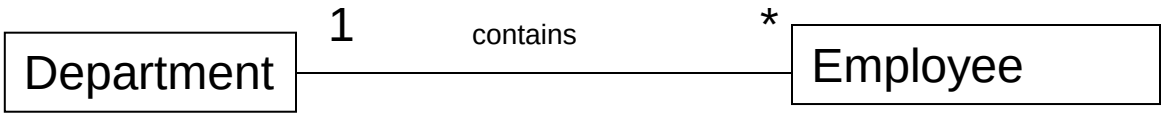


Examples of multiplicities

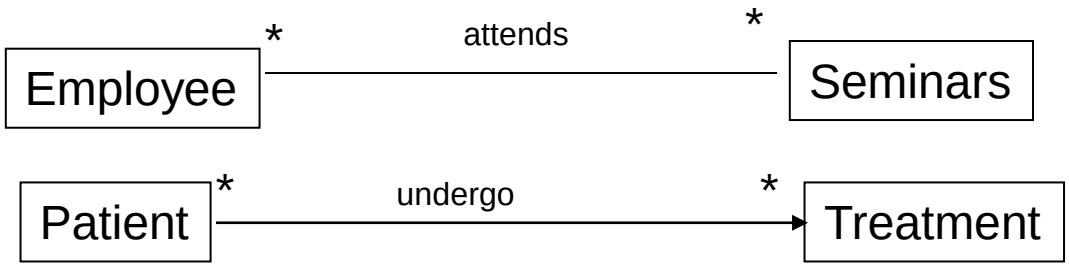
- **One to one association**: has multiplicity of 1 at each end



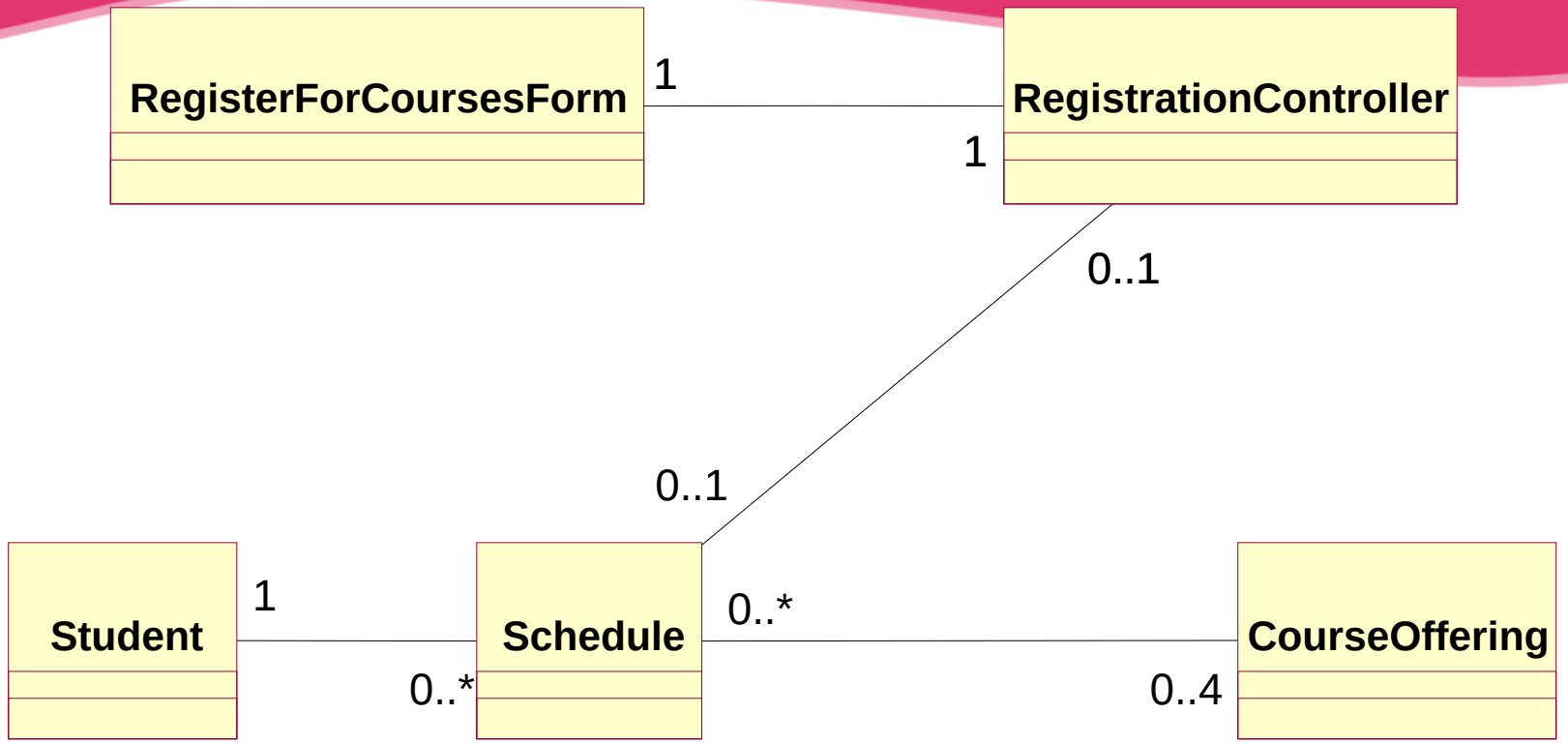
- **One to many association**: has multiplicity of 1 on end and 0..n or * or 1..n On other side



- **Many to many association** : 0..n or 1..n or * on both sides



Example: Multiplicity

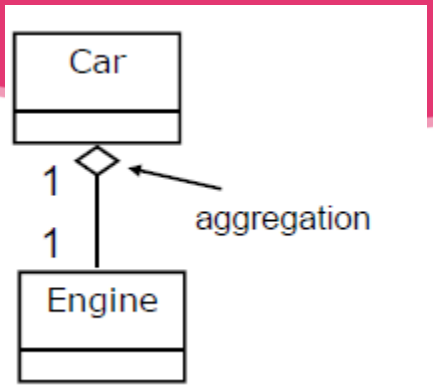


Advantages of using multiplicity

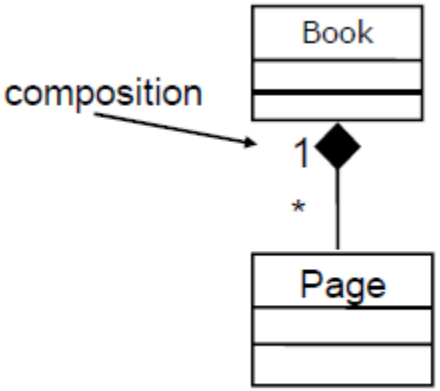
- Adding multiplicity to associations increases the amount of information we capture from the application or solution domain

Types of associations

- **aggregation**: "is part of"
  - symbolized by a clear white diamond

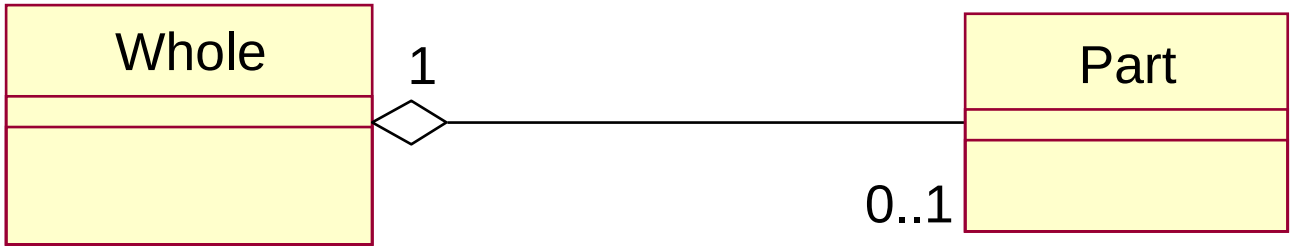


- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

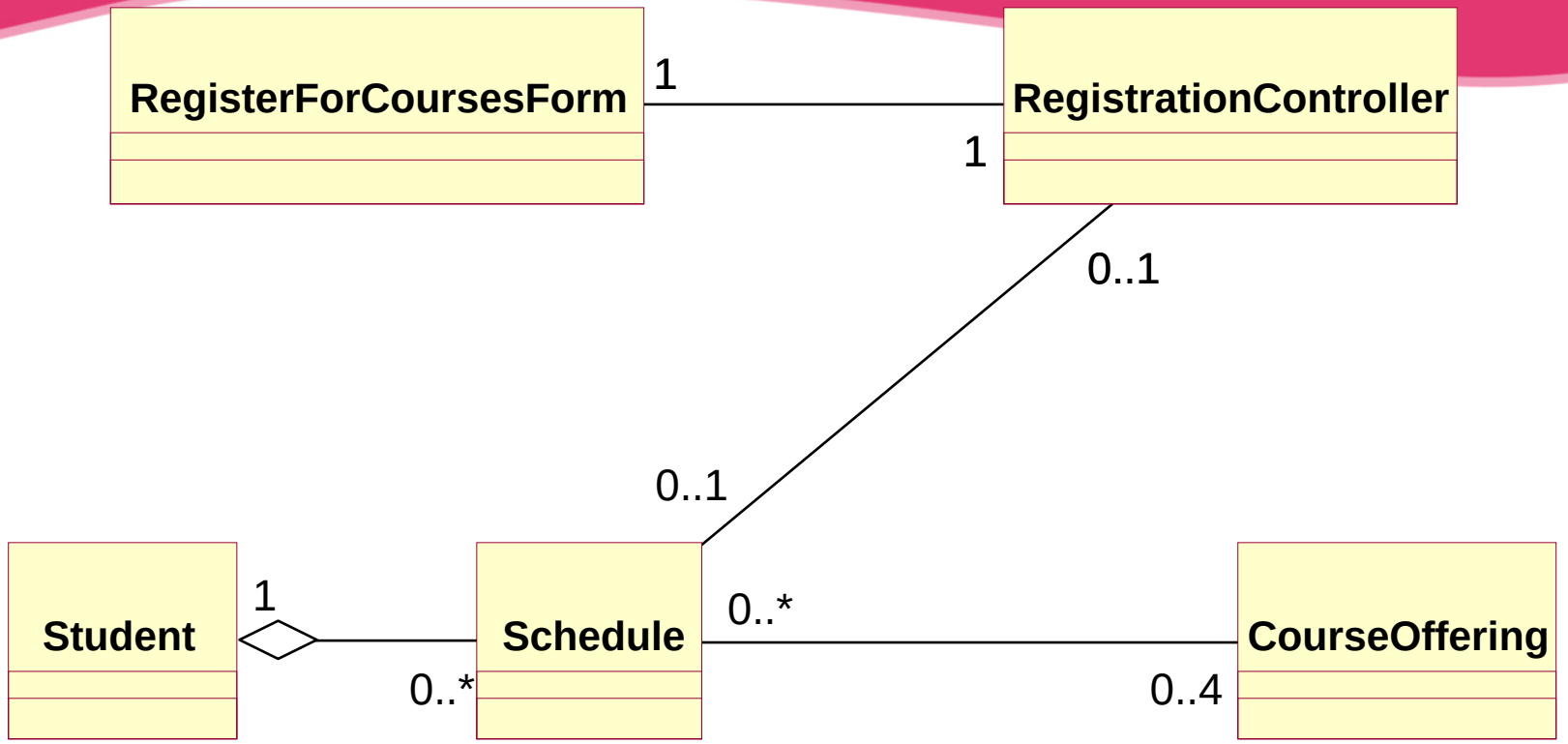


What Is an Aggregation?

- A special form of association that models a whole-part relationship between the aggregate (the whole) and its parts.
  - An aggregation is an “is a part-of” relationship.
- Multiplicity is represented like other associations.



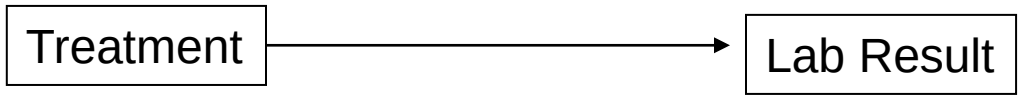
Example: Aggregation



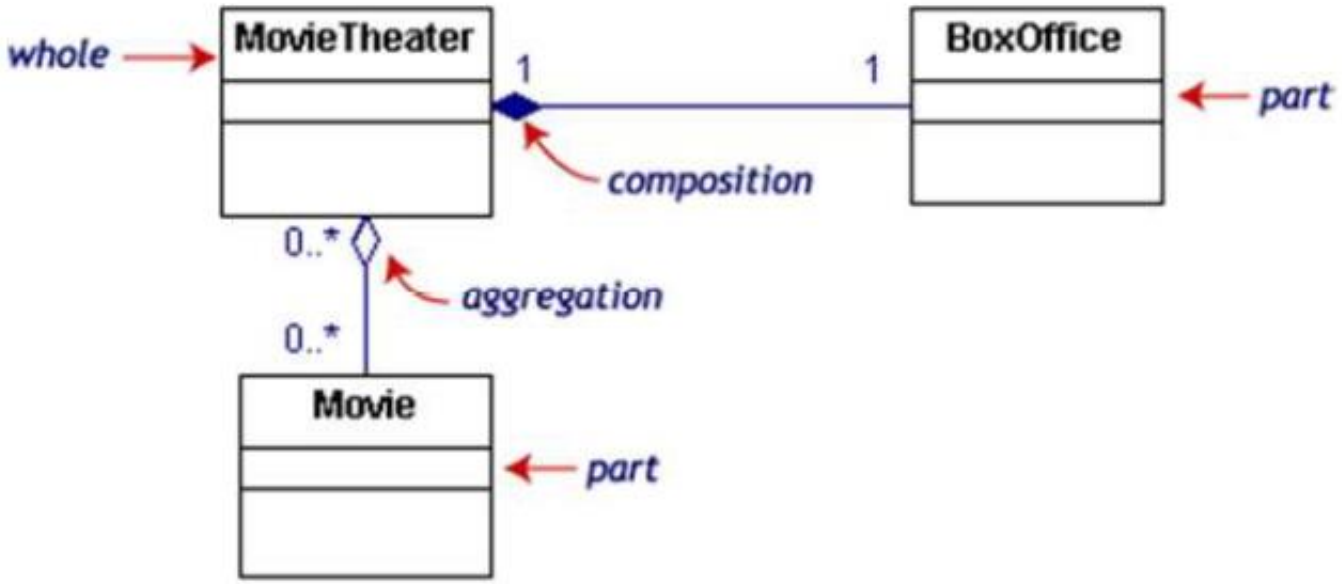
Types of associations

- **Dependency:**
  - Unidirectional
  - “Using” relationship

Ex: Treatment depends on Lab Result



Example pf Composition/Aggregation



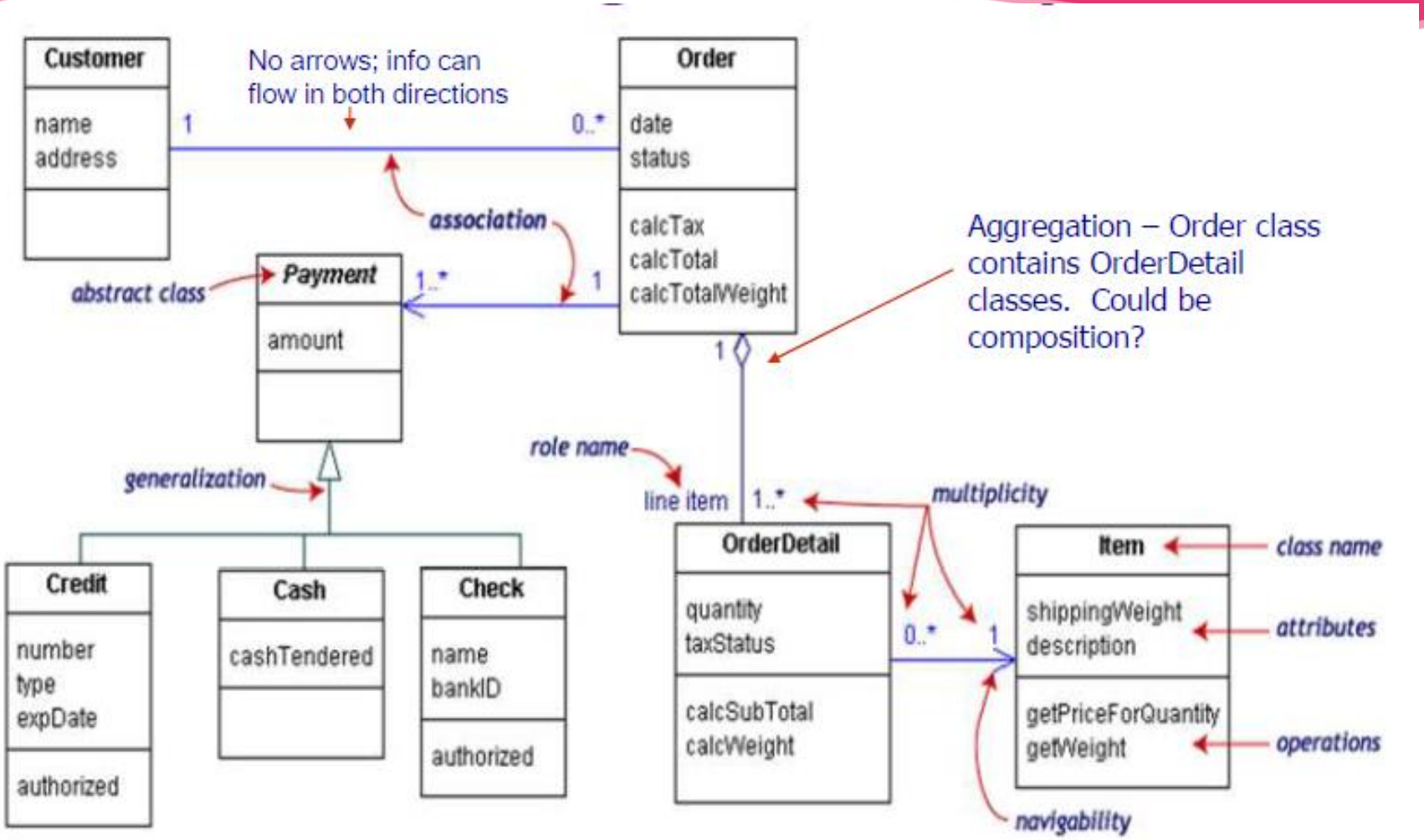
If the movie theater goes away
so does the box office => composition
but movies may still exist => aggregation

Class diagram pros/cons

- Class diagrams are great for:
  - discovering related data and attributes
  - getting a quick picture of the important entities in a system
  - seeing whether you have too few/many classes
  - seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  - spotting dependencies between one class/object and another

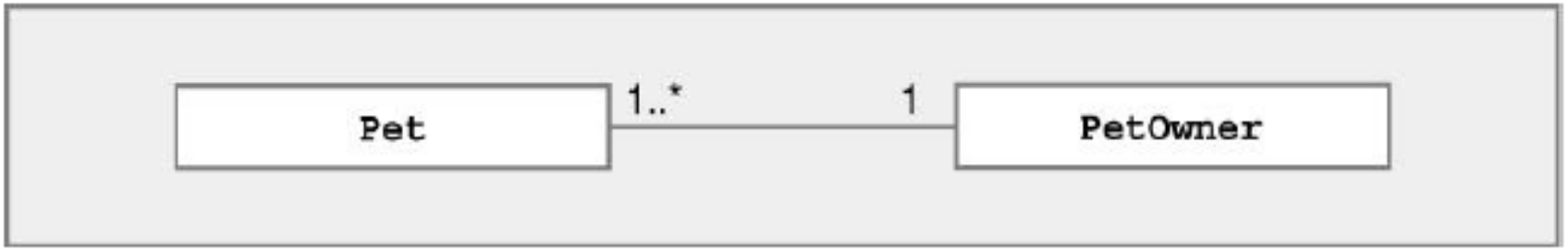
- Not so great for:
  - discovering algorithmic (not data-driven) behavior
  - finding the flow of steps for objects to solve a given problem
  - understanding the app's overall control flow (event-driven? web-based? sequential? etc.)

An example of Class diagram

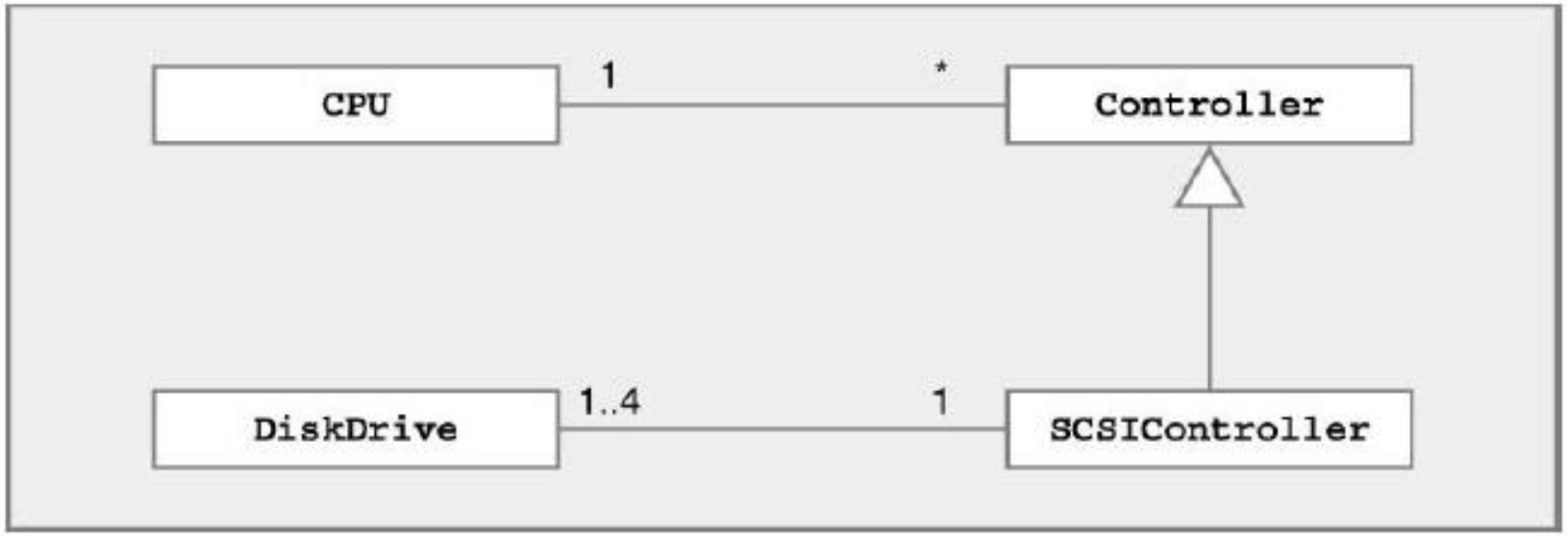


Now it’s time for few examples

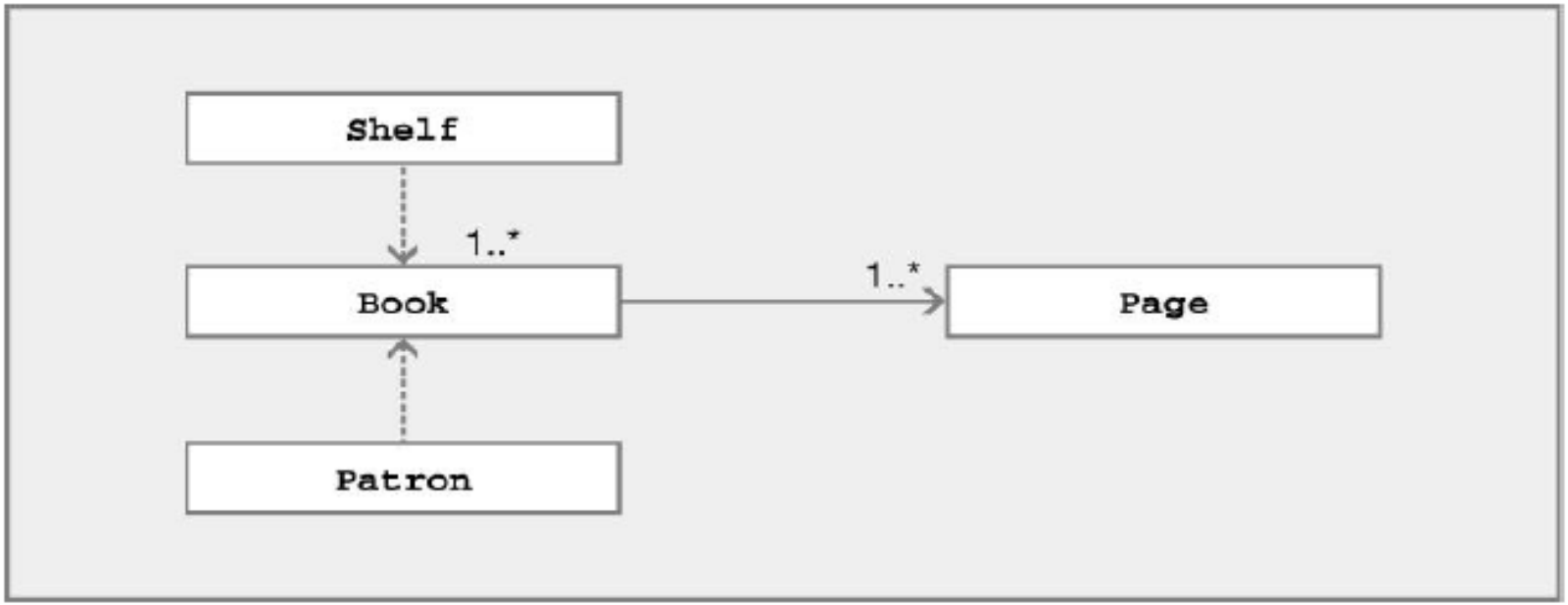
Write the following association in simple language



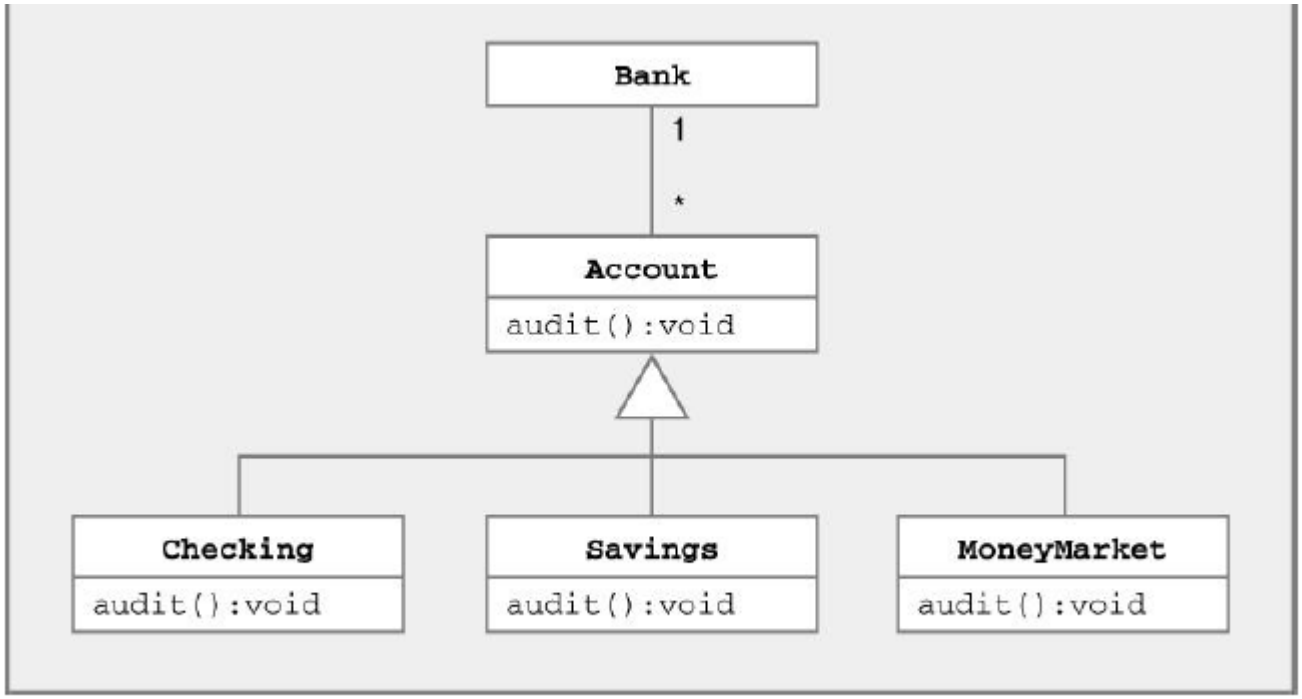
- 1 or more Pets associated with 1 PetOwner
- Each pet has exactly one PetOwner



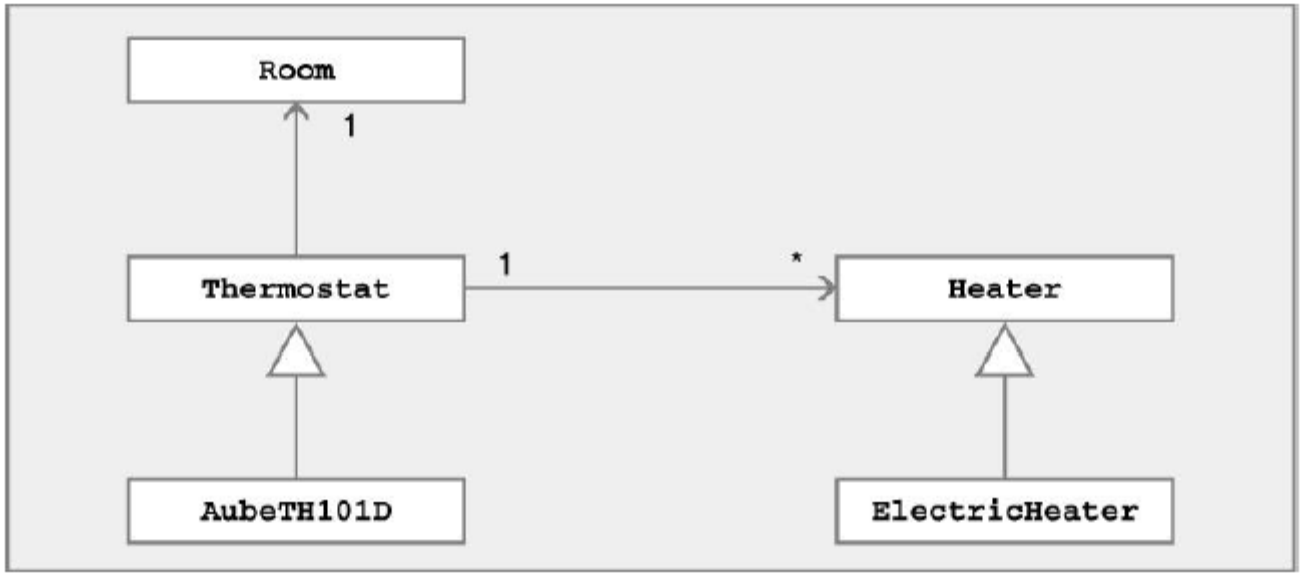
- 1 CPU associated with 0 or more Controllers
- 1-4 DiskDrives associated with 1 SCSIController
- SCSIController is a (specialized) Controller



- 1 or more Book associated with 1 or more Pages
- Patron & Shelf use (depend on) Books



- 1 Bank associated with 0 or more Accounts
- Each Account is associated with exactly one bank
- Checking, Savings, and MoneyMarket are Accounts



- Room has 1 Thermostat
- Each Thermostat is associated with 0 or more Heaters
- A Heather has exactly one Thermostat
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat

Class diagram for University Courses

- Some instructors are professors, while others have job title adjunct (extra)
- Departments offer many courses, but a course may be offered by >1 department
- Courses are taught by instructors, who may teach up to three courses
- Instructors are assigned to one (or more) departments
- One instructor also serves as a department chair

Class Diagram for Univ. Courses