Class Modeling

Object-Oriented Modeling and Design with UML (Second Edition)

Michael Blaha and James Rumbaugh

Pearson/PrenticeHall, 2005

Chapters 3 and 4

Objectives

You will be able to

- Discover classes that represent parts of a system to be designed.
- Draw class diagrams showing the structure of a system.

Why do class diagrams?

Help us to visualize the system.

  - Pictures are easier to understand than words.

- Provide a map of the system showing paths that a stream of execution can take.
- Show where functionality is available and how to get to it.
- Show where information is available and how to get to it.
- Classes’ public attributes and operations define the *interfaces* available for use by other classes.

Why do class diagrams?

- Ultimately provide blueprints for implementation.
- Iterative process
  - Steps along the way from high level descriptions to detailed design.

Class Diagrams

- Describe classes and how they relate to each other.
- For each class:
  - Name
  - Attributes
  - Operations
  - Relationships to other classes

Optional

Relationships

Lines between classes in the class diagram.

Show *associations* between classes.

- Correspond to *communication links* between objects.

Show Navigation Paths

Objects that a given object can *get to* in order to

- Retrieve information
- Invoke operations

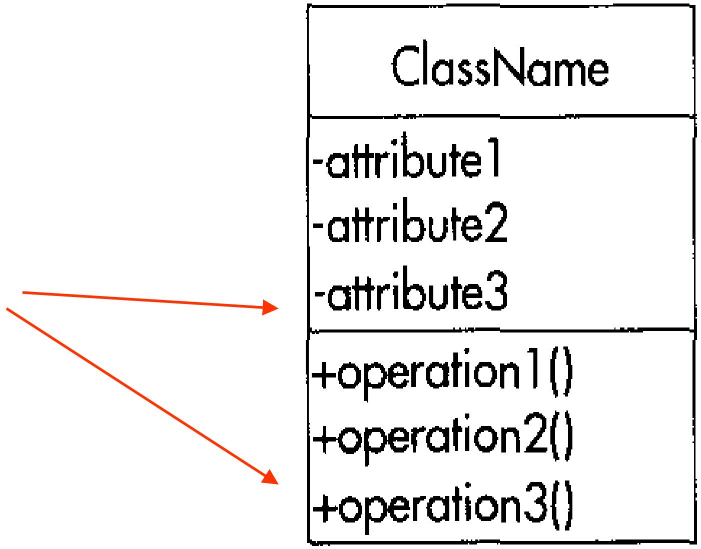
Notation for Classes

Accessibility Indicator

+ means public

-means private

# means protected



Class box is divided into three compartments.

Second and third compartments may be omitted.

Values and Attributes

- An *attribute* is a named property.
- Describes a *value*, not an object.
  - Use associations for objects.

Operations vs. Methods

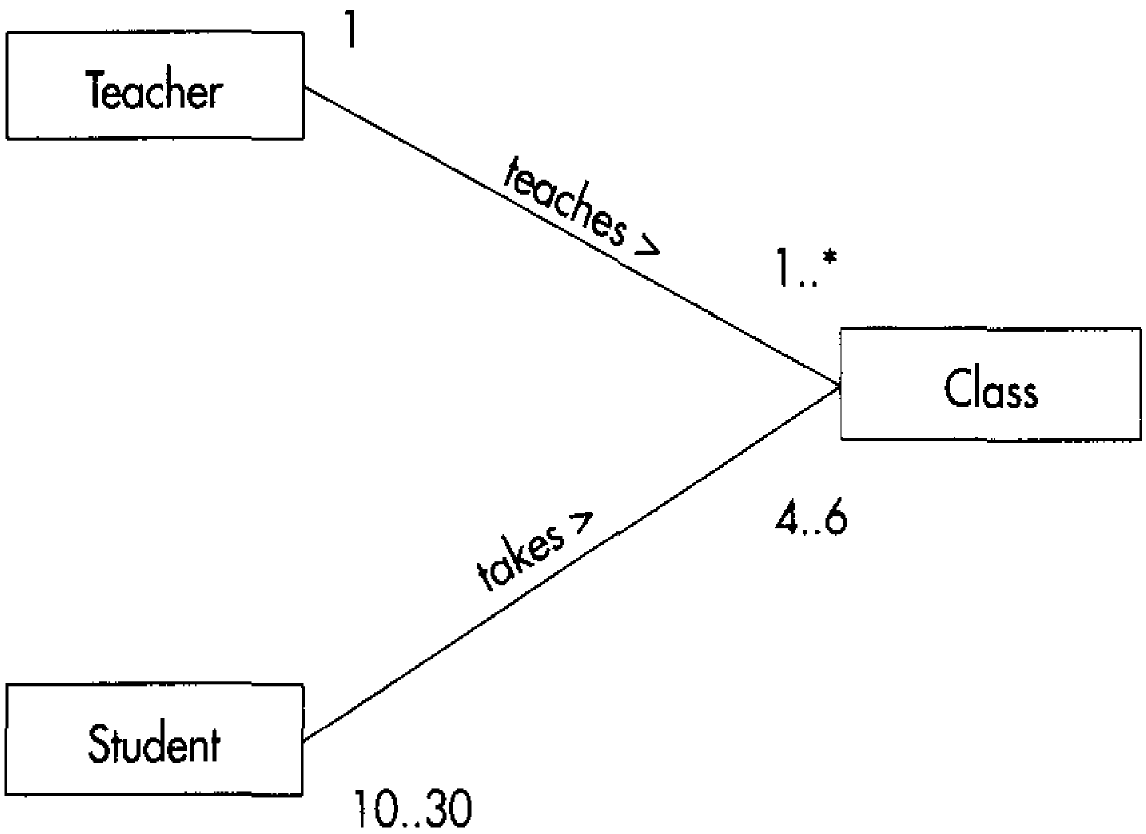
- An *operation* is something that can be done to or by objects of a class.
  - Abstract concept.
  - Implementation unspecified.
- A *method* is the implementation of an operation in an object-oriented language.
  - Specific chunk of code.
- A given operation can be implemented by different methods.

Class Diagrams

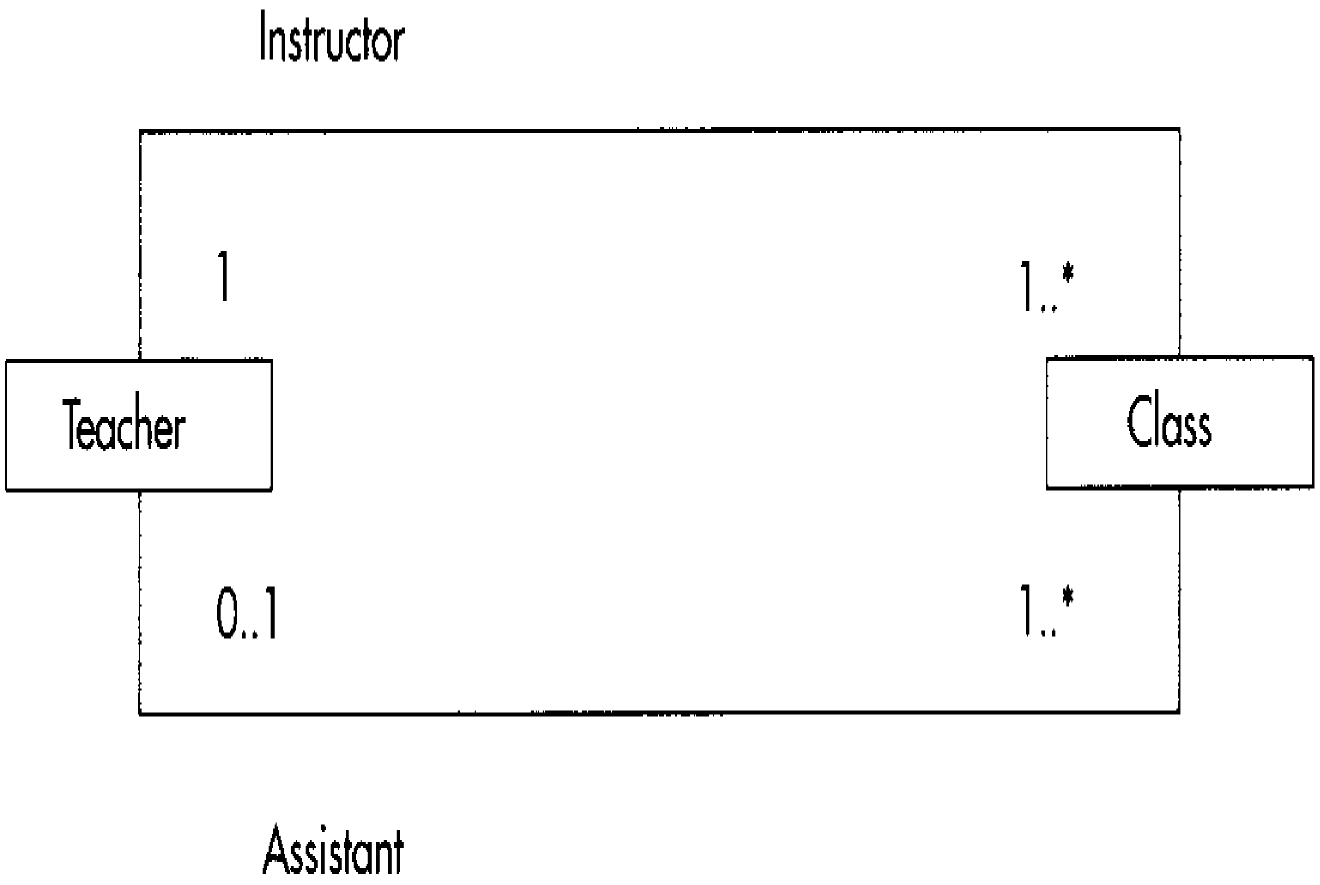
- Real classes typically have far more attributes and operations than can conveniently be shown on a diagram.
- Show the ones that are relevant for the problem at hand.
- Too much detail makes a diagram useless!

  Remember that your diagram is going to be read by humans, not compiled.

Class Diagram Example

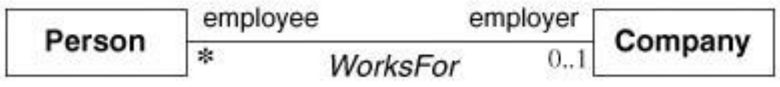


Class Diagram Example



A class can have different roles in different associations.

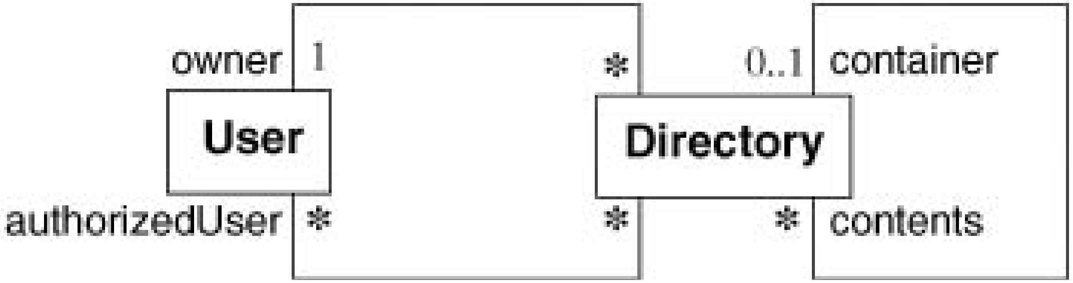
Association End Names



Each *end* of an association can have a name.

From Blaha and Rumbaugh, page 31.

Association End Names



Association end names can be used to distinguish multiple associations between the same pair of classes.

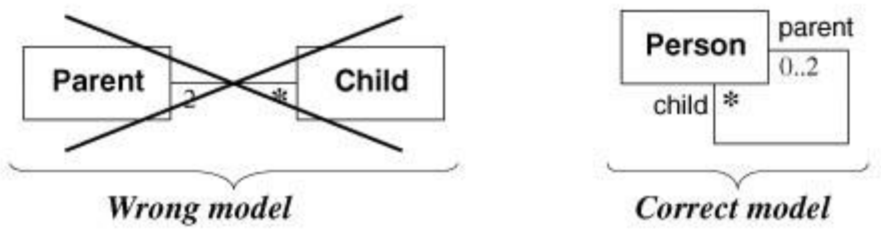
Association end names are *necessary* for associations between objects of the same class.

From Blaha and Rumbaugh, page 32.

Associations Between Objects of the Same Class

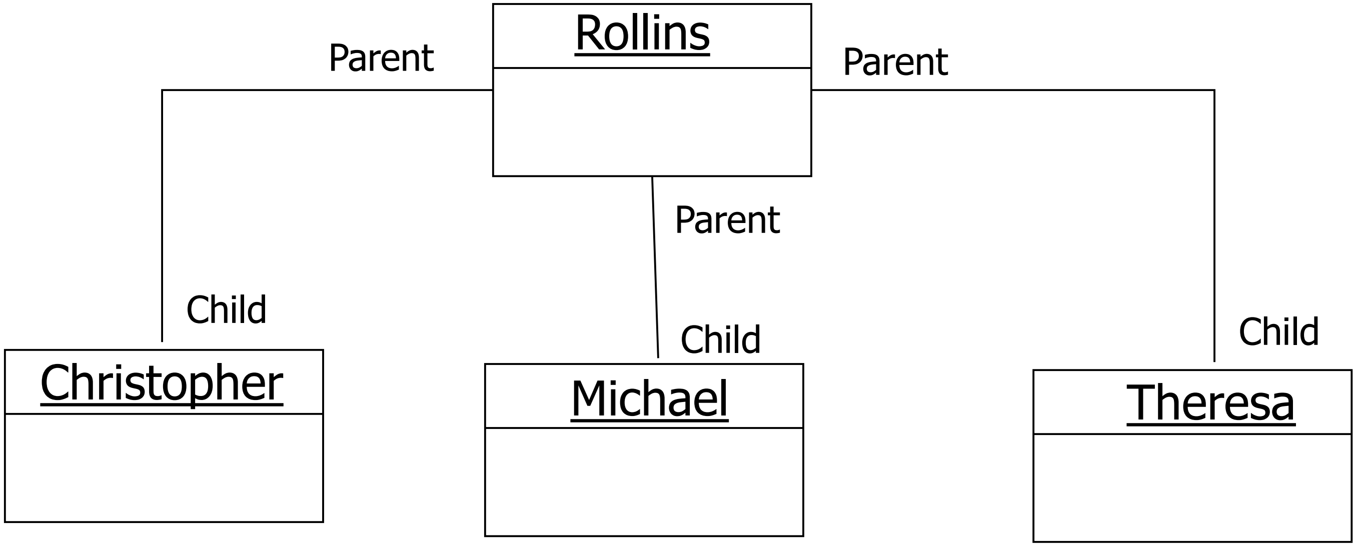
Don’t create a new class just to show an association.

Use an association from the class to itself.



From Blaha and Rumbaugh, page 32.

Corresponding Object Diagram

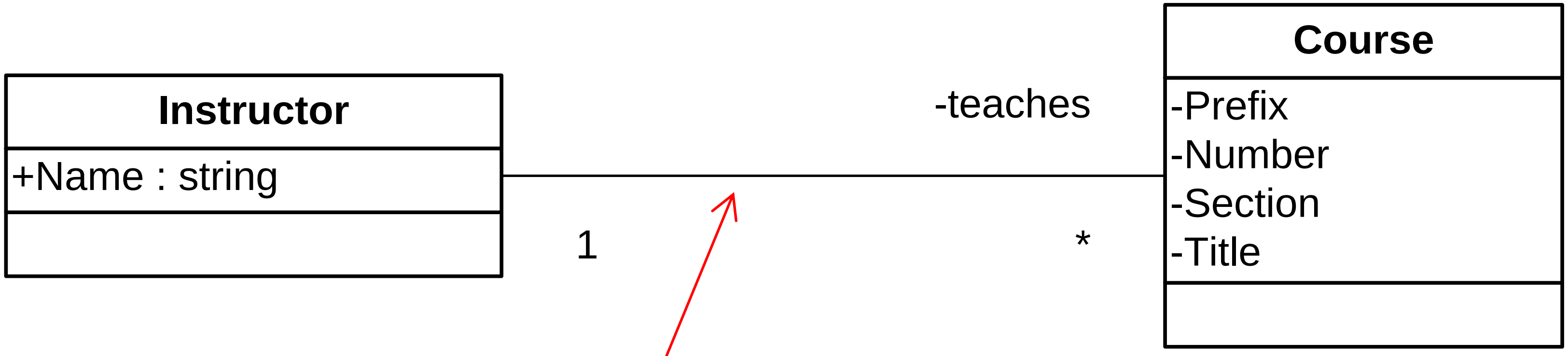


Associations vs. Links

- A *link* is a relationship beween *objects*.
  - An instance of an association.
  - Usually between a pair of objects.
  - Can relate more than two objects.

- An *association* describes multiple links with the same structure and semantics.
  - Association between classes.
  - Link between objects.

Class Diagram Example



An association

Object Diagrams

Notation for Name:

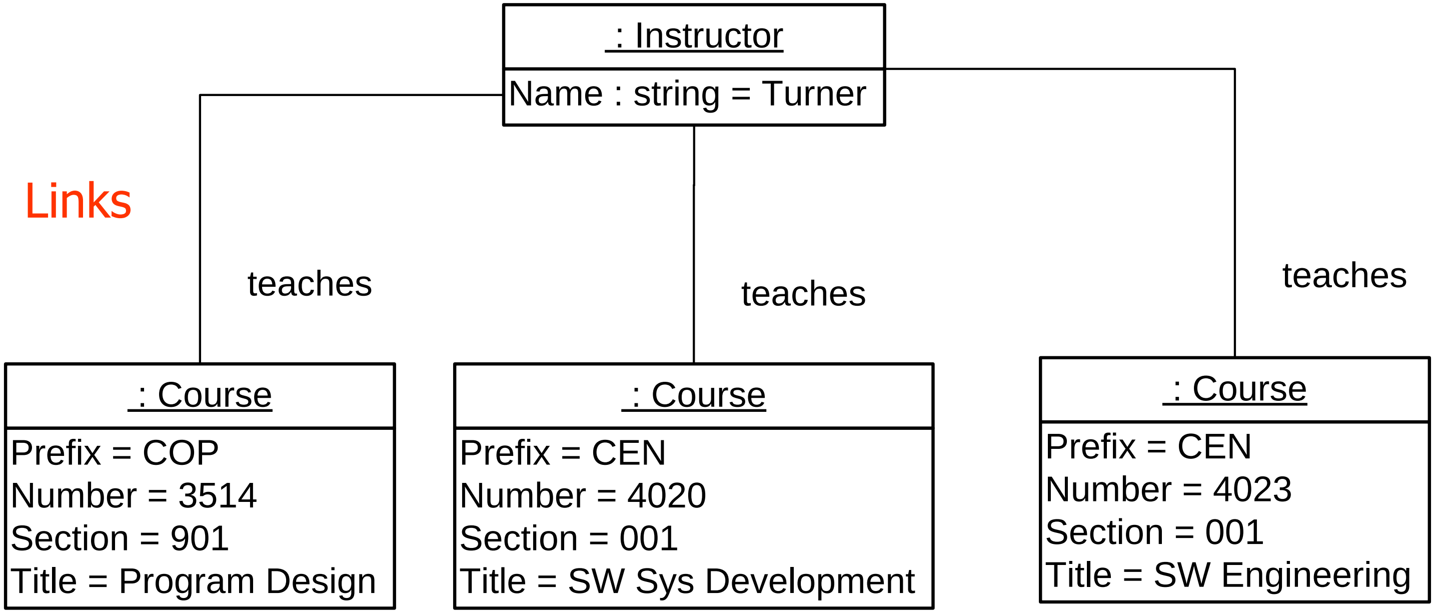
<u>ObjectName : ClassName</u>

Attribute *values* may be specified for an object or may not

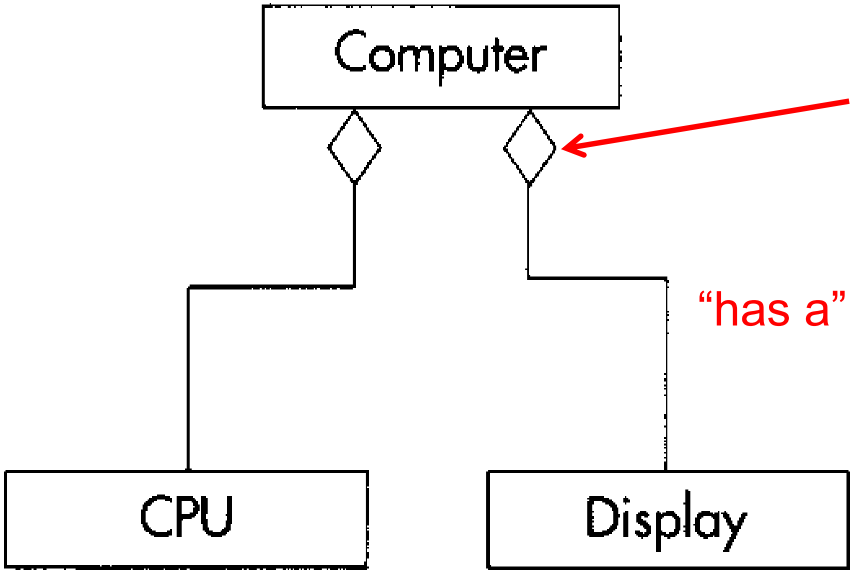
Links replace associations.

- No multiplicity.
- Mulitiple instances possible.

Object Diagram Example



Aggregation



Open diamond indicates aggregation

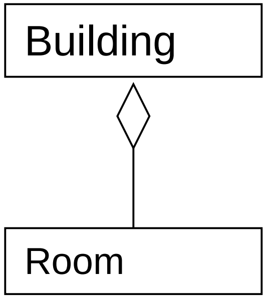
“has a” relationship

Says Class Computer “has a” Display.
Display “is a part of” Computer.

Aggregation

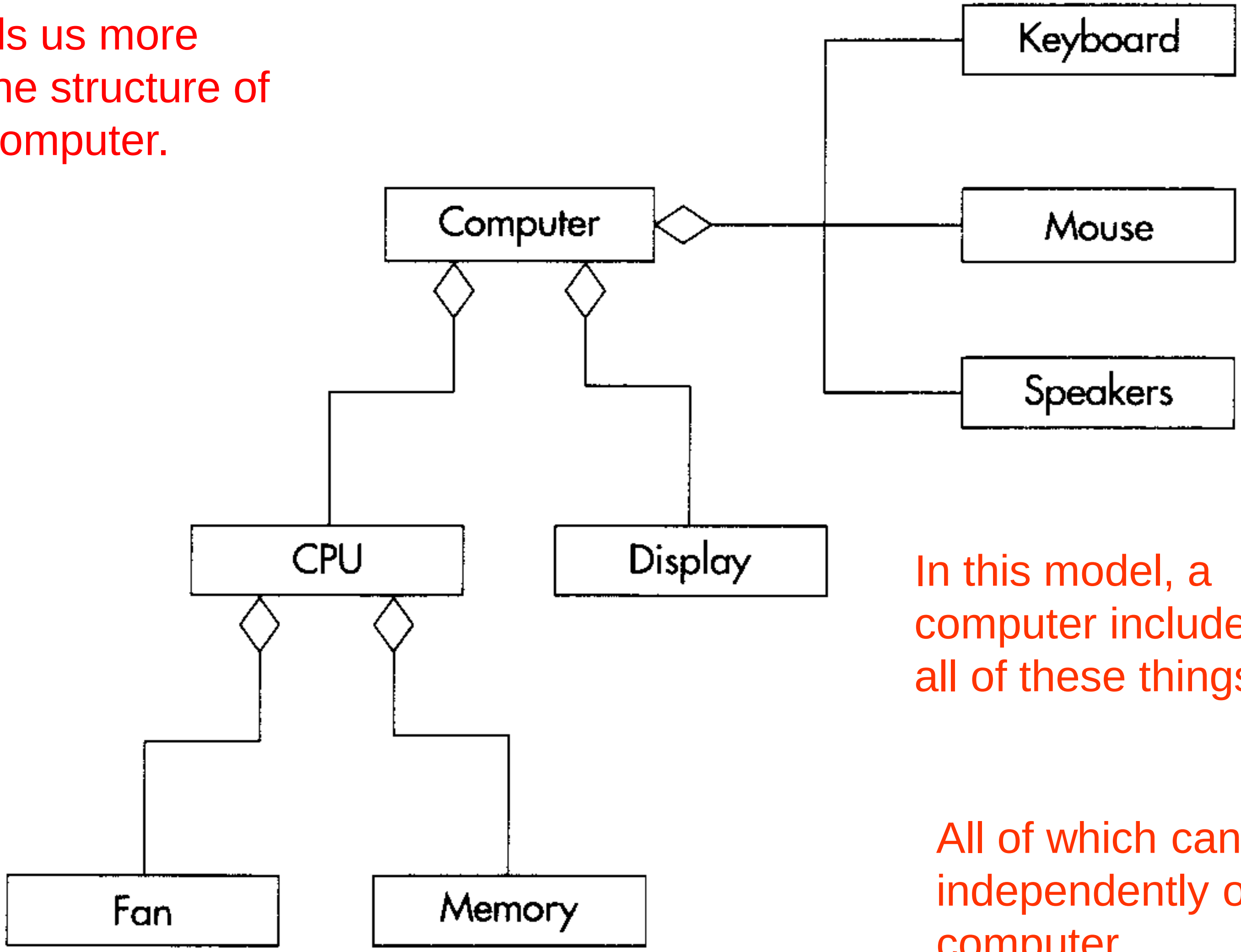
Aggregation implies that *(in this model)* the parts can exist on their own – independently of the class that they are a part of.

For example, you probably would not say:



Aggregation

This tells us more about the structure of class Computer.



In this model, a computer includes all of these things

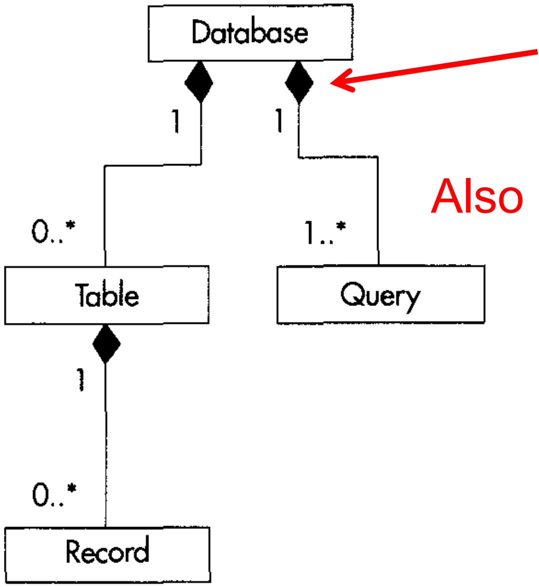
All of which can exist independently of the computer

Aggregation

- An aggregation implies that *(in this model)* objects of the “parts” classes can live on after the object that they are a part of is destroyed.

Composition

We use the *composition* association to say that the parts cannot exist independently of the whole



A *filled* diamond denotes composition.

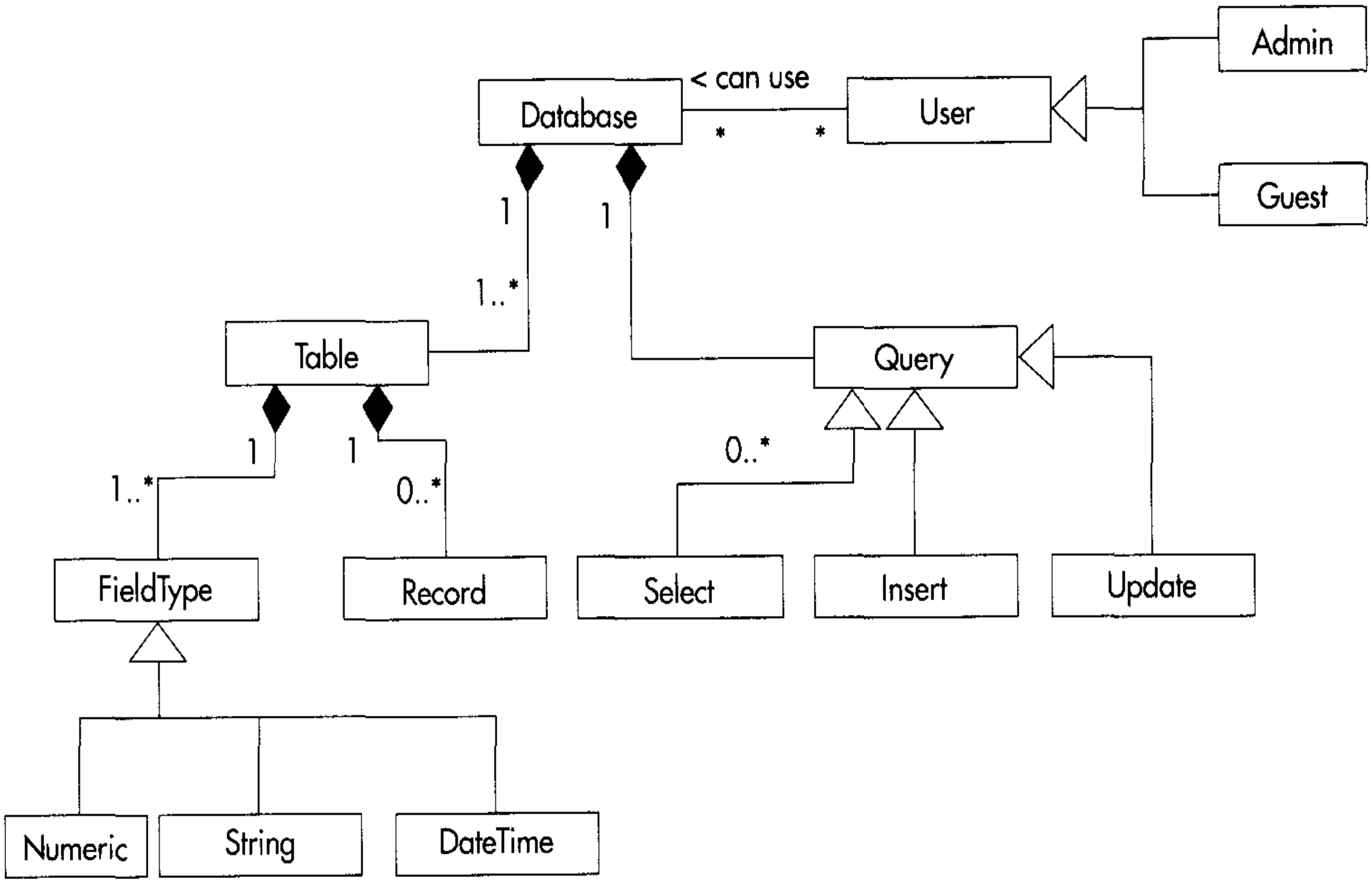
Also a “has a” relationship

This says that a Table consists of Records, and a Record cannot exist except as part of a table.

A Database consists of Tables and Queries, which cannot exist except as part of a Database.

Multiple Kinds of Associations

Aggregation and composition can be combined with other associations.



How would you model this?

- A class schedule consists of a number of schedule entries.
- Each schedule entry says when and where a class will be taught, and who will teach it.
- A class is identified by prefix, number, and section.
- Each class has a descriptive title
- Each instructor will be scheduled to teach one or more classes.

How do we discover classes?

- Look at the application domain.
- Study the use cases.
- Nouns are candidates to be classes.
- Actors are candidates to be classes.
- For each candidate:
  - Create a class corresponding to the candidate.
  - Check whether it has something to do.

How do we discover classes?

- Review your use cases.
- Check if every use case has a home.
    - If not, create a class to be responsible for it.

- What information is needed in the program?
    - Check that every piece of information has home.
    - If not, create one.

How do we discover classes?

- Every fact, or piece of information, should belong to exactly one class.
  - All other classes should get it from the owner.

- If there is not a natural home for some significant piece of information, you might need to create a new class to be the owner.

Object Oriented Design

- Start by constructing a model in the application domain.
  - Real world objects, relationships
  - Capture what we know about the problem
- Transform the problem domain model into a *solution domain model*
  - Blueprint for the system that will be built.

The solution domain model is a transformation of the application domain model.