Class Diagram

5.1 What is UML?

The Unified Modelling Language is a standard graphical language for modelling object oriented software

- At the end of the 1980s and the beginning of 1990s, the first object-oriented development processes appeared
- The proliferation of methods and notations tended to cause considerable confusion
- Two important methodologists Rumbaugh and Booch decided to merge their approaches in 1994.
  - —They worked together at the Rational Software Corporation
- In 1995, another methodologist, Jacobson, joined the team
  - —His work focused on use cases
- In 1997 the Object Management Group (OMG) started the process of UML standardization

UML diagrams

- Class diagrams
  - —describe classes and their relationships
- Interaction diagrams
  - —show the behaviour of systems in terms of how objects interact with each other
- State diagrams and activity diagrams
  - —show how systems behave internally
- Component and deployment diagrams
  - —show how the various components of systems are arranged logically and physically

UML features

- It has detailed *semantics*
- It has *extension* mechanisms
- It has an associated textual language
    —*Object Constraint Language* (OCL)

The objective of UML is to assist in software development

—It is not a *methodology*

What constitutes a good model?

A model should

- use a standard notation
- be understandable by clients and users
- lead software engineers to have insights about the system
- provide abstraction

Models are used:

- to help create designs
- to permit analysis and review of those designs.
- as the core documentation describing the system.

5.2 Essentials of UML Class Diagrams

The main symbols shown on class diagrams are:

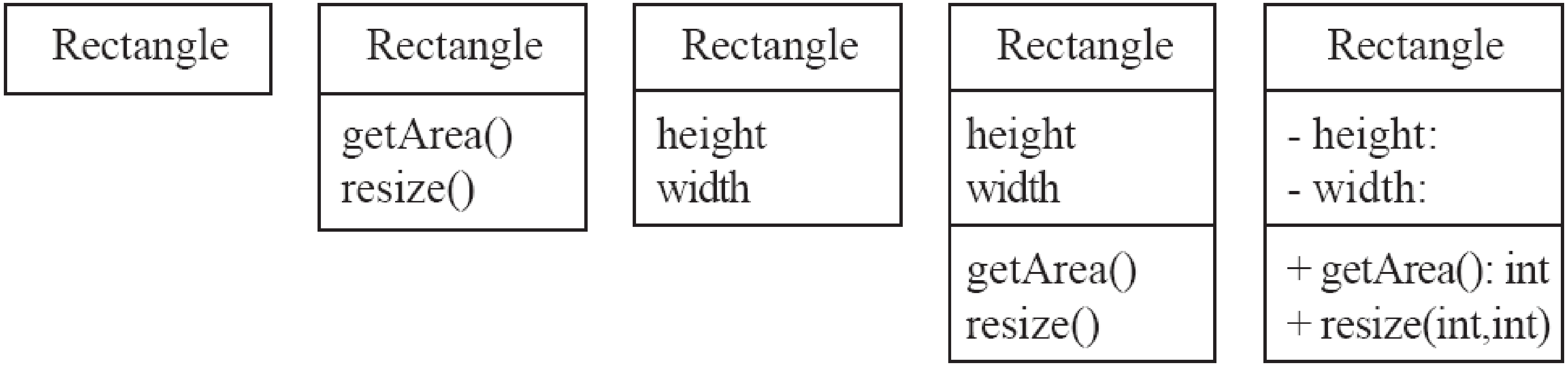
- *Classes*
  - represent the types of data themselves
- *Associations*
  - represent linkages between instances of classes
- *Attributes*
  - are simple data found in classes and their instances
- *Operations*
  - represent the functions performed by the classes and their instances
- *Generalizations* --group classes into inheritance hierarchies

Classes

A class is simply represented as a box with the name of the class inside

- The diagram may also show the attributes and operations
- The complete signature of an operation is:

operationName(parameterName: parameterType …): returnType



Rectangle

Rectangle
getArea()
resize()

Rectangle
height
width

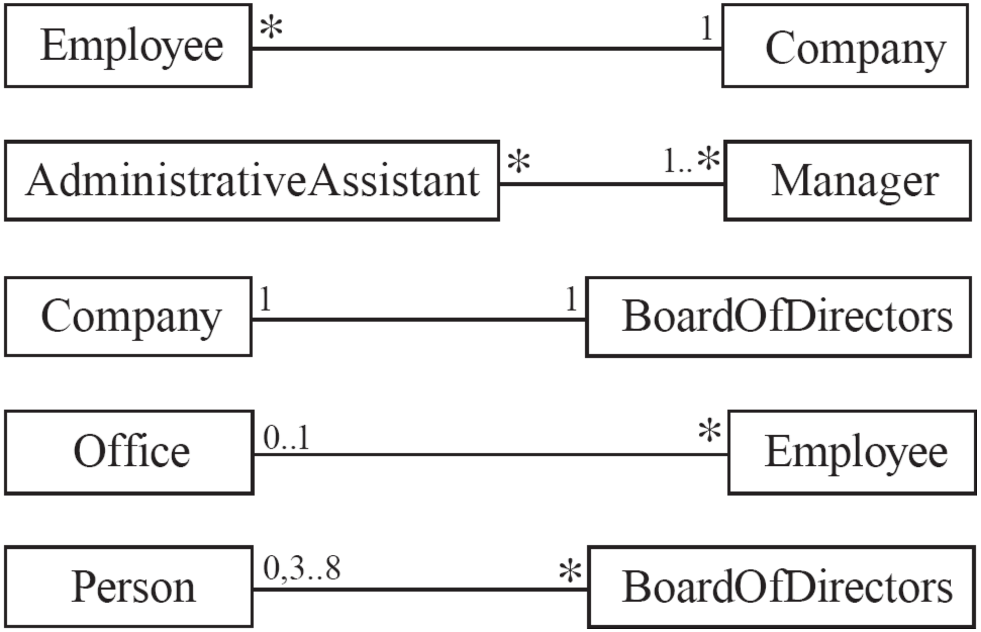
Rectangle
height
width
getArea()
resize()

Rectangle
- height:
- width:
+ getArea(): int
+ resize(int,int)

5.3 Associations and Multiplicity

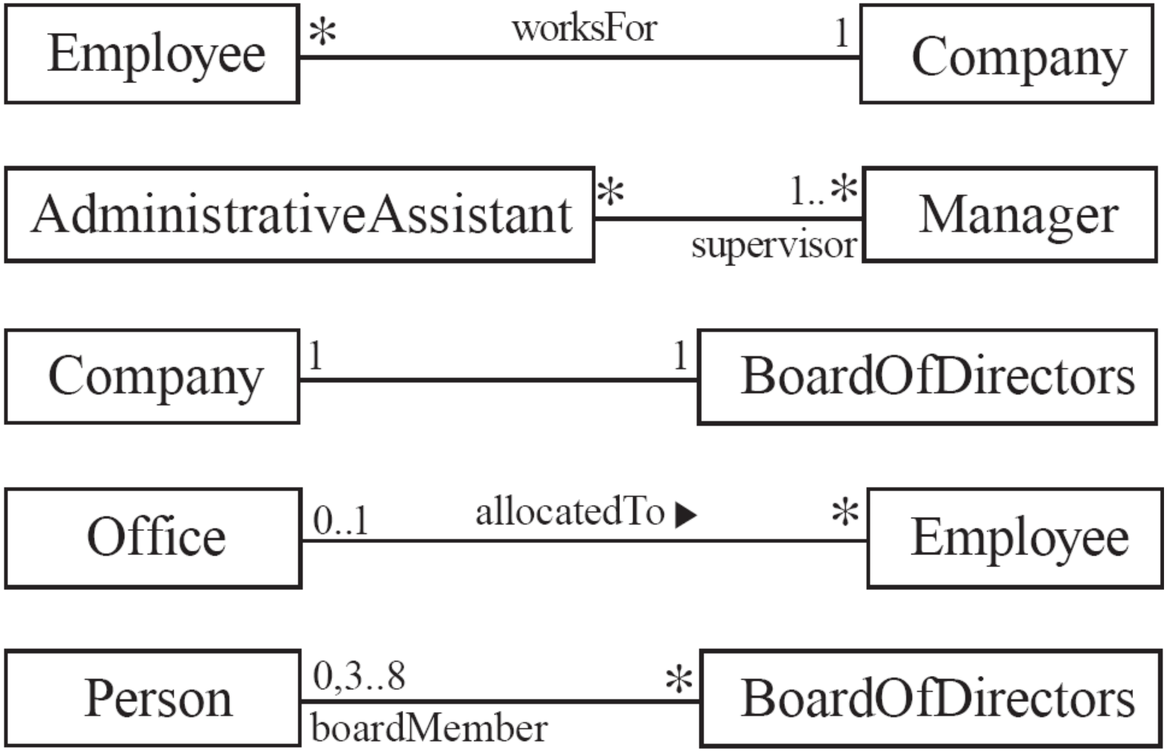
An *association* is used to show how two classes are related to each other

- Symbols indicating *multiplicity* are shown at each end of the association



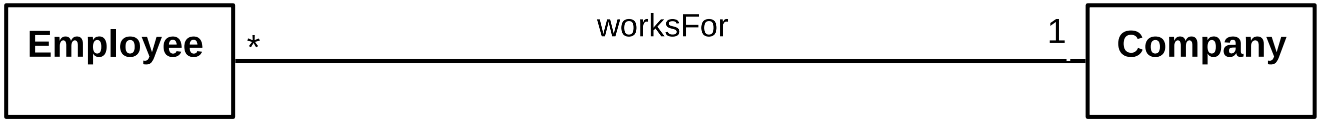
Labelling associations

- Each association can be labelled, to make explicit the nature of the association



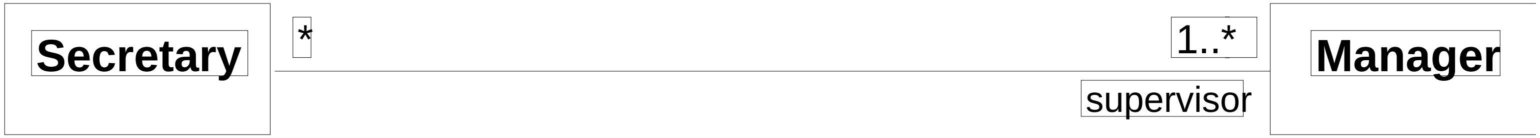
Analyzing and validating associations

- **Many-to-one**
  - —A company has many employees,
  - —An employee can only work for one company.
    - This company will not store data about the moonlighting activities of employees!
  - —A company can have zero employees
    - E.g. a ‘shell’ company
  - —It is not possible to be an employee unless you work for a company



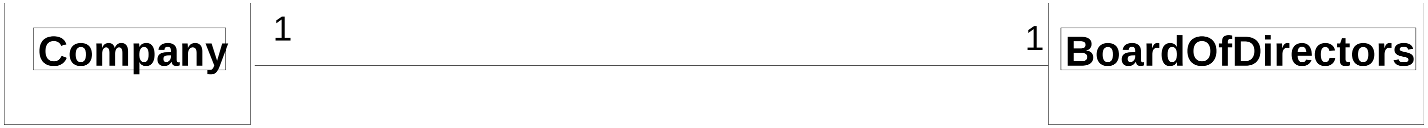
Analyzing and validating associations

- **Many-to-many**
  - —A secretary can work for many managers
  - —A manager can have many secretaries
  - —Secretaries can work in pools
  - —Managers can have a group of secretaries
  - —Some managers might have zero secretaries.
  - —Is it possible for a secretary to have, perhaps temporarily, zero managers?



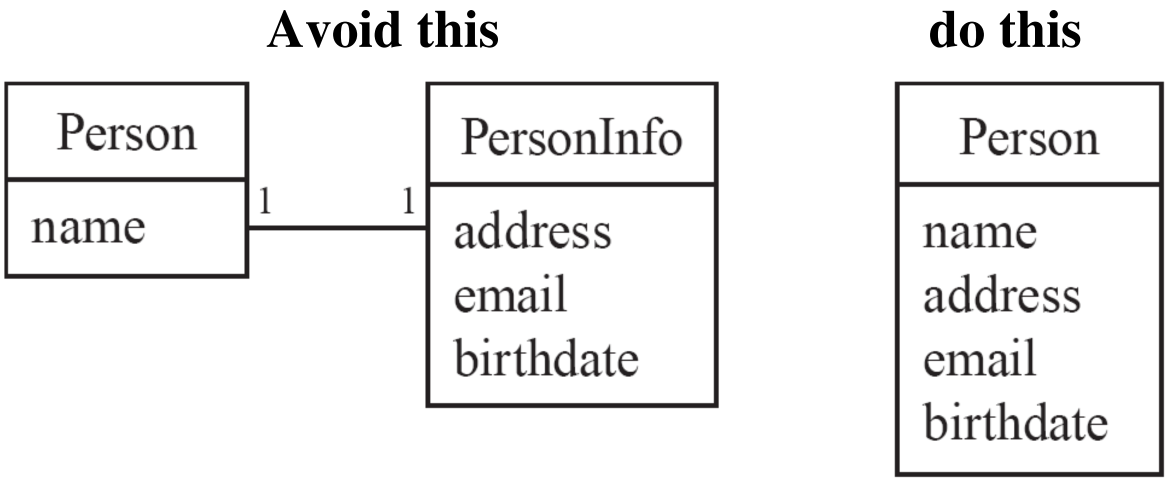
Analyzing and validating associations

- **One-to-one**
  - —For each company, there is exactly one board of directors
  - —A board is the board of only one company
  - —A company must always have a board
  - —A board must always be of some company



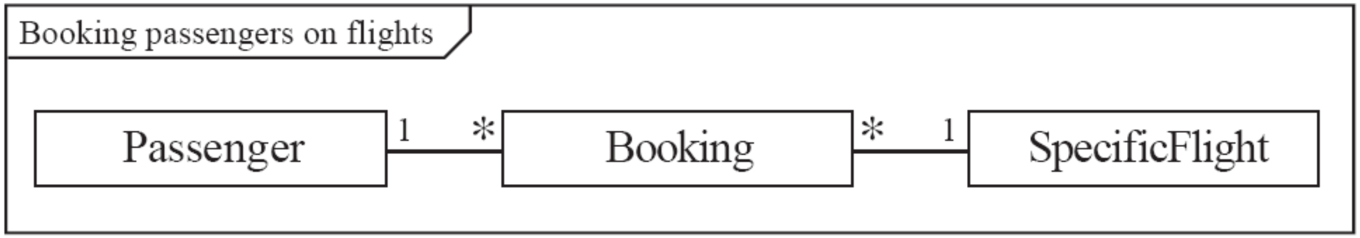
Analyzing and validating associations

Avoid unnecessary one-to-one associations



A more complex example

- A booking is always for exactly one passenger
  - —no booking with zero passengers
  - —a booking could *never* involve more than one passenger.
- A Passenger can have any number of Bookings
  - —a passenger could have no bookings at all
  - —a passenger could have more than one booking



Exercise

Create classes, associations, and multiplicities for the following situations.

a) Vehicles possessing wheels

Exercise

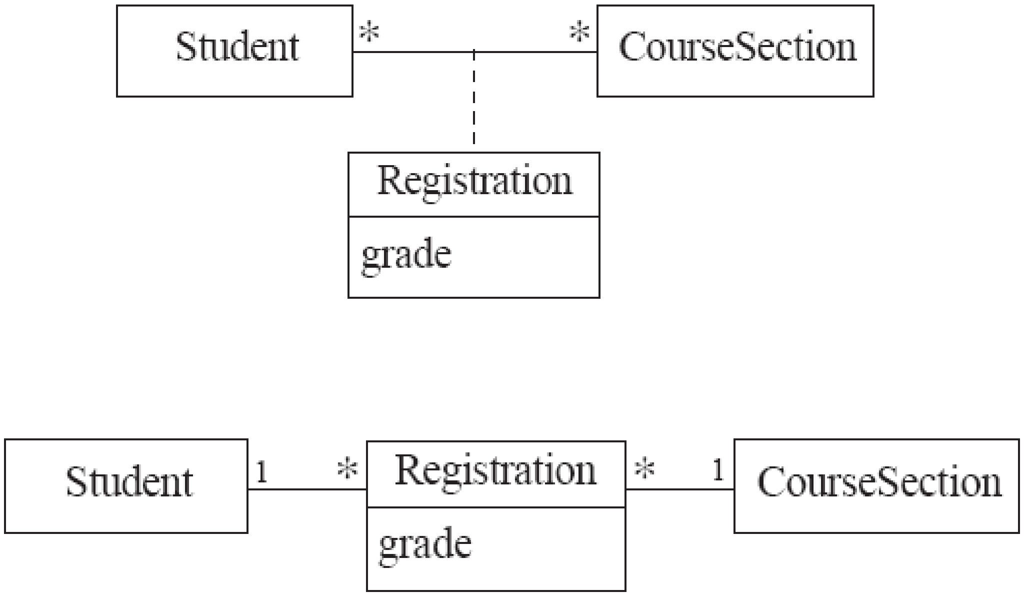
Create classes, associations, and multiplicities for the following situations.

a) Vehicles possessing wheels

b) A video rental shop, where you must be a member before renting something

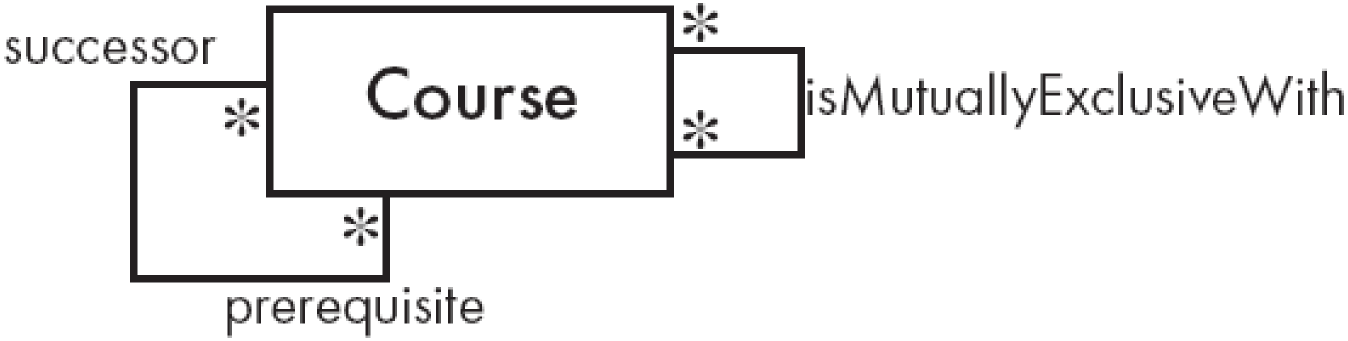
Association classes

- Sometimes, an attribute that concerns two associated classes cannot be placed in either of the classes
- The following are equivalent



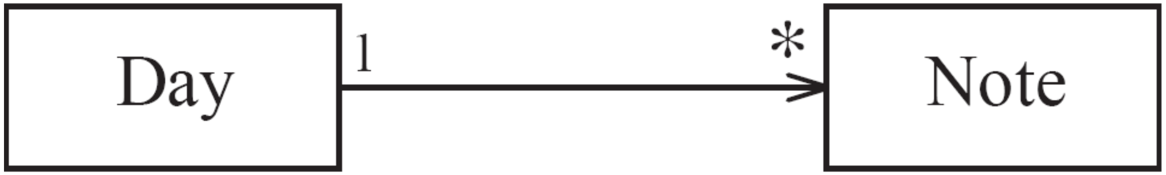
Reflexive associations

- It is possible for an association to connect a class to itself



Directionality in associations

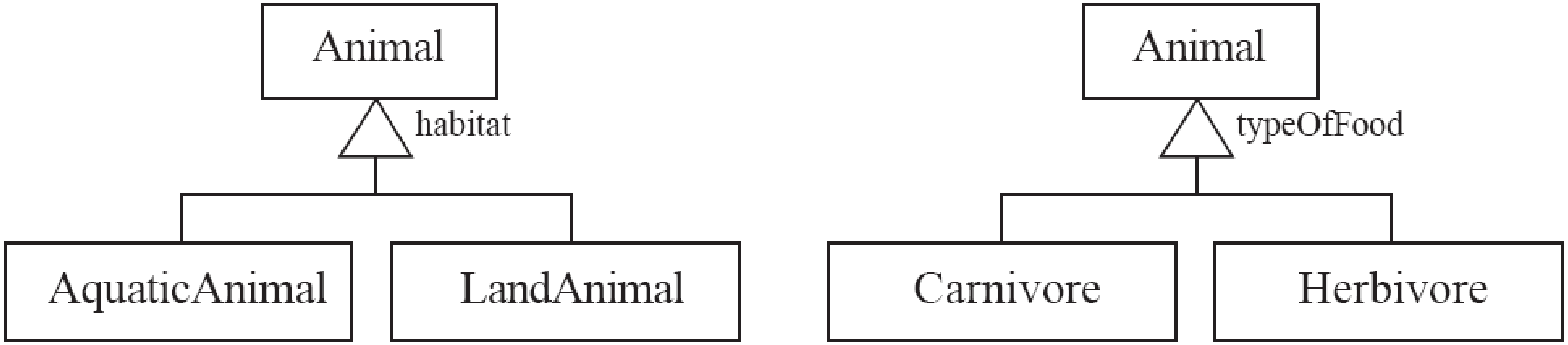
- **Associations are by default *bi-directional***
- **It is possible to limit the direction of an association by adding an arrow at one end**



5.4 Generalization

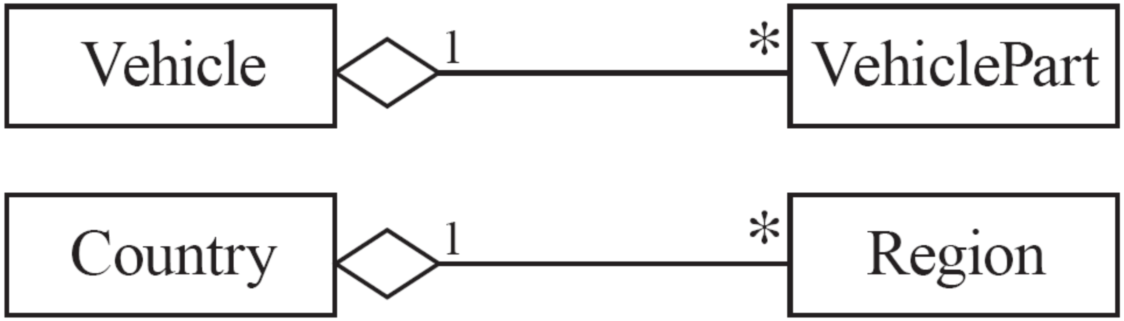
Specializing a superclass into two or more subclasses

- The *discriminator* is a label that describes the criteria used in the specialization



5.6 More Advanced Features: Aggregation

- Aggregations are special associations that represent ‘part-whole’ relationships.
  - —The ‘whole’ side is often called the *assembly* or the *aggregate*
  - —This symbol is a shorthand notation association named **`isPartOf`**



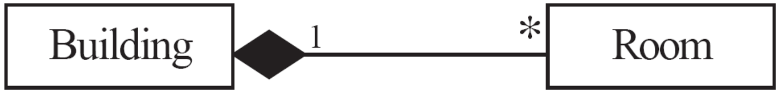
When to use an aggregation

As a general rule, you can mark an association as an aggregation if the following are true:

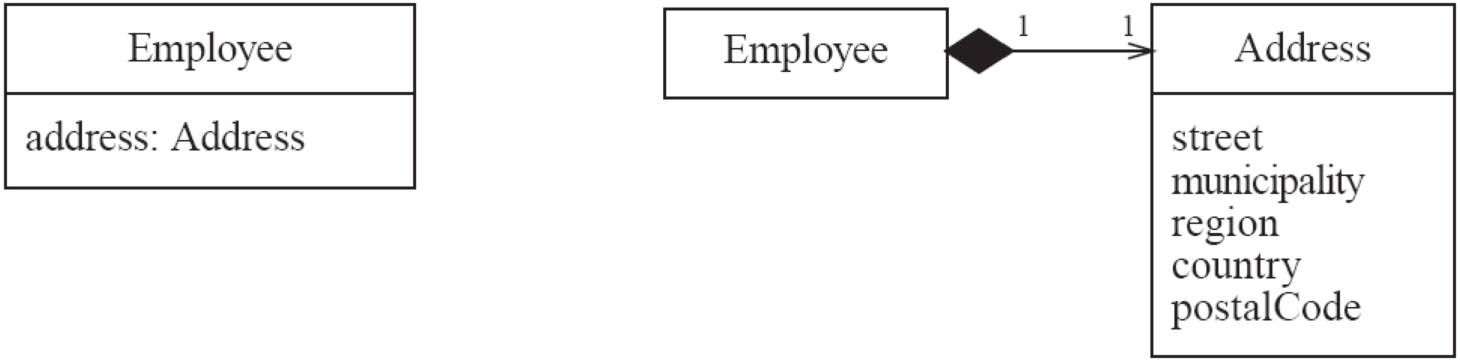
- You can state that
  - —the parts ‘are part of’ the aggregate
  - —or the aggregate ‘is composed of’ the parts
- When something owns or controls the aggregate, then they also own or control the parts

Composition

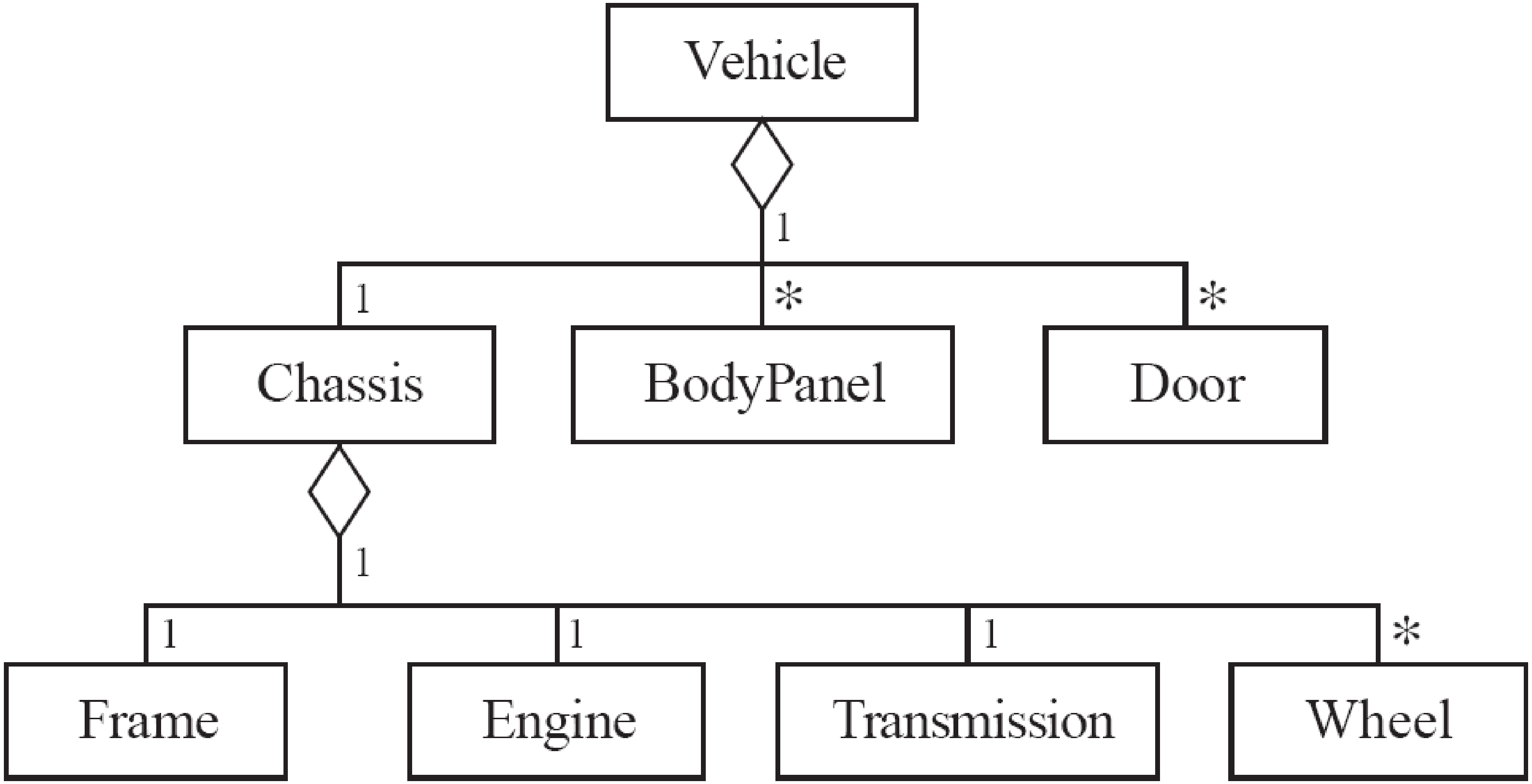
- A *composition* is a strong kind of aggregation
  - —if the aggregate is destroyed, then the parts are destroyed as well



- Two alternatives for addresses



Aggregation hierarchy


Vehicle
1
1
*
*
Chassis
BodyPanel
Door
1
1
1
1
*
Frame
Engine
Transmission
Wheel