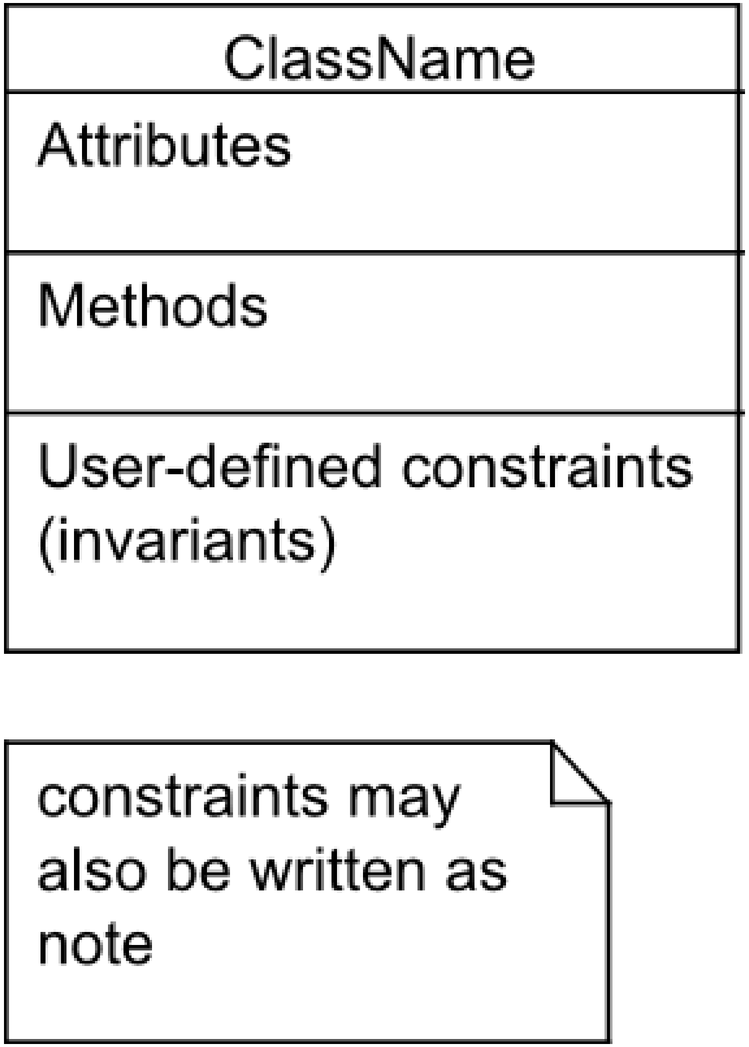
Class Diagrams

Class Diagram

- Classes consist of
  - the class name
    - written in BOLD
  - “features”
    - attributes and methods
  - user-defined constraints
- Note that class diagrams contain only classes, not objects.



ClassName

Attributes

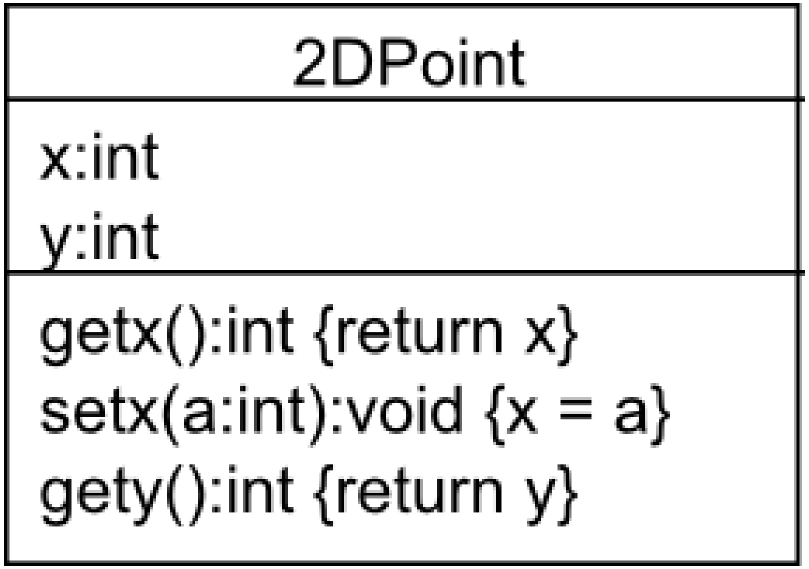
Methods

User-defined constraints (invariants)

constraints may also be written as note

Class Example

- Here is a concrete example of a class called Point, which depicts a 2D point.
- There are no constraints (yet...)
- A class name is written in UpperCamelCase



2DPoint

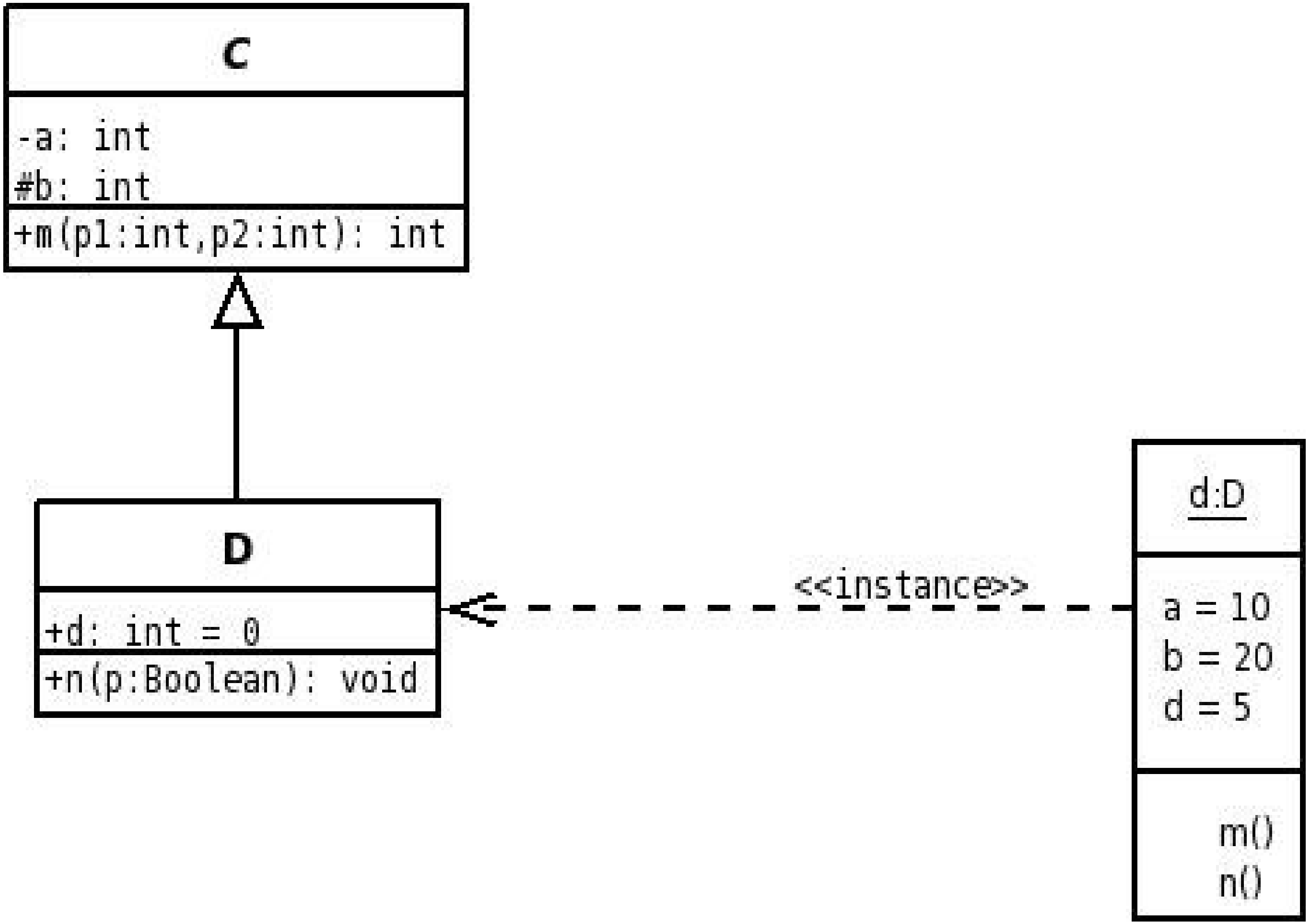
x:int
y:int

getx():int {return x}
setx(a:int):void {x = a}
gety():int {return y}

Visibility

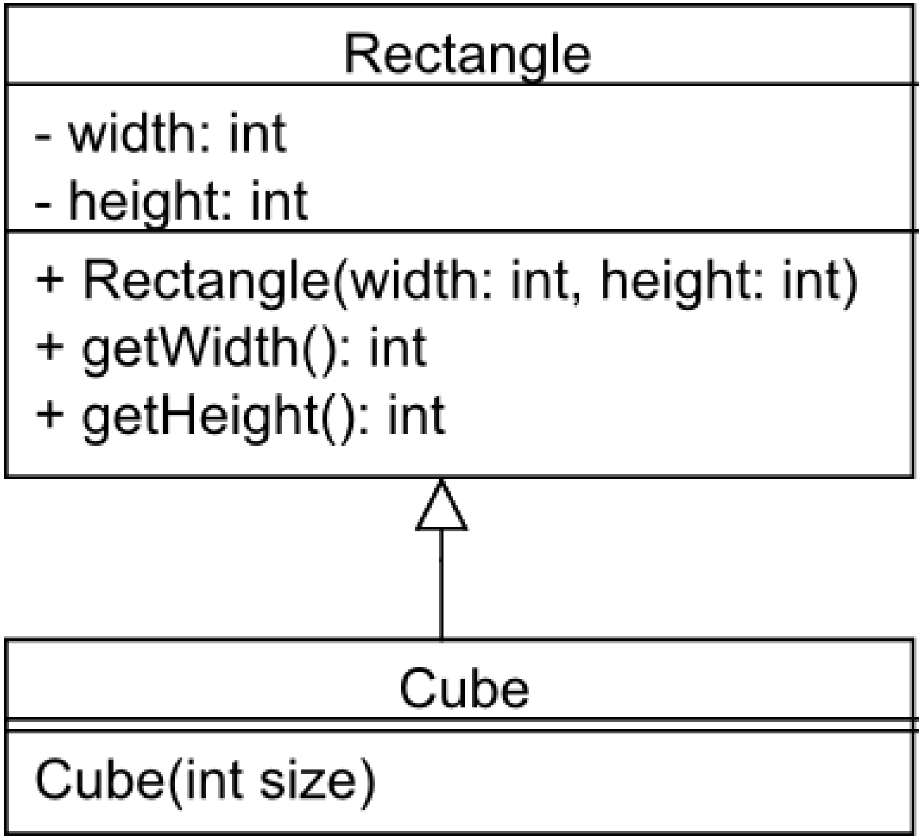
- A set of prefixes for attributes and methods
    - + public – visible to instance of any class
    - # protected – visible to instances of any subclass
    - – private – visible only to instances of the class itself
    - ~ package – visible to instance of any class within enclosing package
- Visibility is a class feature. It is found only in class diagrams and is enforced statically (at compile-time).

Visibility


C
-a: int
#b: int
+m(p1:int,p2:int): int
D
+d: int = 0
+n(p:Boolean): void
<<instance>>
d:D
a = 10
b = 20
d = 5
m()
n()

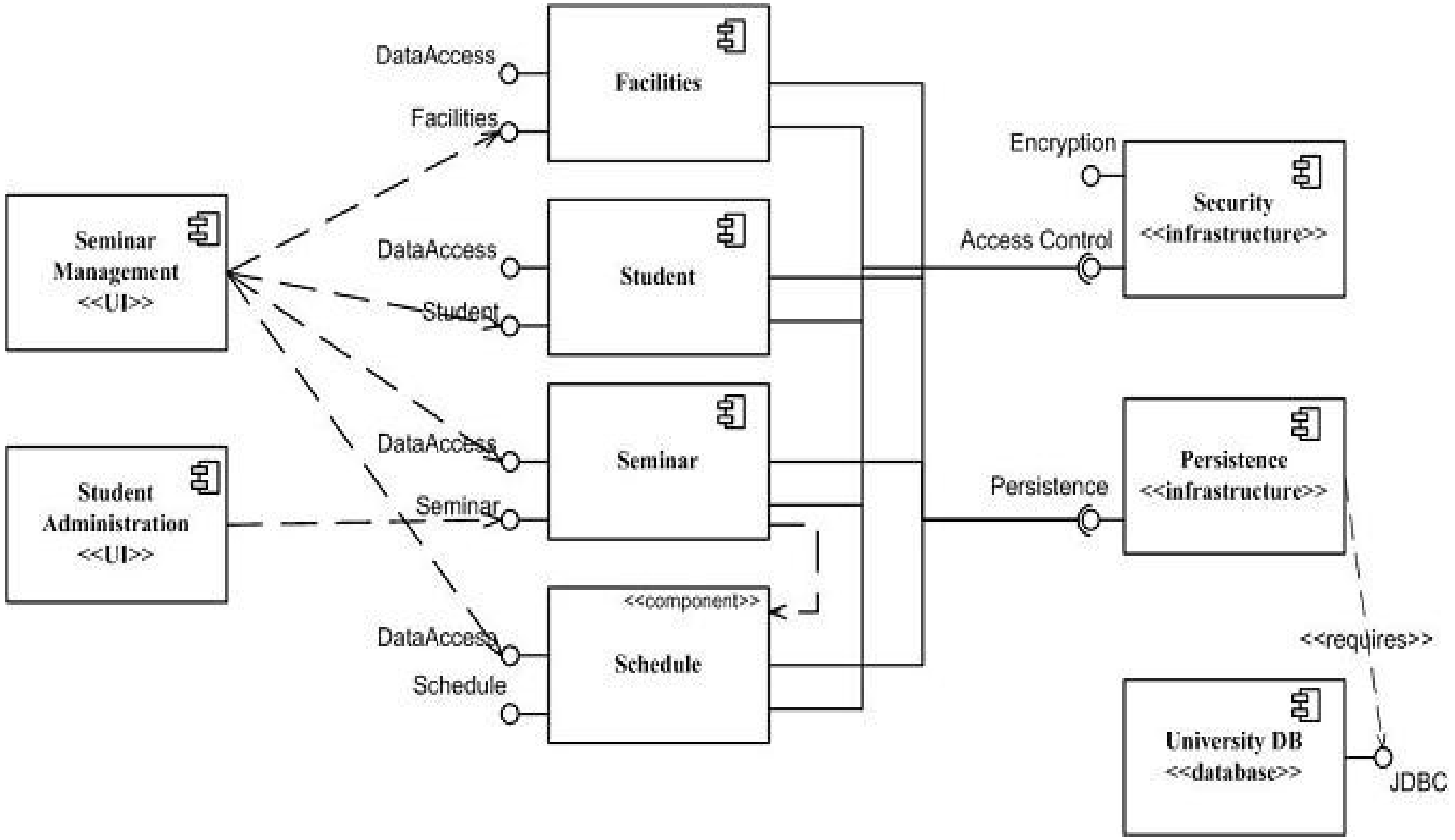
Inheritance

- In UML, inheritance syntax: a line with a hollow arrow.
- In this case, Cube ***is a*** Rectangle (good design).



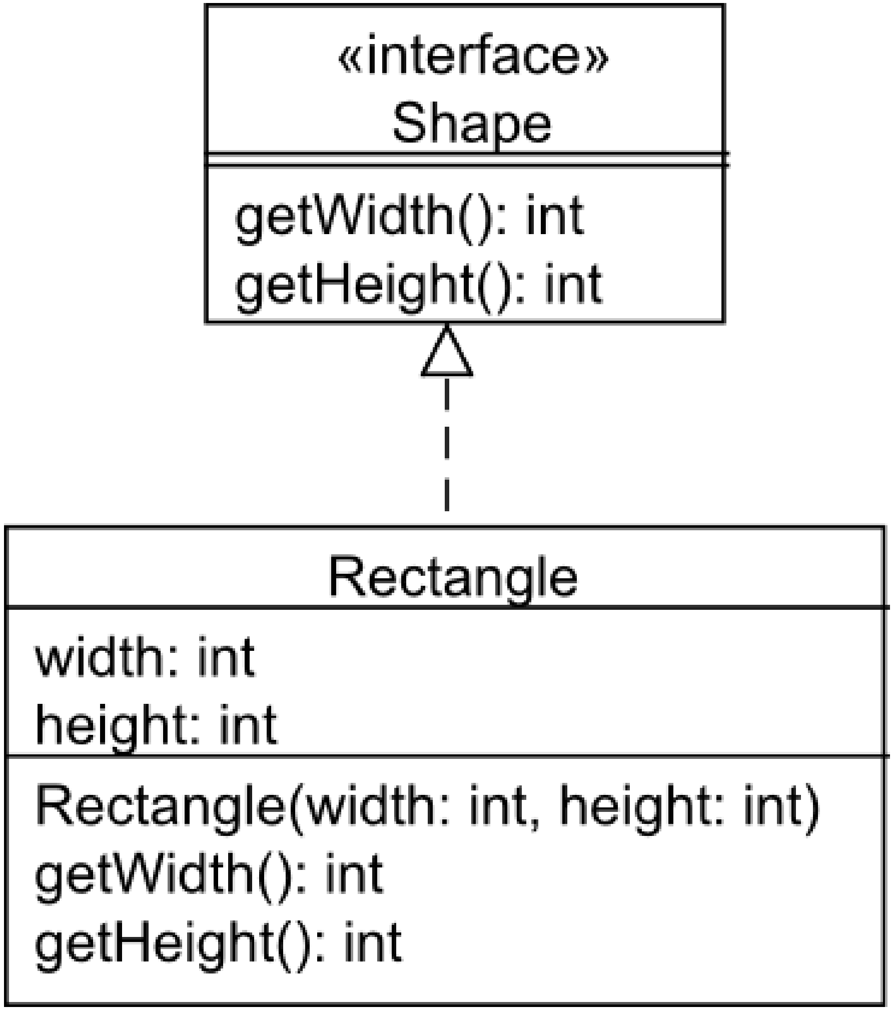
Interfaces

- In UML, interfaces are used to represent require/provide relationships



Interfaces

- Interfaces allow specification of a realization of requires/provide relation.
- Interfaces describe a contract between the class and the outside world.
- This contract is enforced at build time by the compiler.
  - all methods defined by that interface must be implemented by the class.
- UML: <<interface>> stereotype



Note: stereotype and profile are UML's extension mechanisms

Abstract Members

- Abstract methods (in *italic*):

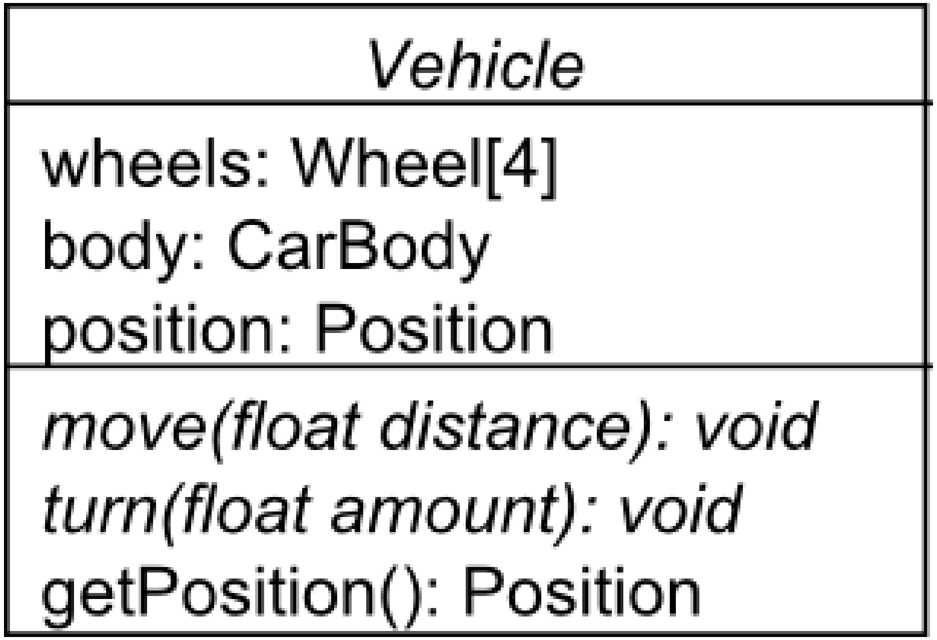
  no implementation given

  → can not instantiate

- Class with at least one abstract method:

  Abstract Class.

- Inherit from Abstract Class and implement the abstract methods.



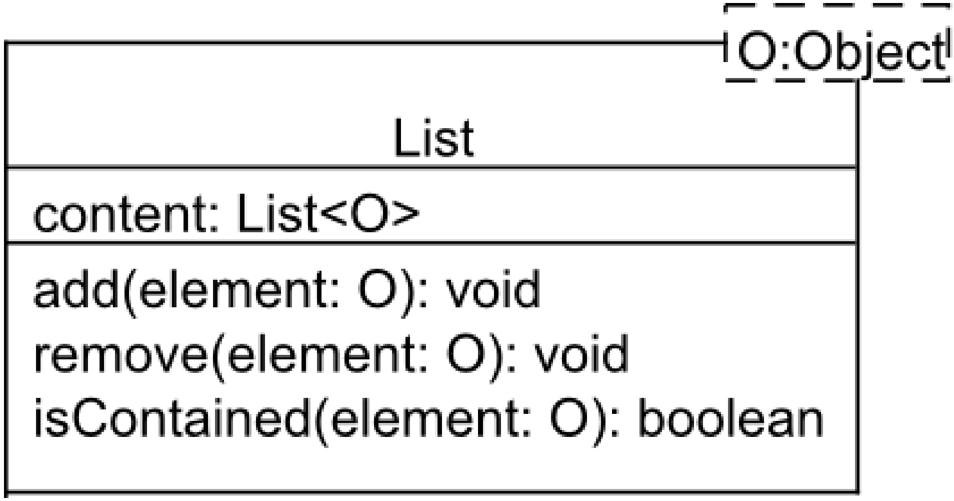
Vehicle

wheels: Wheel[4]
body: CarBody
position: Position

move(float distance): void
turn(float amount): void
getPosition(): Position

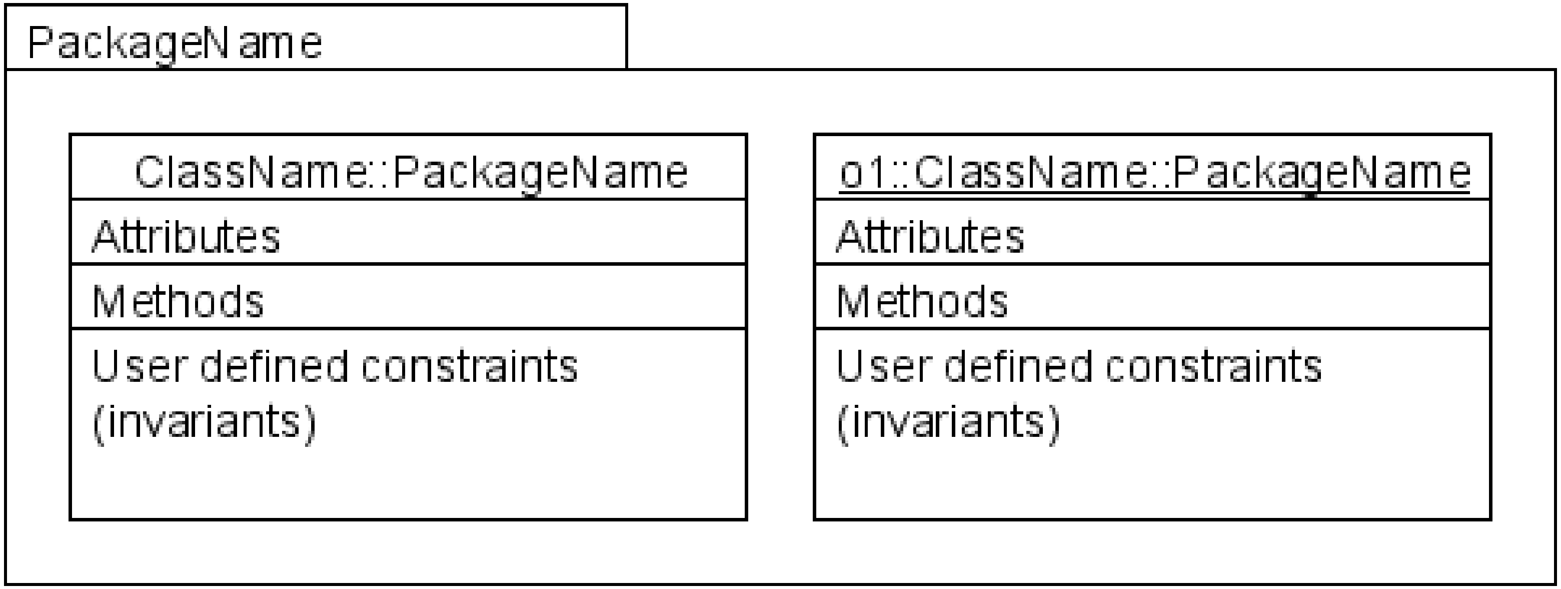
Templates

- As we saw with genericity, Templates are a mechanism to "parametrize" the types of objects in class/method definitions.
- In UML, they are defined with a box in the upper right corner of the class.



Package

- A package allows grouping model elements.
- Can be used for all UML constructs. Most common for Class Diagrams and Use Case Diagrams.
- Classes and objects in package have a prefix:
  - ClassName::PackageName
  - objectName:ClassName::PackageName
- A package may (hierarchically) contain other packages.



PackageName

ClassName::PackageName

Attributes

Methods

User defined constraints (invariants)

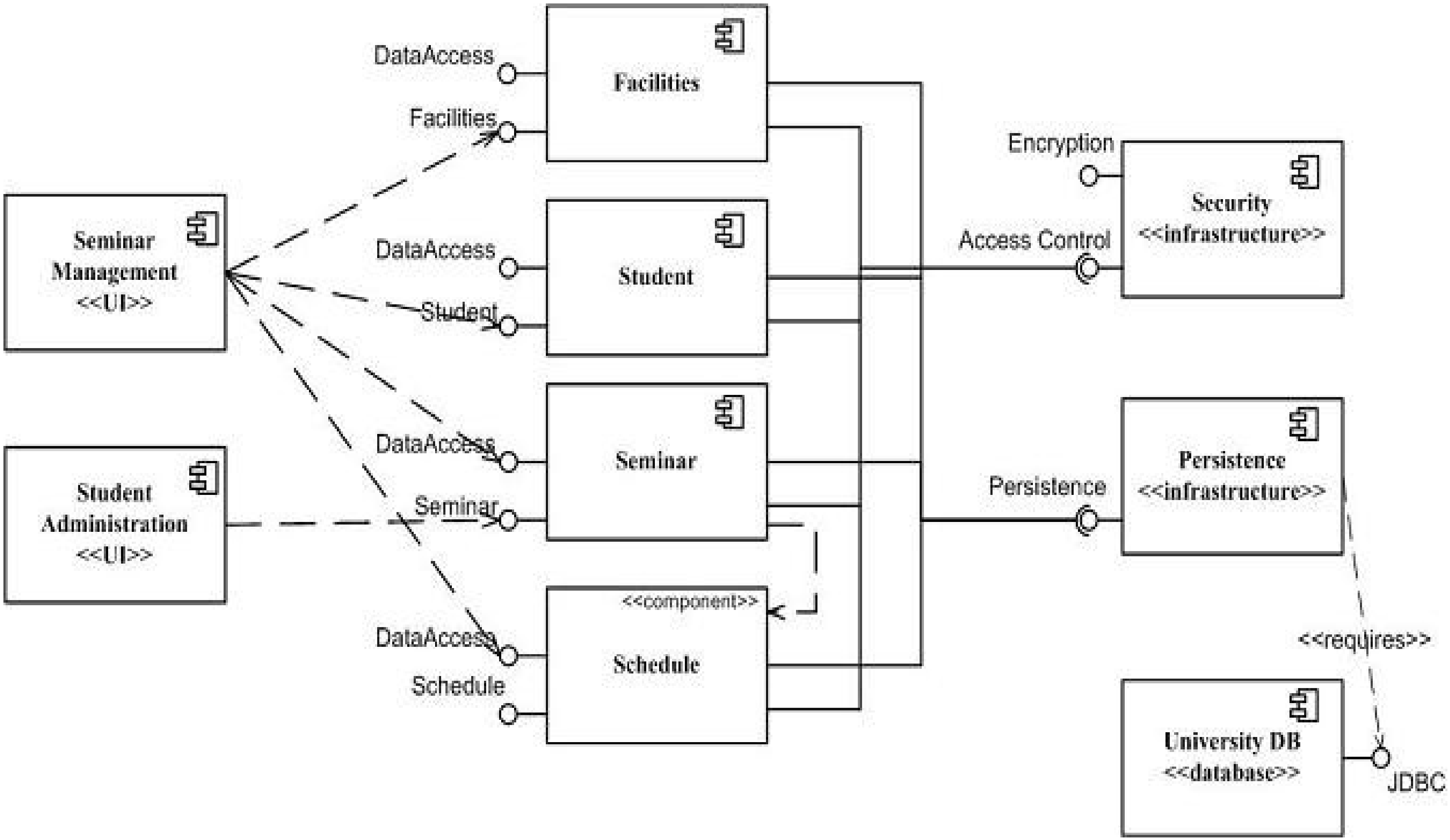
o1::ClassName::PackageName

Attributes

Methods

User defined constraints (invariants)

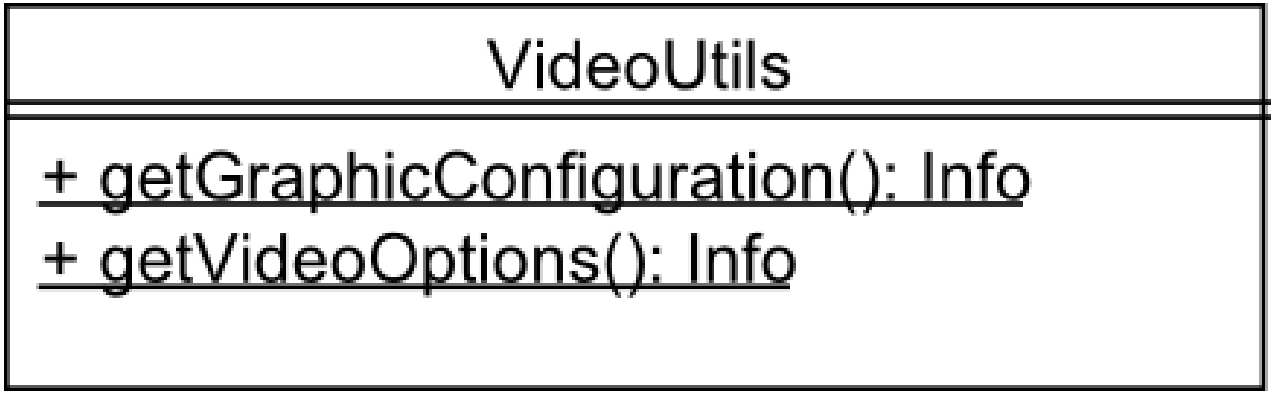
Components



DataAccess
Facilities
Facilities
DataAccess
Student
Student
DataAccess
Seminar
Seminar
<<component>>
DataAccess
Schedule
Schedule
Seminar Management <<UI>>
Student Administration <<UI>>
Encryption
Access Control
Security <<infrastructure>>
Persistence
Persistence <<infrastructure>>
<<requires>>
University DB <<database>>
JDBC

Class Attributes

- Class Attibutes aka static members (either attributes or methods) exist at the class level.
- Only one unique value across all instances of the class.
- They can be used without instantiating an object.
- UML syntax: underlined (why?).

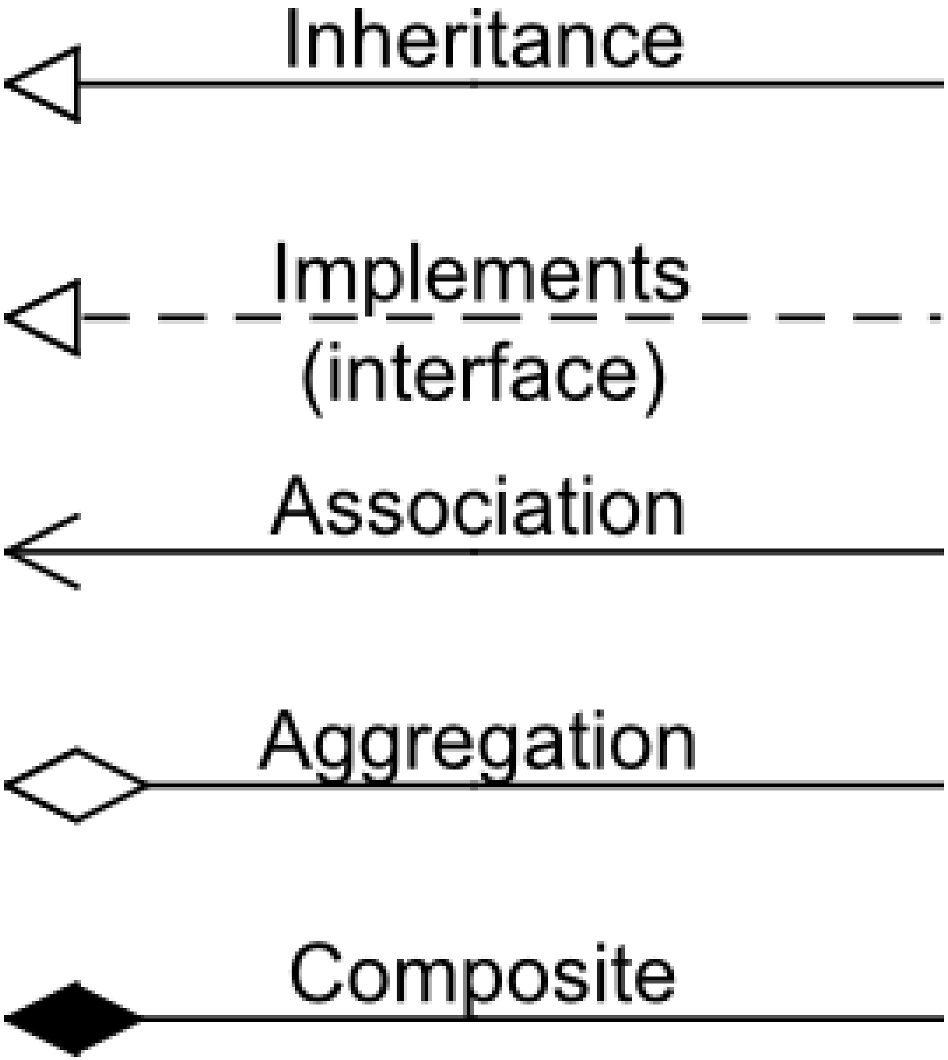


VideoUtils

+ getGraphicConfiguration(): Info
+ getVideoOptions(): Info

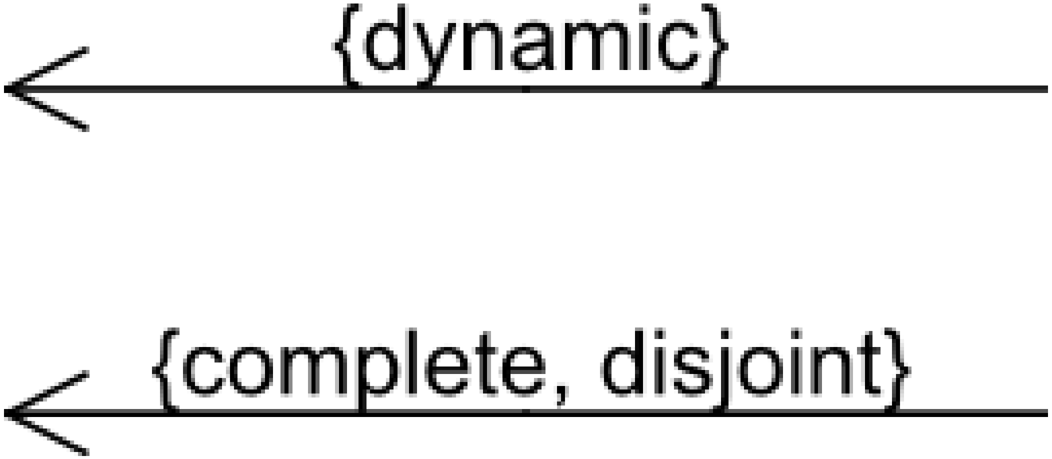
Arrows

- Arrows in UML can have different meaning.
- Hollow arrows describe an inheritance relation.
- Closed arrows are used to describe associations.
- Diamond arrows define composition relations.



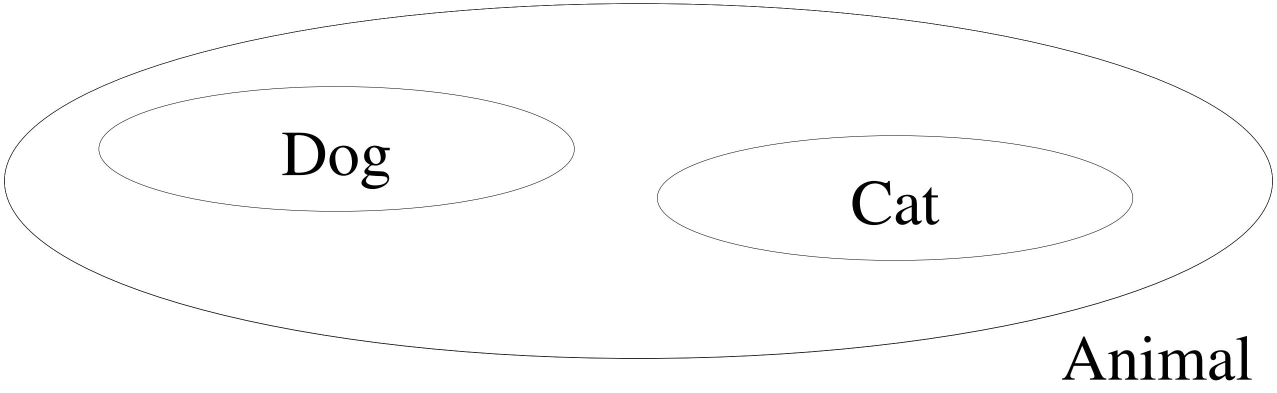
Annotations

- annotations put over arrows in class diagrams to specialize their meaning.

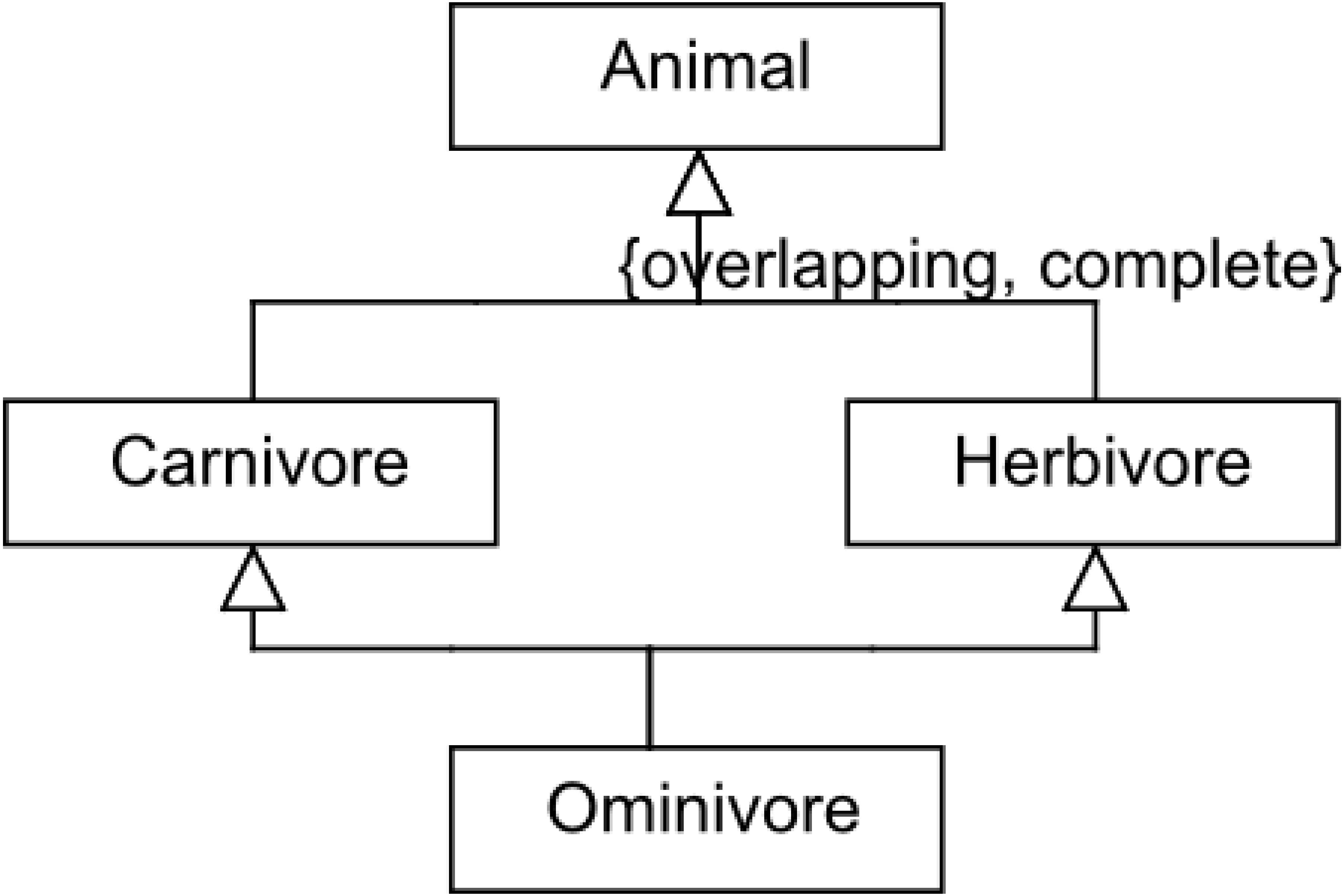


Special Inheritance Relations

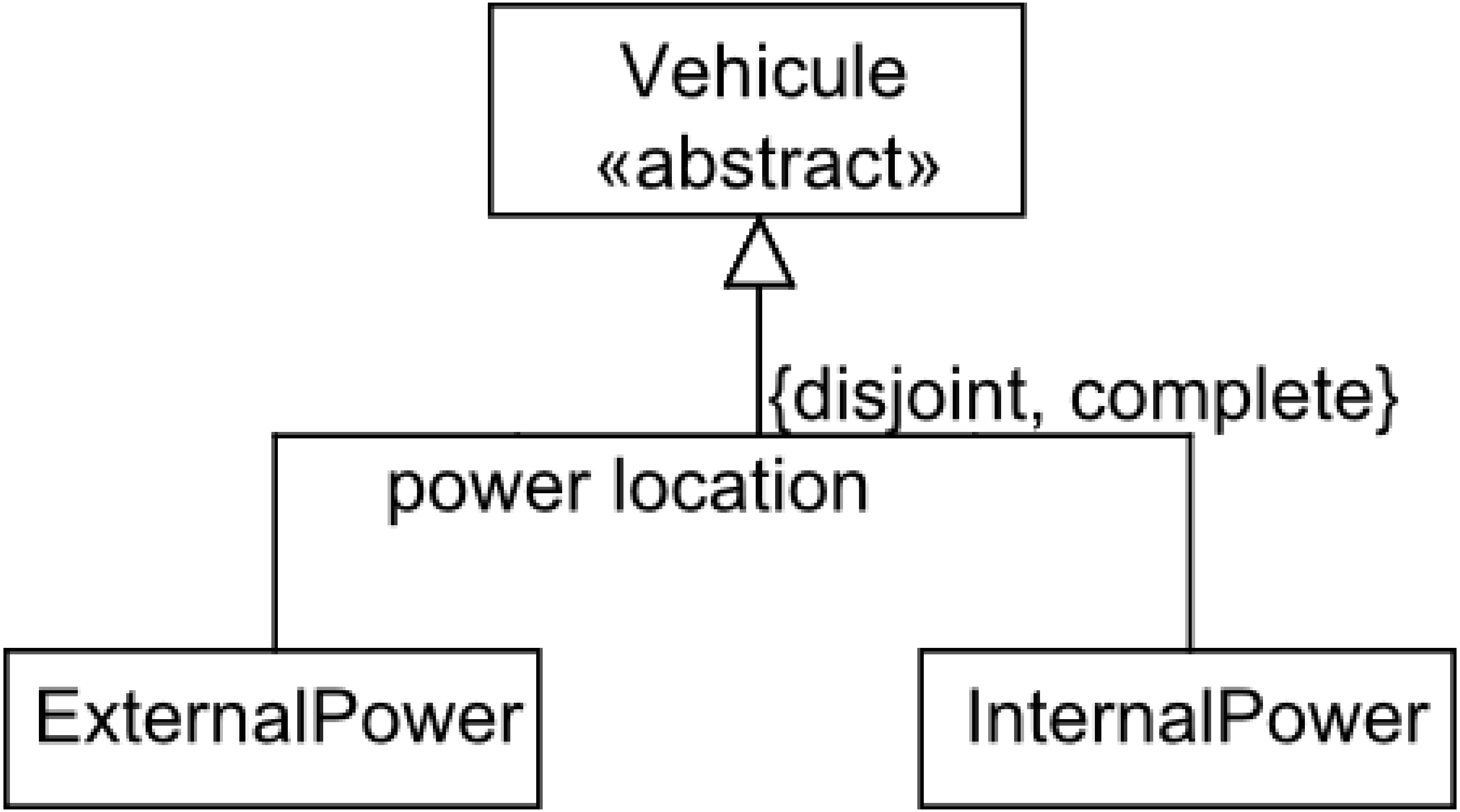
- We can use stereotype relations to better define the type of inheritance.
- The three attributes are
  - Disjoint or Overlapping
  - Complete or Non-complete
  - Dynamic or Static
- These attributes are best understood using set theory.



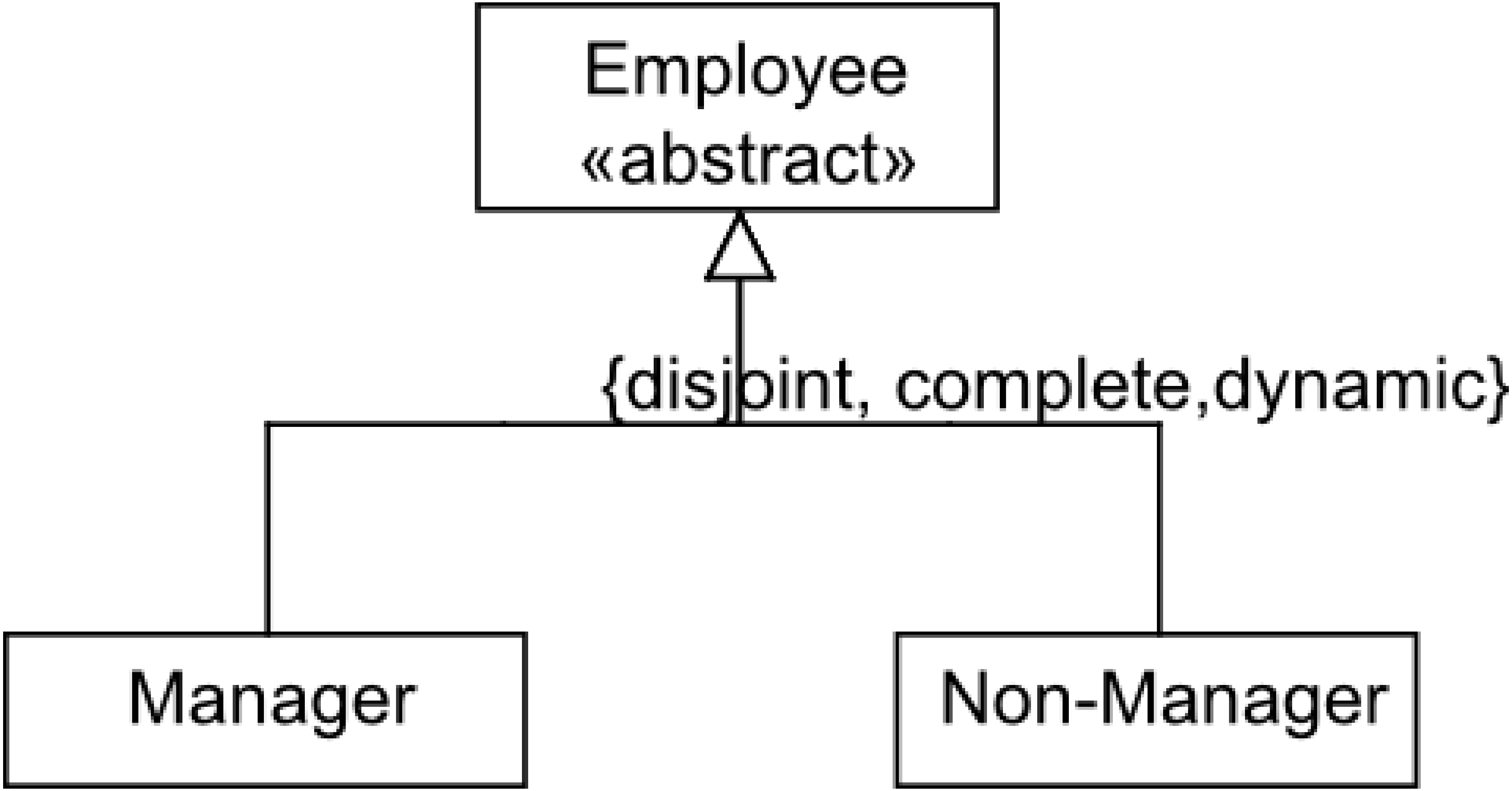
Animal Example



Vehicle Example



Employee Example



Implementation

- Employees can get promotion and become managers.
- Employees can get demoted and become non-managers.
- How would you implement this change?

Non-Manager to Manager

- How do we implement the dynamic change of a Non-Manager becoming a Manager?
- Option A: Create new object Manager. Copy fields. Destroy old object Non-Manager.
- Option B: Flag if Manager or not. So we only need an object employee and it contains all the attributes for Non-Manager and Manager.

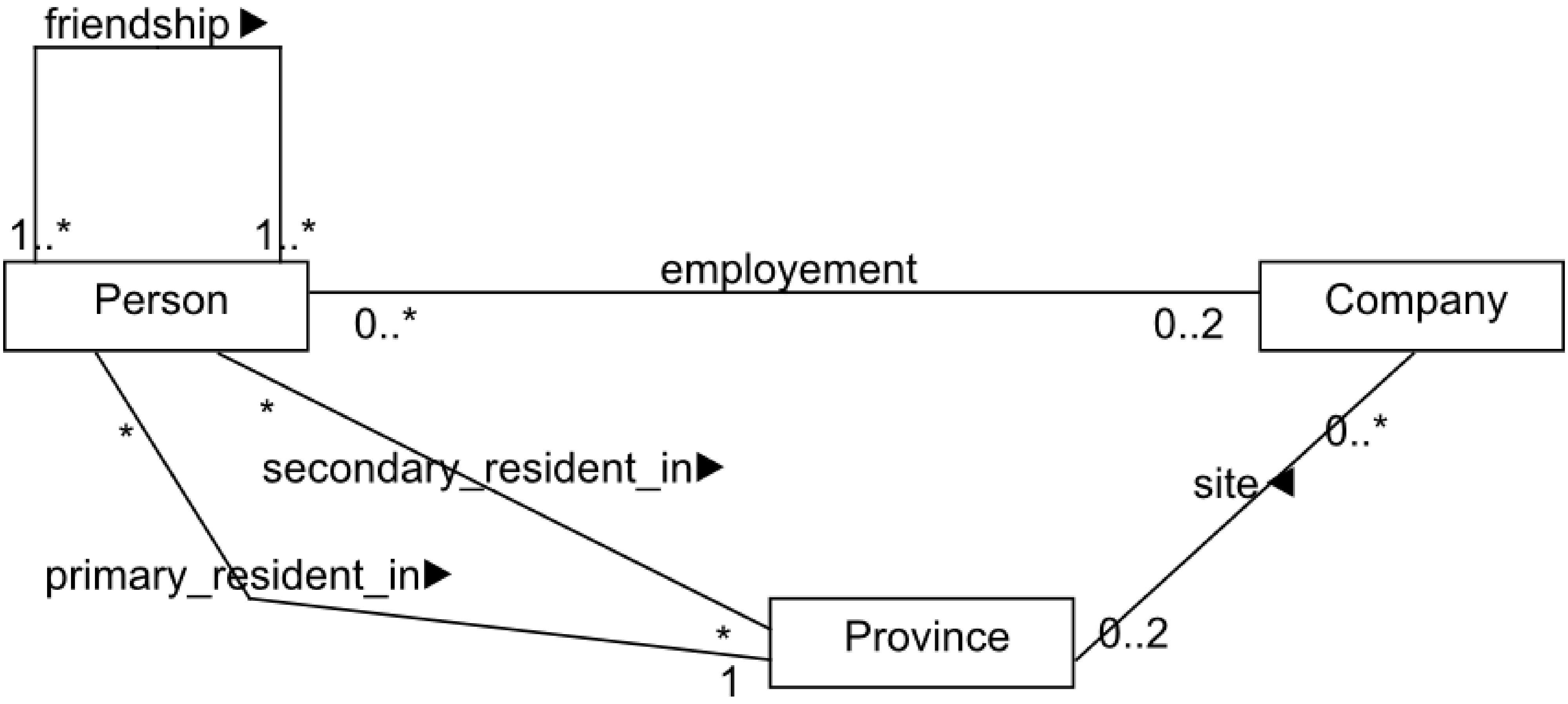
Association

- Associations describe which/how classes interact which each other.
  - You can give an association a name (always a noun)
    - A full black arrow next to a name indicates the direction the diagram can be read.
  - You can put roles at the end of connectors.
  - You can also put numbers to indicate cardinality.

Cardinality of Associations

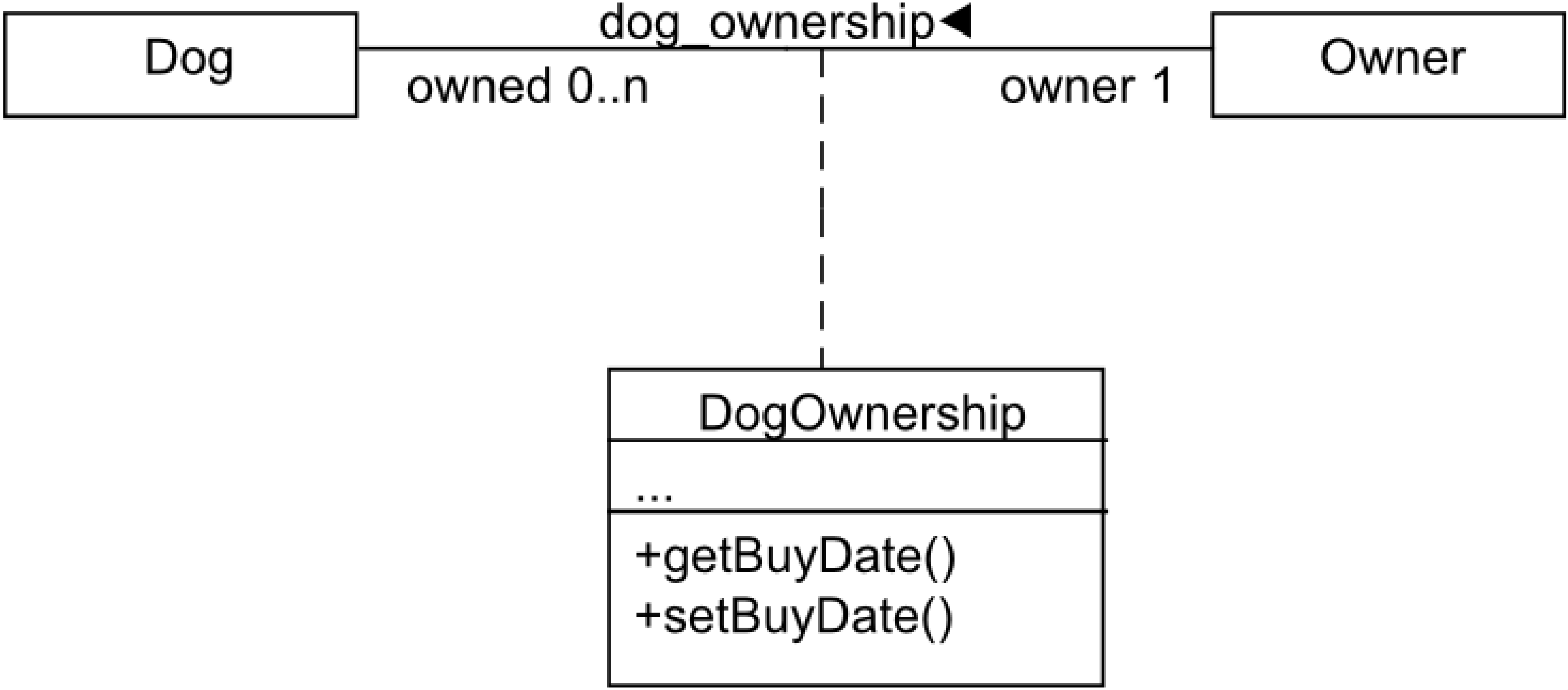
- One-to-one
- Many-to-one or One-to-many
- Many-to-many

Employee Example



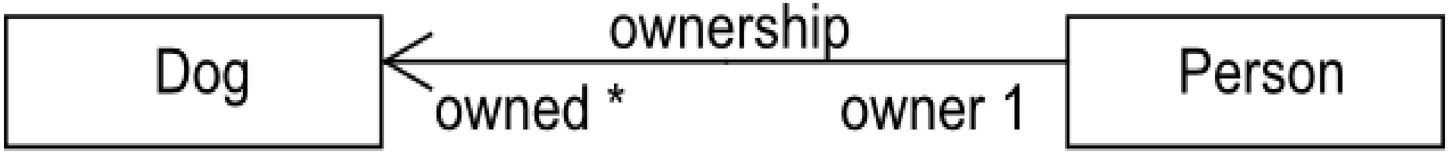
Association Class

- Used to represent associations which contain information.

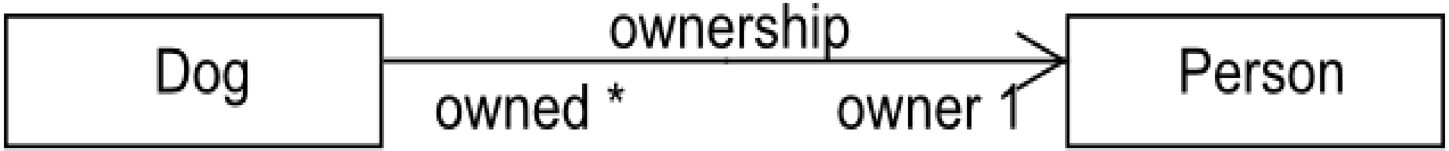


Navigation

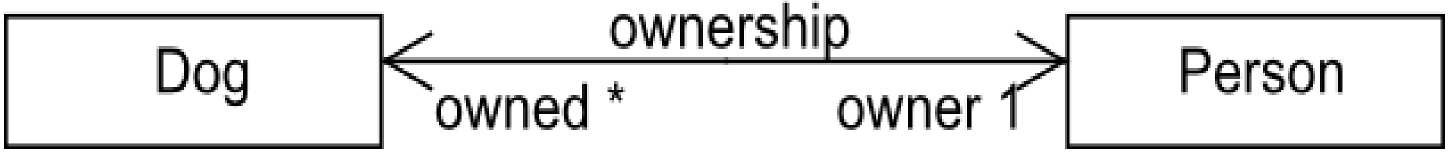
- Person object has a reference to dog object(s).



- Dog object has a reference to person object.



- Both objects have references to each other.



- This relates to performance (at the cost of space).

Association, in whole or in part

Two special types of associations exists:

- Composition
- Aggregation

Which is which?

Scenario 1

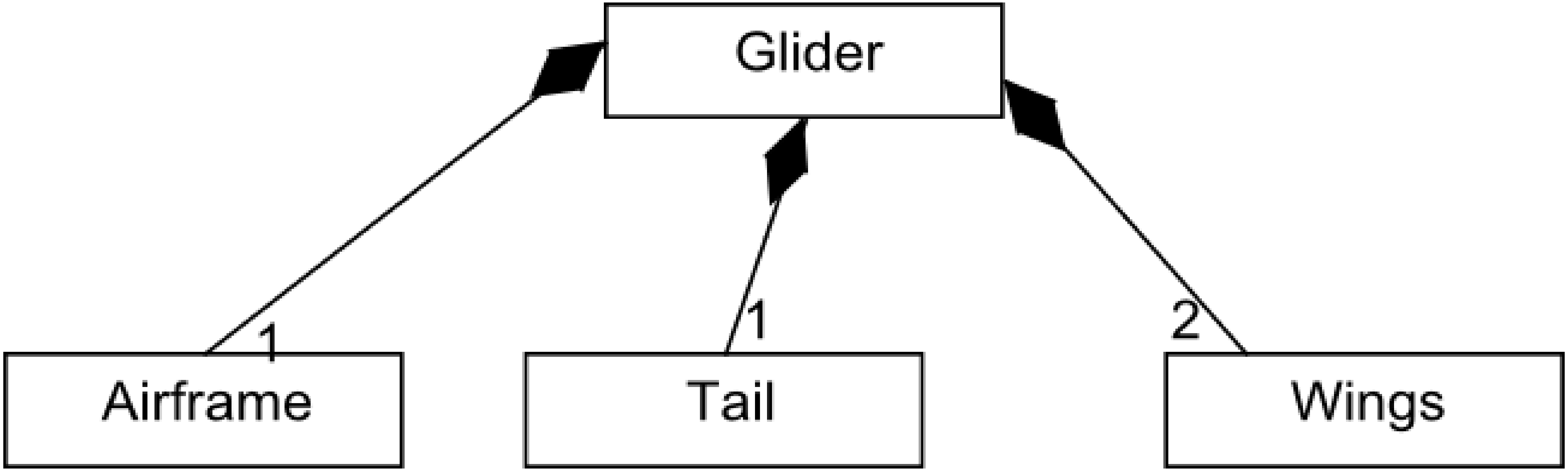
- You can find books in a bookcase.

Scenario 2

- You can find shelves in a bookcase.

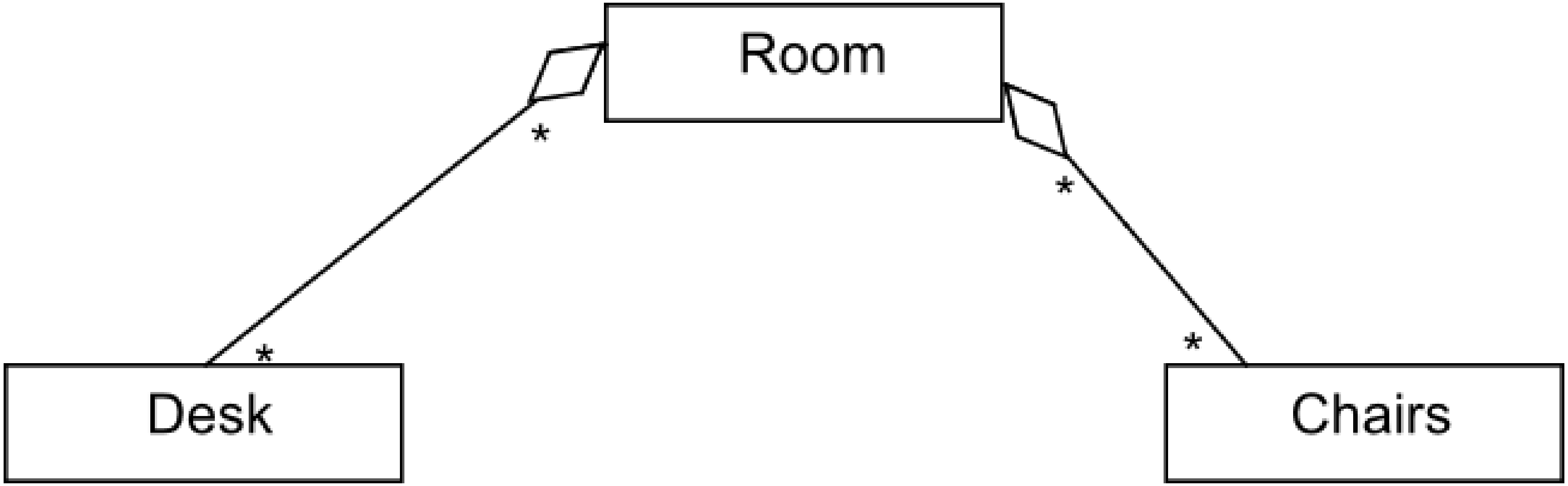
Composition

- A composite object does not exist without its components.
  - If we delete a composite object, we could use cascading-delete to remove it and its components.
- Often, components are of different types
- Each component is a part of a single composite.



Aggregation

- Aggregate (whole) object can exist without aggregands (parts).
- Objects may be part of multiple aggregates.
- Often, components are of the same type.


Room
*
*
*
*
Desk
Chairs