Class & Object Diagrams

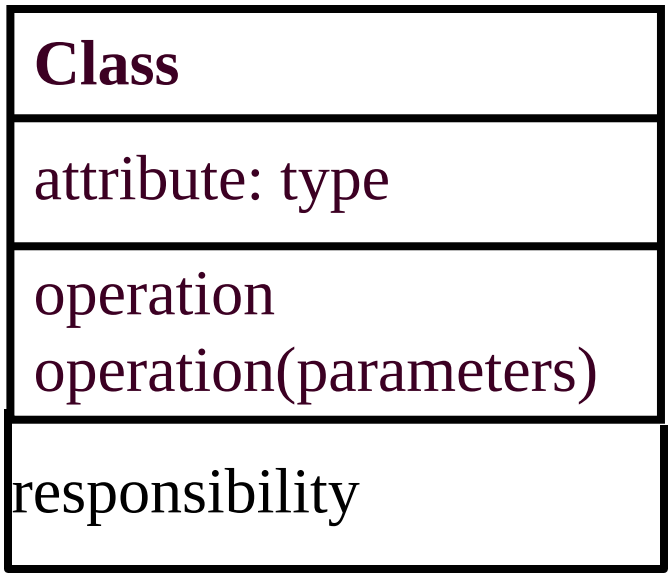
Software Design Methodology

UML Class Diagrams

- The heart of the UML
- Describes the classes in the system and the relations among them
- Supports most OO concepts
  - associations
  - aggregation
  - inheritance (including multiple)

Classes

- Classes are denoted by rectangles.
  - Attributes
  - Operations
  - Responsibilities



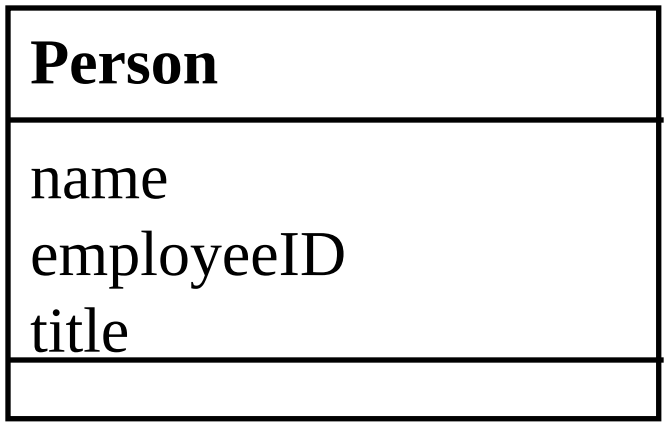
Classes

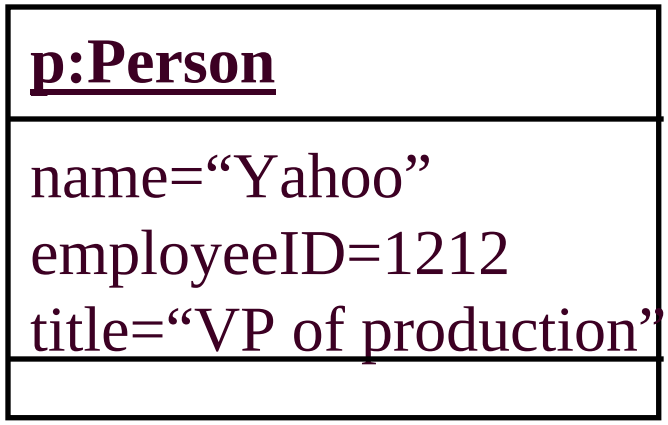
- Attribute: A named property of a class that describes a range of values that instances of the property may hold.
- Operation: Implementation of a service that can be requested.
- Responsibility: A contract or an obligation of a class. As the model is refined the responsibilities are transformed into attributes and operations.

Object

- Object - a concept, an abstraction, a thing meaningful to the domain
  - Joe Smith — Person
  - Lassie — Dog
  - flight #713 — Flight
  - the top window — Window
- Object Class - a collection of all objects having the same set of features.
- Each object has a unique *identity* and is an *instance* of its object class.

Class Diagrams vs. Object Diagrams





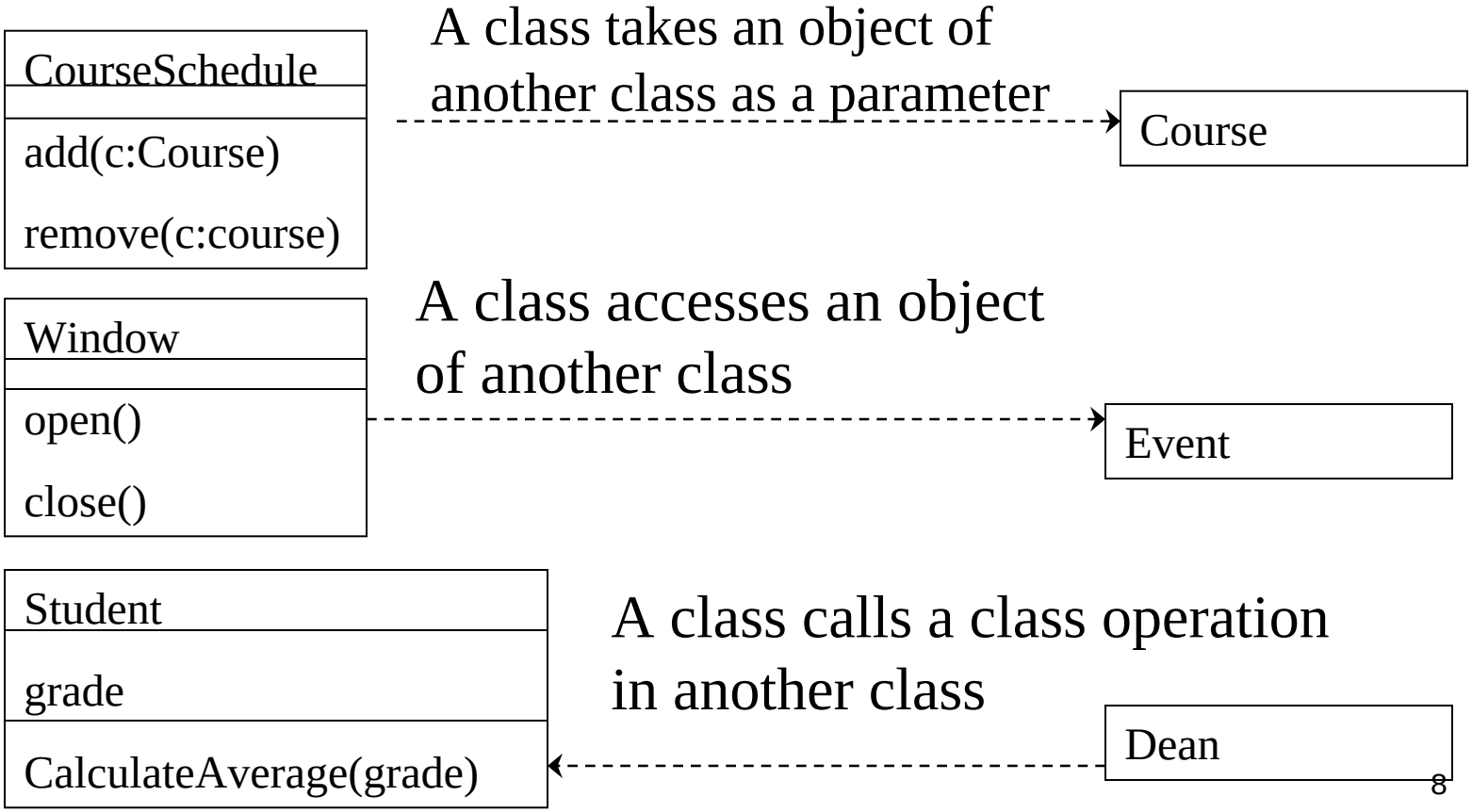
Joe Smith:Person

Relationships

- A relationship is a connection among things.
- Common relationships: Dependencies, Associations, Generalization and Aggregations.

Dependency

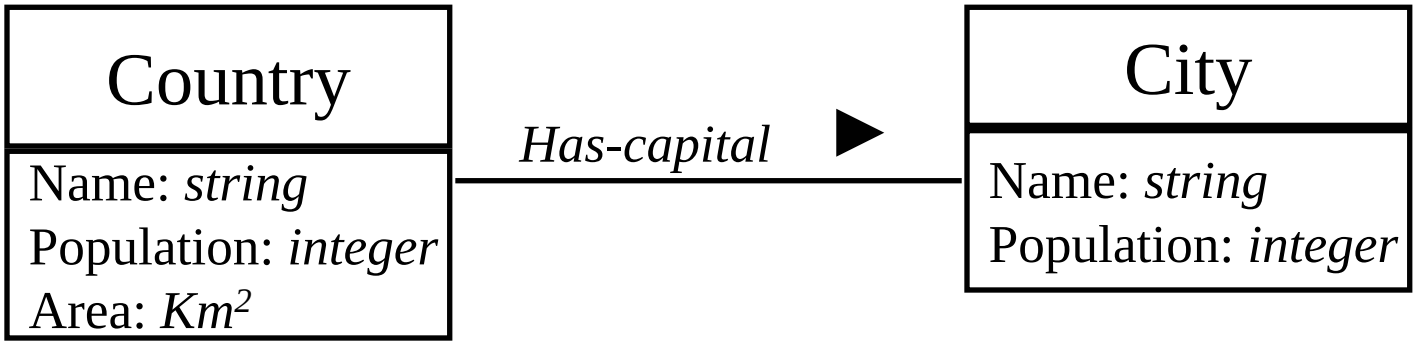
- A dependency states that a change in specification of one thing may effect another thing that uses it



Association

An association is structural relationship that specifies that objects of one thing are connected to objects of another.

Binary association



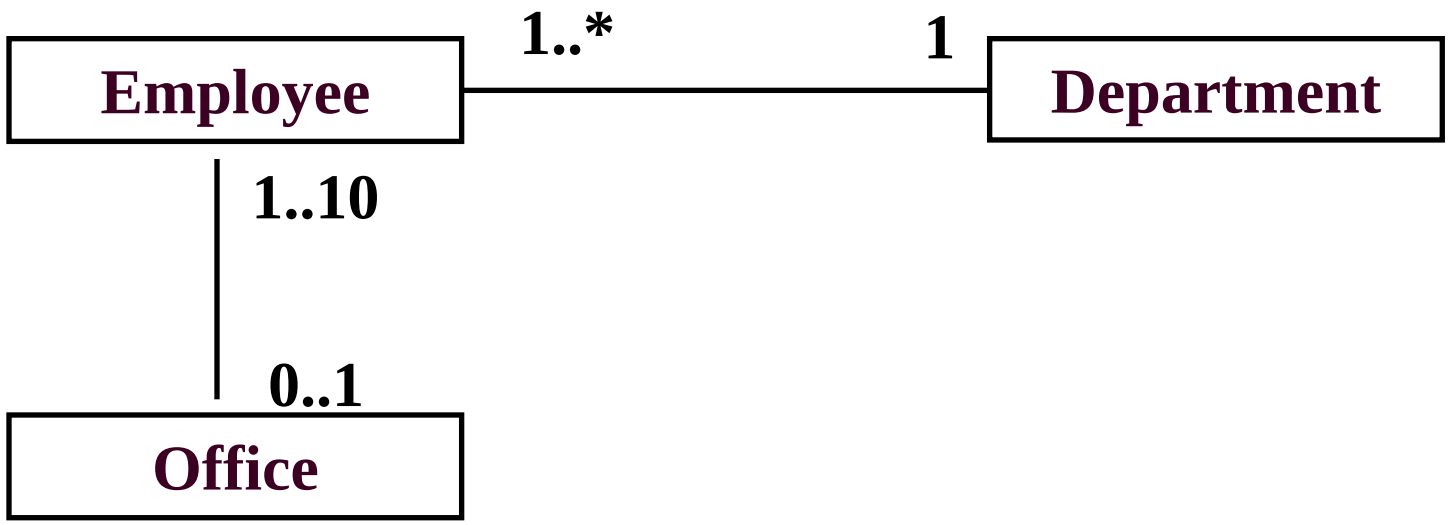
Association Name

- Printed at the center of the line
- Describes the nature of the relationship (usually a verb)
- A direction triangle can be added

Association Multiplicity

- Used when it is important to state how many objects may be connected across an instance of an association.
- When a number is stated at an end of an association it is specified that for each object at the class at the opposite end there must be that many objects at the near end.
- An * denotes many - which can be any number between zero and infinity.

Multiplicity Example



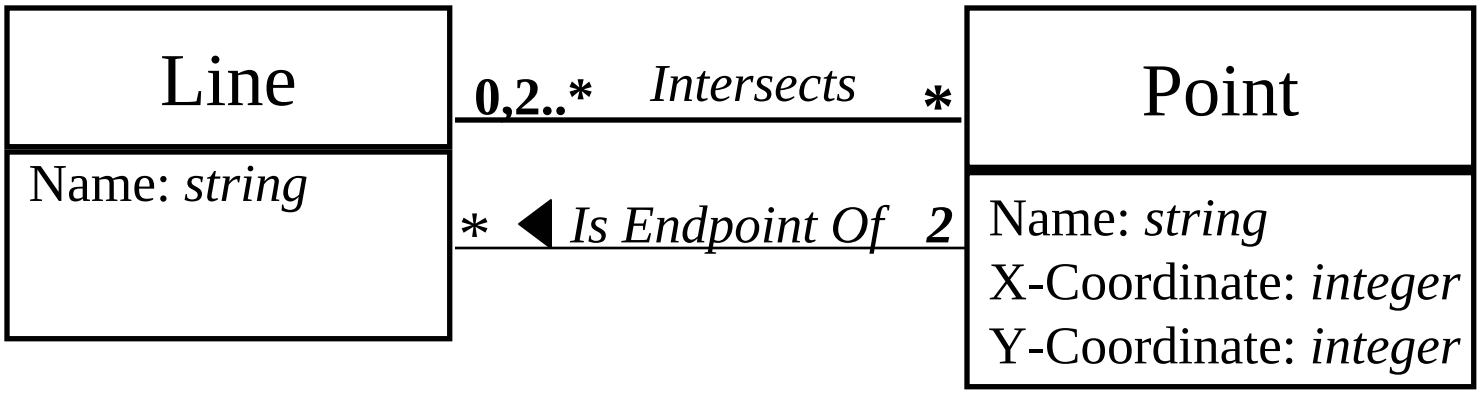
At least one employee in a department

An employee belongs to exactly one department

An employee has zero or one office

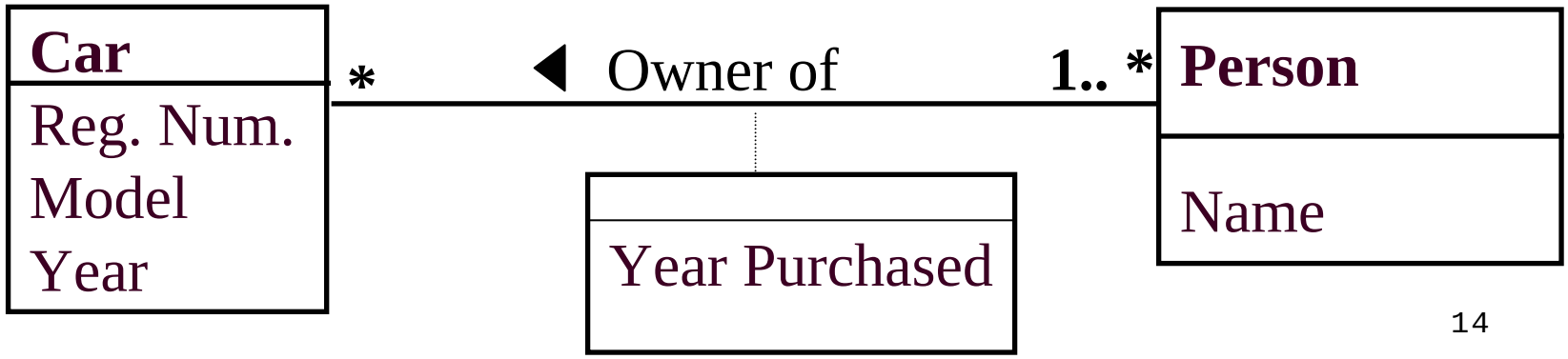
An office is assigned to a number of one up to 10 employees

Multiplicity symbols (participation constraints)

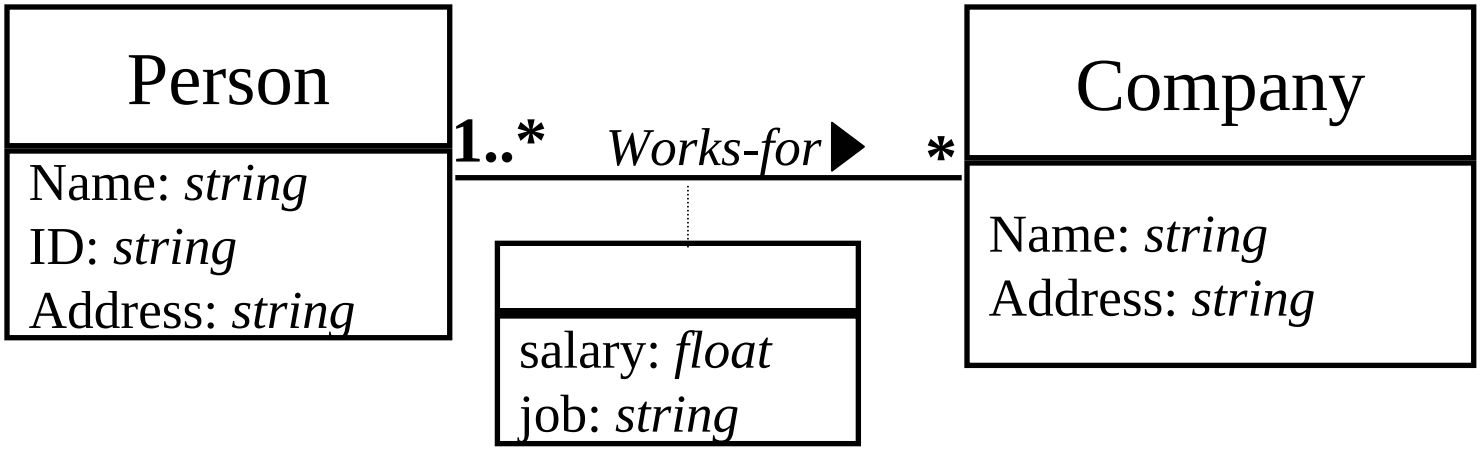


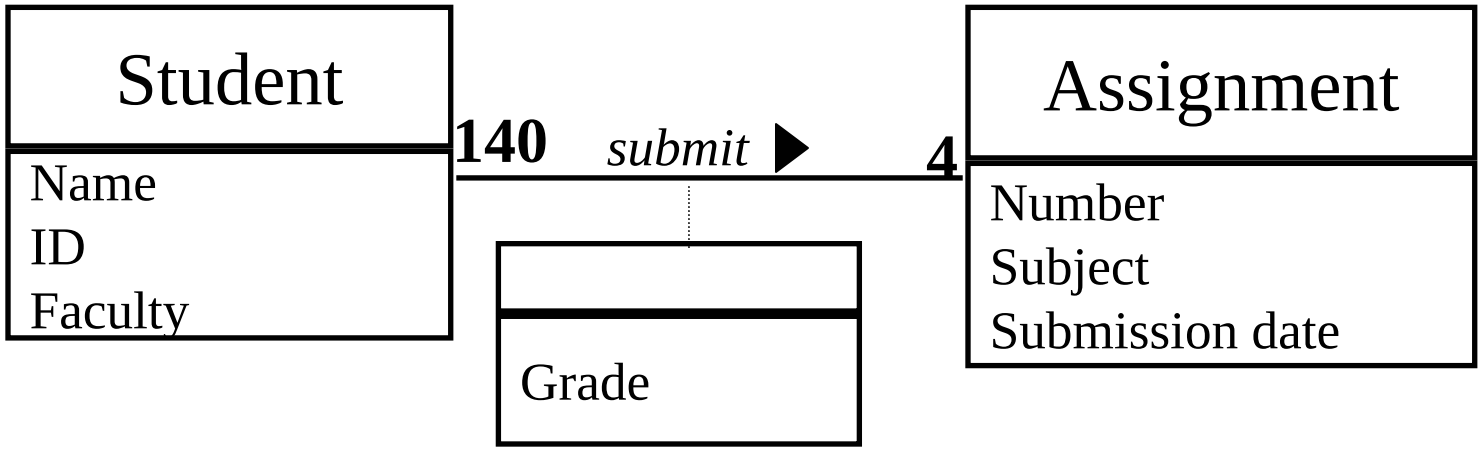
Association Classes

- Denoted as a class attached to the association
- Specify properties of the association
- Does not belong to any of the connected class



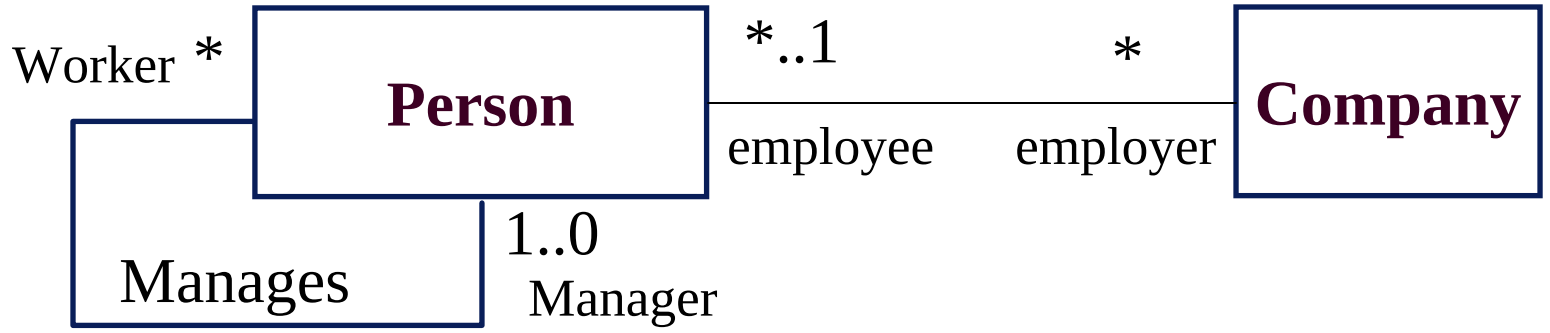
Association Attributes





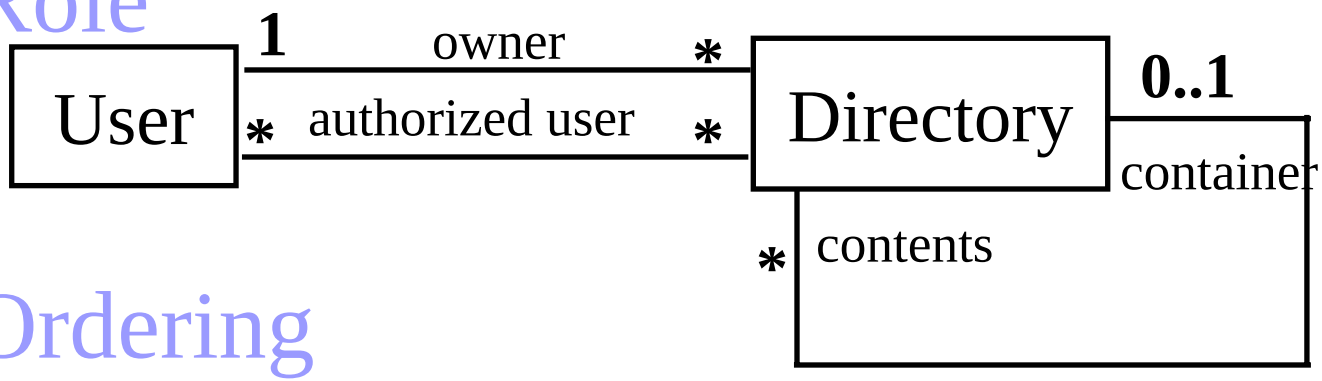
Association Role Names

- Names may be added at each end of the association
- Provide better understanding of the association meaning
- Especially helpful in self-associated classes

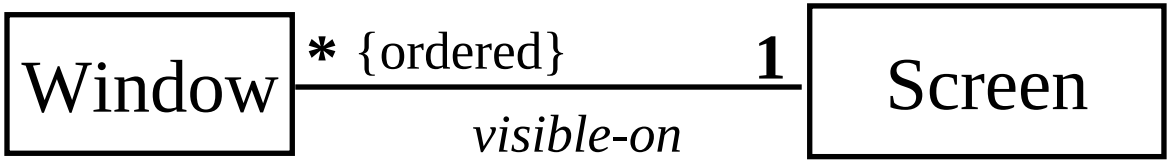


Role Names, Constraints and Qualifiers

Role

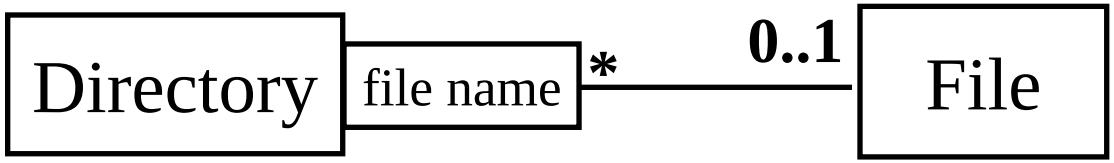


Ordering



Qualifier:

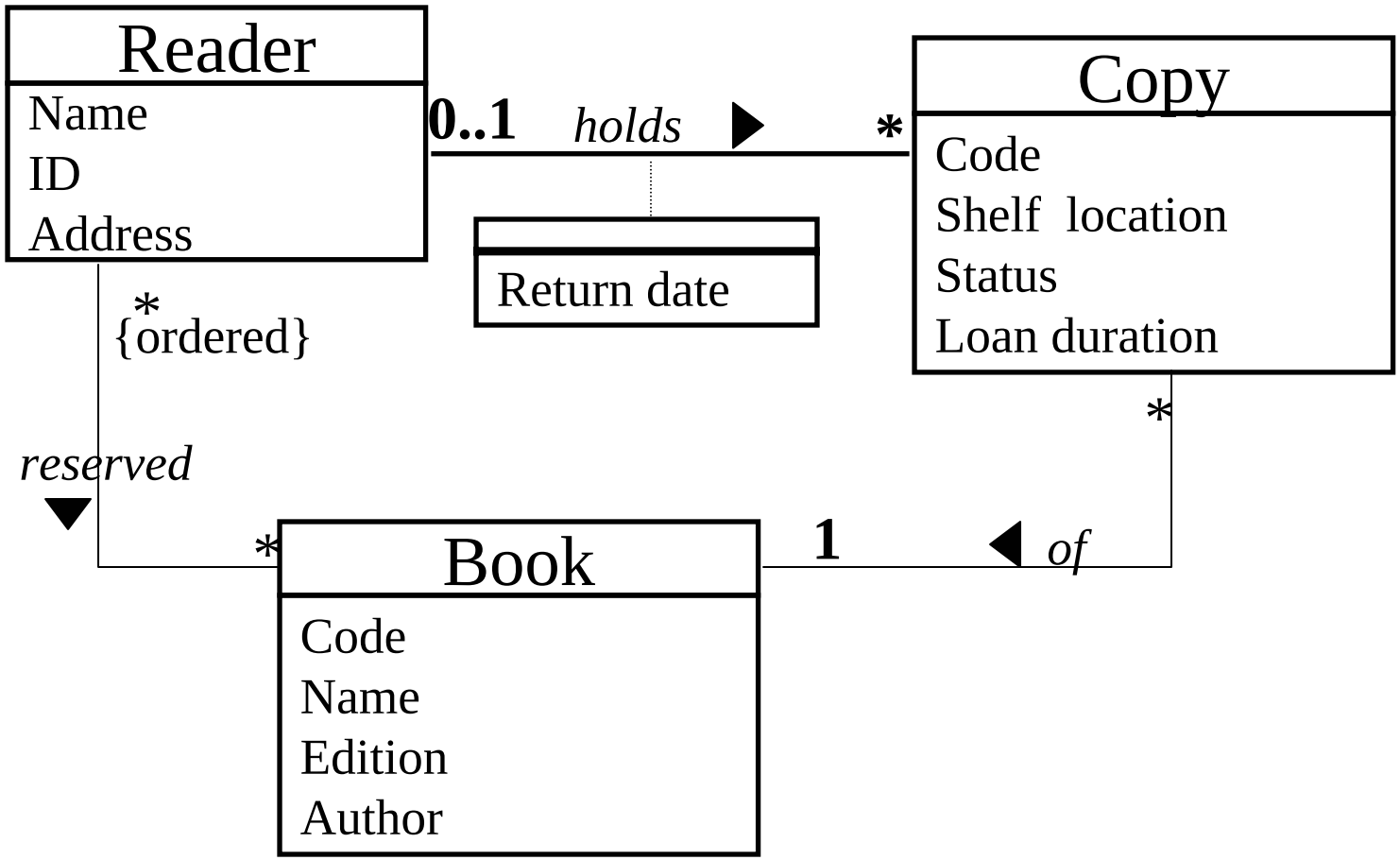
an attribute that reduces the association's multiplicity.



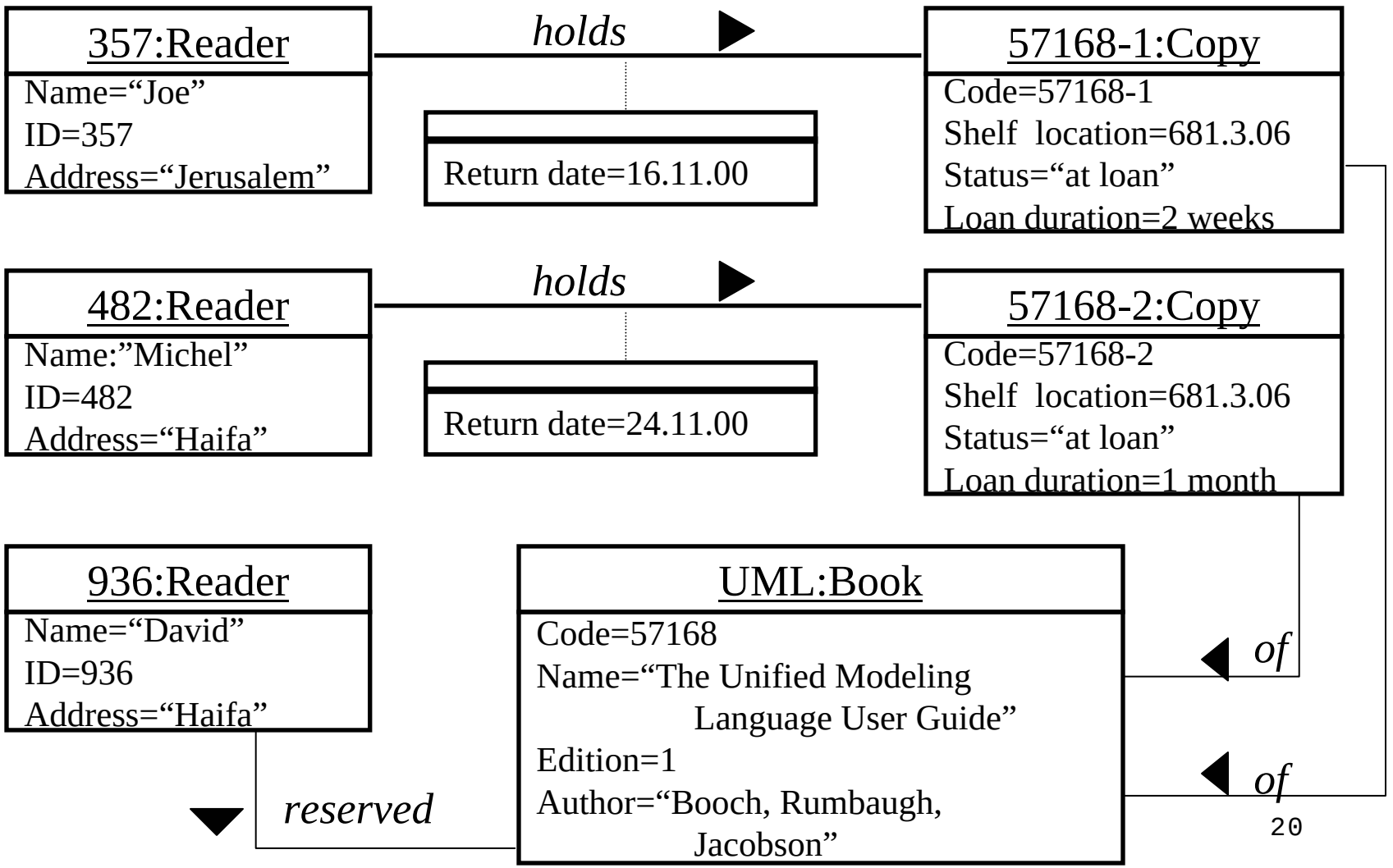
Classes versus Objects

Class Diagram		Object Diagram	
Classes	name attributes operations responsibility	Objects	values for attributes
Association	name multiplicity role	Link	
Association Classes		Link Object	

Class Diagram Example



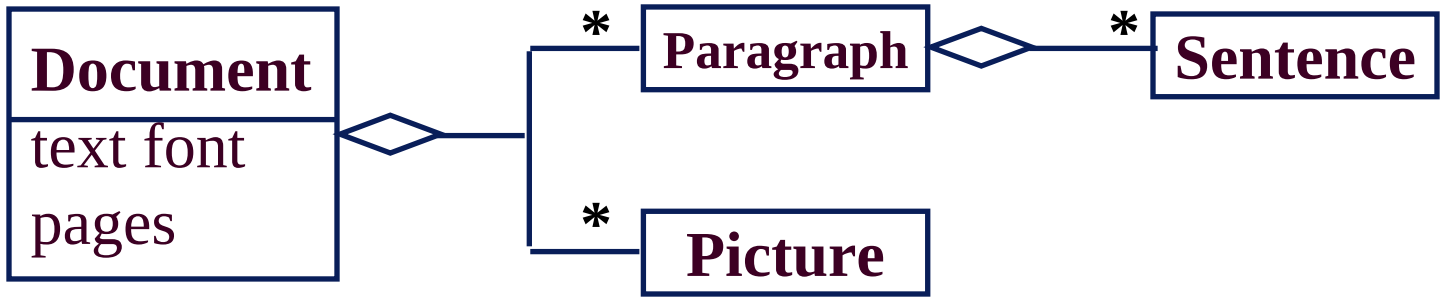
Object Diagram Example



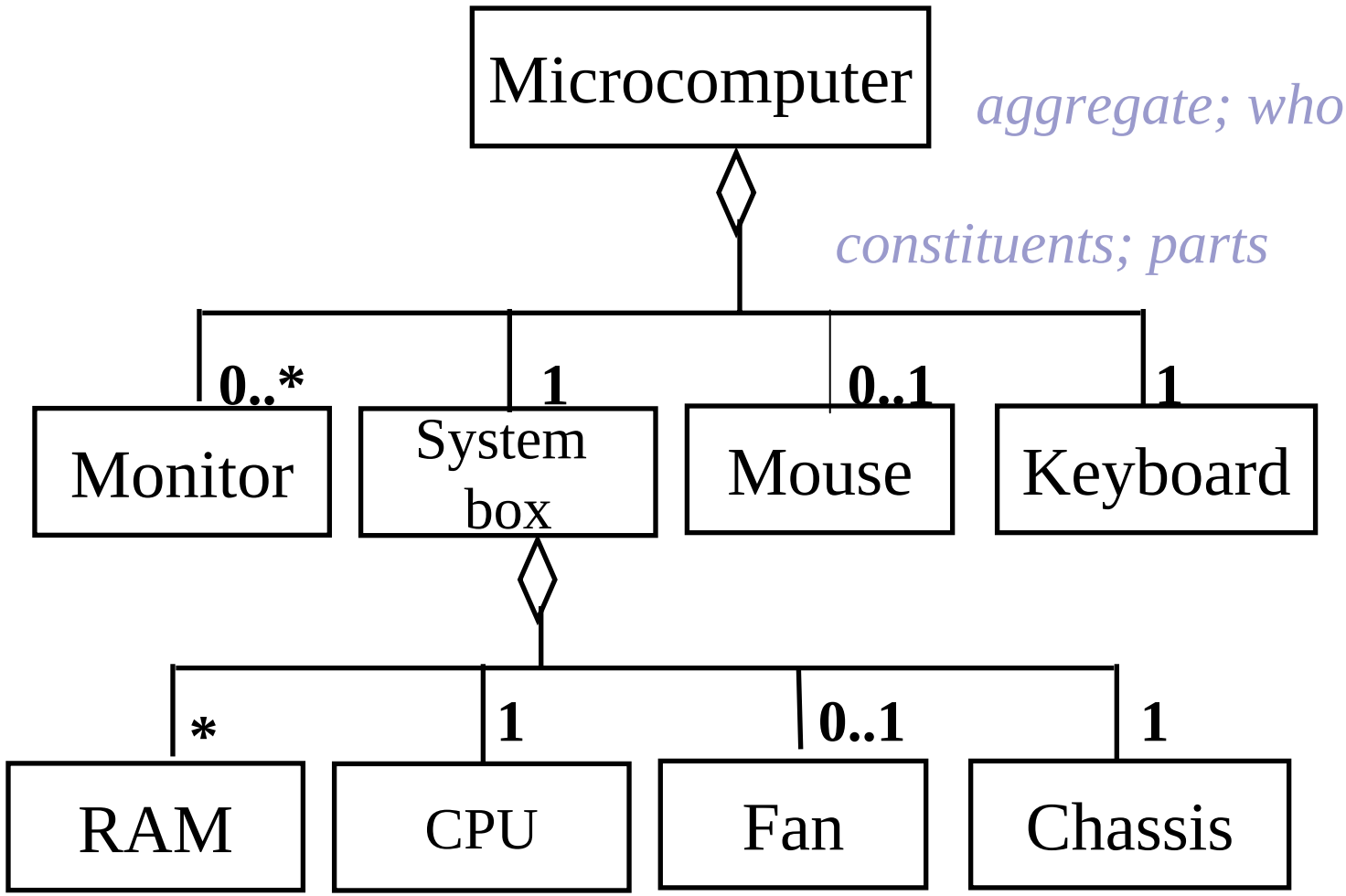
Aggregation

- “whole-part” relationship between classes
- Assemble a class from other classes

  Combined with “many” - assemble a class from a couple of instances of that class
- May be replaced by associations...

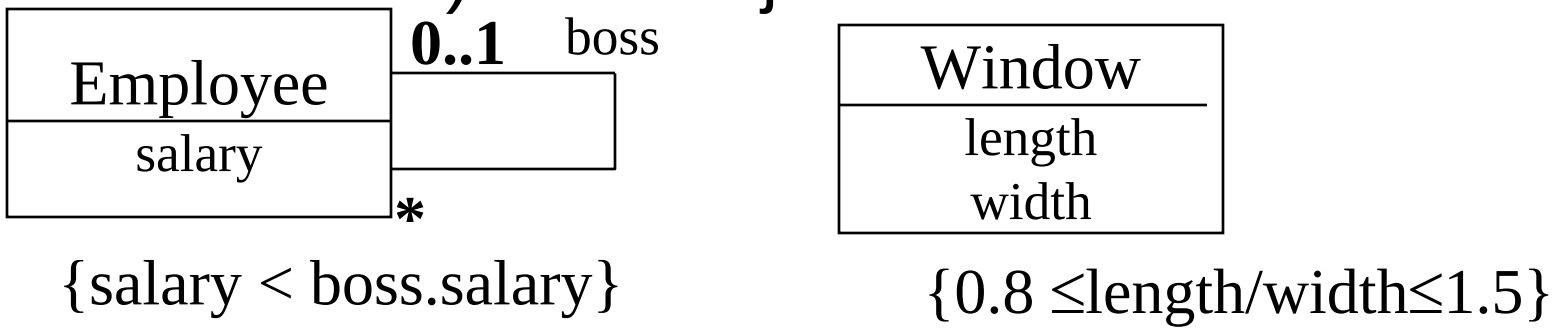


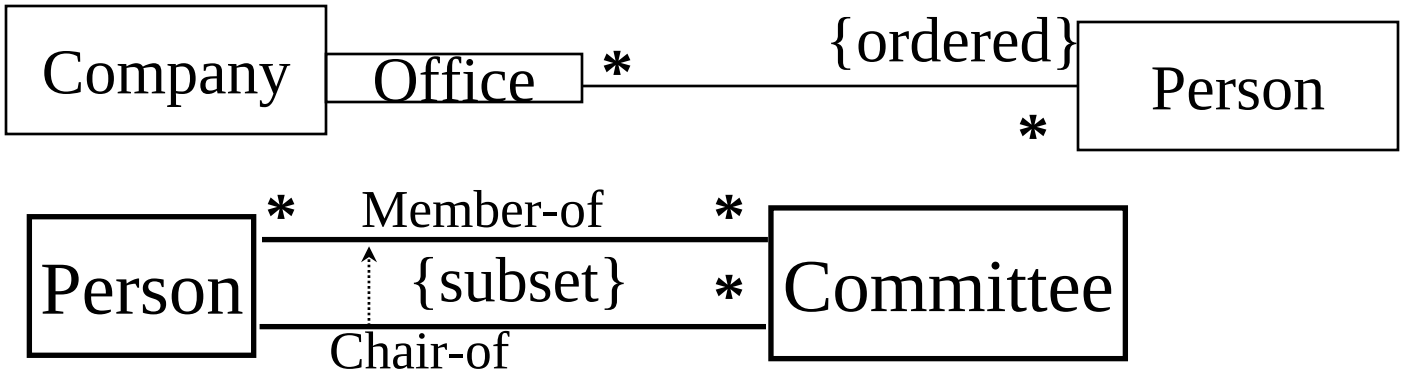
Aggregation (whole-part; part-of; and)



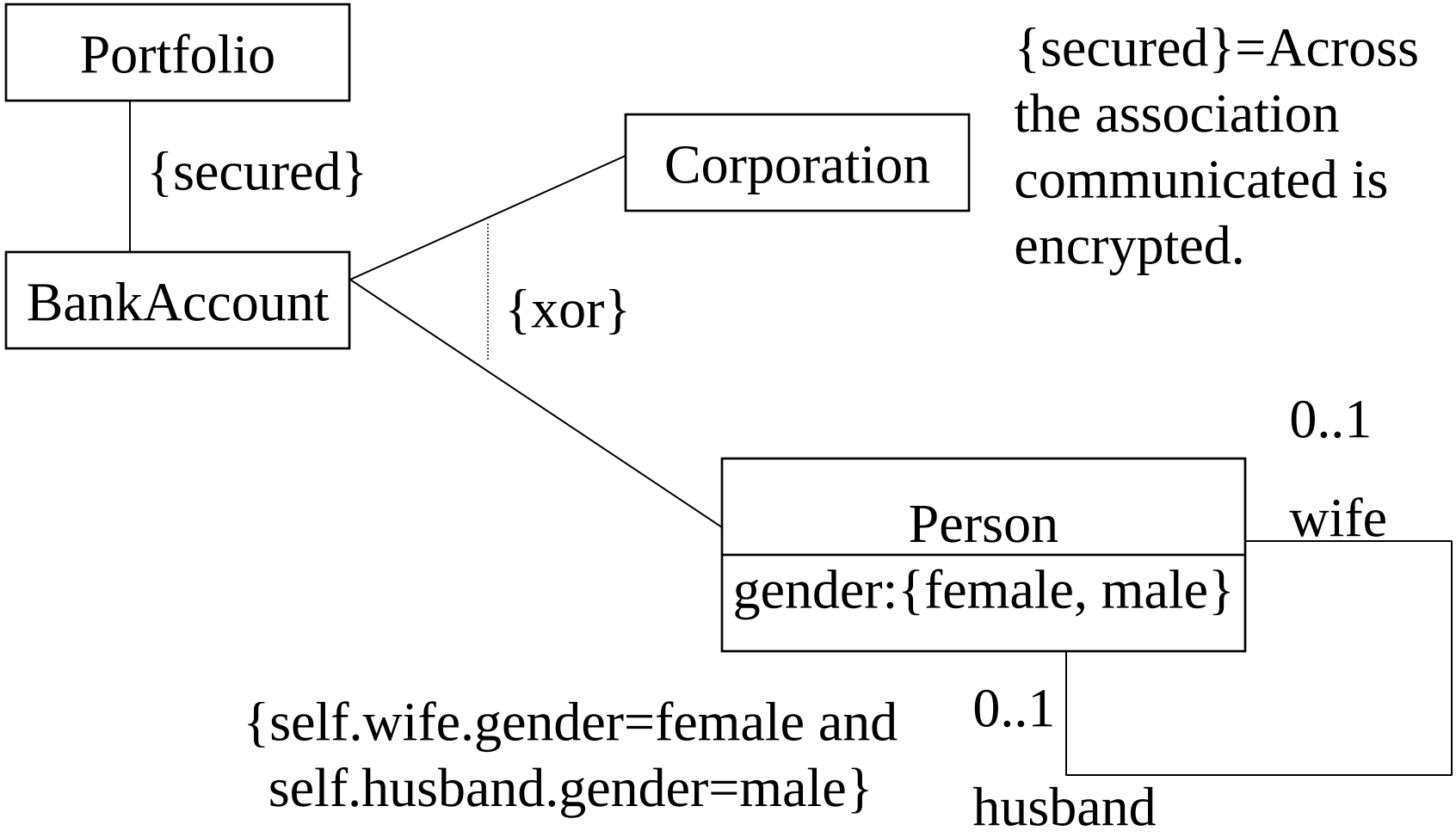
Constraint

A functional relationship between entities (objects, classes, attributes, links, associations) of an object model.





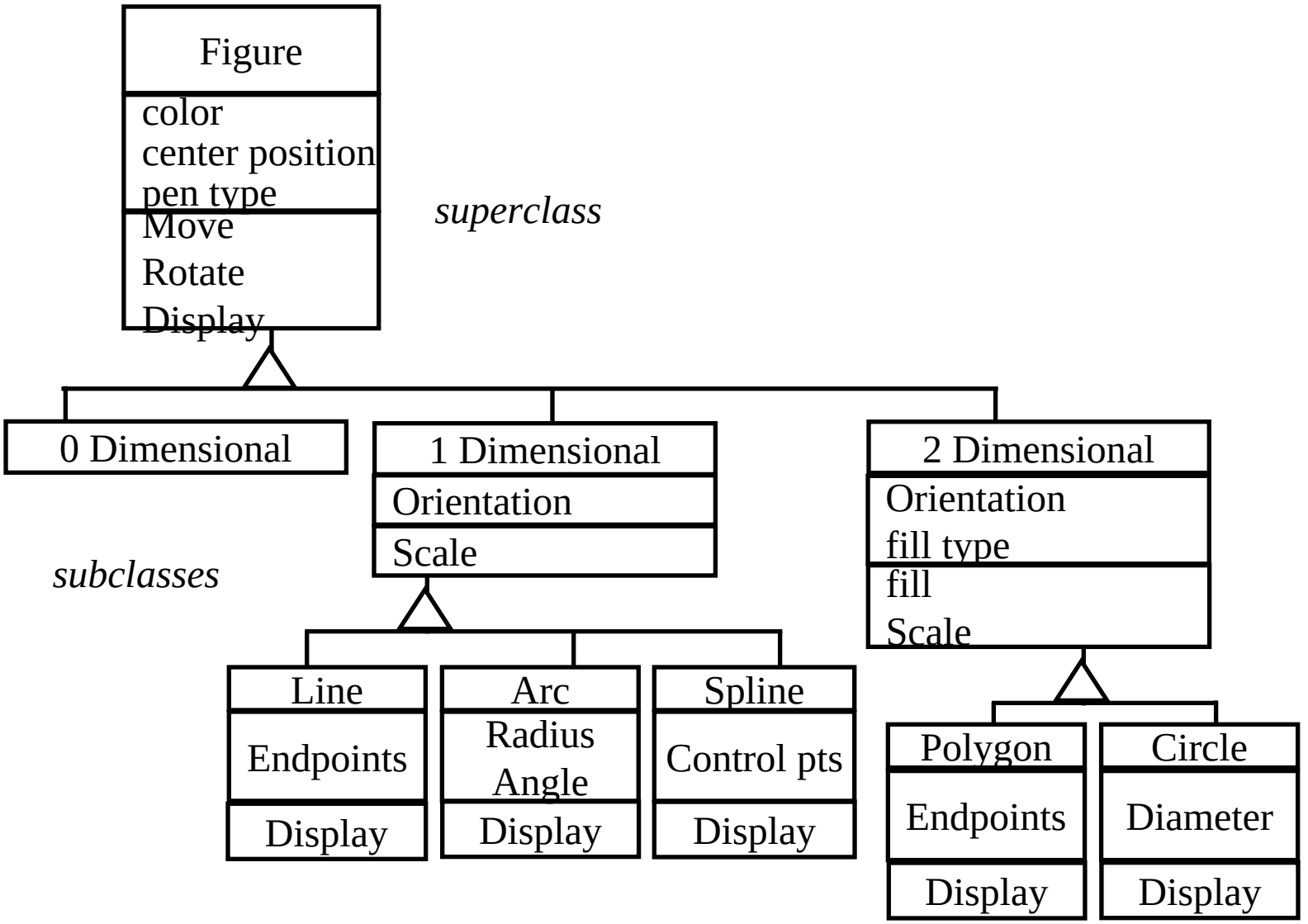
Constraint (Cont.)



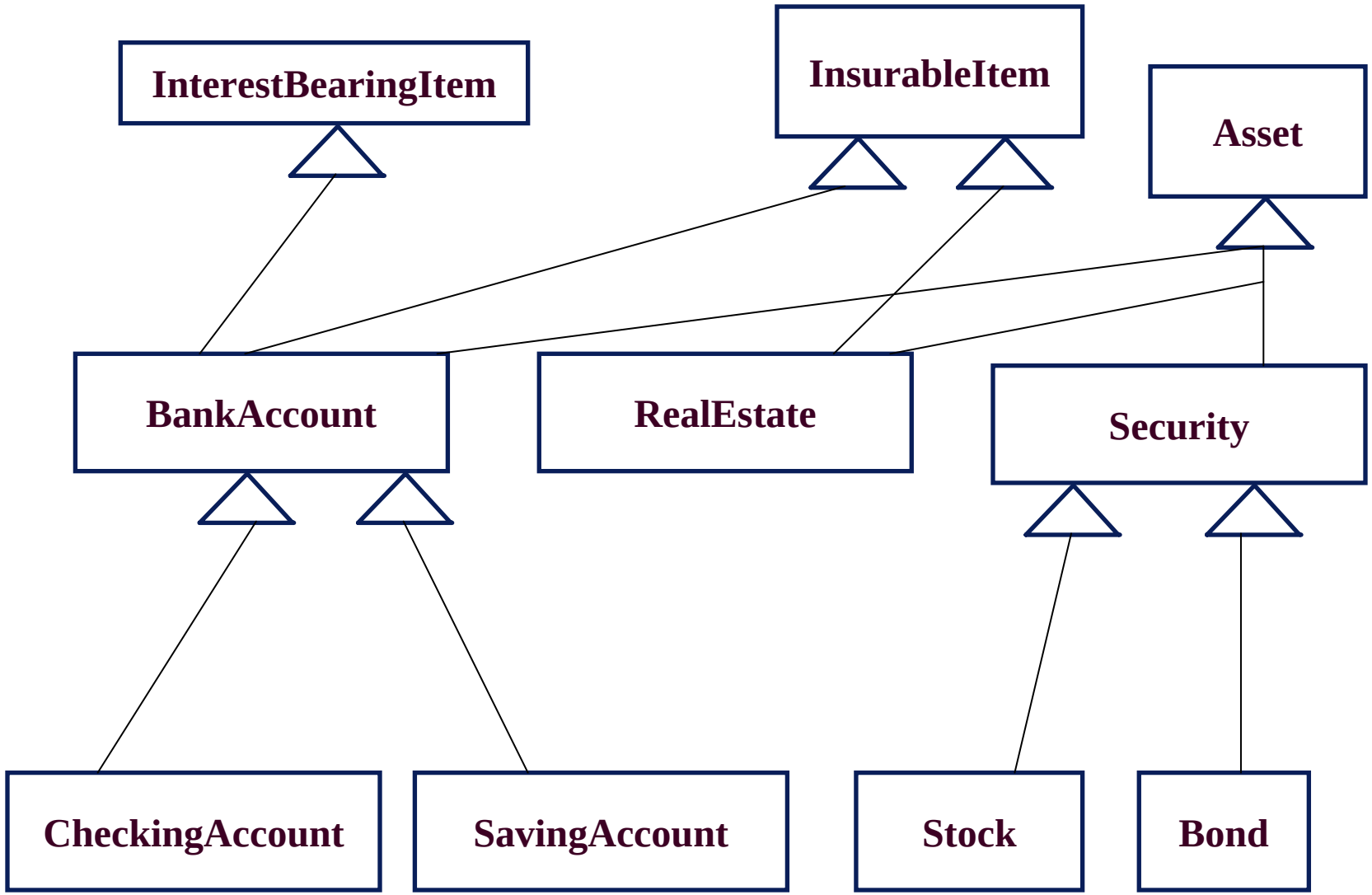
Inheritance (Generalization)

- Denoted by a triangle - connects one superclass to many subclasses
- Multiple inheritance is allowed
  - a subclass may be connected to more than one superclass
  - not recommended - hard to understand, hard to implement
- Four applicable standard constraints:
  - Complete
  - Incomplete
  - disjoint
  - overlapping

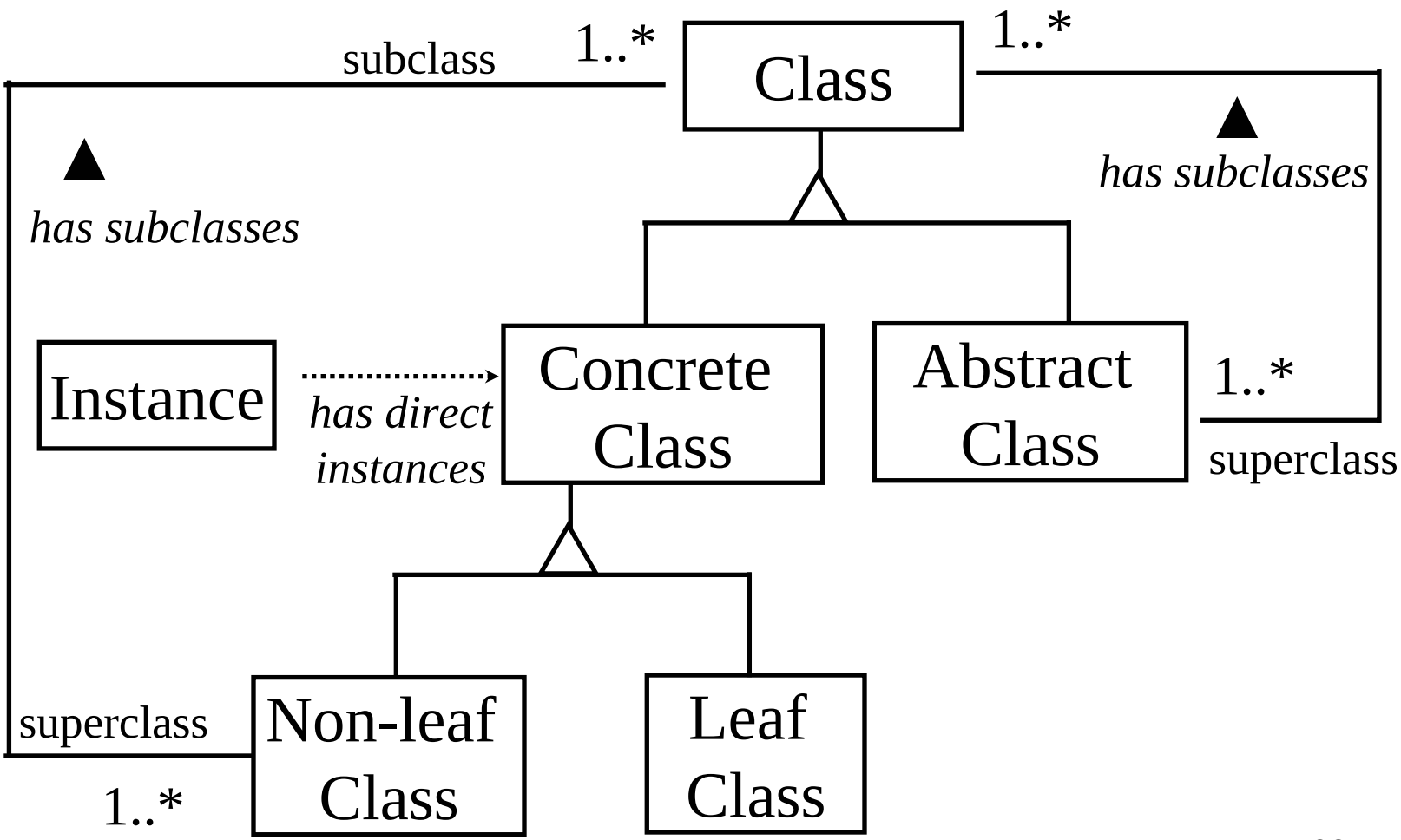
Generalization (gen-spec; kind-of; or)



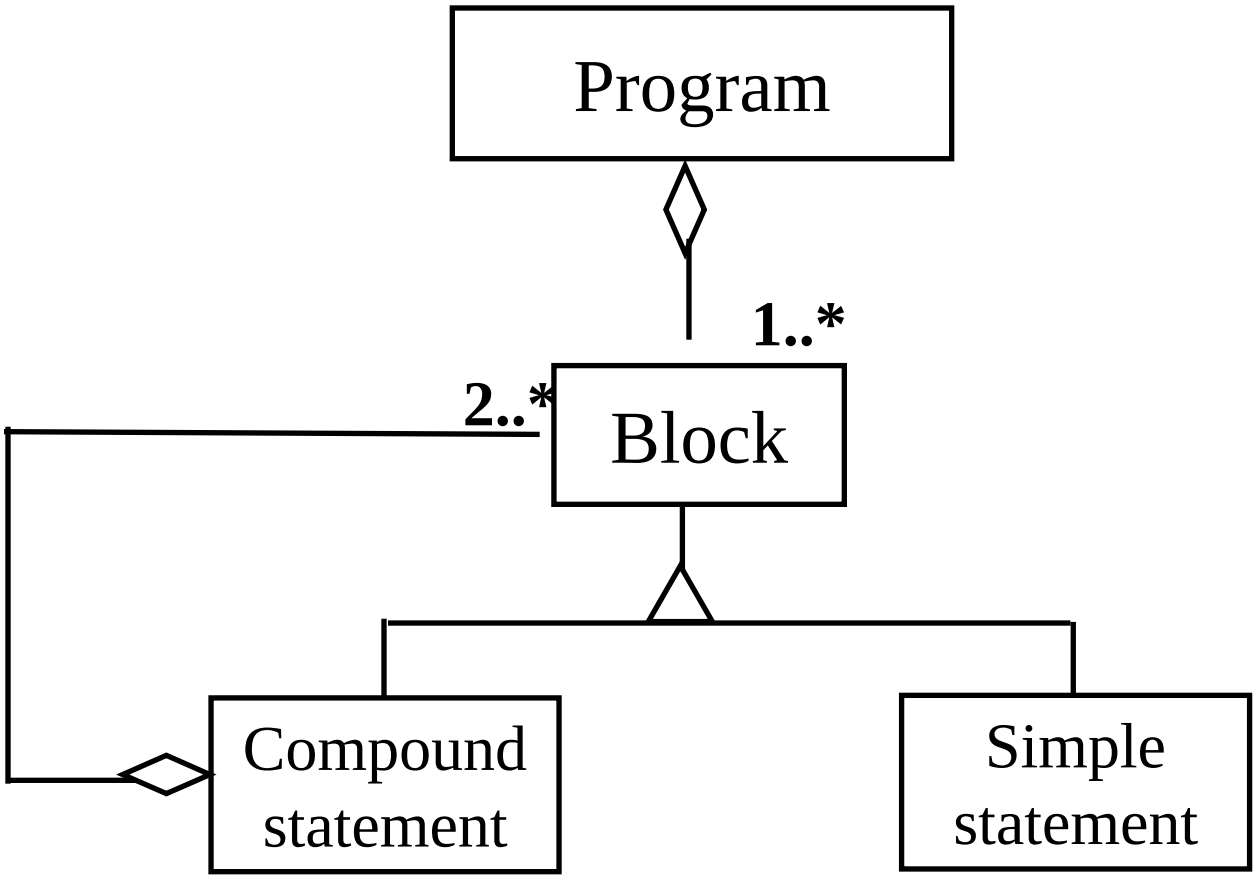
Multiple Inheritance



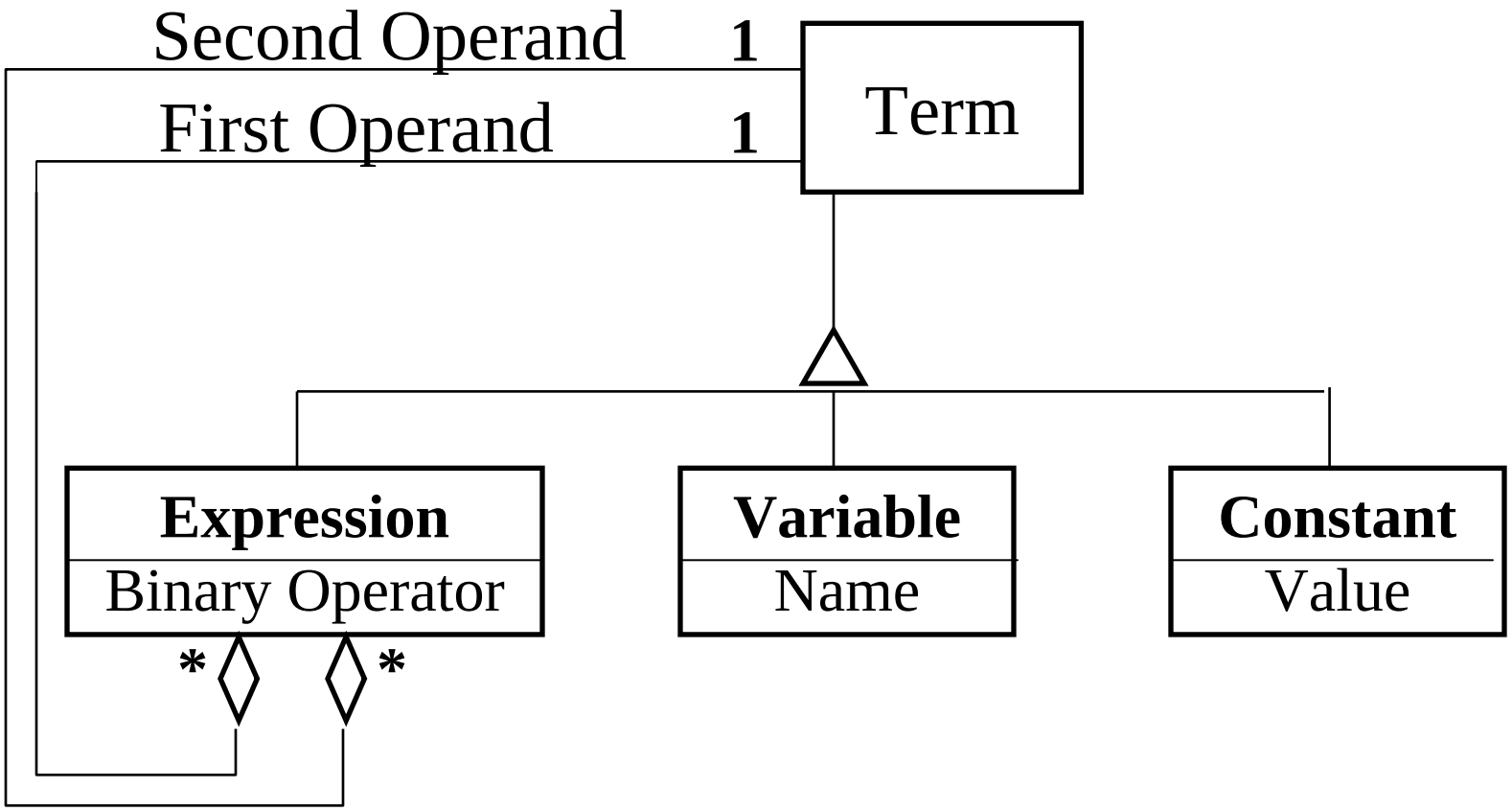
Abstract Class - a class that has no direct instances



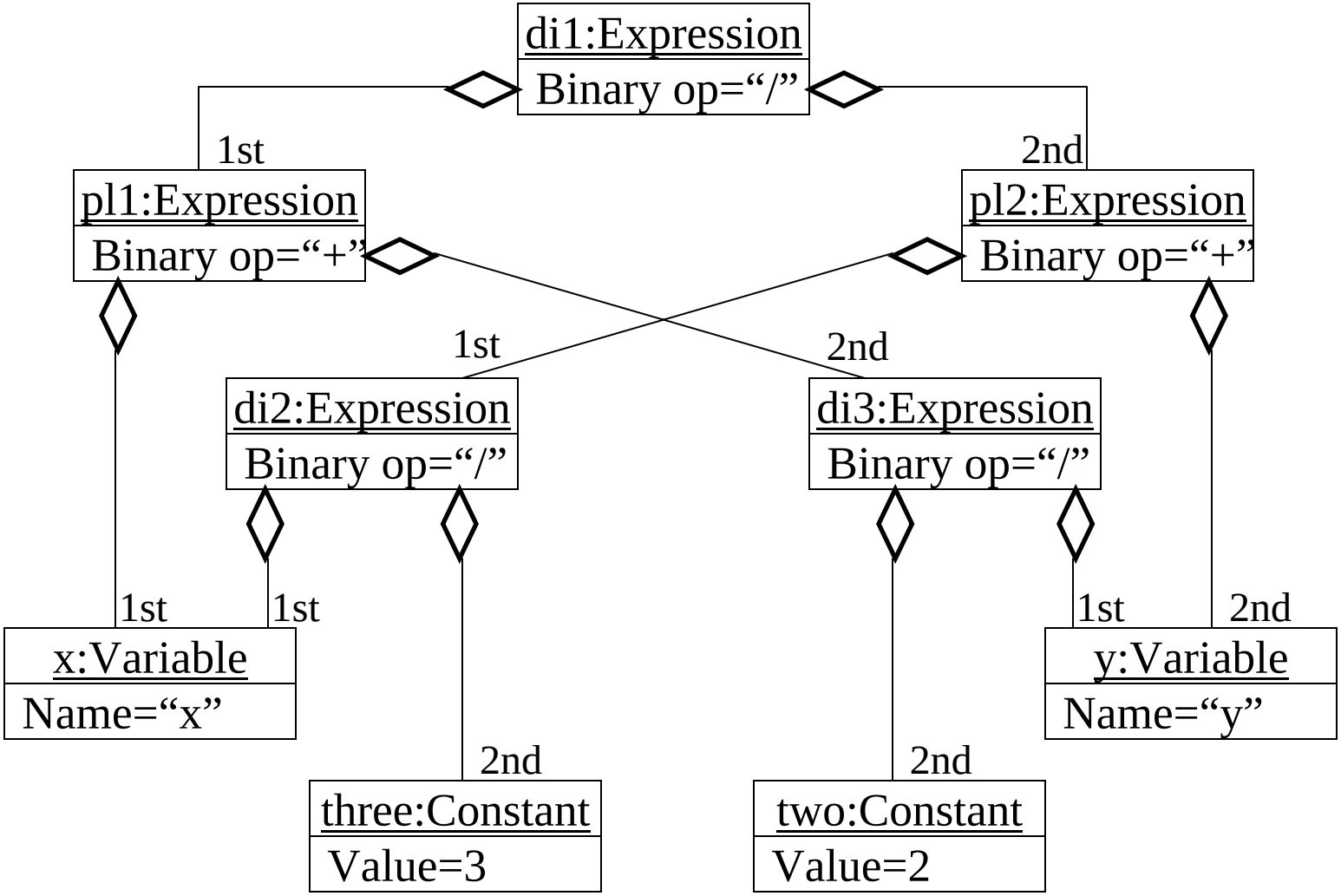
Recursive Aggregation



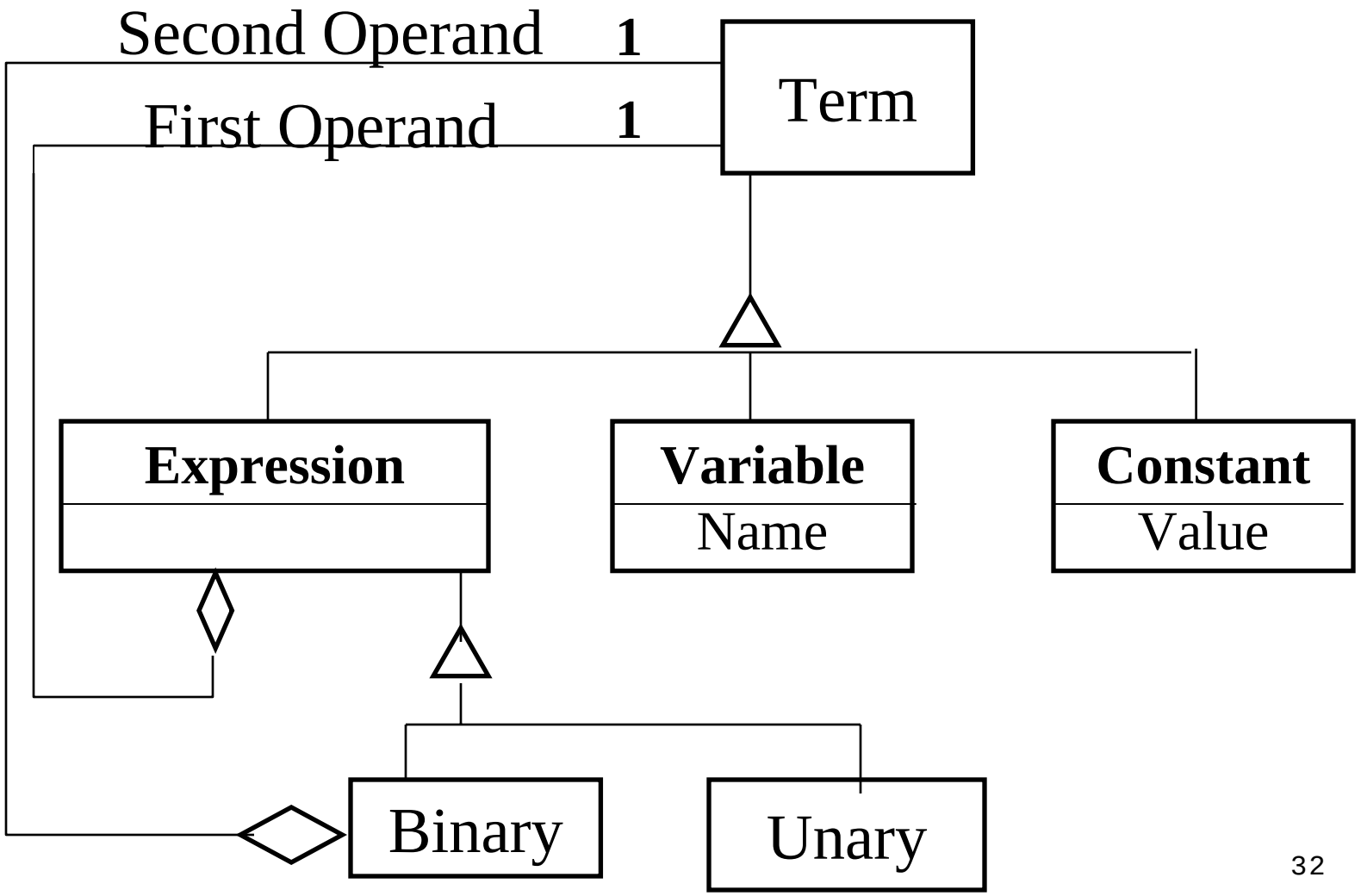
Recursive Terms (Multiple Binary term use)



(x+y/2)/(x/3+y)

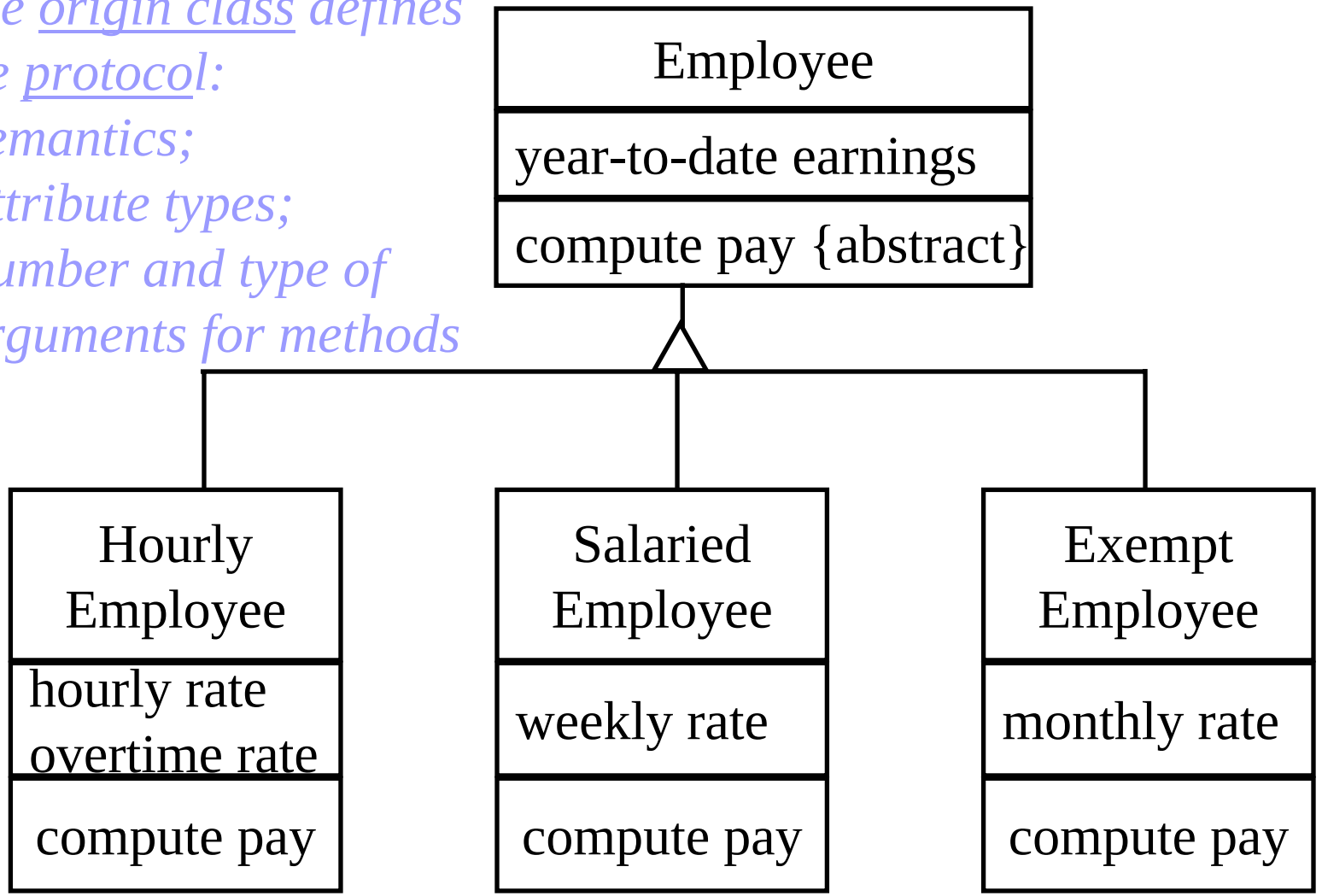

di1:Expression
Binary op="/"
1st
2nd
pl1:Expression
Binary op="+"
pl2:Expression
Binary op="+"
1st
2nd
di2:Expression
Binary op="/"
di3:Expression
Binary op="/"
1st
1st
1st
2nd
x:Variable
Name="x"
y:Variable
Name="y"
2nd
2nd
three:Constant
Value=3
two:Constant
Value=2

Recursive Terms
(Unary &Binary & Single Term Use)



Abstract Operations

The origin class defines the protocol:
- *semantics;*
- *attribute types;*
- *number and type of arguments for methods*



Generalization as Extension and Restriction

Extension: addition of new features.
(e.g. Person vs. Student)
Restriction: constraining ancestor attributes.
(e.g. nullifying some attributes)

Overriding Operations

- Tension between use of inheritance for abstraction vs. implementation reuse.
- Reasons for overriding:
  - Extension
    (e.g. Person has Report_Marriage; Employee has Report_Marriage that extends the code of Report_Marriage)
  - Restriction
    (e.g. Subclass operation use some of superclass sub-operations)

Reasons for overriding (cont.)

- Optimization
  e.g. IntegerSet has findMax; SortedIntegerSet has another implementation for findMax.
- Convenience
  Look for a similar class and use it as superclass. This is ad hoc use which is semantically wrong. Better: define a third common class, from which both will inherit.

Rule for method overriding:

All methods that implement an operation must have the same protocol.