Karlstad University
Department of Information Systems

Adapted for a textbook by Blaha M. and Rumbaugh J.

Object Oriented Modeling and Design

Pearson Prentice Hall, 2005

CLASS MODELING

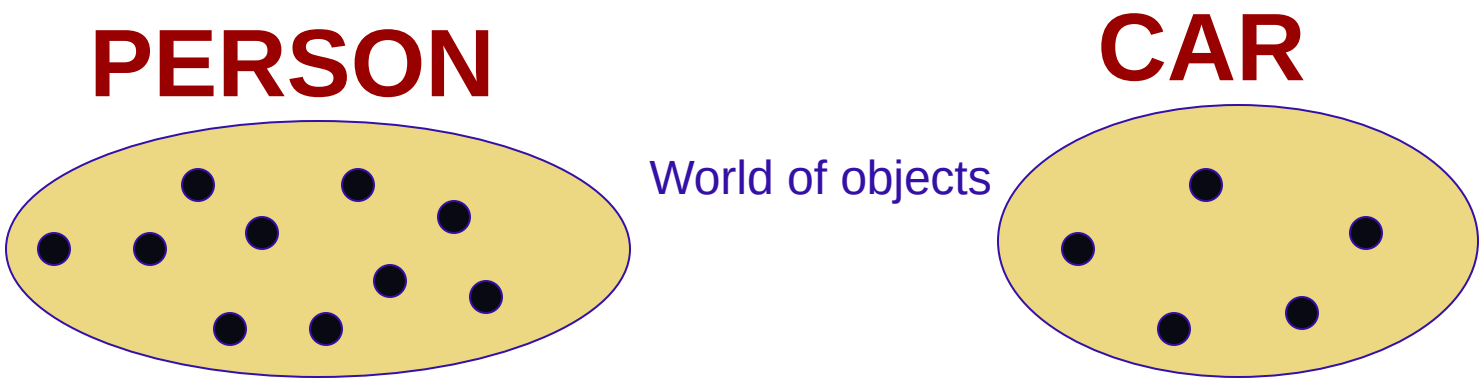
Remigijus GUSTAS
Phone: +46-54 700 17 65
E-mail: Remigijus.Gustas@kau.se
http://www.cs.kau.se/~gustas/

Fundamentals of Object Oriented Technology

- ✓ Object (instance)
- ✓ Class
- ✓ Association
- ✓ Generalization and Inheritance
- ✓ Aggregation and Composition

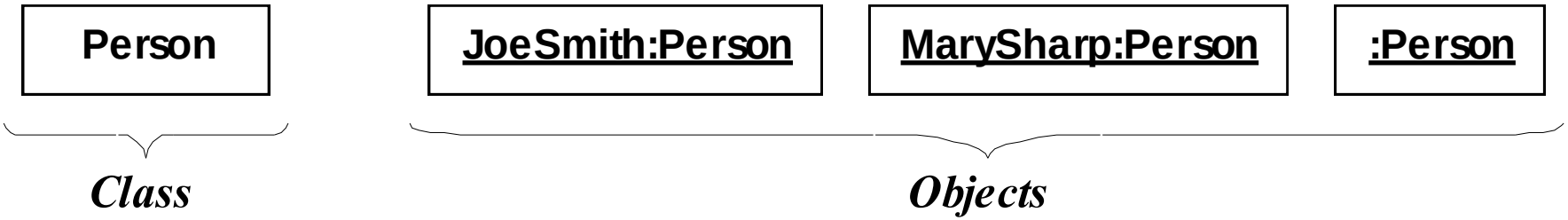
Object and Class

- ✓ Object is an instance of a class.
- ✓ An **object** is a concept or thing that has meaning for an application



- ✓ Can you think of anything that is not an object?

A **class** describes a group of objects with the same properties, behavior and associations

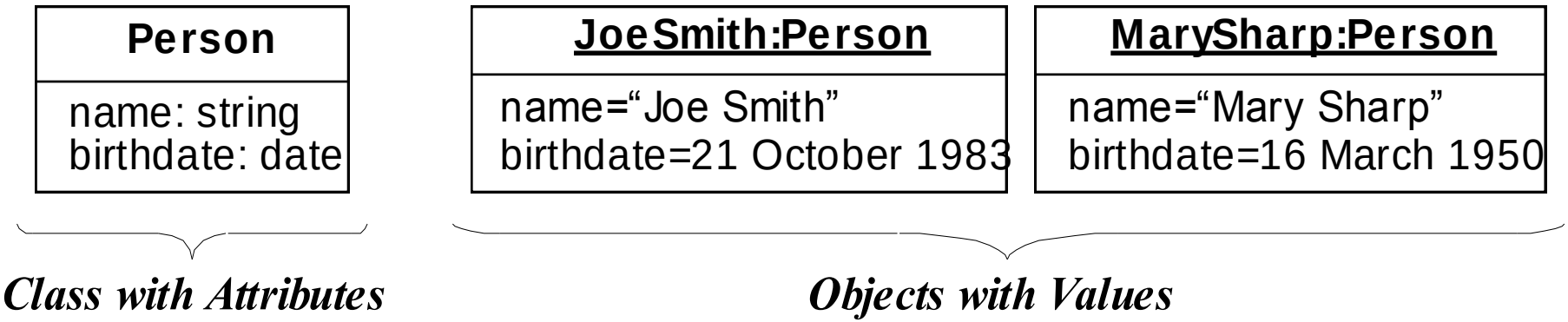


Classes of objects can be:

Tangible	Intangible	Role	Judgments	Relational	Events
Person	Time	Doctor	Good example	Marriage	Sale
Car	Quality	Patient	High pay	Partnership	Purchase
Pencil	Company	Analyst	Frequent flyer	Family	Crash

Values and Attributes

- An **attribute** is a named property of a class that describes **value** held by each object of the class. Value is a piece of data.



Operations and Methods

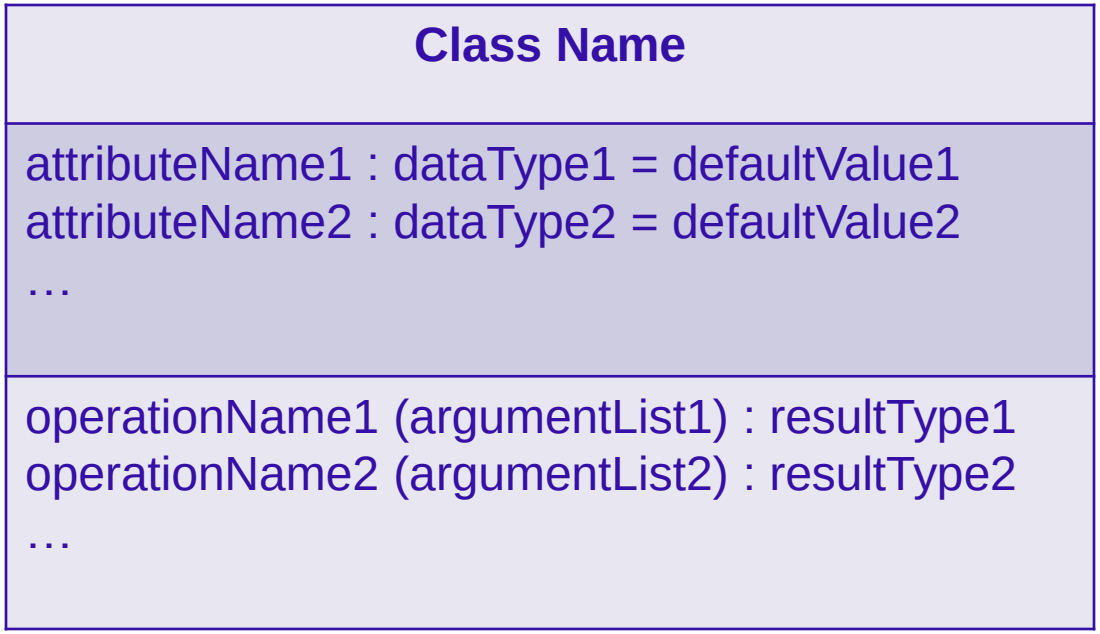
- An **operation** is a function that may be applied to or by objects in a class.
- A **signature** is the number and types of arguments and the type of result value.
- A **method** is the implementation of an operation for a class

Person
name birthdate
changeJob changeAddress

File
fileName sizeInBytes lastUpdate
print

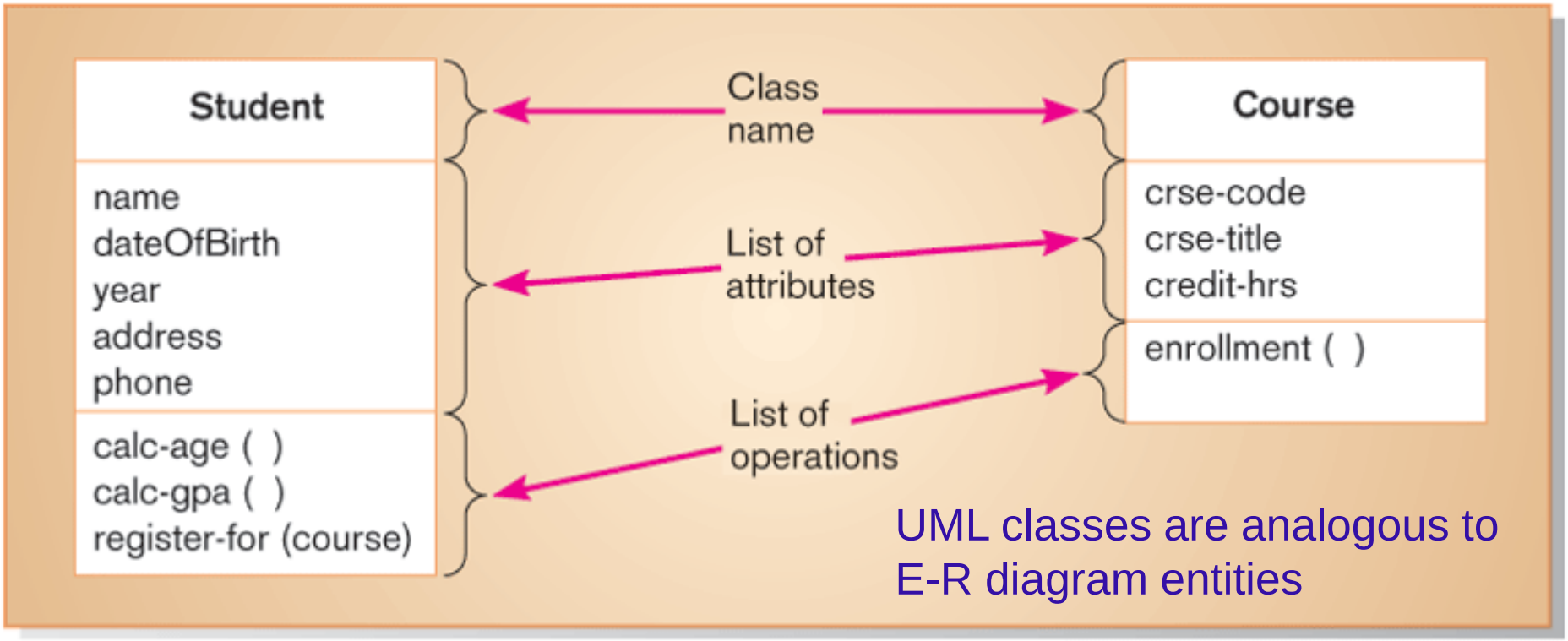
GeometricObject
color position
move (delta : Vector) select (p : Point): Boolean rotate (in angle : float = 0.0)

Summary of Notation for Classes



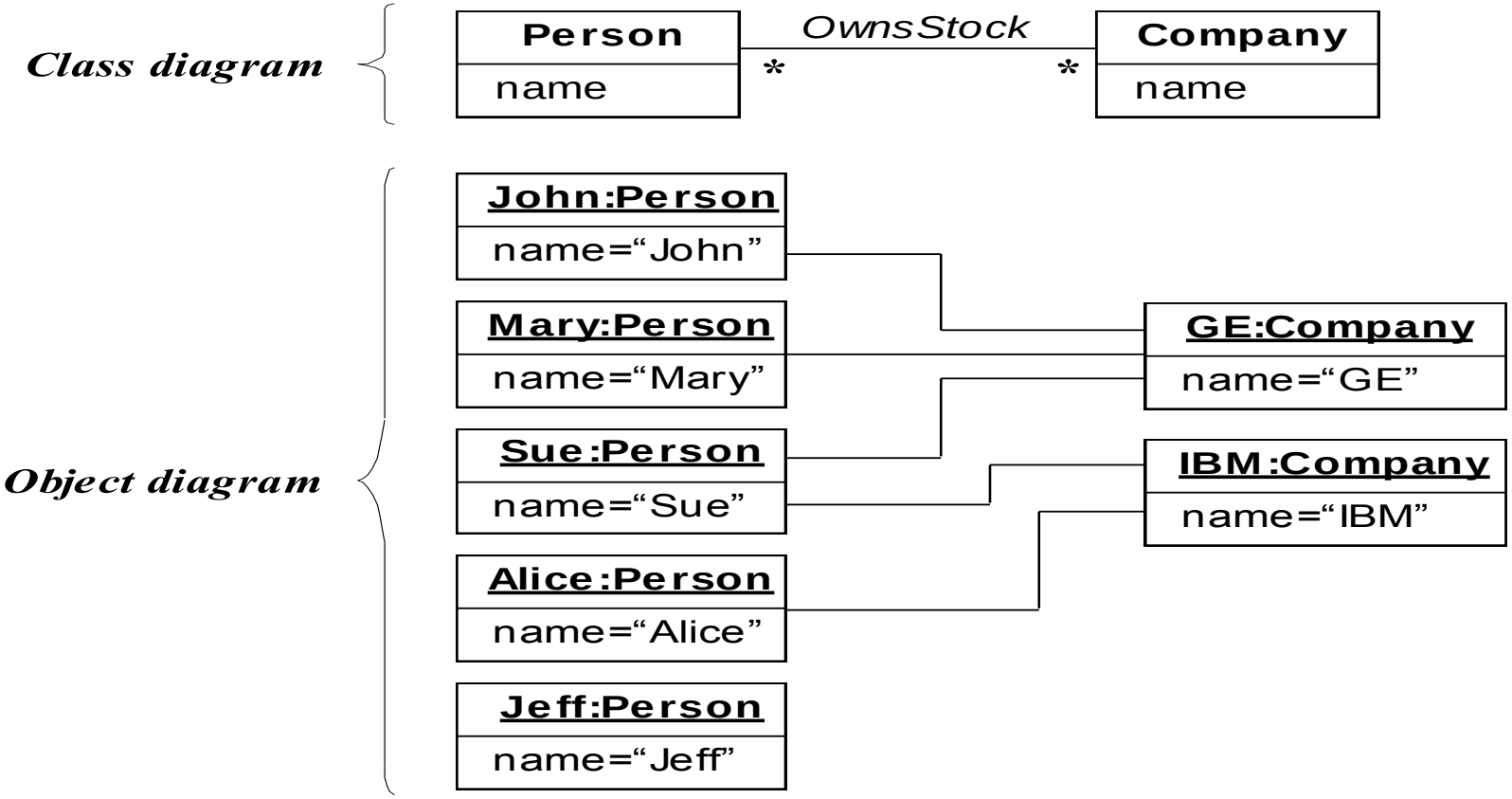
Class Diagram

A diagram showing the static structure of an object-oriented model



Source:George J.F. et al., Object Oriented System Analysis and Design, Prentice Hall, 2004

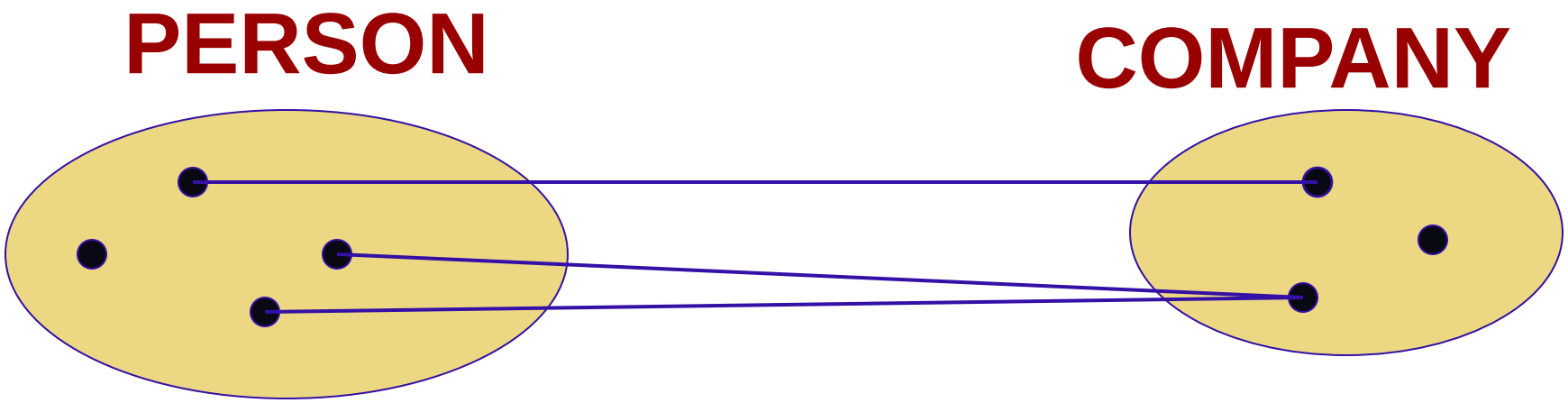
An **association** is a description of links (association instances) with common semantics and structure



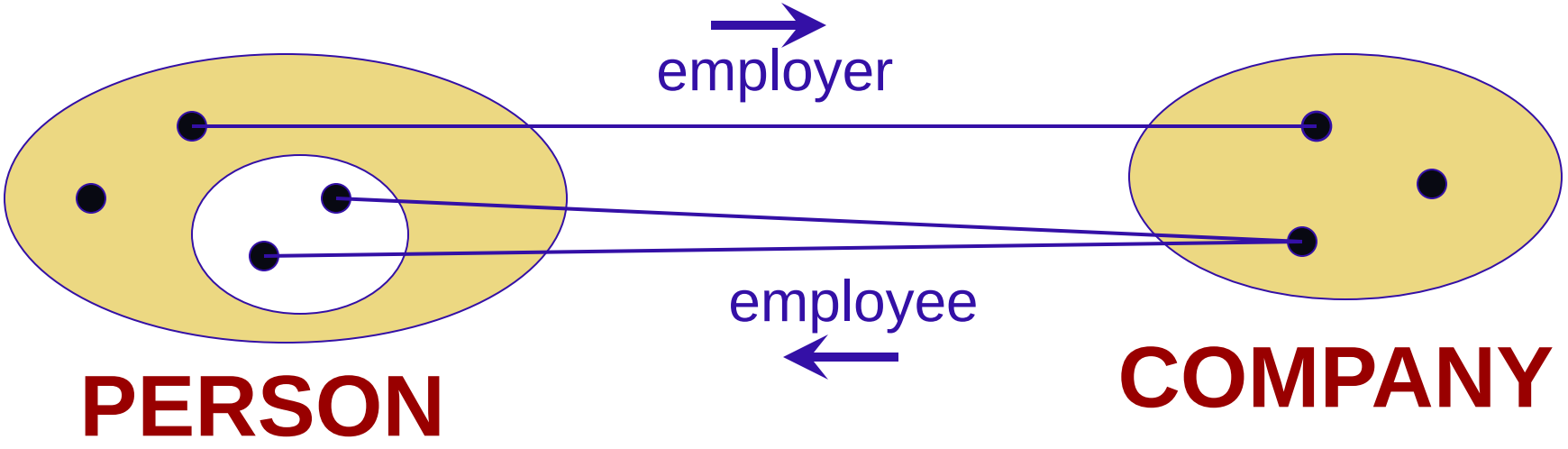
Representing Associations

- ✓ Association: a relationship among instances of object classes
- ✓ Association role: the end of an association where it connects to a class
- ✓ Multiplicity: indicates how many objects participate in a given relationship

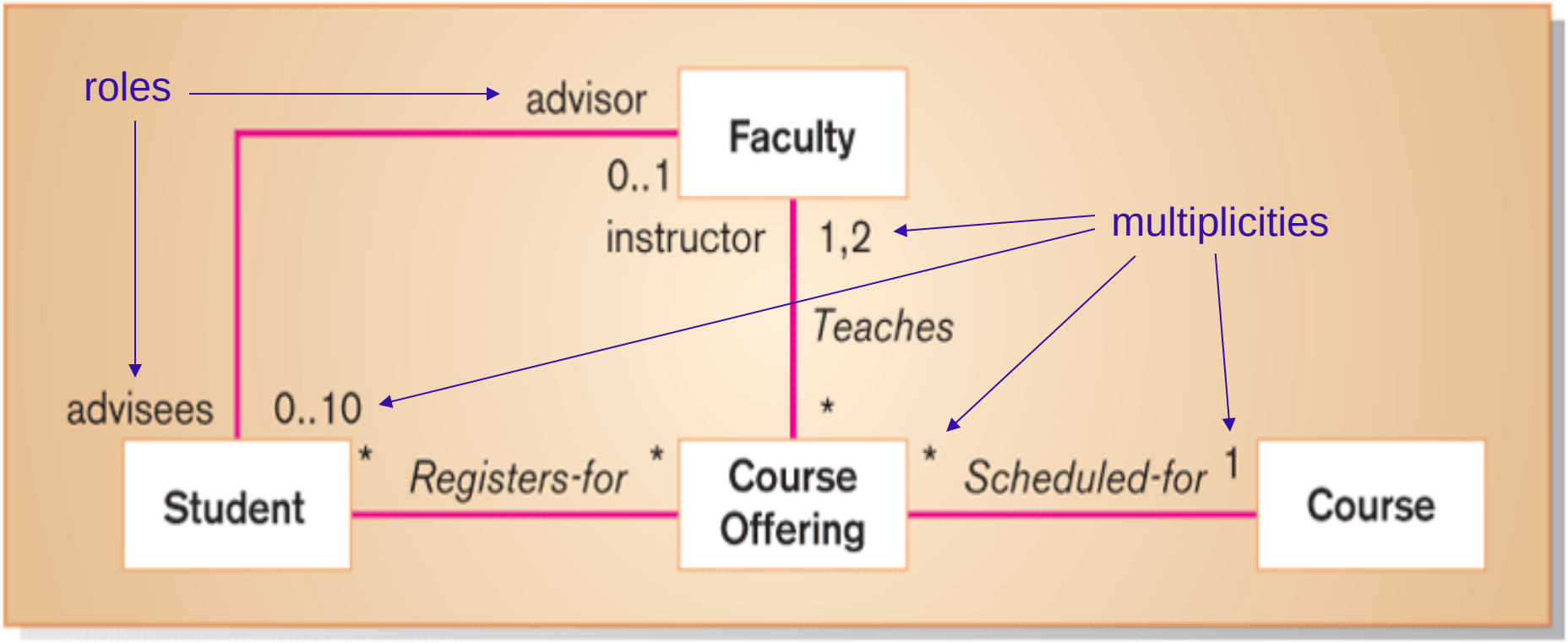
A **reference** is an attribute in one object that refers to another object



Mapping (role, association end) assigns the objects of one class to objects of another class



- ✓ Cardinality constraint notation in conceptual modelling indicates the minimum and maximum number of objects that must result from any one mapping instance. In UML, it is referred to as multiplicity.



Source:George J.F. et al., Object Oriented System Analysis and Design, Prentice Hall, 2004

Multiplicity notation:

0..10 means minimum of 0 and maximum of 10

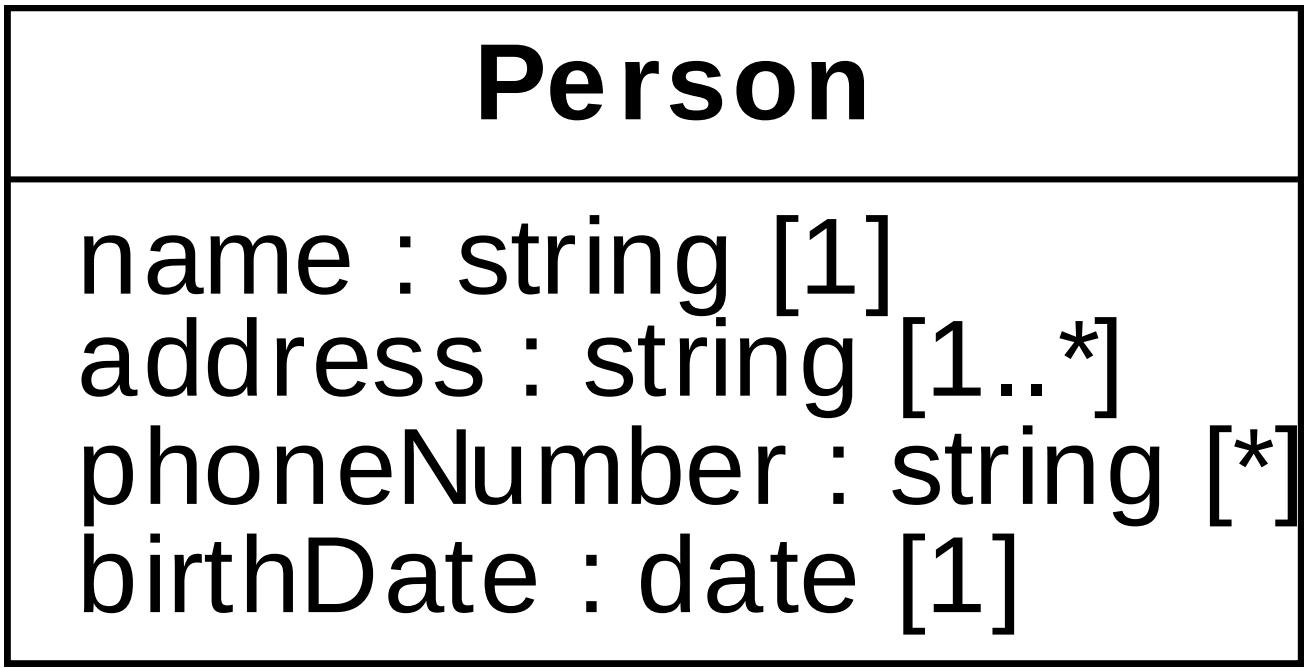
1, 2 means can be either 1 or 2

* means any number

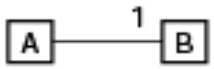
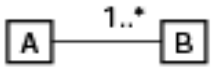
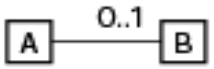
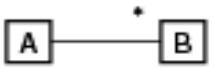
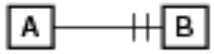
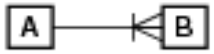
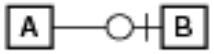
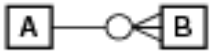
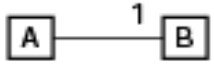
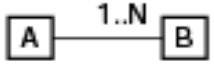
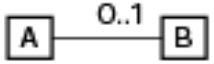
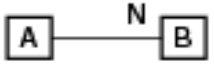
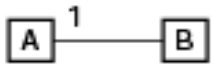
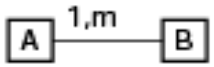
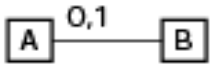
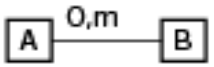
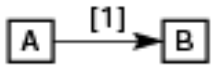
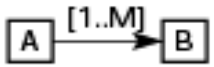
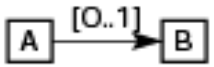
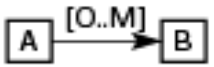
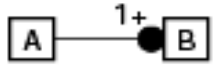
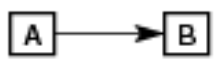
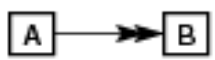
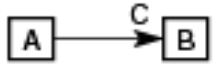
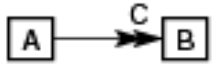
Multiplicity specifies the number of instances one class may relate to a single instance of an associated class

- 1 (**mandatory** mapping – each object **must** map to at least one object)
- 0..1 (**optional** mapping - each object **may** not map to any object)
- 1..* (**mandatory** mapping)
- * (**optional** mapping)
- 2..4 (**mandatory** mapping)

Multiplicity for an attribute



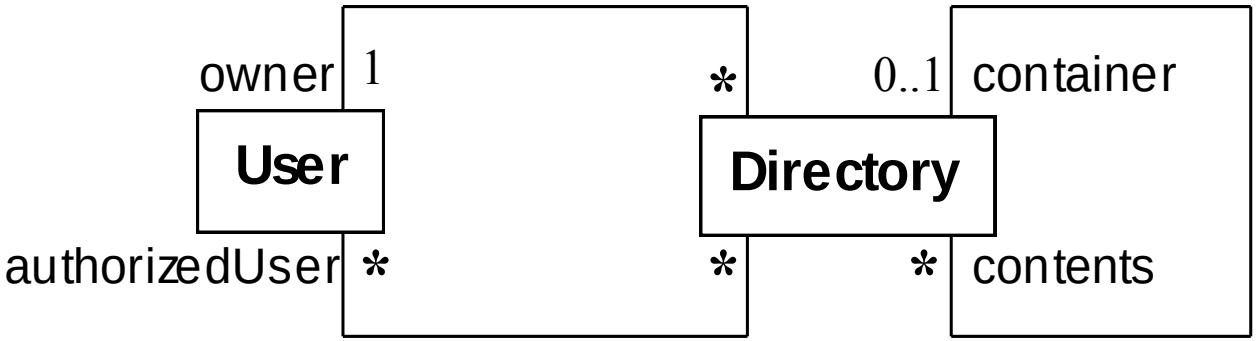
Mapping Notation Survey

Reading Left to Right	An A is always associated with one B	An A is always associated with one or many of B	An A is associated with zero or one of B	An A is associated with zero, one, or many of B
UML				
Martin/Odell (1st edition)				
Booch (2nd edition)				
Coad/Yourdon				
Jacobson (unidirectional)				
OMT				
Shlaer/Mellor				

Source: Martin/Odell, Object Oriented Methods: Foundations, Prentice Hall, 1998

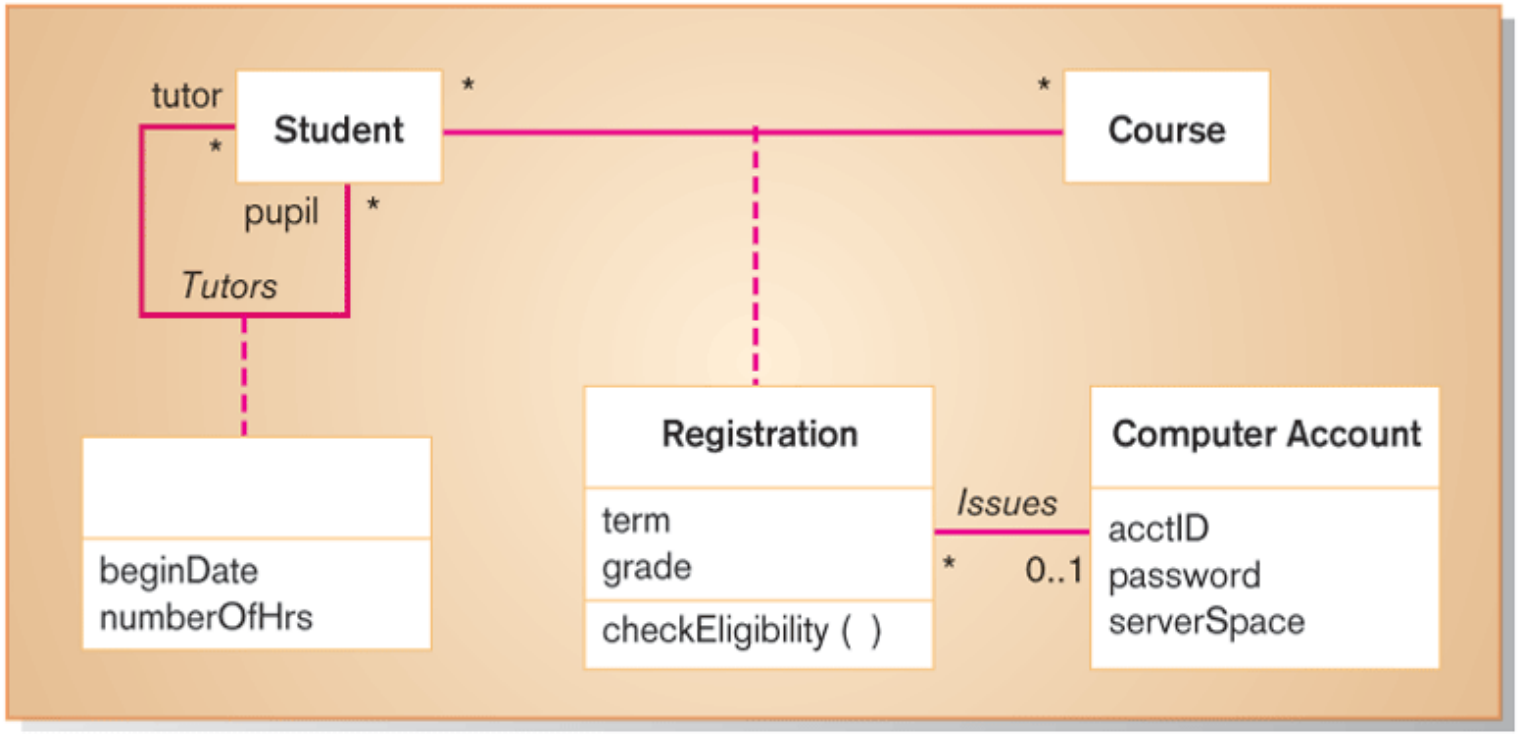
Mapping (association end) Names

- ✓ All names (of mappings) on the far end of association must be unique
- ✓ No association end name should be the same as an attribute name of the class (it is a pseudo attribute of the source class)



Association Class

An association with its own attributes, operations, or relationships



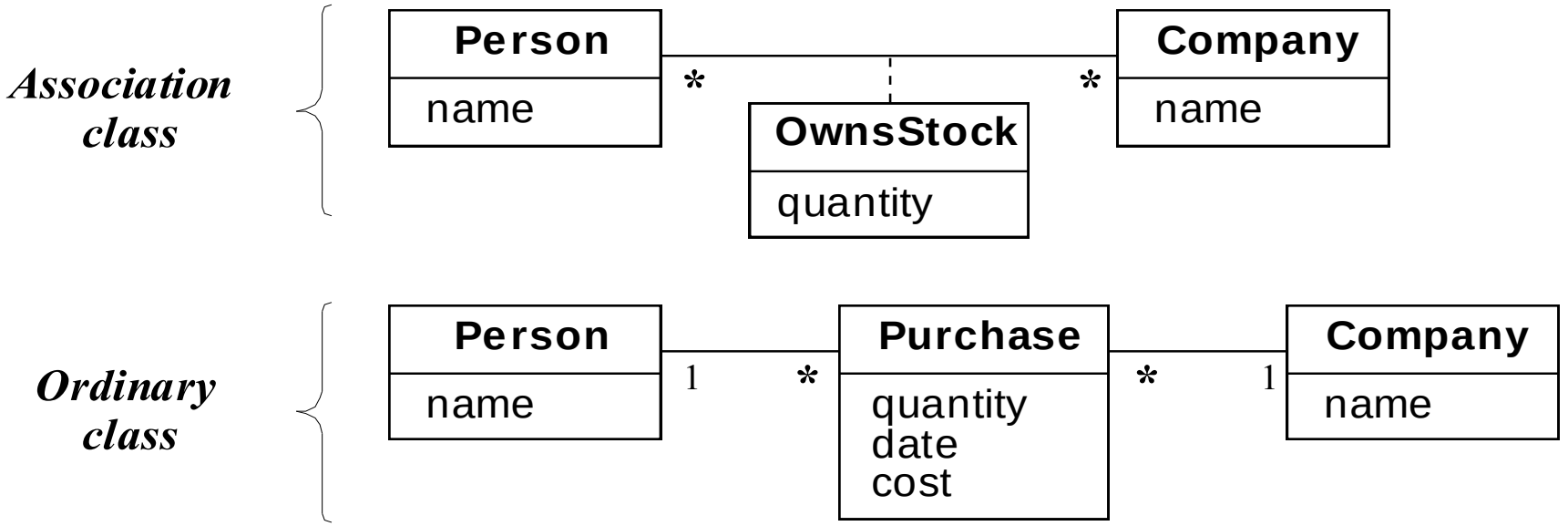
UML association classes are analogous to E-R associative entities.

Source:George J.F. et al., Object Oriented System Analysis and Design, Prentice Hall, 2004

Association Class (cont.)

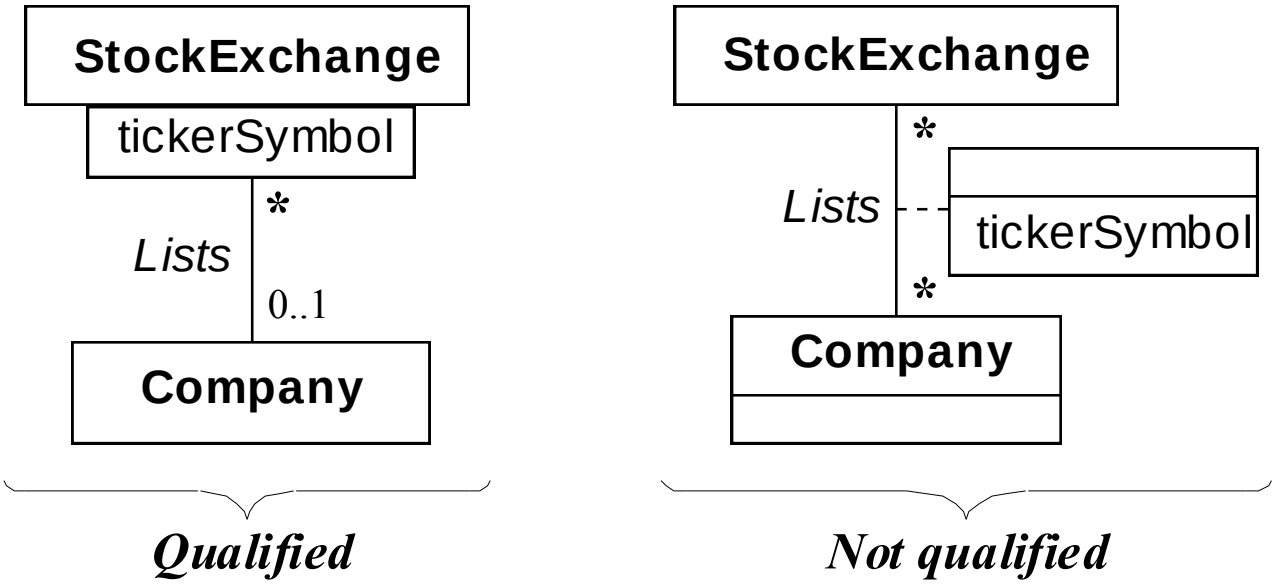
- Association class has only one occurrence for each pair of objects from two classes

  (there can be any number occurrences of Purchase for each Person and Company)



Qualified Association

is association with an attribute called qualifier. It selects among the target objects, reducing multiplicity from ‘multi valued’ to ’single valued’.



Superclass and Subclass

- Subclass: a more specific entity that inherits features of a superclass. It is usually provided by distinct attributes or relationships from other subclasses
- Superclass: a more generic entity for one or more subclass

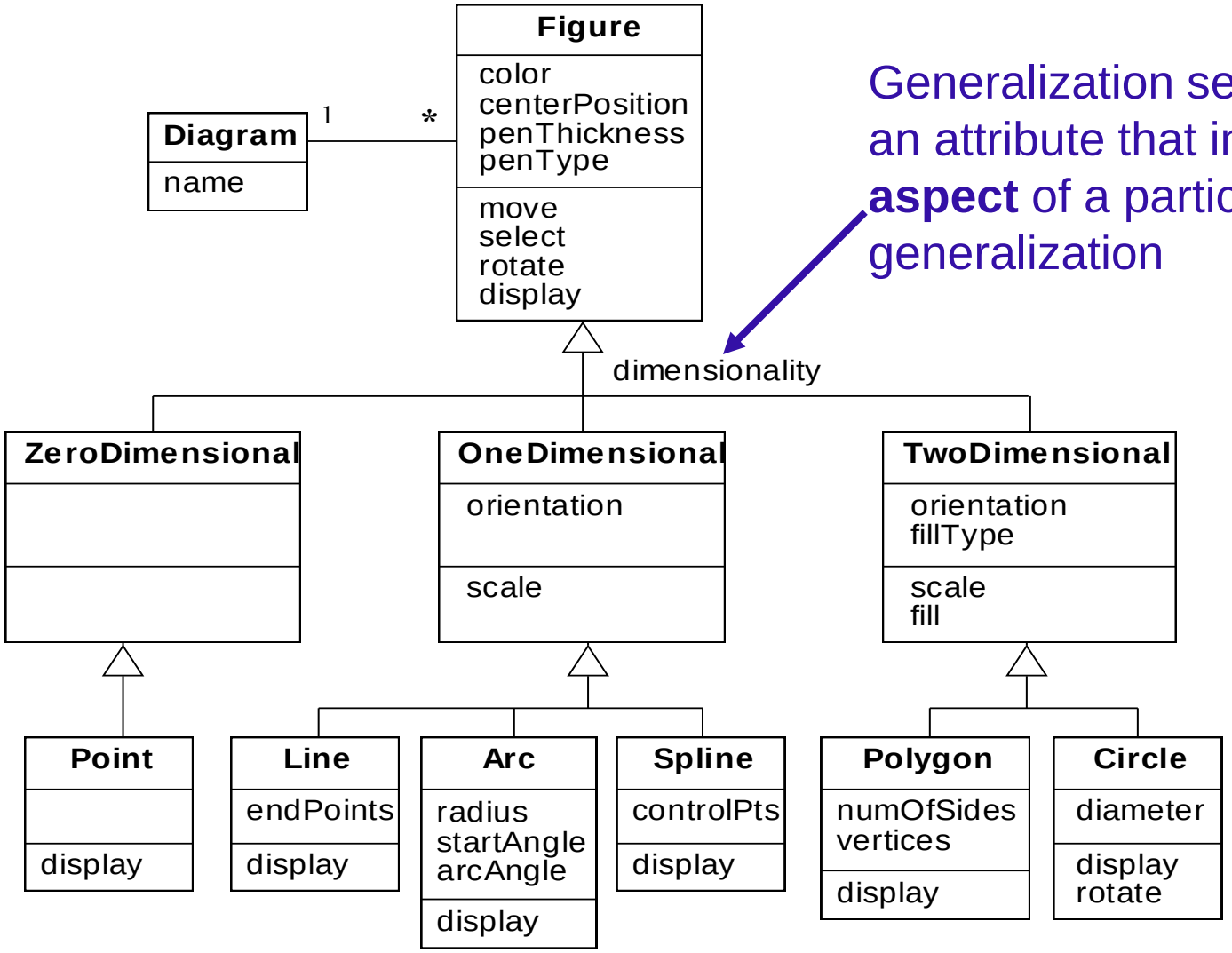
Generalization and Inheritance

- ✓ Generalization is the relation between superclass and subclass
- ✓ Subclass **inherits** attributes, operations, and associations of the superclass
- ✓ Types of superclasses
  - Abstract: cannot have any direct instances
  - Concrete: can have direct instances

Use of Generalization

1. Support for Polymorphism.
   - OO language compiler automatically resolves the call to the method according to the calling object class
2. Model for object taxonomy
   - Organising objects according their similarities and differencies
3. Enables reuse of code
   - If you add a new class, it automatically inherits a superclass behaviour

Generalization organizes classes by their similarities and differences



Overriding

- A subclass may **override** a superclass feature by defining it with the same name.
  - Overriding of methods and default values of attributes.
  - Signature of operation cannot be overriden
  - The same operation may apply to two or more classes in different ways
  - Overriden features should be not inconsistent
- Abstract operations
  - defined in abstract classes
  - defined the protocol, but not the implementation of an operation

Navigation of Class Diagrams

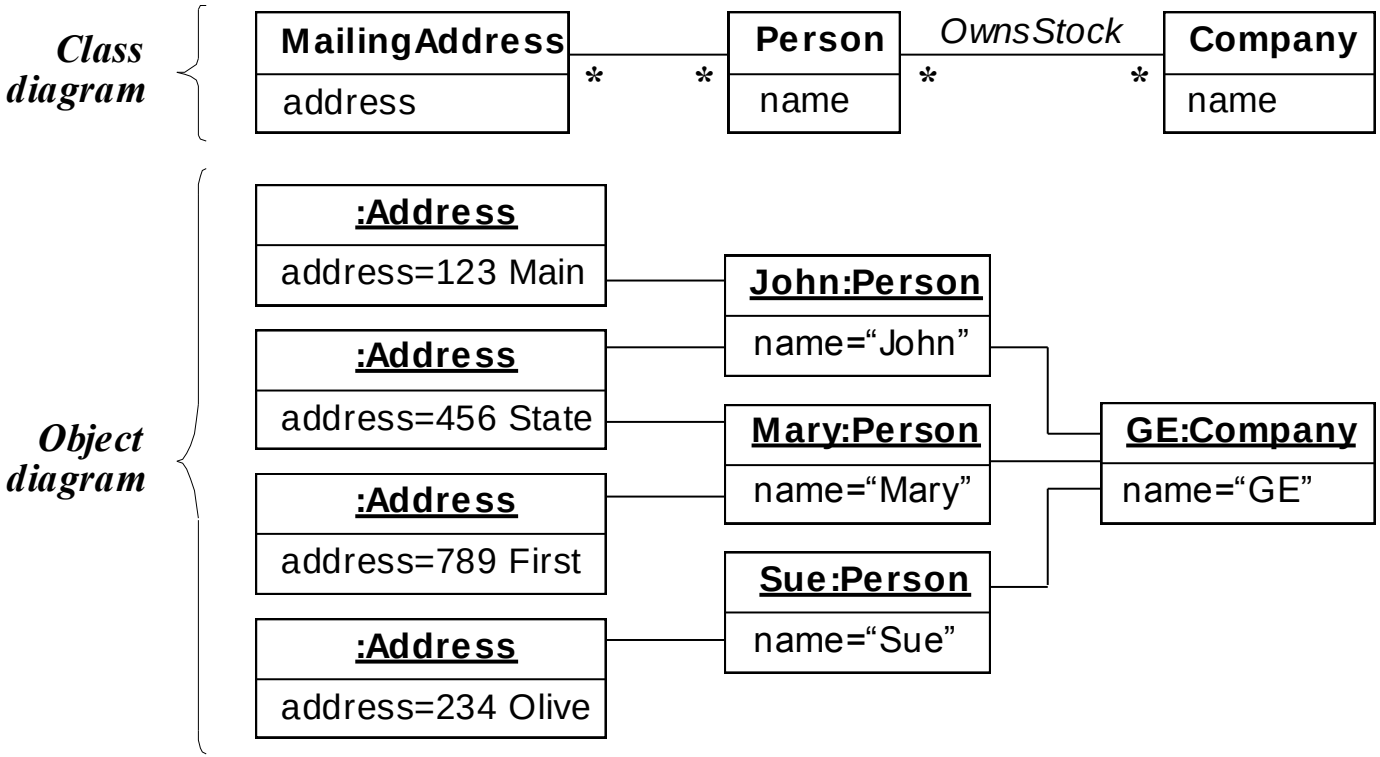
- ✓ Is important because it lets system designer to uncover hidden flaws and omissions by exercising a model
- ✓ UML incorporates Object Constraint Language (OCL)
- ✓ OCL expressions could navigate several association chains.
- ✓ Filters, gualifiers and operations can be used in OCL constructs

OCL expressions

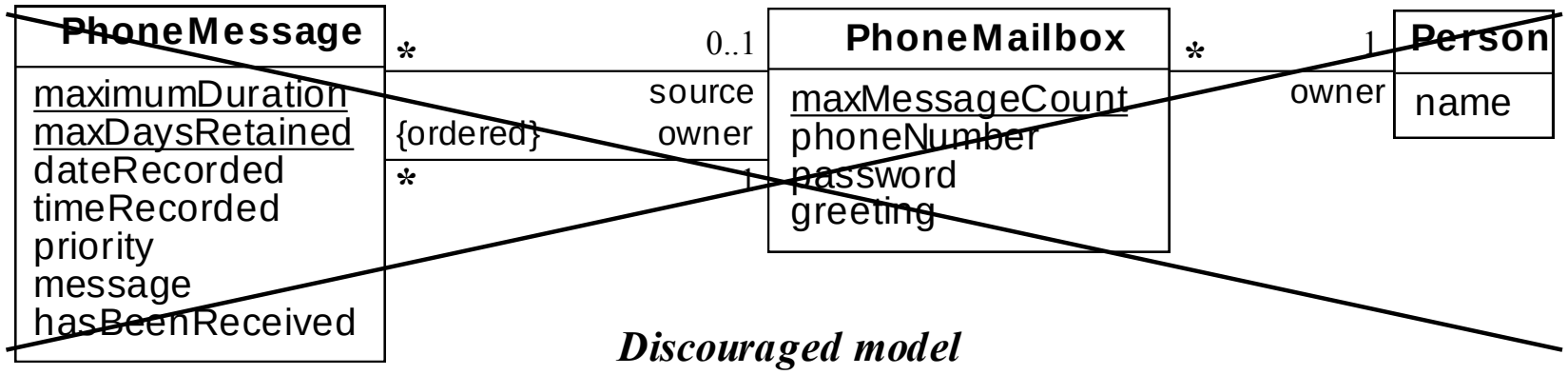
Expression **aCompany.Person.MailingAddress**

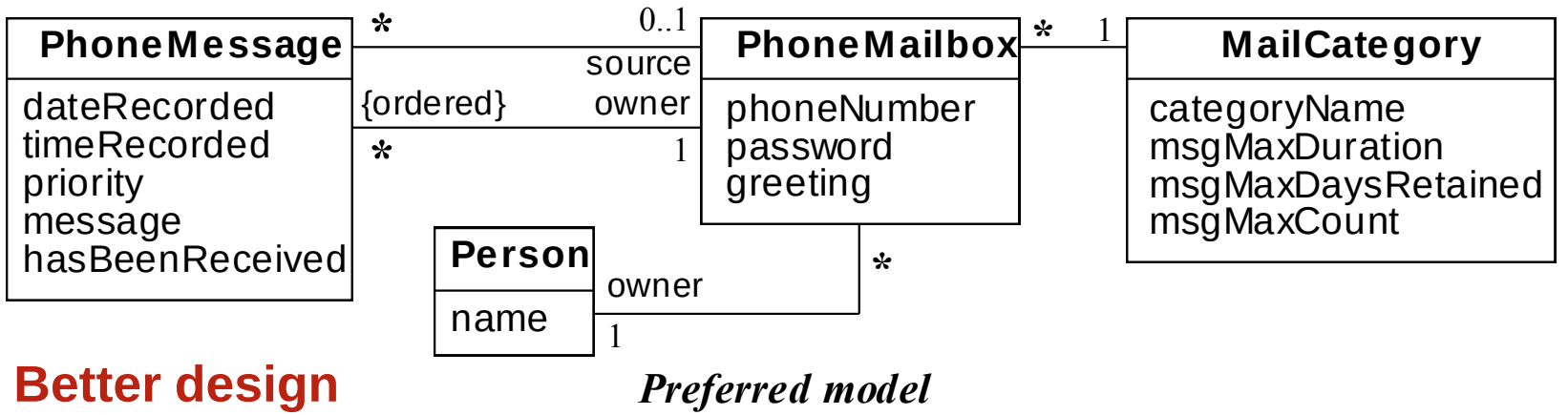
yields a set of addresses for a company (GE:Company).

- Traversals through associations may yield a bag



An underline distinquishes attributes with a **class scope**





Better design

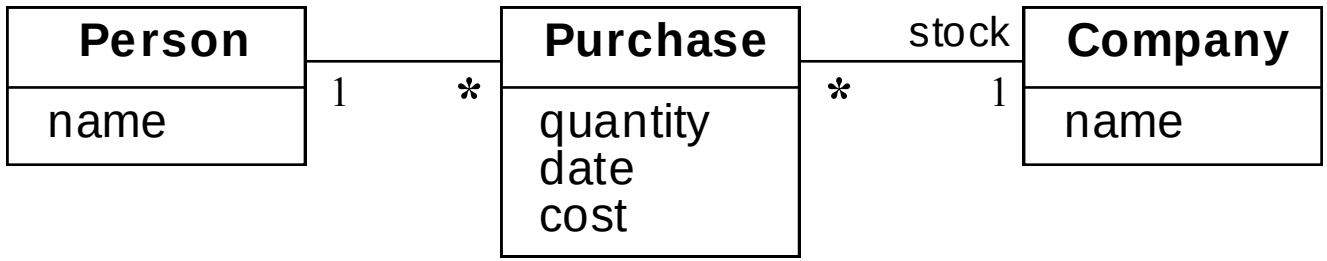
New classes with object scope attributes can be introduced for groups

N-ary associations

- ✓ Associations among three and more classes
- ✓ Most of them can be decomposed into binary associations

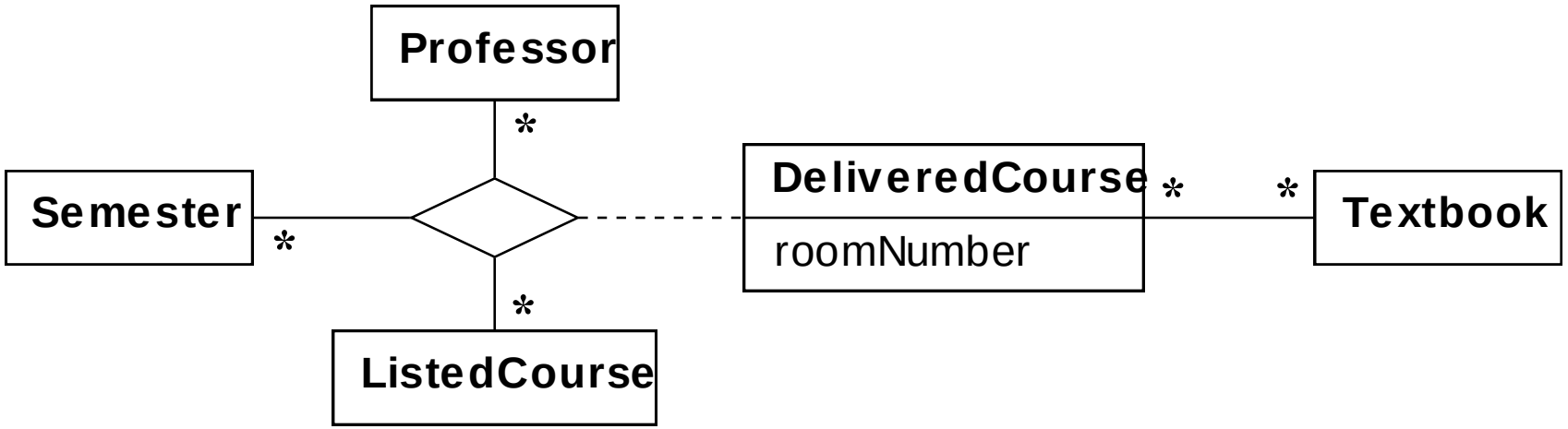
A nonatomic n-ary association—a person makes the purchase of stock in a company...

Can be restated as...

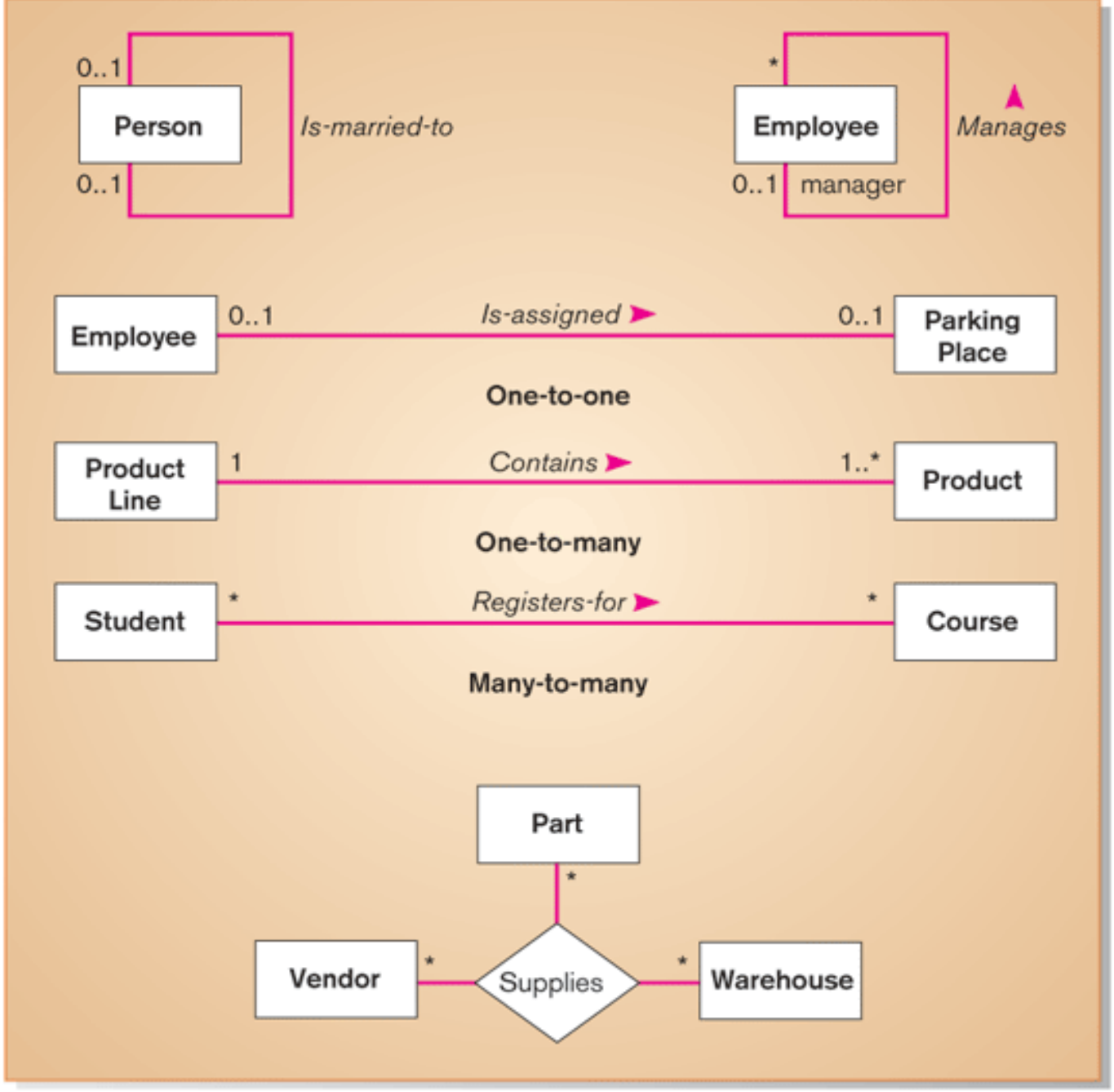


N – ary associations can be viewed as association classes

- ✓ Cannot be traversed like binary associations
- ✓ OCL does not support notation for traversing n-ary associations



Associations of different degrees



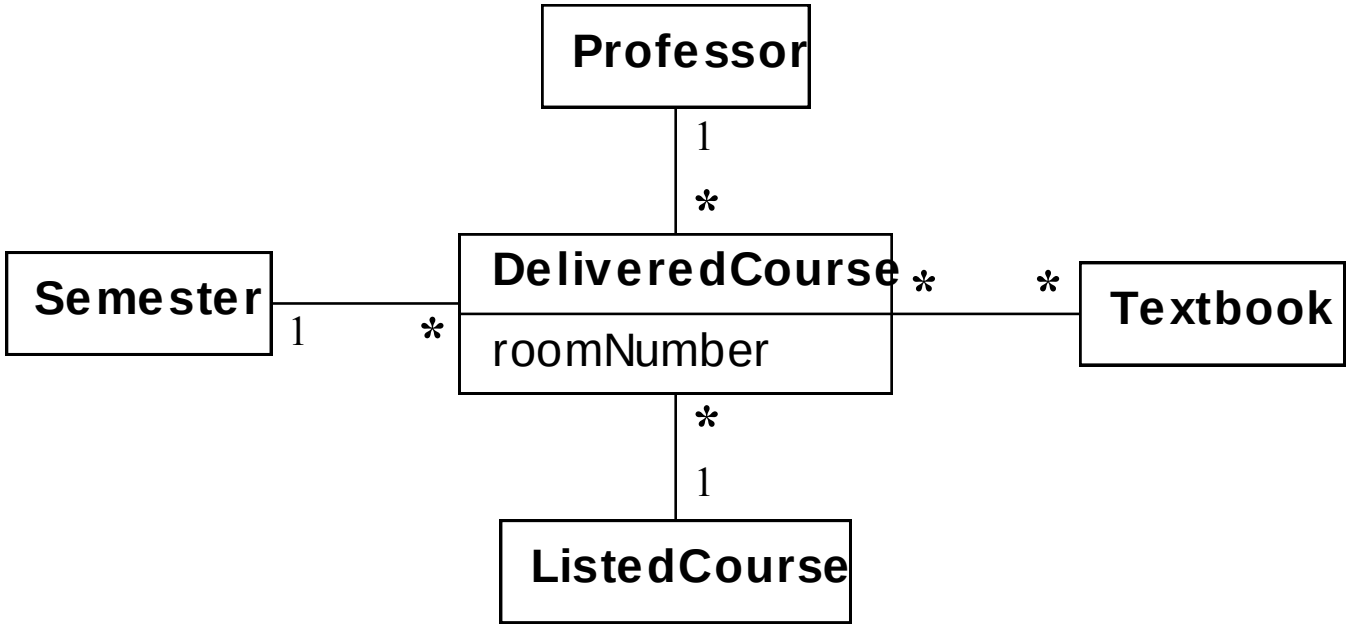
UML associations are analogous to E-R relationships.

UML multiplicities are analogous to E-R cardinalities.

Source:George J.F. et al., Object Oriented System Analysis and Design, Prentice Hall, 2004

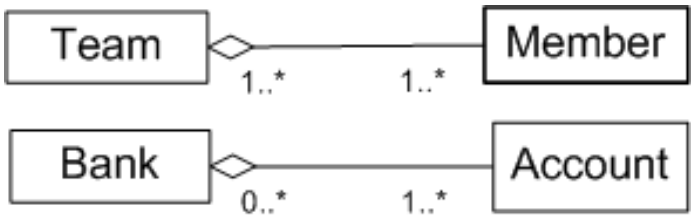
N-ary associations as classes

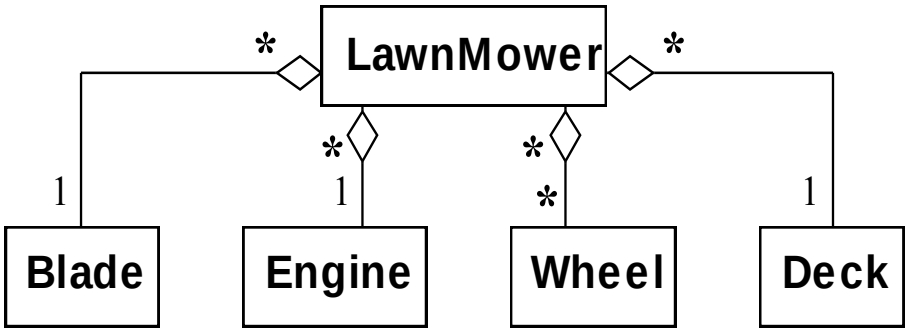
- ✓ Typical programming language cannot express n-ary associations



Aggregation

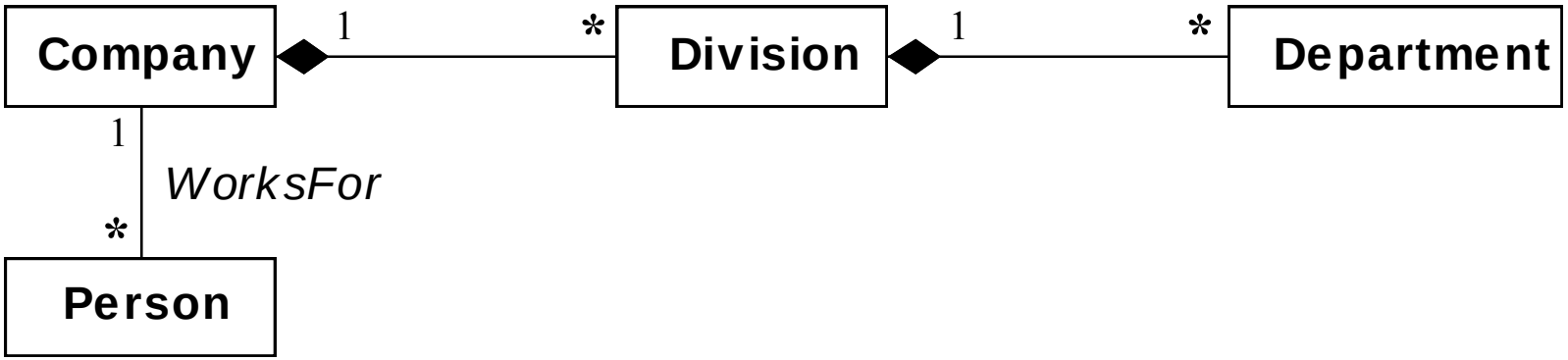
- ✓ is a strong form of association in which an aggregate object is formed using other objects as parts
- ✓ An aggregate object is treated as a unit in many operations





Composition

- ✓ Is a form of aggregation with two additional constraints
- ✓ Deletion of an assembly objects triggers deletions of all constituent objects



Composition and Aggregation

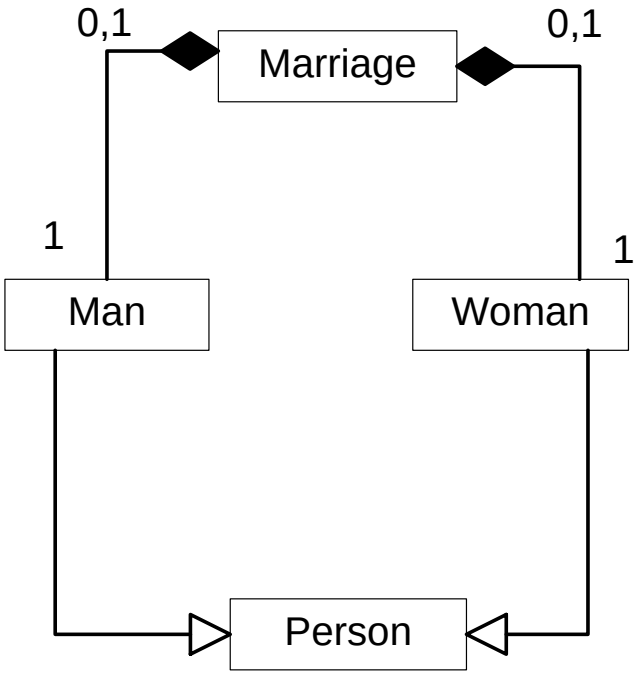
Properties of Aggregation:

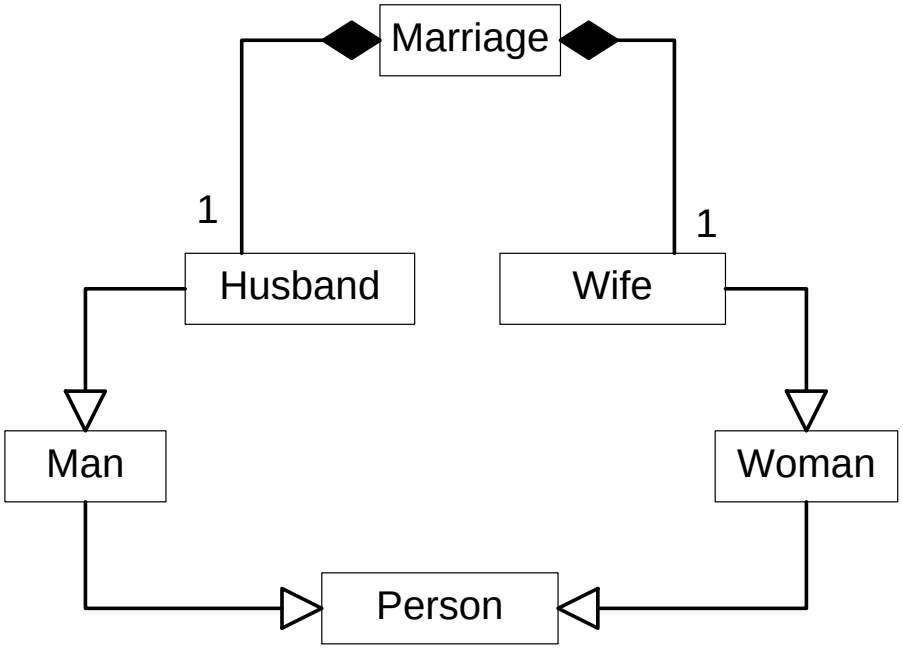
- Transitive
- Asymmetric (if A is a part of B, then B is not part of A)

Properties of Composition:

- Transitive
- Asymmetric
- Part is existent dependent on a whole
- Part cannot have more than one whole

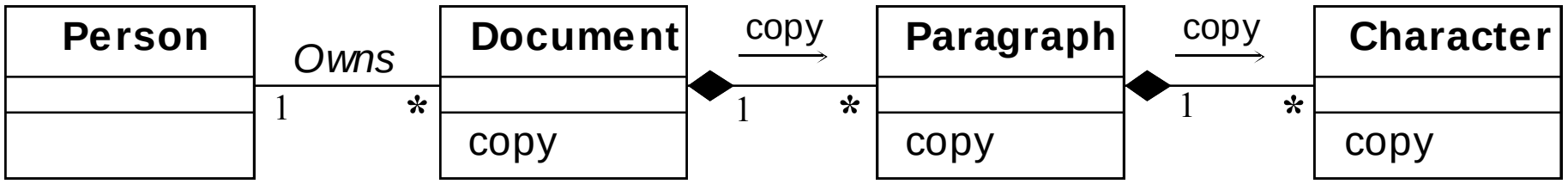
Composition: Two cases





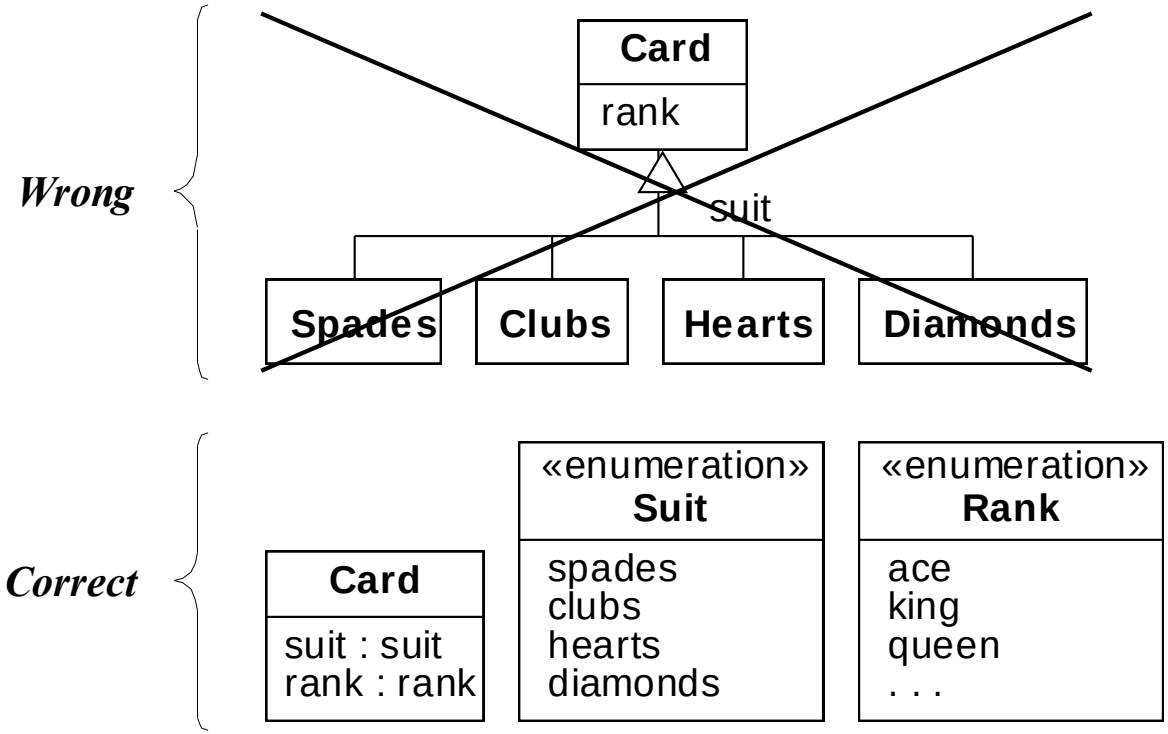
Propagation of Operations

- ✓ Propagation is a powerful way for specifying continuum of behavior
- ✓ Operations are propagated to parts via composition relations



Enumeration

- ✓ Data Types include numbers, strings and enumerations.
- ✓ An enumeration is a data type that has a finite set of values.

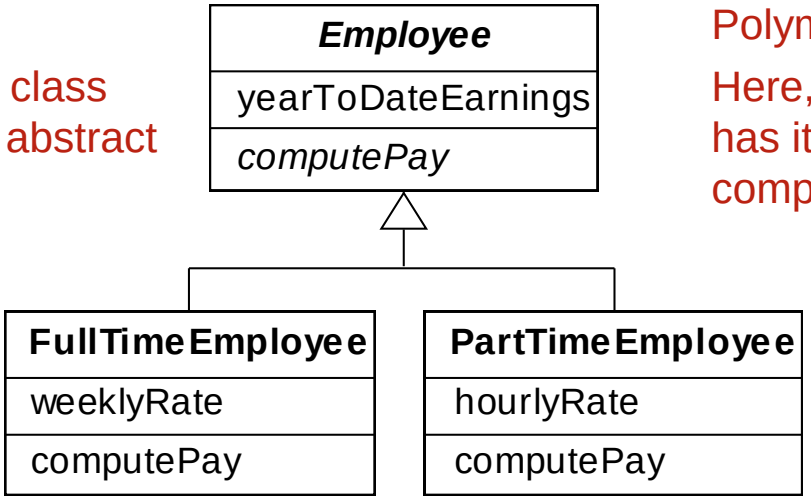


Abstract Classes

- ✓ Abstract class has no direct instances but more specific classes have direct instances
- ✓ A concrete class can have direct instances. Only concrete classes may be leaf classes in a generalization hierarchy

Abstraction:

Employee is an abstract class and *computePay()* is an abstract operation (italicized)

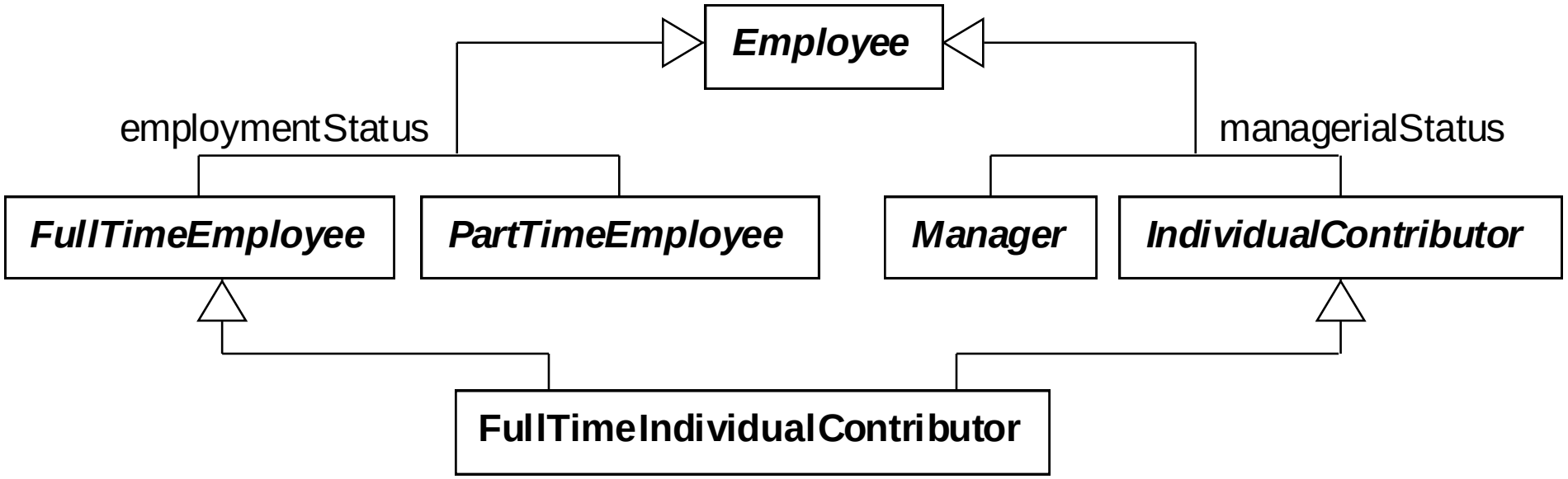


Polymorphism:

Here, each type of Employee has its own version of computePay()

Multiple Inheritance

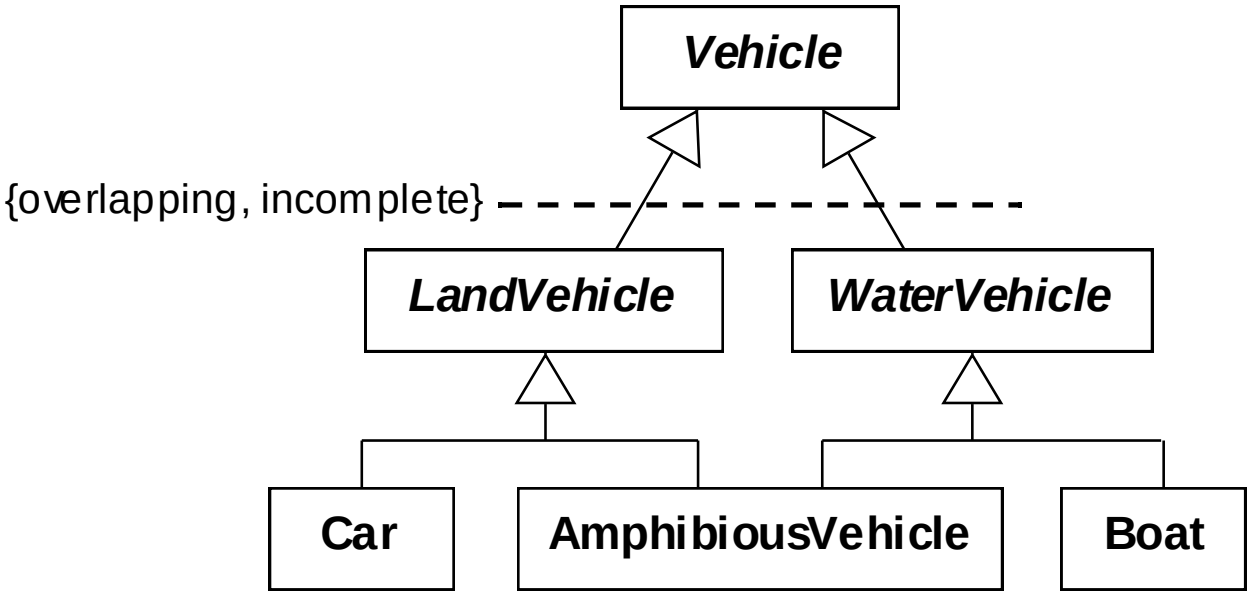
- ✓ permits a class to have more than one superclass



Multiple inheritance from sets of disjoint classes is impossible

Multiple Inheritance

- ✓ Conflicts between features of different inheritance paths must be resolved

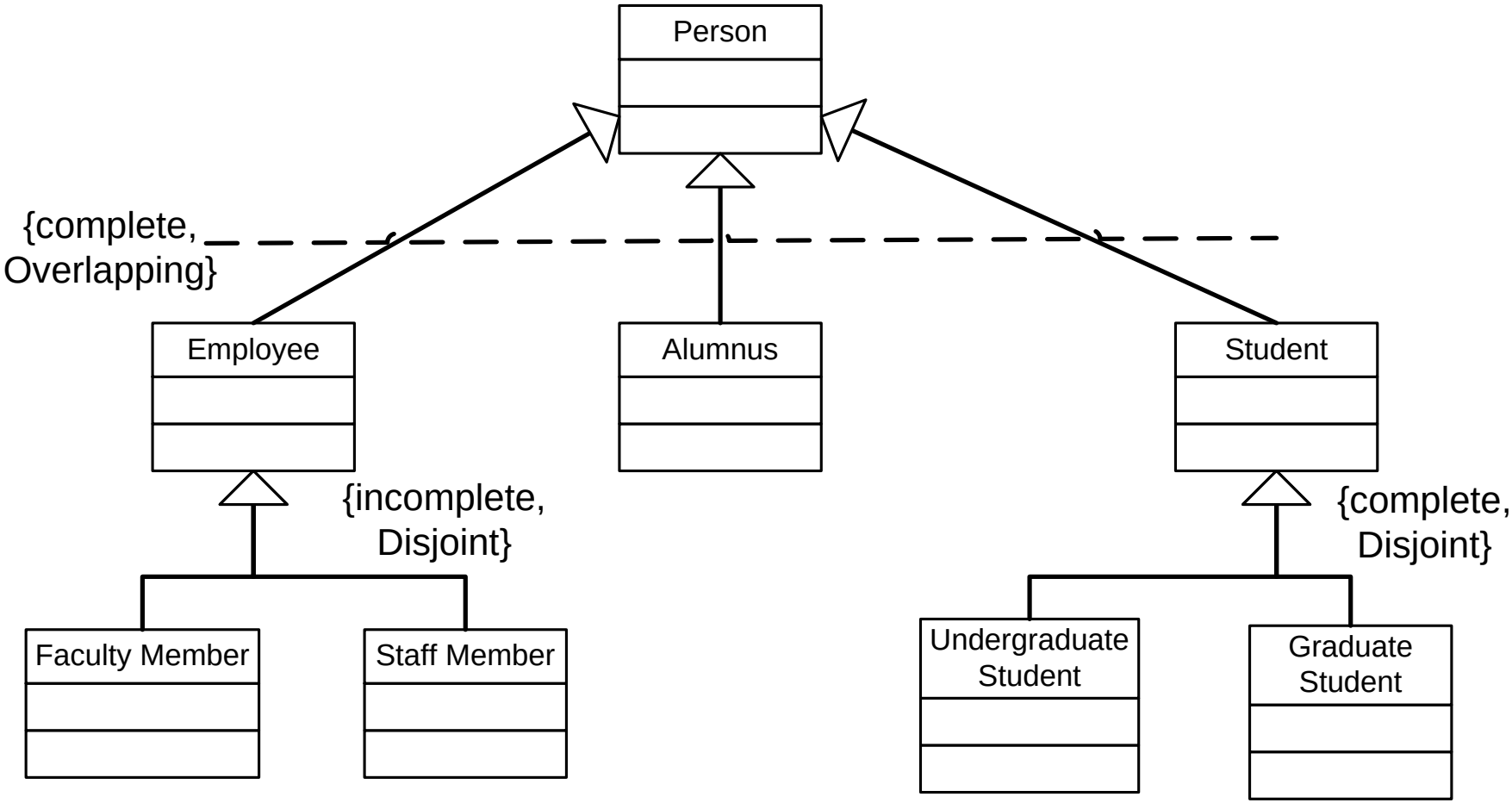


Multiple inheritance from overlapping classes

Constraints for Superclass/Subclass Relations

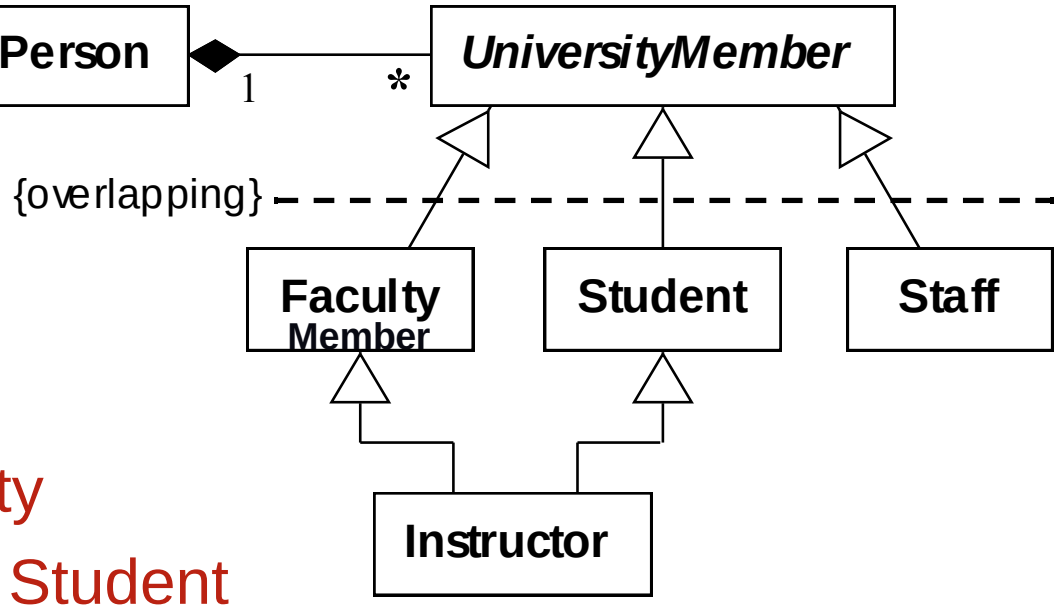
- ✓ Complete (Total). All subclasses have been specified. No additional subclasses are expected
- ✓ Incomplete (Partial) The list of subclasses is incomplete. Additional subclasses are expected
- ✓ Disjoint (partition): Subclasses are mutually exclusive. Each object can be instantiated in only one subclass
- ✓ Overlapping: Subclasses can share objects. An object can belong to more than one subclass

Superclass/Subclass Relations



Multiple Classification (MC)

- ✓ An instance of a class is inherently an instance of a superclass (MC is permitted in UML)
- ✓ Most OO languages handle multiple classification poorly (delegation and nested generalization)

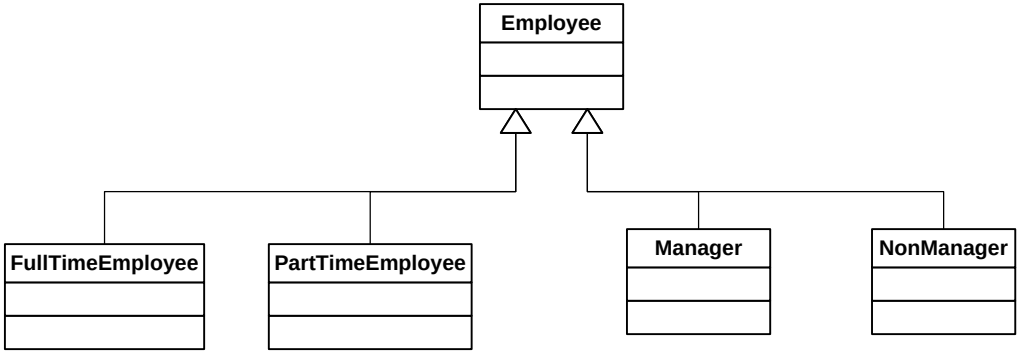


e.g.:

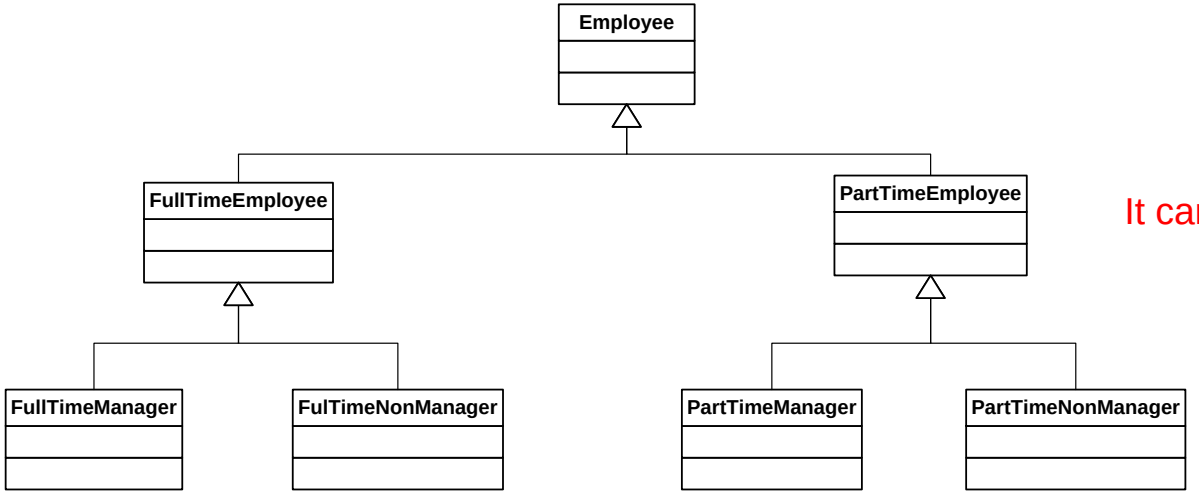
an Instructor

is both Faculty Member and Student

Nested Generalization



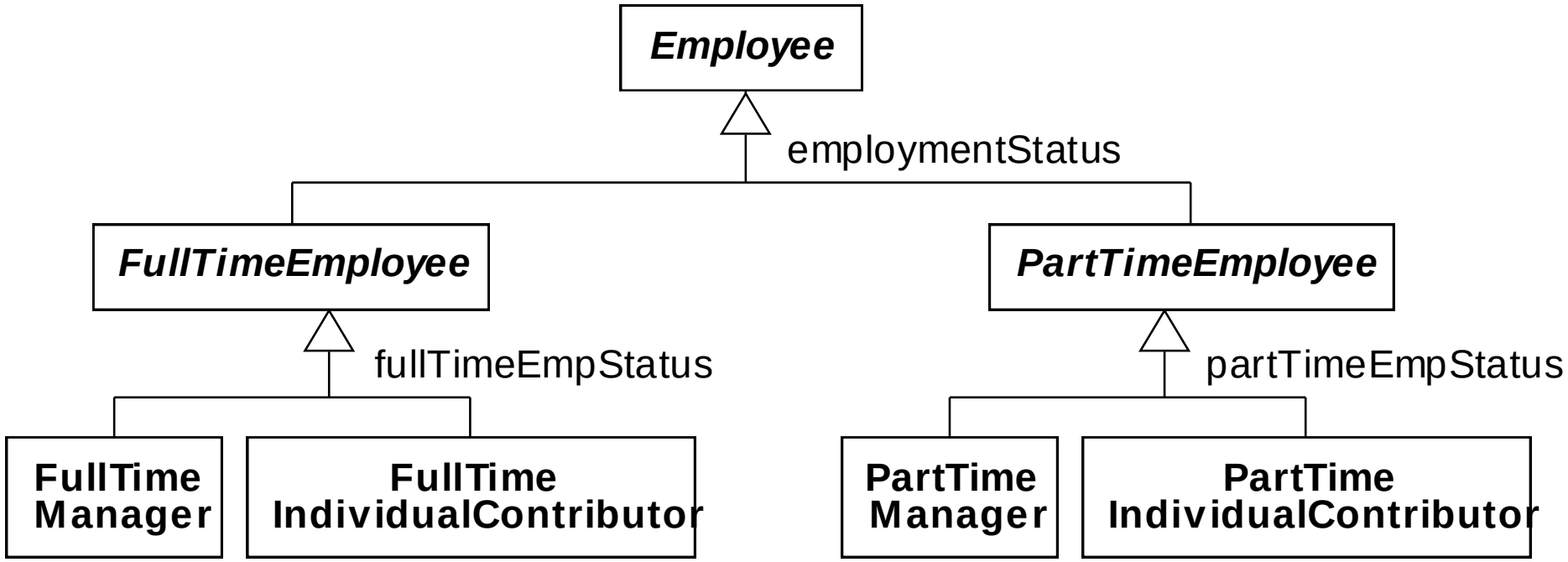
Nested generalization is a workaround for a multiple classification, which is not handled by object-oriented languages. For instance, we cannot make an instance of FullTimeEmployee and Manager at the same time.



It can be done in this schema

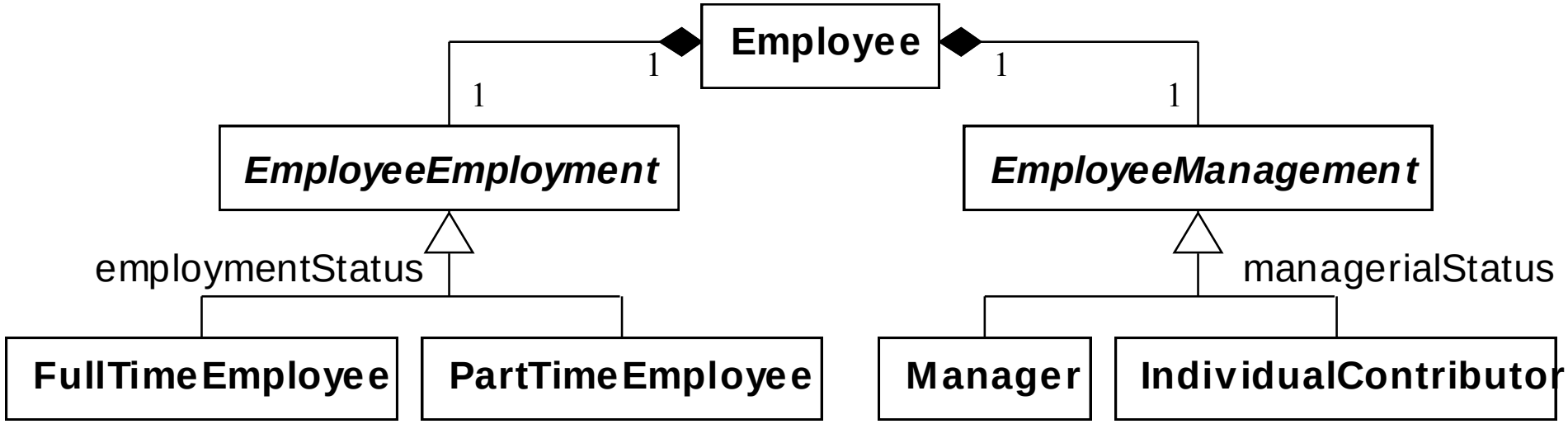
Nested Generalization

- ✓ Preserves object identity, but duplicates declarations (and code)



Delegation

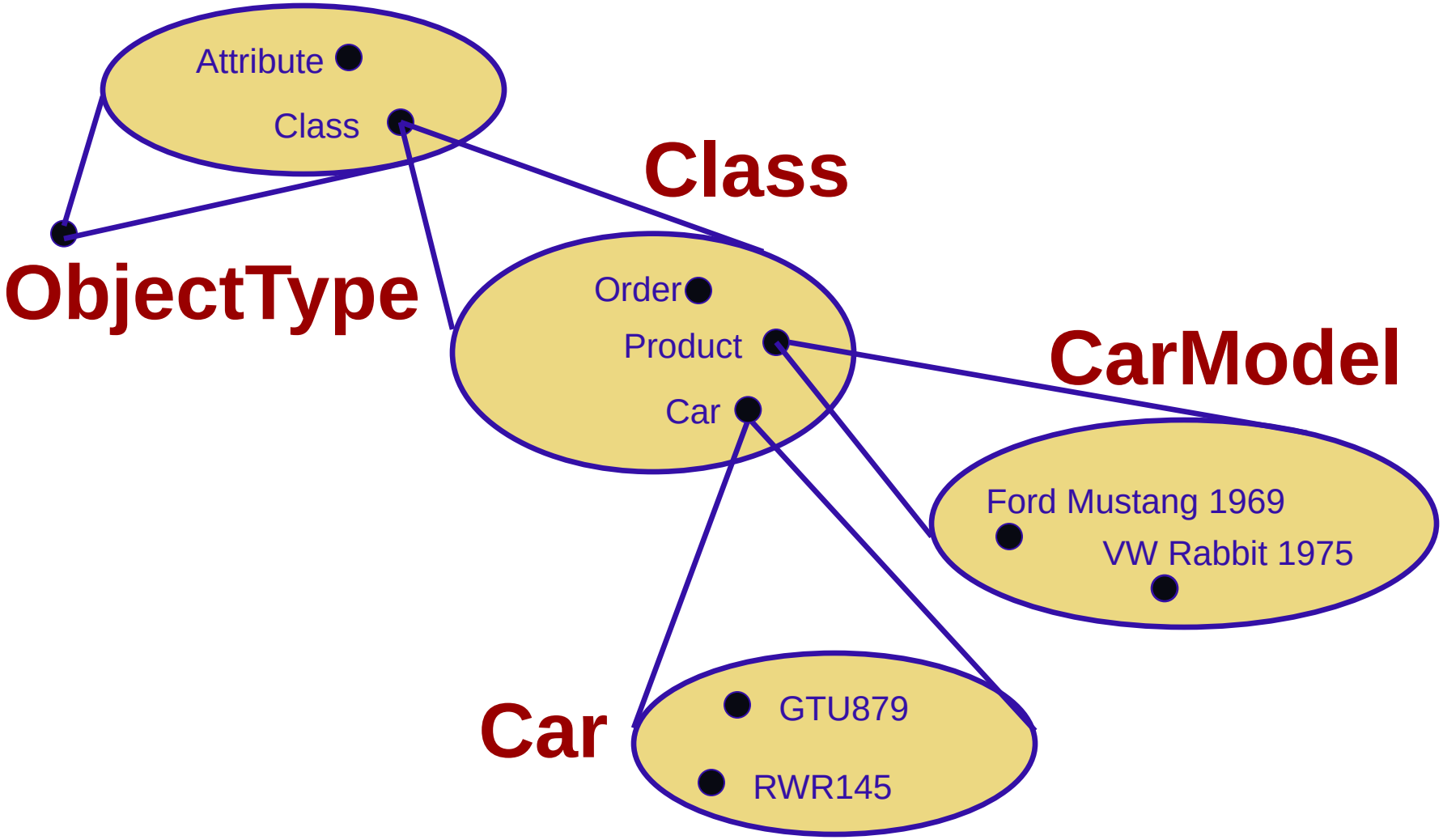
- ✓ Replaces a unique object identity by an artificially created group of objects
- ✓ The composite object must catch operations and delegate to appropriate part



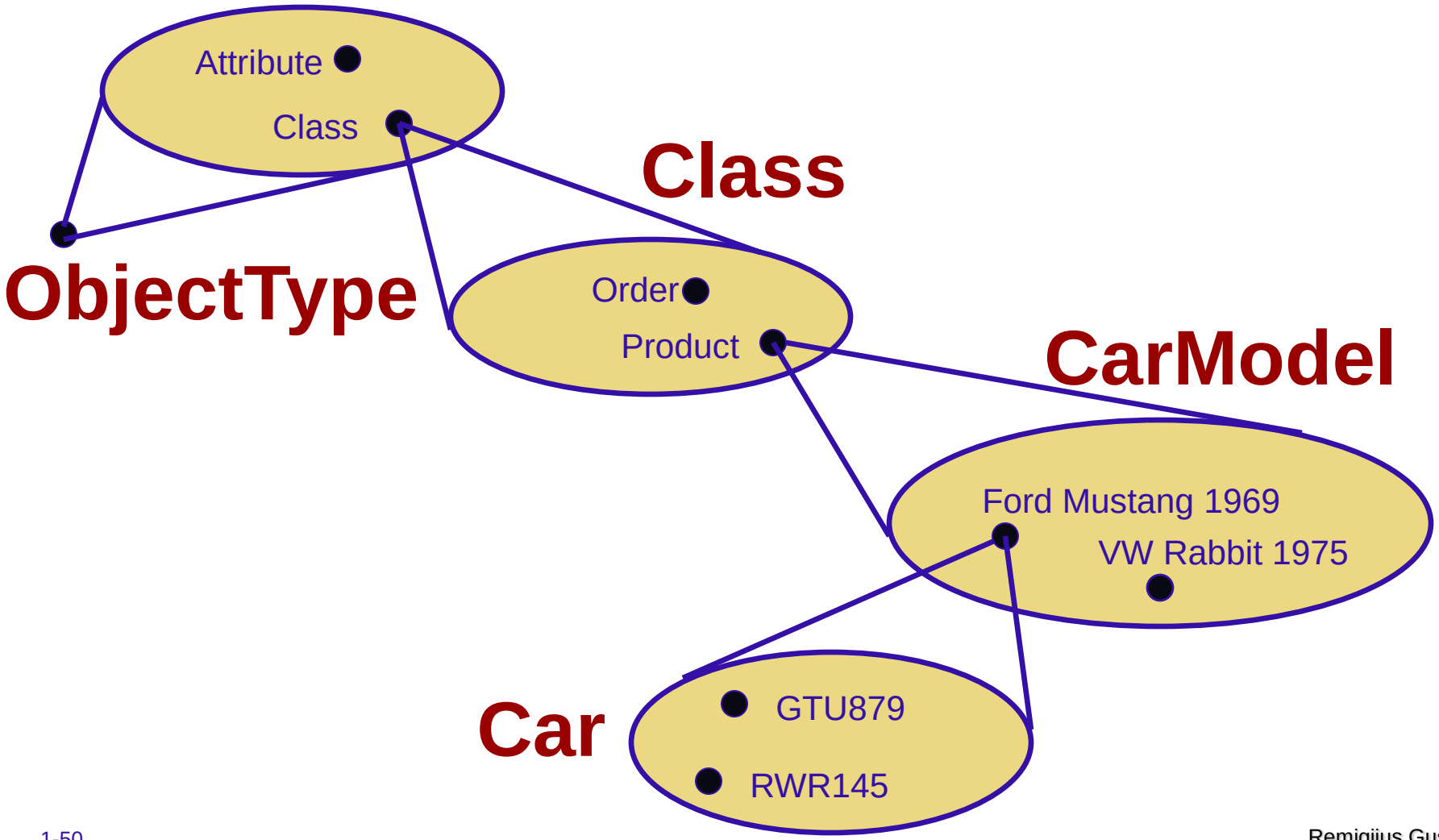
Conventional Modelling Levels

Level Title	Describes
Object level	Data and Processes
Modeling level	Metadata about data and processes
Meta-modeling level	Metadata about object types

Three Modeling Levels

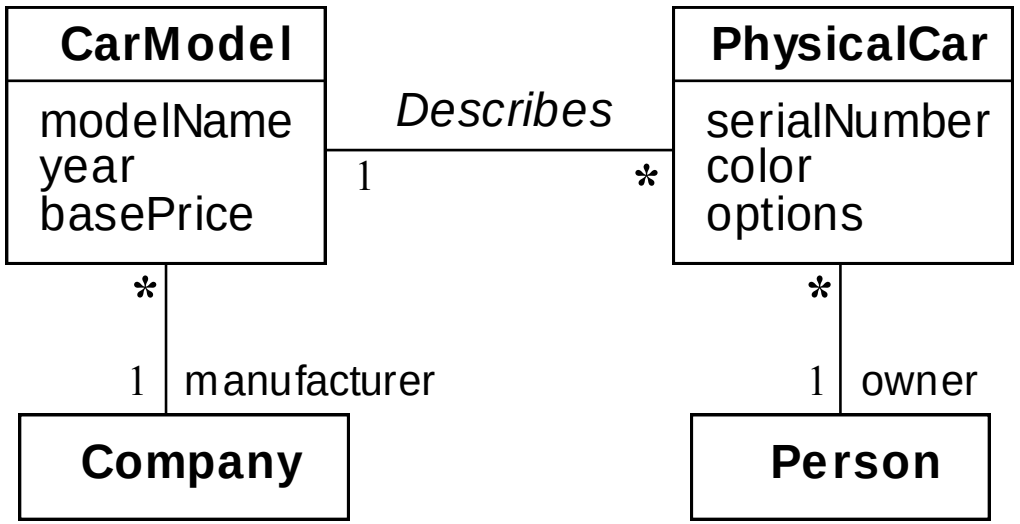


Instances of meta-classes can be viewed as classes



Metadata and Meta-classes

- ✓ Metadata is a description about other data
- ✓ Class definition is metadata
- ✓ Classes can be considered as objects, which in turn have their own classes (meta-classes)

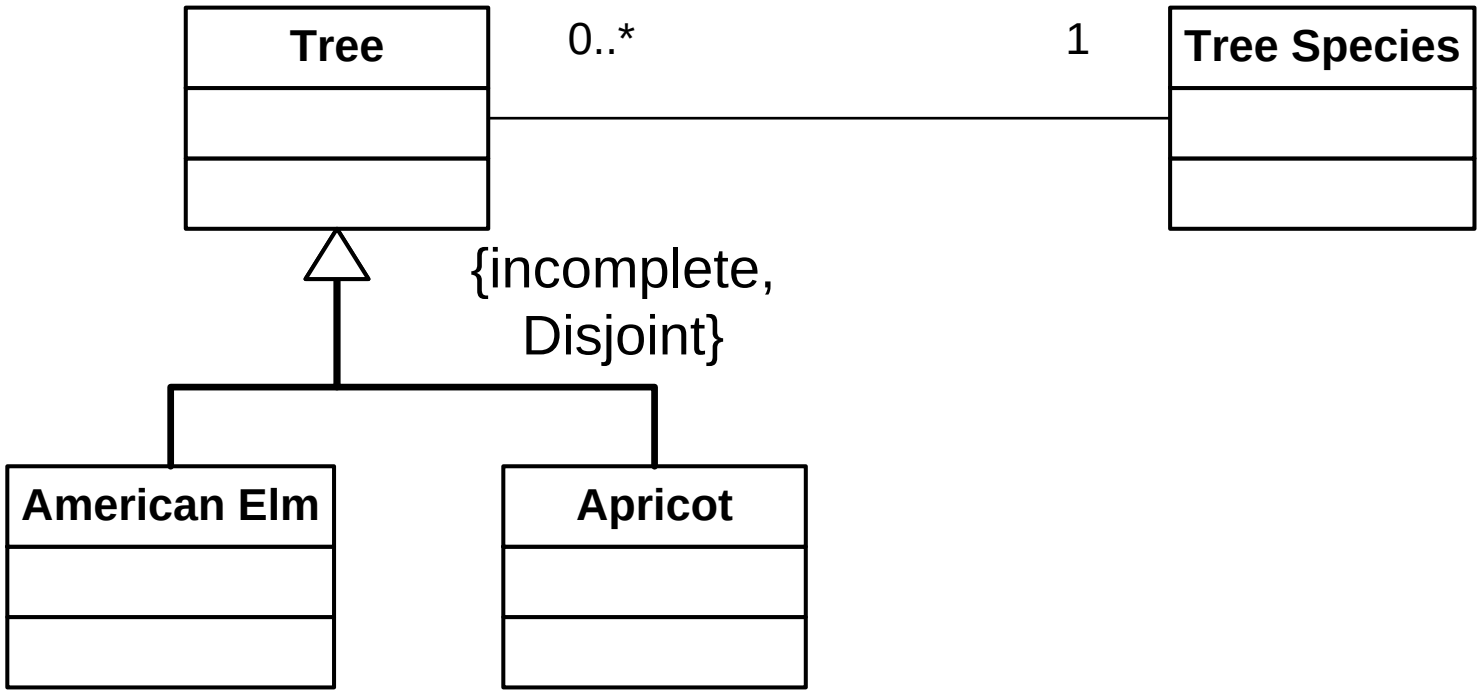


Problems with conventional tools

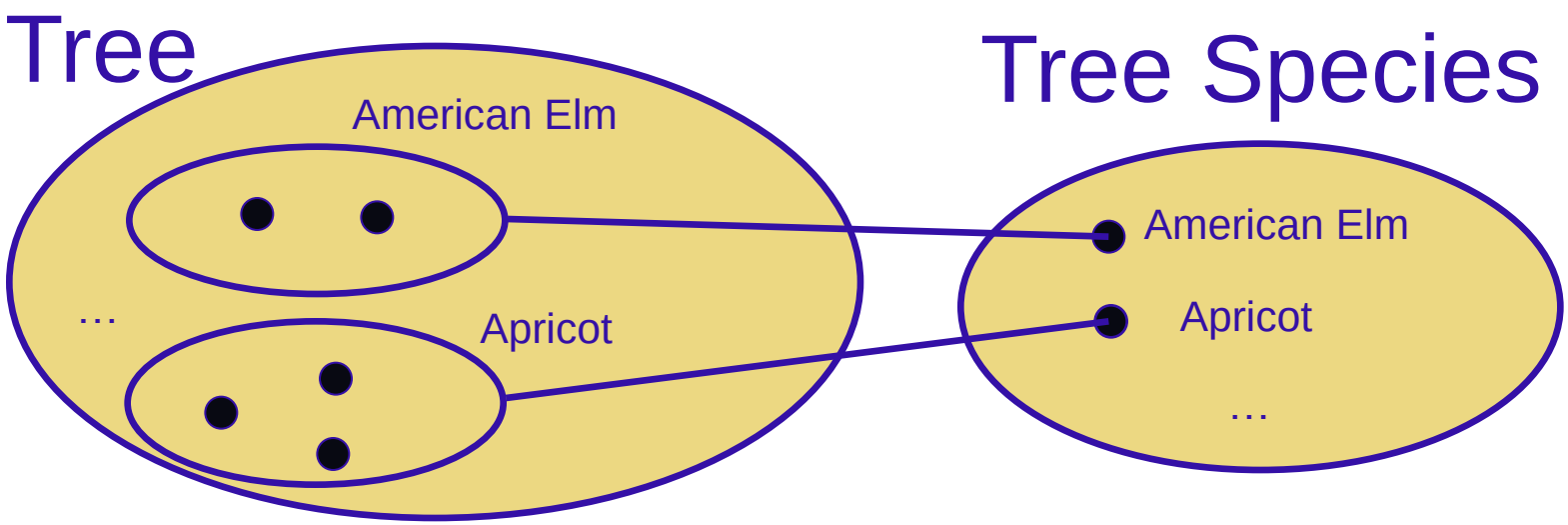
1) Most Case tools maintain data and metadata as physically separate levels (semantic gap between two levels of abstraction).
2) Modifying of meta-meta-data is usually impossible. Some OO languages (Lisp, Smaltalk) allows model inspection and alteration at run time (C++, Java deal with metadata at compile time)
3) Tools typically support three-level modeling, adding levels is typically impossible

Objects as Classes

A **power type** is a class whose instances are subtypes of another class



Power Types



- ✓ All objects types are objects, but not all objects are objects types
- ✓ Modeling of Power Types is not expressed by Object-oriented approaches

Reification

- ✓ promotion of something that is not an object (class, attribute, method, constraint, control information) into object
- ✓ can be helpful for meta applications

For instance, various data management services can be reified for providing a general purpose solution to access data for multiple users.

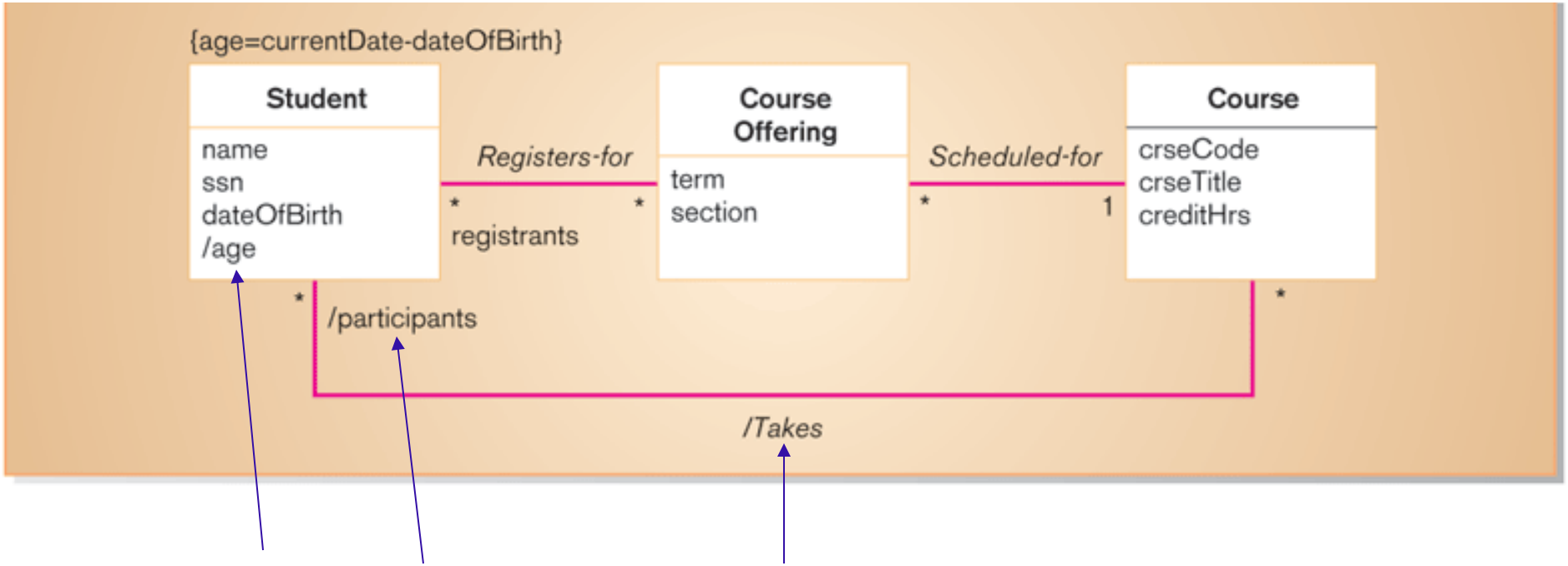
For instance, such classes as TV, CD-player could be promoted to objects for in the Product class

The issue of maintaining integrity should be addressed when adding, modifying or removing products

Derived Element

- ✓ is a function of one or more base elements.
- ✓ A derived element is redundant, because the base elements determine it.
- ✓ Classes, associations and attributes can be derived.
- ✓ The notation of derived element is a slash in front of the element name.
- ✓ The derivation rules (as other constraints) can be represented in a ’dog-eared’ comment box or delimited with braces

Derived Attributes and Associations



Derived items are represented with a slash (/).

Derived attributes are calculated based on other attributes

Packages

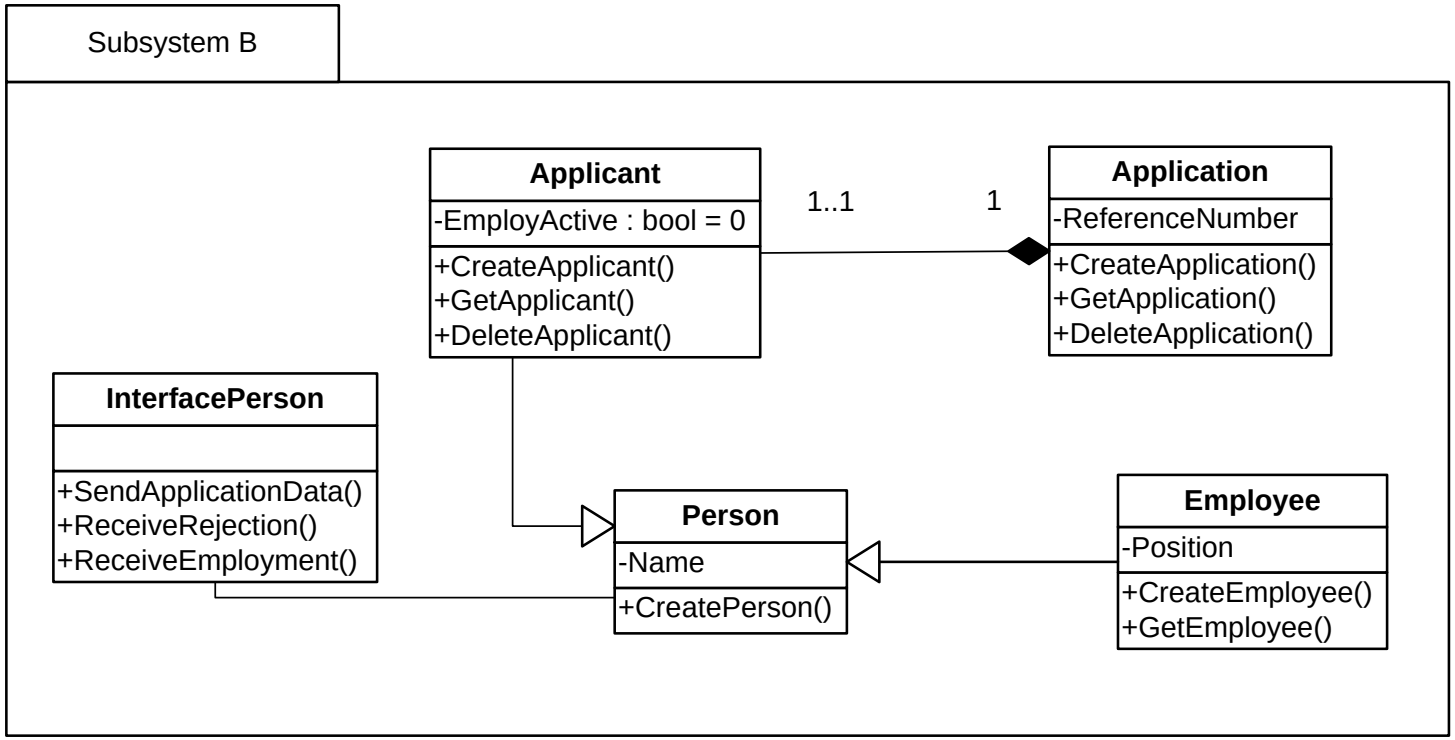
- ✓ A package provides a grouping mechanism for elements (classes, relations and lesser packages) with a common theme.
- ✓ A package partitions a model, making it easier to understand and manage .
- ✓ Class and association names are unique within each package.
- ✓ Consistent names are used across packages.
- ✓ If other packages refer to a class, they can use a class icon that contains only the class name.

Notation:

box with a tab

Grouping Classes into Packages

A package provides a grouping mechanism for elements (classes, relations and lesser packages) with a common theme.



Packages

Classes can be repeated in different packages, but normally each association belongs to a single package.

If classes are represented in different packages, they form a bridge between two or more packages.

Dependency between packages indicates that packages can access (or import) model elements from other packages.