## UNIT – 1 : Introduction-Modeling Concepts, class Modeling

**What is Object Orientation? What is OO development? OO themes; Evidence for usefulness of OO development; OO modeling history**

**Modeling as Design Technique: Modeling; abstraction; The three models.**

**Class Modeling: Object and class concepts; Link and associations concepts; Generalization and inheritance; A sample class model; Navigation of class models; Practical tips.**

# Object-Oriented Modelling and Design

- **Object-Oriented Modelling and Design** is a way of thinking about problems using models organized around real world concepts. The fundamental construct is the object, which combines both data and behaviour.
- An object has:
  - *state* - descriptive characteristics
  - *behaviors* - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviors associated with a bank account include the ability to make deposits and withdrawals

# Objects

- ***Objects have three responsibilities:***

***What they know about themselves – (e.g., Attributes)***

***What they do – (e.g., Operations)***

***What they know about other objects – (e.g., Relationships)***

Software objects model read-world objects or abstract concepts

- dog, bicycle, Bank account

Real-world objects have states and behaviors

- Dogs' states: name, color, breed, hungry
- Dogs' behaviors: barking fetching
- Bicycle ‘s State :
- Bicycle’ s behavior :

# What is Object-Orientation about?

- One of the key challenges faced by Computer Scientist is how to handle complexity.
- Two main concepts used to manage complexity are *Modularity* and *Abstractions*.
  - **Modularity** means breaking a large system up into smaller pieces until each peace becomes simple enough to be handled easily.
  - **Abstraction** focus on essential aspects of an application while ignoring details.
- Over the years, computer scientists have developed a number of approaches to achieve modularity and abstraction.
- The latest of these approaches is *Object-Orientation* or *OO* for short.
- The key concept in OO is of course *Object*,

- The process for OO development and graphical notation for representing OO concepts consists of building a model of an application and then adding details to it during design.

The methodology has the following stages:

- **System conception** : Software development begins with business analysis or users conceiving an application and formulating tentative requirements
- **Analysis :** The analyst must work with the requestor to understand the problem, because problem statements are rarely complete or correct.

The analysis model is a precise abstraction of what the desired system must do, not how it will be done.

It should not contain implementation decisions.

The analysis model has 2 parts:

- **Domain model -** a description of the real-world objects reflected within the system

Eg: Domain objects for a stock broker

- **Application – model -** a description of the parts of the application system itself that are visible to the user.

Eg:- Application might include stock, bond, trade and commission.

Application objects might control the execution of trades and present the results.

# Object-oriented methodology

- **System design:** The development teams devise a high – level strategy – the system architecture for solving the application problem.

  They also establish policies that will serve as a default for the subsequent, more detailed portions of design.

  The system designer must decide what performance characteristics to optimize, choose a strategy of attacking the problem and make tentative resource allocations.

- **Class design :** The class designer adds details to the analysis model in accordance with the system design strategy.

  The focus of class design is the data structures and algorithms needed to implement each class.

- **Implementation** : Implementers translate the classes and relationships developed during class design into particular programming language, database or hardware.

  During implementation, it is important to follow good software engineering practice so that traceability to the design is apparent and so that the system remains flexible and extensible.

# Modeling as a Design Technique

# What is Modeling

- Modeling consists of building an abstraction of reality.

- Abstractions are simplifications because:
  - They ignore irrelevant details and
  - They only represent the relevant details.

- What is *relevant* or *irrelevant* depends on the purpose of the model.

A model is a simplification of reality.

A model may provide
  - blueprints of a system
  - Organization of the system
  - Dynamic of the system

# MODEL

- **A model is an abstraction, before building any system a prototype may be developed**. The main purpose of model is for understanding of the system.

- Designer build different kinds of models for various purposes before constructing things.

- For example car , airplane, blueprints of machine parts, Plan for house construction etc., Models serve many purposes

# Importance of Modeling

- Models help us
  - to visualize a system as it is or as we want it to be.
  - to specify the structure or behavior of a system.
  - in providing a template that guides us in constructing a system.
  - in providing documenting the decisions we have made.

# Purpose of Modeling

Designers build many kinds of models for various purposes before constructing things.

Models serve several purposes –

- Testing a physical entity before building (simulation)
- Communication with customer
- Visualization…eg movies,television,Ad
- Reduction of complexity
- Better understanding of the problem

# Three models

We use three kinds of models to describe a system from different view points.

1. **Class Model describe the structure of objects in the system –their attributes & their relationships**.

   Class diagram express the class model.

2. **State model—The state model captures control ie aspect of system that describes the sequences of operations that occur.**

   Show how systems behave internally

   state diagram express the state model.

3. **Interaction Model—for the interaction among objects.**

   Show the behaviour of systems in terms of how objects interact with each other.

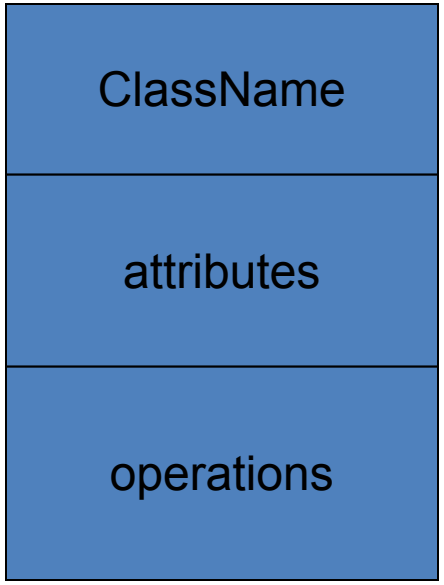
# Class modeling

1. Object and Class Concepts.
2. Link and Association Concepts
3. Generalization and Inheritance
4. A Sample Class Model
5. Navigation of Class Models

# Class diagrams

- Class diagrams provide a graphic notation for modeling classes and their relationships, thereby describing possible objects.

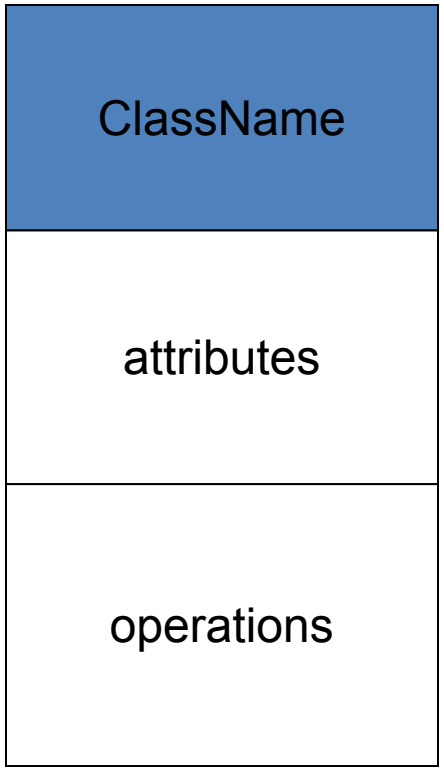
**Note:** An object diagram shows individual objects and their relationships.




A *class* describe a group of objects with the same properties (attributes) , operations, relationships, and semantics.

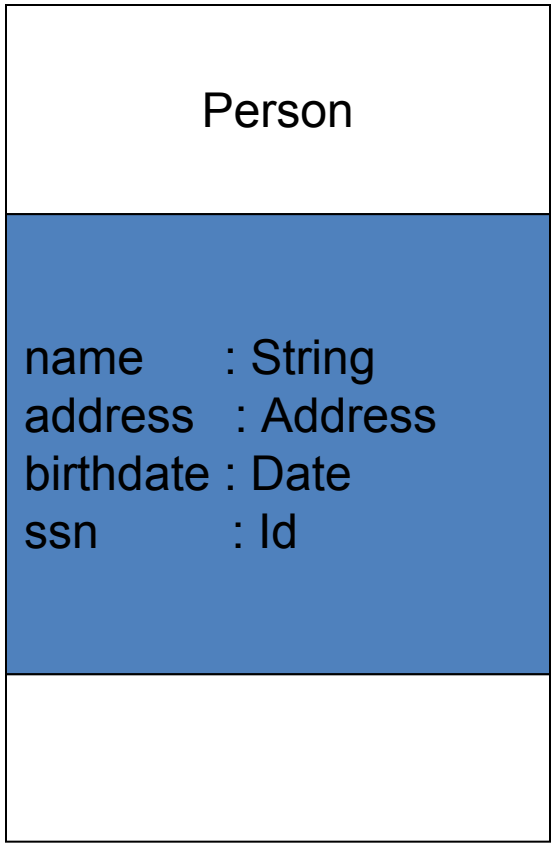
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

# Class Names



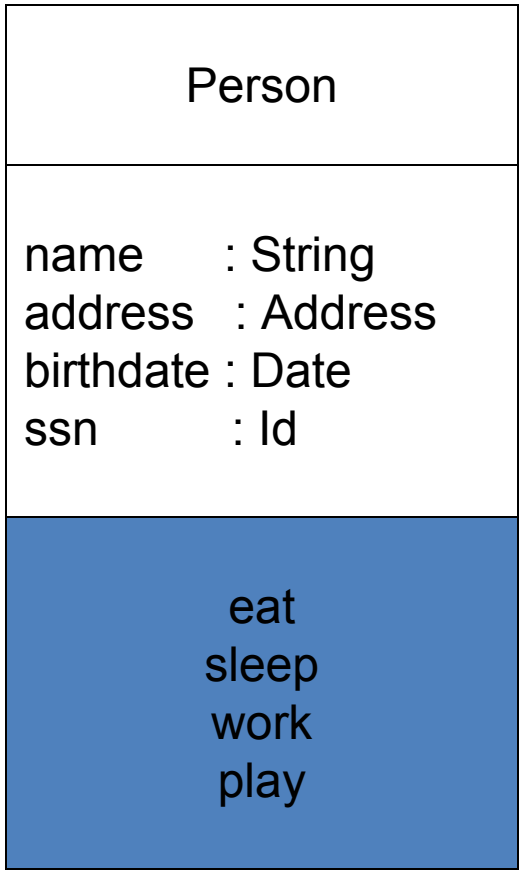

The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

# Class Attributes



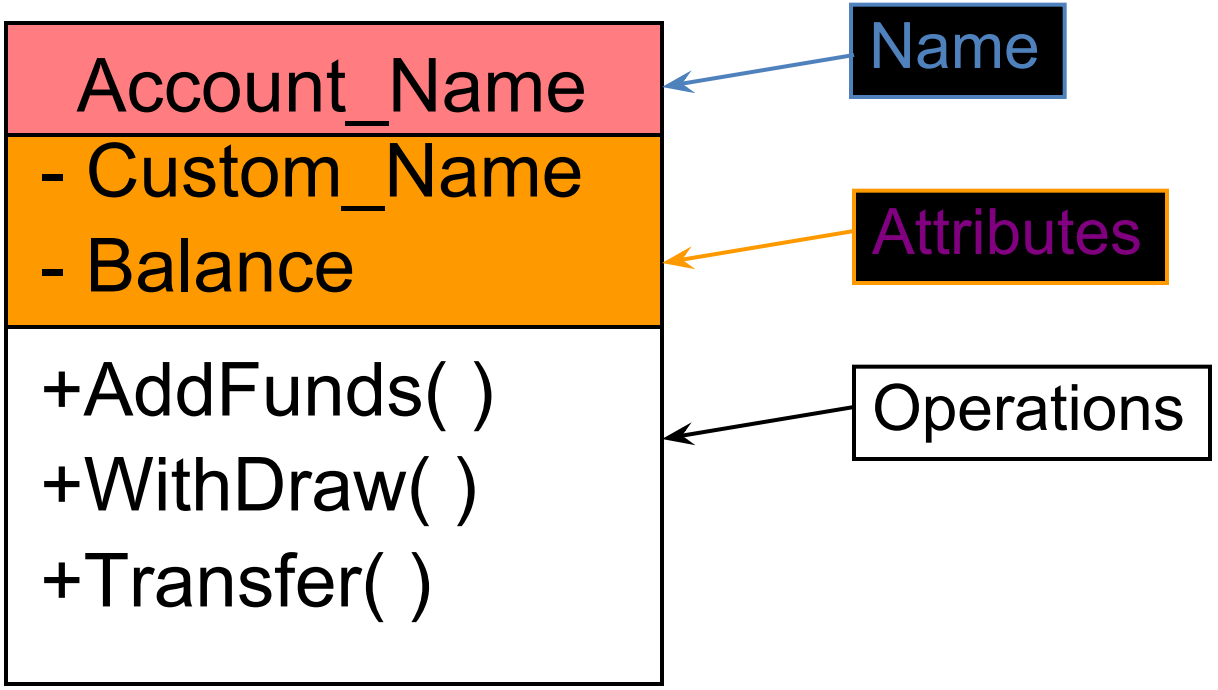

An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.

# Class Operations

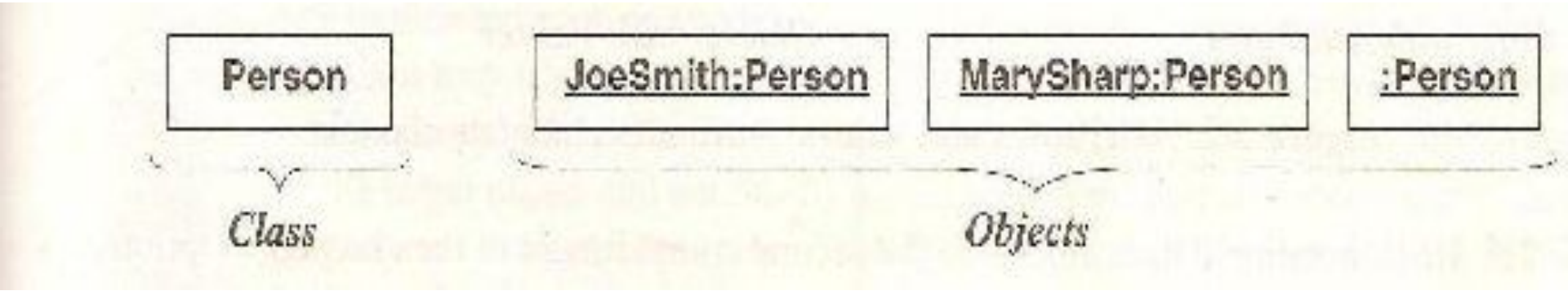



*Operations* describe the class behavior and appear in the third compartment.

# An example of Class



# Conventions used (UML) in Class Diagrams




## Conventions used (UML):

- UML symbol for both classes and objects is box.

- Objects are modeled using box with object name followed by colon followed by class name, Both the names are underlined.

- Use boldface to list class name, center the name in the box and capitalize the first letter. Use singular nouns for names of classes.

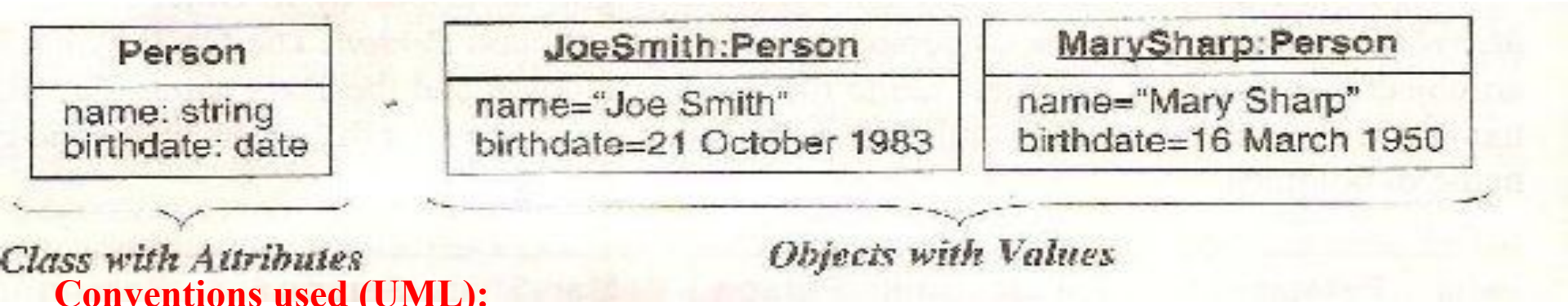
- To run together multiword names (such as JoeSmith), separate the words With intervening capital letter.

# Values and Attributes:

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

E.g. Attributes: Name, bdate, weight.

Values: JoeSmith, 21 October 1983, 64.




## Conventions used (UML):

List attributes in the 2nd compartment of the class box.

A colon precedes the type, an equal sign precedes default value.

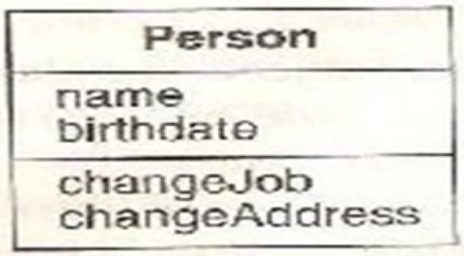
Show attribute name in regular face, left align the name in the box and use small case for the first letter.

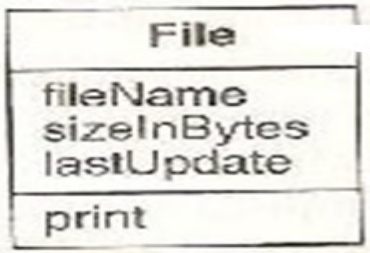
Similarly we may also include values of attribute in the 2nd compartment of object boxes with same conventions.

# Operations and Methods:

- **An operation** is a function or procedure that maybe applied to or by objects in a class.
- E.g. Hire, fire and pay dividend are operations on Class Company. Open, close, hide and redisplay are operations on class window.
- **A method** is the implementation of an operation for a class.
- E.g. In class file, print is an operation you could implement different methods to print files.

Note: Same operation may apply to many different classes. Such an operation is polymorphic.




UML conventions used –

List operations in 3rd compartment of class box.

List operation name in regular face, left align and use lower case for first letter.

# Links and Association

**Links and associations are the means for establishing relationships among objects and classes.**

- A link is a physical or conceptual connection among objects.

E.g. JoeSmith *WorksFor Simplex Company.*

- An association is a description of a group of links with common structure and common semantics.

E.g. a person *WorksFor a company.*

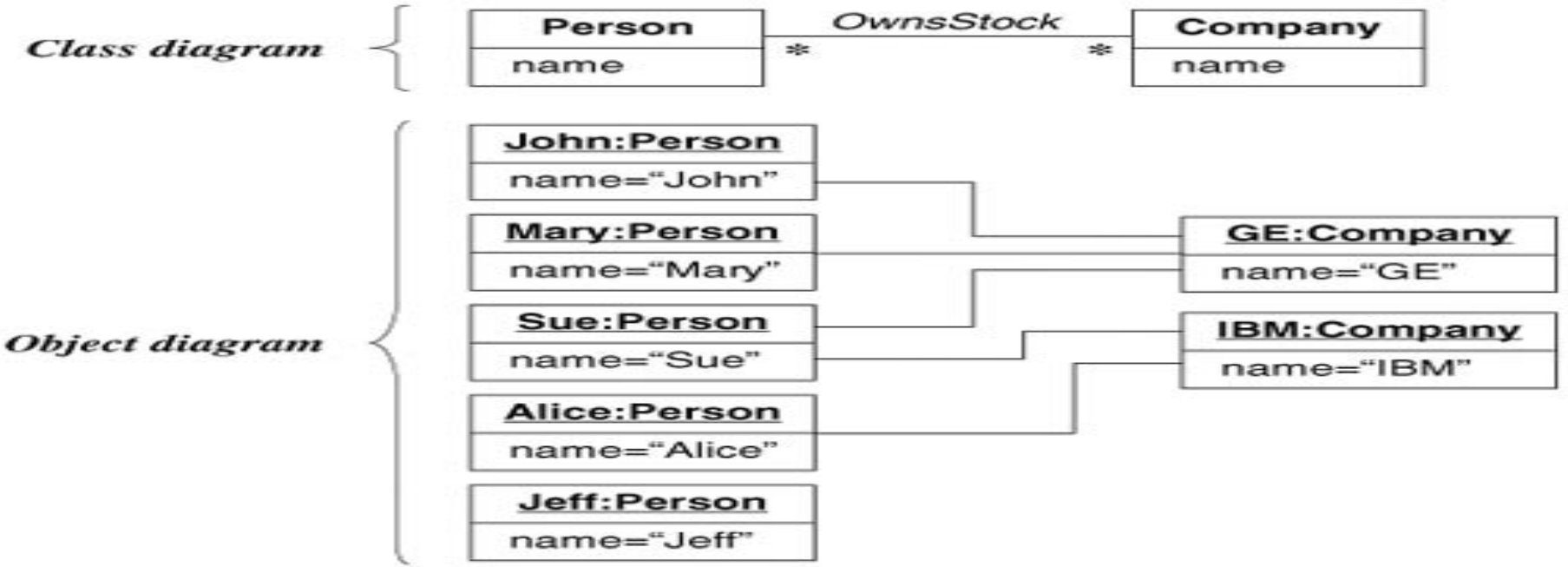



**Figure 3.7 Many-to-many association.** An association describes a set of potential links in the same way that a class describes a set of potential objects.



**Conventions used (UML):**

Link is a line between objects

Association connects related classes and is also denoted by a line.

Show link and association names in italics.

# Associations: Multiplicity

- Multiplicity defines the number of objects associated with an instance of the association.
- UML diagrams explicitly list multiplicity at the end of association lines.
- Intervals are used to express multiplicity

| Indicator | Meaning |
|---|---|
| 0..1 | Zero or one |
| 1 | One only |
| 0..* | Zero or more |
| 1..* | One or more |
| n | Only *n* (where *n* > 1) |
| 0..n | Zero to *n* (where *n* > 1) |
| 1..n | One to *n* (where *n* > 1) |

| | | |
|---|---|---|
| min..max notation (related to at least min objects and at most max objects) | 0..* | related to zero or more objects |
| | 0..1 | related to no object or at most one object |
| | 1..* | related to at least one object |
| | 1..1 | related to exactly one object. |
| | 3..5 | related to at least three objects and at most five objects |
| short hand notation | 1 | same as 1..1 |
| | * | same as 0..* |

Figure 3.8 shows a one-to-one association and some corresponding links.

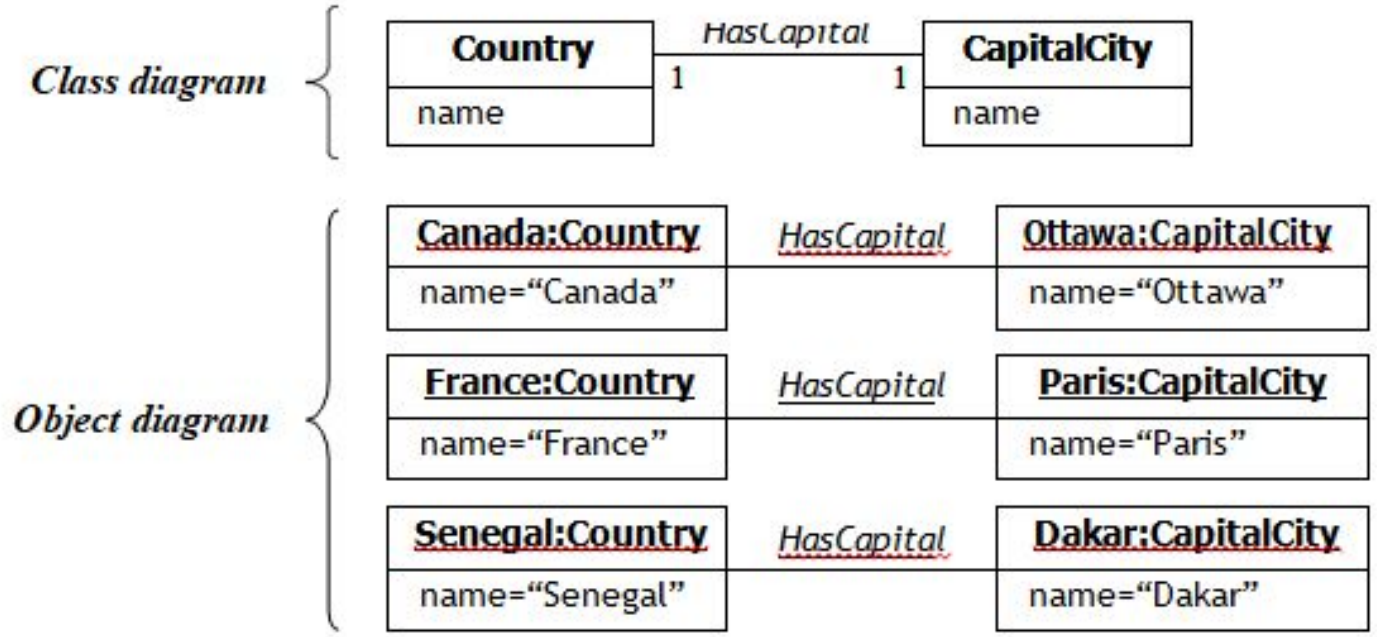


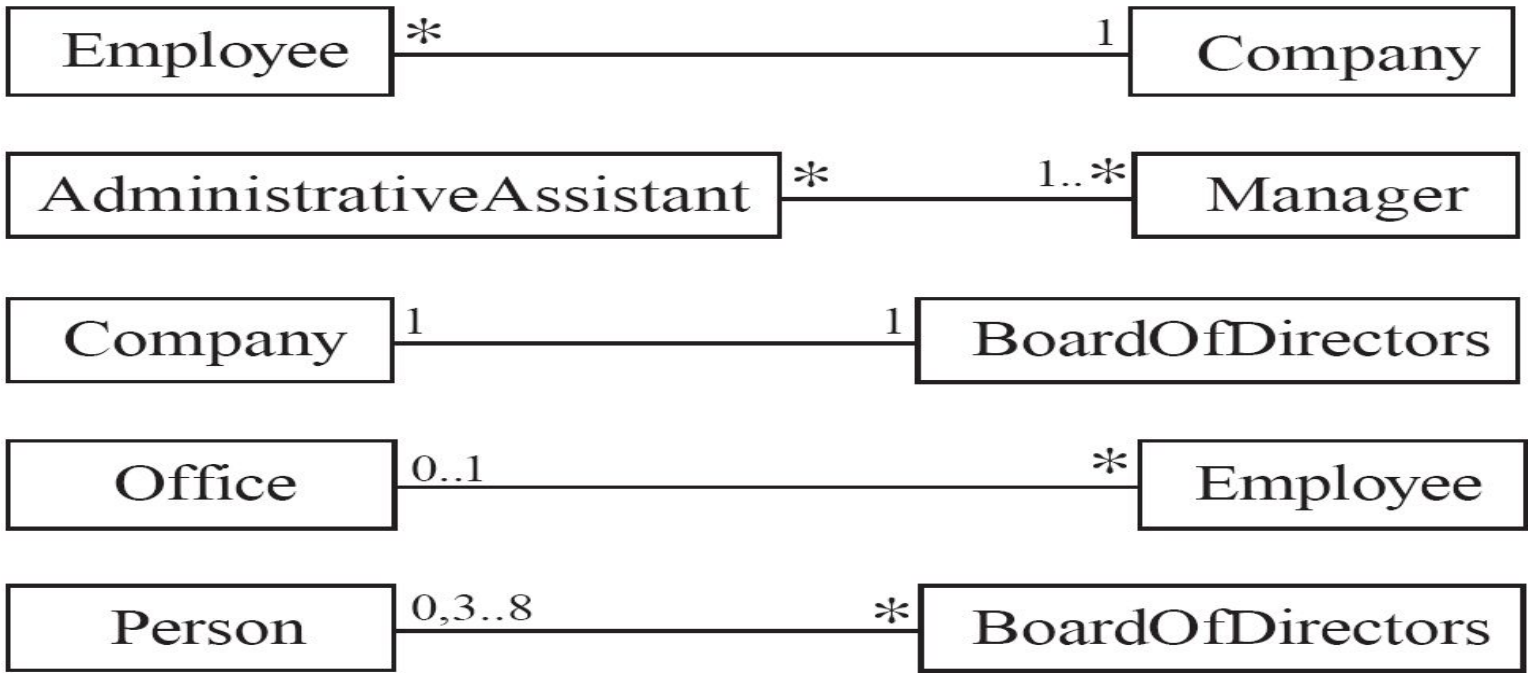

**Figure 3.8** **One-to-one association**. Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class.

A multiplicity of "many" specifies that an object may be associated with multiple objects.

# Associations and Multiplicity

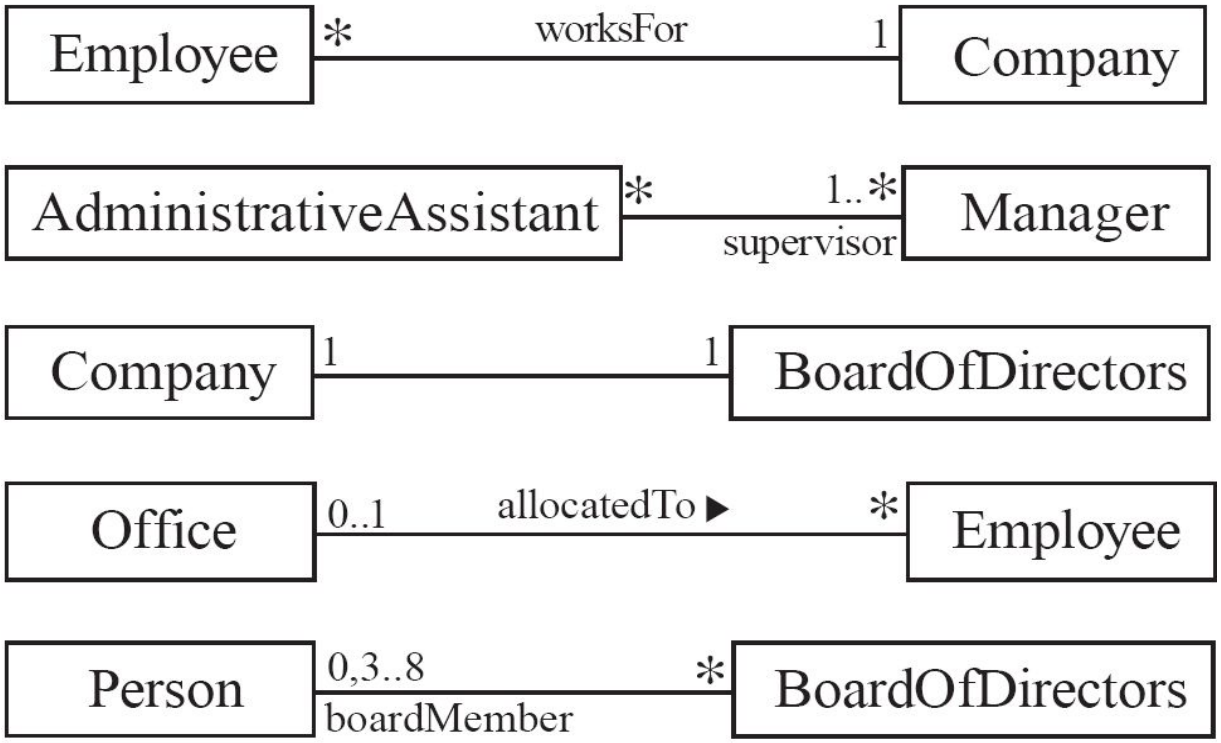
An *association* is used to show how two classes are related to each other

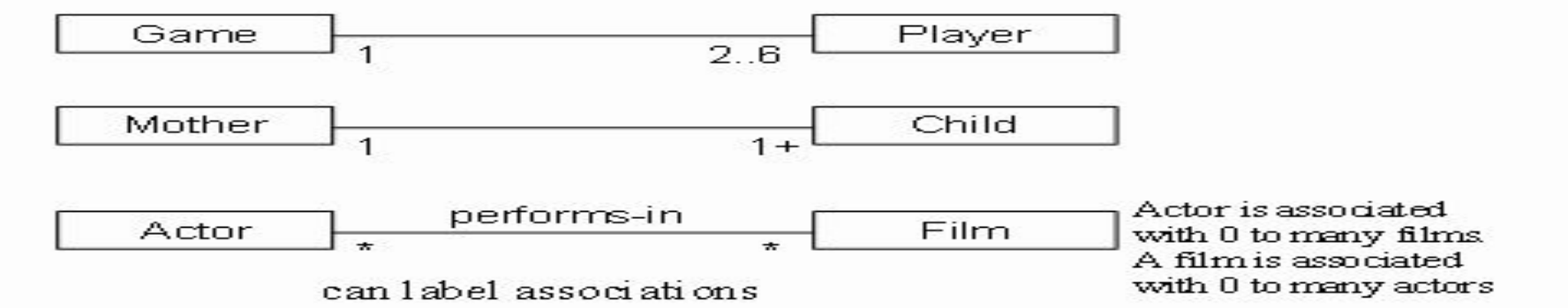
- Symbols indicating *multiplicity* are shown at each end of the association



# Labelling associations

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



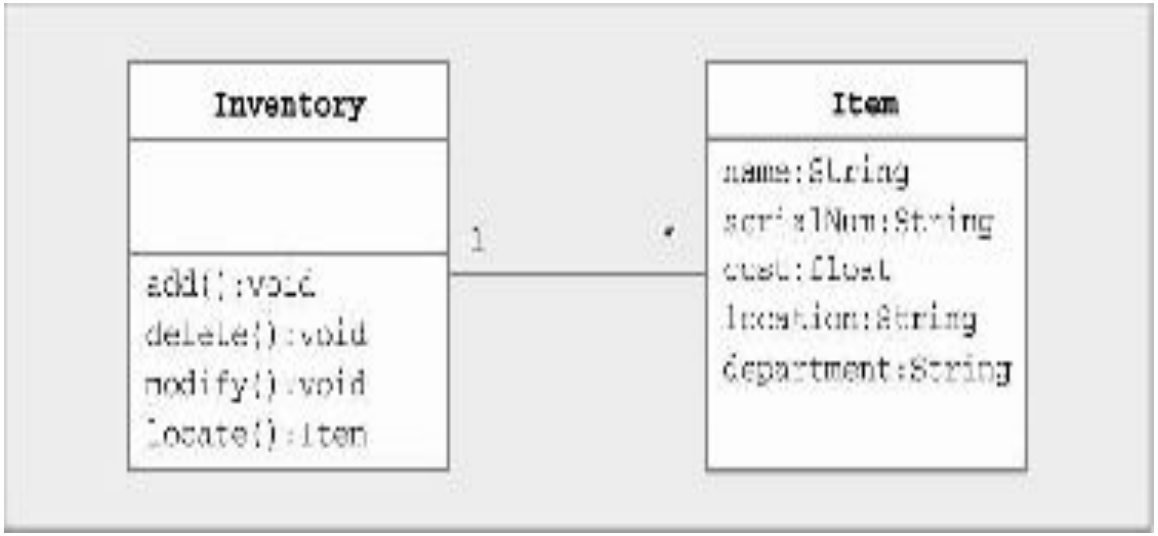



## Association – Multiplicity

- A teacher teaches 1 to 3 courses (subjects)
- Each course is taught by only one teacher.
- A student can take between 1 to 5 courses.
- A course can have 10 to 300 students.

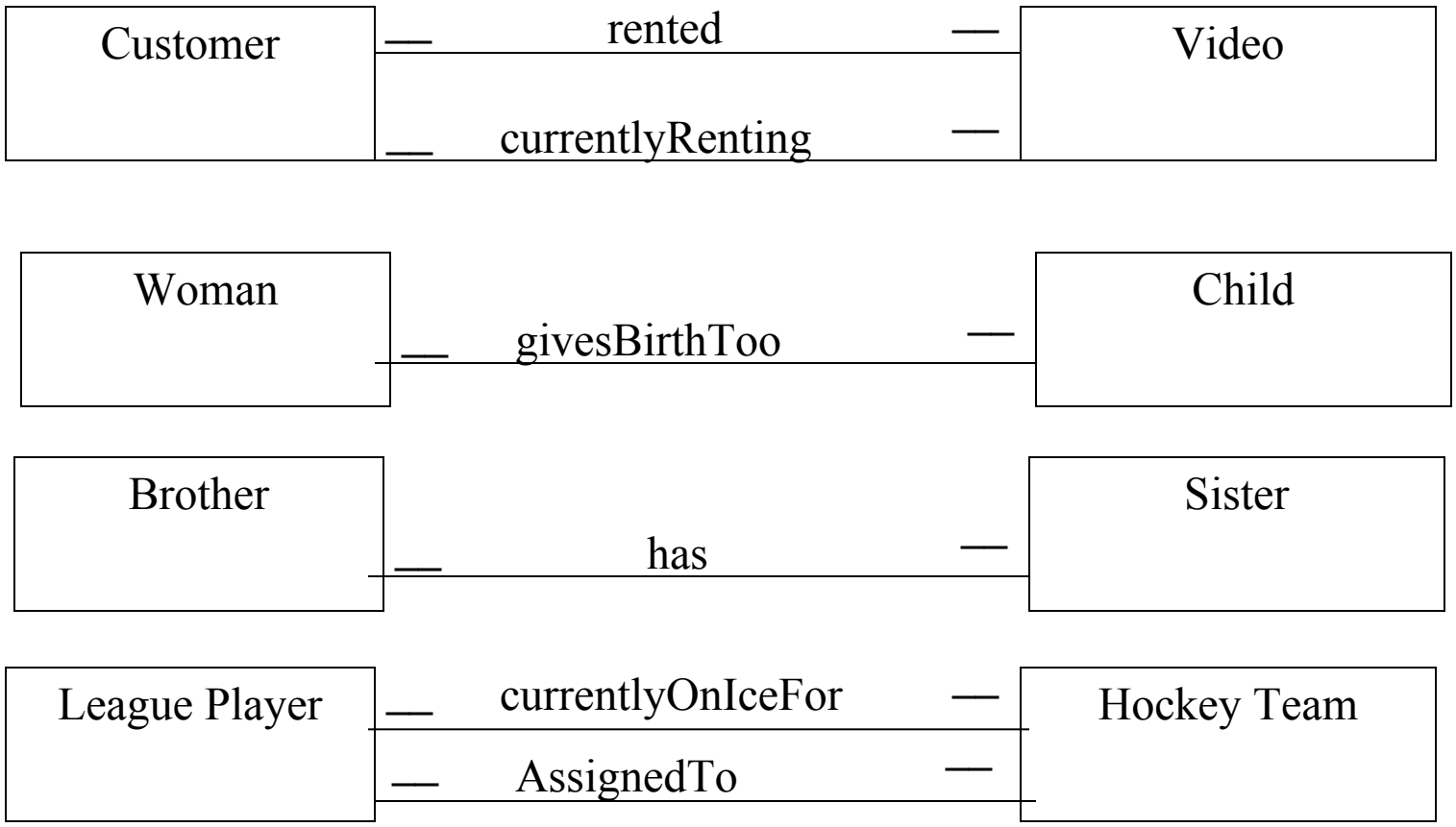
Many-to-one
– Bank has many ATMs, ATM knows only 1 bank


One-to-many
– Inventory has many items, items know 1 inventory



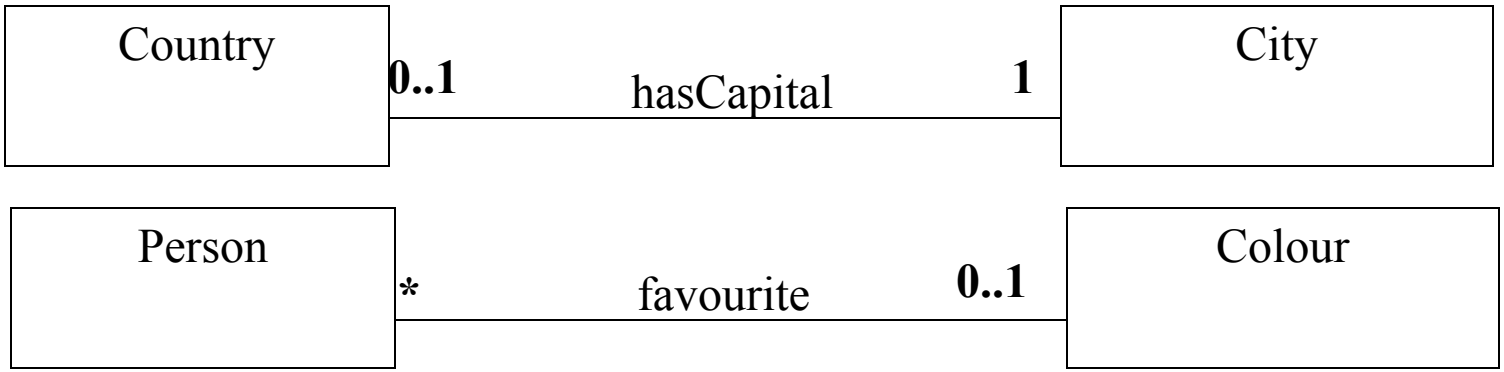
# Question

- Label the multiplicities for the following examples:



# Question

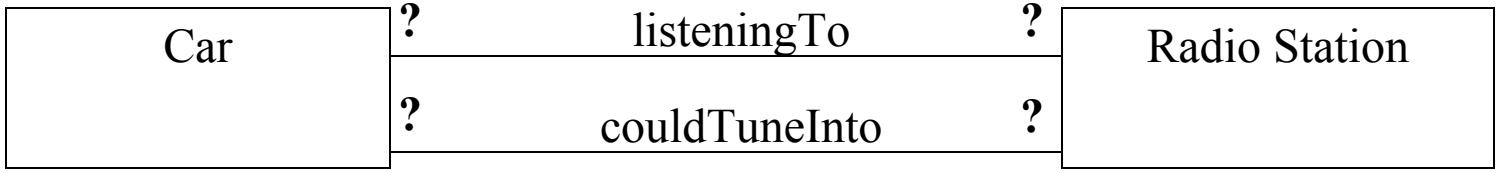
- In words, what do these diagrams mean?

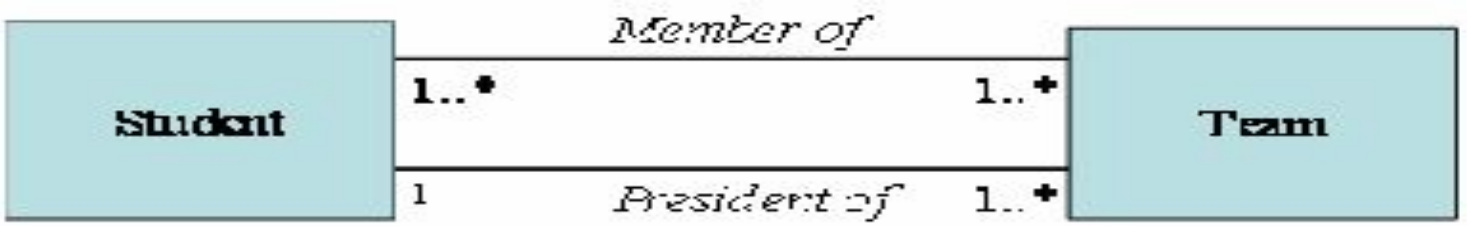



- A Country has *one and only one city as its capital*
- A City _____
- A Colour _____
- A Person _____

# Another Question:

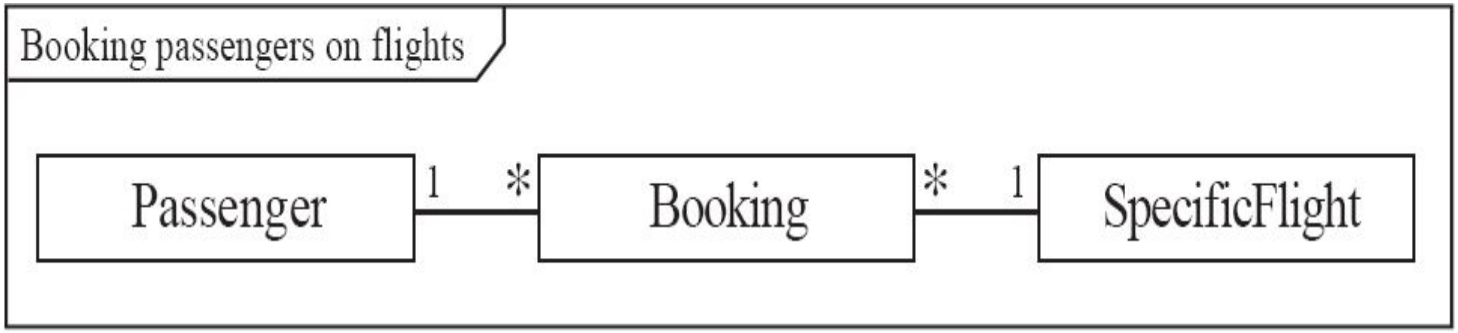
- Correctly label this diagrams multiplicity:



# 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



# Question

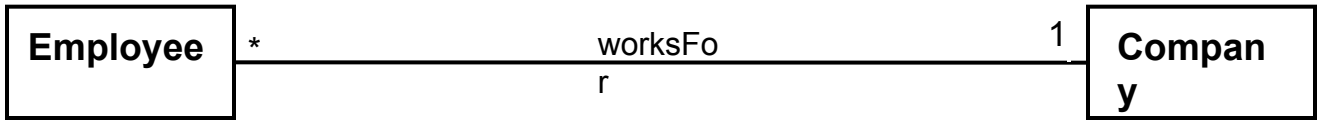
- Create two or three classes linked by associations to represent the following situations:
- *A landlord renting apartments to tenant*
- *An author writing books distributed by publishers*

– Label the multiplicities (justify why you picked them)

– Give each class you choose at least 1 attribute

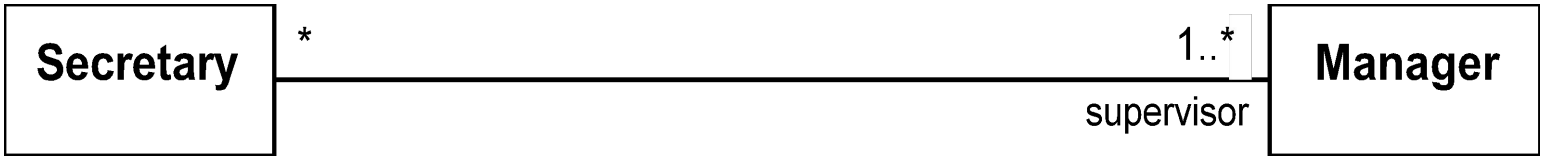
# Analyzing and validating associations

- **Many-to-one**
  - A company has many employees,
  - An employee can only work for one company.
  - 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



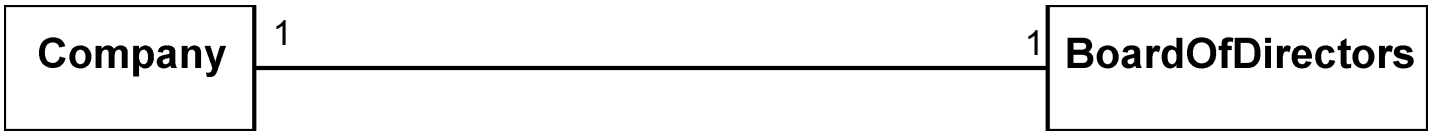
# Analyzing and validating associations

- **Many-to-many**
  - A secretary can work for many managers
  - A manager can have many secretaries
  - Managers can have a group of secretaries
  - Some managers might have zero secretaries.



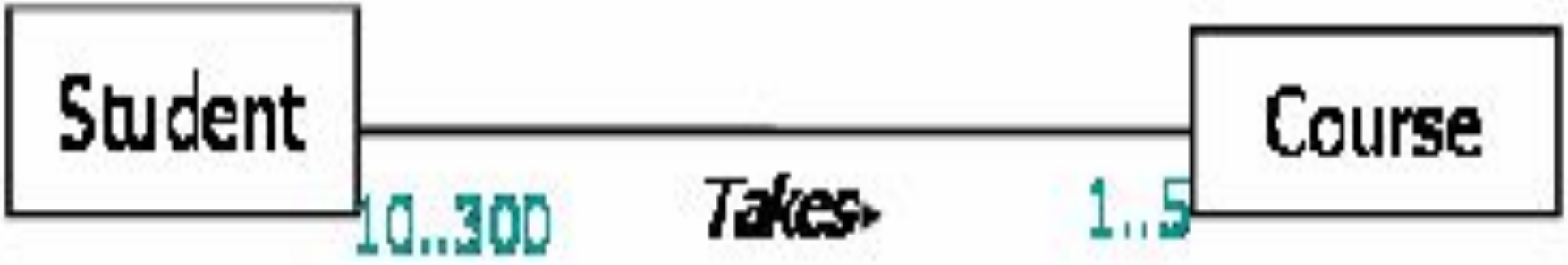
# Analyzing and validating associations

- **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



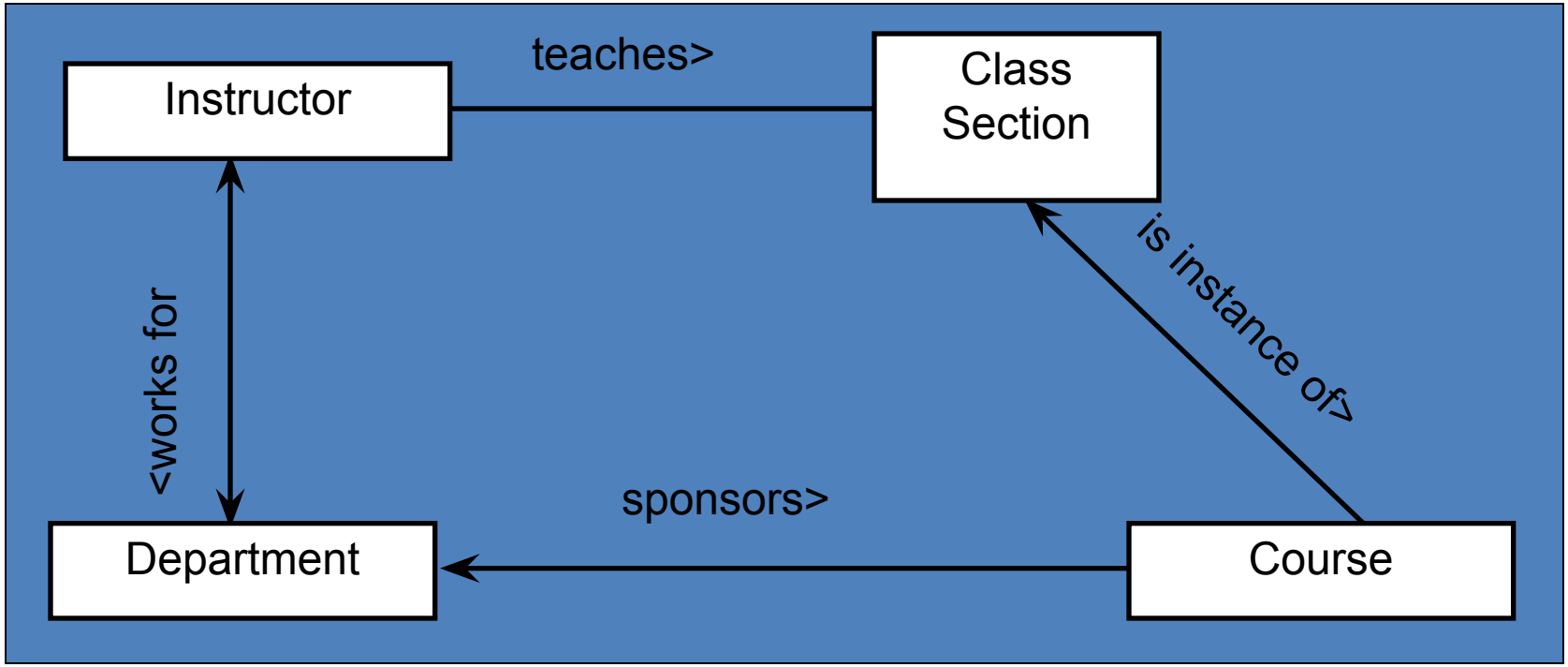
# Association - Multiplicity

- A Student can take up to five Courses.
- Student has to be enrolled in at least one course.
- Up to 300 students can enroll in a course.
- A class should have at least 10 students.



# Navigation

- The navigation of associations can be
  - uni-directional
  - bi-directional
  - unspecified
- Navigation is specified by the arrow, not the label



# Association Ends

- Associations have ends. They are called ‘Association Ends’.
- They may have names - Rolenames (which often appear in problem descriptions).

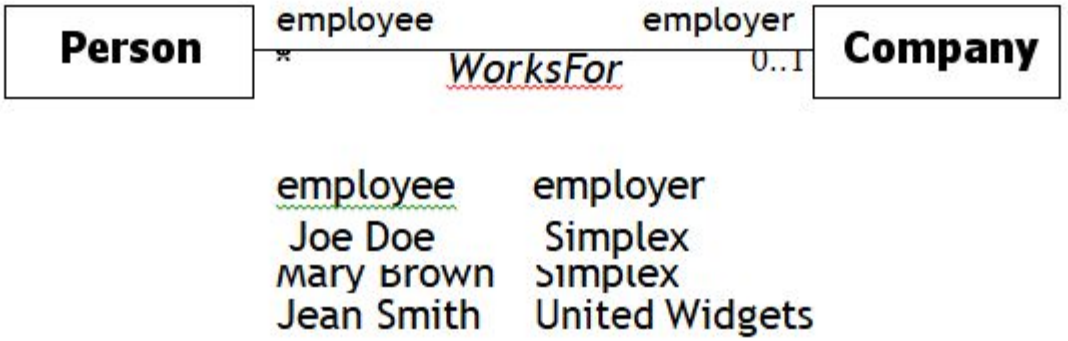



**Figure 3.12 Association end names.** Each end of an association can have a name.

A person is an employee with respect to company.

A company is an employer with respect to a person

# Associations (cont.)

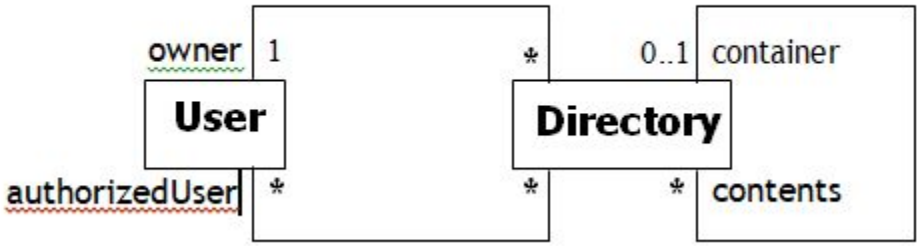
- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.
- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations

. **Note 1: Association end names are optional.**

. **Note 2: Association end names are necessary for associations between two objects**

of the same class. They can also distinguish multiple associations between a pair of classes.

E.g. each directory has exactly one user who is an owner and many users who are authorized to use the directory.
*container* and *contents* distinguish the two usages of *Di- rectory* in the self-association. A directory may contain many lesser directories and may op- tionally be contained itself.
.



# Association Relationships (Cont'd)

We can indicate the *multiplicity* of an association by adding *multiplicity relationships* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:

# Association Relationships (Cont'd)

The example indicates that every *Instructor* has one or more *Students*:

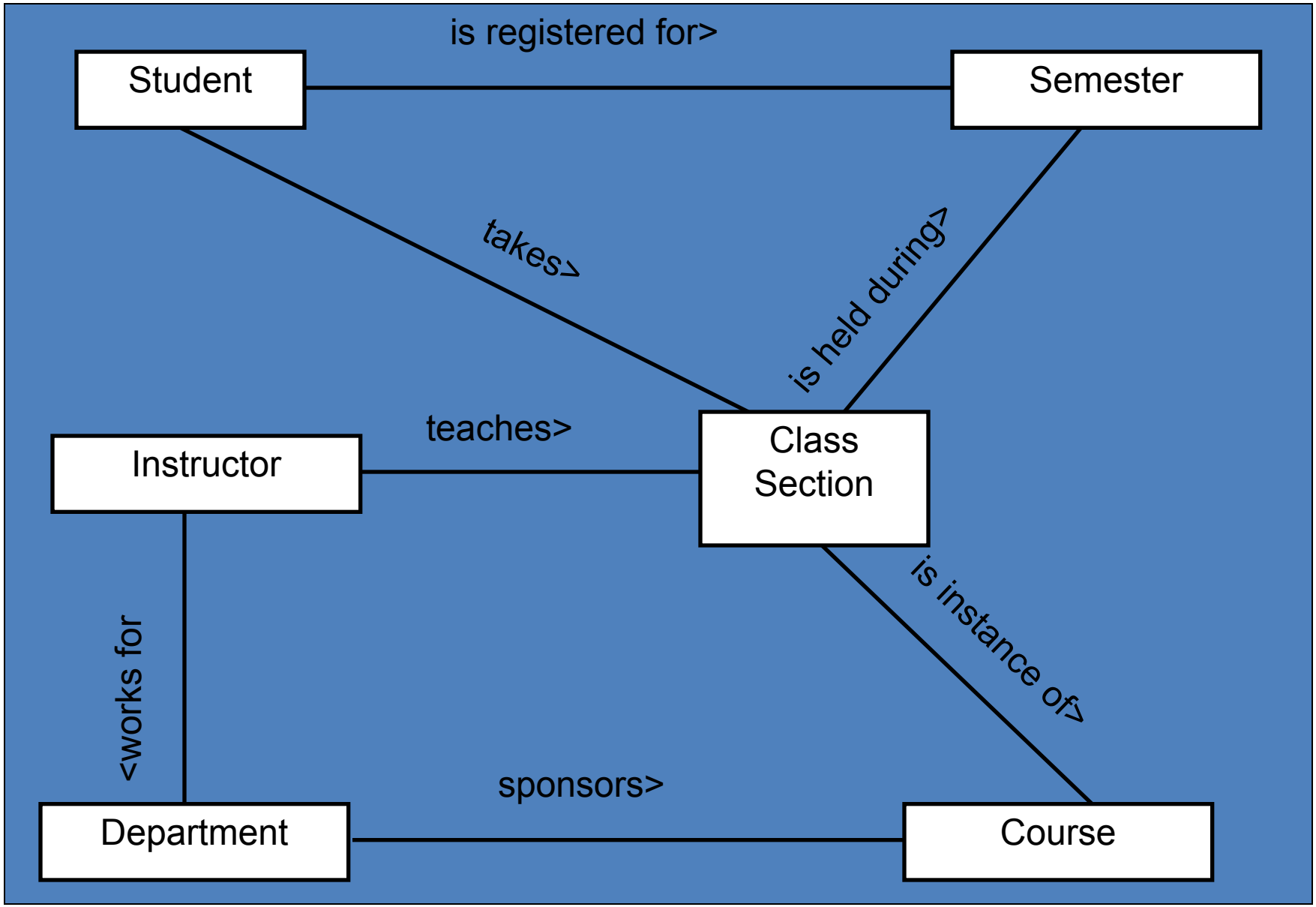
# Association Relationships (Cont'd)

We can also indicate the behavior of a class in an association (*i.e.*, the *role* of an class) using *rolenames*.

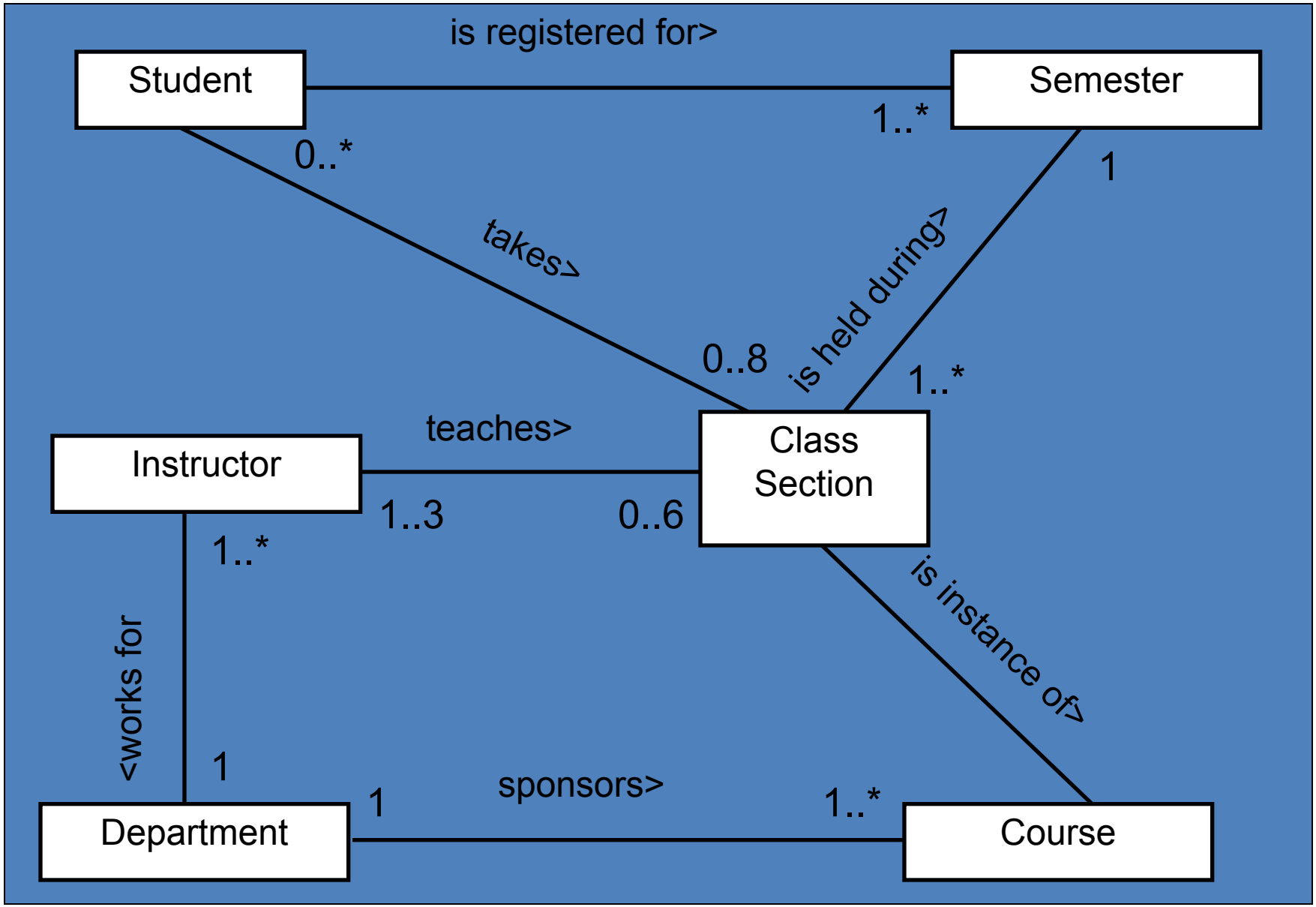
# Association - Multiplicity

- A cricket team has 11 players. One of them is the captain.
- A player can play only for one Team.
- The captain leads the team members.

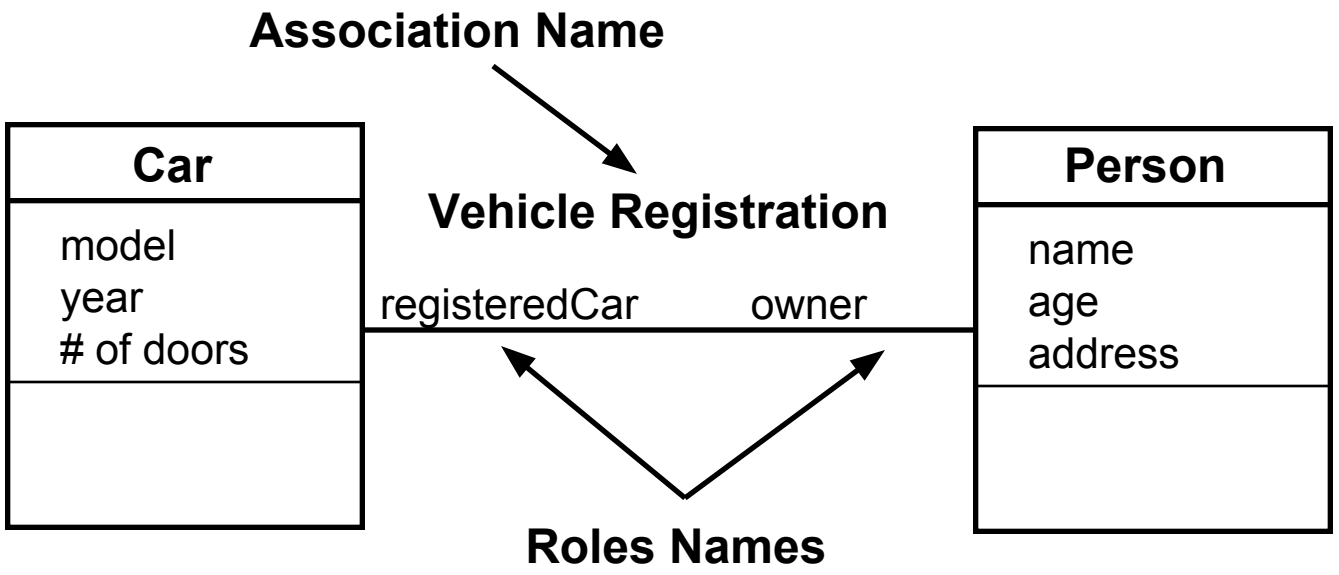
# Associations



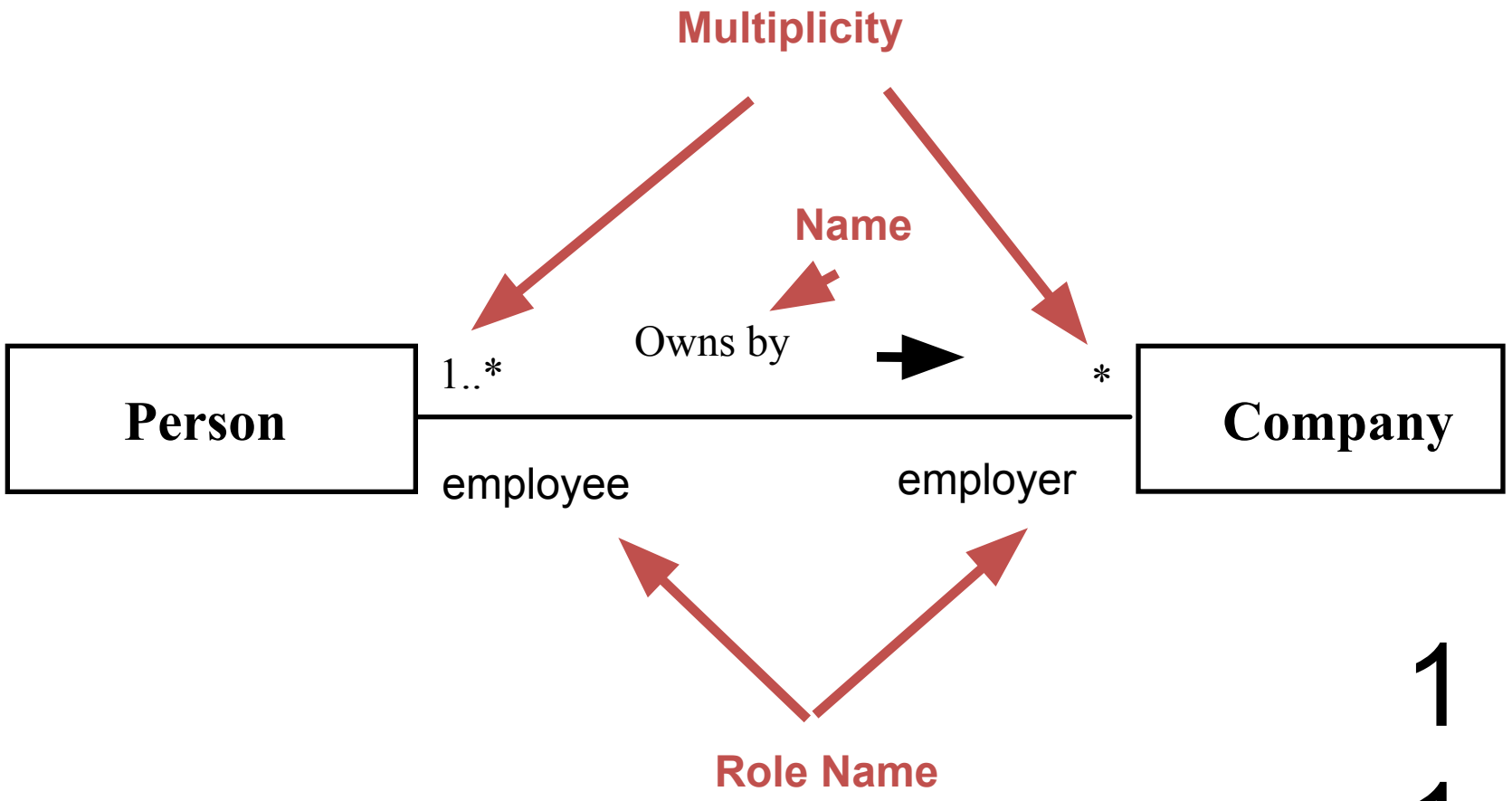
# Multiplicity Constraints



# Associations (3)



# Associations (4)




1

1

# Associations (cont.)

- Multiplicity
  - the number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

**Multiplicity Indicators**

| | |
|---|---|
| Exactly one | 1 |
| Zero or more (unlimited) | * (0..*) |
| One or more | 1..* |
| Zero or one (optional association) | 0..1 |
| Specified range | 2..4 |
| Multiple, disjoint ranges | 2, 4..6, 8 |

# Qualified Association

- A qualified association is an association in which an attribute called Qualifier the objects for a ‘many’ association’ end.
- A qualifier selects among the target objects, reducing the effective multiplicity from ‘many’ to ‘one’.
- Both below models are acceptable but the qualified model adds information.

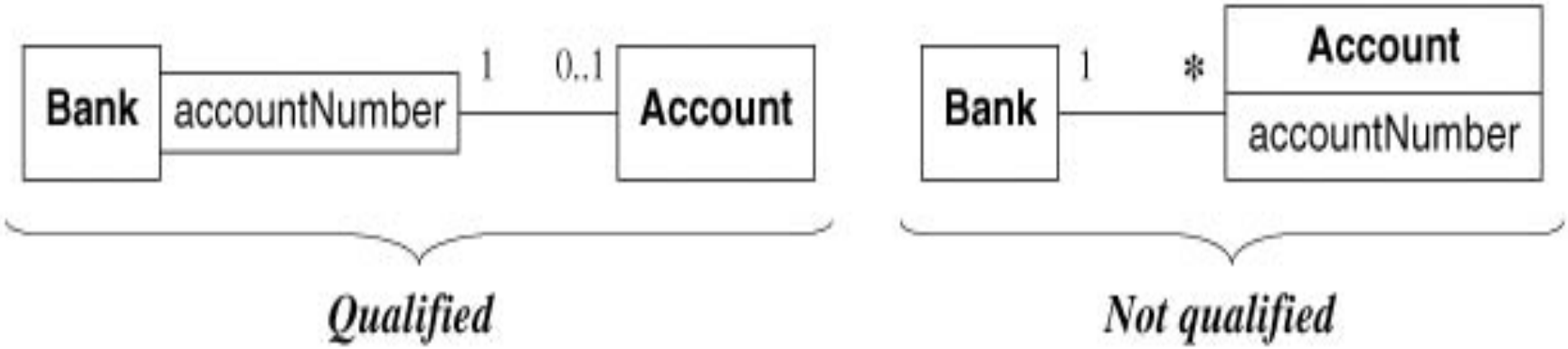
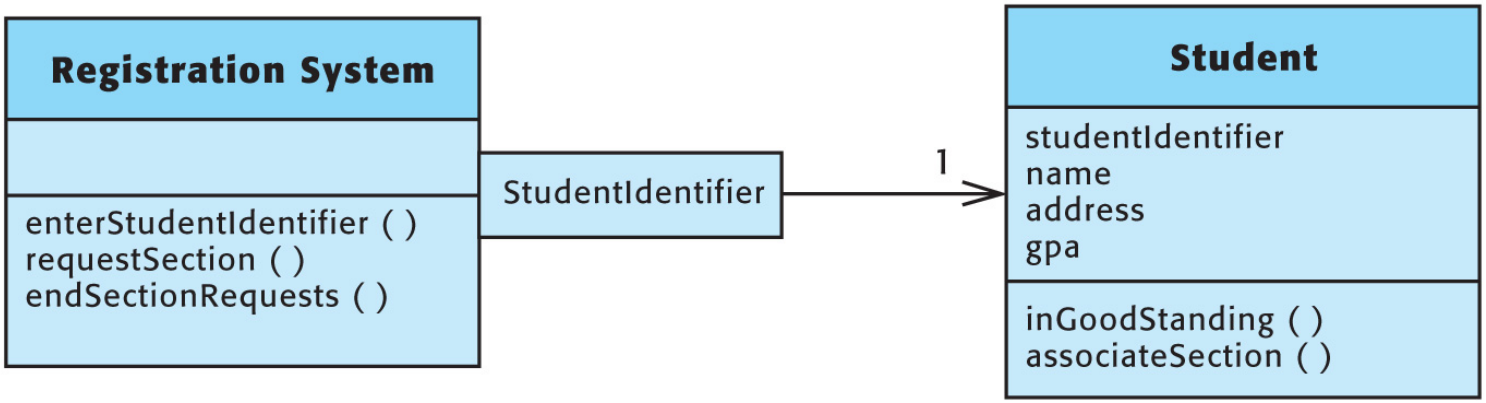



**Figure 3.22 Qualified association.** Qualification increases the precision of a model.

# Qualified Associations

(continued)

**FIGURE 9.2**



# Qualified Association

- Example:

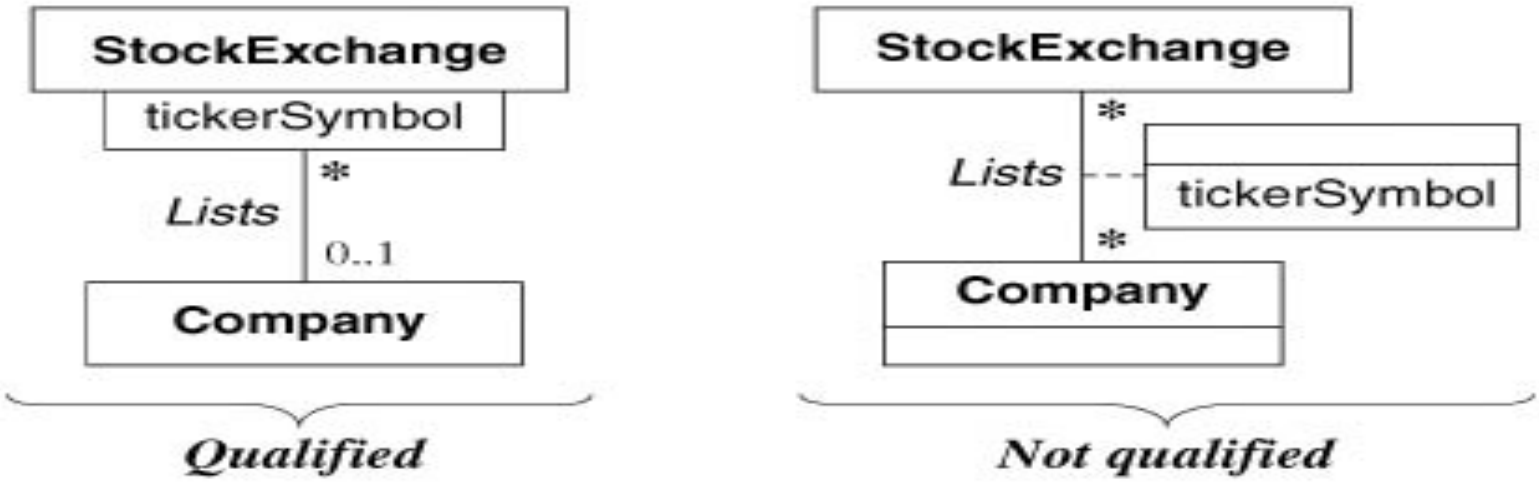
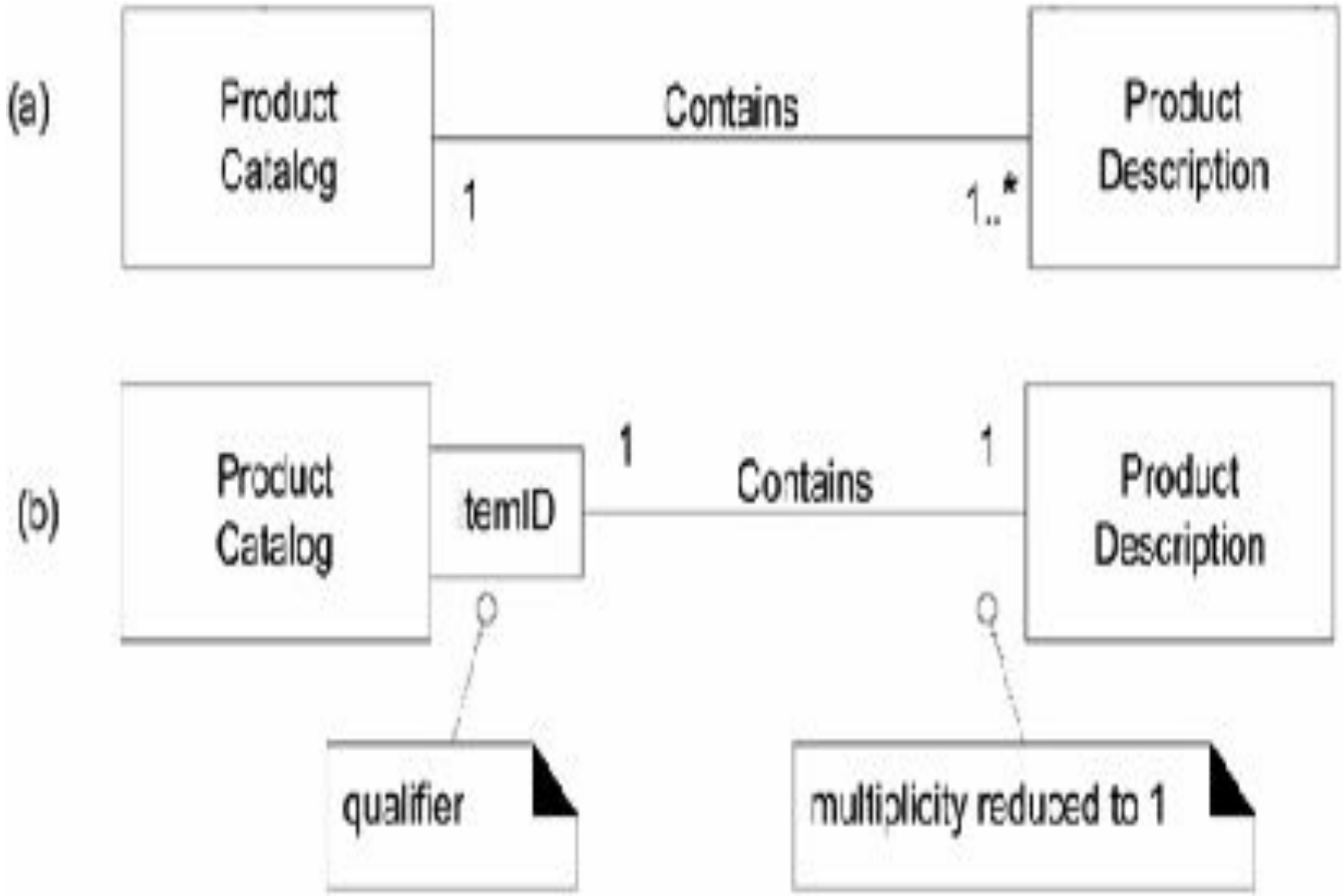
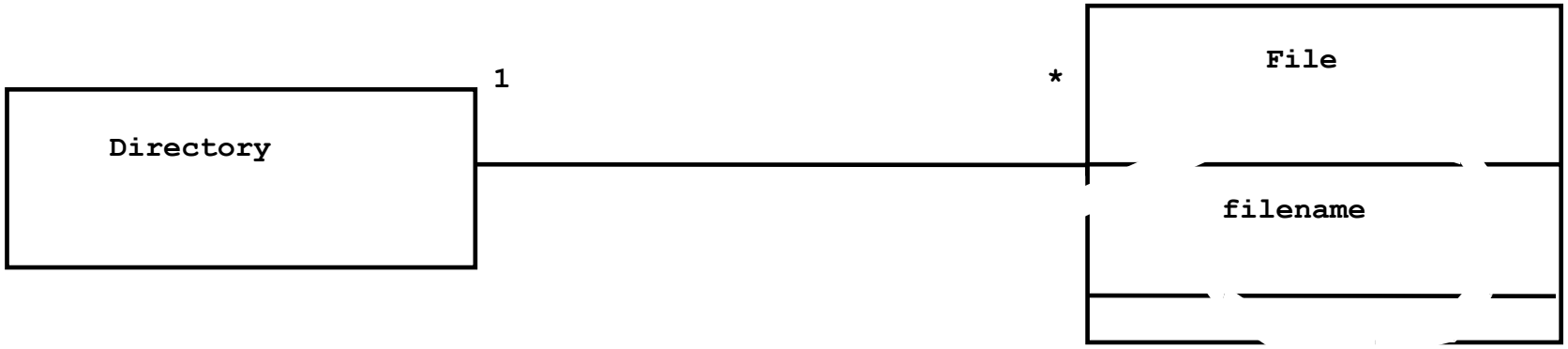


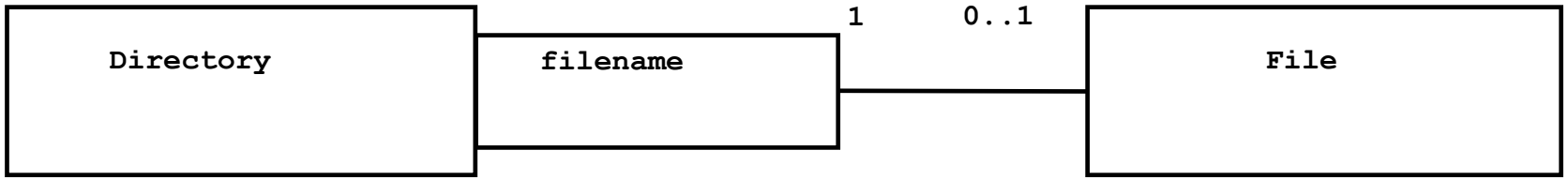

**Figure 3.23 Qualified association.** Qualification also facilitates traversal of class models.


(a)
Product Catalog
Contains
1
1..*
Product Description
(b)
Product Catalog
itemID
1
Contains
1
Product Description
qualifier
multiplicity reduced to 1

**W ithout qualification**




**W ith qualification**




Example of how a qualified association reduces multiplicity (UML class diagram). Adding a qualifier clarifies the class diagram and increases the conveyed information. In this case, the model including the qualification denotes that the name of a file is unique within a directory.

## *Qualified Associations*

You Tube link for Qualified Association

https://www.youtube.com/watch?v=NmY_jTTY1Nc

# Association: ordering

- On a ‘many” association end, sometimes, it is required that objects have an explicit order. In this case the ordering is an inherent part of the association

- The ordering is an inherent part of the association

# Association: ordering

- writing —{ordered}‖ next to appropriate association end.
- **Example:** A workstation screen contains a number of overlapping windows.
- Each window on a screen occurs at most once.
- The windows have an explicit order so only the topmost window is visible.
- The ordering is an inherent part of the association. You can indicate an ordered set of objects by writing “{ordered}” next to the appropriate association end.

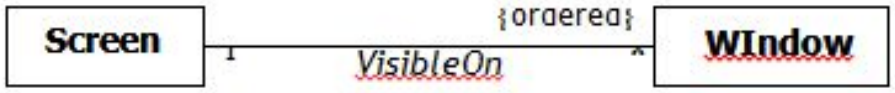



**Figure 3.15 Ordering the objects for an association end.** Ordering sometimes occurs for “many” multiplicity.

# Association: bag, sequence

- A **bag** is a collection of elements with duplicates allowed.

- A **sequence** is an ordered collection of elements with duplicates allowed

# Association: sequence

- Example : An itinerary is a sequence of airports and the same airport can be visited more than once.
- Sequence is ordered bag allow duplicates,
- {ordered} and {sequence} only difference is sequence allows duplicates as shown in figure.
- If you spec- ify *{bag}* or *{sequence}*, then there can be multiple links for a pair of objects.
- Note that the *{ordered}* and the *{sequence}* annotations are the same, except that the first disallows duplicates and the other allows them. A sequence association is an ordered bag, *while an ordered association is an ordered set.*

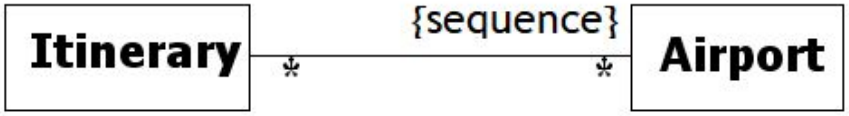



**Figure 3.16 An example of a sequence**. An itinerary may visit multiple airports, so you should use *{sequence}* and not *{ordered}*.

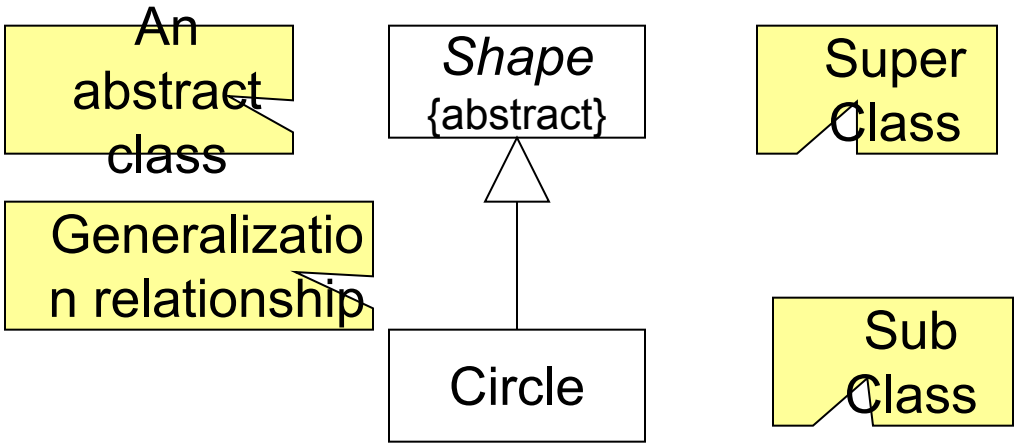
# Generalization

- Deriving a class out of a parent class having some inherited property(from the parent class) and some new property of the derived class.
- The term generalization is for the inheritance in the bottom to the up direction i.e. from derived class to the parent class.
- Generalization is the relationship between a class (**superclass**) and one or more **variations** of the class (**subclasses**).
- A superclass holds **common** attributes, attributes and associations.
- The subclasses **adds specific** attributes, operations, and associations. They i**nherit** the features of their superclass.
- **Generalization** is called a "**IS A"** relationship

# Generalization

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class).
  - "is kind of" relationship.

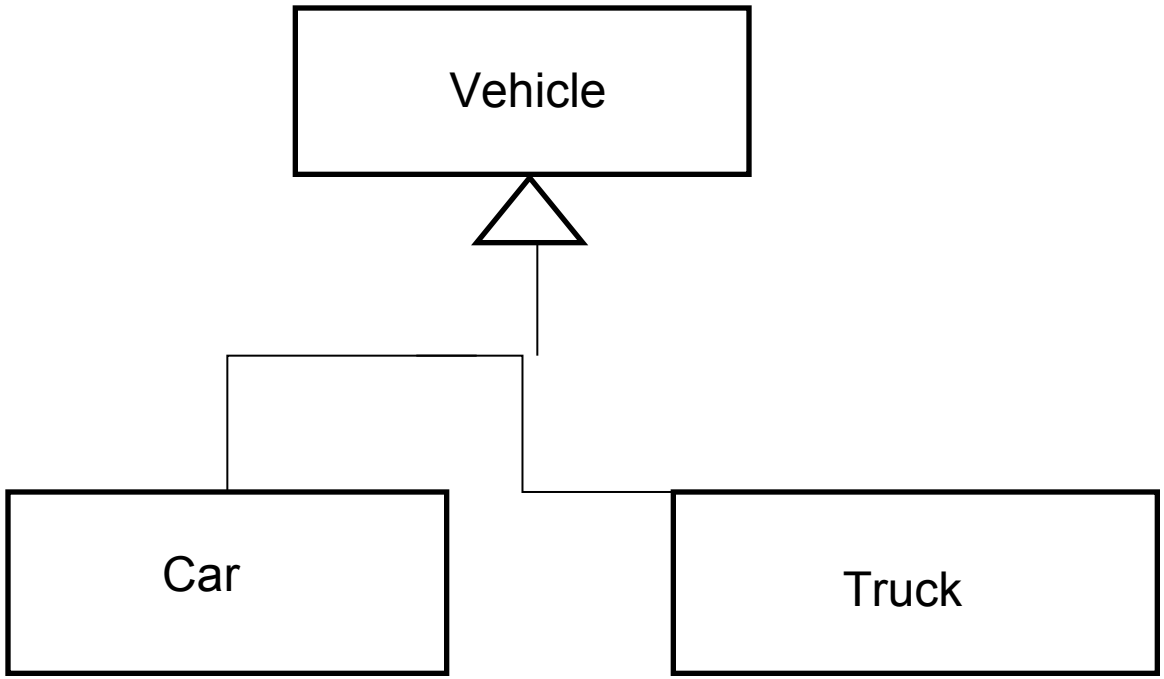
**{abstract}** is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized



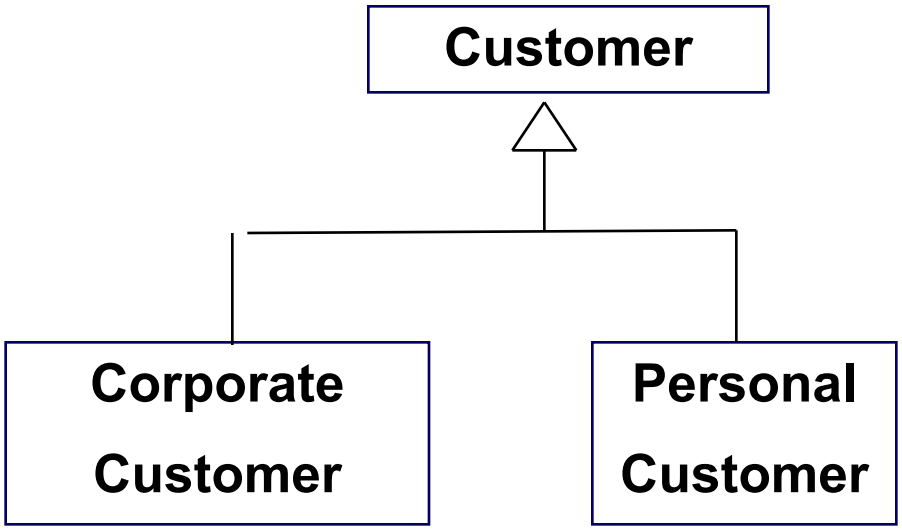
# Generalization

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

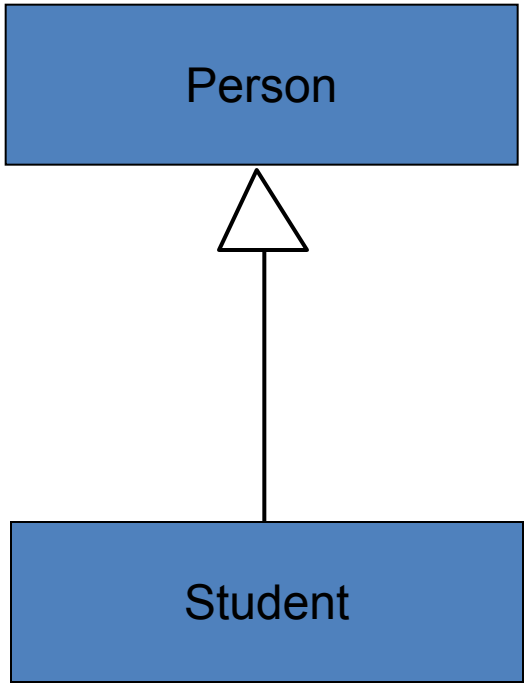
- It is represented by a solid line with a large arrow head pointing towards the parent class.
- Example:



# Generalization

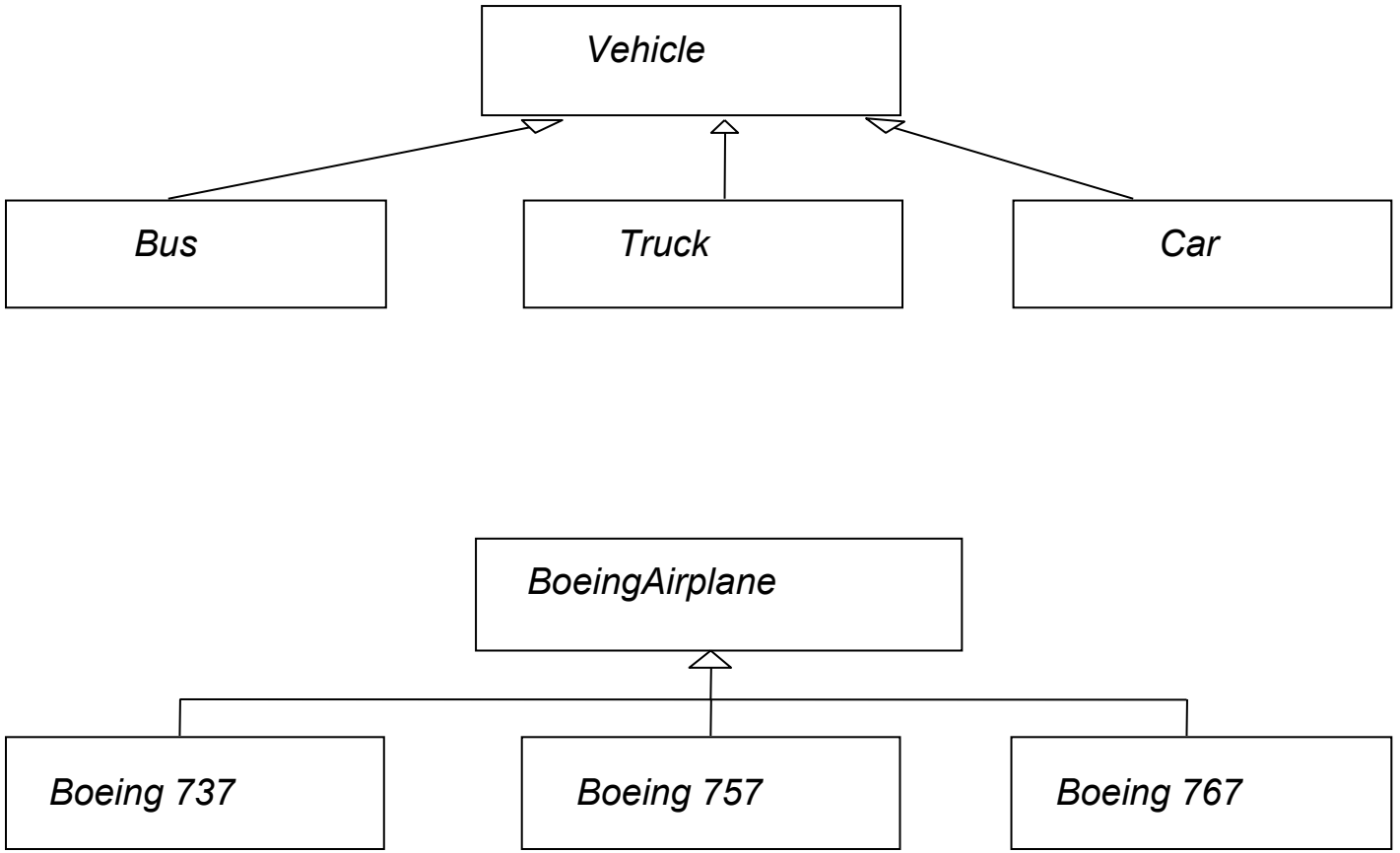


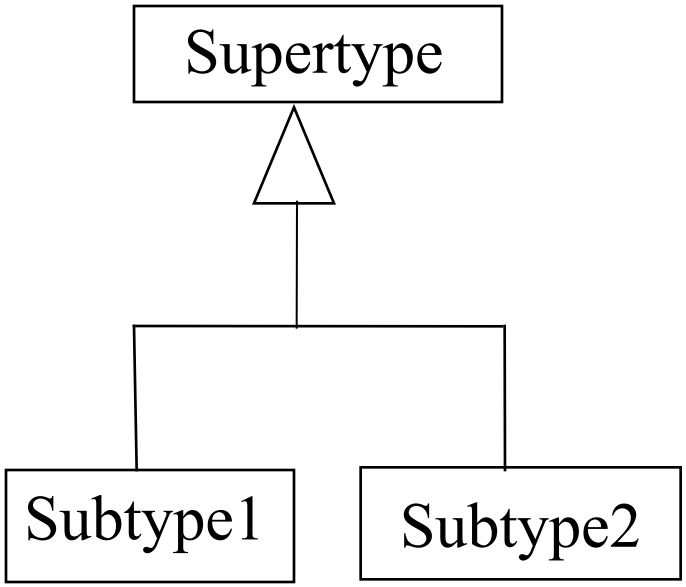
# Generalization Relationships



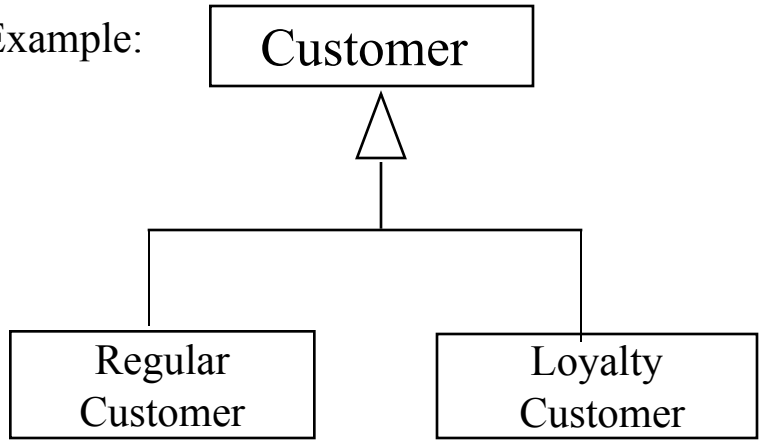

A *generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

# Generalization Relationships




Supertype
Subtype1
Subtype2

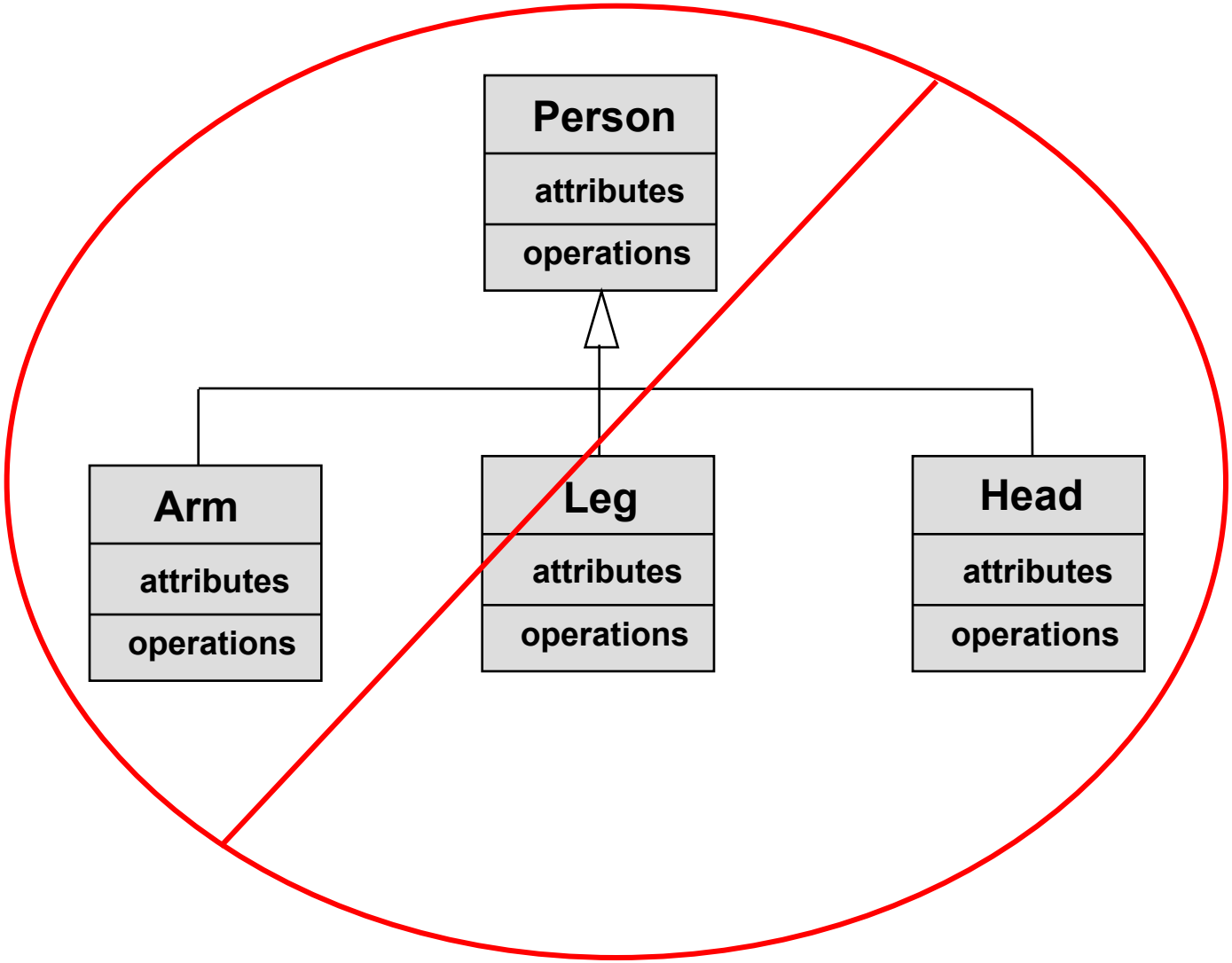
Example:


Customer
Regular Customer
Loyalty Customer

-

# *Poor Generalization Example*

***(violates the "is a" or "is a kind of" heuristic)***



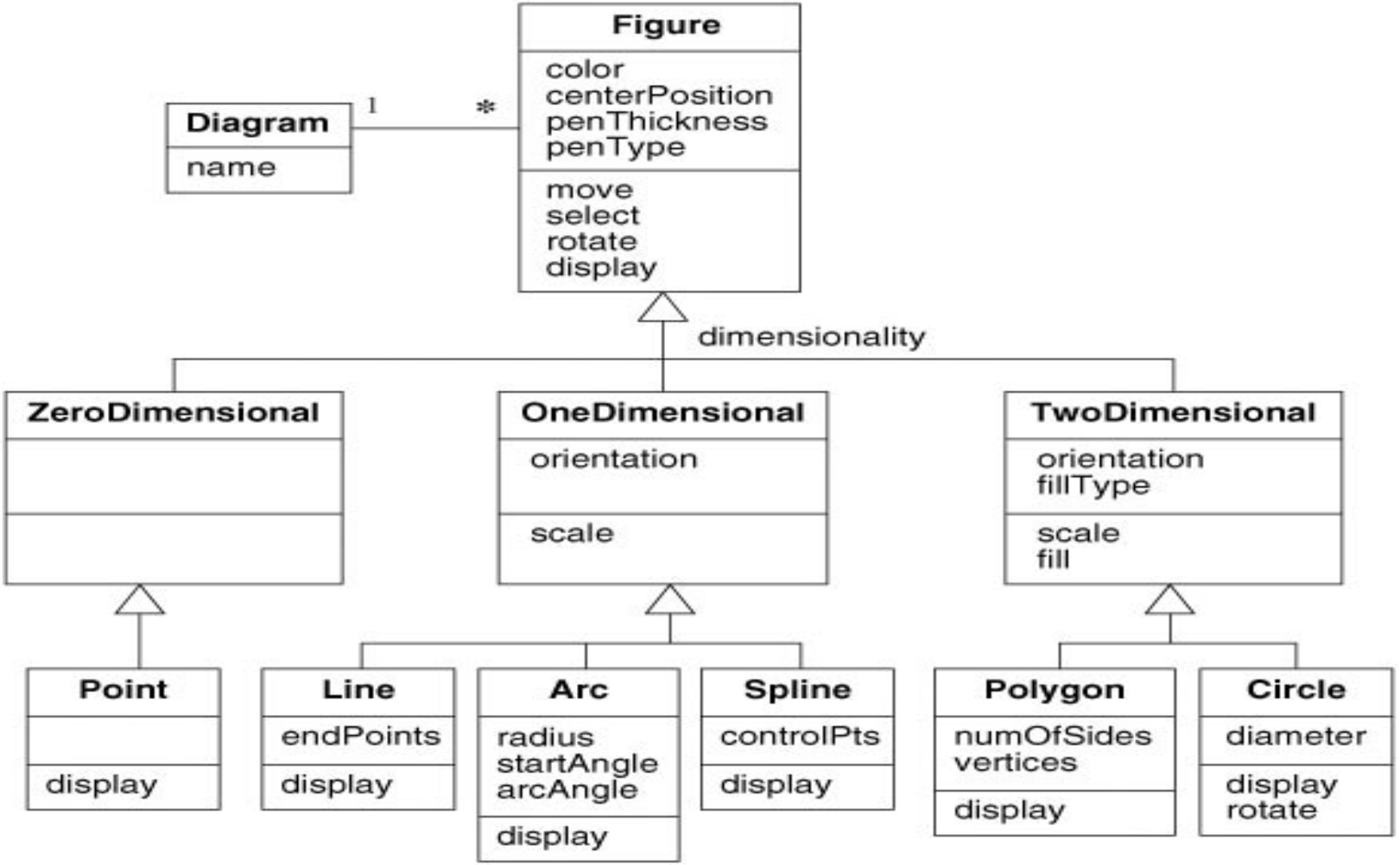
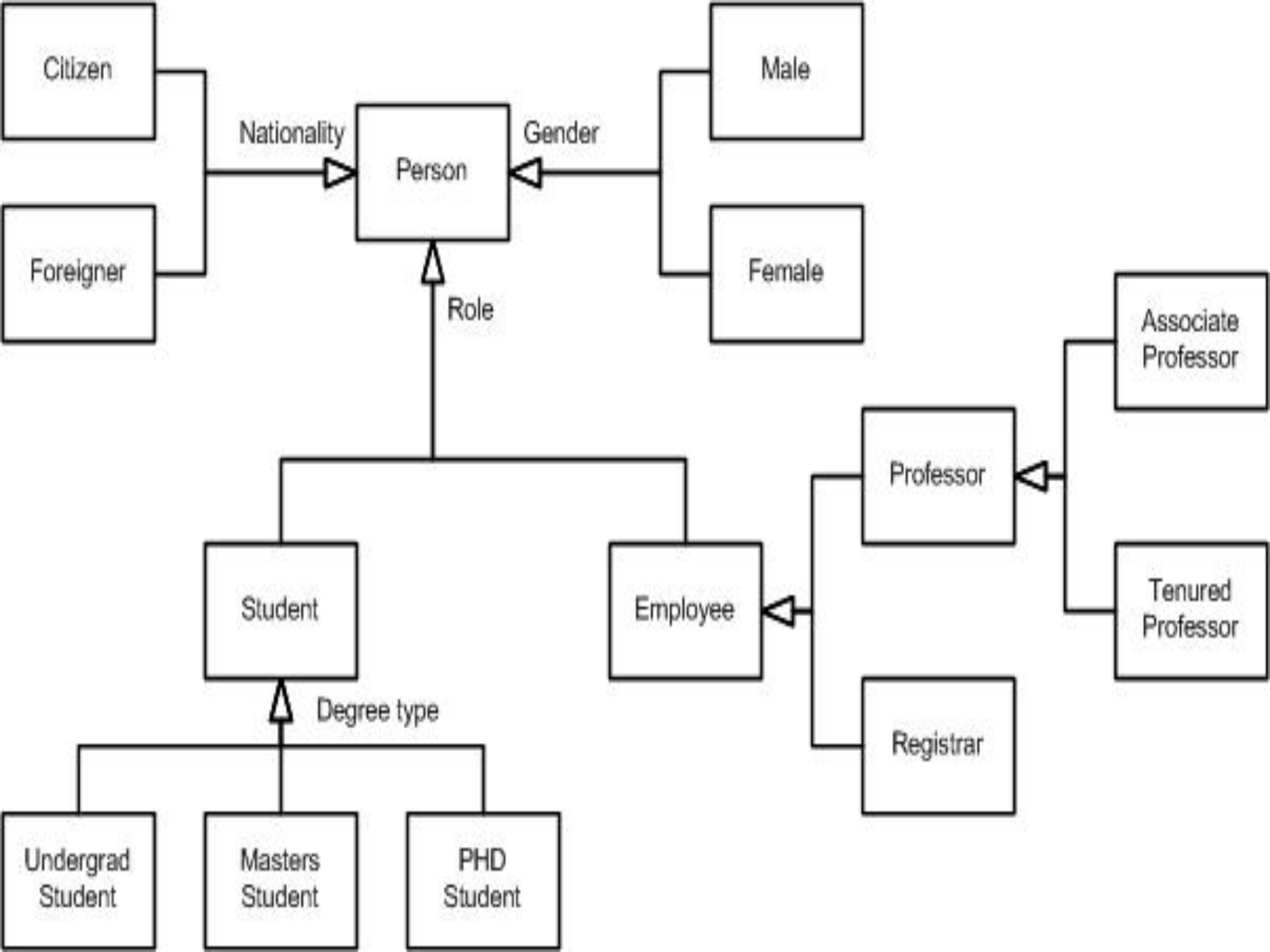



**Figure 3.25 Inheritance for graphic figures.** Each subclass inherits the attributes, operations, and associations of its superclasses.


Citizen
Foreigner
Nationality
Person
Gender
Male
Female
Role
Student
Employee
Professor
Associate Professor
Tenured Professor
Registrar
Degree type
Undergrad Student
Masters Student
PHD Student

# Use of generalization

**Used for three purposes:**

- **Support of polymorphism:**

  polymorphism increases the flexibility of software.

  Adding a new subclass and automatically inheriting superclass behavior.

- **Structuring the description of objects:**

  Forming a classification, organizing objects according to their similarities. It is much more profound than modeling each class individually and in isolation of other similar classes.

- **Enabling code reuse:**

  Reuse is more productive than repeatedly writing code from scratch.

# Generalization, Specialization, and Inheritance

- The terms generalization, specialization, and inheritance all refer to aspects of the same idea.
- Generalization and specialization concern a relationship among classes and take opposite perspectives, viewed from the superclass or from the subclasses.
- Generalization derives from the fact that the superclass generalizes the subclasses
- Specialization refers to the fact that the subclasses refine or specialize the superclass.
- Inheritance is the mechanism for sharing attributes, operations, and associations via the generalization/specialization relationship.
  - Generalization represents a relationship at the conceptual level
  - Inheritance is an implementation technique

# Exercise 3.13 a or b

Attempt 3.13 a or b

3.13 Prepare a class diagram for each group of classes. Add at least 10 relationships (associations and generalizations) to each diagram. Use association names and association end names where needed. Also use qualified associations and show multiplicity. You do not need to show a tributes or operations. As you prepare the diagrams, you may add classes. Be sure to expla your diagrams.

a. (6) school, playground, principal, school board, classroom, book, student, teacher, cafeter restroom, computer, desk, chair, ruler, door, swing

b. (4) automobile, engine, wheel, brake, brake light, door, battery, muffler, tail pipe

# Exercise 3.13 a : Solution

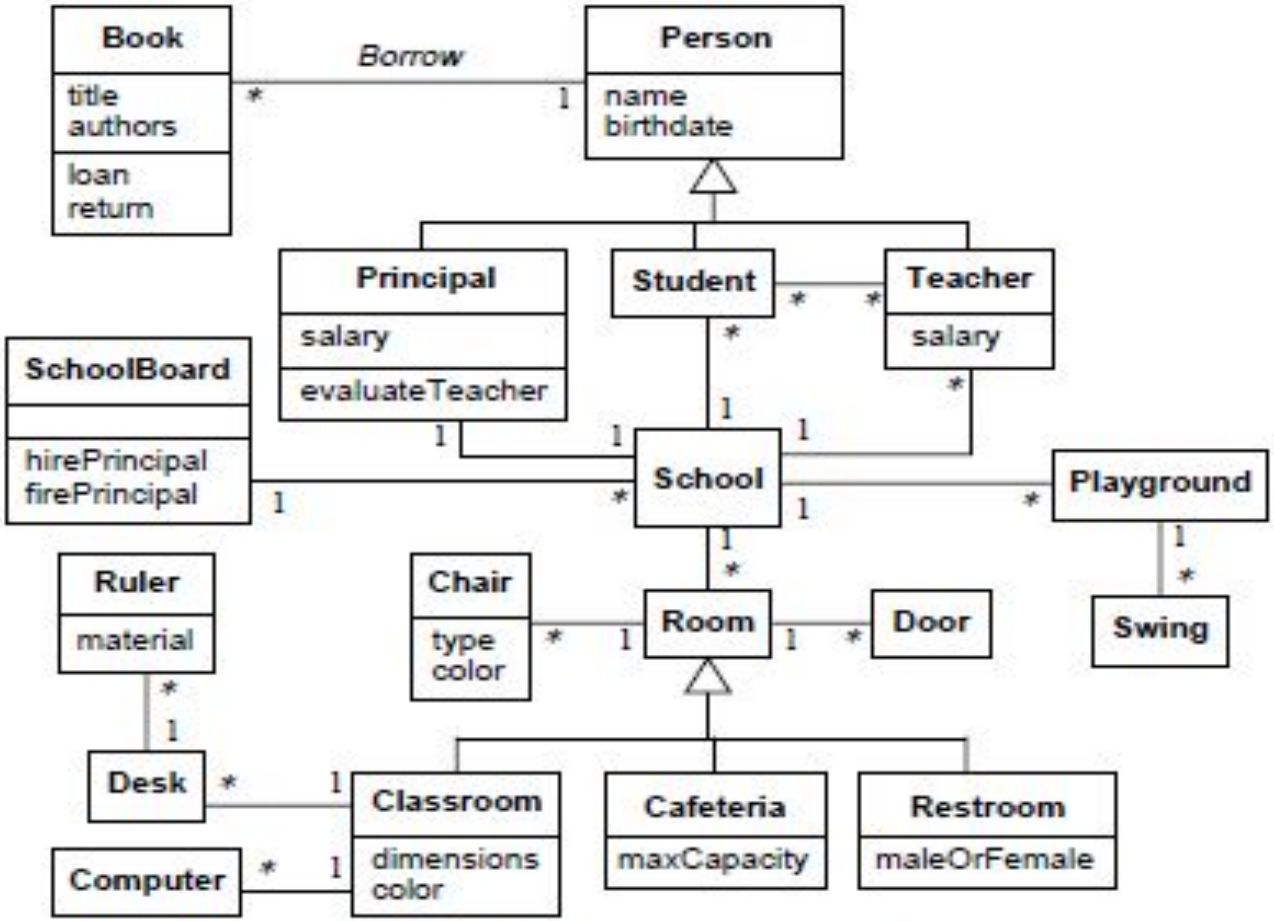



**Figure A3.11** Class diagram for a school

# Exercise 3.13 b : Solution

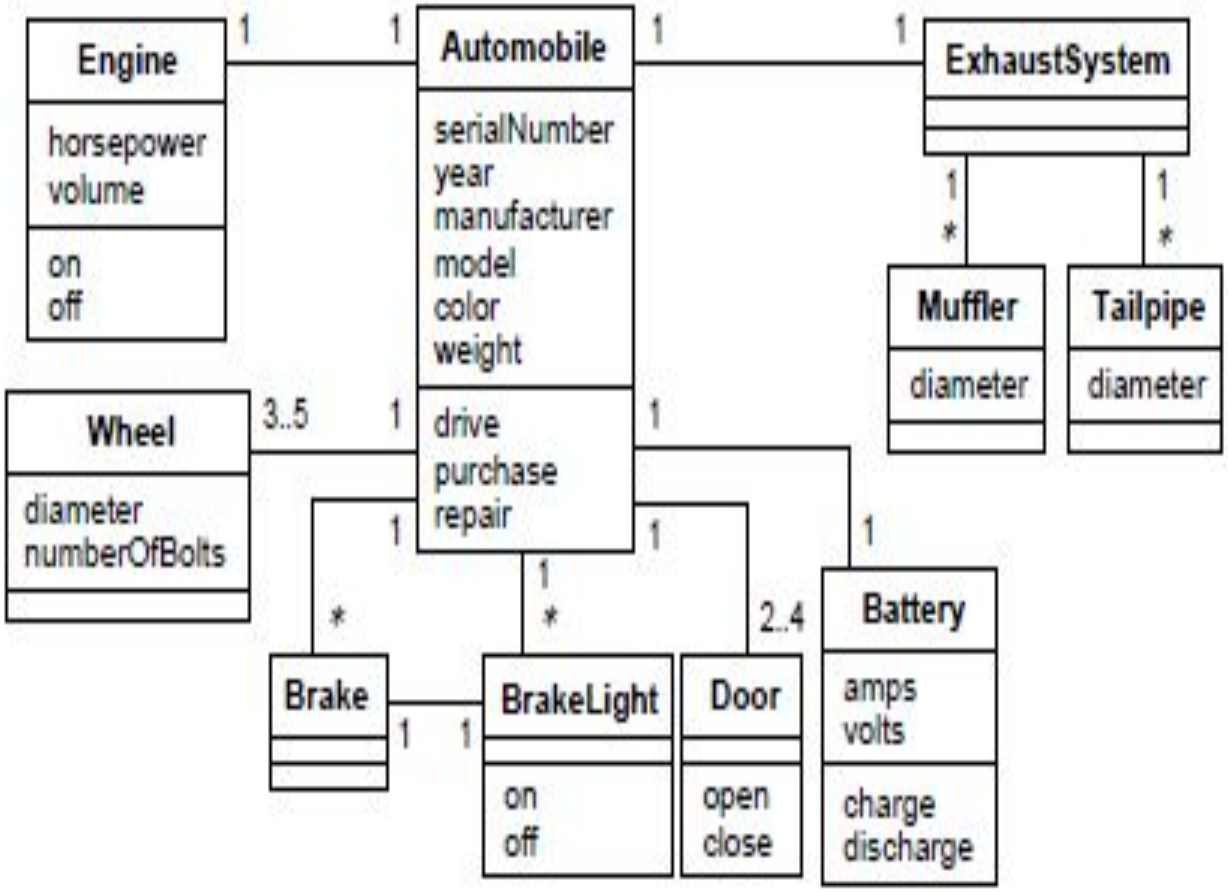



Figure A3.12 Class diagram for an automobile

## Practical Tips

**Scope**. Don't begin class modeling by merely jotting down classes, associations, and in- heritance. First, you must understand the problem to be solved. You must exercise judgment in deciding which objects to show and which objects to ignore. A model represents only the relevant aspects of a problem.

**Simplicity**. Strive to keep your models simple. A simple model is easier to understand and takes less development effort. Try to use a minimal number of classes that are clearly defined and not redundant. Be suspicious of classes that are difficult to define. You may need to reconsider such classes and restructure the model.

**Diagram layout**. Draw your diagrams in a manner that elicits symmetry. Often there is a superstructure to a problem that lies outside the notation. Try to position important classes so that they are visually prominent on a diagram. Try to avoid crossing lines.

**Names.** Carefully choose names. Names are important and carry powerful connotations. Names should be descriptive, crisp, and unambiguous. Choosing good names is one of the most difficult aspects of modeling. You should use singular nouns for the names of classes.

**References.** Do not bury object references inside objects as attributes. Instead, model these as associations. This is clearer and captures the true intent rather than an imple- mentation approach. (Section 3.2.1)

**Multiplicity**. Challenge association ends with a multiplicity of one. Often the object on either end is optional and zero-or-one multiplicity may be more appropriate. Other times “many” multiplicity is needed.

**Association end names.** Be alert for multiple uses of the same class. Use association end names to unify references to the same class.

**Bags and sequences.** An ordinary binary association has at most one link for a pair of objects. However, you can permit multiple links for a pair of objects by annotating an association end with {bag} or {sequence}.

**Attributes of associations.** During analysis, do not collapse attributes of associations into one of the related classes. You should directly describe the objects and links in your models. During design and implementation, you can always combine information for more efficient execution.

**Qualified associations**. Challenge association ends with a multiplicity of “many.” A qualifier can often improve the precision of an association and highlight important navigation paths.

**Generalization levels**. Try to avoid deeply nested generalizations.

**Overriding features**. You may override methods and default values of attributes. How- ever, you should never override a feature so that it is inconsistent with the signature or semantics of the original inherited feature.

**Reviews**. Try to get others to review your models. Expect that your models will require revision. Class models require revision to clarify names, improve abstraction, repair errors, add information, and more accurately capture structural constraints. Nearly all of our models have required several revisions.

**Documentation**. Always document your models. The diagram specifies the structure of a model but cannot describe the rationale. The written explanation guides the reader and explains subtle reasons for why the model was constructed a particular way.