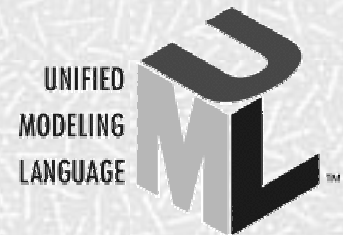


# Object Modeling with UML: Introduction to UML

Cris Kobryn  
Chief Technologist  
E.solutions, EDS

November 1999

---



© 1999 OMG and Tutorial Contributors: EDS, IBM, Enea Data, Klasse Objecten, ObjectTime Ltd., Rational Software, Unisys

# Overview

---

- Tutorial series
- Quick tour
- Structural modeling
- Use case modeling

# Tutorial Series

---

- Introduction to UML
  - November 1999, Cambridge, US
- Behavioral Modeling with UML
  - January 2000, Mesa, Arizona, US
- Advanced Modeling with UML
  - March 2000, Denver, US
- Metadata Integration with UML, XMI and MOF
  - June 2000, Oslo, Norway



# Tutorial Goals

---

- What you will learn:
  - what the UML is and what is it not
  - UML's basic constructs, rules and diagram techniques
  - how the UML can model large, complex systems
  - how the UML can specify systems in an implementation-independent manner
  - how UML, XMI and MOF can facilitate metadata integration
- What you will not learn:
  - Object Modeling 101
  - object methods or processes
  - Metamodeling 101

# Quick Tour

---

- Why do we model?
- What is the UML?
- Foundation elements
- Unifying concepts
- Language architecture
- Relation to other OMG technologies



# Why do we model?

---

- Provide structure for problem solving
- Experiment to explore multiple solutions
- Furnish abstractions to manage complexity
- Reduce time-to-market for business problem solutions
- Decrease development costs
- Manage the risk of mistakes

# The Challenge



Tijuana “shantytown”:  
<http://www.macalester.edu/~jschatz/residential.html>

# The Vision



Fallingwater:

<http://www.adelaide.net.au/~jpolias/FLW/Images/FallingWater.jpeg>



# Why do we model graphically?

---

- Graphics reveal data.
  - Edward Tufte  
*The Visual Display of Quantitative Information*, 1983
- 1 bitmap = 1 megaword.
  - Anonymous visual modeler

# Quick Tour

---

- The UML is a graphical language for
  - specifying
  - visualizing
  - constructing
  - documentingthe artifacts of software systems
- Added to the list of OMG adopted technologies in November 1997 as UML 1.1
- Most recent minor revision is UML 1.3 (November 1999)

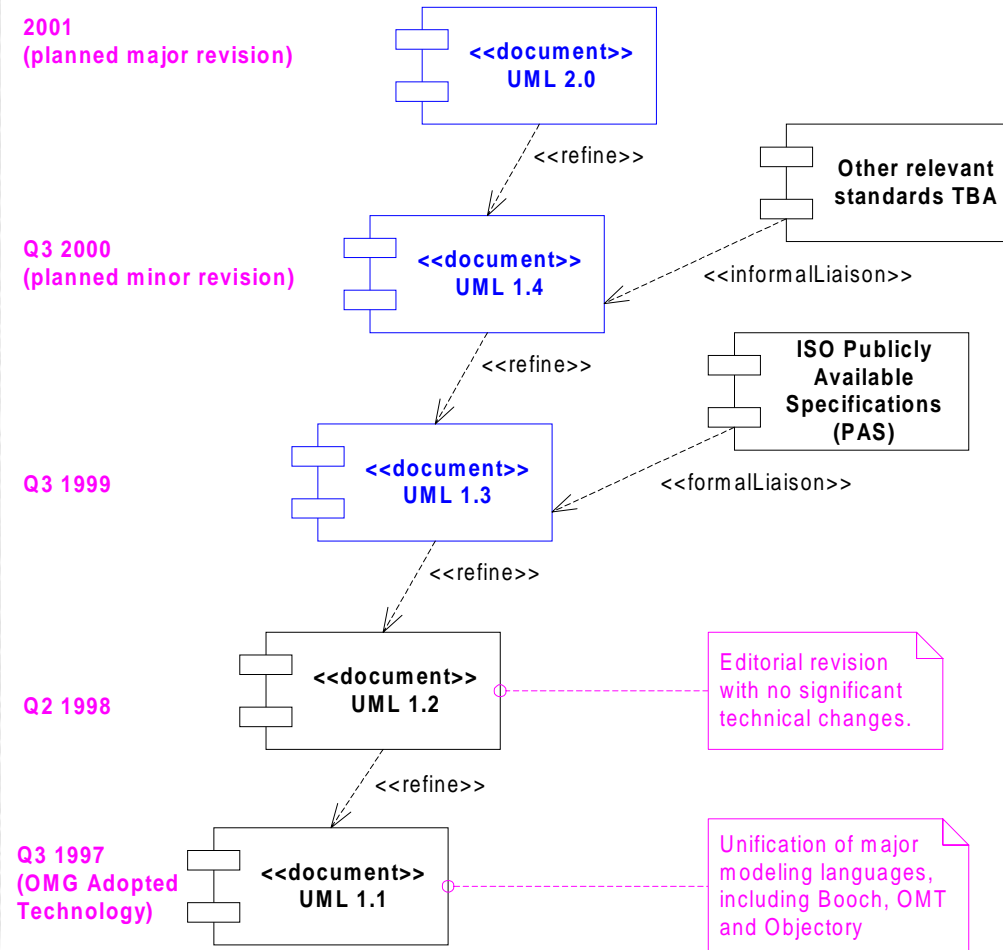


# UML Goals

- Define an easy-to-learn but semantically rich visual modeling language
- Unify the Booch, OMT, and Objectory modeling languages
- Include ideas from other modeling languages
- Incorporate industry best practices
- Address contemporary software development issues
  - scale, distribution, concurrency, executability, etc.
- Provide flexibility for applying different processes
- Enable model interchange and define repository interfaces



# OMG UML Evolution



# OMG UML Contributors

Contributors = Original Submitters + RTF1 + RTF2 + ...

Aonix  
Colorado State University  
Computer Associates  
Concept Five  
Data Access  
EDS  
Enea Data  
Hewlett-Packard  
IBM  
I-Logix  
Inline Software  
Intellicorp  
Klasse Objecten  
Lockheed Martin

Microsoft  
ObjecTime  
Oracle  
Ptech  
OAO Technology Solutions  
Rational Software  
Reich  
SAP  
Softeam  
Sterling Software  
Sun  
Taskon  
Telelogic  
Unisys  
...

# OMG UML 1.3 Specification

---

- UML Summary
- UML Semantics
- UML Notation Guide
- UML Standard Profiles
  - Software Development Processes
  - Business Modeling
- UML CORBAfacility Interface Definition
- UML XML Metadata Interchange DTD
- Object Constraint Language



# Tutorial Focus: the Language

- language = syntax + semantics
  - syntax = rules by which language elements (e.g., words) are assembled into expressions (e.g., phrases, clauses)
  - semantics = rules by which syntactic expressions are assigned meanings
- *UML Notation Guide* – defines UML's graphic syntax
- *UML Semantics* – defines UML's semantics

# Foundation Concepts

---

- Building blocks
- Well-formedness rules

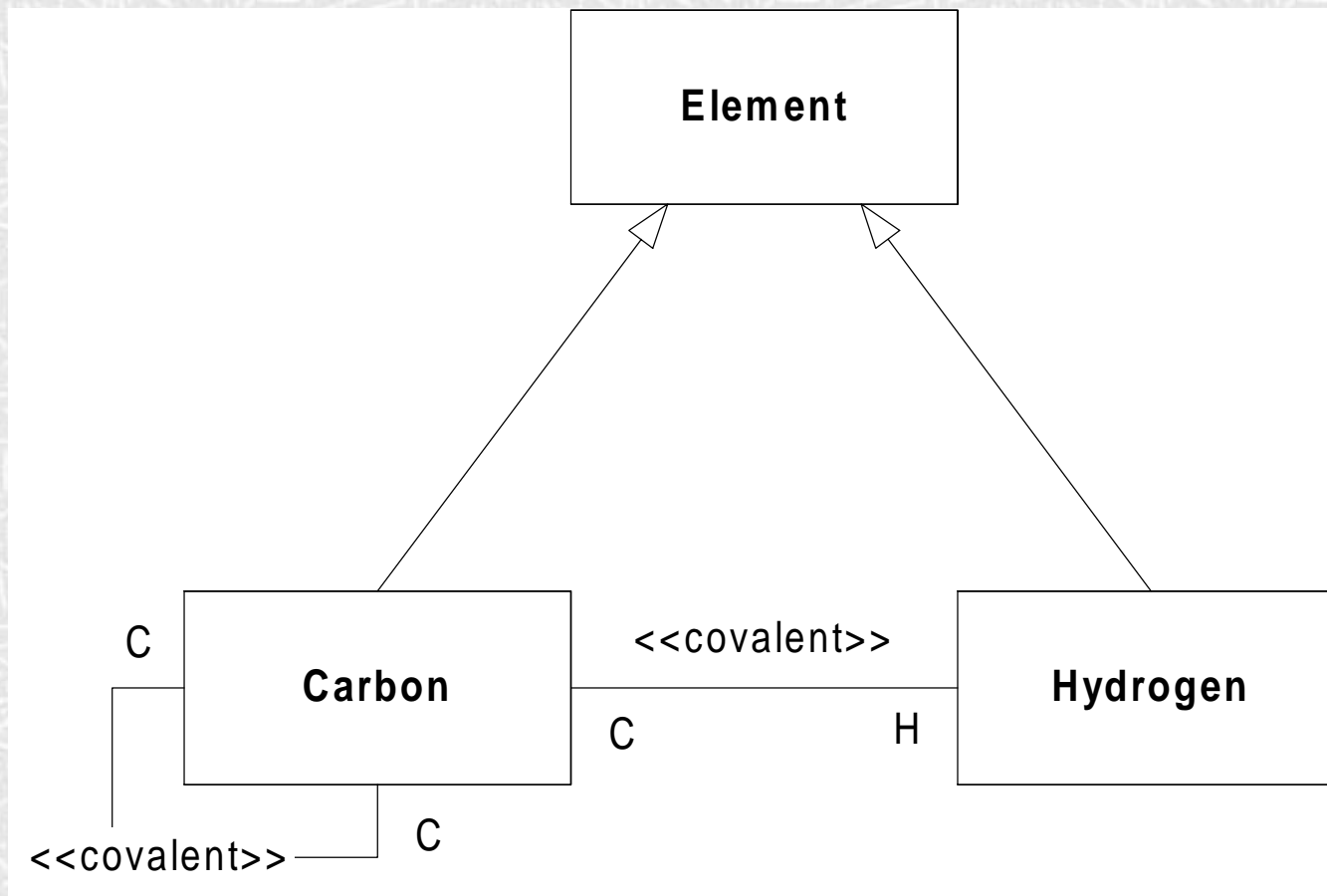


# Building Blocks

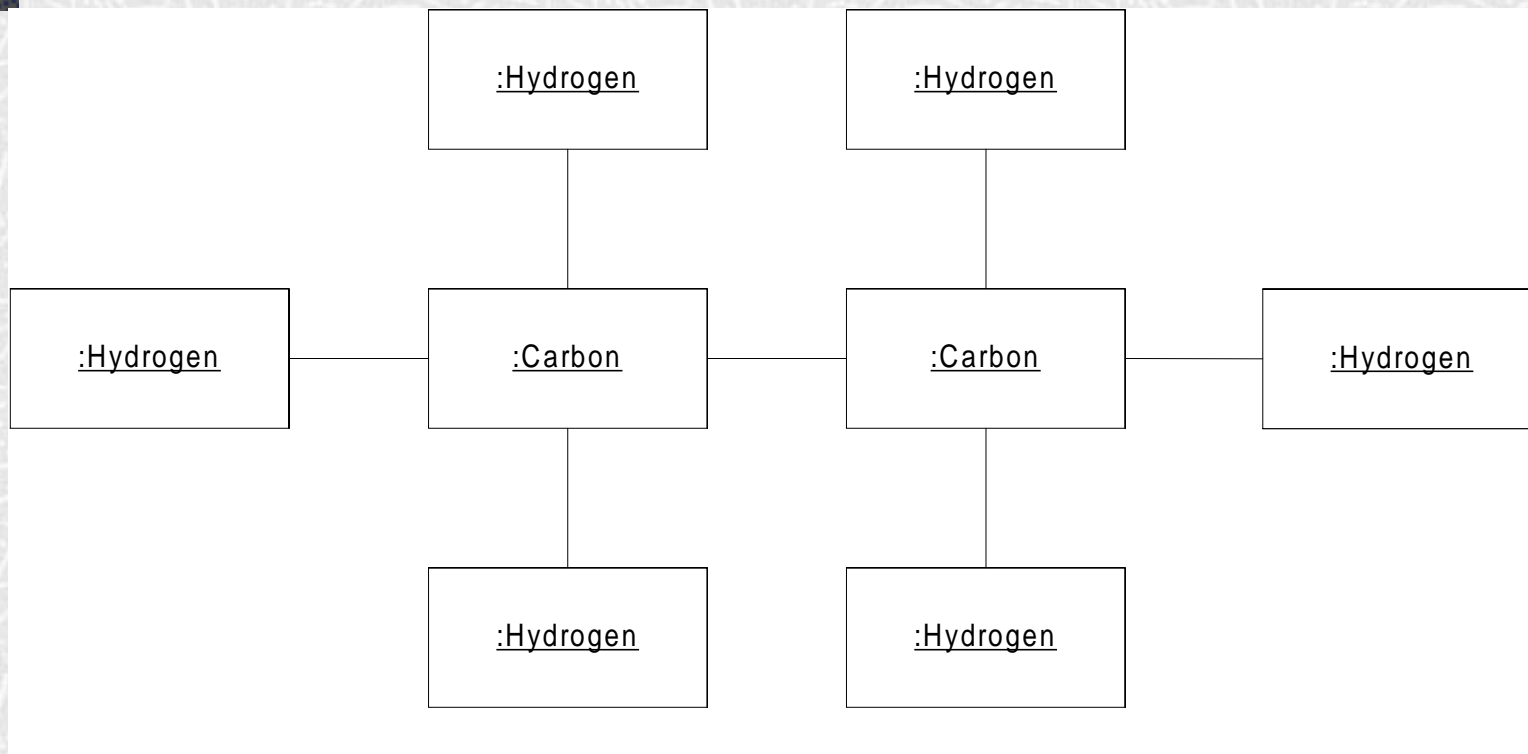
- The basic building blocks of UML are:
  - model elements (classes, interfaces, components, use cases, etc.)
  - relationships (associations, generalization, dependencies, etc.)
  - diagrams (class diagrams, use case diagrams, interaction diagrams, etc.)
- Simple building blocks are used to create large, complex structures
  - cf. elements, bonds and molecules in chemistry
  - cf. components, connectors and circuit boards in hardware



# Diagram: Classifier View



# Diagram: Instance View



# Well-Formedness Rules

---

- Well-formed: indicates that a model or model fragment adheres to all semantic and syntactic rules that apply to it.
- UML specifies rules for:
  - naming
  - scoping
  - visibility
  - integrity
  - execution (limited)
- However, during iterative, incremental development it is expected that models will be incomplete and inconsistent.



# Well-Formedness Rules (cont'd)

- Example of semantic rule: Class [1]
  - **English:** If a Class is concrete, all the Operations of the Class should have a realizing Method in the full descriptor.
  - **OCL:** `not self.isAbstract implies self.allOperations->forAll (op | self.allMethods->exists (m | m.specification->includes(op)))`

# Well-Formedness Rules (cont'd)

---

- Example of syntactic rules: Class
  - **Basic Notation:** A class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines.
  - **Presentation Option:** Either or both of the attribute and operation compartments may be suppressed.
- Example of syntactic guideline: Class
  - **Style Guideline:** Begin class names with an uppercase letter.



# Unifying Concepts

---

- classifier-instance dichotomy
  - e.g., an object is an instance of a class OR a class is the classifier of an object
- specification-realization dichotomy
  - e.g., an interface is a specification of a class OR a class is a realization of an interface
- analysis-time vs. design-time vs. run-time
  - modeling phases (“process creep”)
  - usage guidelines suggested, not enforced

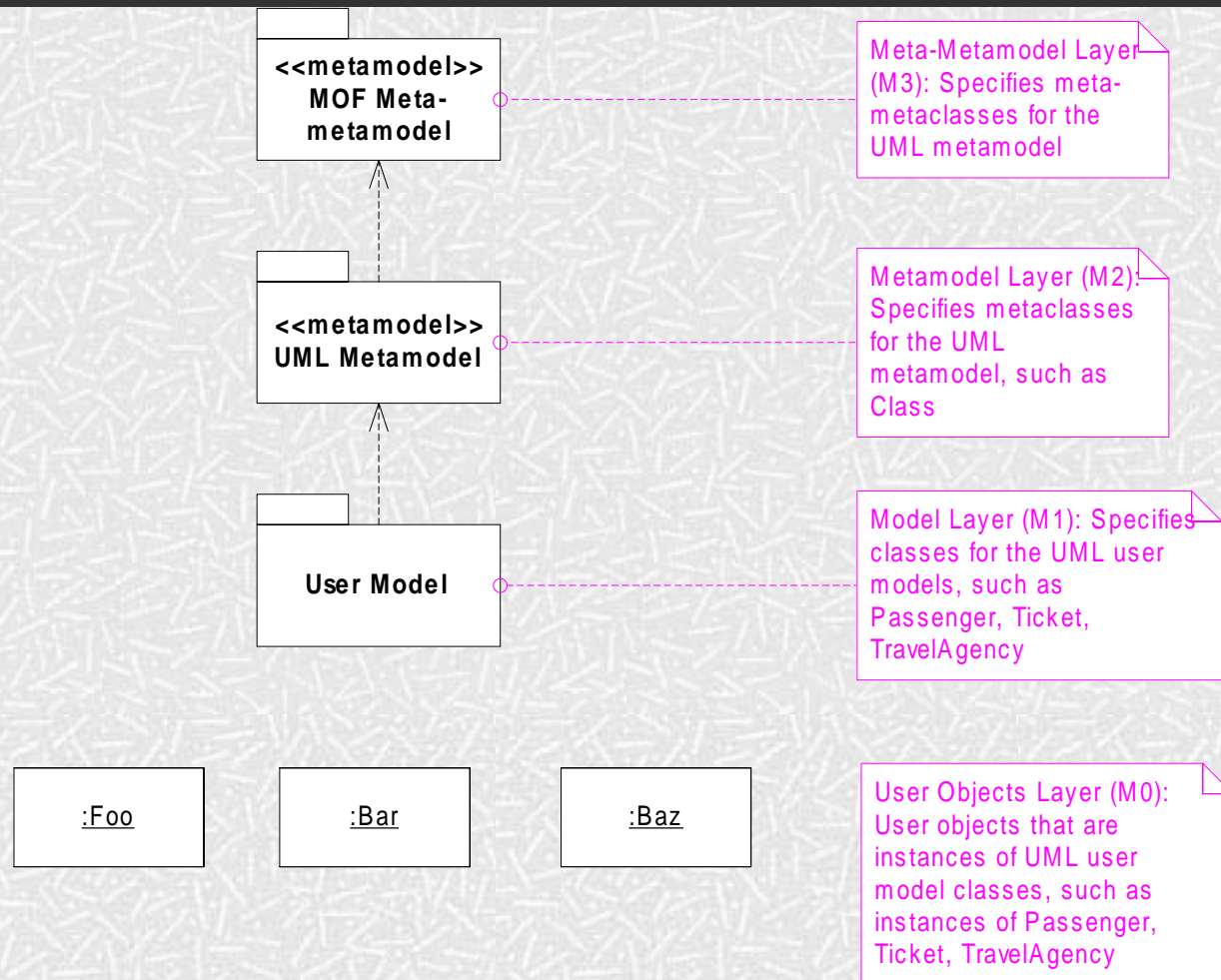


# Language Architecture

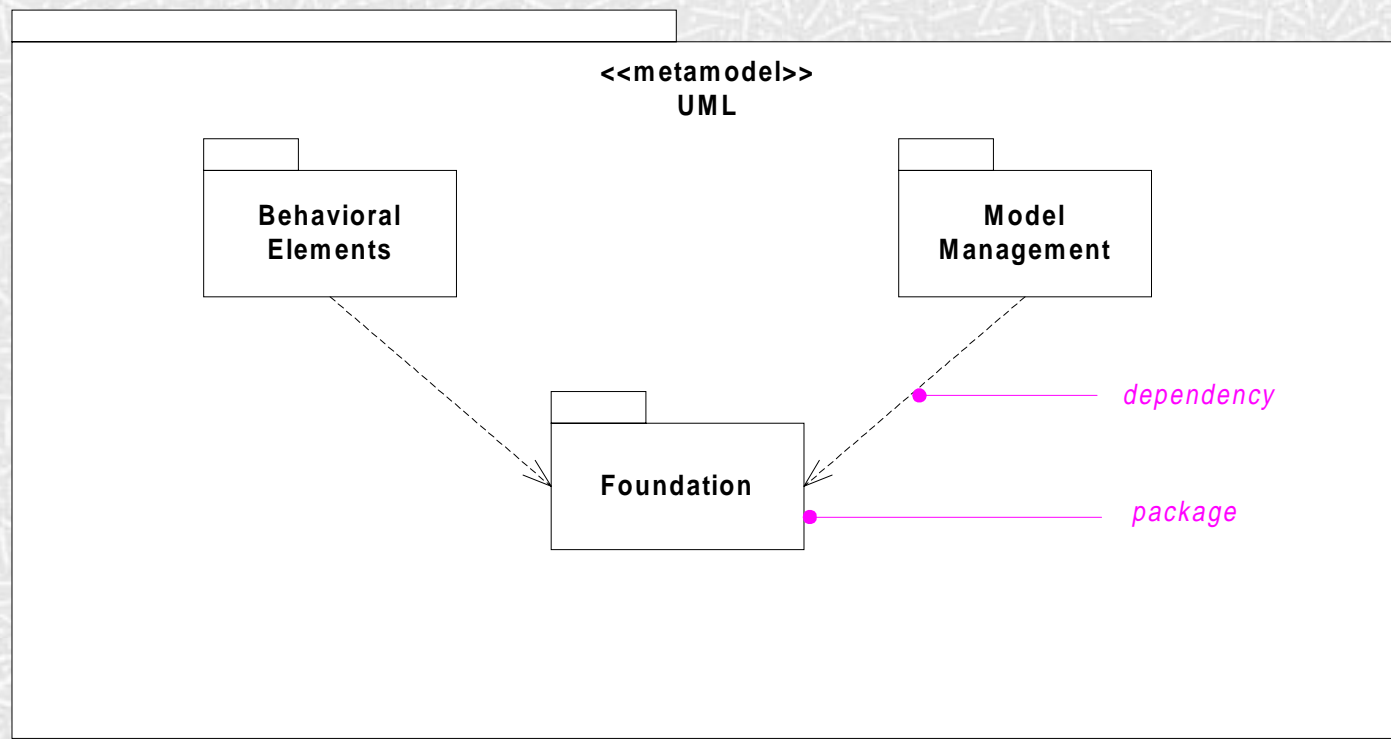
---

- Metamodel architecture
- Package structure

# Metamodel Architecture

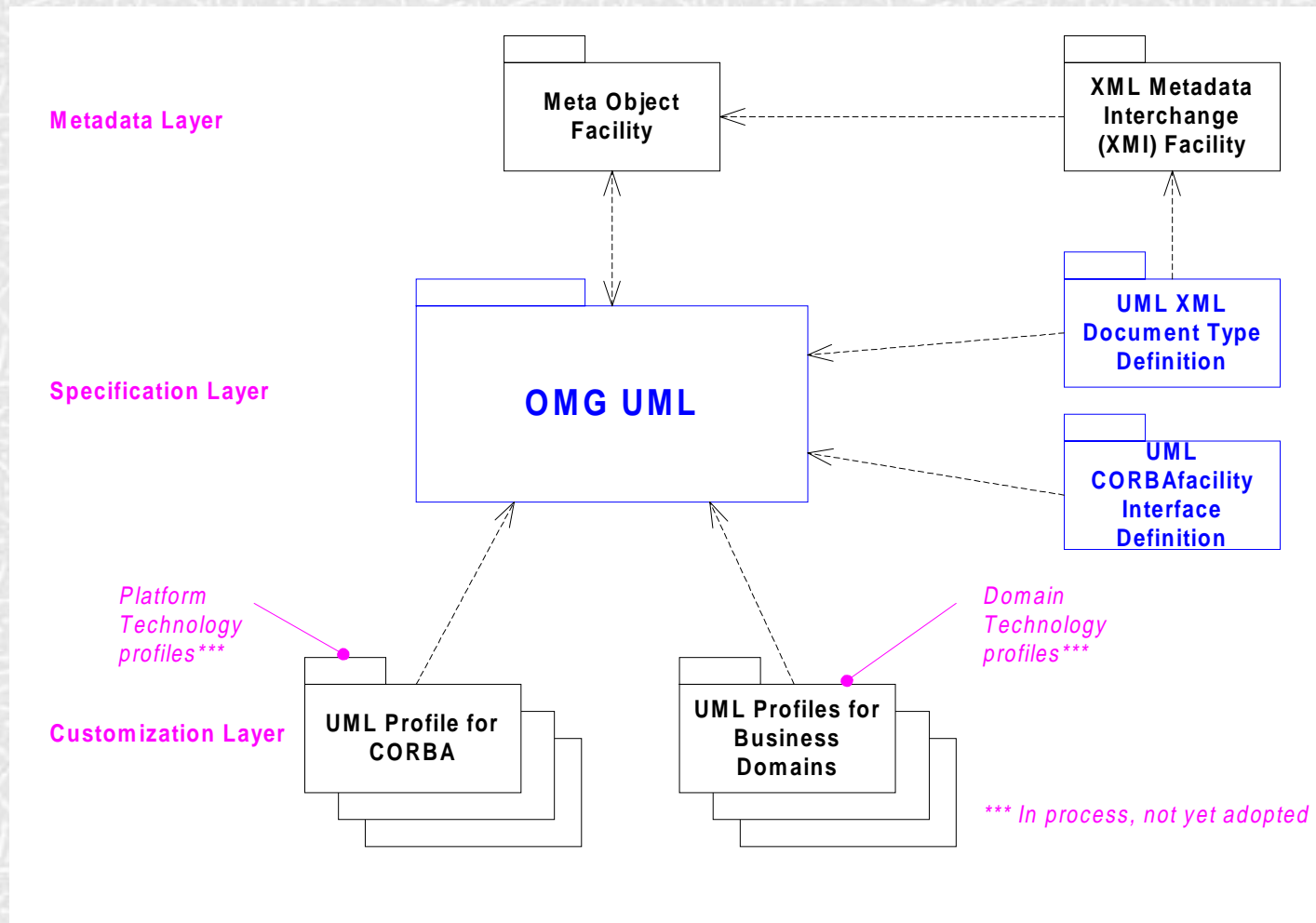


# Package Structure





# Relation to Other OMG Technologies



# Structural Modeling

---

- What is structural modeling?
- Core concepts
- Diagram tour
- When to model structure
- Modeling tips
- Example: Interface-based design

# What is structural modeling?

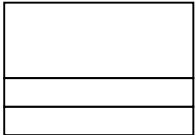
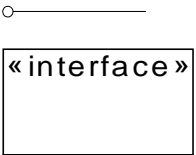

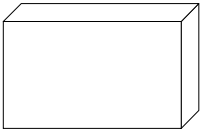
---

- Structural model: a view of an system that emphasizes the structure of the objects, including their classifiers, relationships, attributes and operations.



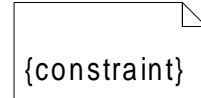
# *Structural Modeling:*

## Core Elements

Construct	Description	Syntax
<b>class</b>	a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.	
<b>interface</b>	a named set of operations that characterize the behavior of an element.	
<b>component</b>	a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces.	
<b>node</b>	a run-time physical object that represents a computational resource.	

# *Structural Modeling:*





## Core Elements (cont'd)

Construct	Description	Syntax
<b>constraint</b> <sup>1</sup>	a semantic condition or restriction.	

<sup>1</sup> An extension mechanism useful for specifying structural elements.

# *Structural Modeling:*


## Core Relationships

Construct	Description	Syntax
<b>association</b>	a relationship between two or more classifiers that involves connections among their instances.	
<b>aggregation</b>	A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.	
<b>generalization</b>	a taxonomic relationship between a more general and a more specific element.	
<b>dependency</b>	a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).	



# *Structural Modeling:*

## Core Relationships (cont'd)

Construct	Description	Syntax
<b>realization</b>	a relationship between a specification and its implementation.	

# Structural Diagram Tour

---

- Show the static structure of the model
  - the entities that exist (e.g., classes, interfaces, components, nodes)
  - internal structure
  - relationship to other entities
- Do not show
  - temporal information
- Kinds
  - static structural diagrams
    - class diagram
    - object diagram
  - implementation diagrams
    - component diagram
    - deployment diagram

# Static Structural Diagrams

---

- Shows a graph of classifier elements connected by static relationships.
- kinds
  - class diagram: classifier view
  - object diagram: instance view



# Classes

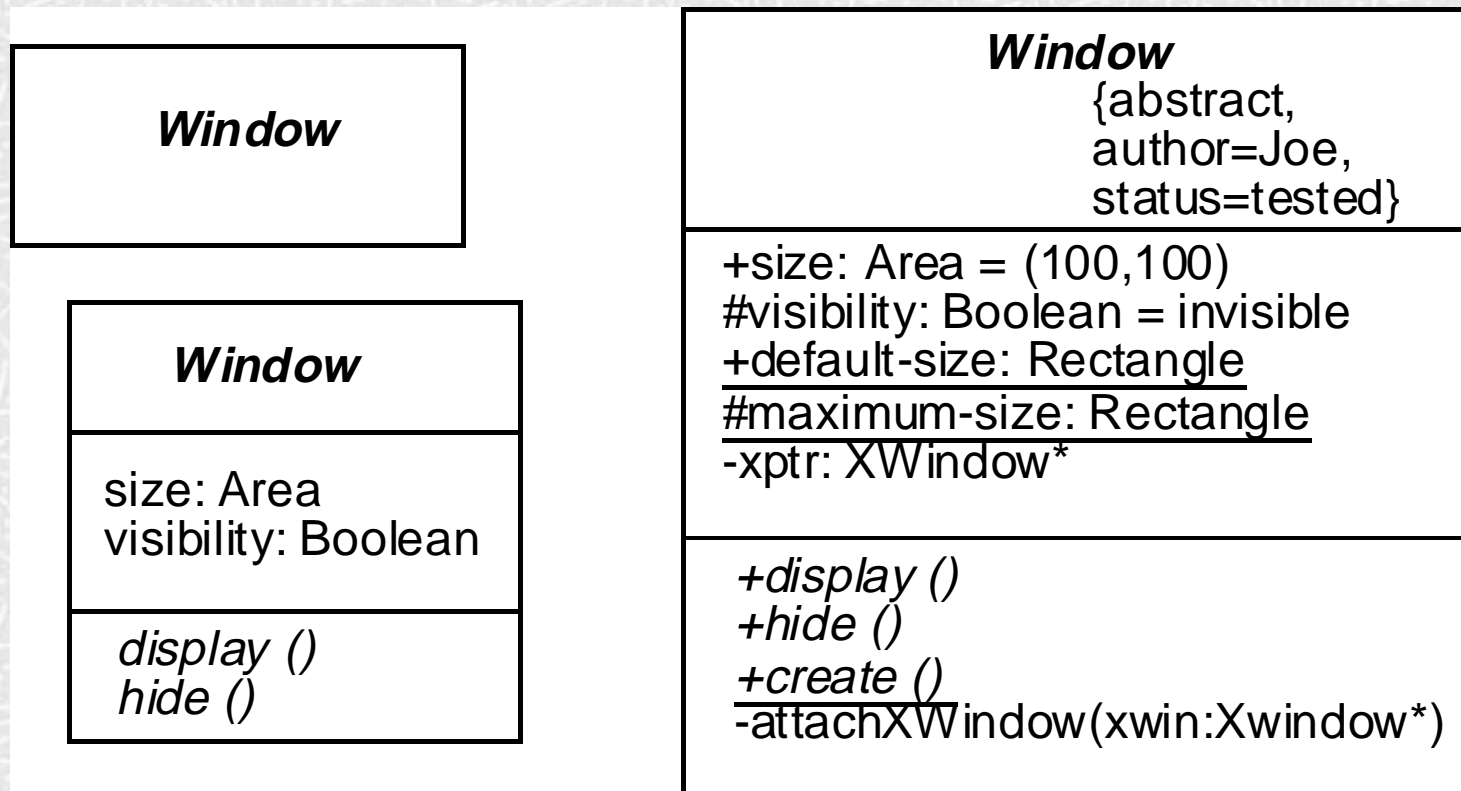


Fig. 3-17, *UML Notation Guide*

# Classes: compartments with names

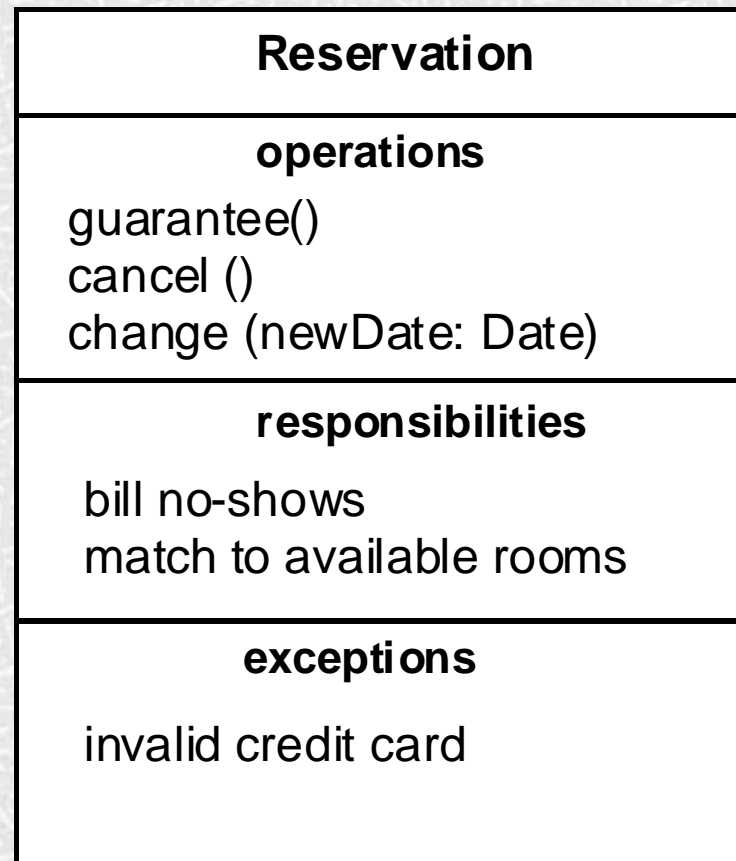


Fig. 3-20, *UML Notation Guide*

# Classes: method body

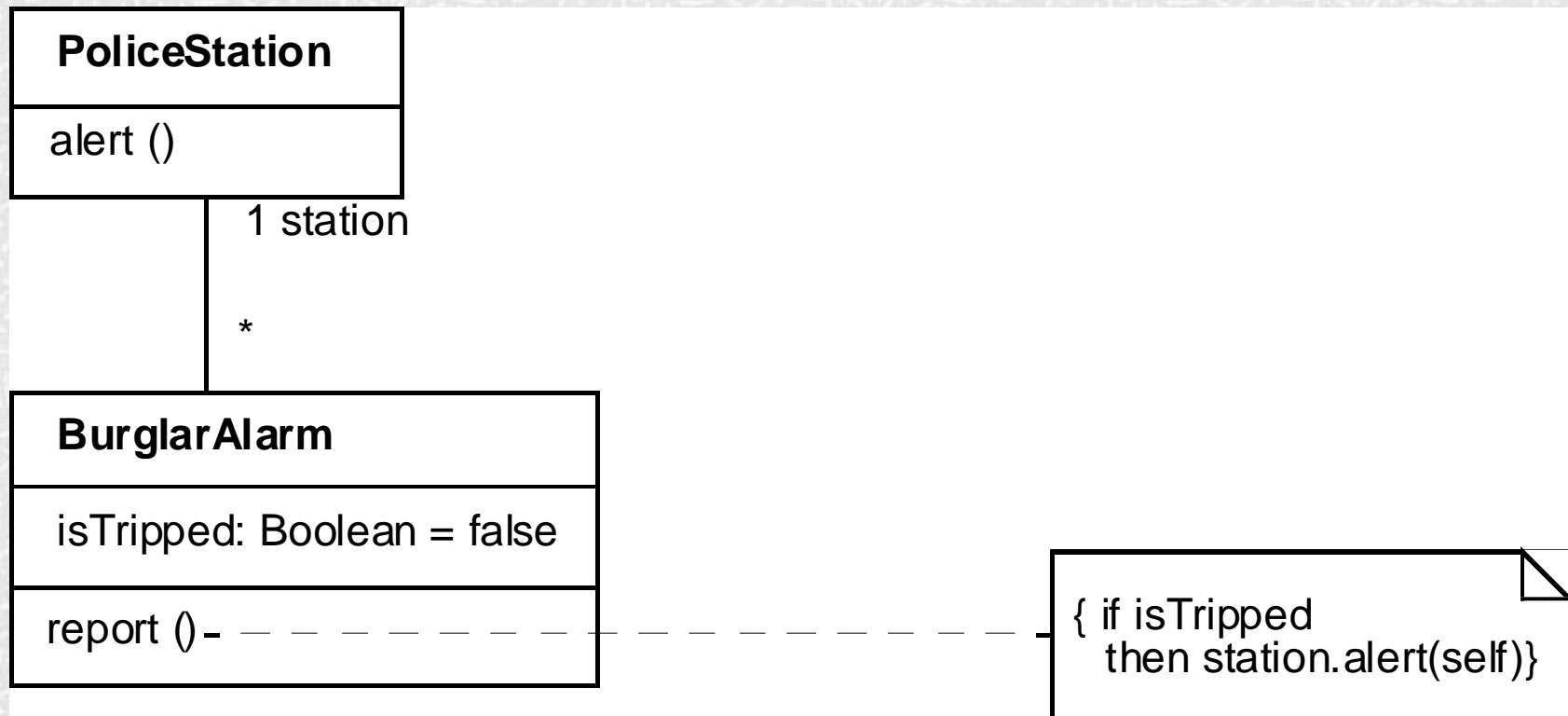


Fig. 3-21, *UML Notation Guide*



# Interfaces

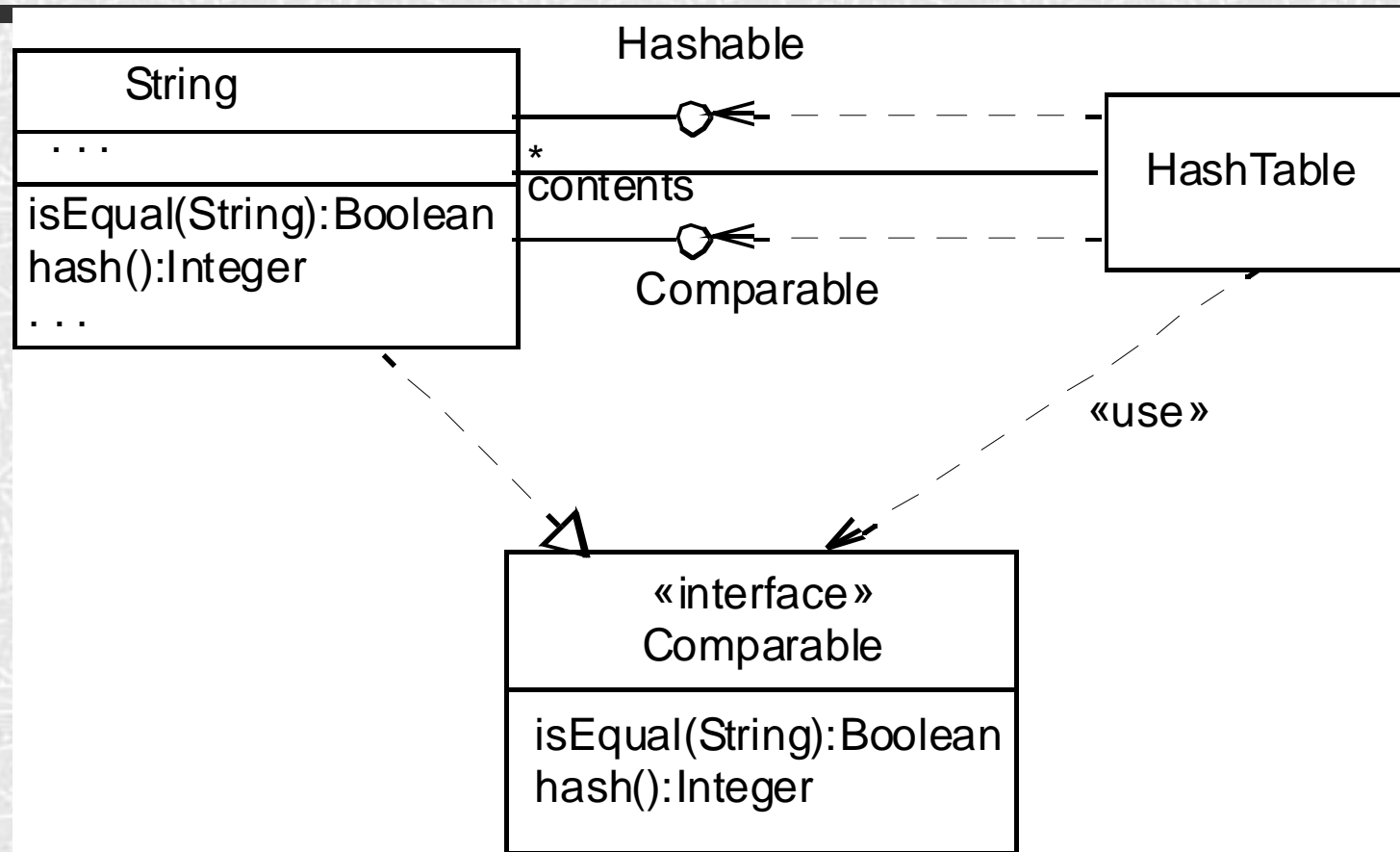


Fig. 3-24, *UML Notation Guide*

# Associations

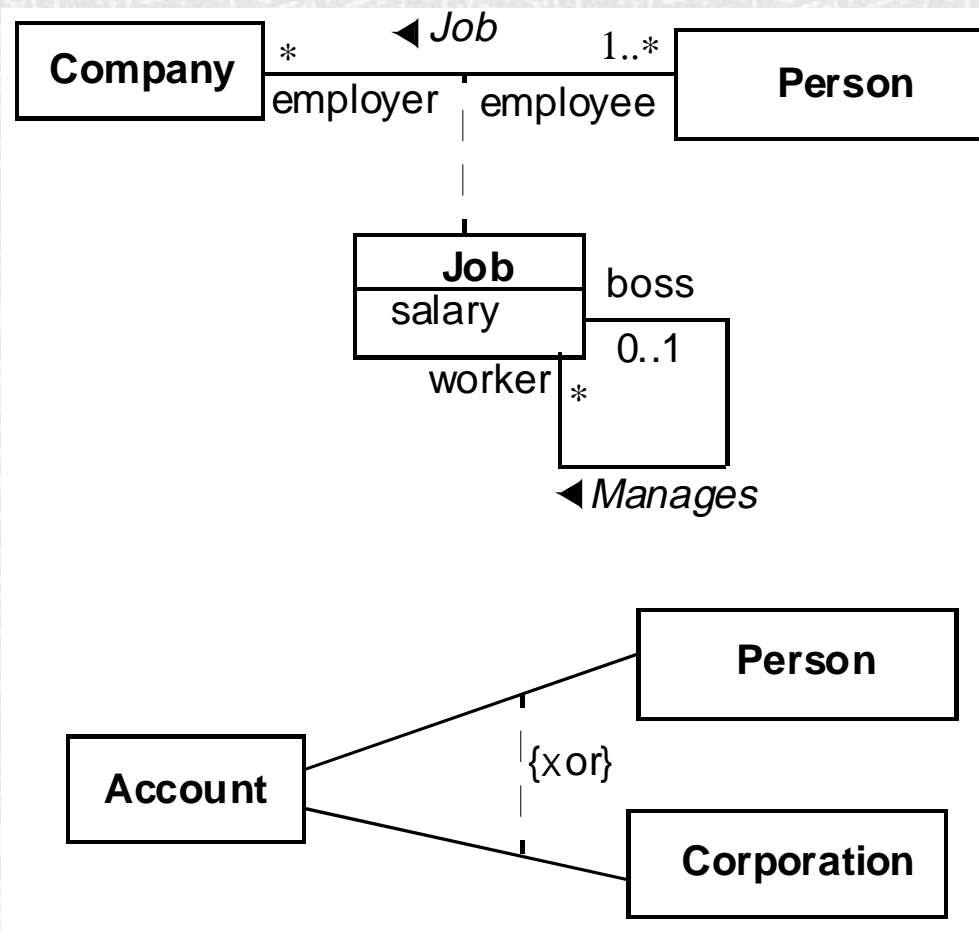


Fig. 3-31, *UML Notation Guide*  
Object Modeling with UML

# Association Ends

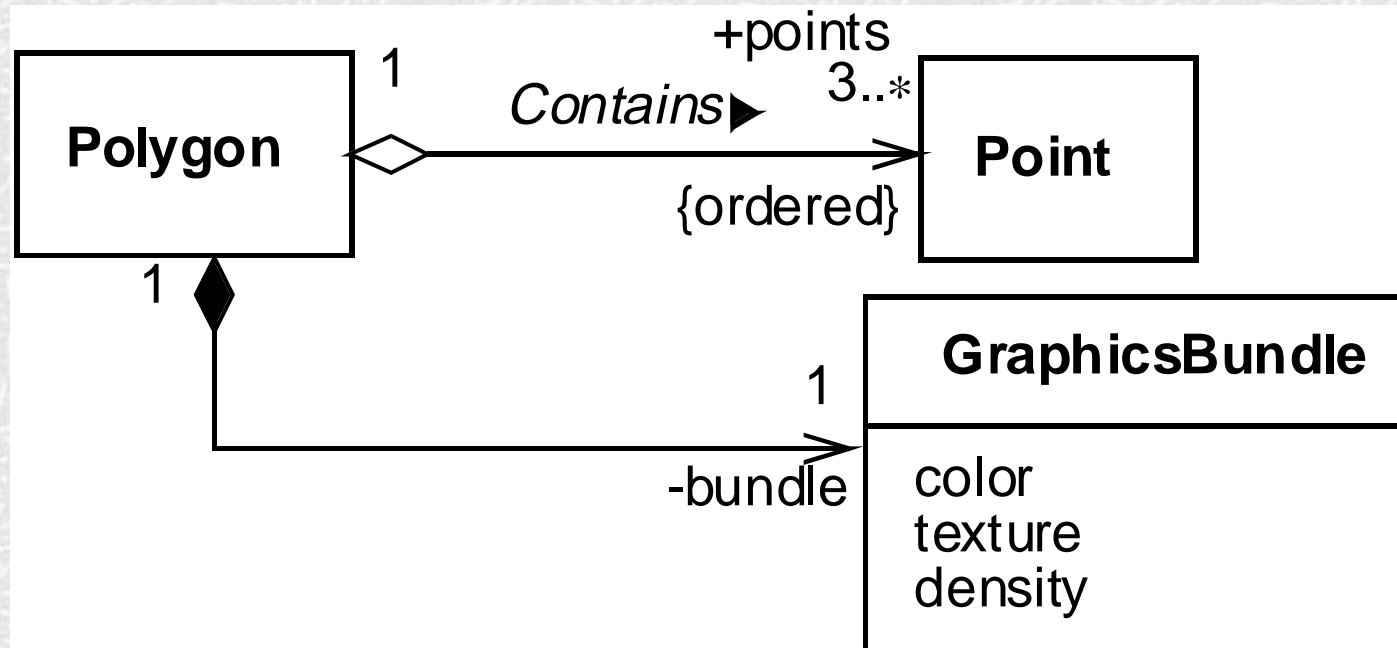


Fig. 3-32, *UML Notation Guide*



# Ternary Associations

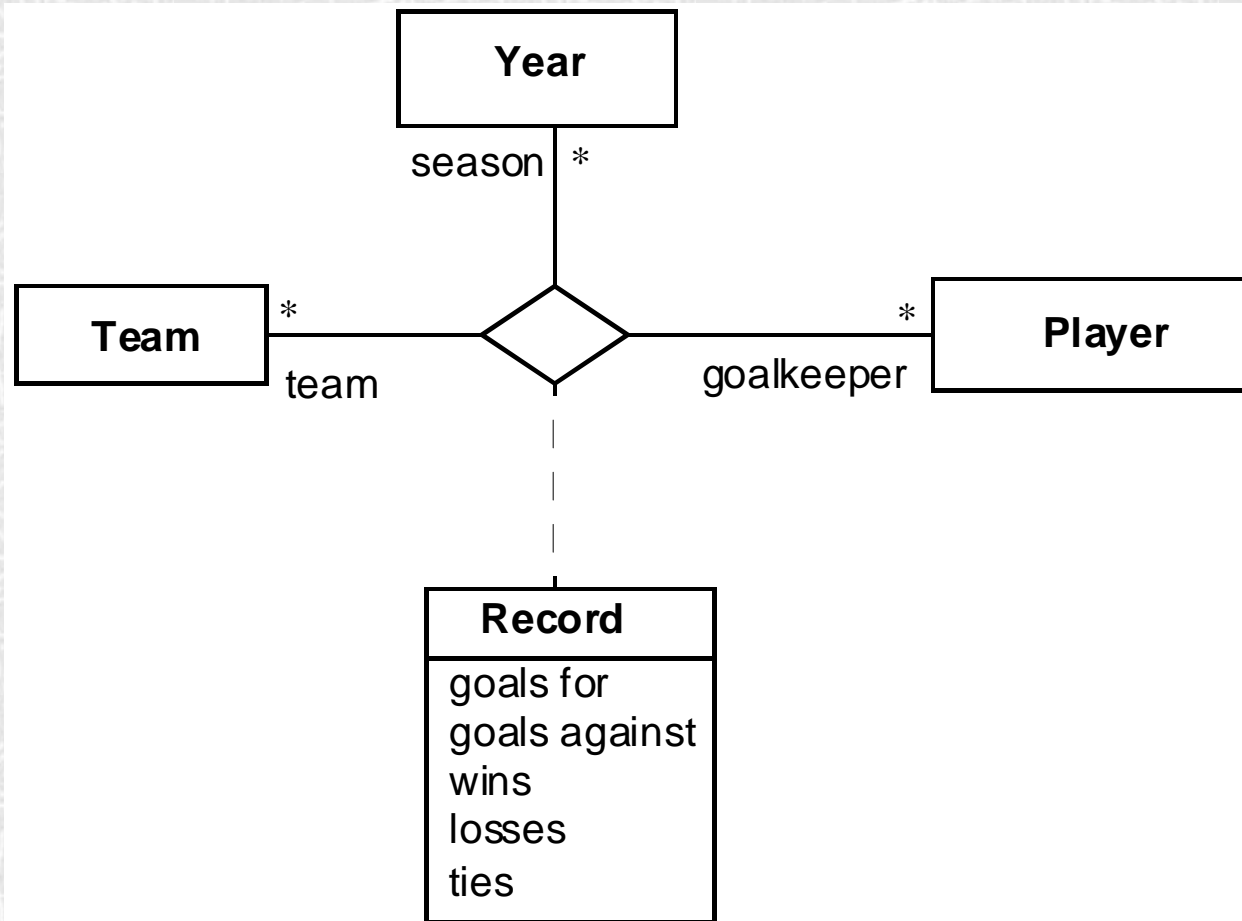


Fig. 3-31, *UML Notation Guide*

# Composition

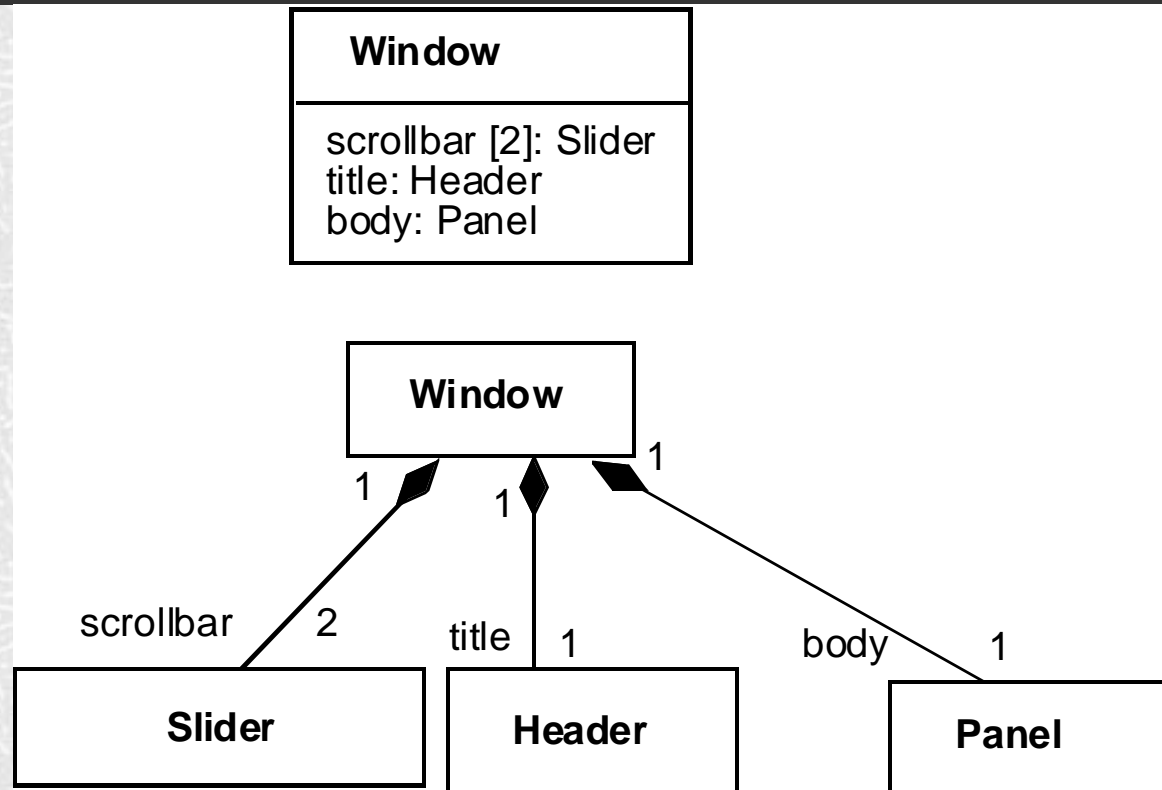


Fig. 3-36, *UML Notation Guide*

# Composition

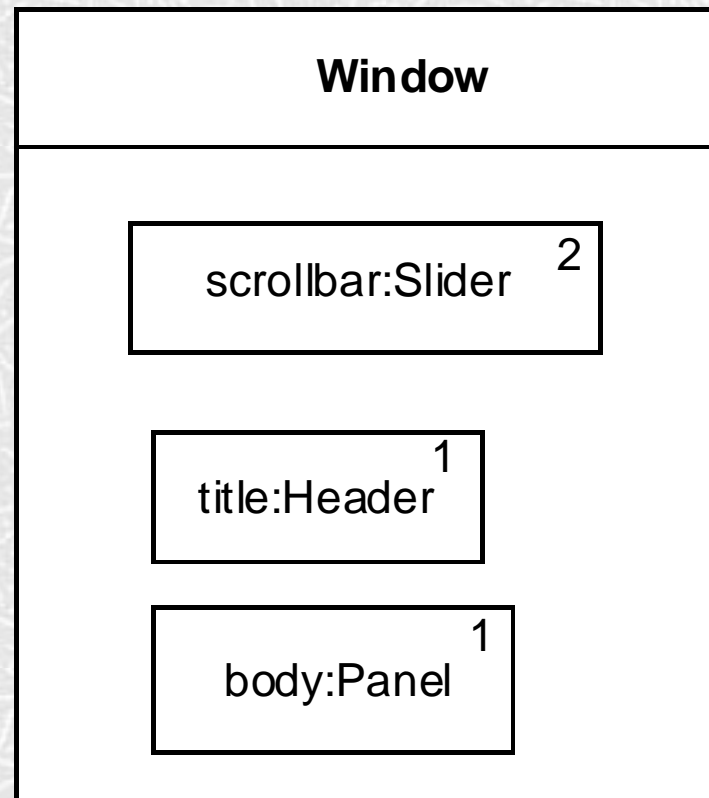


Fig. 3-36, *UML Notation Guide*



# Generalization

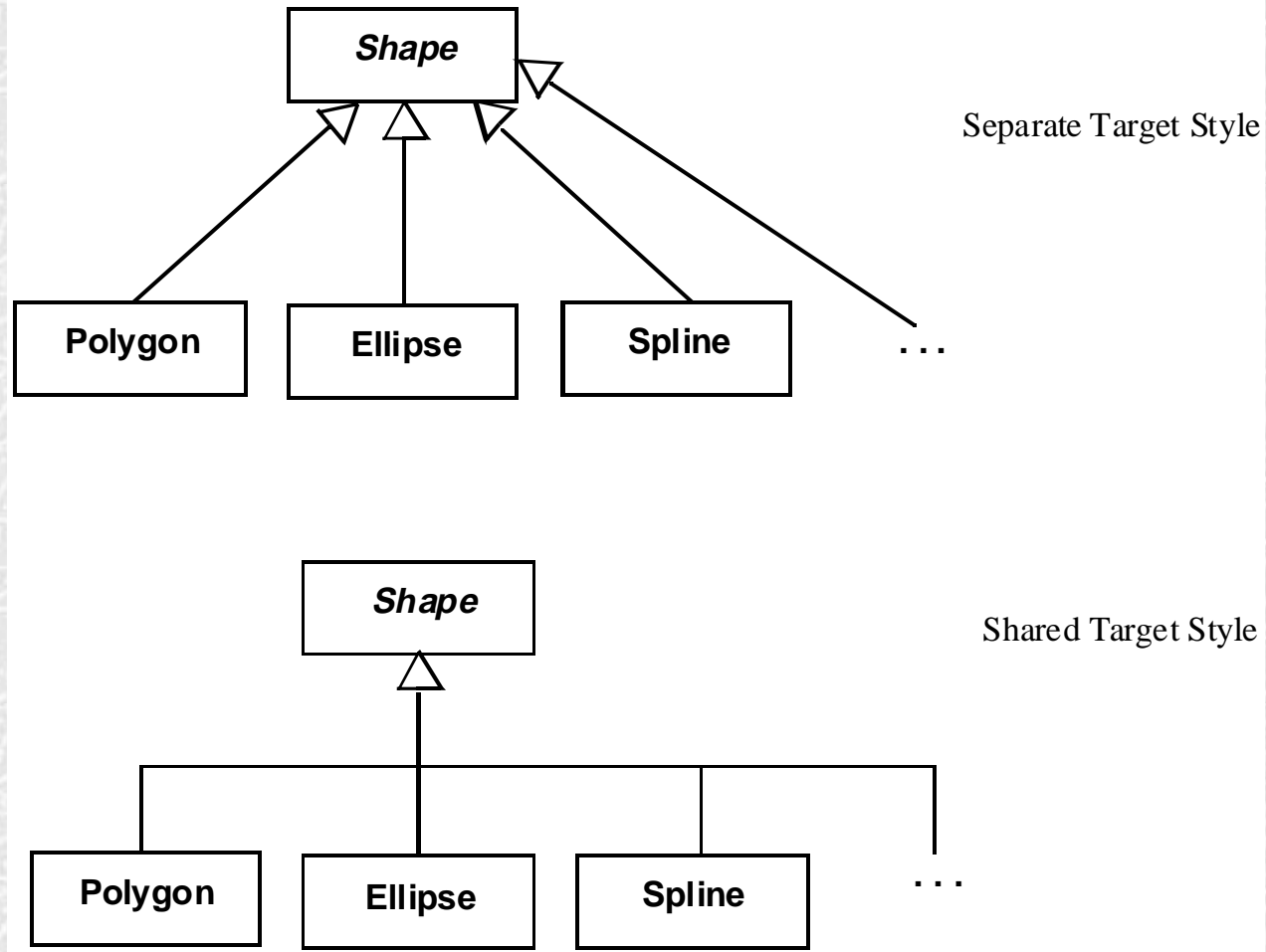


Fig. 3-38, *UML Notation Guide*  
Object Modeling with UML

# Generalization

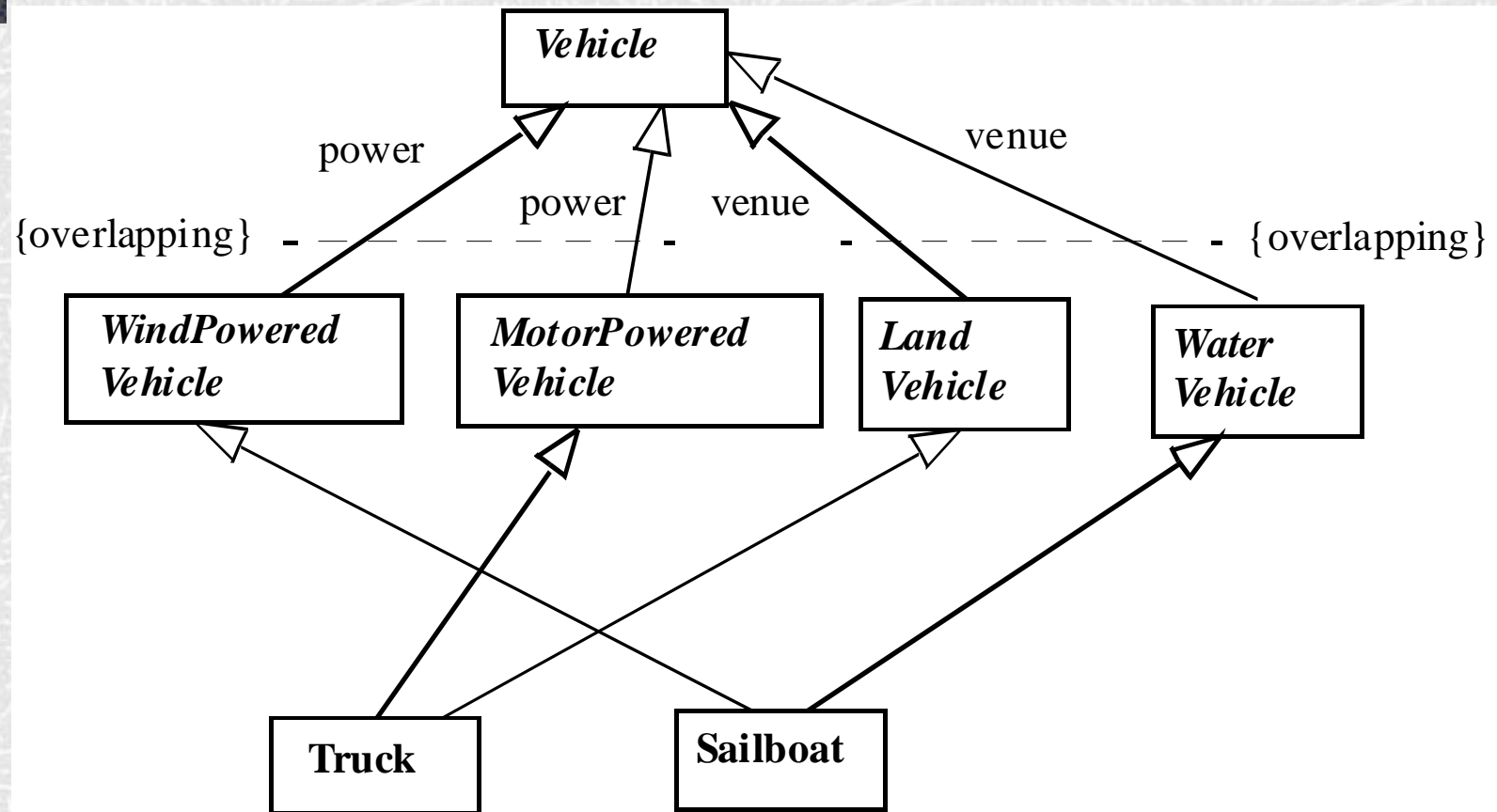


Fig. 3-39, *UML Notation Guide*

# Dependencies

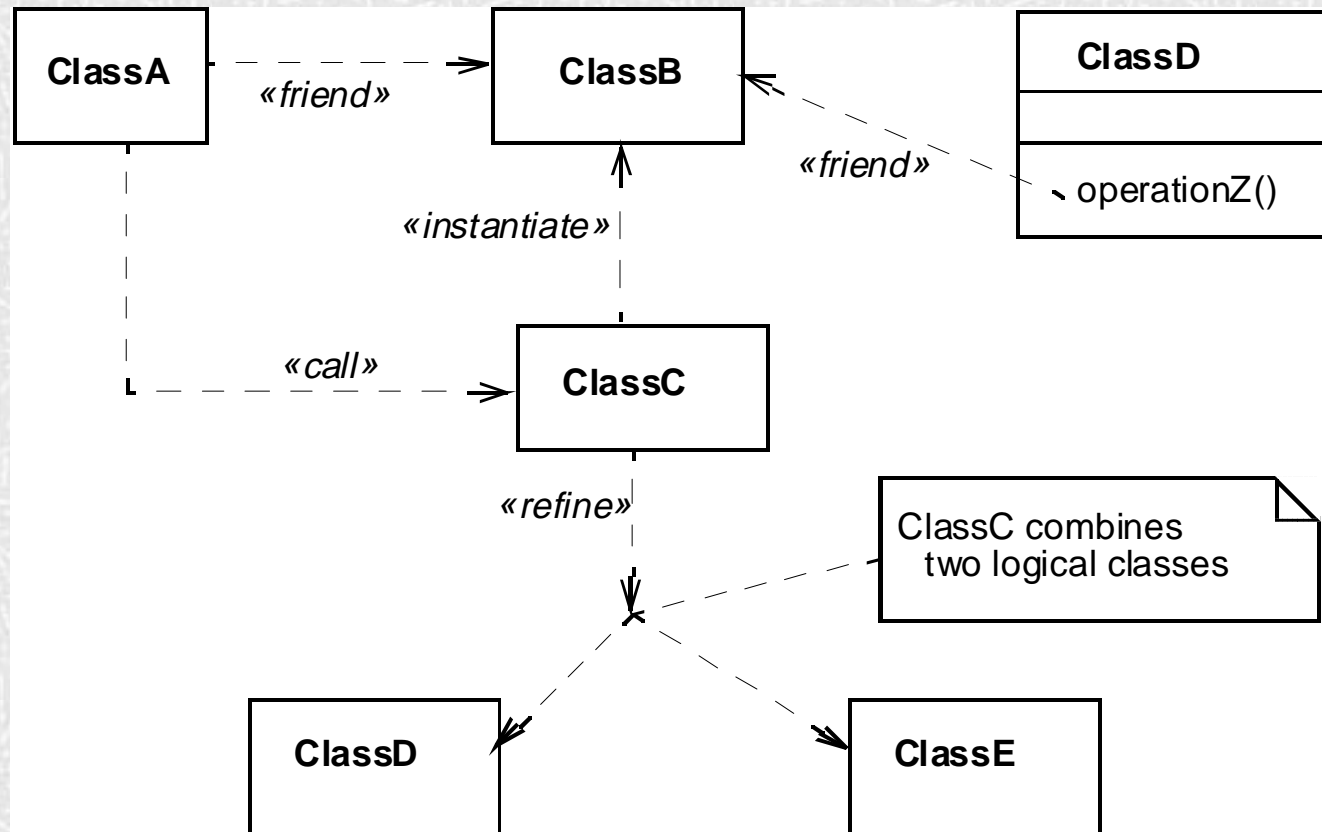


Fig. 3-41, UML Notation Guide



# Dependencies

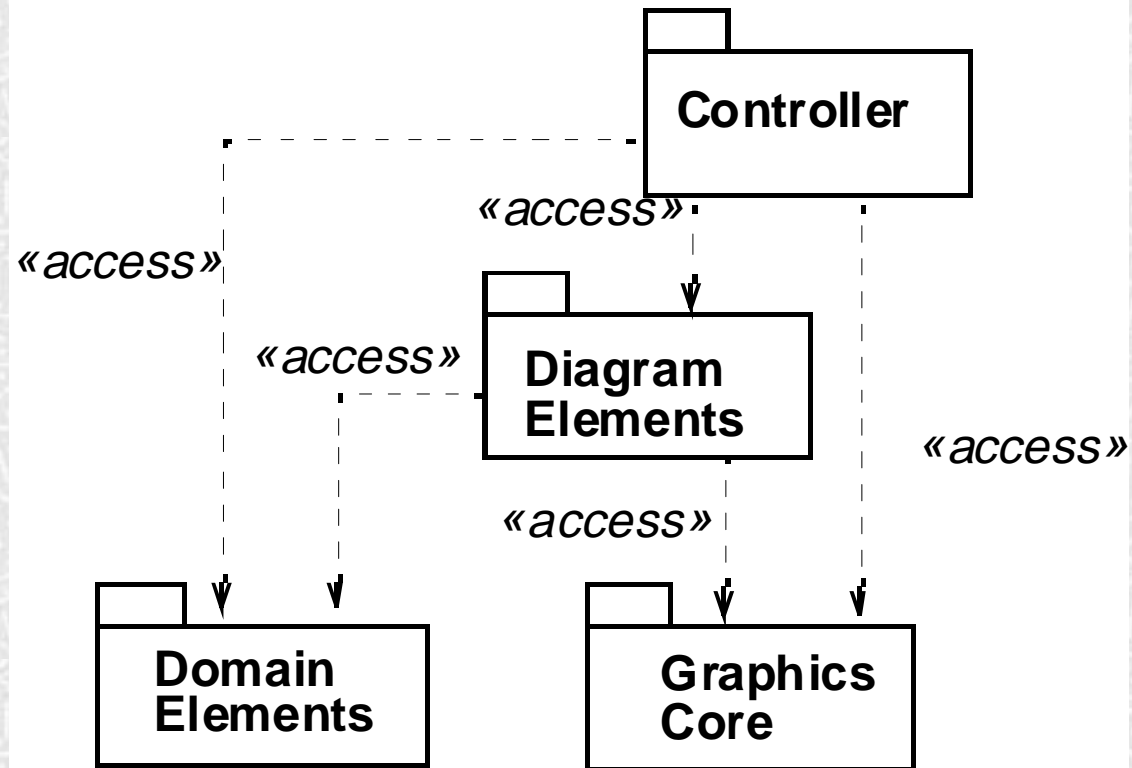


Fig. 3-42, *UML Notation Guide*

# Objects

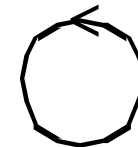
triangle : Polygon

center = (0,0)  
vertices = ((0,0),(4,0),(4,3))  
borderColor = black  
fillColor = white

triangle: Polygon

triangle

:Polygon



scheduler

Fig. 3-29, *UML Notation Guide*

# Composite objects

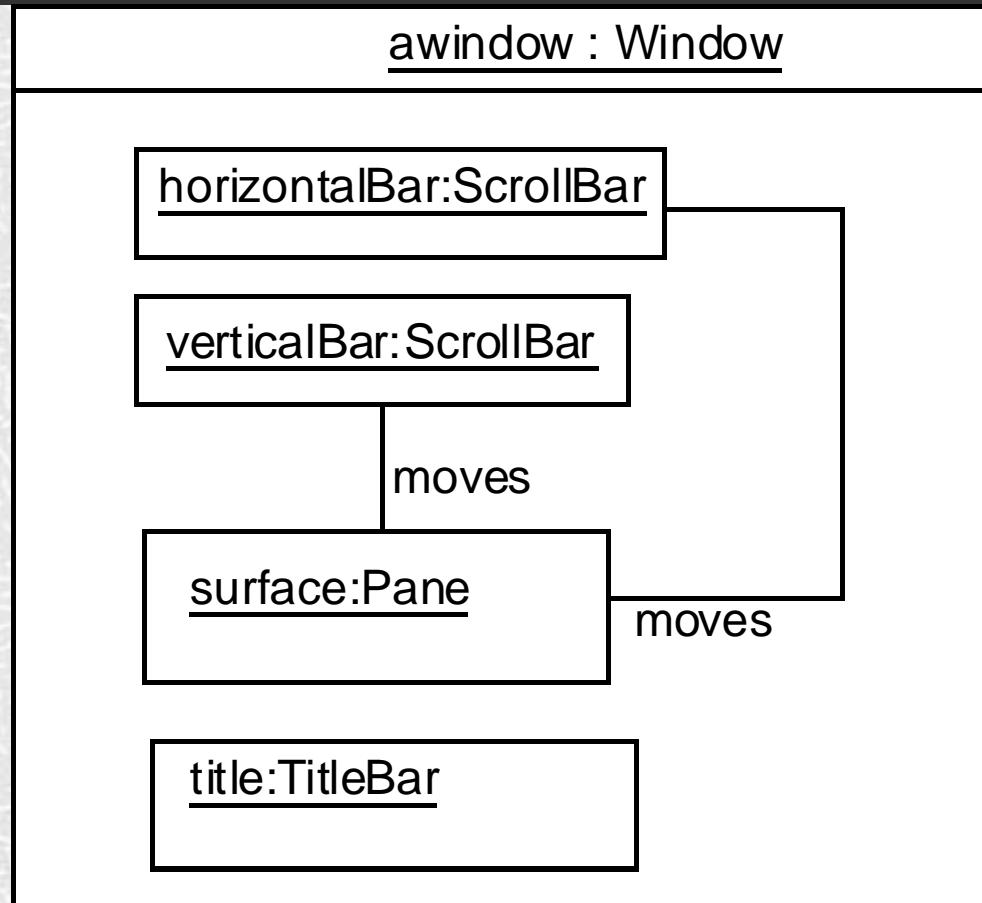


Fig. 3-30, *UML Notation Guide*



# Links

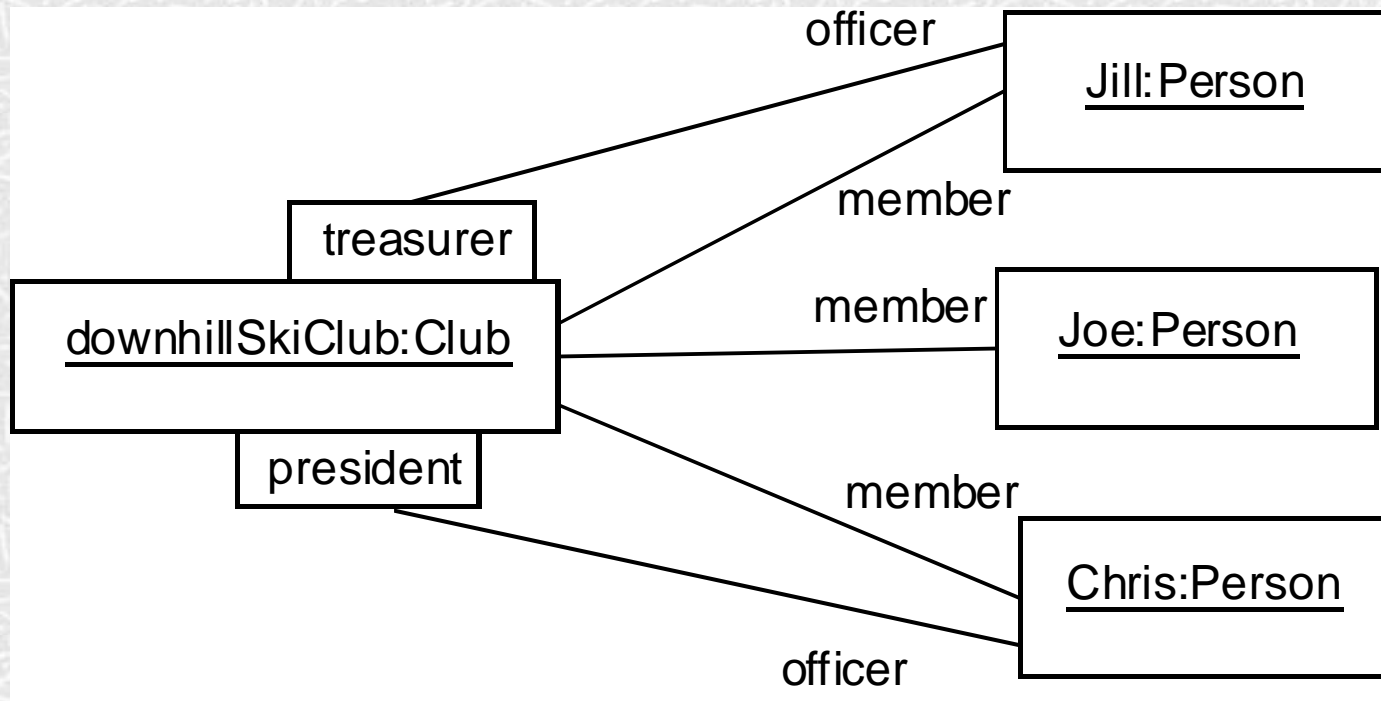


Fig. 3-37, *UML Notation Guide*

# Constraints and Comments

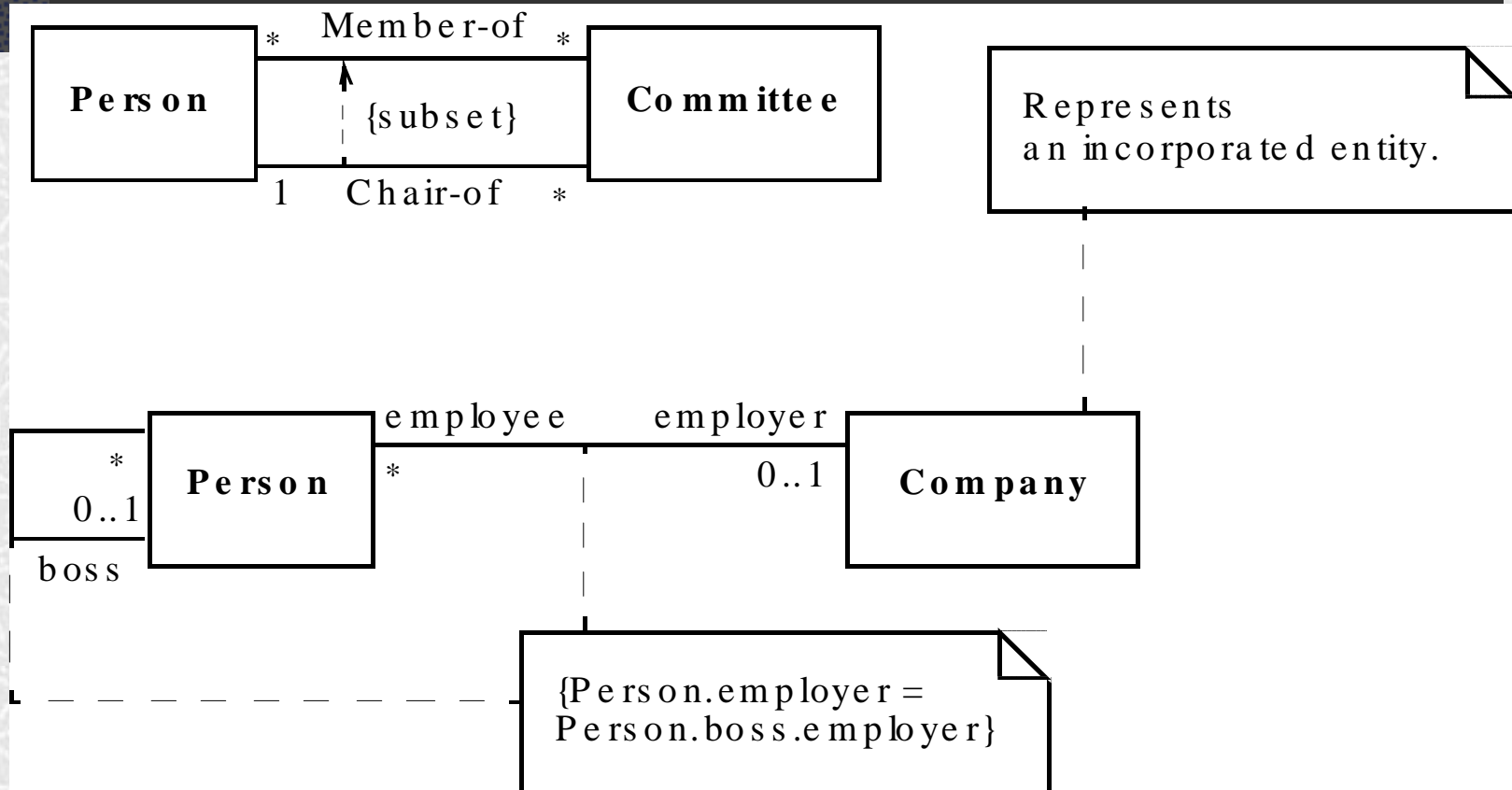


Fig. 3-15, *UML Notation Guide*

# Implementation Diagrams

---

- Show aspects of model implementation, including source code structure and run-time implementation structure
- Kinds
  - component diagram
  - deployment diagram



# Component Diagram

---

- Shows the dependencies among software components
- Components include
  - source code components
  - binary code components
  - executable components

# Components

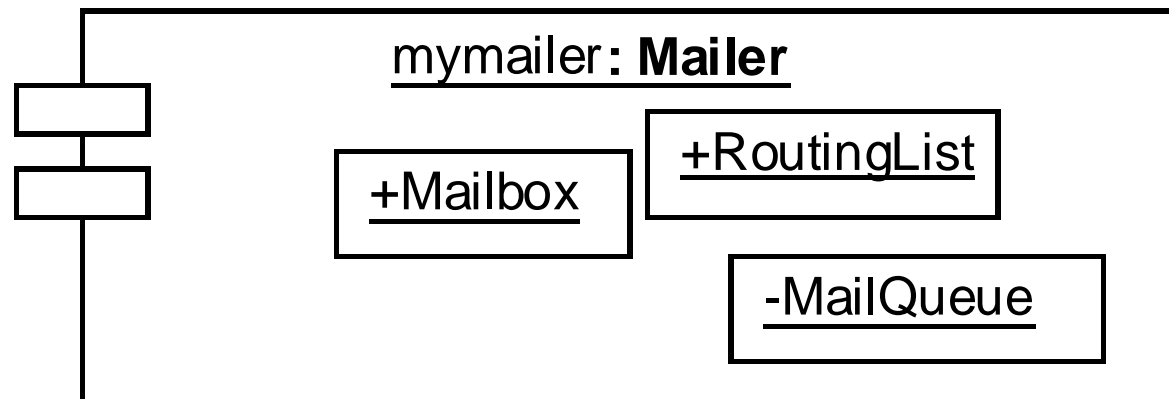
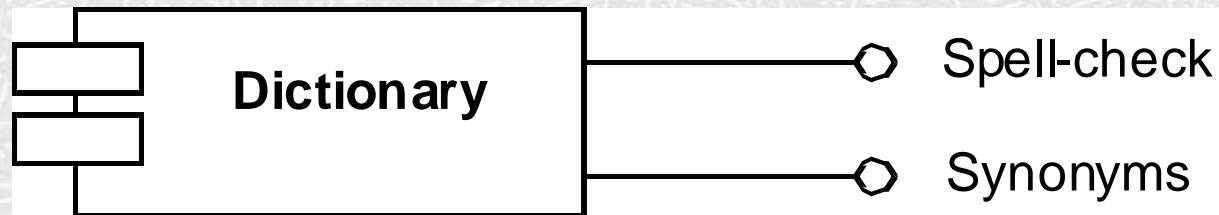


Fig. 3-84, *UML Notation Guide*

# Component Diagram

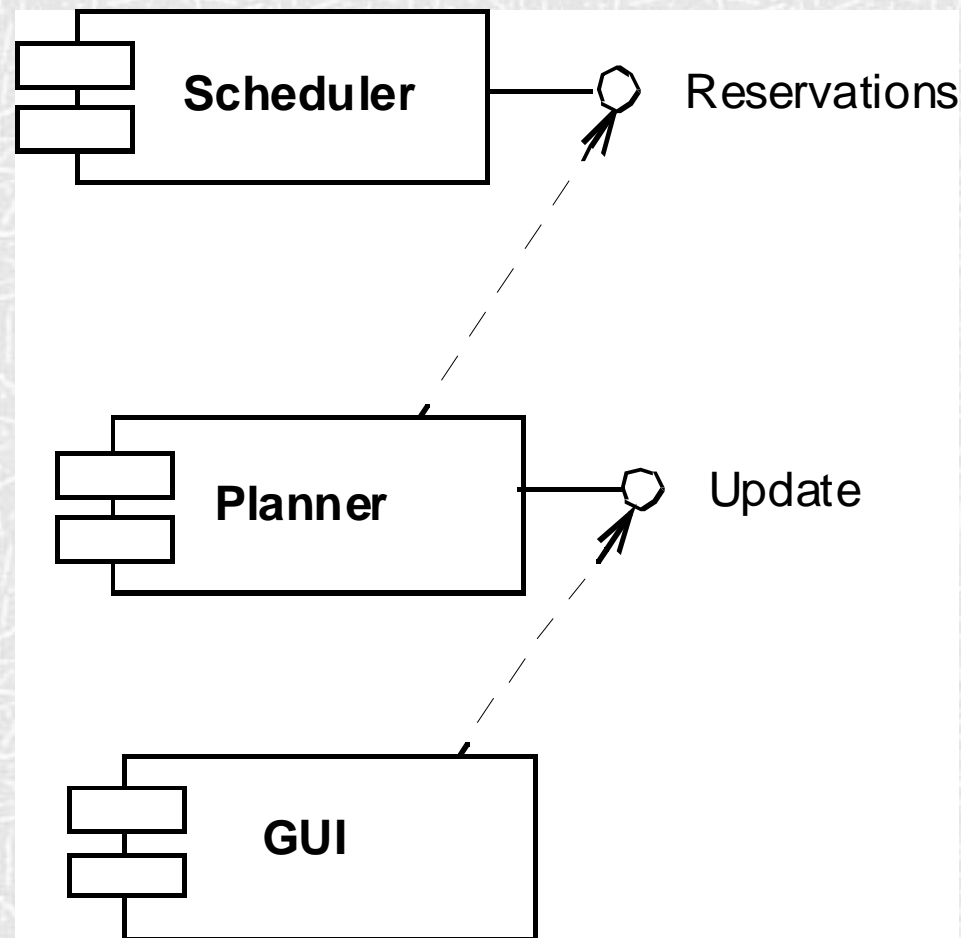


Fig. 3-81, *UML Notation Guide*

Object Modeling with UML



# Deployment Diagram

---

- Shows the configuration of run-time processing elements and the software components, processes and objects that live on them
- Deployment diagrams may be used to show which components may run on which nodes

# Deployment Diagram

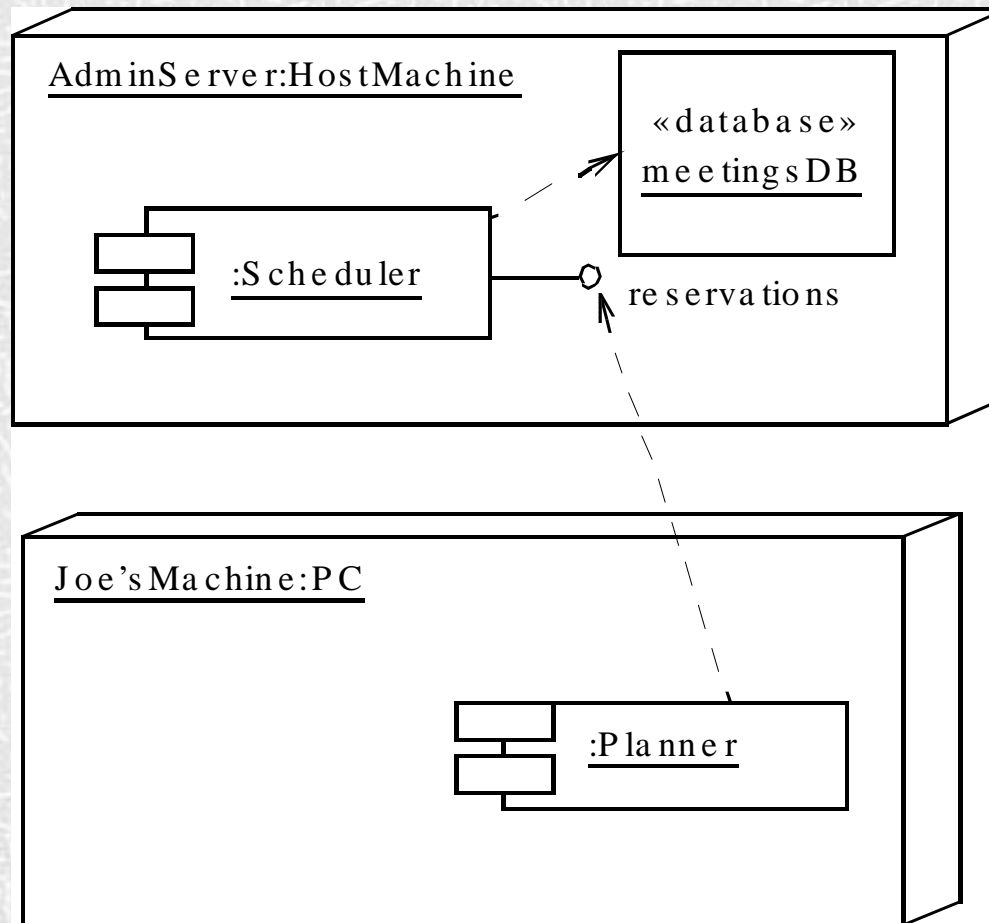


Fig. 3-82, *UML Notation Guide*

# Deployment Diagram (cont'd)

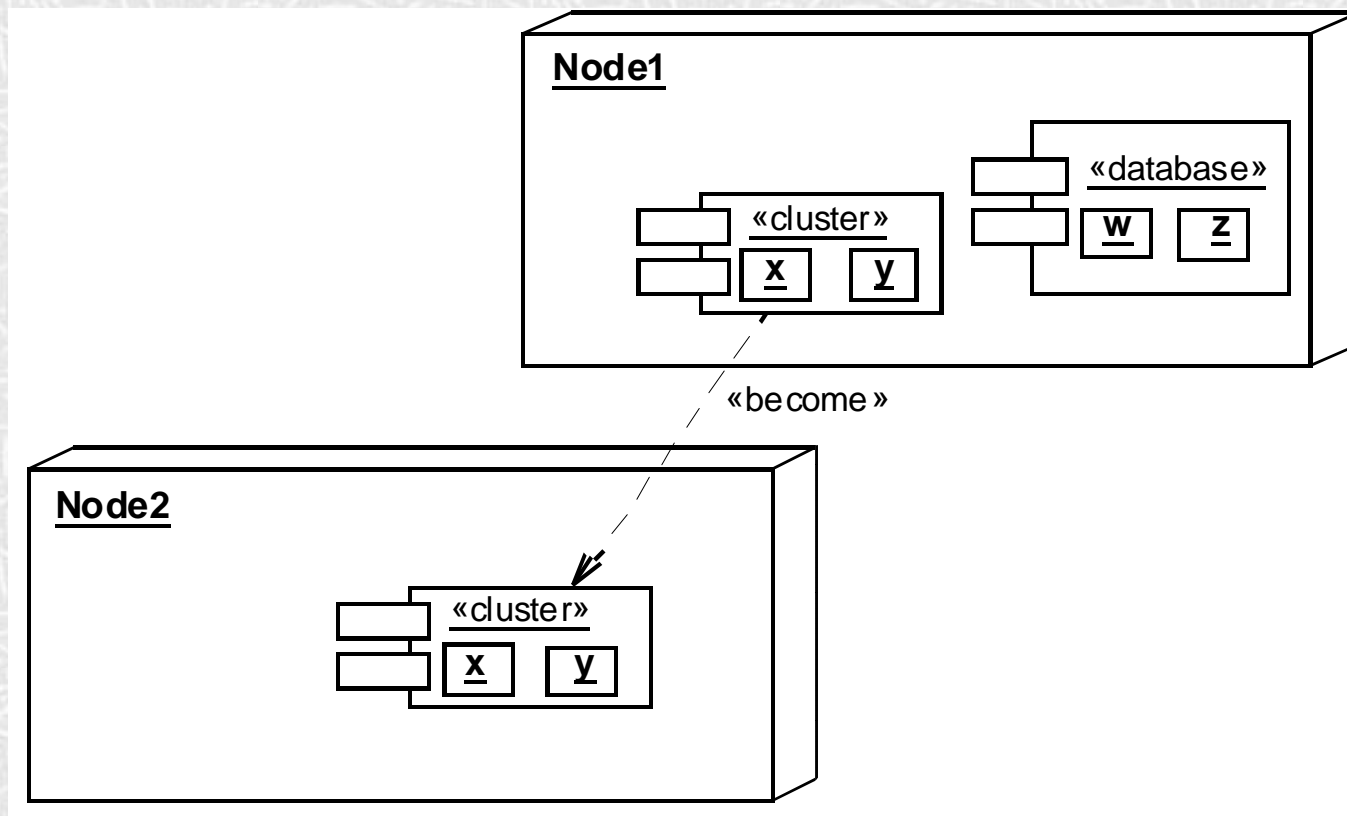


Fig. 3-83, *UML Notation Guide*



# When to model structure

---

- Adopt an opportunistic top-down+bottom-up approach to modeling structure
  - Specify the top-level structure using “architecturally significant” classifiers and model management constructs (packages, models, subsystems; see Tutorial 3)
  - Specify lower-level structure as you discover detail re classifiers and relationships
- If you understand your domain well you can frequently start with structural modeling; otherwise
  - If you start with use case modeling (as with a use-case driven method) make sure that your structural model is consistent with your use cases
  - If you start with role modeling (as with a collaboration-driven method) make sure that your structural model is consistent with your collaborations

# Structural Modeling Tips

---

- Define a “skeleton” (or “backbone”) that can be extended and refined as you learn more about your domain.
- Focus on using basic constructs well; add advanced constructs and/or notation only as required.
- Defer implementation concerns until late in the modeling process.
- Structural diagrams should
  - emphasize a particular aspect of the structural model
  - contain classifiers at the same level of abstraction
- Large numbers of classifiers should be organized into packages (see Lecture 3)



# Example: Interface-based design

```
module POS
{
    typedef long    POSId;
    typedef string Barcode;

    interface InputMedia
    {
        typedef string OperatorCmd;
        void          BarcodeInput(in Barcode Item);
        void          KeypadInput(in OperatorCmd Cmd);
    };

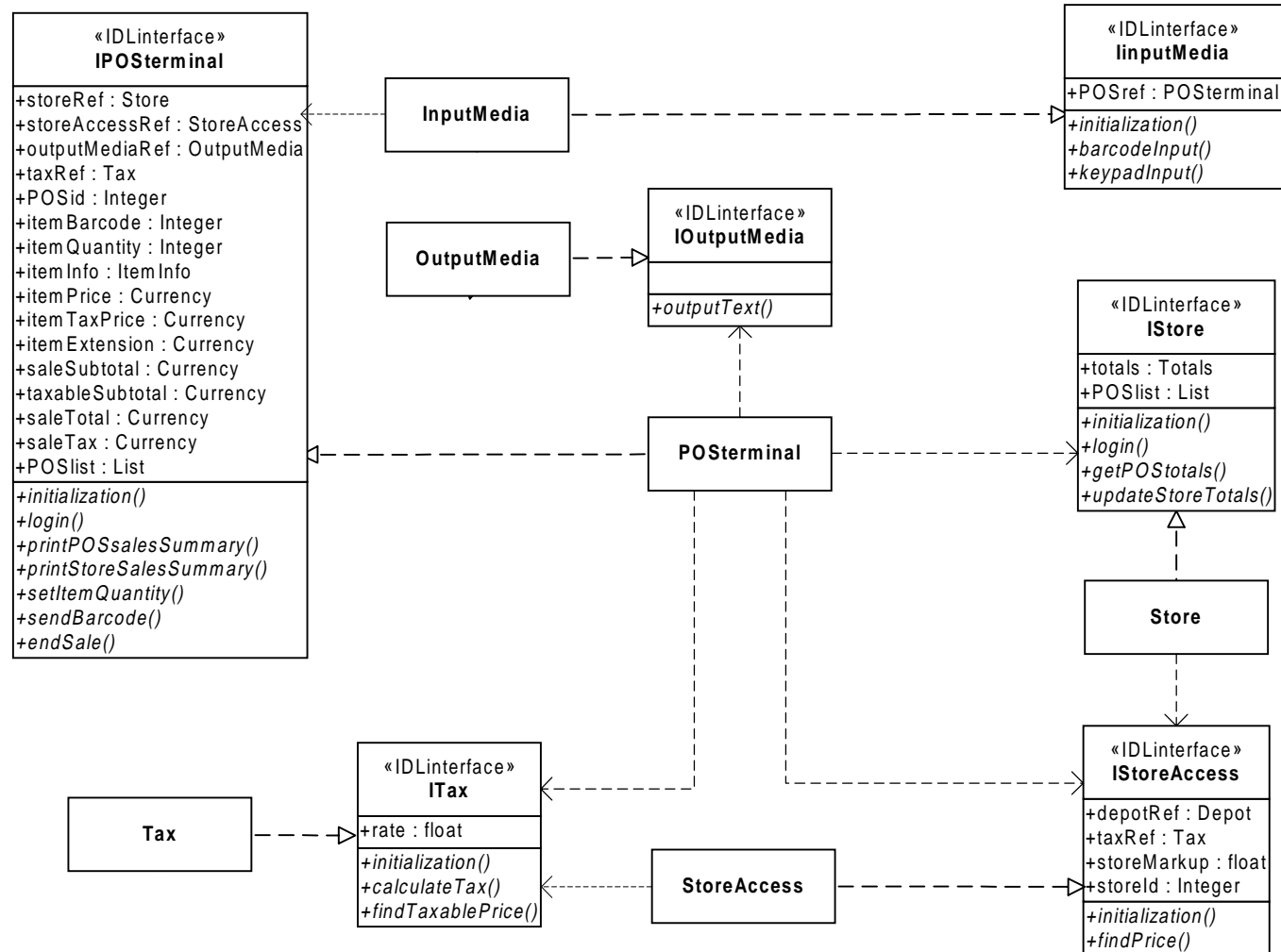
    interface OutputMedia
    { .... }
```

Ch. 23, *CORBA Fundamentals and Programming* (1<sup>st</sup> ed.), [Siegel 96]



Chapter to appear, *CORBA Fundamentals and Programming* (2d ed.), [Siegel 00]

Point-of-Sale



# Use Case Modeling

---

- What is use case modeling?
- Core concepts
- Diagram tour
- When to model use cases
- Modeling tips
- Example: Online HR System

# What is use case modeling?

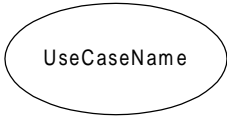
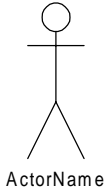
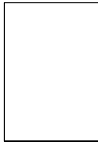
---

- use case model: a view of a system that emphasizes the behavior as it appears to outside users. A use case model partitions system functionality into transactions ('use cases') that are meaningful to users ('actors').






# *Use Case Modeling:*


## Core Elements

Construct	Description	Syntax
<b>use case</b>	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
<b>actor</b>	A coherent set of roles that users of use cases play when interacting with these use cases.	
<b>system boundary</b>	Represents the boundary between the physical system and the actors who interact with the physical system.	

# Use Case Modeling: Core Relationships

Construct	Description	Syntax
<b>association</b>	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
<b>extend</b>	A relationship from an <i>extension</i> use case to a <i>base</i> use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	
<b>generalization</b>	A taxonomic relationship between a more general use case and a more specific use case.	

# Use Case Modeling: Core Relationships (cont'd)

Construct	Description	Syntax
<b>include</b>	An relationship from a <i>base</i> use case to an <i>inclusion</i> use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	<code>&lt;&lt;include&gt;&gt;</code> 



# Use Case Diagram Tour

---

- Shows use cases, actor and their relationships
- Use case internals can be specified by text and/or interaction diagrams (see Lecture 2)
- Kinds
  - use case diagram
  - use case description

# Use Case Diagram

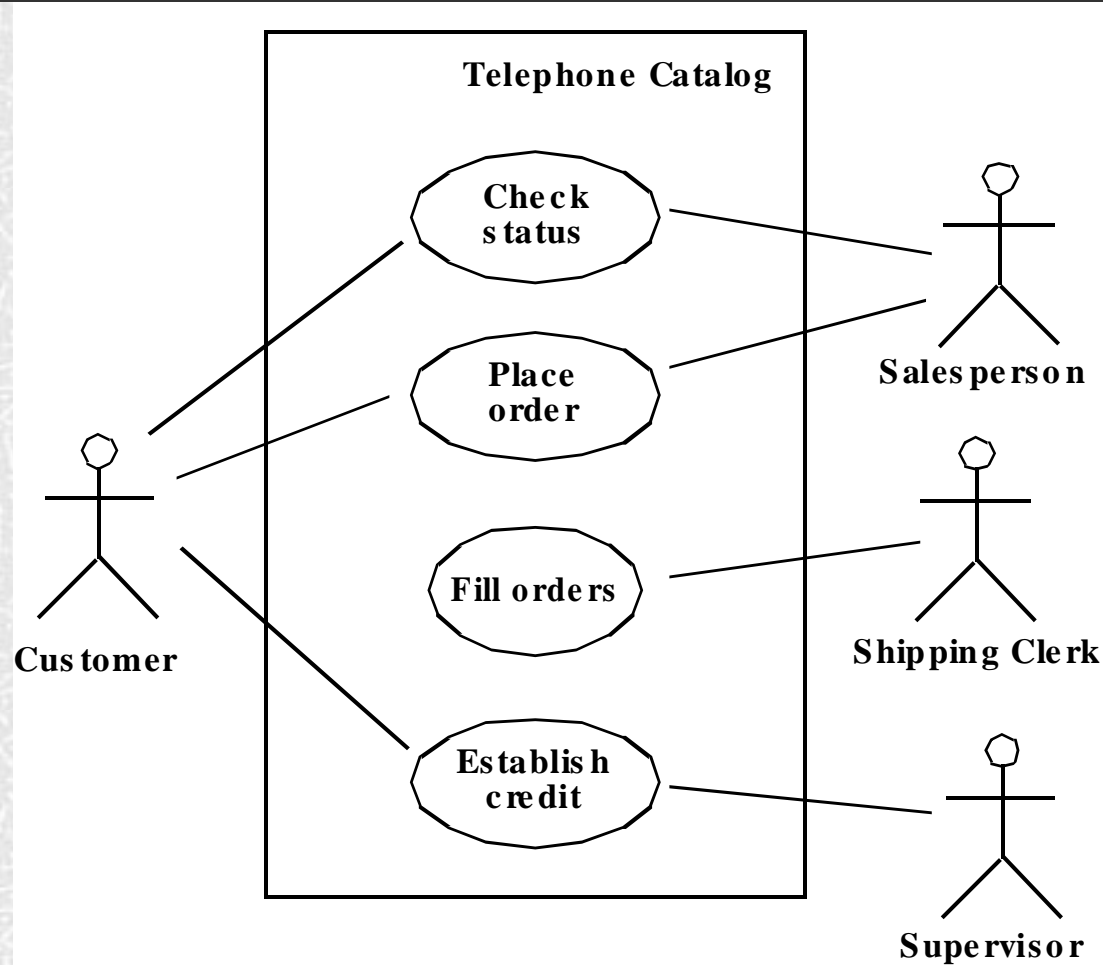


Fig. 3-44, *UML Notation Guide*  
Object Modeling with UML

# Use Case Relationships

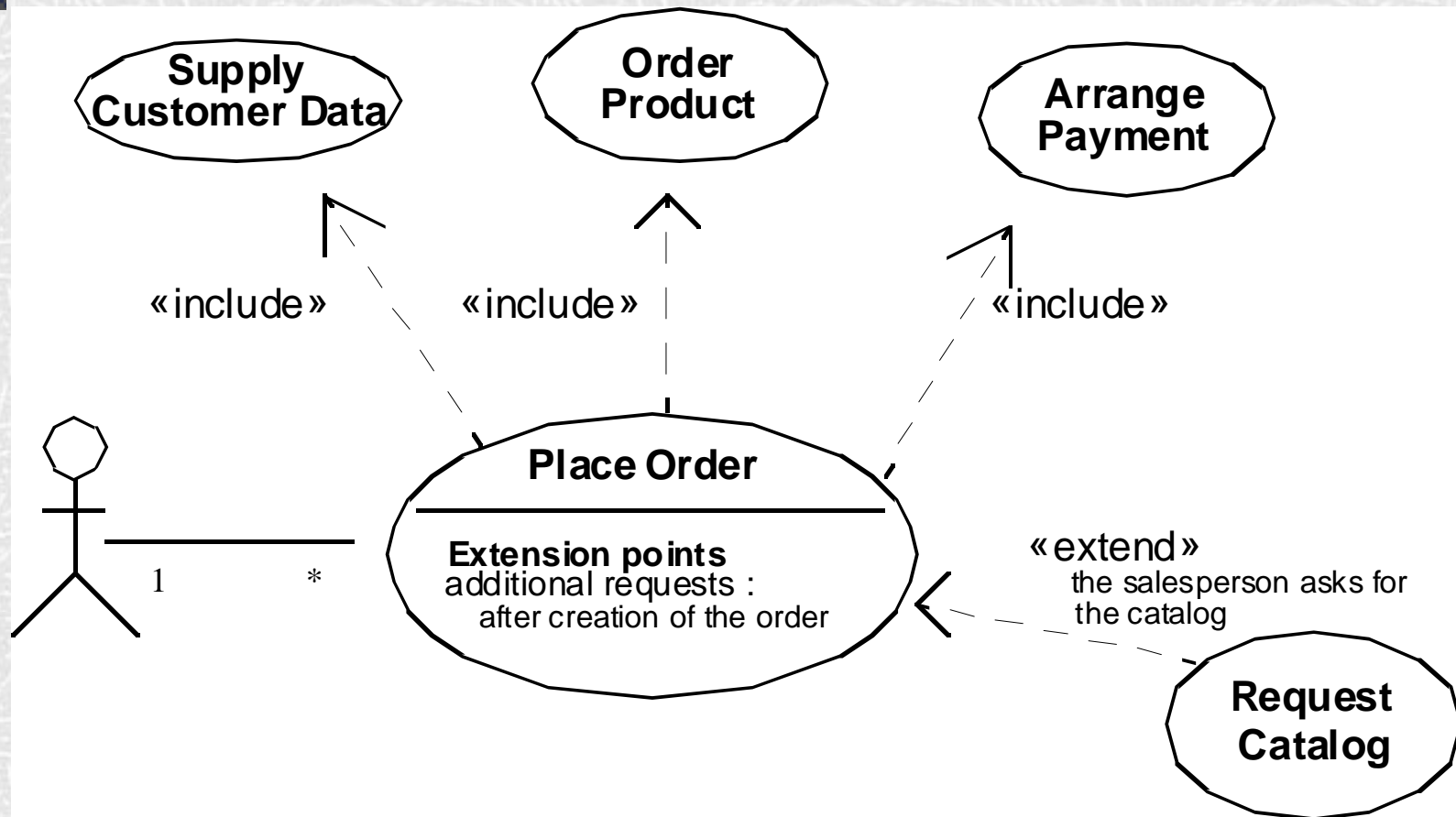


Fig. 3-45, *UML Notation Guide*  
Object Modeling with UML



# Actor Relationships

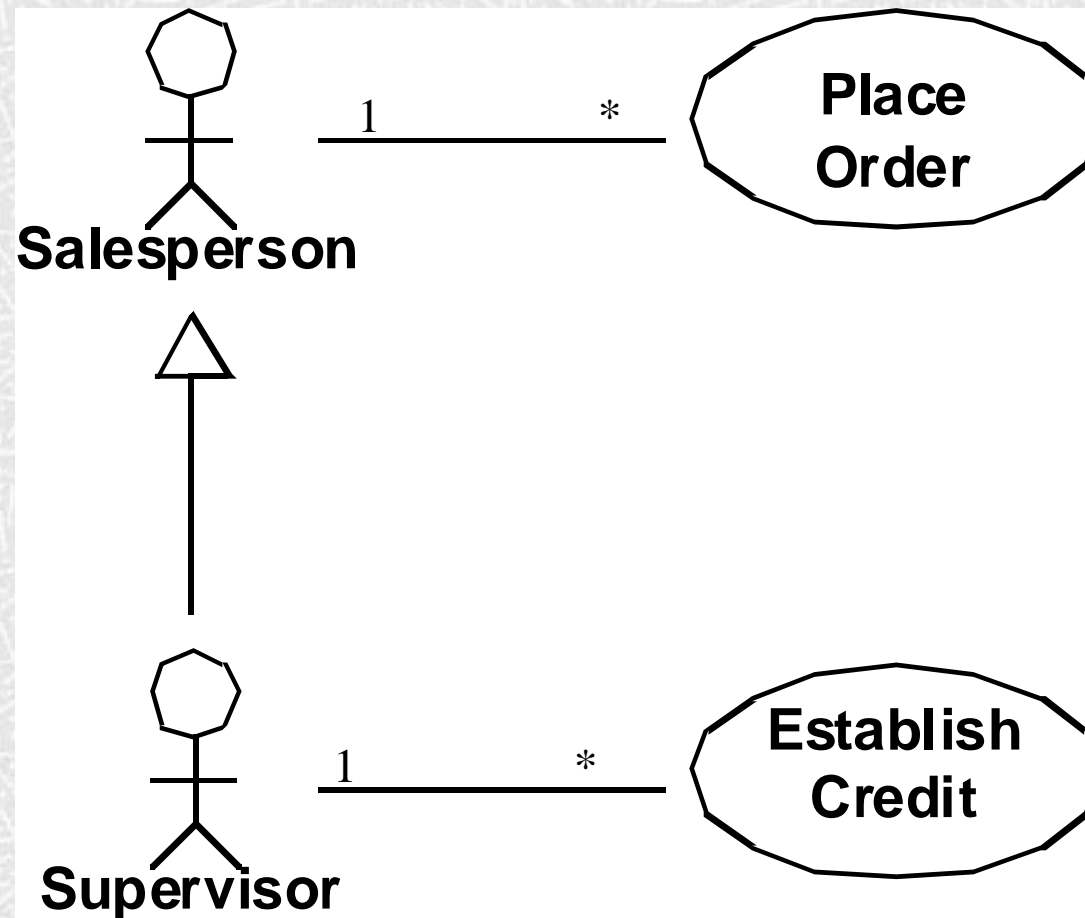


Fig. 3-46, *UML Notation Guide*  
Object Modeling with UML

# Use Case Description: Change Flight Itinerary

■ **Actors:** traveler, client account db, airline reservation system

■ **Preconditions:**

- Traveler has logged on to the system and selected 'change flight itinerary' option

■ **Basic course**

- System retrieves traveler's account and flight itinerary from client account database
- System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
- System asks traveler for new departure and destination information; traveler provides information.
- If flights are available then
- ...
- System displays transaction summary.

■ **Alternative courses**

- If no flights are available then ...

# When to model use cases

---

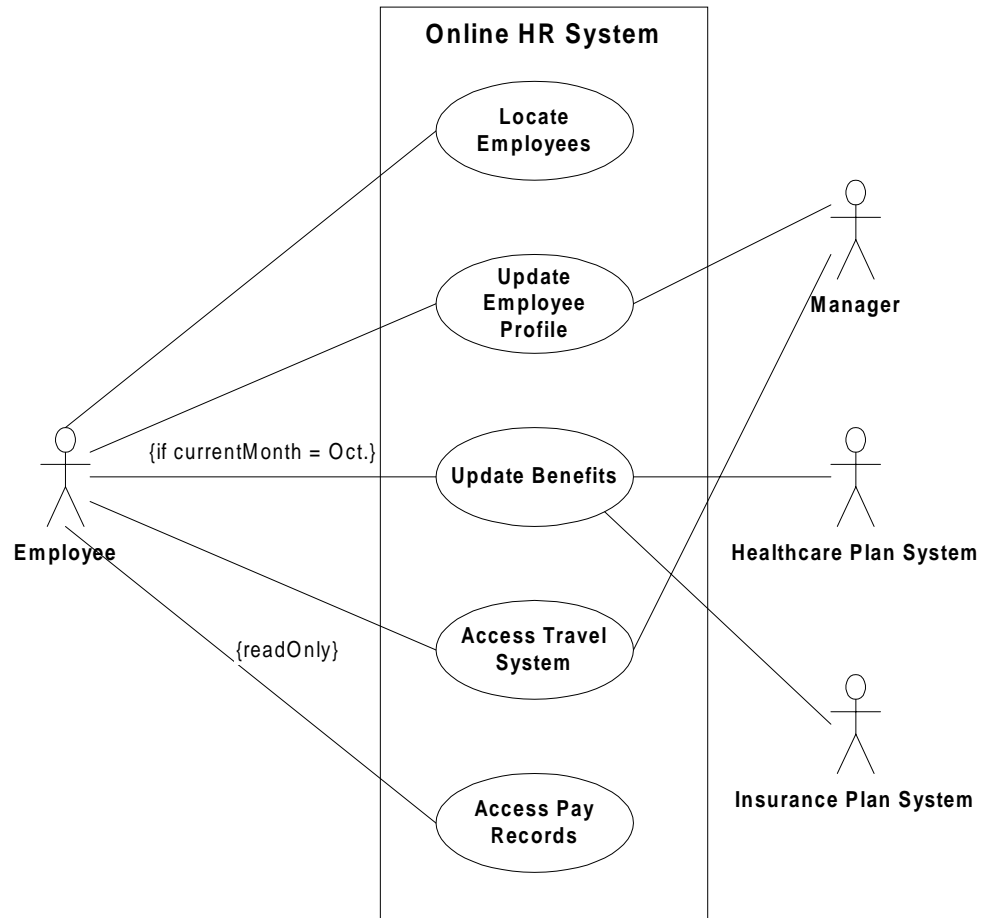
- Model user requirements with use cases.
- Model test scenarios with use cases.
- If you are using a use-case driven method
  - start with use cases and derive your structural and behavioral models from it.
- If you are not using a use-case driven method
  - make sure that your use cases are consistent with your structural and behavioral models.



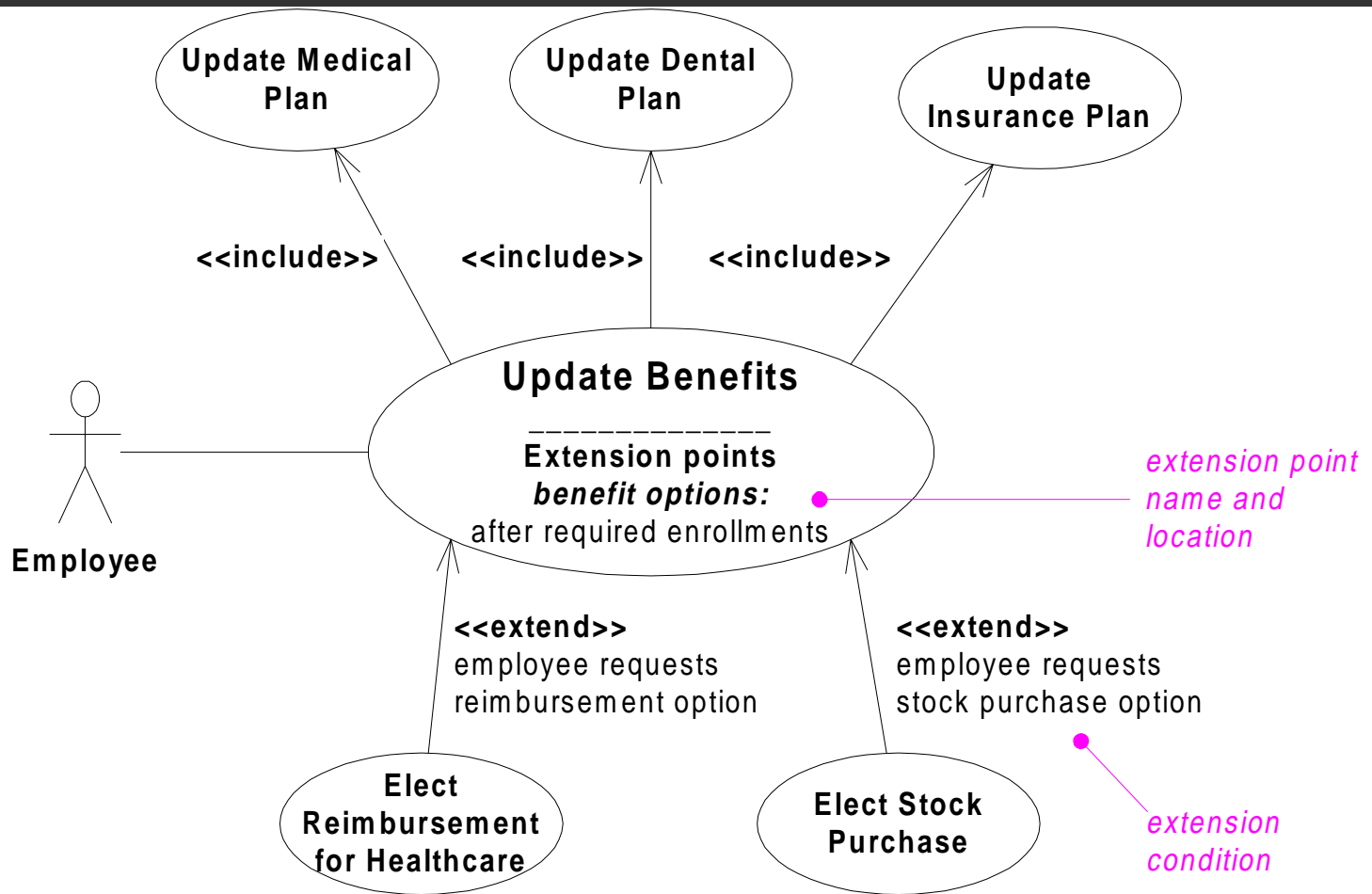
# Use Case Modeling Tips

- Make sure that each use case describes a significant chunk of system usage that is understandable by both domain experts and programmers
- When defining use cases in text, use nouns and verbs accurately and consistently to help derive objects and messages for interaction diagrams (see Lecture 2)
- Factor out common usages that are required by multiple use cases
  - If the usage is required use <<include>>
  - If the base use case is complete and the usage may be optional, consider use <<extend>>
- A use case diagram should
  - contain only use cases at the same level of abstraction
  - include only actors who are required
- Large numbers of use cases should be organized into packages (see Lecture 3)

# Example: Online HR System



# Online HR System: Use Case Relationships





# Online HR System: Update Benefits Use Case

■ **Actors:** employee, employee account db, healthcare plan system, insurance plan system

■ **Preconditions:**

- Employee has logged on to the system and selected 'update benefits' option

■ **Basic course**

- System retrieves employee account from employee account db
- System asks employee to select medical plan type; **include** Update Medical Plan.
- System asks employee to select dental plan type; **include** Update Dental Plan.
- ...

■ **Alternative courses**

- If health plan is not available in the employee's area the employee is informed and asked to select another plan...

# Wrap Up

---

- Ideas to take away
- Preview of next tutorial
- References
- Further info



# Ideas to Take Away

- UML is effective for modeling large, complex software systems
- It is simple to learn for most developers, but provides advanced features for expert analysts, designers and architects
- It can specify systems in an implementation-independent manner
- 10-20% of the constructs are used 80-90% of the time
- Structural modeling specifies a skeleton that can be refined and extended with additional structure and behavior
- Use case modeling specifies the functional requirements of system in an object-oriented manner



# Preview - Next Tutorial

---

- Behavioral Modeling with UML
  - Behavioral modeling overview
  - Interactions
  - Collaborations
  - Statecharts
  - Activity Graphs

# References

---

- *OMG UML Specification v. 1.3*, OMG doc# ad/06-08-99
- [Kobryn 00] *UML 2001: A Standardization Odyssey*, Communications of the ACM, Oct. 1999.
- [Siegel 96] *CORBA Fundamentals and Programming*, Wiley, 1996.
- [Kobryn 00] Chapter to appear in [Siegel 00] *CORBA Fundamentals and Programming* (2<sup>nd</sup> ed.), Wiley, 2000.

# Further Info

---

- web:
  - [uml.shl.com](http://uml.shl.com)
  - [www.omg.org](http://www.omg.org)
- email
  - [uml-rtf@omg.org](mailto:uml-rtf@omg.org)
  - [ckobryn@acm.org](mailto:ckobryn@acm.org)
- conferences & workshops
  - UML World 2000, NYC, March '00
  - UML '00, York, England, Oct. '00