Subject: Object Oriented Analysis and Design

Module Number- 2: Basic Structural Modelling

SME NAME: ENTER NAME
SUBMISSION DATE: MENTION DATE
VERSION CODE: TO BE FILLED AT HO
RELEASED DATE: TO BE FILLED AT HO

Basic Structural Modeling

Classes and Objects

- Object-oriented modelers use classes, objects, and their relationships to describe things and how they work.
- Classes and objects describe the elements in the system, while the relationships reveal communication and interaction.
- An object exists in the real world. It can be a part of any type of system, for example, a machine, an organization, or a business.
- Some objects tend toward the theoretical (such as implementation of objects in a software system): You can derive them by analyzing the structure and behavior of objects in the real world.

Basic Structural Modeling

- A class describes the properties and behavior of a type of object.
- All objects can reflect some class. From a class, a system will create individual instances, or objects. Put another way, objects are instantiated from a class.
- We use classes to discuss systems and to classify the objects you identify in the real world.

Basic Structural Modeling

- An object-oriented model of any system—business, information machines, or anything—relies on real concepts from the problem domain, making the models understandable and easy to communicate.
- If you build a system for an insurance company, it should reflect the concepts in the insurance business.
- If you build a system for the military, the concepts from that world should be used to model the system.
- With a clearly modeled understanding of the primary concepts of a business, the designer can also adjust to change.

Basic Structural Modeling

- In UML, a class is the most common classifier. UML includes a large number of other classifiers, such as use cases, signals, activities, or components.
- A class is the most general of classifiers as it can describe an object in any type of system—information, technical, embedded real-time, distributed, software, and business.
- Artifacts in a business, information that should be stored or analyzed, and roles that the actors in the business play often turn into classes in information and business systems.
- Examples of classes in business and information systems include:

  Customer; Agreement; Invoice; Debt; Asset; Quotation

Basic Structural Modeling

Class Diagram

- A class diagram describes the static view of a system in terms of classes and relationships among the classes.
- Classes not only show the structure of your information but also describe behavior.
- To create a class diagram, you need to identify and describe the main items in the system.
- After the definition of a number of classes, the modeler can define the relationships between classes. A class is drawn with a rectangle, often divided into three compartments

  - The name compartment

  - The attributes compartment

  - The operations compartment

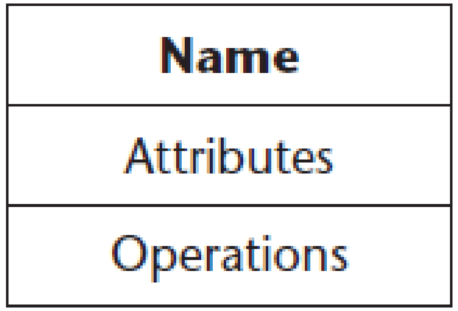


Figure: A class in UML.

Basic Structural Modeling

Finding Classes

- The modeler must use the knowledge of experts in the problem domain along with an understanding of the user and system in a creative endeavor to capture the system.
- Good classes reflect the problem domain and have real names.
- When you are looking for classes, a good use-case model helps tremendously. Rely on the use-case model and requirements specification in the search for classes.
- Pose the following types of questions:

Basic Structural Modeling

- Do I have information that should be stored or analyzed? If there is any information that has to be stored, transformed, analyzed, or handled in some other way, then it is a possible candidate for a class. The information might include concepts that must always be registered in the system or events or transactions that occur at a specific moment.

- Do I have external systems? If so, they are normally of interest when you model. The external system might be seen as classes that your system contains or should interact with.

Basic Structural Modeling

- Do I have any reusable patterns, class libraries, or components? If you have patterns, class libraries, or components from earlier projects, colleagues, or manufacturers, they normally contain class candidates.
- Are there devices that the system must handle? Any technical devices connected to the system turn into class candidates that handle these devices.
- Do I have organizational parts? Representing an organization can be done with classes, especially in business models.
- Which roles do the actors in the business play? These roles can be seen as classes, such as user, system operator, customer, and so on.

Basic Structural Modeling

Name Compartment

- The top compartment of the class rectangle contains the name of the class; it is capitalized and centered in boldface.
- Again, the name should be derived from the problem domain and should be as unambiguous as possible.
- Therefore, it should be a noun, for example, Invoice or Debt. The class name should not have a prefix or a suffix.

Basic Structural Modeling

Attributes Compartment

- Classes have attributes that describe the characteristics of the objects.
- Following Figure shows the class Car with attributes of registration number, data, speed, and direction.
- The correct class attributes capture the information that describes and identifies a specific instance of the class.
- Furthermore, the purpose of the system also influences which attributes should be used.
- Attributes have values in object instances of the class.

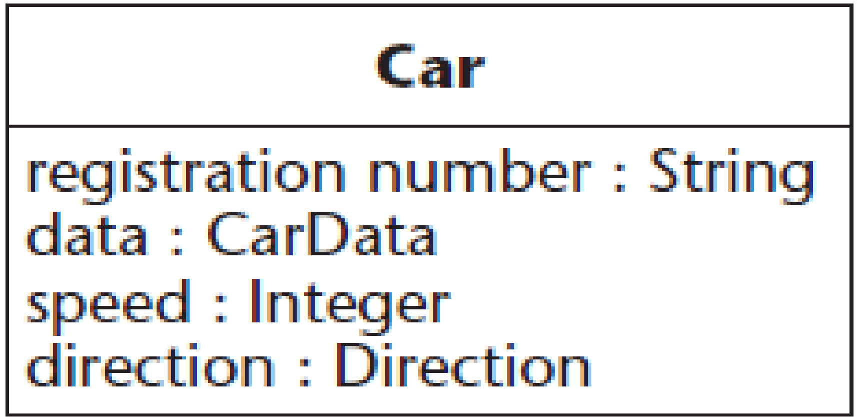


Figure A class, Car, with the attributes registration number, data, speed, and direction. Attribute names typically begin with a lowercase letter.

Basic Structural Modeling

- An attribute has a type, which tells you what kind of attribute it is.
- Typical attribute types are integer, Boolean, string, date, real, point, and enumeration, which are called data types. Their instances are values (not objects).
- The attributes can have different visibility. Visibility describes whether the attribute can be referenced from classes other than the one in which they are defined.

  - If an attribute has the visibility of **public**, it can be used and viewed outside that class.

  - If an attribute has the visibility of **private**, you cannot access it from other classes.

  - If an attribute is **protected** it is private to the class, but visible to any subclasses through generalization and specialization.

Basic Structural Modeling

- Additional kinds of visibility might be defined using a profile, or extension, for a particular programming language, but public and private are normally all that are necessary to express your class diagrams.
- Public is usually expressed with a plus sign (+), private with a minus sign (–), as shown in Figure.
- Protected is generally shown with the pound (#) sign. If no sign is displayed, this means that the visibility is undefined (there is no default visibility).
- Individual tool vendors implementing UML often use their own notation for the different types of visibility.

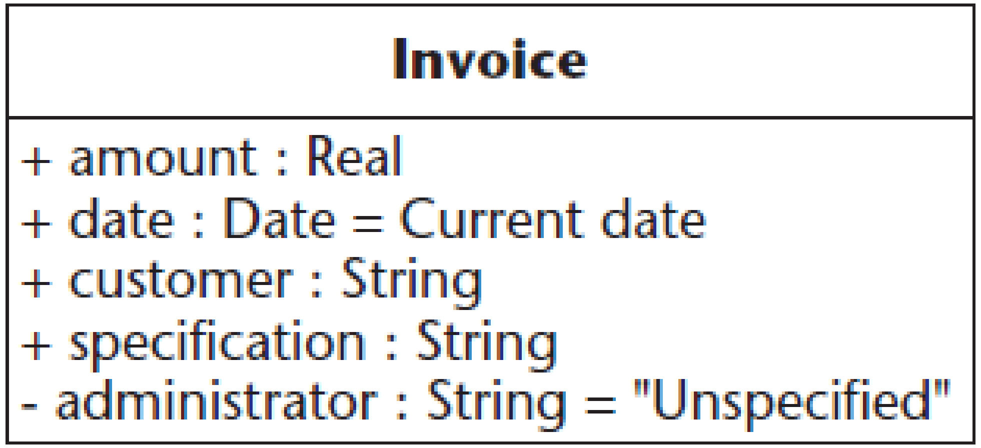

Invoice
+ amount : Real
+ date : Date = Current date
+ customer : String
+ specification : String
- administrator : String = "Unspecified"

Figure: A class with attributes and their default values.

Basic Structural Modeling

- Above Figure shows an attribute with a default value. It’s assigned at the same time an object of the class is created.
- An attribute can also be defined as a class-scope attribute, as shown in the following Figure.
- This means that the attribute is shared between all objects of a certain class (sometimes called a class variable).
- By convention, display the class-scope attribute with an underline.
- UML tools might display class-scoped attributes in different ways, and the meaning of these will differ in models fitting a profile for a specific programming language. For example, in Java a class variable will mean a static variable.

Basic Structural Modeling

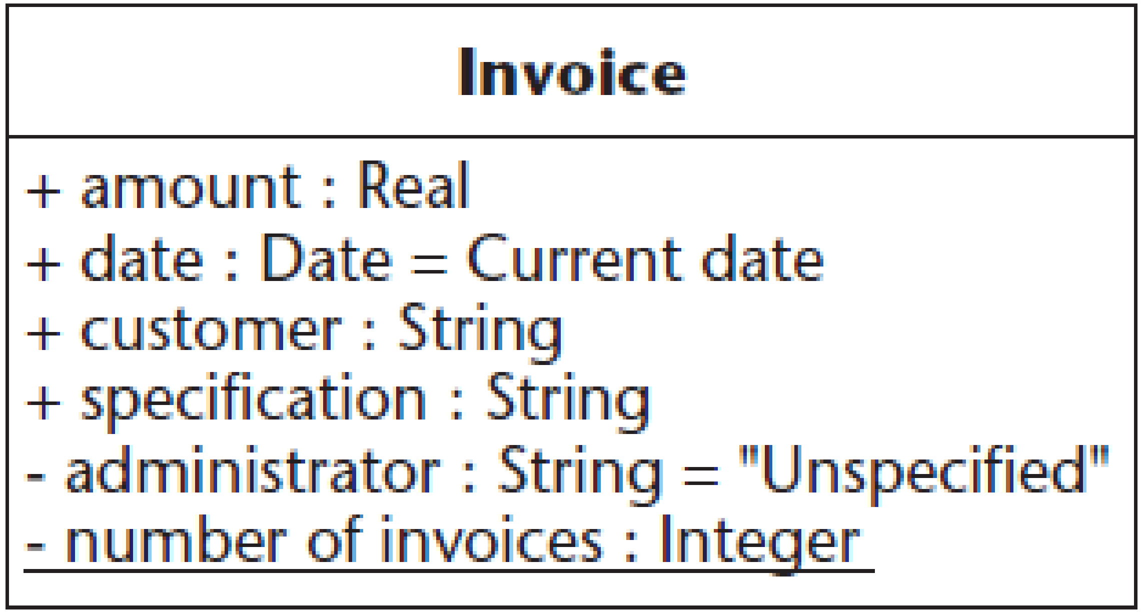

Invoice
+ amount : Real
+ date : Date = Current date
+ customer : String
+ specification : String
- administrator : String = "Unspecified"
- number of invoices : Integer

Figure A class with a class-scope attribute. The attribute number of invoices is used to count the invoices; the value of this attribute is the same in all objects because the attribute is shared between them.

Basic Structural Modeling

- A property-string can be used to further describe an attribute.
- A property string is written within curly braces; it is a comma-separated list of property values that apply to the attribute.
- For example, as shown in following Figure, an attribute can have a property of {readOnly}. Other properties include {ordered}, {sequence} and others.
- The property-string on an attribute shows how, in this case, the UML designers use an extension mechanism in a standard way to enhance the language.

Basic Structural Modeling

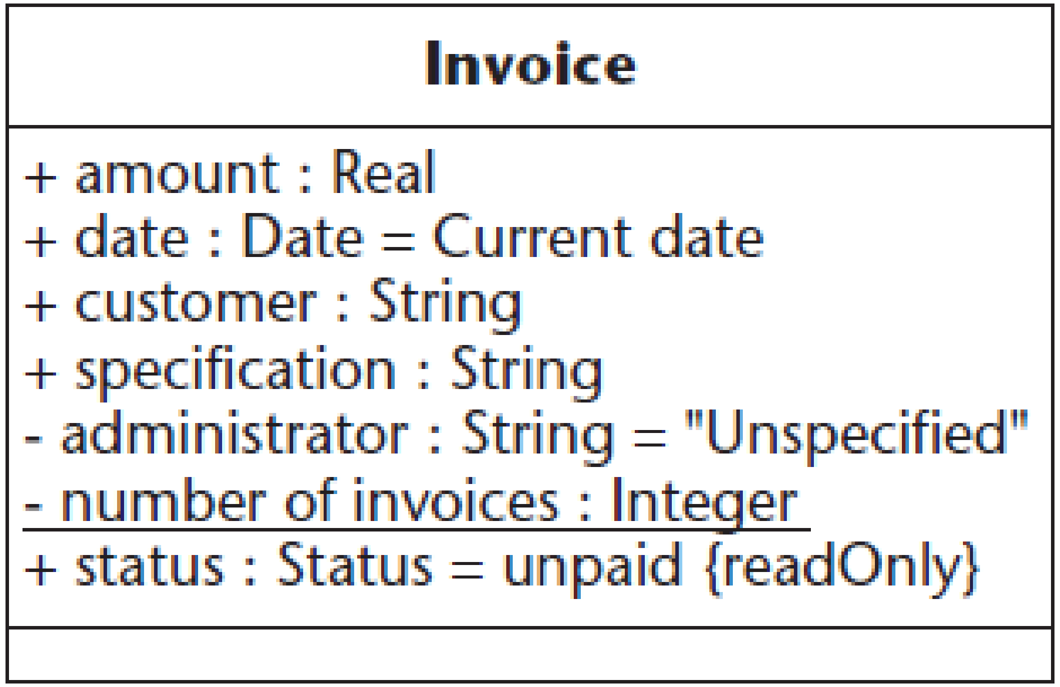

Invoice
+ amount : Real
+ date : Date = Current date
+ customer : String
+ specification : String
- administrator : String = "Unspecified"
- number of invoices : Integer
+ status : Status = unpaid {readOnly}

Figure An attribute with a property-list of {readOnly}.

Basic Structural Modeling

- UML has formal syntax for the description of an attribute:

  visibility / name : type [multiplicity] = default-value { property-string}

- You must have a name, but all other parts are optional.
- A “/” indicates that the attribute is derived. For example, the age of a person might be derived as the current date minus the date of birth.
- Multiplicity shows the number of instances of the attribute in square brackets (for example, [0..1]) and can be omitted for multiplicities of exactly one.
- The property-string can be used to specify other information about the attribute, such as that the attribute should be persistent.

Basic Structural Modeling

Java Implementation

- Translating the model class into code requires specific details, as shown in the following Figure and represented in the code.

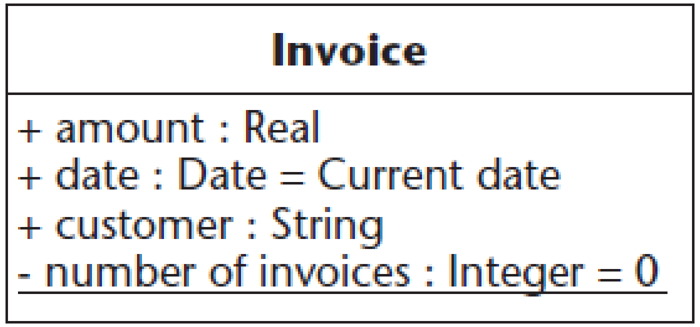

Invoice
+ amount : Real
+ date : Date = Current date
+ customer : String
- number of invoices : Integer = 0

Figure: An Invoice class.

```
public class Invoice
{
    public double amount;
    public Date date = new Date();
    public String customer;
    static private int number_of_invoices = 0;

    // Constructor, called every time an objects is created
    public Invoice ()
    {
        // Other initialization

        number_of_invoices++; // Increment the class attribute
    }
    // Other methods go here
} ;
```

Basic Structural Modeling

Operations Compartment

- Following Figure demonstrates that a class has both attributes and operations.
- Operations manipulate the attributes or perform other actions. Operations are normally called functions.
- An operation is described with a return-type, a name, and zero or more parameters.
- Together, the return-type, name, and parameters are the signature of the operation. The signature describes everything needed for the operation.
- The operations in a class describe what the class can do (not how), that is, what services it offers; thus they could be seen as the interface to the class.
- Just like an attribute, an operation can have visibility and scope.

Basic Structural Modeling

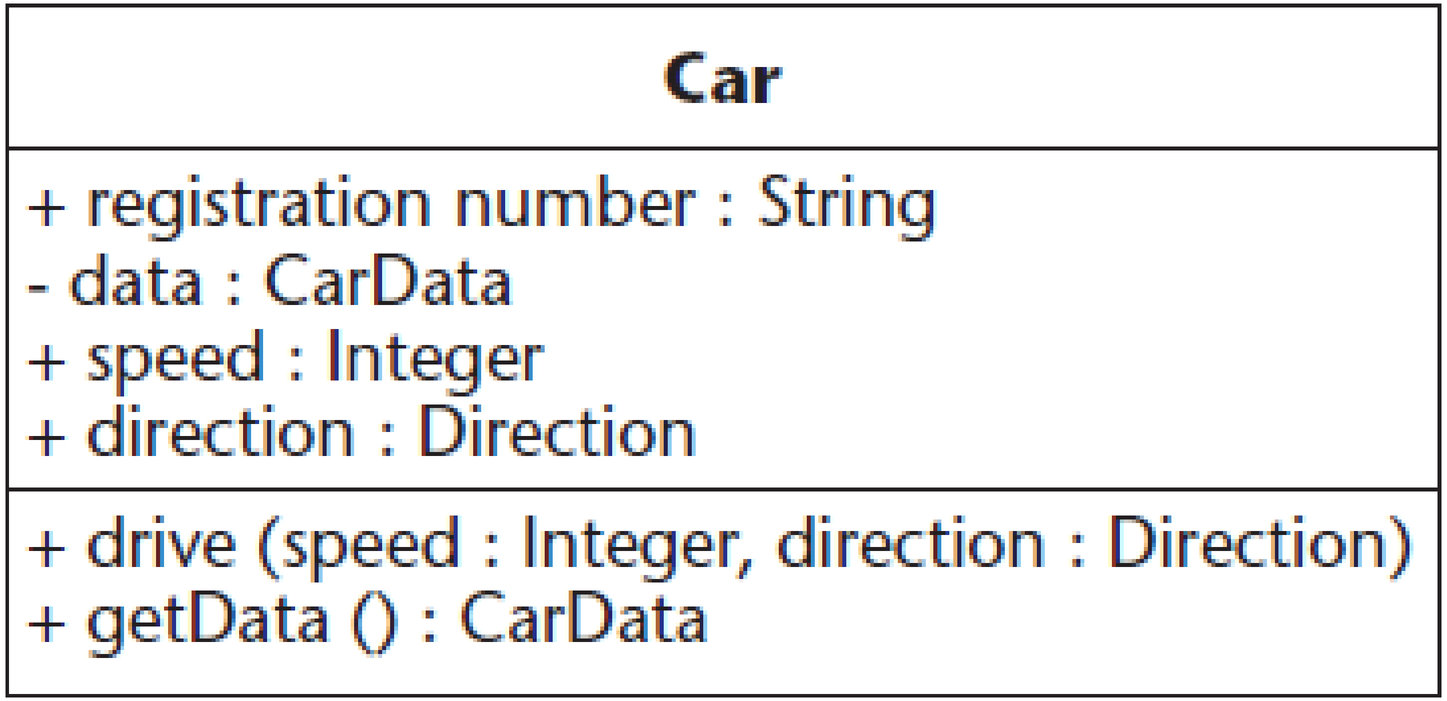


Figure: The class Car has attributes and operations. The operation drive has two parameters, *speed* and *direction*. The operation getData has a return type, CarData.

Basic Structural Modeling

- A class can also have class-scope operations, as shown in the adjacent Figure.
- A class-scope operation can be called without having an object of the class, but it is restricted to accessing only class-scope attributes.
- Class-scope operations are defined to carry out generic operations such as creating objects and finding objects when a specific object is not involved (except as possibly the result of the operation).

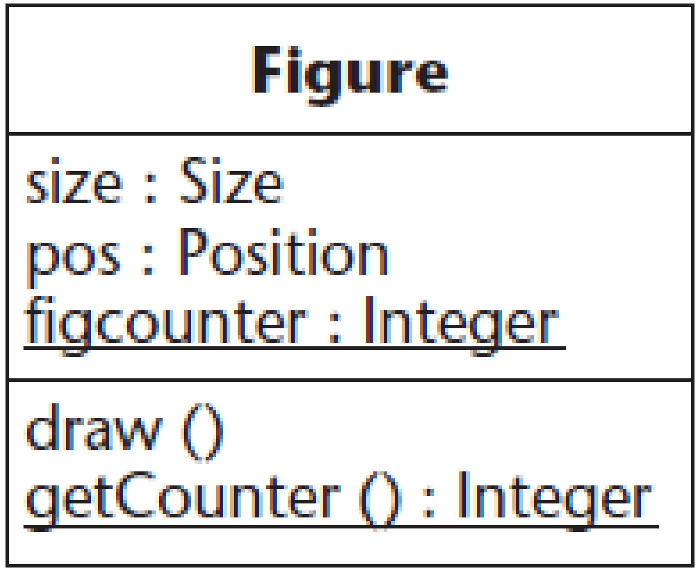


Figure Class-scope operation getCounter.

Basic Structural Modeling

- The formal syntax for an operation is:

  visibility name (parameter-list) : return-type-expression { property-string}

  where parameter-list is a comma-separated list of formal parameters, each specified using the syntax:

  direction name : type-expression [multiplicity] = default-value

  { property-string }

Basic Structural Modeling

- Visibility is the same as for attributes (+ for public, – for private, # for protected).
- Not all operations need to have a return-type, parameters, or a property-string, but operations of the same name must always have a unique signature (return-type, name, parameters).
- Adjacent Figure shows public operations with varying signatures.

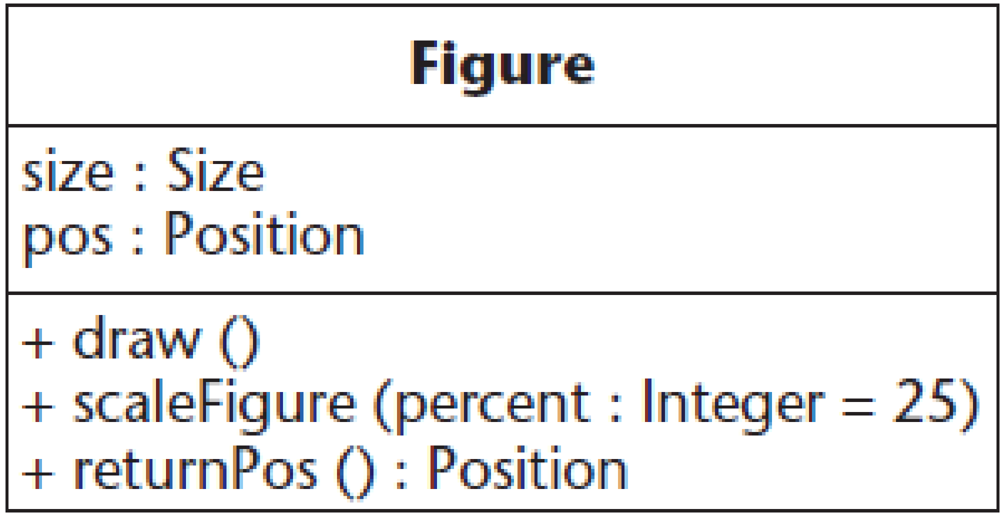


Figure
size : Size
pos : Position
+ draw ()
+ scaleFigure (percent : Integer = 25)
+ returnPos () : Position

Figure: Operation signatures.

Basic Structural Modeling

- A parameter's direction indicates whether it is being sent into or out of the operation (in, inout, out). It is also possible to have default values on parameters, which means that if the caller of the operation doesn't provide a parameter, the parameter will use the specified default value, as shown in this Figure.

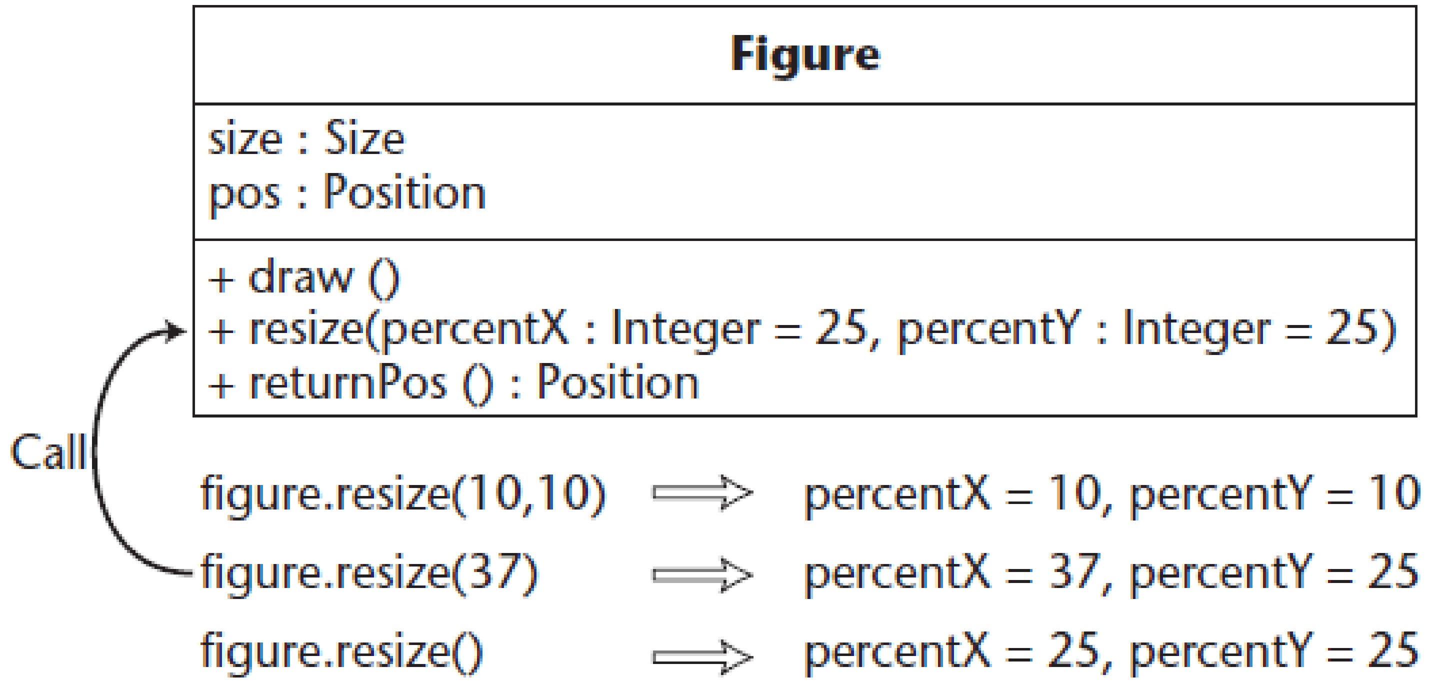


Figure
size : Size
pos : Position
+ draw ()
+ resize(percentX : Integer = 25, percentY : Integer = 25)
+ returnPos () : Position
Call
figure.resize(10,10) ⟹ percentX = 10, percentY = 10
figure.resize(37) ⟹ percentX = 37, percentY = 25
figure.resize() ⟹ percentX = 25, percentY = 25

Figure: Default values for parameters.

Basic Structural Modeling

Note: The operation is a part of the interface for a class; the implementation of an operation is called a method.

- A **persistent class** is one whose objects exist after the program that created it ends.
- Persistent class objects store themselves in a database, a file, or some other permanent storage, and typically have a class-scope operation to handle the storing of the objects, for example, store (), load (), create ().
- A class can be described as persistent by using one of the UML extensions.
- Many modelers put the persistent property in the name compartment (when shown in a class diagram, a property is put within curly braces, as in {persistent}).
- If you are modeling for a specific platform, that platform likely has mechanisms for indicating persistence. For example, for Enterprise JavaBeans (EJB), you would use a stereotype for an entity bean to show persistence.

Basic Structural Modeling

- A simple translation from UML to Java for adjacent Figure shows how the information in the diagram reflects information in a Java class. A development support tool can enable the generation of such basic class information from UML. The Java code for the class in this Figure is:

```
public class Figure
{
        private int x = 0;
        private int y = 0;
        public void draw ()
        {
        // Java code for drawing the figure
        }
} ;
```

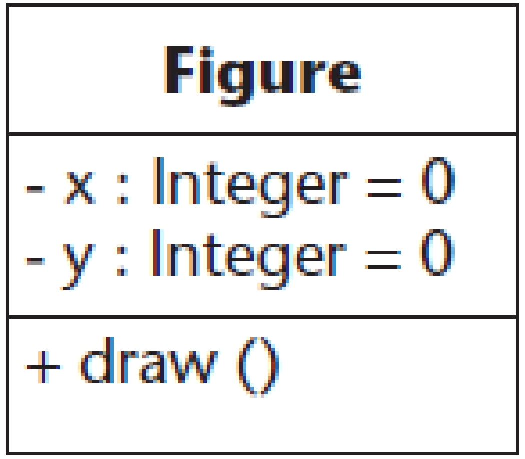


Figure: A Figure class.

Basic Structural Modeling

- The Java code for creating figure objects and calling the draw operation is:

```
Figure fig1 = new Figure();
Figure fig2 = new Figure();
fig1.draw();
fig2.draw();
```

- When objects are created, normally they should initialize attributes and links to other objects.
- It is possible to have an operation called create that is a class-scope operation used to create and initiate the object.
- It is also possible to have an operation with the same name as the class, which would correspond to a constructor in a programming language (such as in C++ or Java).

Basic Structural Modeling

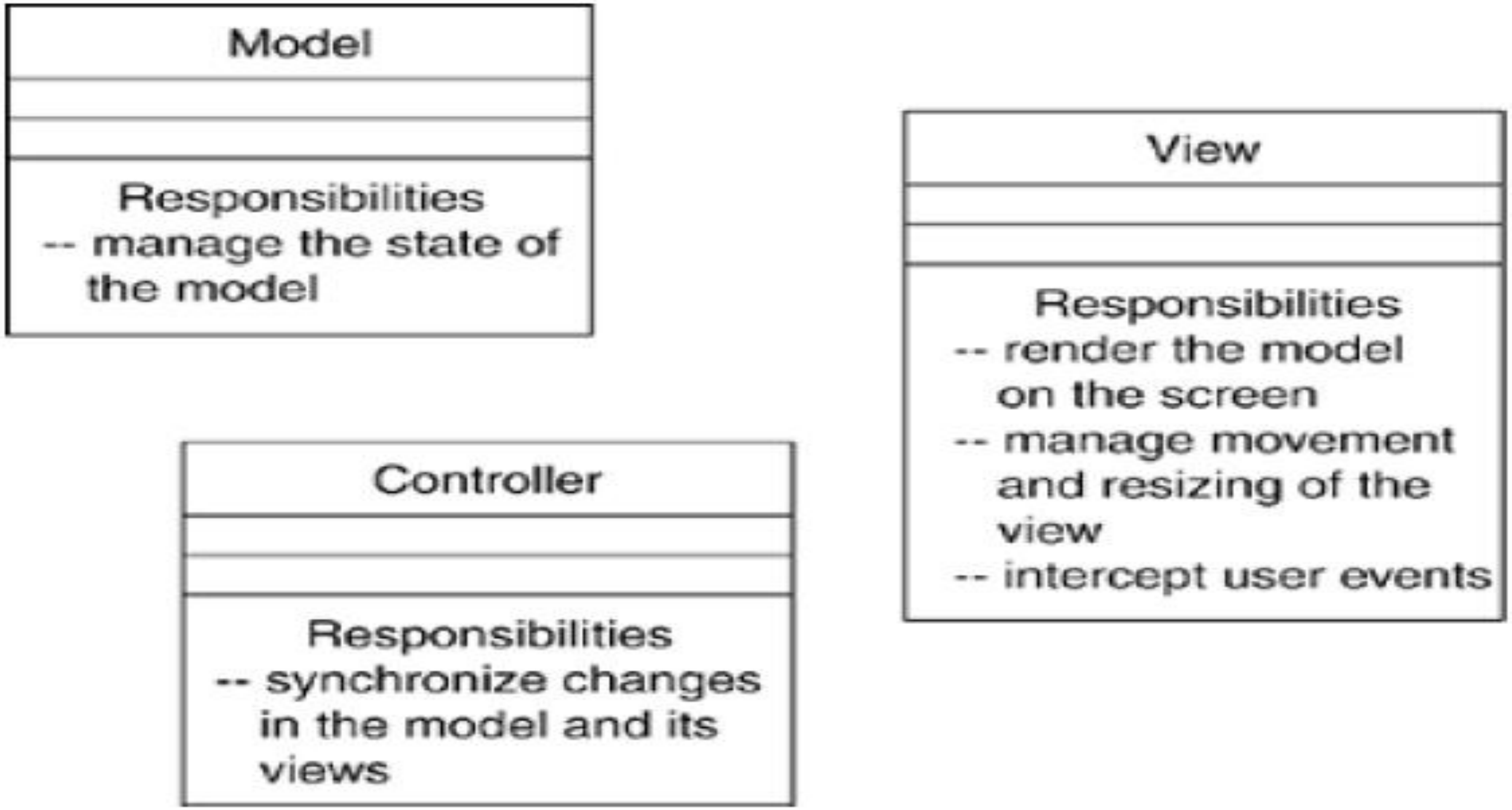
Modeling the Distribution of Responsibilities in a System

Once you start modeling more than just a handful of classes, you will want to be sure that your abstractions provide a balanced set of responsibilities.

To model the distribution of responsibilities in a system,

- Identify a set of classes that work together closely to carry out some behavior.
- Identify a set of responsibilities for each of these classes.
- Look at this set of classes as a whole, split classes that have too many responsibilities into smaller abstractions, collapse tiny classes that have trivial responsibilities into larger ones, and reallocate responsibilities so that each abstraction reasonably stands on its own.
- Consider the ways in which those classes collaborate with one another, and redistribute their responsibilities accordingly so that no class within a collaboration does too much or too little.

Basic Structural Modeling


Model
Responsibilities
-- manage the state of the model
View
Responsibilities
-- render the model on the screen
-- manage movement and resizing of the view
-- intercept user events
Controller
Responsibilities
-- synchronize changes in the model and its views

Basic Structural Modeling

Primitive Types

- A **primitive type** is not a class and has no substructure but defines a data type.
- UML uses a handful of primitive types, including “string” for a set of characters, “integer” for numbers, and “Boolean” for true/false values.
- In a model without deployment detail, a simple subset of normal types could be used (for example, integer, string, float).
- The primitive types are used for return-types, parameters, and attributes. Classes defined in any class diagram in the model can also be used to type attributes, return types, or parameters.
- A modeler or a profile could further define general types such as date, real, long, and so on for a specific language.

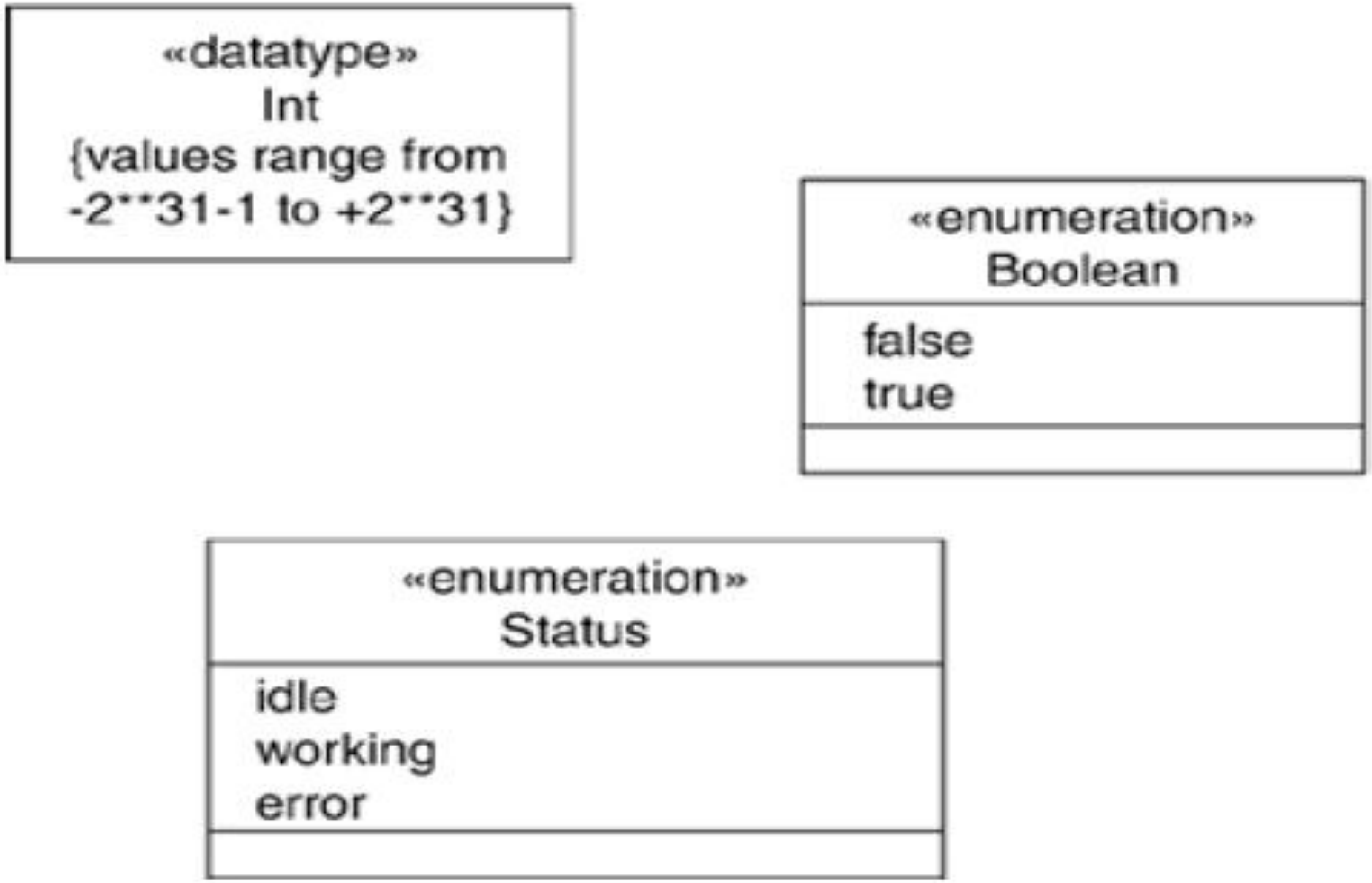
Basic Structural Modeling

Modeling Primitive Types

To model primitive types,

- Model the thing you are abstracting as a class or an enumeration, which is rendered using class notation with the appropriate stereotype.
- If you need to specify the range of values associated with this type, use constraints.

Basic Structural Modeling



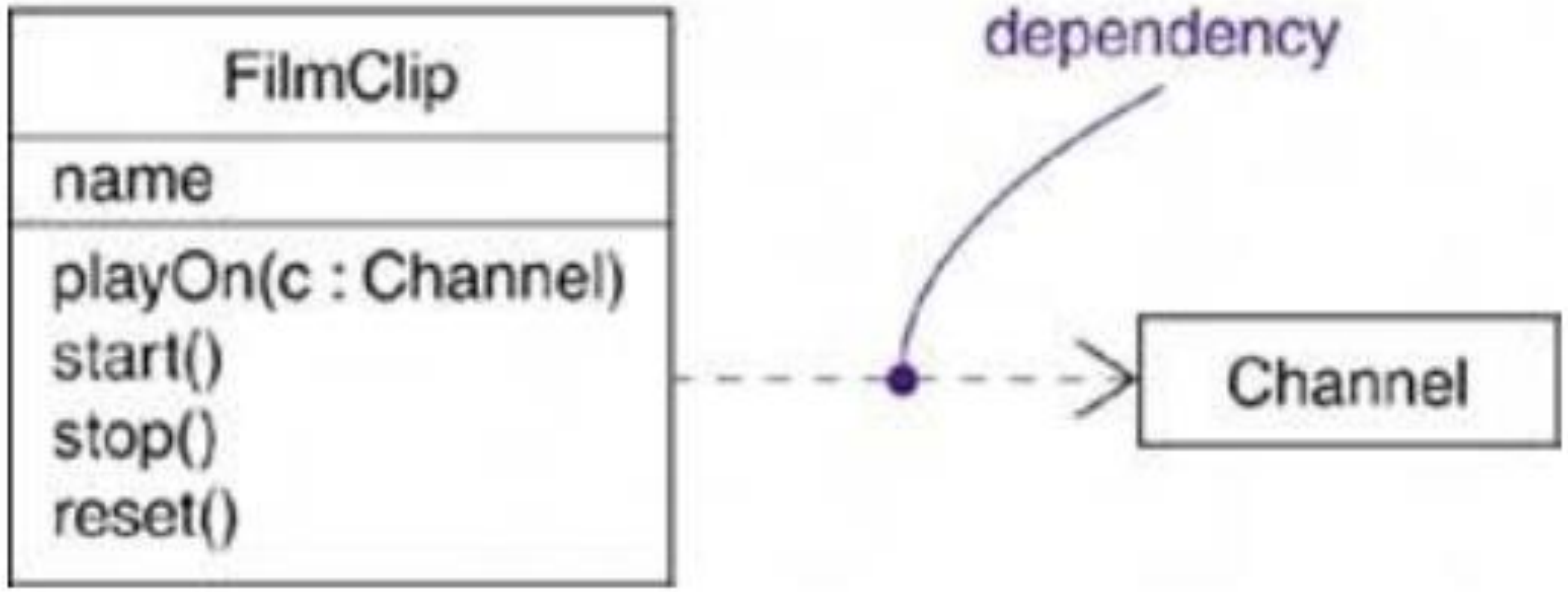
Basic Structural Modeling

Relationships

Class diagrams consist of classes and the relationships among them. The relationships that can be used are dependencies, generalizations, associations, and abstractions/realizations.

- A ***dependency*** is a relationship that states that one thing (for example, class Window) uses the information and services of another thing (for example, class Event), but not necessarily the reverse.
- A dependency is a relationship between elements, one independent and one dependent. A change in the independent element will affect the dependent element.
- Graphically, a dependency is rendered as a dashed directed line, directed to the thing being depended on. Choose dependencies when you want to show one thing using another.

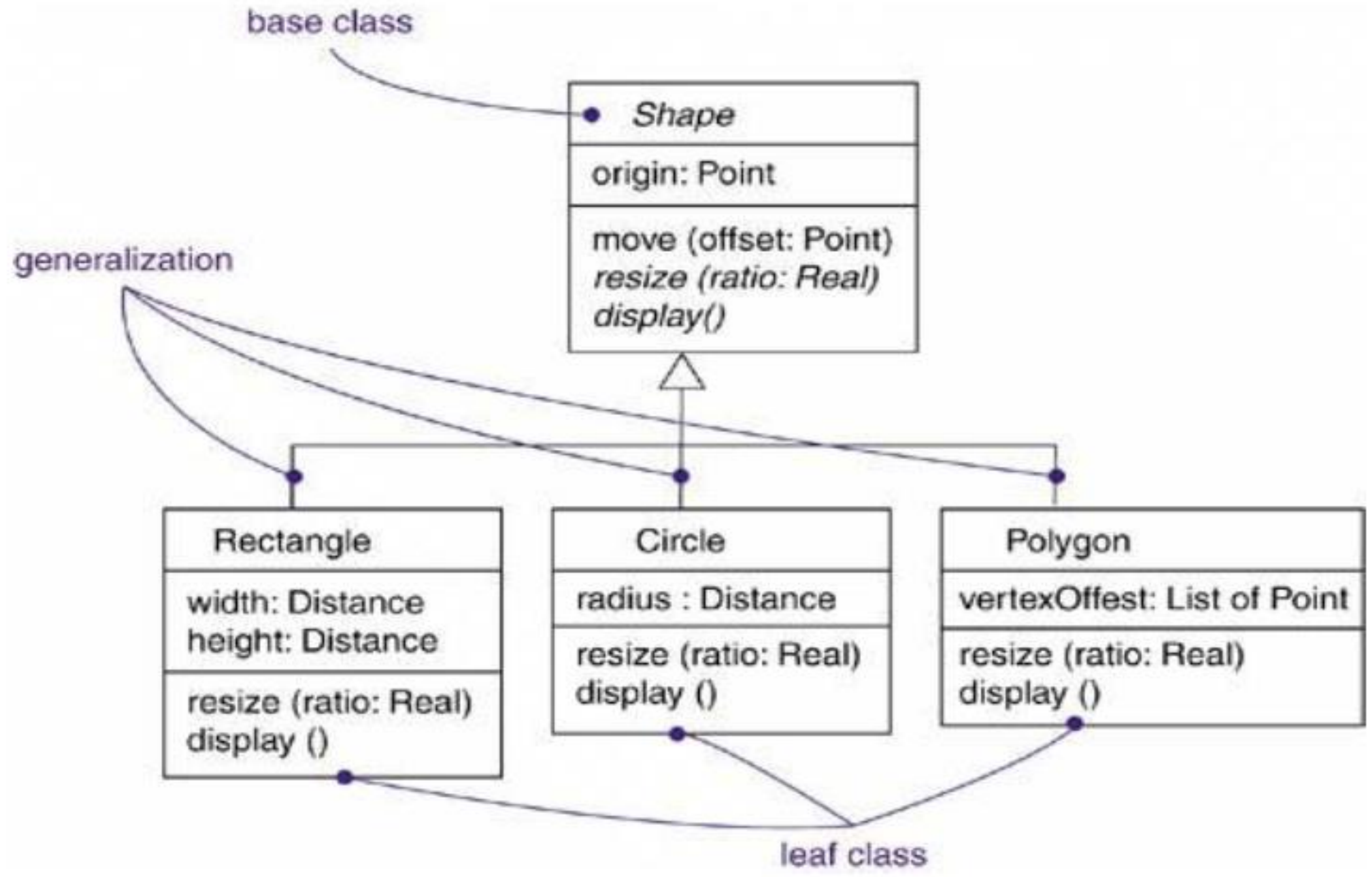
Basic Structural Modeling



Basic Structural Modeling

- A ***generalization*** is a relationship between a more general kind of thing (called the superclass or parent) and a more specific element (called the subclass or child).
- The more specific element can contain only additional information. An instance (an object is an instance of a class) of the more specific element may be used wherever the more general element is allowed.
- Generalization is sometimes called an "is-a-kind-of" relationship: one thing (like the class BayWindow) is-a-kind-of a more general thing (for example, the class Window). An objects of the child class may be used for a variable or parameter typed by the parent, but not the reverse.

Basic Structural Modeling



Basic Structural Modeling

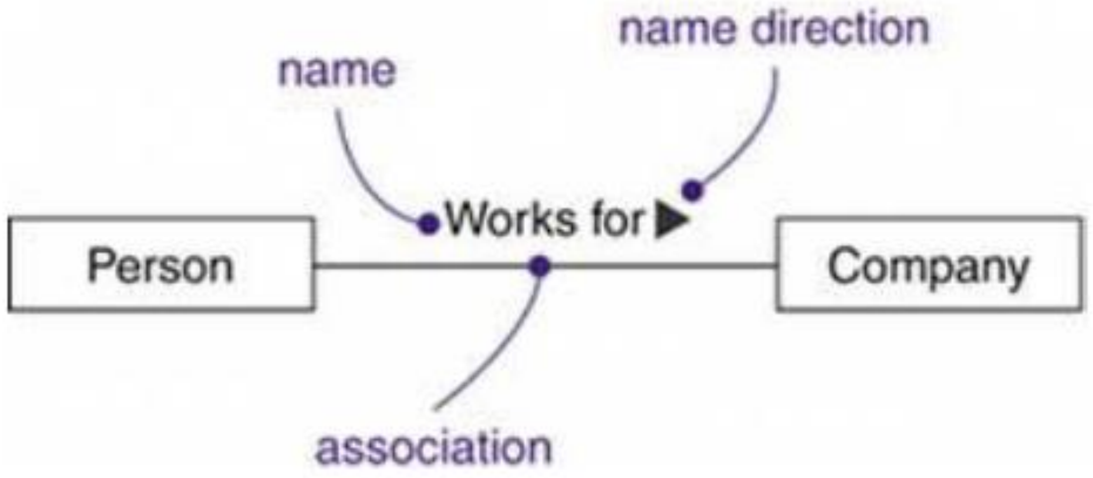
- An ***association*** is a connection between classes, which means that it is also a connection between objects of those classes.
- In UML, an association is defined as a relationship that describes a set of links, where link is defined as a semantic connection among a tuple of objects.
- Given an association connecting two classes, you can relate objects of one class to objects of the other class. It's quite legal to have both ends of an association circle back to the same class. This means that, given an object of the class, you can link to other objects of the same class.

Basic Structural Modeling

Beyond this basic form, there are four adornments that apply to associations.

Name

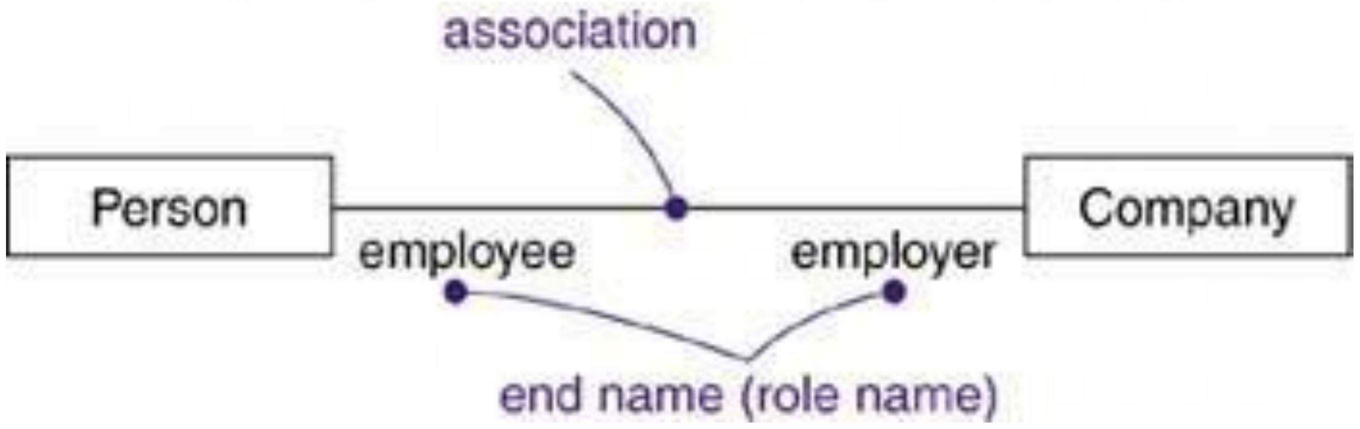
- An association can have a name, and you use that name to describe the nature of the relationship. So that there is no ambiguity about its meaning, you can give a direction to the name by providing a direction triangle that points in the direction you intend to read the name.



Basic Structural Modeling

Role

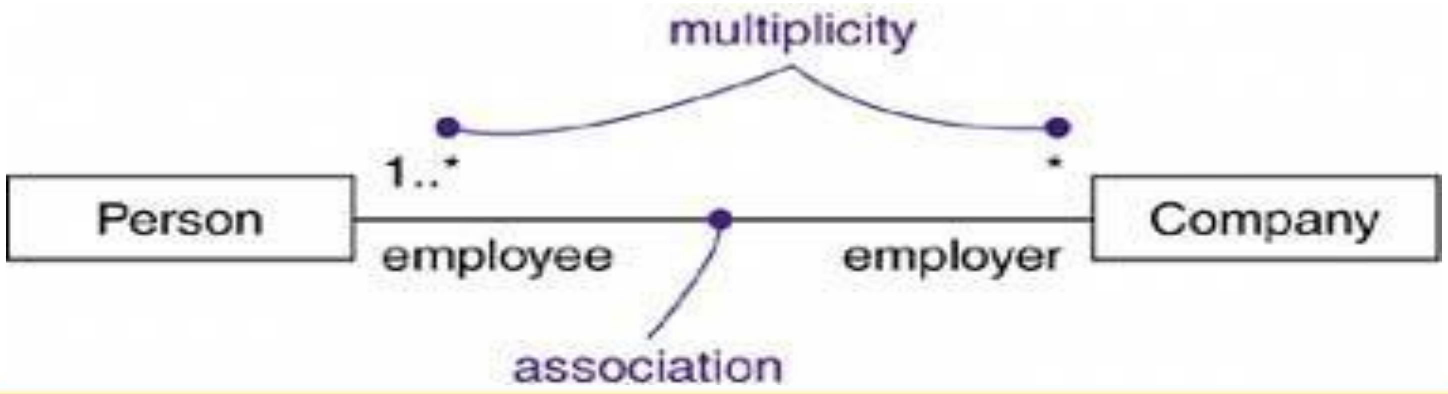
- When a class participates in an association, it has a specific role that it plays in that relationship; a role is just the face of the class at the far end of the association presents to the class at the near end of the association.
- You can explicitly name the role a class plays in an association. The role played by an end of an association is called an end name (in UML1, it was called a role name). the class Person playing the role of employee is associated with the class Company playing the role of employer.



Basic Structural Modeling

Multiplicity

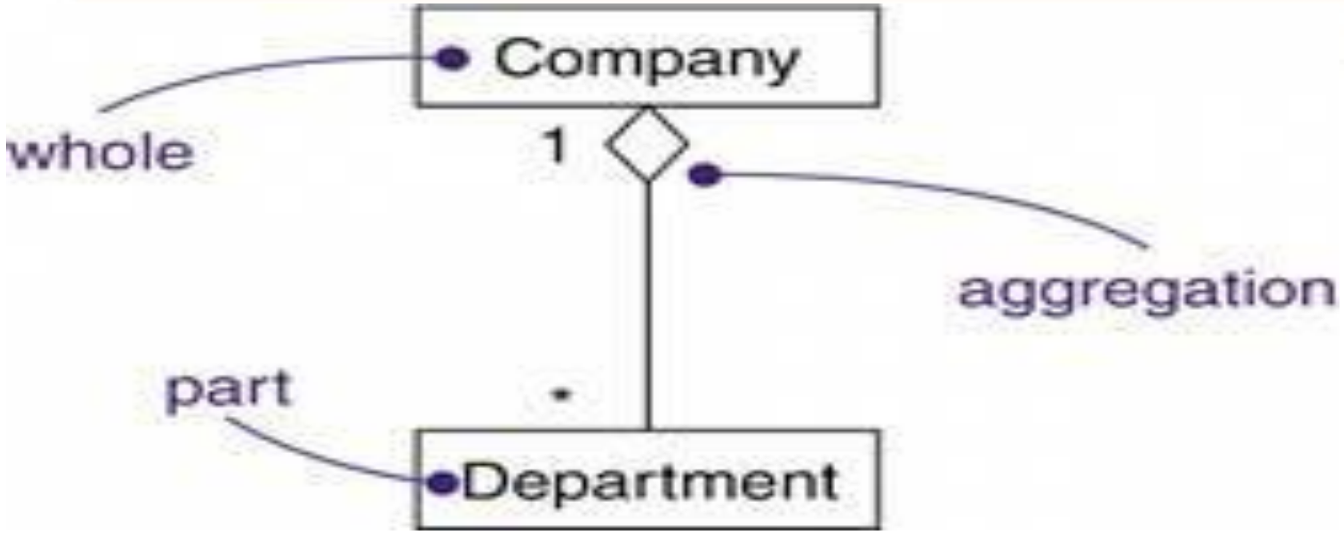
- An association represents a structural relationship among objects. In many modeling situations, it's important for you to state how many objects may be connected across an instance of an association. This "how many" is called the *multiplicity* of an association's role. It represents a range of integers specifying the possible size of the set of related objects.
- The number of objects must be in the given range. You can show a multiplicity of exactly one (1), zero or one (0..1), many (0..*), or one or more (1..*). You can give an integer range (such as 2..5). You can even state an exact number (for example, 3, which is equivalent to 3..3).



Basic Structural Modeling

Aggregation

- A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other.
- Sometimes you will want to model a "whole/part" relationship, in which one class represents a larger thing (the "whole"), which consists of smaller things (the "parts"). This kind of relationship is called ***aggregation***, which represents a "has-a" relationship



Basic Structural Modeling

- An ***abstraction*** is a relationship between two descriptions of the same thing, but at different levels. A realization is a type of abstraction that shows a model element that realizes a more general element.

Basic Structural Modeling

Common Modeling Techniques

Modeling Simple Dependencies

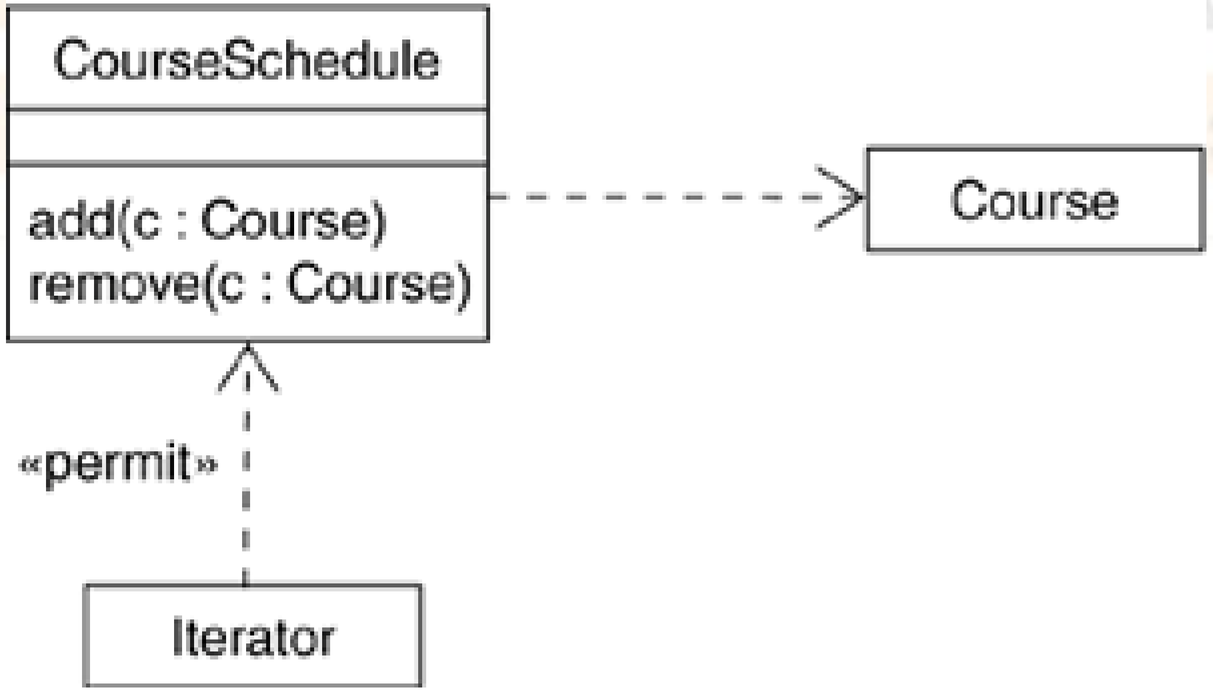
A common kind of dependency relationship is the connection between a class that uses another class as a parameter to an operation.

To model this using relationship,

- Create a dependency pointing from the class with the operation to the class used as a parameter in the operation.

Basic Structural Modeling

A set of classes drawn from a system that manages the assignment of students and instructors to courses in a university. This figure shows a dependency from Course Schedule to Course, because Course is used in both the add and remove operations of Course Schedule.



Basic Structural Modeling

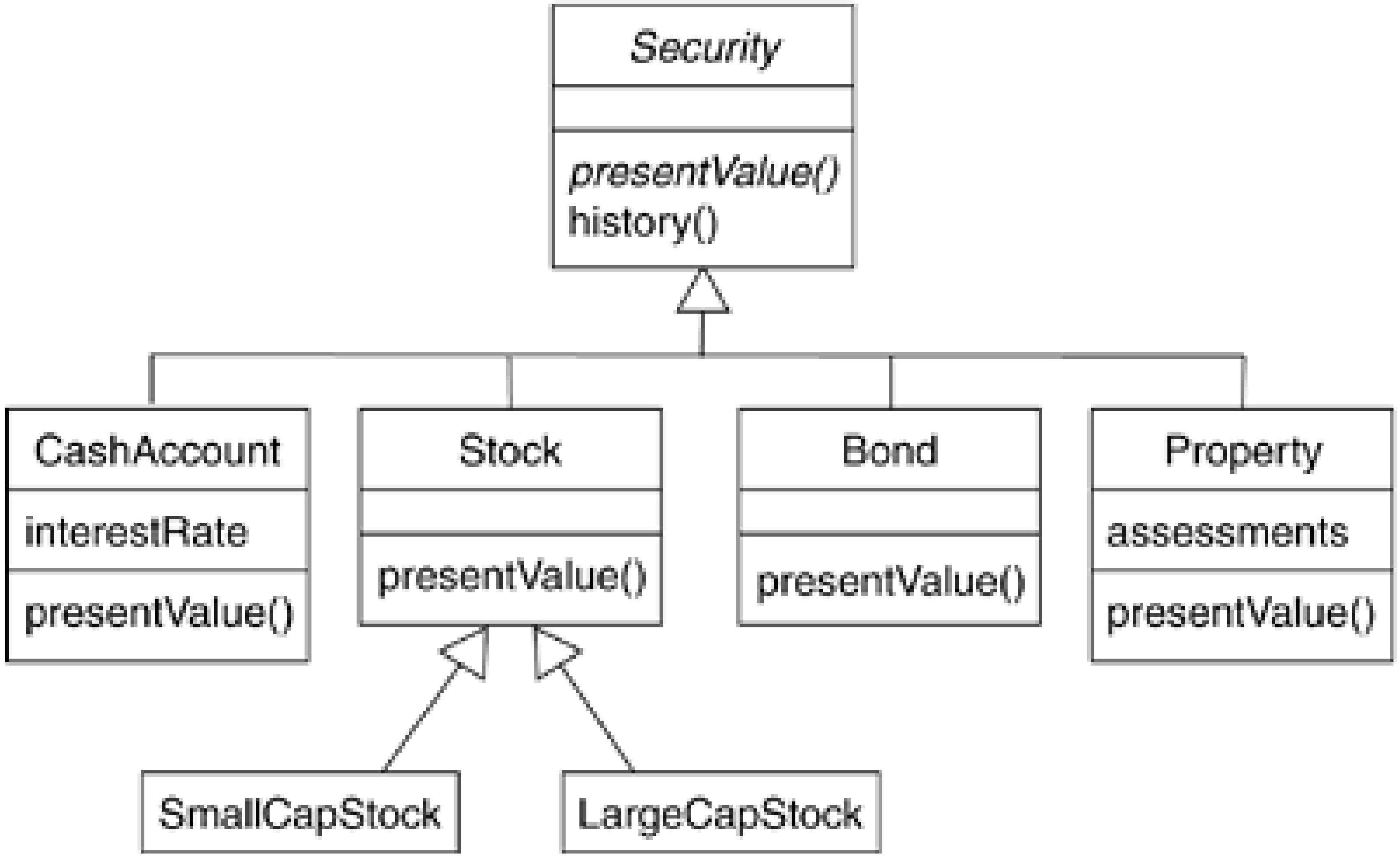
Modeling Single Inheritance

In modeling the vocabulary of your system, a better way would be to extract any common structural and behavioral features and place them in more-general classes from which the specialized ones inherit.

To model inheritance relationships,

- Given a set of classes, look for responsibilities, attributes, and operations that are common to two or more classes.
- Elevate these common responsibilities, attributes, and operations to a more general class. If necessary, create a new class to which you can assign these elements.
- Specify that the more-specific classes inherit from the more-general class by placing a generalization relationship that is drawn from each specialized class to its more-general parent.

Basic Structural Modeling



Basic Structural Modeling

Modeling Structural Relationships

When you model with dependencies or generalization relationships, you may be modeling classes that represent different levels of importance or different levels of abstraction. Given a dependency between two classes, one class depends on another but the other class has no knowledge of the one.

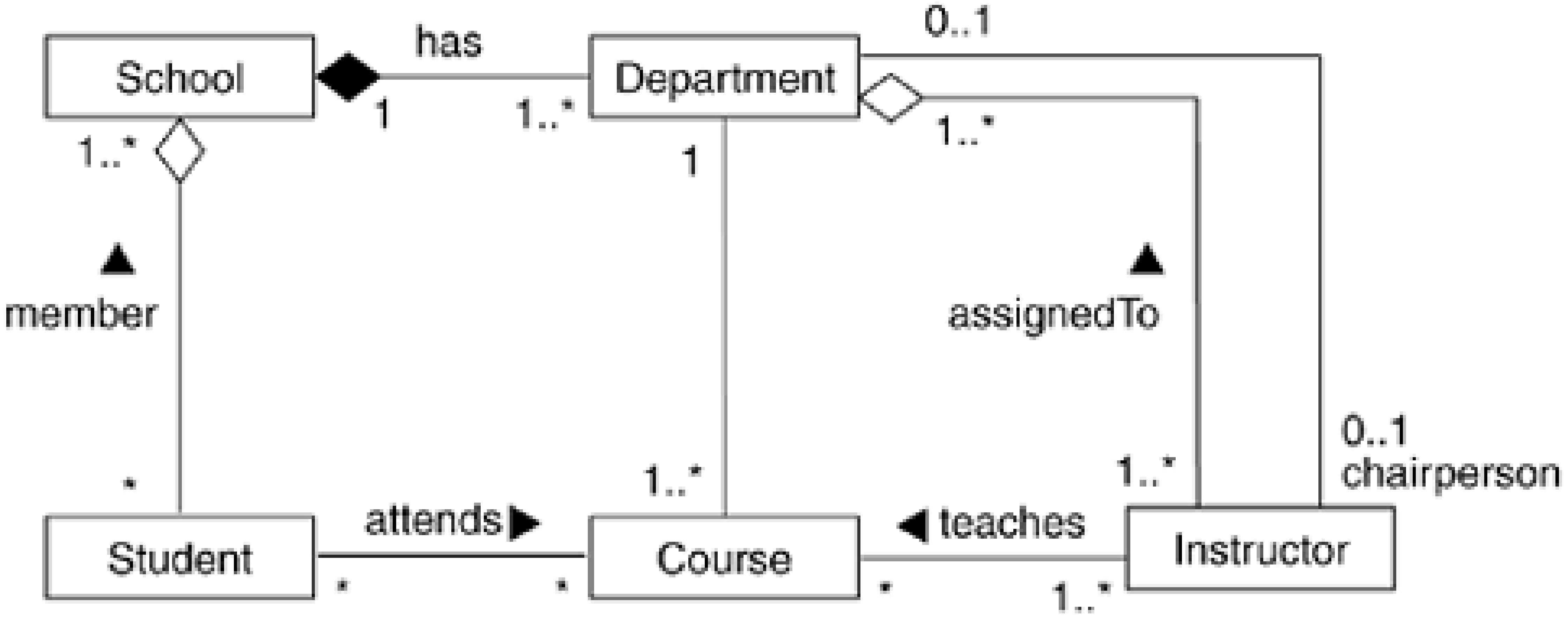
To model structural relationships,

- For each pair of classes, if you need to navigate from objects of one to objects of another, specify an association between the two. This is a data-driven view of associations.

Basic Structural Modeling

- For each pair of classes, if objects of one class need to interact with objects of the other class other than as local variables in a procedure or parameters to an operation, specify an association between the two. This is more of a behavior-driven view of associations.
- For each of these associations, specify a multiplicity (especially when the multiplicity is not *, which is the default), as well as role names (especially if they help to explain the model).
- If one of the classes in an association is structurally or organizationally a whole compared with the classes at the other end that look like parts, mark this as an aggregation by adorning the association at the end near the whole with a diamond.

Basic Structural Modeling



Basic Structural Modeling

Common Mechanisms

Terms and Concepts

- A ***note*** is a graphical symbol for rendering constraints or comments attached to an element or a collection of elements. Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.
- A ***stereotype*** is an extension of the vocabulary of the UML, allowing you to create new kinds of building blocks similar to existing ones but specific to your problem. Graphically, a stereotype is rendered as a name enclosed by guillemets (French quotation marks of the form « »), placed above the name of another element.
- Optionally the stereotyped element may be rendered by using a new icon associated with that stereotype.

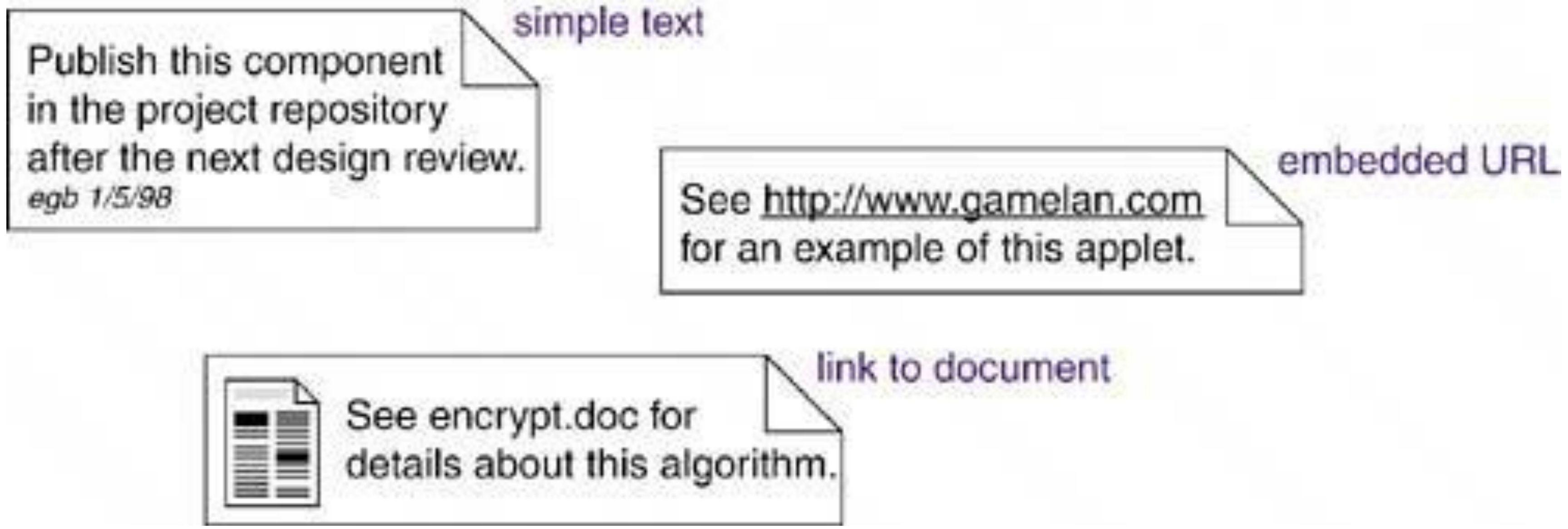
Basic Structural Modeling

- A ***tagged value*** is a property of a stereotype, allowing you to create new information in an element bearing that stereotype. Graphically, a tagged value is rendered as a string of the form name = value within a note attached to the object.
- A ***constraint*** is a textual specification of the semantics of a UML element, allowing you to add new rules or to modify existing ones. Graphically, a constraint is rendered as a string enclosed by brackets and placed near the associated element or connected to that element or elements by dependency relationships. As an alternative, you can render a constraint in a note.

Basic Structural Modeling

Notes

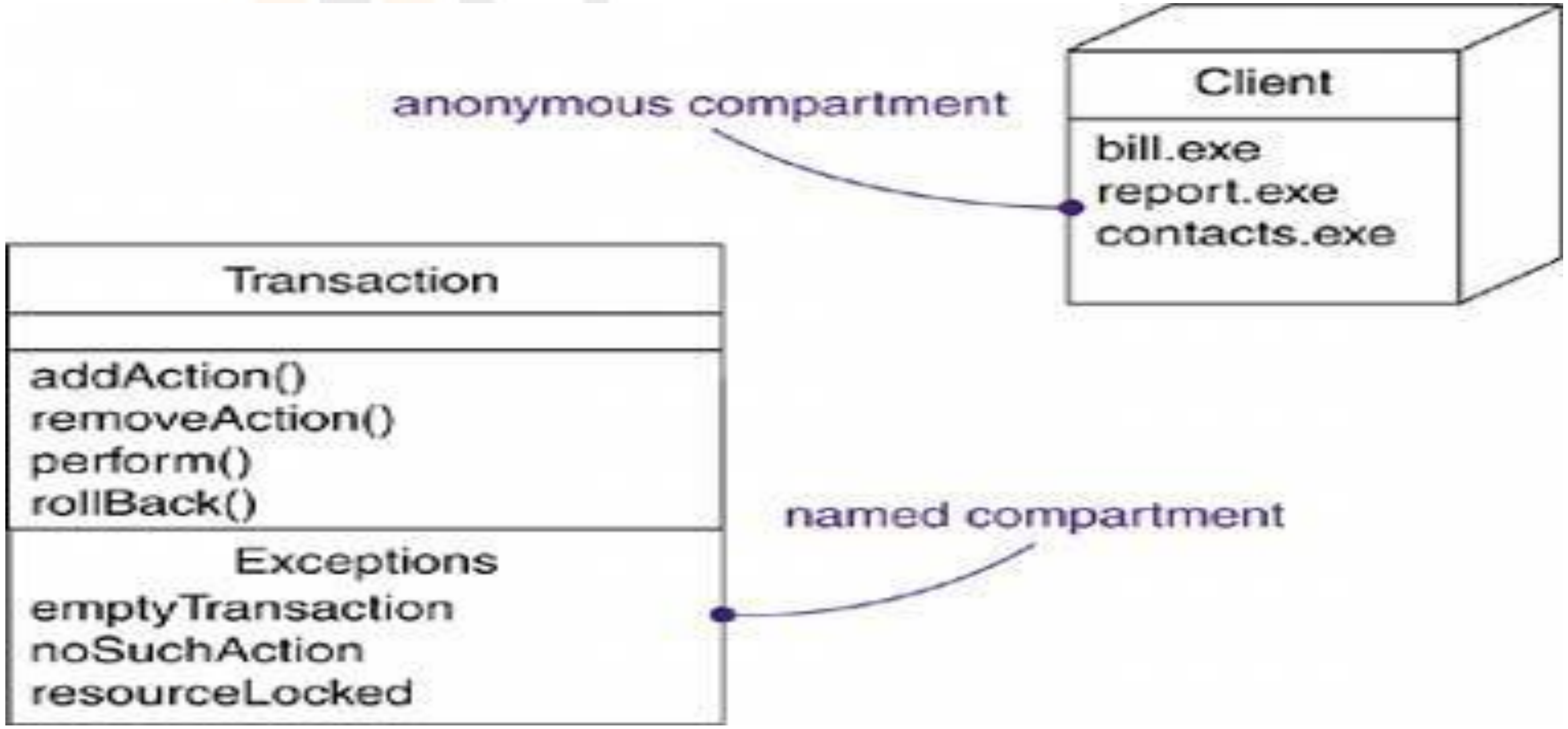
- A note that renders a comment has no semantic impact, meaning that its contents do not alter the meaning of the model to which it is attached. This is why notes are used to specify things like requirements, observations, reviews, and explanations, in addition to rendering constraints.
- A note may contain any combination of text or graphics


simple text
Publish this component in the project repository after the next design review.
egb 1/5/98
embedded URL
See http://www.gamelan.com for an example of this applet.
link to document
See encrypt.doc for details about this algorithm.

Basic Structural Modeling

Other Adornments

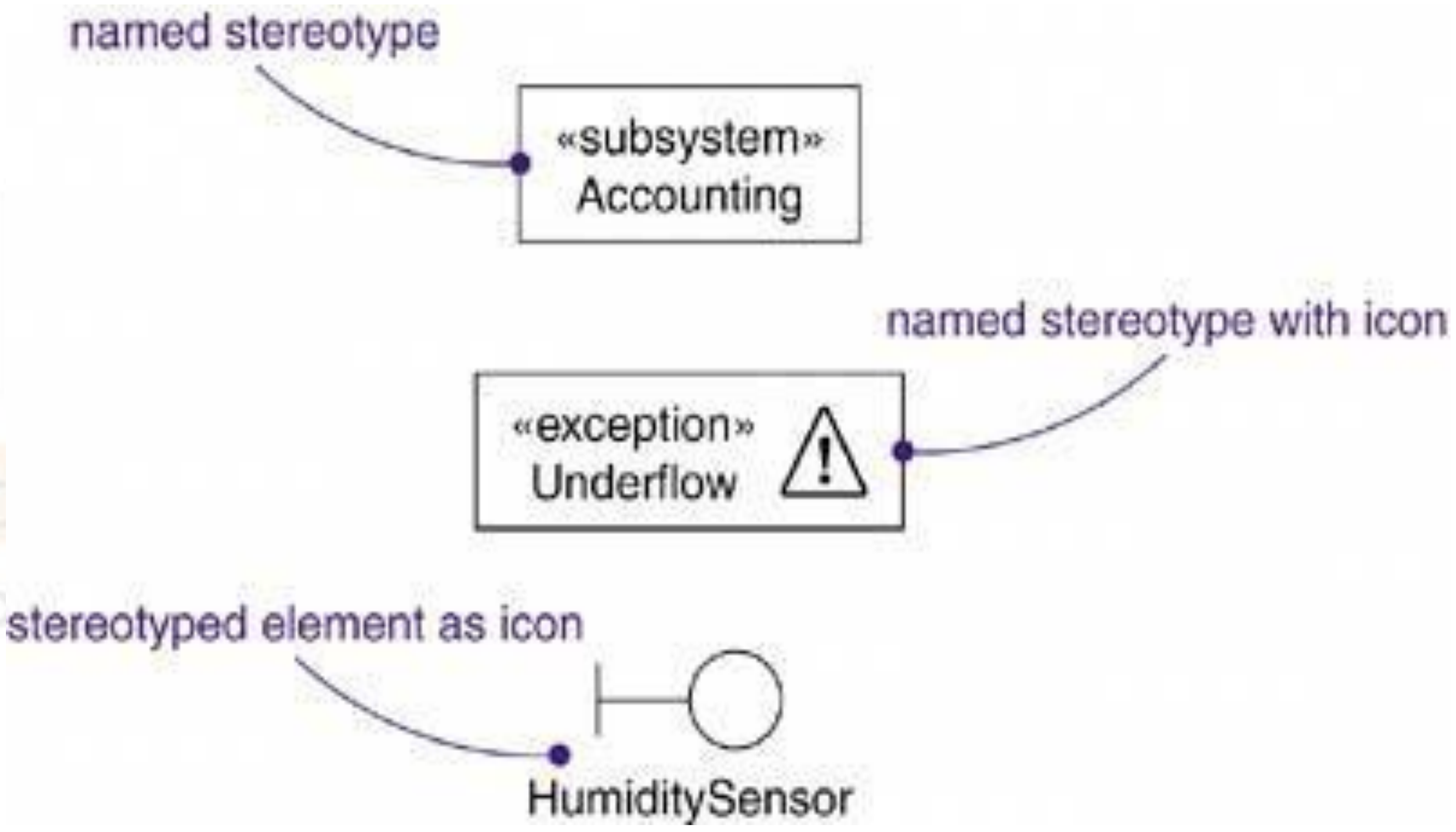
Adornments are textual or graphical items that are added to an element's basic notation and are used to visualize details from the element's specification



Basic Structural Modeling

Stereotypes

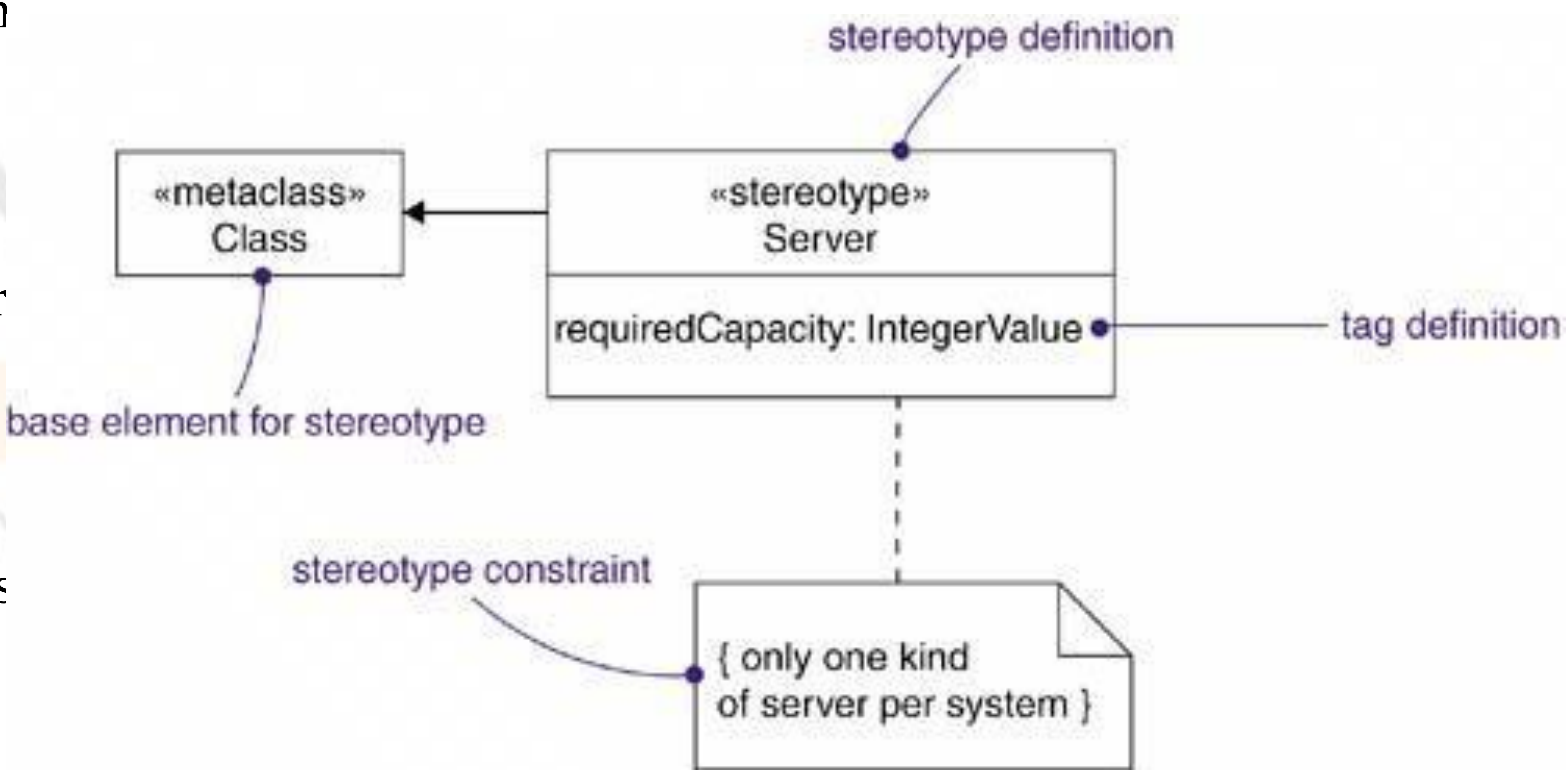
- The UML provides a language for structural things, behavioral things, grouping things, and notational things. These four basic kinds of things address the overwhelming majority of the systems you'll need to model.
- In its simplest form, a stereotype is rendered as a name enclosed by guillemets (for example, «name») and placed above the name of



Basic Structural Modeling

Tagged Values

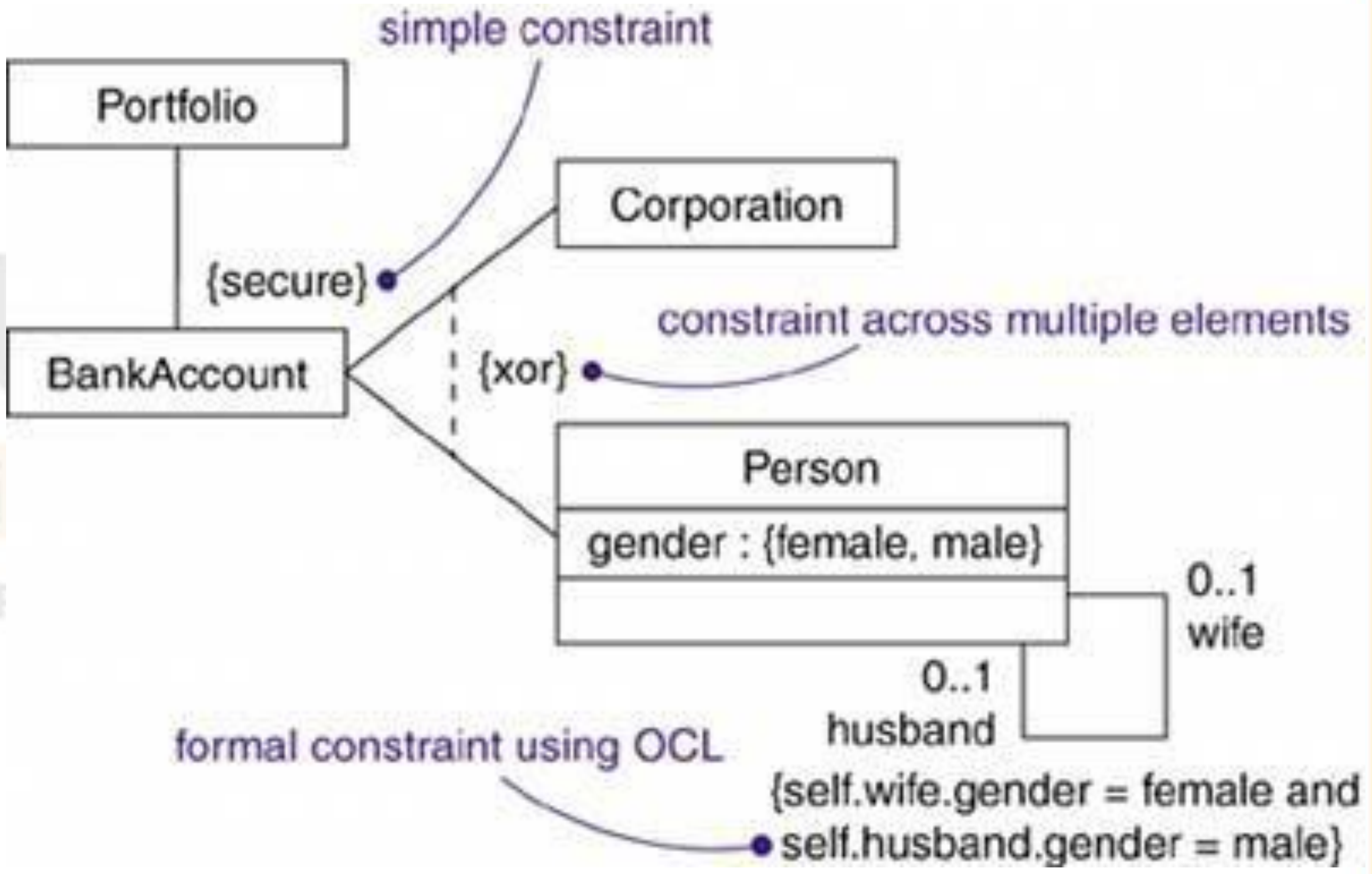
- Every thing in the UML has its own set of properties: classes have names, attributes, and operations; associations have names and two or more ends, each with its own properties; and so on. With stereotypes, you can add new things to the UML; with tagged values, you can add new properties to a stereotype.



Basic Structural Modeling

Constraints

- Everything in the UML has its own semantics. Generalization (usually, if you know what's good for you) implies the Liskov substitution principle, and multiple associations connected to one class denote distinct relationships. With constraints, you can add new semantics or extend existing rules. A constraint specifies conditions that a run-time configuration must satisfy to conform to the model.



simple constraint
Portfolio
Corporation
{secure}
constraint across multiple elements
{xor}
BankAccount
Person
gender : {female, male}
0..1
wife
0..1
husband
formal constraint using OCL
{self.wife.gender = female and
self.husband.gender = male}

- Stereotype: Specifies that the classifier is a stereotype that may be applied to other elements

Basic Structural Modeling

Common Modeling Techniques

Modeling Comments

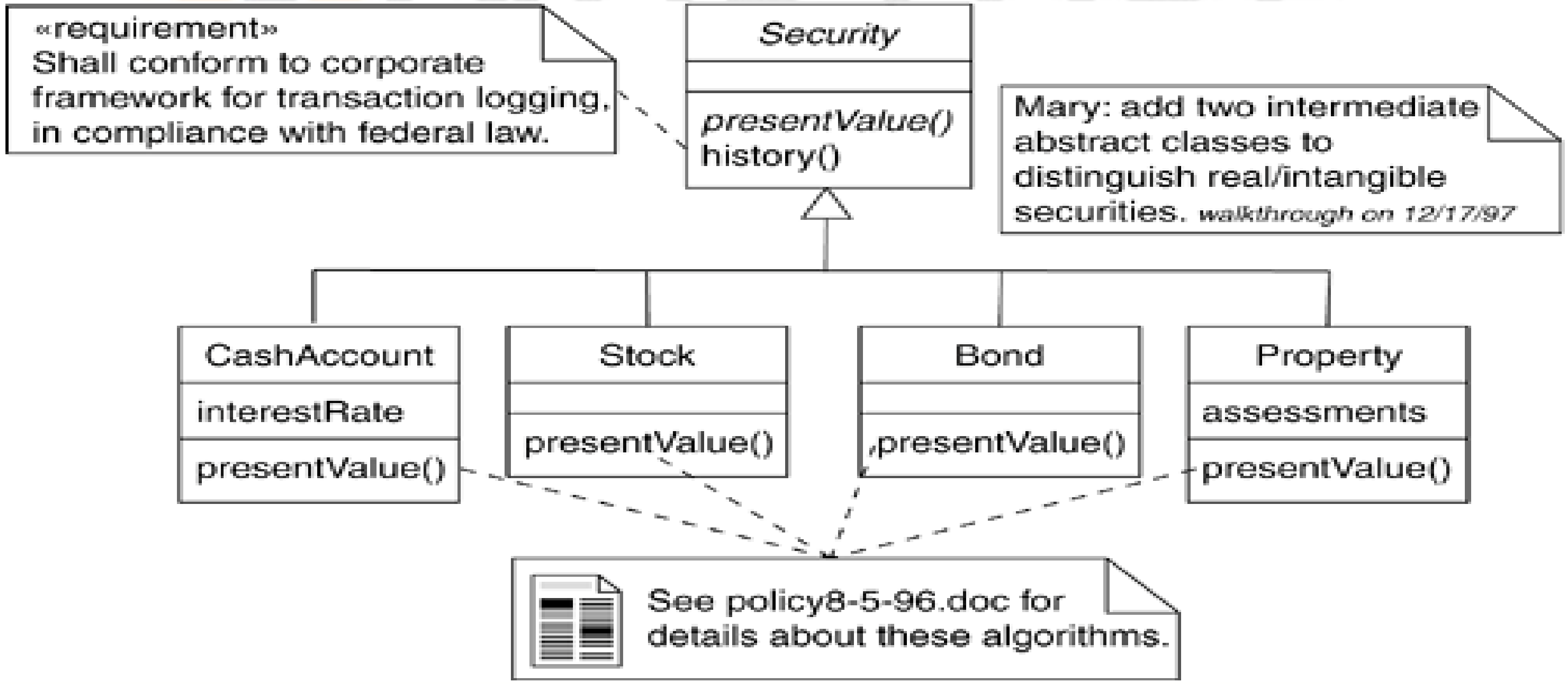
The most common purpose for which you'll use notes is to write down free-form observations, reviews, or explanations.

To model a comment,

- Put your comment as text in a note and place it adjacent to the element to which it refers. You can show a more explicit relationship by connecting a note to its elements using a dependency relationship.
- Remember that you can hide or make visible the elements of your model as you see fit. This means that you don't have to make your comments visible everywhere the elements to which it is attached are visible. Rather, expose your comments in your diagrams only insofar as you need to communicate that information in that context.

Basic Structural Modeling

- If your comment is lengthy or involves something richer than plain text, consider putting your comment in an external document and linking or embedding that document in a note attached to your model.
- As your model evolves, keep those comments that record significant decisions that cannot be inferred from the model itself, and unless they are of historic interest discard the others.


«requirement» Shall conform to corporate framework for transaction logging, in compliance with federal law.
Security
presentValue()
history()
Mary: add two intermediate abstract classes to distinguish real/intangible securities. walkthrough on 12/17/97
CashAccount
interestRate
presentValue()
Stock
presentValue()
Bond
presentValue()
Property
assessments
presentValue()
See policy8-5-96.doc for details about these algorithms.

Basic Structural Modeling

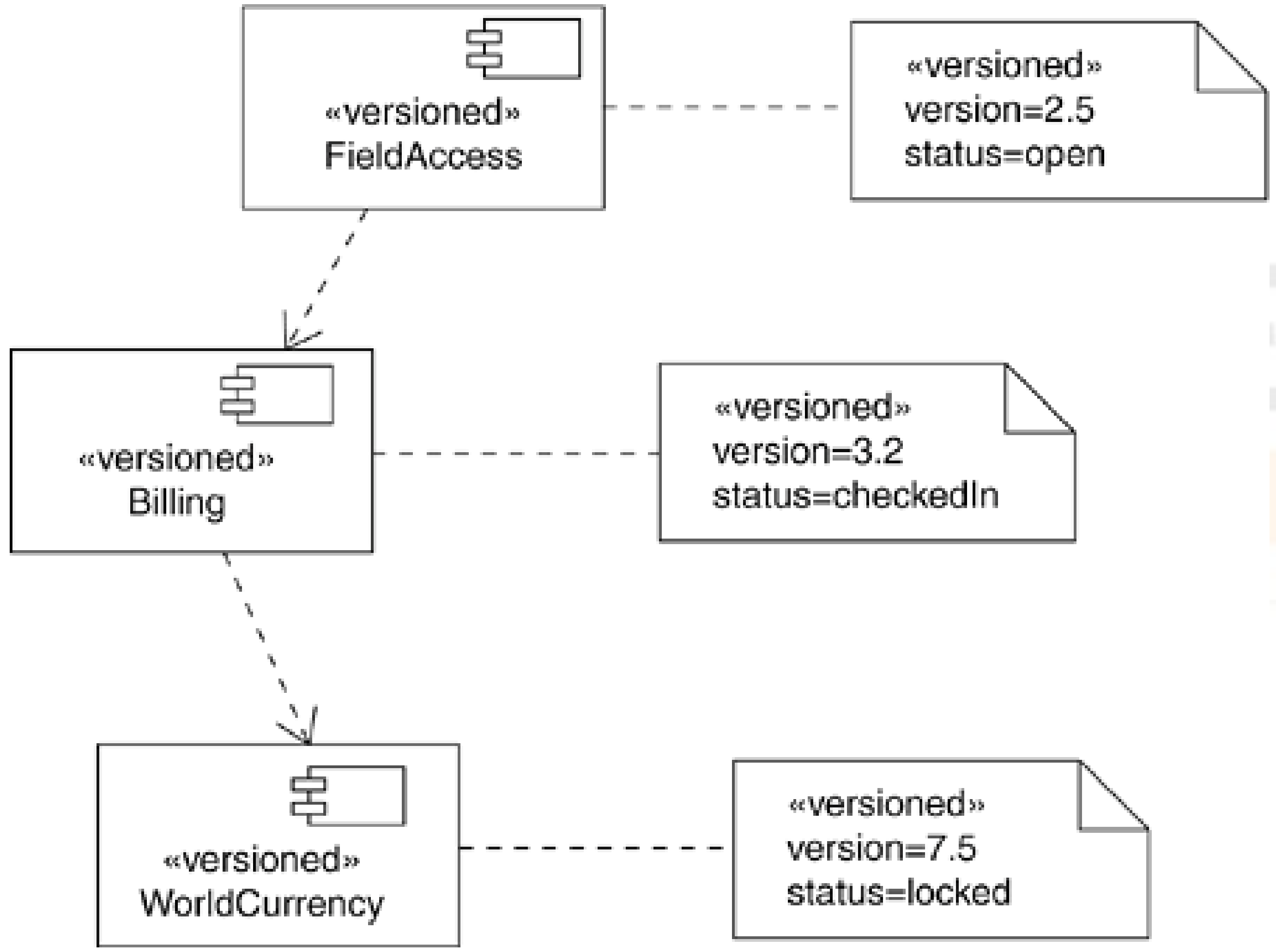
Modeling New Properties

The basic properties of the UML's building blocks attributes and operations for classes, the contents of packages

To model new properties,

- First, make sure there's not already a way to express what you want by using basic UML.
- If you re convinced there's no other way to express these semantics, define a stereotype and add the new properties to the stereotype. The rules of generalization apply tagged values defined for one kind of stereotype apply to its children.

Basic Structural Modeling



Basic Structural Modeling

Modeling New Semantics

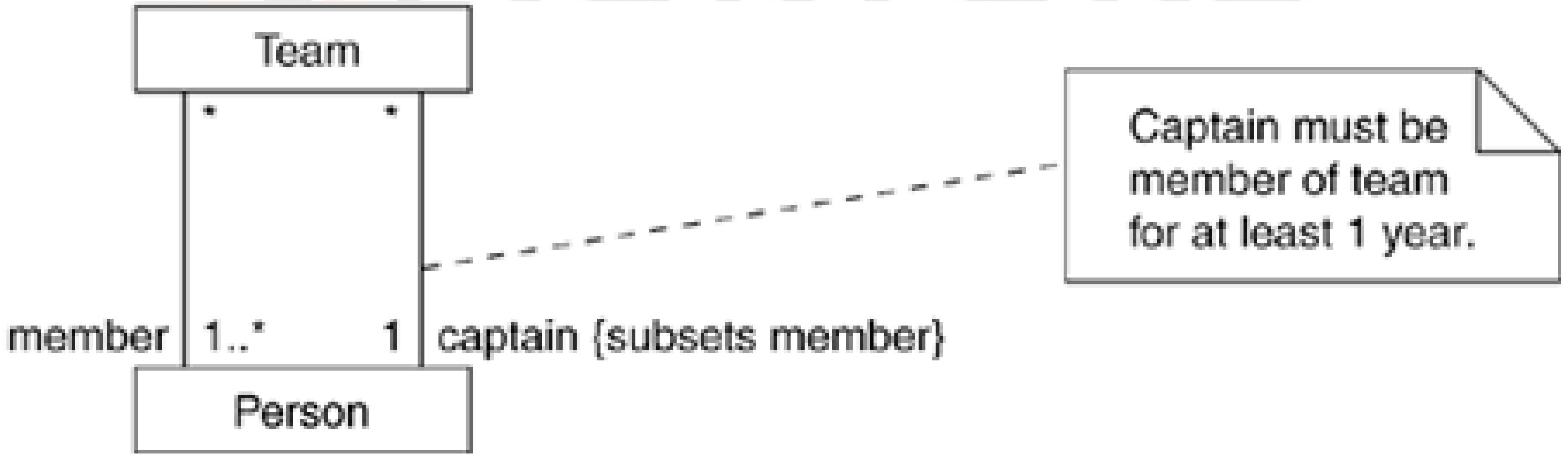
When you create a model using the UML, you work within the rules the UML lays down. However, if you find yourself needing to express new semantics about which the UML is silent or that you need to modify the UML's rules, then you need to write a constraint.

To model new semantics,

- First, make sure there's not already a way to express what you want by using basic UML.
- If you re convinced there's no other way to express these semantics, write your new semantics in a constraint placed near the element to which it refers. You can show a more explicit relationship by connecting a constraint to its elements using a dependency relationship.

Basic Structural Modeling

- If you need to specify your semantics more precisely and formally, write your new semantics using OCL.



Basic Structural Modeling

Diagrams

Terms and Concepts

- A ***system*** is a collection of subsystems organized to accomplish a purpose and described by a set of models, possibly from different viewpoints.
- A ***subsystem*** is a grouping of elements, some of which constitute a specification of the behavior offered by the other contained elements.
- A ***model*** is a semantically closed abstraction of a system, meaning that it represents a complete and self-consistent simplification of reality, created in order to better understand the system.

Basic Structural Modeling

- A ***view*** is a projection into the organization and structure of a system's model, focused on one aspect of that system.

- A ***diagram*** is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).

Basic Structural Modeling

In modeling real systems, irrespective of the problem domain, we can create the same kinds of diagrams, because they represent common views into common models.

Typically, we view the static parts of a system using one of the following diagrams.

1. Class diagram
2. Component diagram
3. Composite structure diagram
4. Object diagram
5. Deployment diagram
6. Artifact diagram

Basic Structural Modeling

We can often use five additional diagrams to view the dynamic parts of a system.

1. Use case diagram
2. Sequence diagram
3. Communication diagram
4. State diagram
5. Activity diagram

Basic Structural Modeling

Structural Diagrams

- The UML's structural diagrams exist to visualize, specify, construct, and document the *static aspects* of a system.
- We can think of the static aspects of a system as representing its relatively stable skeleton and scaffolding.
- Just as the static aspects of a house encompass the existence and placement of such things as walls, doors, windows, pipes, wires, and vents, so too do the static aspects of a software system encompass the existence and placement of such things as classes, interfaces, collaborations, components, and nodes.

1. Class diagram	Classes, interfaces, and collaborations
2. Component diagram	Components
3. Object diagram	Objects
4. Deployment diagram	Nodes

Basic Structural Modeling

Behavioral Diagrams

- The UML's behavioral diagrams are used to visualize, specify, construct, and document the *dynamic aspects* of a system.
- We can think of the dynamic aspects of a system as representing its changing parts. Just as the dynamic aspects of a house encompass airflow and traffic through the rooms of a house, so too do the dynamic aspects of a software system encompass such things as the flow of messages over time and the physical movement of components across a network.

1. Use case diagram	Organizes the behaviors of the system
2. Sequence diagram	Focuses on the time ordering of messages
3.Collaboration diagram	Focuses on the structural organization of objects that exchange messages
4. State diagram	Focuses on the changing state of a system driven by events
5. Activity diagram	Focuses on the flow of control from activity to activity

Basic Structural Modeling

Common Modeling Techniques

Modeling Different Views of a System

- To model a system from different views, Decide which views you need to best express the architecture of your system and to expose the technical risks to your project.
- The five views of an architecture described earlier are a good starting point.
- For each of these views, decide which artifacts you need to create to capture the essential details of that view.
- As part of your process planning, decide which of these diagrams you'll want to put under some sort of formal or semi-formal control.
- These are the diagrams for which you'll want to schedule reviews and to preserve as documentation for the project.

Basic Structural Modeling

For example, if you are modeling a simple monolithic application that runs on a single machine, you might need only the following handful of diagrams.

Use case view	Use case diagrams
Design view	Class diagrams (for structural modeling)
Interaction view	Interaction diagrams (for behavioral modeling)
Implementation view	Composite structure diagrams
Deployment view	None required

Basic Structural Modeling

- Similarly, if yours is a **client/server system**, you'll probably want to include component diagrams and deployment diagrams to model the physical details of your system. Finally, if you are modeling a complex, distributed system, you'll need to employ the full range of the UML's diagrams in order to express the architecture of your system and the technical risks to your project, as in the following.

Use case view	Use case diagrams; Sequence diagrams
Design view	Class diagrams (for structural modeling);
	Interaction diagrams (for behavioral modeling);
	State diagrams (for behavioral modeling); Activity diagrams
Interaction view	Interaction diagrams (for behavioral modeling)
Implementation view	Class diagrams; Composite structure diagrams
Deployment view	Deployment diagrams

Basic Structural Modeling

Modeling Different Levels of Abstraction

Not only do you need to view a system from several angles, you'll also find people involved in development who need the same view of the system but at different levels of abstraction.

To model a system at different levels of abstraction by presenting diagrams with different levels of detail,

- Consider the needs of your readers, and start with a given model.
- If your reader is using the model to construct an implementation, she'll need diagrams that are at a lower level of abstraction, which means that they'll need to reveal a lot of detail. If she is using the model to present a conceptual model to an end user, she'll need diagrams that are at a higher level of abstraction, which means that they'll hide a lot of detail.

Basic Structural Modeling

- Depending on where you land in this spectrum of low-to-high levels of abstraction, create a diagram at the right level of abstraction by hiding or revealing the following four categories of things from your model:

    1. Building blocks and relationships: Hide those that are not relevant to the intent of your diagram or the needs of your reader.

    2. Adornments: Reveal only the adornments of these building blocks and relationships that are essential to understanding your intent.

    3. Flow: In the context of behavioral diagrams, expand only those messages or transitions that are essential to understanding your intent.

    4. Stereotypes: In the context of stereotypes used to classify lists of things, such as attributes and operations, reveal only those stereotyped items that are essential to understanding your intent.

Basic Structural Modeling

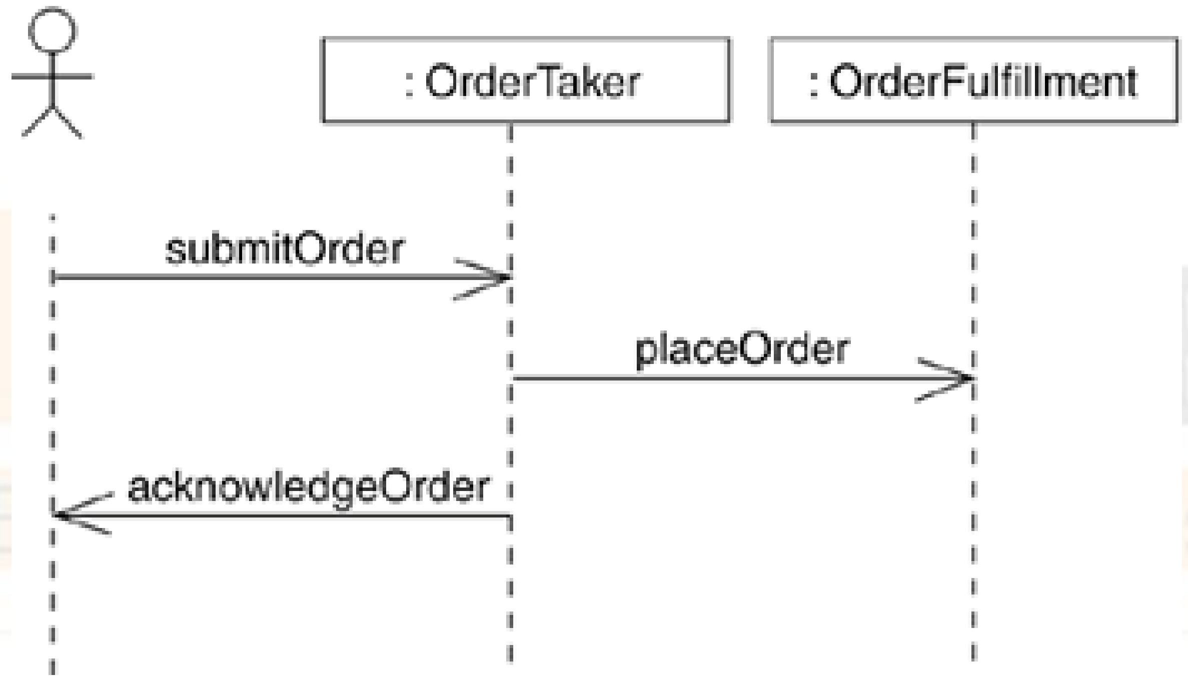
To model a system at different levels of abstraction by creating models at different levels of abstraction,

- Consider the needs of your readers and decide on the level of abstraction that each should view, forming a separate model for each level.
- In general, populate your models that are at a high level of abstraction with simple abstractions and your models that are at a low level of abstraction with detailed abstractions. Establish trace dependencies among the related elements of different models.
- In practice, if you follow the five views of an architecture, there are four common situations you'll encounter when modeling a system at different levels of abstraction:

Basic Structural Modeling

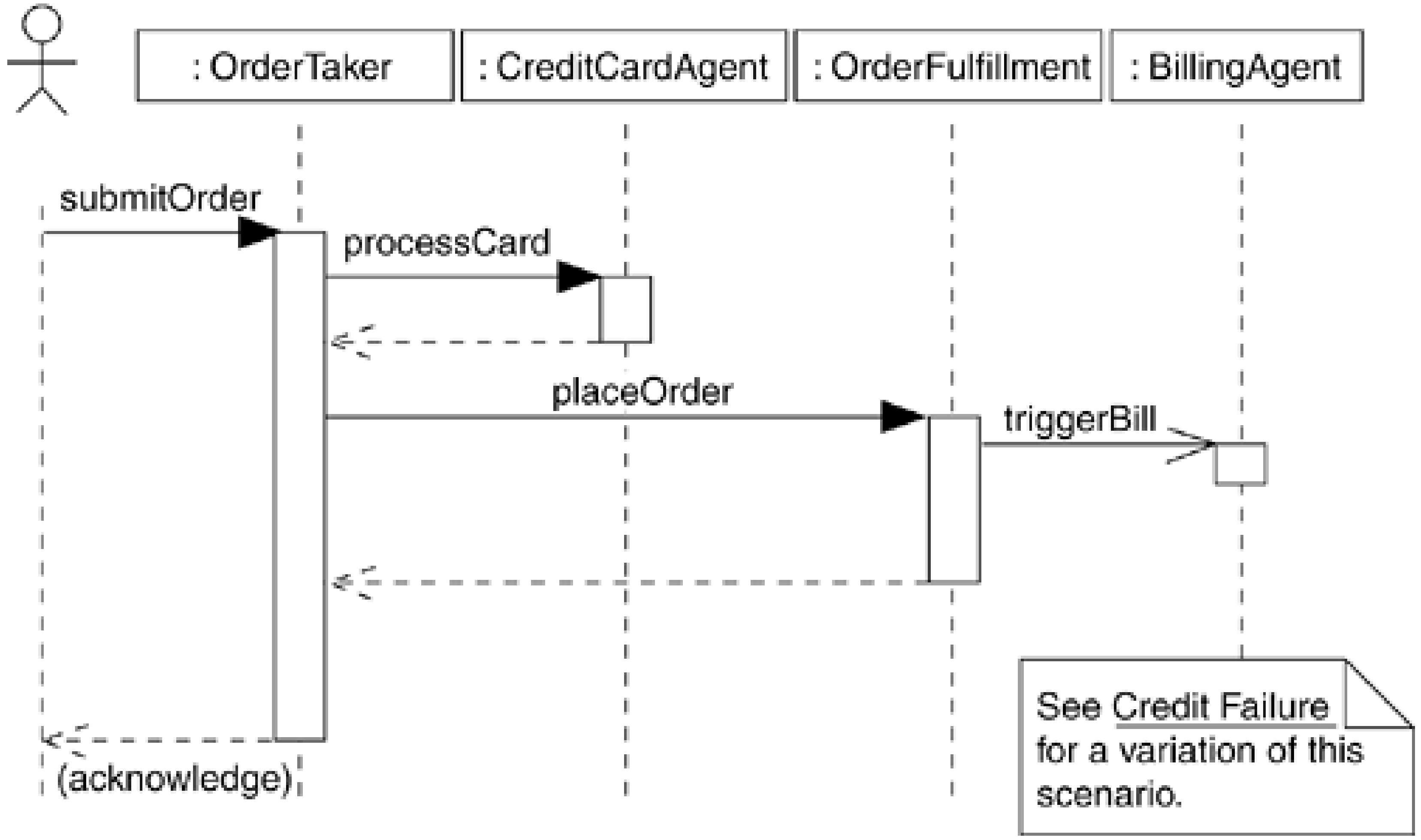
1. Use cases and their realization: Use cases in a use case model will trace to collaborations in a design model.
2. Collaborations and their realization: Collaborations will trace to a society of classes that work together to carry out the collaboration.
3. Components and their design: Components in an implementation model will trace to the elements in a design model.
4. Nodes and their components: Nodes in a deployment model will trace to components in an implementation model.

Basic Structural Modeling



Higher Level of Abstraction

Basic Structural Modeling



Lower level of Abstraction

Basic Structural Modeling

Modeling Complex Views

To model complex views,

- First, convince yourself that there is no meaningful way to present this information at a higher level of abstraction, perhaps eliding some parts of the diagram and retaining the detail in other parts.
- If you've hidden as much detail as you can and your diagram is still complex, consider grouping some of the elements in packages or in higher-level collaborations, then render only those packages or collaborations in your diagram.
- If your diagram is still complex, use notes and color as visual cues to draw the reader's attention to the points you want to make.
- If your diagram is still complex, print it in its entirety and hang it on a convenient large wall. You lose the interactivity that an online version of the diagram brings, but you can step back from the diagram and study it for common patterns.

Basic Structural Modeling

ADVANCED STRUCTURAL MODELING

Terms and Concepts

A ***class diagram*** is a diagram that shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs.

Common Properties

A class diagram is just a special kind of diagram and shares the same common properties as do all other diagrams name and graphical content that are a projection into a model. What distinguishes a class diagram from other kinds of diagrams is its particular content.

Basic Structural Modeling

Common Uses

- You use class diagrams to model the static design view of a system. This view primarily supports the functional requirements of a system. The services of the system should provide to its end users.
- When you model the static design view of a system, you'll typically use class diagrams in one of three ways.

Basic Structural Modeling

1. To model the vocabulary of a system

Modeling the vocabulary of a system involves making a decision about which abstractions are a part of the system under consideration and which fall outside its boundaries. You use class diagrams to specify these abstractions and their responsibilities.

2. To model simple collaborations

A collaboration is a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. For example, when you re modeling the semantics of a transaction in a distributed system, you can't just stare at a single class to understand what's going on. Rather, these semantics are carried out by a set of classes that work together. You use class diagrams to visualize and specify this set of classes and their relationships.

Basic Structural Modeling

3. To model a logical database schema

Think of a schema as the blueprint for the conceptual design of a database. In many domains, you'll want to store persistent information in a relational database or in an object-oriented database. You can model schemas for these databases using class diagrams.

Basic Structural Modeling

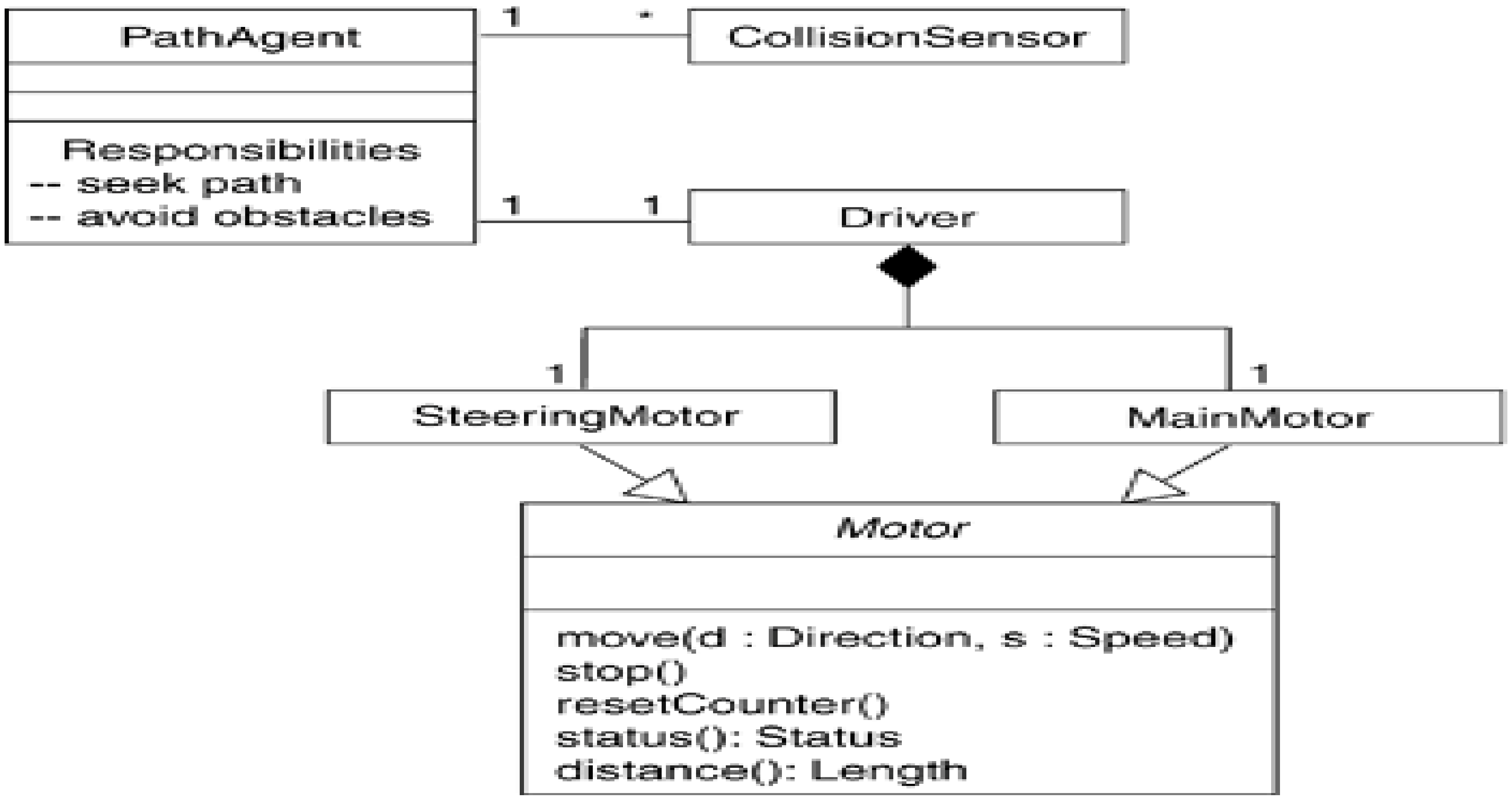
Common Modeling Techniques

Modeling Simple Collaborations

To model a collaboration,

- Identify the mechanism you'd like to model. A mechanism represents some function or behavior of the part of the system you are modeling that results from the interaction of a society of classes, interfaces, and other things.
- For each mechanism, identify the classes, interfaces, and other collaborations that participate in this collaboration. Identify the relationships among these things as well.
- Use scenarios to walk through these things. Along the way, you'll discover parts of your model that were missing and parts that were just plain semantically wrong.
- Be sure to populate these elements with their contents. For classes, start with getting a good balance of responsibilities. Then, over time, turn these into concrete attributes and operations.

Basic Structural Modeling



Basic Structural Modeling

Modeling a Logical Database Schema

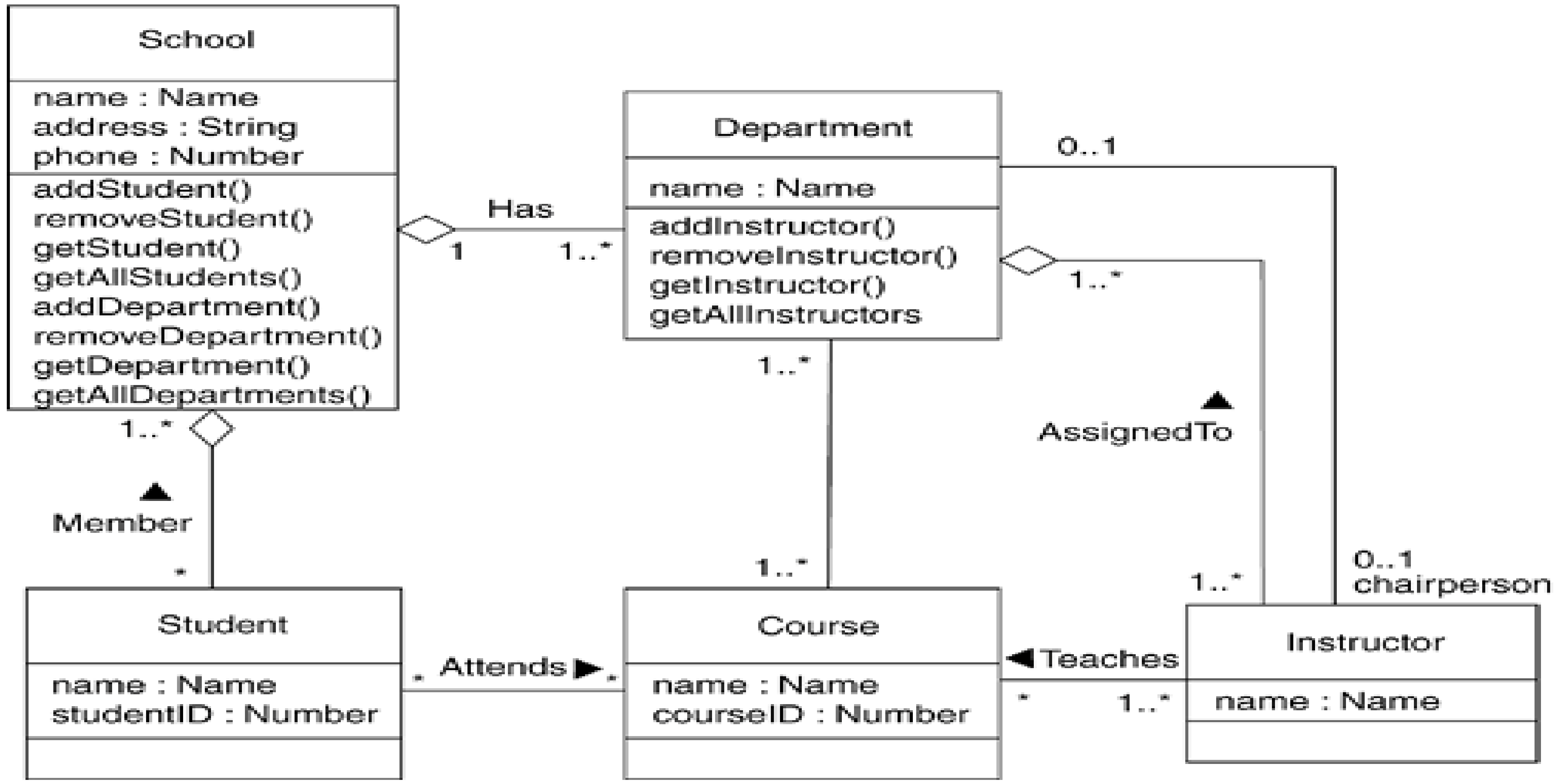
To model a schema,

- Identify those classes in your model whose state must transcend the lifetime of their applications.
- Create a class diagram that contains these classes. You can define your own set of stereotypes and tagged values to address database-specific details.
- Expand the structural details of these classes. In general, this means specifying the details of their attributes and focusing on the associations and their multiplicities that relate these classes.

Basic Structural Modeling

- Watch for common patterns that complicate physical database design, such as cyclic associations and one-to-one associations. Where necessary, create intermediate abstractions to simplify your logical structure.
- Consider also the behavior of these classes by expanding operations that are important for data access and data integrity. In general, to provide a better separation of concerns, business rules concerned with the manipulation of sets of these objects should be encapsulated in a layer above these persistent classes.
- Where possible, use tools to help you transform your logical design into a physical design.

Basic Structural Modeling



Basic Structural Modeling

Forward and Reverse Engineering

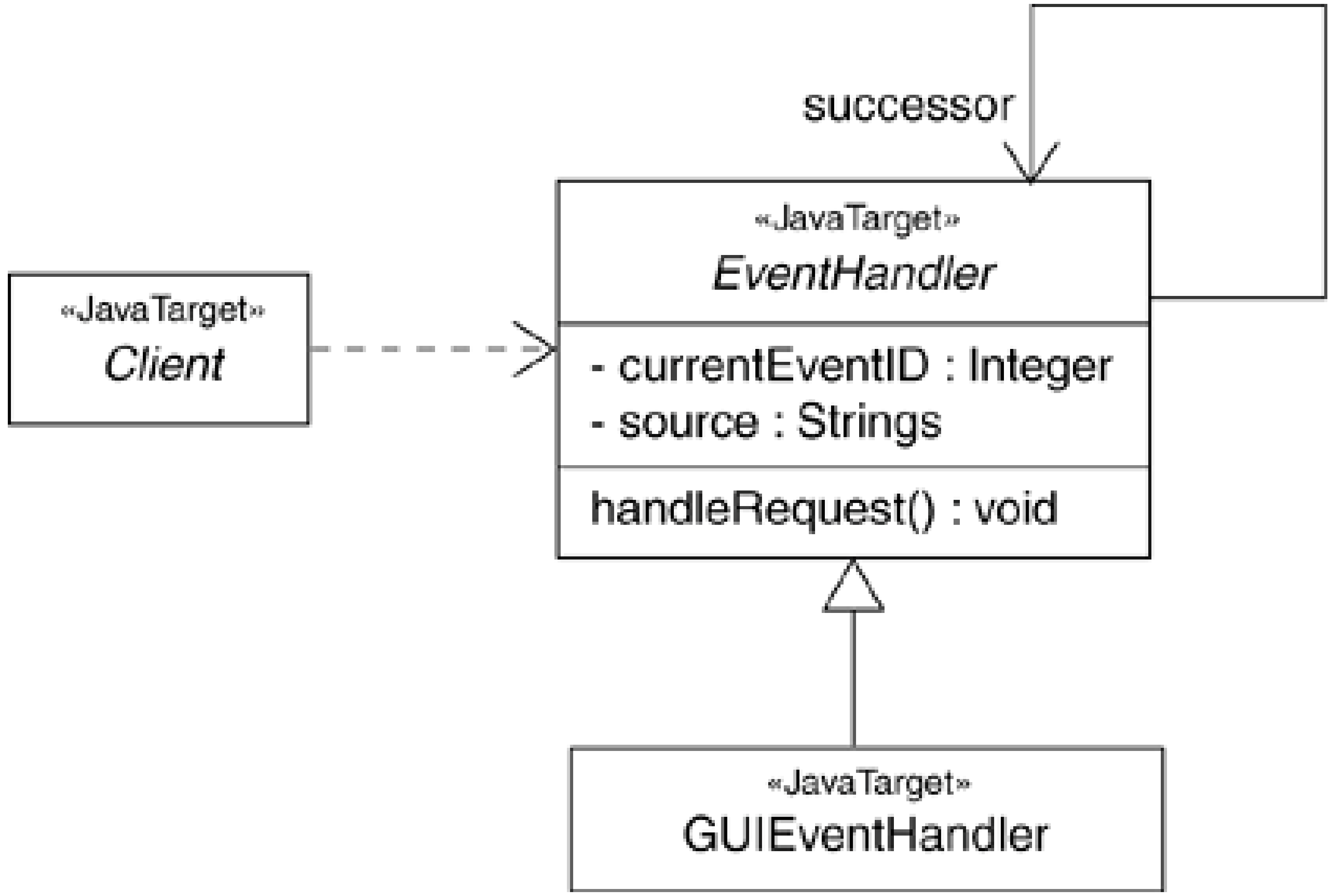
- ***Forward engineering*** is the process of transforming a model into code through a mapping to an implementation language.
- Forward engineering results in a loss of information, because models written in the UML are semantically richer than any current object-oriented programming language.
- In fact, this is a major reason why you need models in addition to code.
- Structural features, such as collaborations, and behavioral features, such as interactions, can be visualized clearly in the UML, but not so clearly from raw code.

Basic Structural Modeling

To forward engineer a class diagram,

- Identify the rules for mapping to your implementation language or languages of choice. This is something you'll want to do for your project or your organization as a whole.
- Depending on the semantics of the languages you choose, you may want to constrain your use of certain UML features. For example, the UML permits you to model multiple inheritance, but Smalltalk permits only single inheritance. You can choose to prohibit developers from modeling with multiple inheritance (which makes your models language-dependent), or you can develop idioms that transform these richer features into the implementation language (which makes the mapping more complex).
- Use tagged values to guide implementation choices in your target language. You can do this at the level of individual classes if you need precise control. You can also do so at a higher level, such as with collaborations or packages.
- Use tools to generate code.

Basic Structural Modeling



Basic Structural Modeling

All of these classes specify a mapping to Java, as noted in their stereotype. Forward engineering the classes in this diagram to Java is straightforward, using a tool. Forward engineering the class EventHandler yields the following code.

```
public abstract class EventHandler{
        EventHandler successor;
        private Integer currentEventID;
        private String source;

        EventHandler() {}
        public void handleRequest() {}
}
```

Basic Structural Modeling

- ***Reverse engineering*** is the process of transforming code into a model through a mapping from a specific implementation language.
- Reverse engineering results in a flood of information, some of which is at a lower level of detail than you'll need to build useful models.
- At the same time, reverse engineering is incomplete. There is a loss of information when forward engineering models into code, and so you can't completely recreate a model from code unless your tools encode information in the source comments that goes beyond the semantics of the implementation language.

Basic Structural Modeling

To reverse engineer a class diagram,

- Identify the rules for mapping from your implementation language or languages of choice. This is something you'll want to do for your project or your organization as a whole.
- Using a tool, point to the code you'd like to reverse engineer. Use your tool to generate a new model or modify an existing one that was previously forward engineered. It is unreasonable to expect to reverse engineer a single concise model from a large body of code. You need to select portion of the code and build the model from the bottom.

Basic Structural Modeling

- Using your tool, create a class diagram by querying the model. For example, you might start with one or more classes, then expand the diagram by following specific relationships or other neighboring classes. Expose or hide details of the contents of this class diagram as necessary to communicate your intent.
- Manually add design information to the model to express the intent of the design that is missing or hidden in the code.

Basic Structural Modeling

1. Advanced Classes

Terms and Concepts

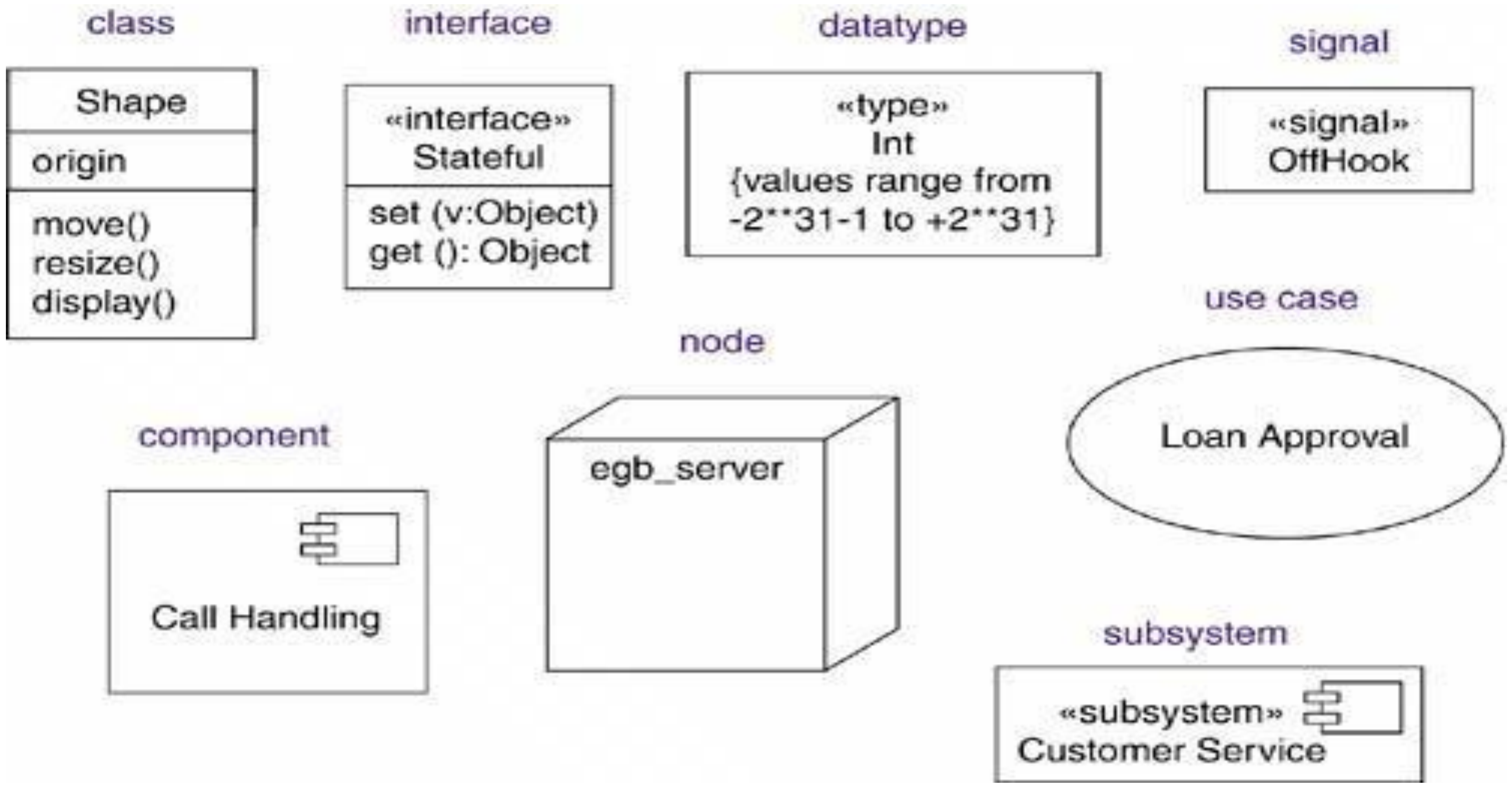
- A ***classifier*** is a mechanism that describes structural and behavioral features. Classifiers include classes, associations, interfaces, data-types, signals, components, nodes, use cases, and subsystems.

- The most important kind of classifier in the UML is the class. A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. Classes are not the only kind of classifier, however. The UML provides a number of other kinds of classifiers to help you model.

Basic Structural Modeling

Interface	A collection of operations that are used to specify a service of a class or a component
Datatype	A type whose values are immutable, including primitive built-in types (such as numbers and strings) as well as enumeration types (such as Boolean)
Association	A description of a set of links, each of which relates two or more objects.
Signal	The specification of an asynchronous message communicated between instances
Component	A modular part of a system that hides its implementation behind a set of external interfaces
Node	A physical element that exists at run time and that represents a computational resource, generally having at least some memory and often processing capability
Use case	A description of a set of a sequence of actions, including variants, that a system performs that yields an observable result of value to a particular actor
Subsystem	A component that represents a major part of a system

Basic Structural Modeling



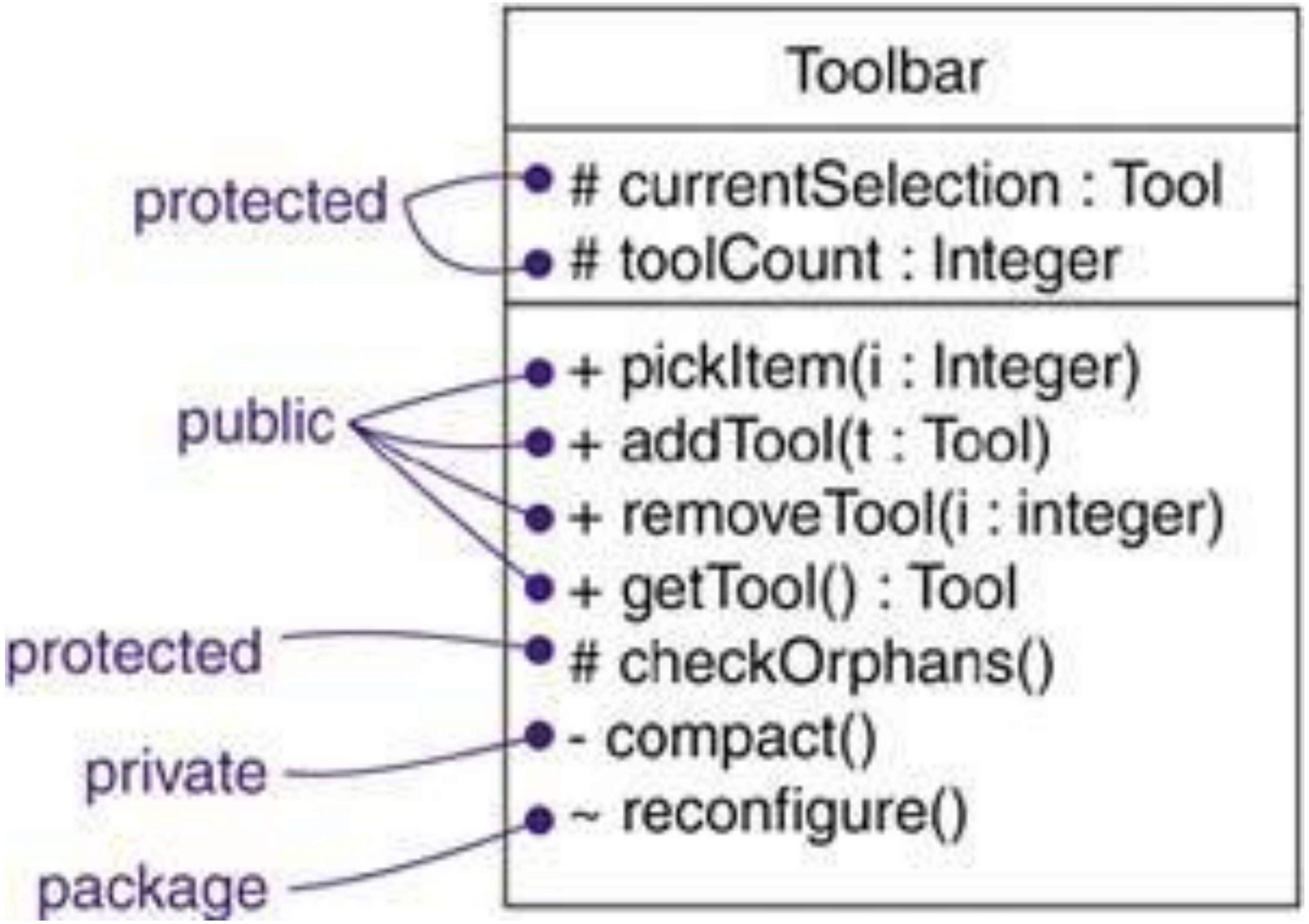
Basic Structural Modeling

Visibility

One of the design details you can specify for an attribute or operation is visibility. The visibility of a feature specifies whether it can be used by other classifiers. In the UML, you can specify any of four levels of visibility.

1. Public	Any outside classifier with visibility to the given classifier can use the feature; specified by prepending the symbol +.
2. Protected	Any descendant of the classifier can use the feature; specified by prepending the symbol #.
3. Private	Only the classifier itself can use the feature; specified by prepending the symbol -.
4. Package	Only classifiers declared in the same package can use the feature; specified by prepending the symbol ~.

Basic Structural Modeling

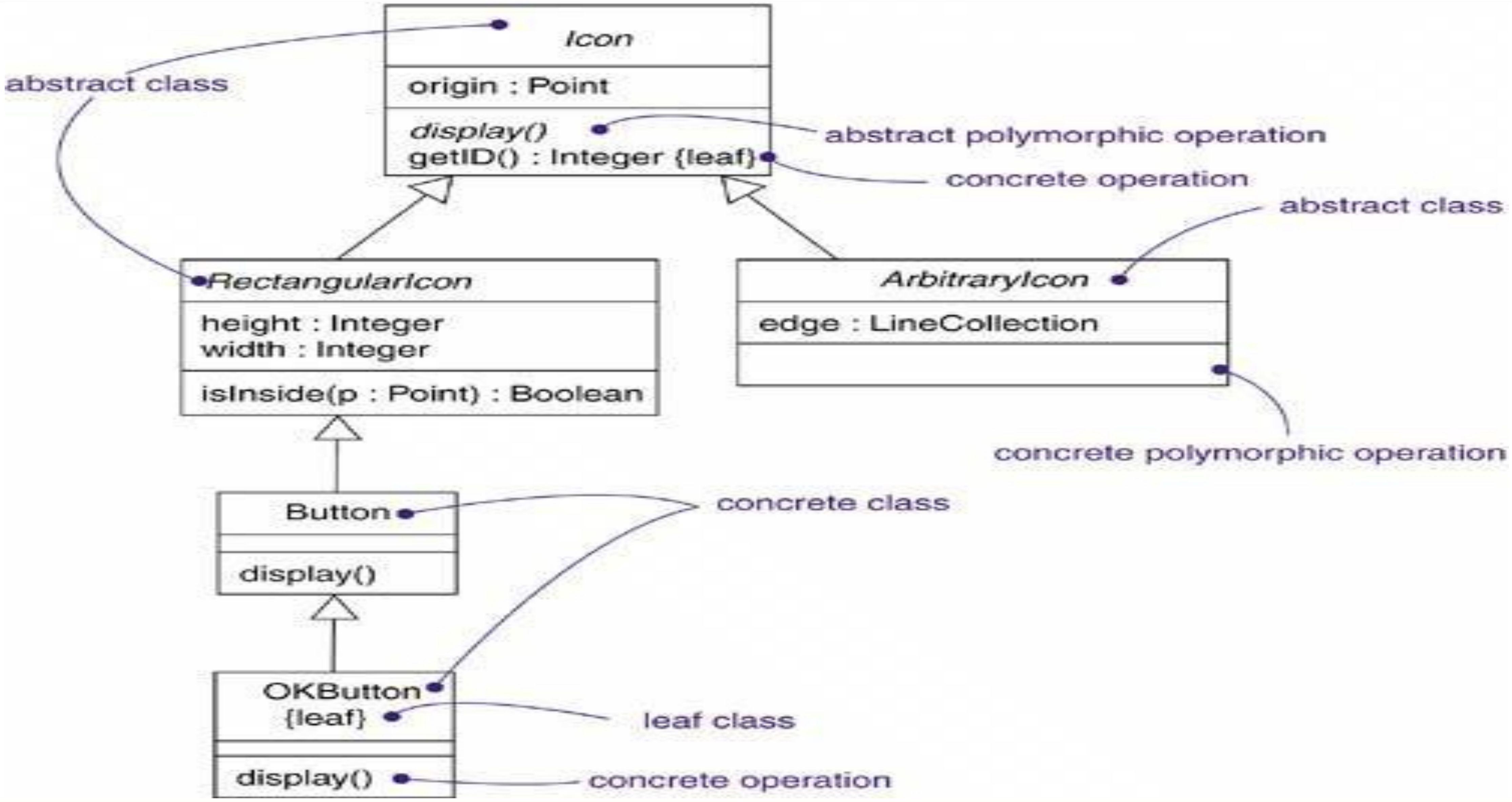


Basic Structural Modeling

Abstract, Leaf, and Polymorphic Elements

- We use generalization relationships to model a lattice of classes, with more-generalized abstractions at the top of the hierarchy and more-specific ones at the bottom.
- Within these hierarchies, it's common to specify that certain classes are abstract meaning that they may not have any direct instances.
- In the UML, we specify that a class is abstract by writing its name in italics.
- Icon, RectangularIcon, and ArbitraryIcon are all abstract classes.
- By contrast, a concrete class (such as Button and OKButton) may have direct instances.

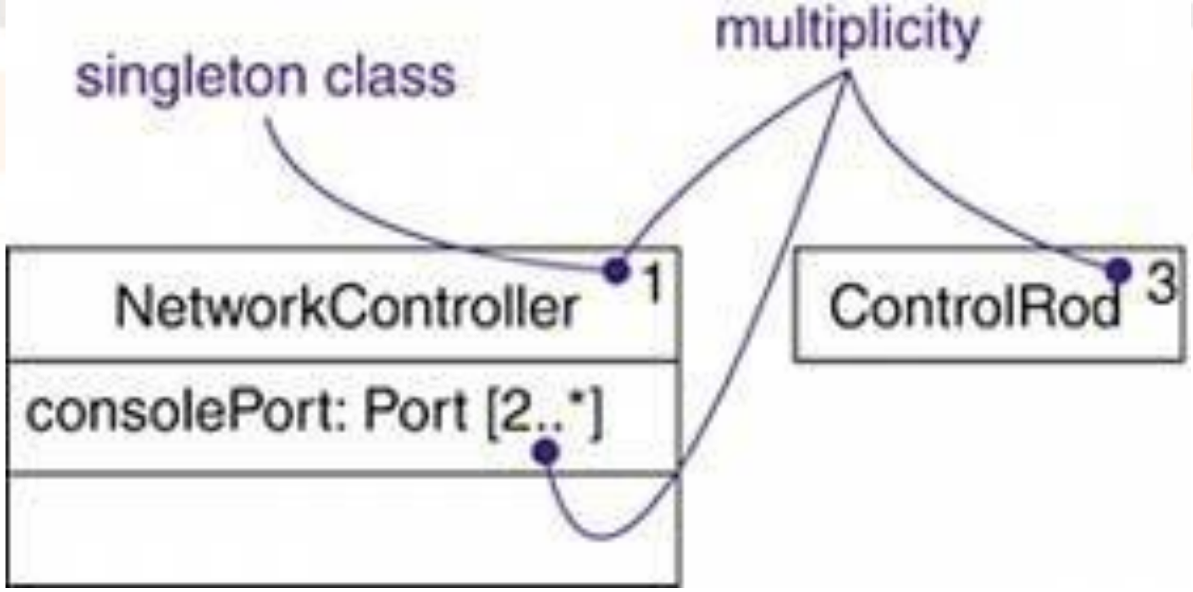
Basic Structural Modeling



Basic Structural Modeling

Multiplicity

Whenever we use a class, it's reasonable to assume that there may be any number of instances of that class (unless, of course, it is an abstract class and may not have any direct instances, although there may be any number of instances of its concrete children).



Basic Structural Modeling

Attributes

At the most abstract level, when you model a class's structural features (that is, its attributes), you simply write each attribute's name.

visibility] name

[':' type] ['[' multiplicity] ']']

['=' initial-value]

[property-string {',' property-string}]

Basic Structural Modeling

For example, the following are all legal attribute declarations:

Origin	Name only
+ origin	Visibility and name
origin : Point	Name and type
name : String[0..1]	Name, type, and multiplicity
origin : Point = (0,0)	Name, type, and initial value
id: Integer {readonly}	Name and property

Basic Structural Modeling

Operations

At the most abstract level, when we model a class's behavioral features, we can also specify the parameters, return type, concurrency semantics, and other properties of each operation. Collectively, the name of an operation plus its parameters (including its return type, if any) is called the operation's signature.

[visibility] name ['(' parameter-list ')']

[':' return-type]

[property-string {',' property-string}]

Basic Structural Modeling

- For example, the following are all legal operation declarations:

display	Name only
+ display	Visibility and name
set(n : Name, s : String)	Name and parameters
getID() : Integer	Name and return type
restart() {guarded}	Name and property

Basic Structural Modeling

- In an operation's signature, you may provide zero or more parameters, each of which follows the syntax

[direction] name : type [= default-value]

- Direction may be any of the following values:

in	An input parameter; may not be modified
out	An output parameter; may be modified to communicate information to the caller
inout	An input parameter; may be modified to communicate information to the caller

Basic Structural Modeling

In addition to the leaf and abstract properties described earlier, there are defined properties that you can use with operations.

1. query	Execution of the operation leaves the state of the system unchanged. In other words, the operation is a pure function that has no side effects.
2. sequential	Callers must coordinate outside the object so that only one flow is in the object at a time. In the presence of multiple flows of control, the semantics and integrity of the object cannot be guaranteed.
3. guarded	The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by sequentializing all calls to all of the object's guarded operations. In effect, exactly one operation at a time can be invoked on the object, reducing this to sequential semantics.

Basic Structural Modeling

4. concurrent	The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by treating the operation as atomic. Multiple calls from concurrent flows of control may occur simultaneously to one object on any concurrent operation, and all may proceed concurrently with correct semantics; concurrent operations must be designed so that they perform correctly in case of a concurrent sequential or guarded operation on the same object.
5. static	The operation does not have an implicit parameter for the target object; it behaves like a traditional global procedure.

Basic Structural Modeling

Template Classes

- A template is a parameterized element. In such languages as C++ and Ada, you can write template classes, each of which defines a family of classes.
- A template may include slots for classes, objects, and values, and these slots serve as the template's parameters. You can't use a template directly; you have to instantiate it first. Instantiation involves binding these formal template parameters to actual ones. For a template class, the result is a concrete class that can be used just like any ordinary class.
- The most common use of template classes is to specify containers that can be instantiated for specific elements, making them type-safe. For example, the following C++ code fragment declares a parameterized Map class.

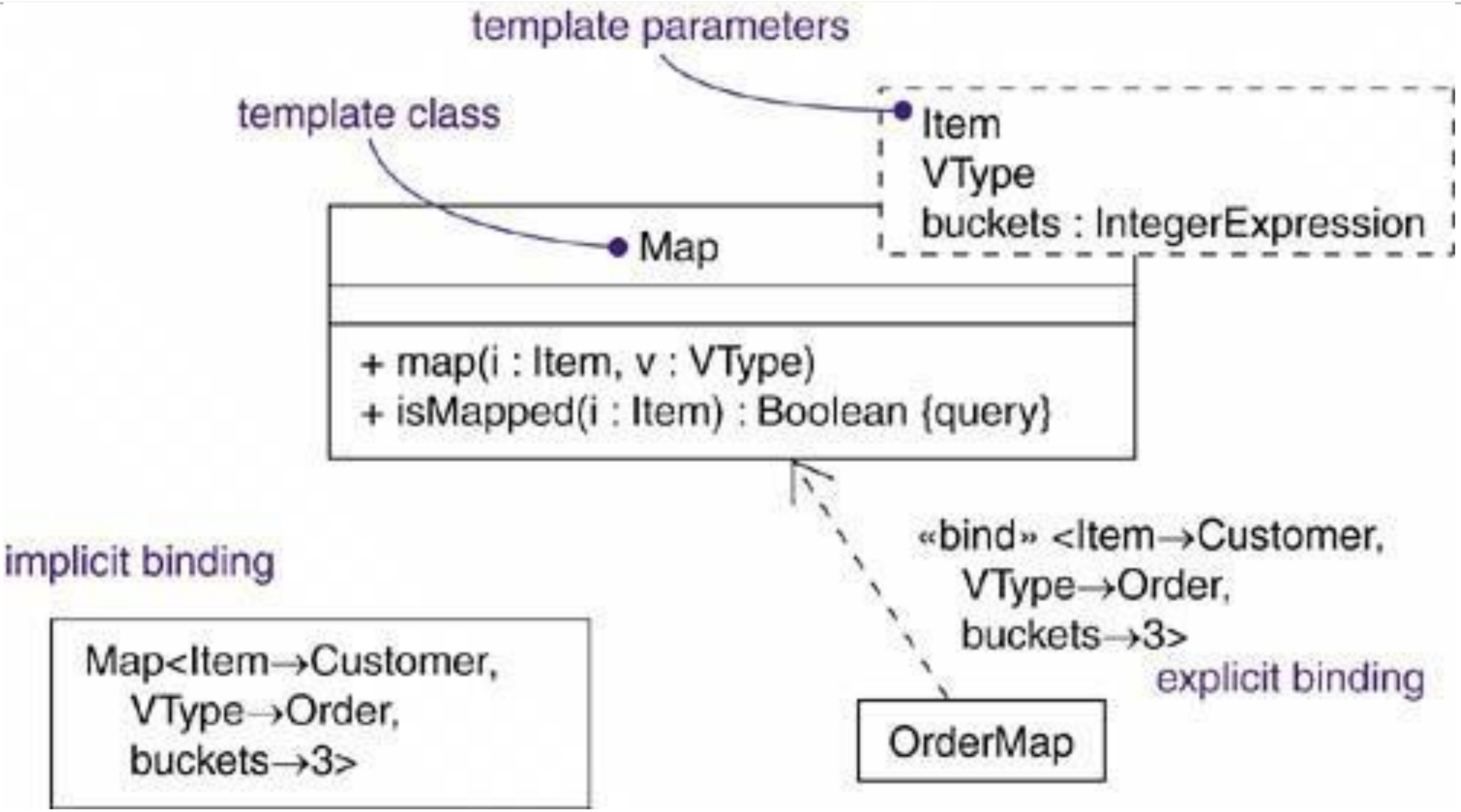
Basic Structural Modeling

```
template<class Item, class VType, int Buckets>
class Map {
        public:
        virtual map(const Item&, const VType&);
        virtual Boolean isMappen(const Item&) const;
        ...
};
```

You might then instantiate this template to map Customer objects to Order objects.

m : Map<Customer, Order, 3>;

Basic Structural Modeling



Basic Structural Modeling

Standard Elements

All of the UML's extensibility mechanisms apply to classes.

The UML defines four standard stereotypes that apply to classes.

1. metaclass	Specifies a classifier whose objects are all classes
2. powertype	Specifies a classifier whose objects are classes that are the children of a given parent class
3.stereotype	Specifies that the classifier is a stereotype that may be applied to other elements
4. utility	Specifies a class whose attributes and operations are all static scoped

Basic Structural Modeling

Common Modeling Techniques

Modeling the Semantics of a Class

To model the semantics of a class, choose among the following possibilities, arranged from informal to formal.

- Specify the responsibilities of the class. A responsibility is a contract or obligation of a type or class and is rendered in a note attached to the class, or in an extra compartment in the class icon.
- Specify the semantics of the class as a whole using structured text, rendered in a note (stereotyped as semantics) attached to the class.
- Specify the body of each method using structured text or a programming language, rendered in a note attached to the operation by a dependency relationship.

Basic Structural Modeling

- Specify the pre- and post-conditions of each operation, plus the invariants of the class as a whole, using structured text. These elements are rendered in notes (stereotyped as precondition, postcondition, and invariant) attached to the operation or class by a dependency relationship.
- Specify a state machine for the class. A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.
- Specify internal structure of the class.
- Specify a collaboration that represents the class. A collaboration is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. A collaboration has a structural part as well as a dynamic part, so you can use collaborations to specify all dimensions of a class's semantics.
- Specify the pre- and postconditions of each operation, plus the invariants of the class as a whole, using a formal language such as OCL.

Basic Structural Modeling

Advanced Relationships

Terms and Concepts

- A ***relationship*** is a connection among things. In object-oriented modeling, the four most important relationships are *dependencies*, *generalizations*, *associations*, and *realizations*. Graphically, a relationship is rendered as a path, with different kinds of lines used to distinguish the different relationships.

Dependencies

- A ***dependency*** is a using relationship, specifying that a change in the specification of one thing (for example, class SetTopController) may affect another thing that uses it (for example, class ChannelIterator), but not the reverse. Graphically, a dependency is rendered as a dashed line, directed to the thing that is depended on. Apply dependencies when you want to show one thing using another.

Basic Structural Modeling

The UML defines a number of stereotypes that may be applied to dependency relationships. There are a number of stereotypes, which can be organized into several groups.

First, there are stereotypes that apply to dependency relationships among classes and objects in class diagrams.

1. bind	Specifies that the source instantiates the target template using the given actual parameters
2.derive	Specifies that the source may be computed from the target
3.permit	Specifies that the source is given special visibility into the target

Basic Structural Modeling

4.instanceOf	Specifies that the source object is an instance of the target classifier. Ordinarily shown using text notation in the form source : Target
5.instantiate	Specifies that the source creates instances of the target
6.powertype	Specifies that the target is a powertype of the source; a powertype is a classifier whose objects are the children of a given parent
7. refine	Specifies that the source is at a finer degree of
8. use	Specifies that the semantics of the source element depends on the semantics of the public part of the target

Basic Structural Modeling

There are two stereotypes that apply to dependency relationships among packages.

1.import	Specifies that the public contents of the target package enter the public namespace of the source, as if they had been declared in the source.
2.access	Specifies that the public contents of the target package enter the private namespace of the source. The unqualified names may be used within the source, but they may not be re-exported.

Two stereotypes apply to dependency relationships among use cases:

1. extend	Specifies that the target use case extends the behavior of the source
2.include	Specifies that the source use case explicitly incorporates the behavior of another use case at a location specified by the source

Basic Structural Modeling

One stereotype you'll encounter in the context of interactions among objects is

1. send	Specifies that the source class sends the target event

Finally, one stereotype that you'll encounter in the context of organizing the elements of your system into subsystems and models is

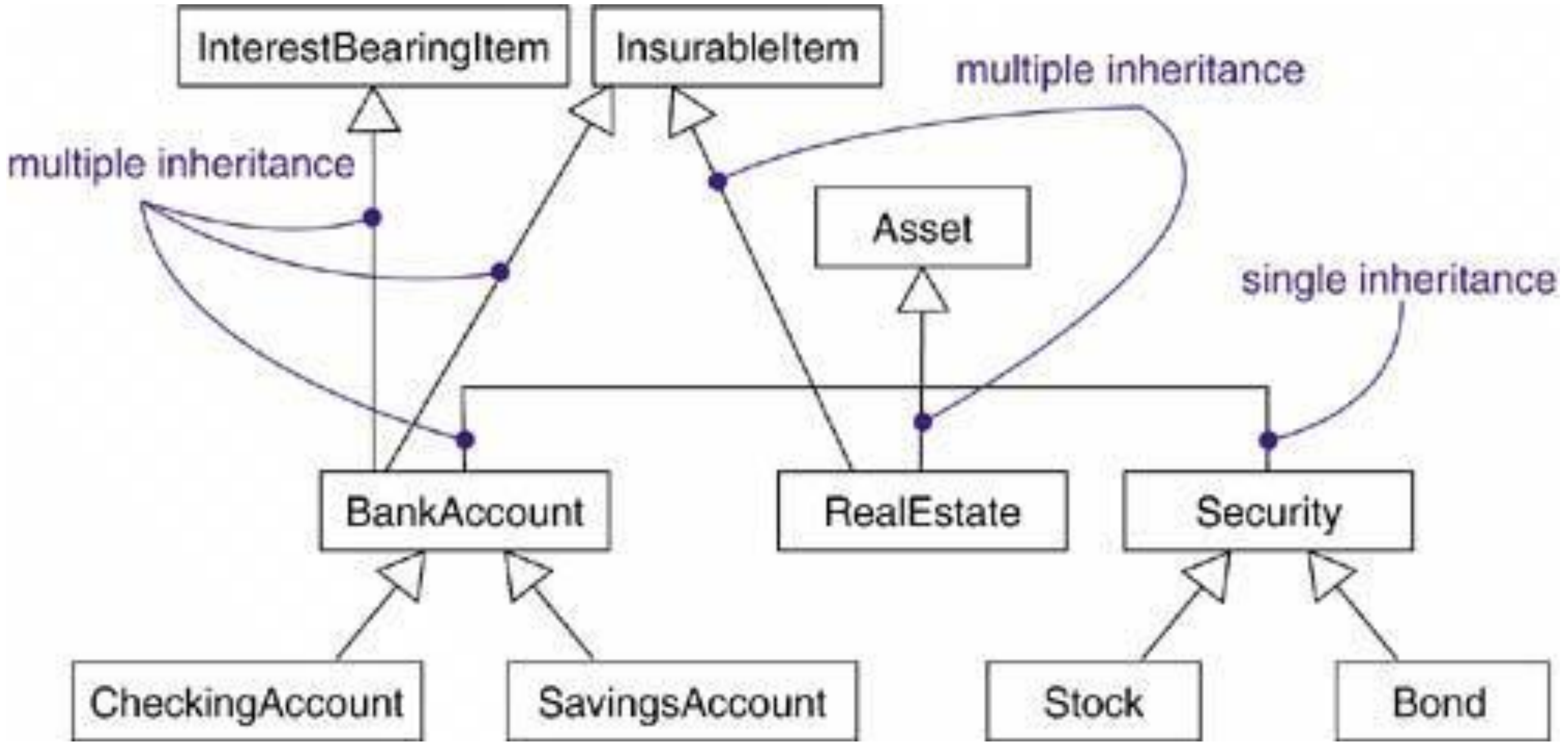
1.TRace	Specifies that the target is a historical predecessor of the source from an earlier stage of development

Basic Structural Modeling

Generalizations

A ***generalization*** is a relationship between a general classifier (called the superclass or parent) and a more specific classifier (called the subclass or child). For example, you might encounter the general class Window with its more specific subclass, MultiPaneWindow. With a generalization relationship from the child to the parent, the child (MultiPaneWindow) will inherit all the structure and behavior of the parent (Window).

Basic Structural Modeling



Basic Structural Modeling

A plain, unadorned generalization relationship is sufficient for most of the inheritance relationships you'll encounter. However, if you want to specify a shade of meaning, the UML defines four constraints that may be applied to generalization relationships:

1. complete	Specifies that all children in the generalization have been specified in the model (although some may be elided in the diagram) and that no additional children are permitted
2.incomplete	Specifies that not all children in the generalization have been specified (even if some are elided) and that additional children are permitted
3. disjoint	Specifies that objects of the parent may have no more than one of the children as a type. For example, class Person can be specialized into disjoint classes Man and Woman.
4.overlapping	Specifies that objects of the parent may have more than one of the children as a type. For example, class Vehicle can be specialized into overlapping subclasses LandVehicle and WaterVehicle (an amphibious vehicle is both).

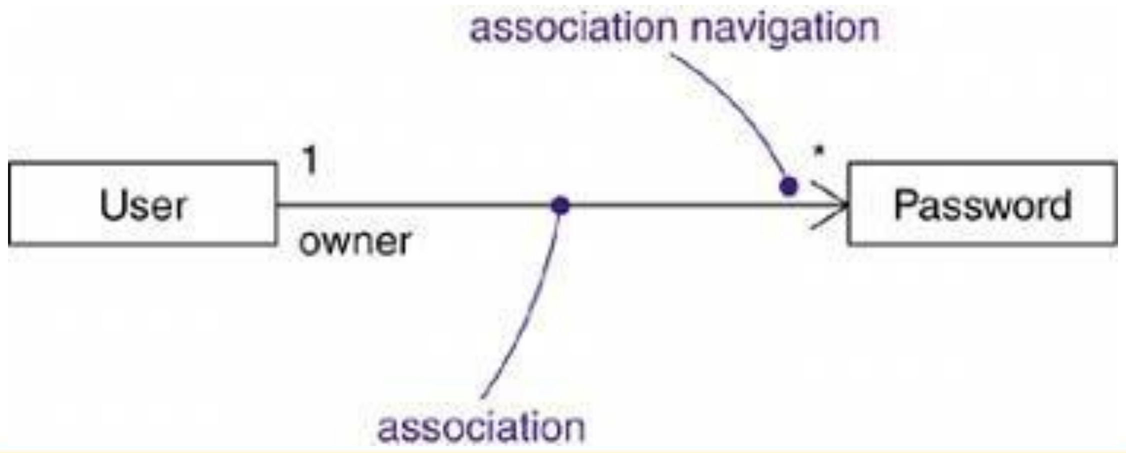
Basic Structural Modeling

Associations

An ***association*** is a structural relationship, specifying that objects of one thing are connected to objects of another. For example, a Library class might have a one-to-many association to a Book class, indicating that each Book instance is owned by one Library instance.

Navigation

Given a plain, unadorned association between two classes, such as Book and Library, it's possible to navigate from objects of one kind to objects of the other kind. Unless otherwise specified, navigation across an association is bidirectional.

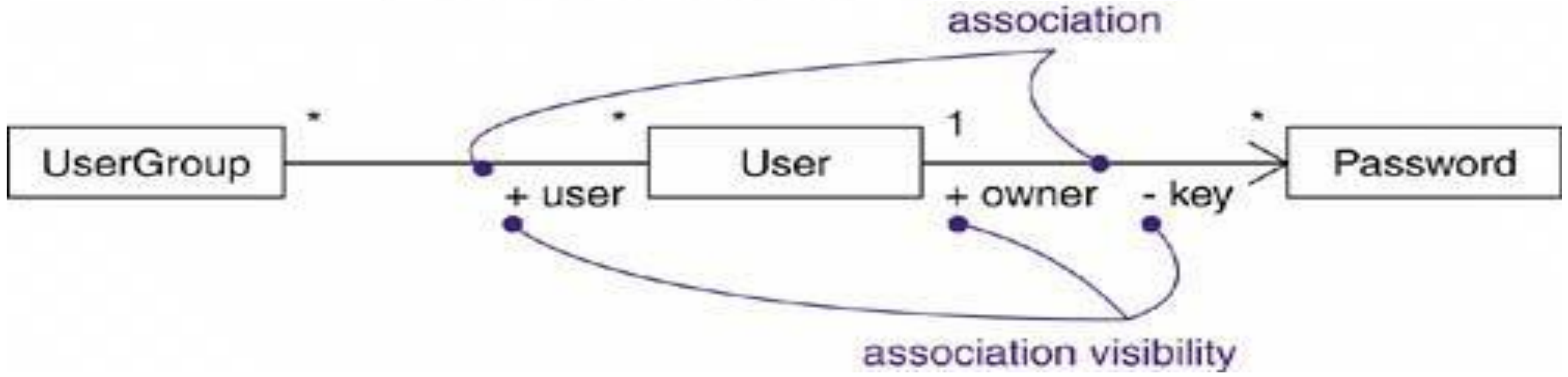


Basic Structural Modeling

Visibility

Given an association between two classes, objects of one class can see and navigate to objects of the other unless otherwise restricted by an explicit statement of navigation. However, there are circumstances in which you'll want to limit the visibility across that association relative to objects outside the association.

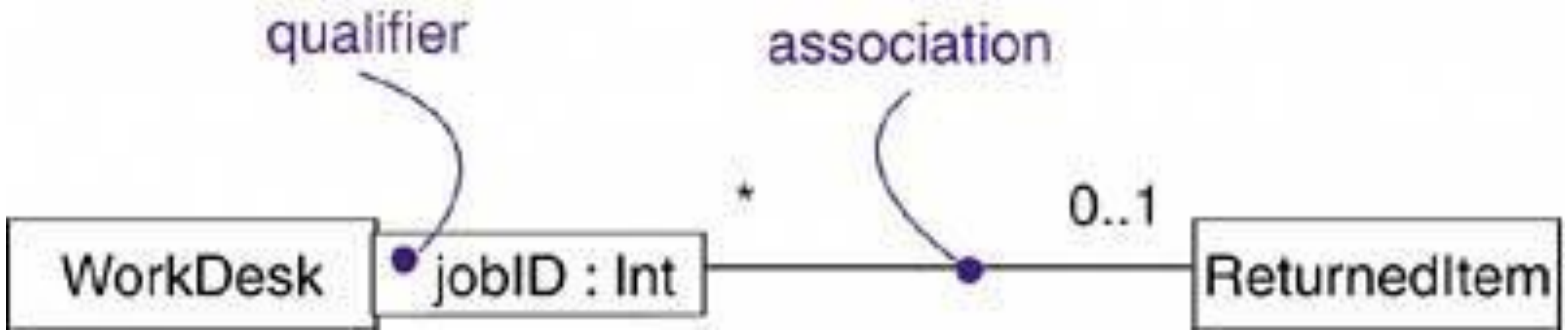
There is an association between UserGroup and User and another between User and Password. Given a User object, it's possible to identify its corresponding Password objects. However, a Password is private to a User, so it shouldn't be accessible from the outside (unless, of course, the User explicitly exposes access to the Password,



Basic Structural Modeling

Qualification

In the context of an association, one of the most common modeling idioms you'll encounter is the problem of lookup. Given an object at one end of an association, how do you identify an object or set of objects at the other end? For example, consider the problem of modeling a work desk at a manufacturing site at which returned items are processed to be fixed

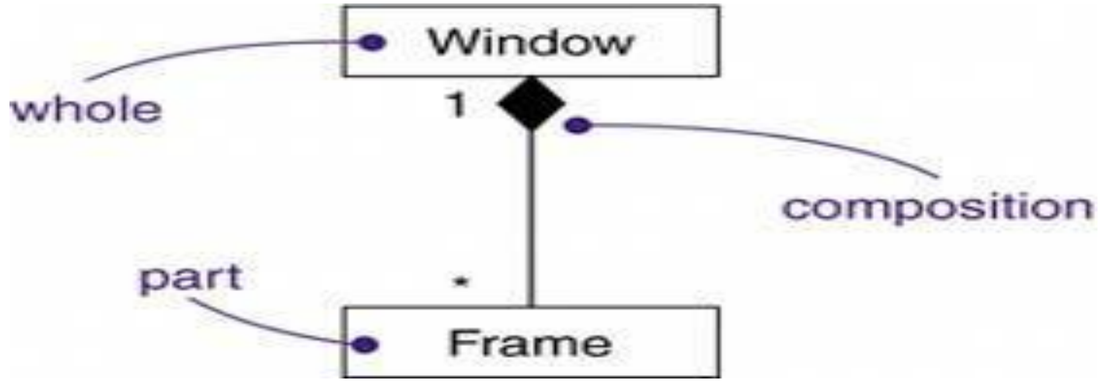


Basic Structural Modeling

Composition

Aggregation turns out to be a simple concept with some fairly deep semantics. Simple aggregation is entirely conceptual and does nothing more than distinguish a "whole" from a "part." Simple aggregation does not change the meaning of navigation across the association between the whole and its parts, nor does it link the lifetimes of the whole and its parts.

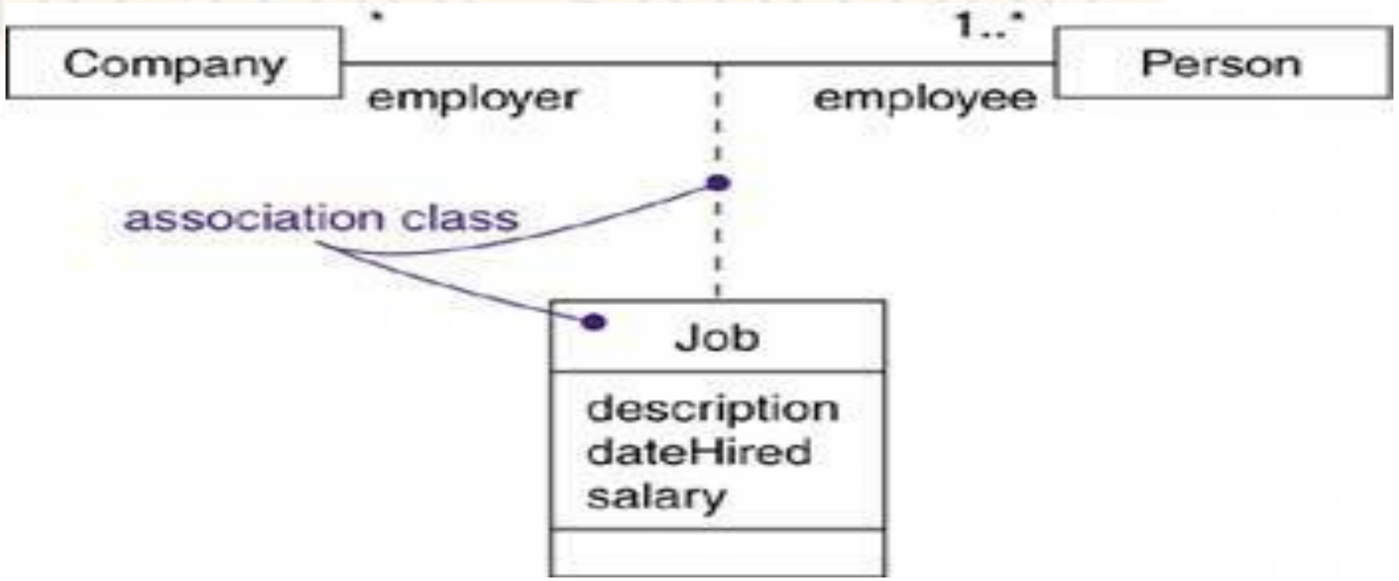
In a composite aggregation, an object may be a part of only one composite at a time. For example, in a windowing system, a Frame belongs to exactly one Window. This is in contrast to simple aggregation, in which a part may be shared by several wholes. For example, in the model of a house, a Wall may be a part of one or more Room objects.



Basic Structural Modeling

Association Classes

In an association between two classes, the association itself might have properties. For example, in an employer/employee relationship between a Company and a Person, there is a Job that represents the properties of that relationship that apply to exactly one pairing of the Person and Company. It wouldn't be appropriate to model this situation with a Company to Job association together with a Job to Person association.



Basic Structural Modeling

Constraints

These simple and advanced properties of associations are sufficient for most of the structural relationships you'll encounter. However, if you want to specify a shade of meaning, the UML defines five constraints that may be applied to association relationships. First, you can specify whether the objects at one end of an association (with a multiplicity greater than one) are ordered or unordered.

1. ordered	Specifies that the set of objects at one end of an association are in an explicit order.
2. set	The objects are unique with no duplicates.
3. bag	The objects are non-unique, may be duplicates.
4. ordered set	The objects are unique but ordered.
5. list or sequence	The objects are ordered, may be duplicates.
6. readonly	A link, once added from an object on the opposite end of the association, may not be modified or deleted. The default in the absence of this constraint is unlimited changeability.

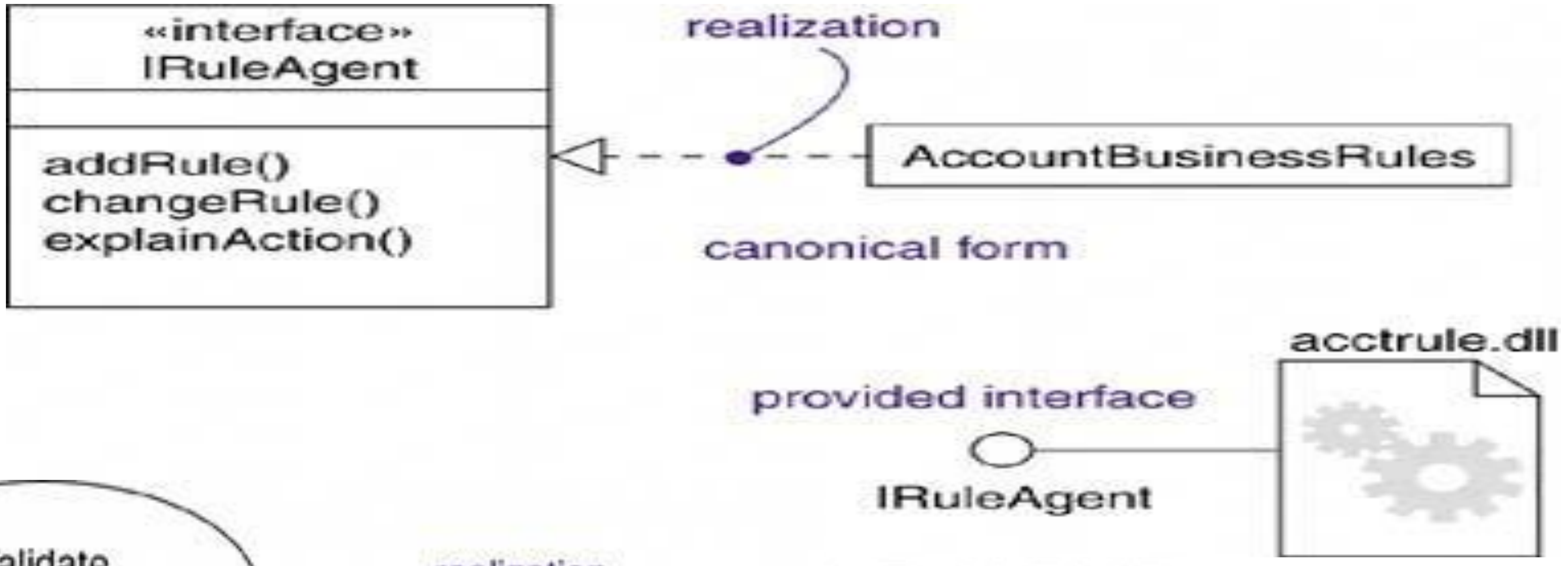
Basic Structural Modeling

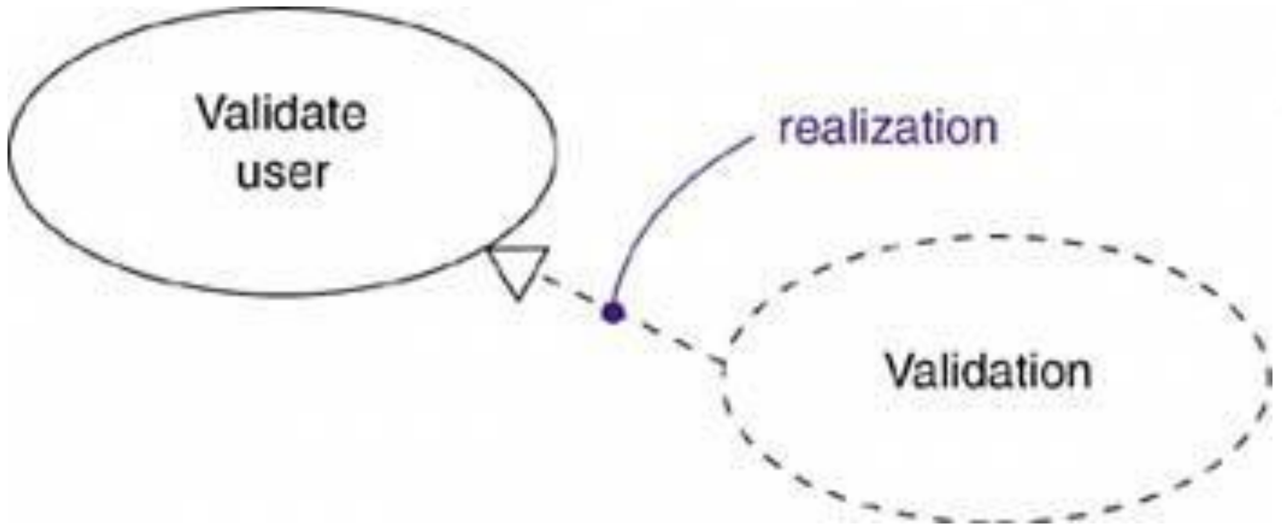
Realizations

A realization is a semantic relationship between classifiers in which one classifier specifies a contract that another classifier guarantees to carry out. Graphically, a realization is rendered as a dashed directed line with a large open arrowhead pointing to the classifier that specifies the contract.

Realization is different enough from dependency, generalization, and association relationships that it is treated as a separate kind of relationship. Semantically, realization is somewhat of a cross between dependency and generalization, and its notation is a combination of the notation for dependency and generalization.

Basic Structural Modeling





Basic Structural Modeling

Topic (Times New Roman Bold-24pt)

Explanation (Times New Roman-20pt- Line spacing 1.5pt)

Basic Structural Modeling

Topic (Times New Roman Bold-24pt)

Explanation (Times New Roman-20pt- Line spacing 1.5pt)

Basic Structural Modeling

Topic (Times New Roman Bold-24pt)

Explanation (Times New Roman-20pt- Line spacing 1.5pt)

Basic Structural Modeling

Explanation (Times New Roman-20pt- Line spacing 1.5pt)

Basic Structural Modeling

Topic (Times New Roman Bold-24pt)

Explanation (Times New Roman-20pt- Line spacing 1.5pt)

Subject (Helvetica Bold- 24pt)

Self Assessment Question

1. Question (Times New Roman-20pt- Line spacing 1.5pt)

   a. Options

   b. Options

   c. Options

   d. Options

   Answer: Options

Subject (Helvetica Bold- 24pt)

Document Link

Topic	URL	Notes
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT

Subject (Helvetica Bold- 24pt)

Video Link

Topic	URL	Notes
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT

Subject (Helvetica Bold- 24pt)

E- Book Link

Ebook name	Chapter	Page No.	Notes	URL
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT
Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT	Times New Roman-14PT