UML (Unified Modeling Language)

5 Advanced Classes

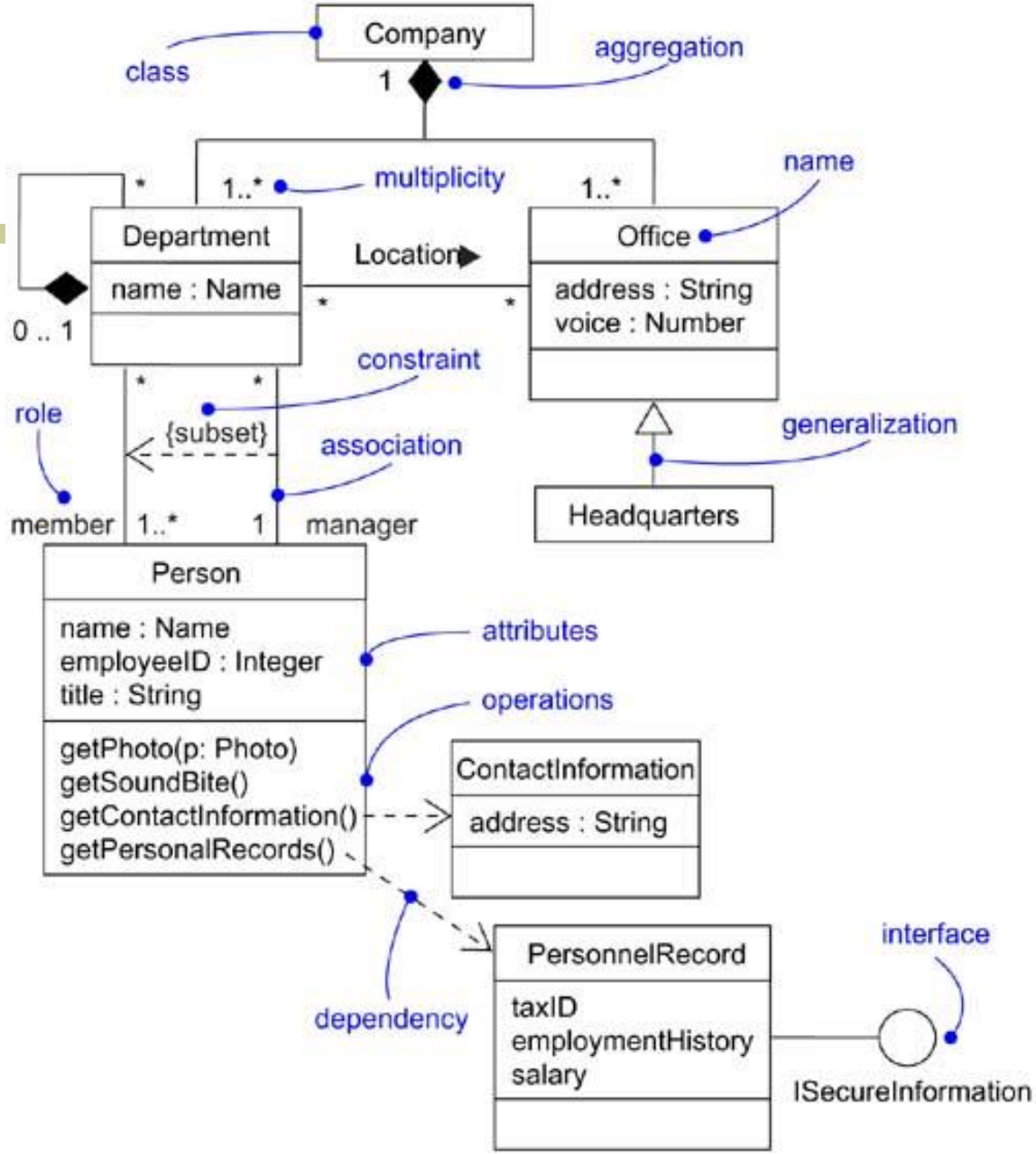
Software Institute of
Nanjing University

1. Class Diagrams

- A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
- Class diagrams is used to model the static design view of a system.

1.1 Common Properties

- A class diagram is just a special kind of diagram : a name and graphical content that are a projection into a model.
- What distinguishes a class diagram from all other kinds of diagrams is its particular content.

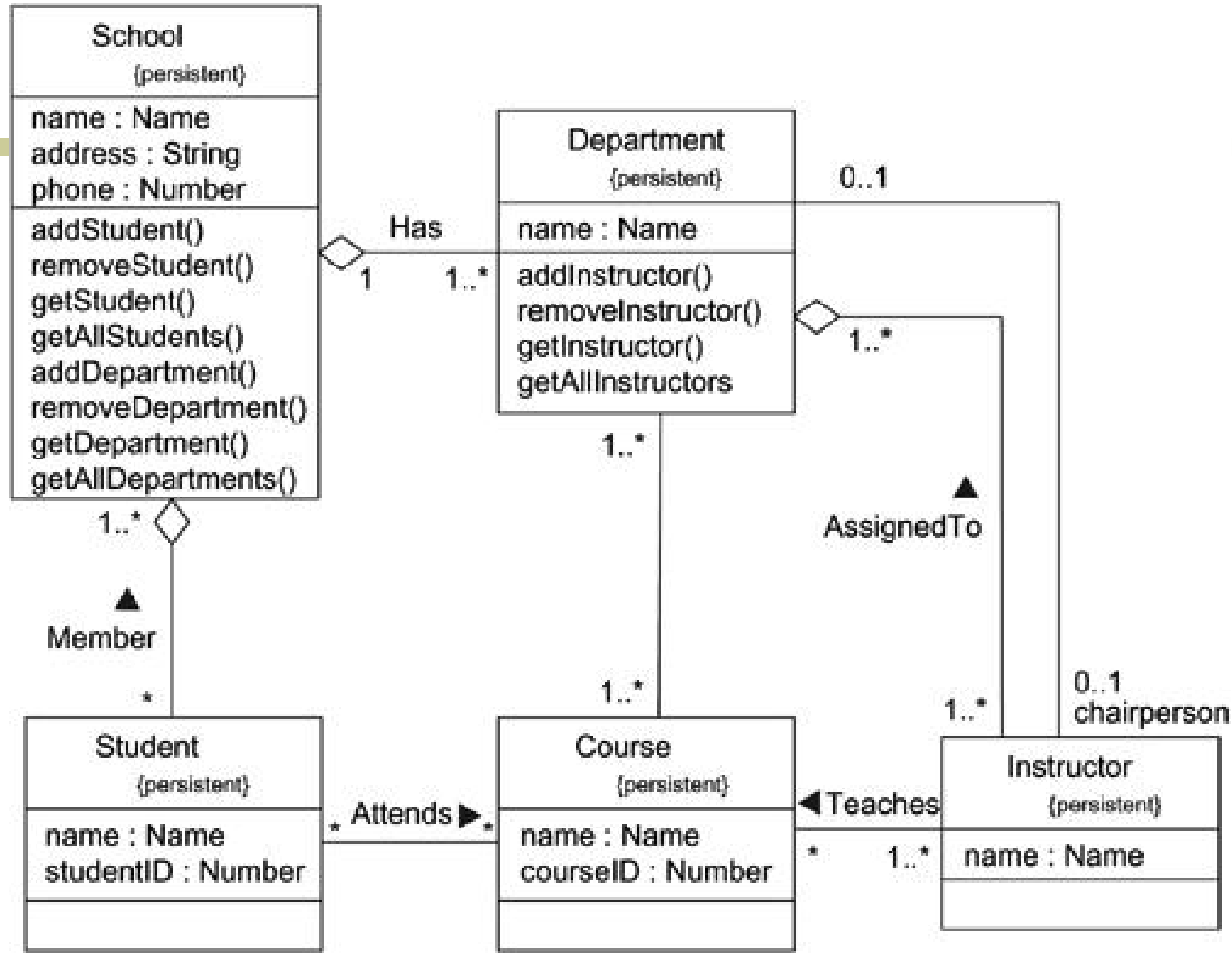

Company
class
aggregation
1
multiplicity
1..*
1..*
name
Department
name : Name
Location
Office
address : String
voice : Number
0 .. 1
constraint
role
{subset}
association
generalization
member
1..*
1
manager
Headquarters
Person
name : Name
employeeID : Integer
title : String
attributes
operations
getPhoto(p: Photo)
getSoundBite()
getContactInformation()
getPersonalRecords()
ContactInformation
address : String
dependency
PersonnelRecord
taxID
employmentHistory
salary
interface
ISecureInformation

1.2 Contents

- Class diagrams commonly contain the following things:
  - Classes
  - Interfaces
  - Collaborations
  - Dependency, generalization, and association relationships

1.3 Common Uses

- When you model the static design view of a system, you'll typically use class diagrams in one of three ways.
  - To model the vocabulary of a system
  - To model simple collaborations
  - To model a logical database schema


School
{persistent}
name : Name
address : String
phone : Number
addStudent()
removeStudent()
getStudent()
getAllStudents()
addDepartment()
removeDepartment()
getDepartment()
getAllDepartments()
Has
1
1..*
Department
{persistent}
name : Name
addInstructor()
removeInstructor()
getInstructor()
getAllInstructors
0..1
1..*
1..*
AssignedTo
1..*
Member
*
Student
{persistent}
name : Name
studentID : Number
* Attends *
1..*
Course
{persistent}
name : Name
courseID : Number
Teaches
*
1..*
1..*
0..1
chairperson
Instructor
{persistent}
name : Name

1.4 Forward and Reverse Engineering

- *Forward engineering* is the process of transforming a model into code through a mapping to an implementation language.
- *Reverse engineering* is the process of transforming code into a model through a mapping from a specific implementation language.

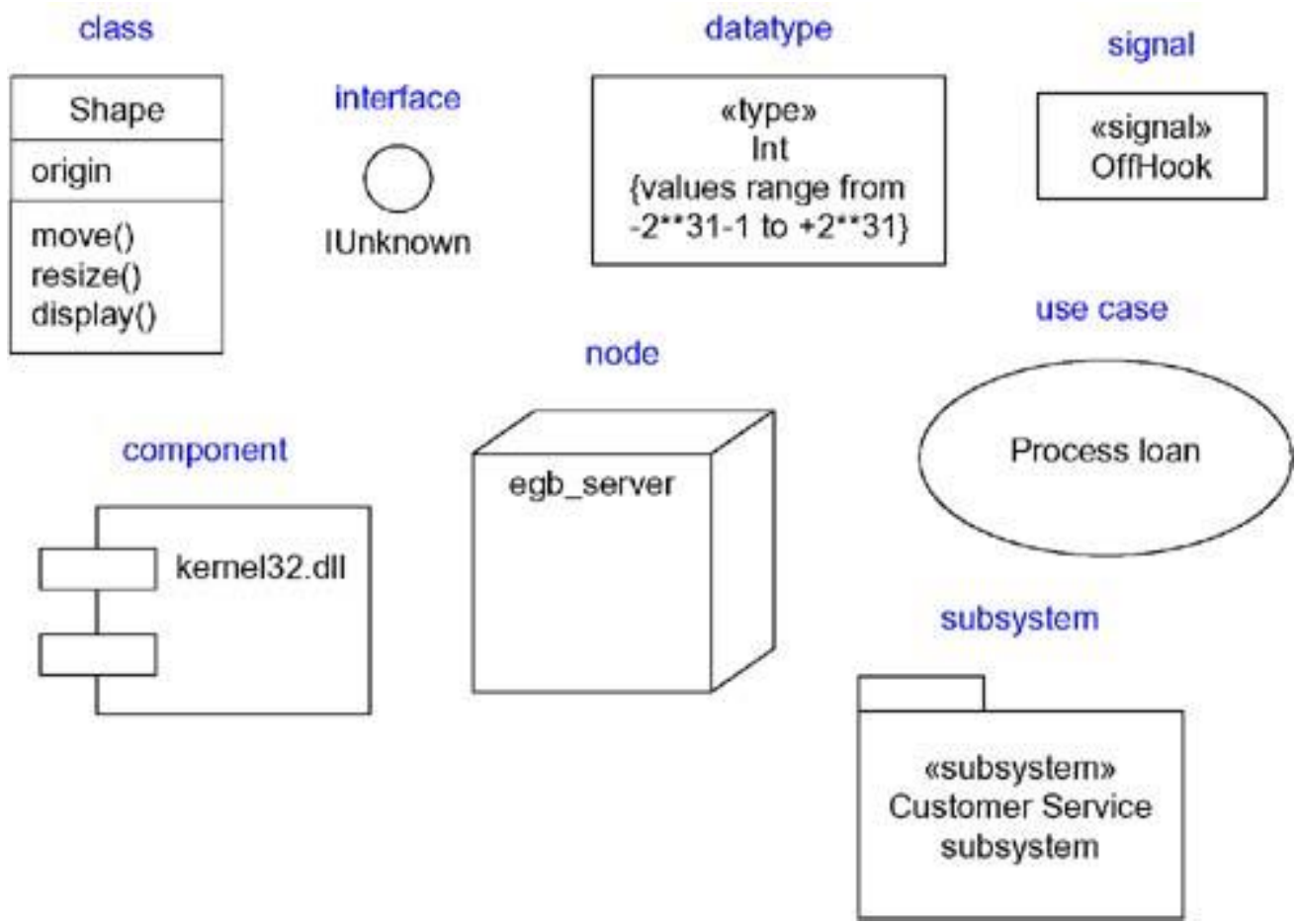
2. Advanced Classes

- Classes are just one kind of an even more general building block in the UML—classifiers.
  - A classifier is a mechanism that describes structural and behavioral features.
- Classifiers (and especially classes) have a number of advanced features

2.1 Classifier

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

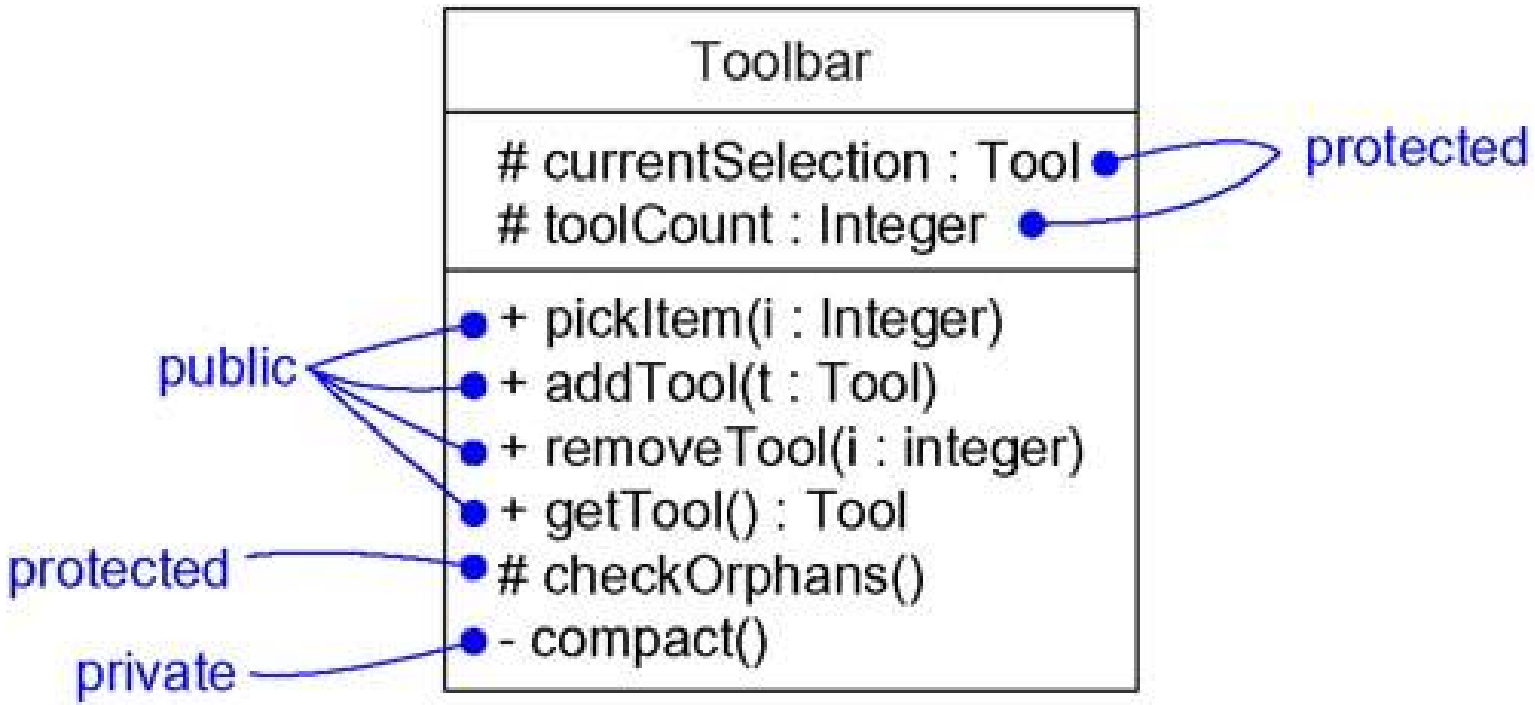
2.1 Classifier (cont')



2.2 Visibility

- public
  - Any outside classifier with visibility to the given classifier can use the feature; specified by prepending the symbol +
- protected
  - Any descendant of the classifier can use the feature; specified by prepending the symbol #
- Private
  - Only the classifier itself can use the feature; specified by prepending the symbol -

2.2 Visibility (cont')

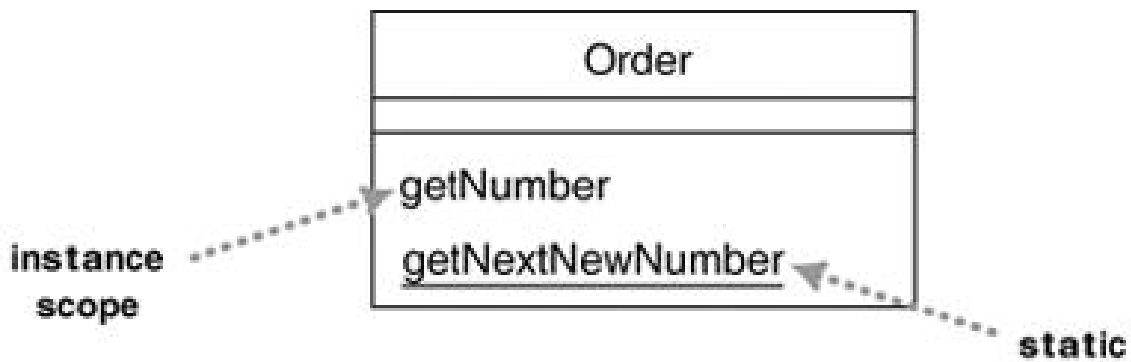


2.3 Scope

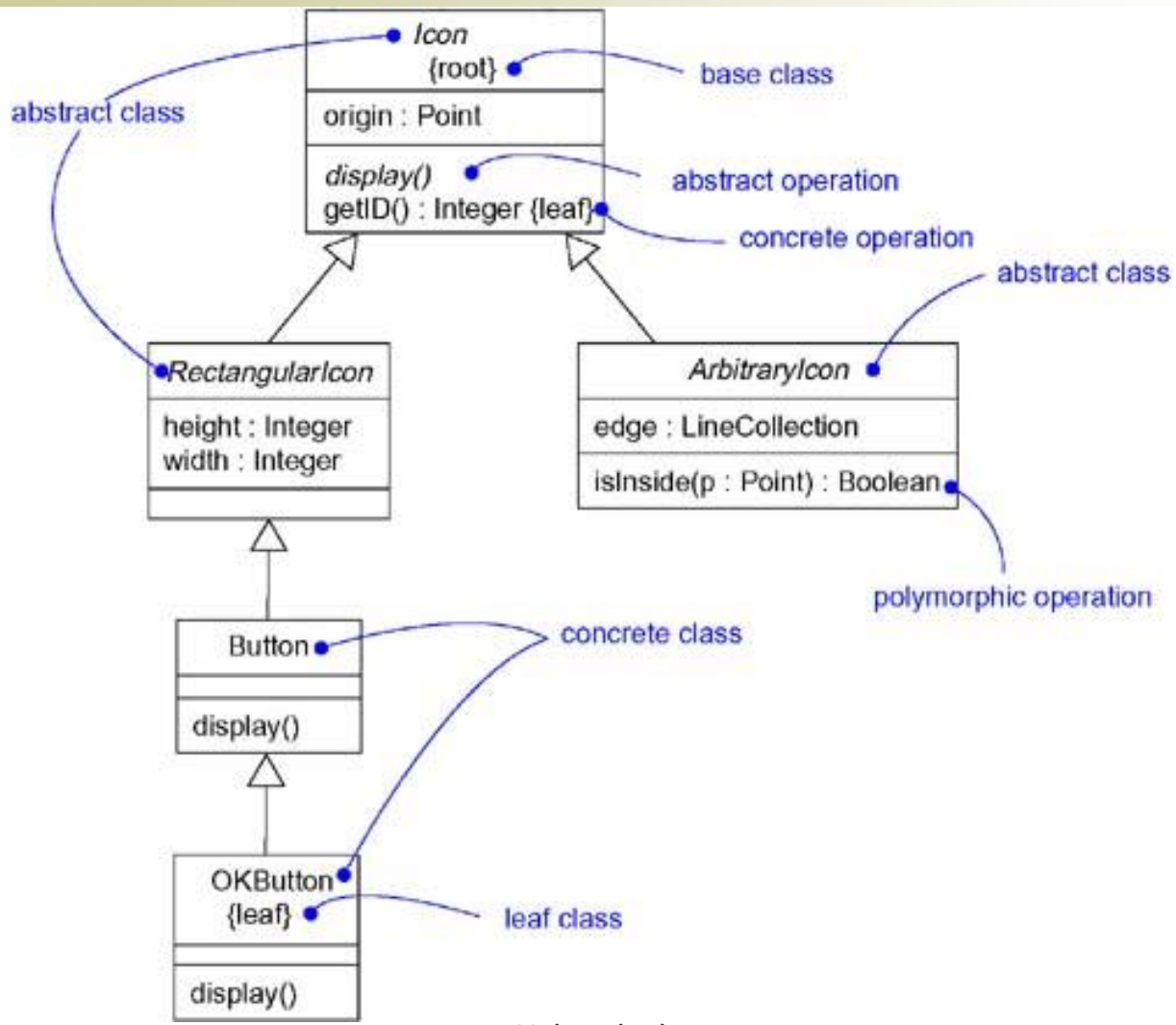
- instance
  - Each instance of the classifier holds its own value for the feature.
- classifier (static)
  - There is just one value of the feature for all instances of the classifier.

2.3 Scope (cont')

- Static Operations and Attributes
  - Static features are underlined on a class diagram



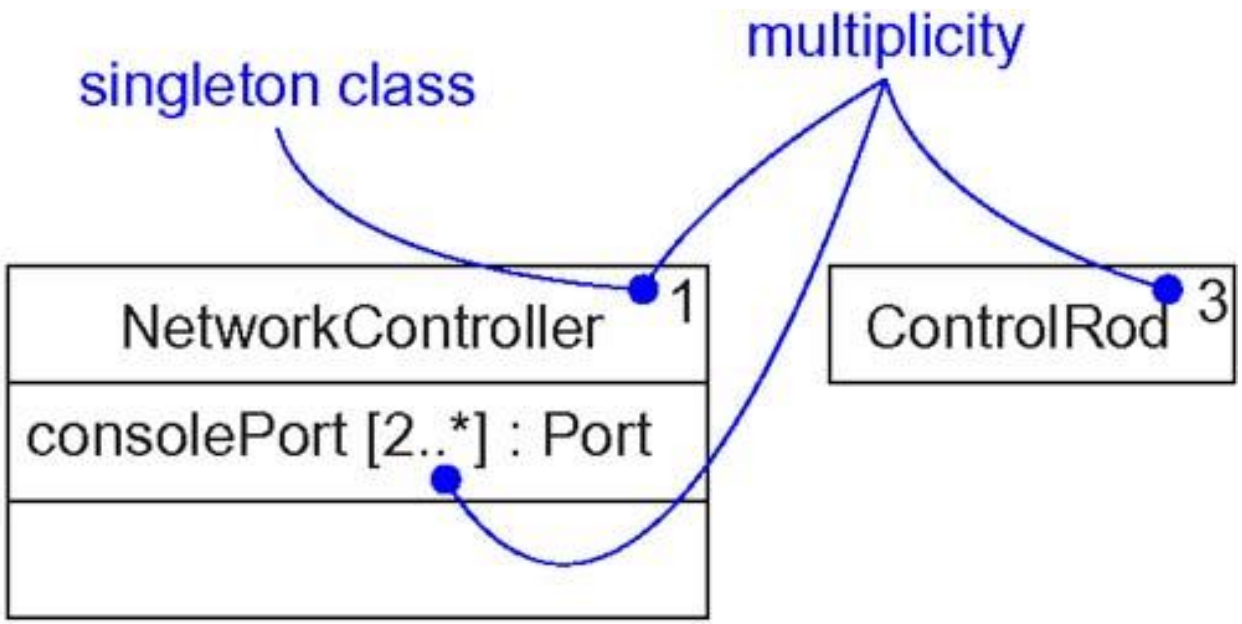
2.4 Abstract Elements



2.5 Multiplicity

- The number of instances a class may have is called its *multiplicity*.
- In the UML, you can specify the multiplicity of a class by writing a multiplicity expression in the upper-right corner of the class icon.
- Multiplicity applies to attributes. You can specify the multiplicity of an attribute by writing a suitable expression in brackets just after the attribute name.

2.5 Multiplicity (cont')



2.6 Attributes

- In its full form, the syntax of an attribute in the UML is
- [visibility] name [multiplicity] [: type] [= initial-value] [{property-string}]

2.6 Attributes (con't)

- There are three defined properties that you can use with attributes.
  - changeable (default)
    - There are no restrictions on modifying the attribute's value.
  - readOnly
    - This keyword to mark a property that can only be read by clients and that cannot be updated.
  - frozen (it was dropped from UML 2)
    - The attribute's value may not be changed after the object is initialized.

2.7 Operations

- In its full form, the syntax of an operation in the UML is
- [visibility] name [(parameter-list)] [: return-type] [{property-string}]

2.7 Operations (cont')

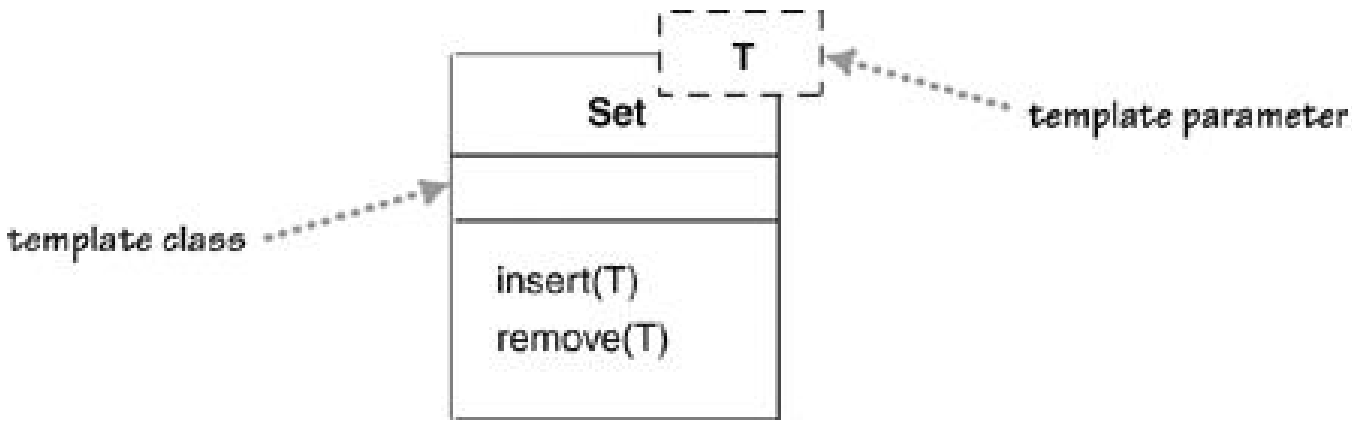
- each of parameters follows the syntax
- [direction] name : type [= default-value]
- Direction may be any of the following values:
  - in : An input parameter; may not be modified
  - Out : An output parameter; may be modified to communicate information to the caller
  - Inout : An input parameter; may be modified

2.7 Operations (cont')

- In addition to the leaf property described earlier, there are four defined properties that you can use with operations.
- isQuery
  - 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.
- sequential
- guarded
- concurrent

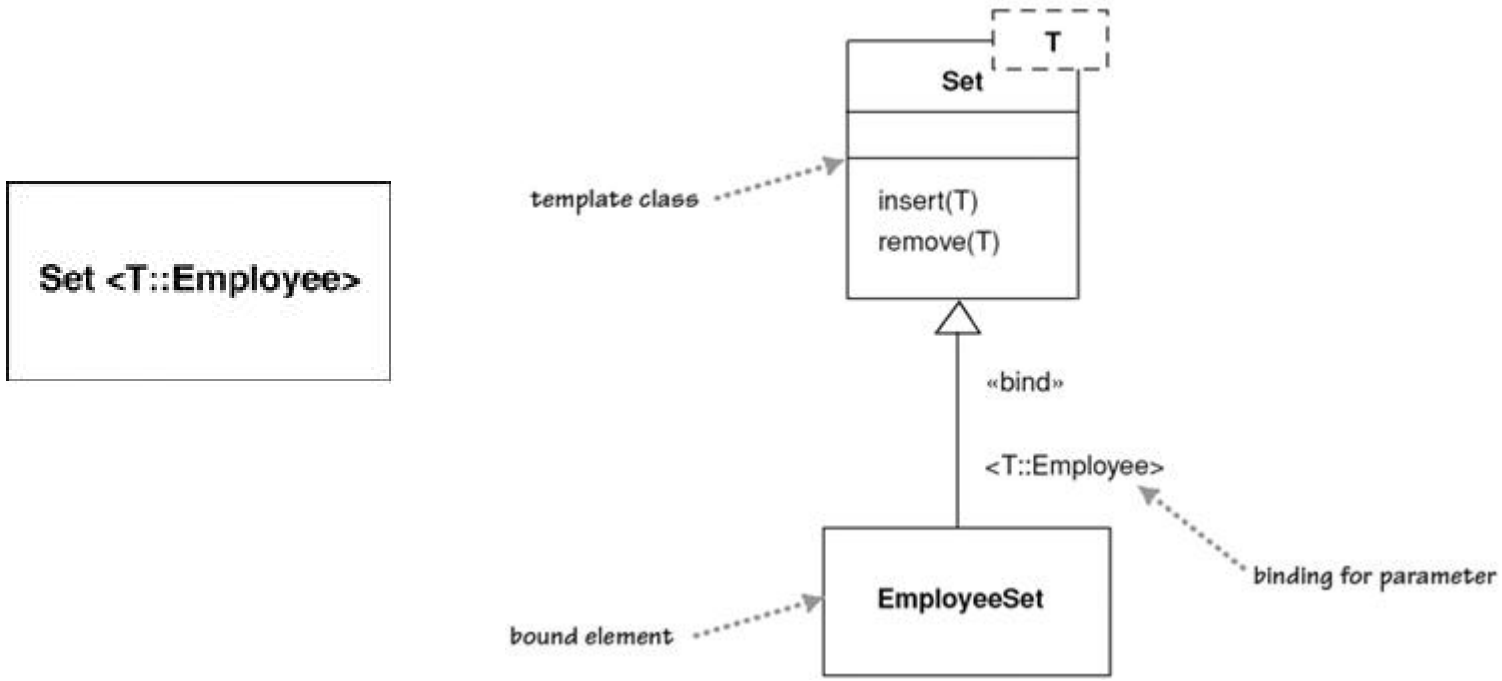
2.8 Template Classes

- Several languages, most noticeably C++, have the notion of a parameterized class, or template.



2.8 Template Classes (cont')

- A use of a parameterized class, such as Set<Employee>, is called a derivation. You can show a derivation in two wavs.



3. Advanced Relationships

- Dependencies, generalizations, and associations are the three most important relational building blocks of the UML.
- These relationships have a number of properties beyond those described in the previous section.
- You can also model multiple inheritance, navigation, composition, refinement, and other characteristics.
- Using a fourth kind of relationship—realization—you can model the connection between an interface and a class or component, or between a use case and a collaboration

3.1 dependency

- A *dependency* is a using relationship, specifying that a change in the specification of one thing may affect another thing that uses it, but not necessarily the reverse.
- Graphically, a dependency is rendered as a dashed line, directed to the thing that is depended on.

3.1 dependency (cont’)

- among classes and objects
  - <<bind>>
  - <<derive>>
  - <<friend>>
  - <<instanceOf>>
  - <<instantiate>>
  - <<powertype>>
  - <<refine>>
  - <<use>>

3.1 dependency (cont')

- among packages
  - <<access>>
  - <<import>>
- among use cases
  - <<extend>>
  - <<include>>

3.1 dependency (cont’)

- among objects
  - <<become>>
  - <<call>>
  - <<copy>>
- One stereotype in state machines
  - <<send>>
- organizing the elements of your system into subsystems and models
  - <<trace>>

3.2 generalization

- A class that has exactly one parent is said to use *single inheritance*.
- a class with more than one parent is said to use *multiple inheritance*.
- UML defines one stereotype and four constraints that may be applied to generalization relationships.

3.2 generalization (cont’)

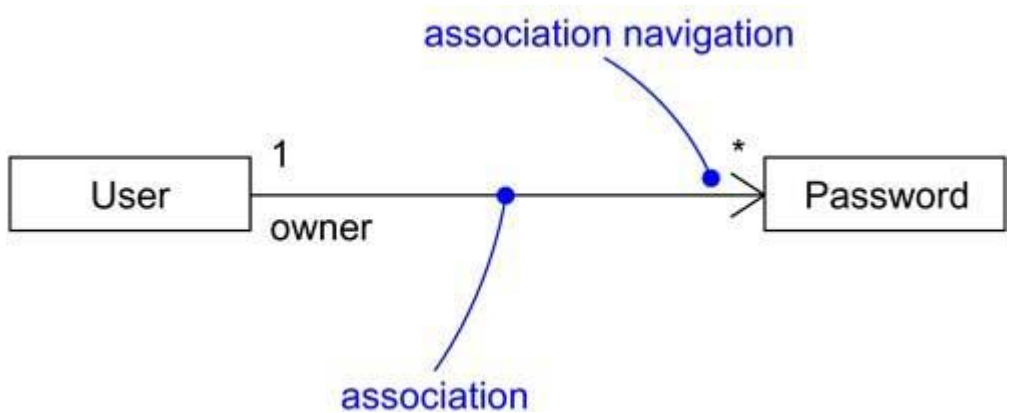
- four standard constraints
  - {complete}
  - {incomplete}
  - {disjoint}
  - {overlapping}
- One stereotype
  - <<implementation>>

3.3 Association

- An *association* is a structural relationship, specifying that objects of one thing are connected to objects of another.
- There are four basic adornments that apply to an association: a name, the role at each end of the association, the multiplicity at each end of the association, and aggregation.
- For advanced uses, there are a number of other properties you can use to model subtle details, such as navigation, qualification, and aggregation.

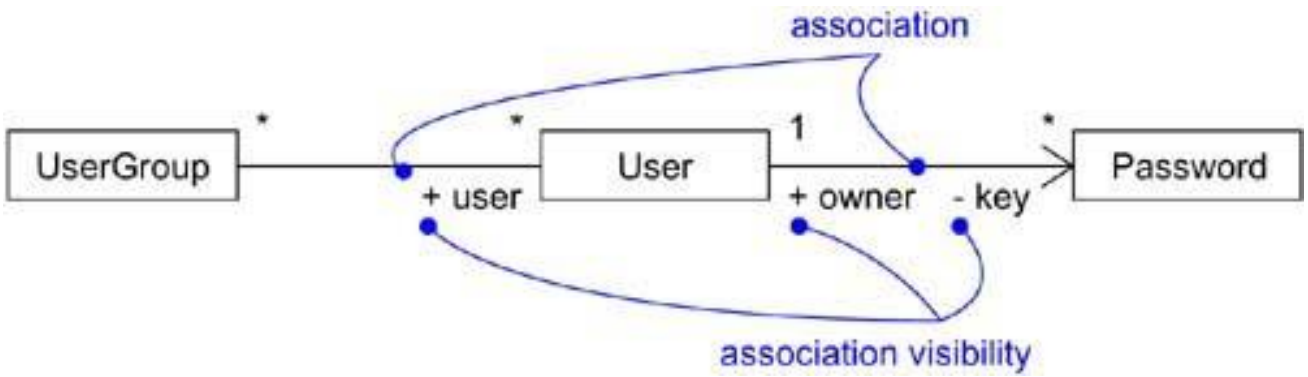
3.3.1 Navigation

- Unless otherwise specified, navigation across an association is bidirectional.
- However, there are some circumstances in which you'll want to limit navigation to just one direction.



3.3.2 Visibility

- There are circumstances in which you'll want to limit the visibility across that association relative to objects outside the association.

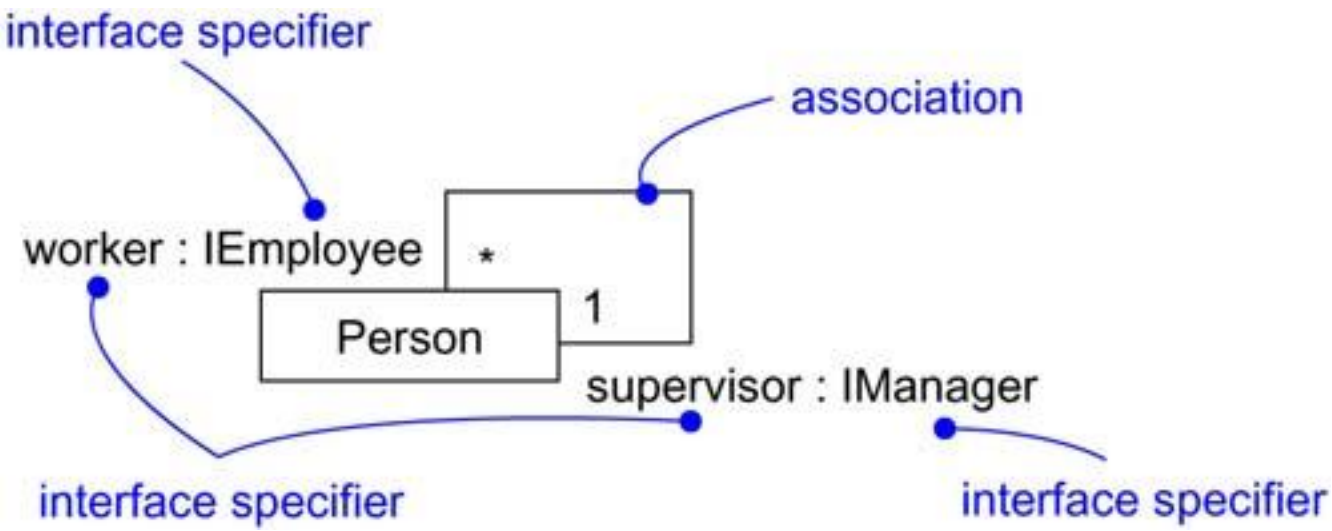


3.3.3 Qualification

- A *qualified association* is the UML equivalent of a programming concept variously known as associative arrays, maps, hashes, and dictionaries.

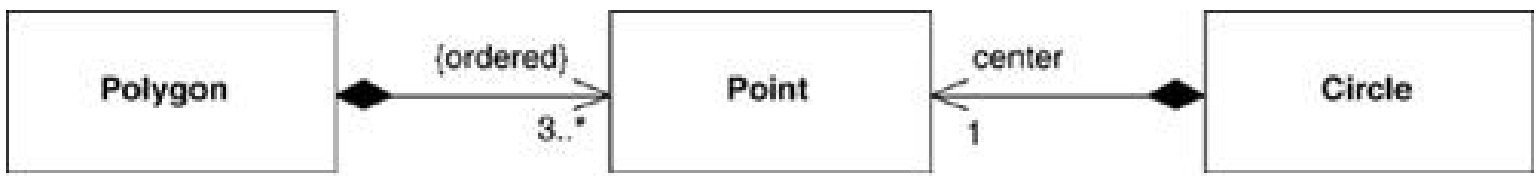
3.3.4 Interface Specifier

- In the context of an association with another target class, a source class may choose to present only part of its face to the world.



3.3.5 Composition

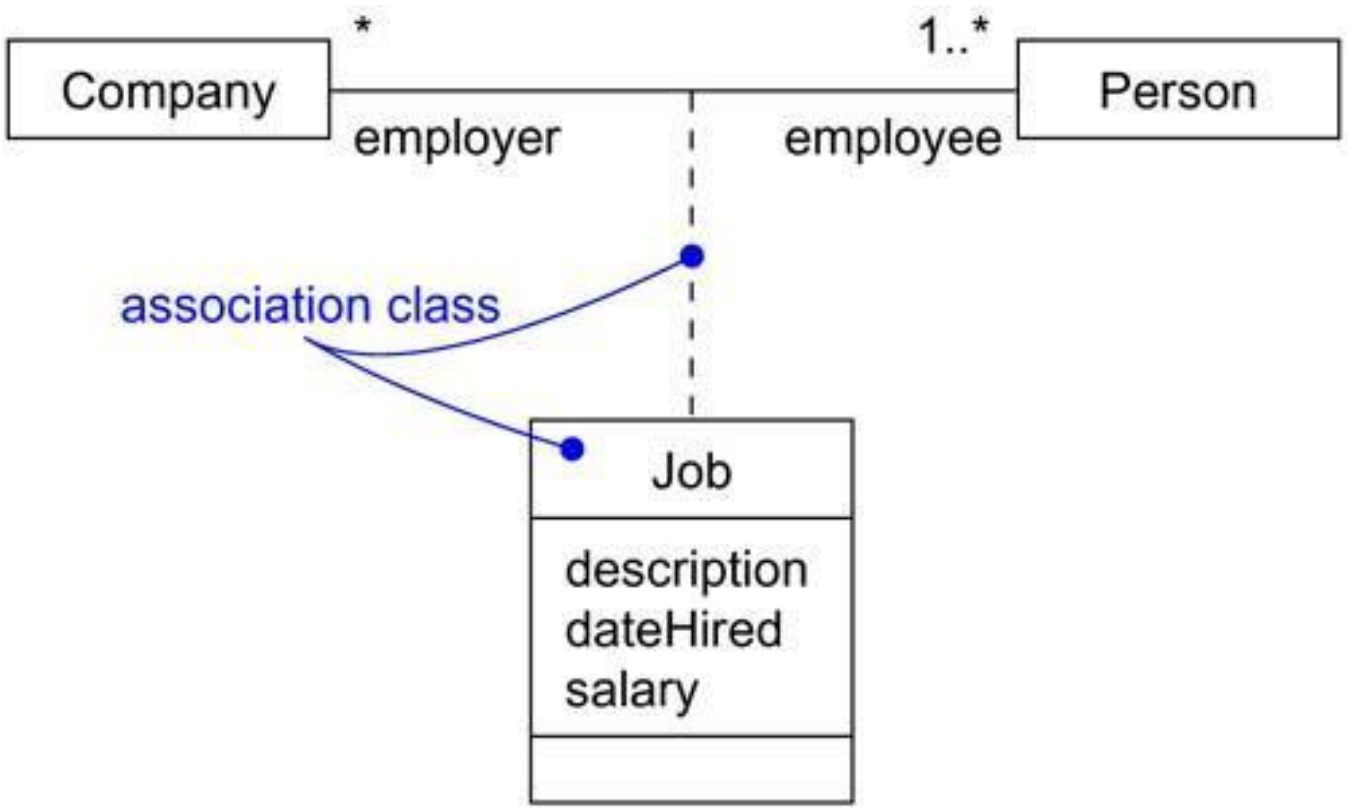
- *Composition* is a form of aggregation, with strong ownership and coincident lifetime as part of the whole.



3.3.6 Association Classes

- In an association between two classes, the association itself might have properties.
- In the UML, you'd model this as an association class, which is a modeling element that has both association and class properties.
- You render an association class as a class symbol attached by a dashed line to an association

3.3.6 Association Classes (cont')



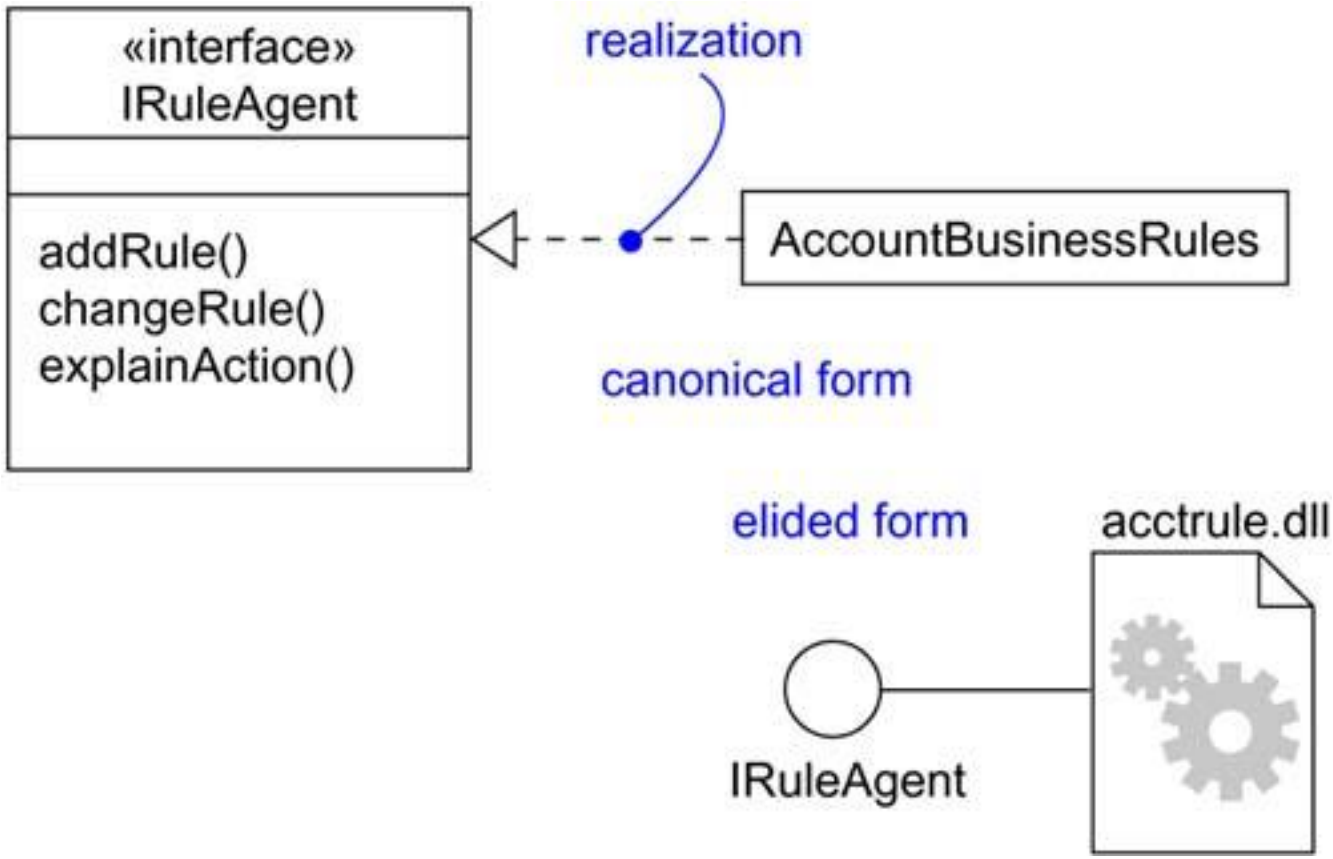
3.3.7 Constraints

- {implicit}
- {ordered}
- {xor}
- {subset}
- {changeable}
- {addOnly}
- {frozen}

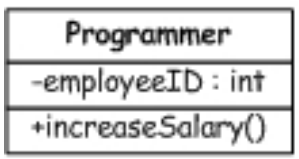
4. Realization

- 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.

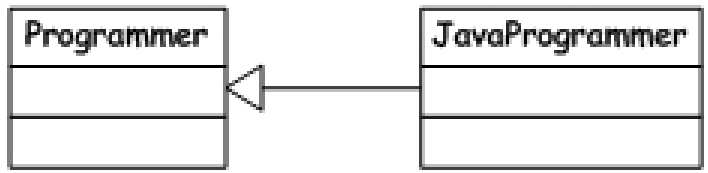
4. Realization (cont')



5. UML mapping Java



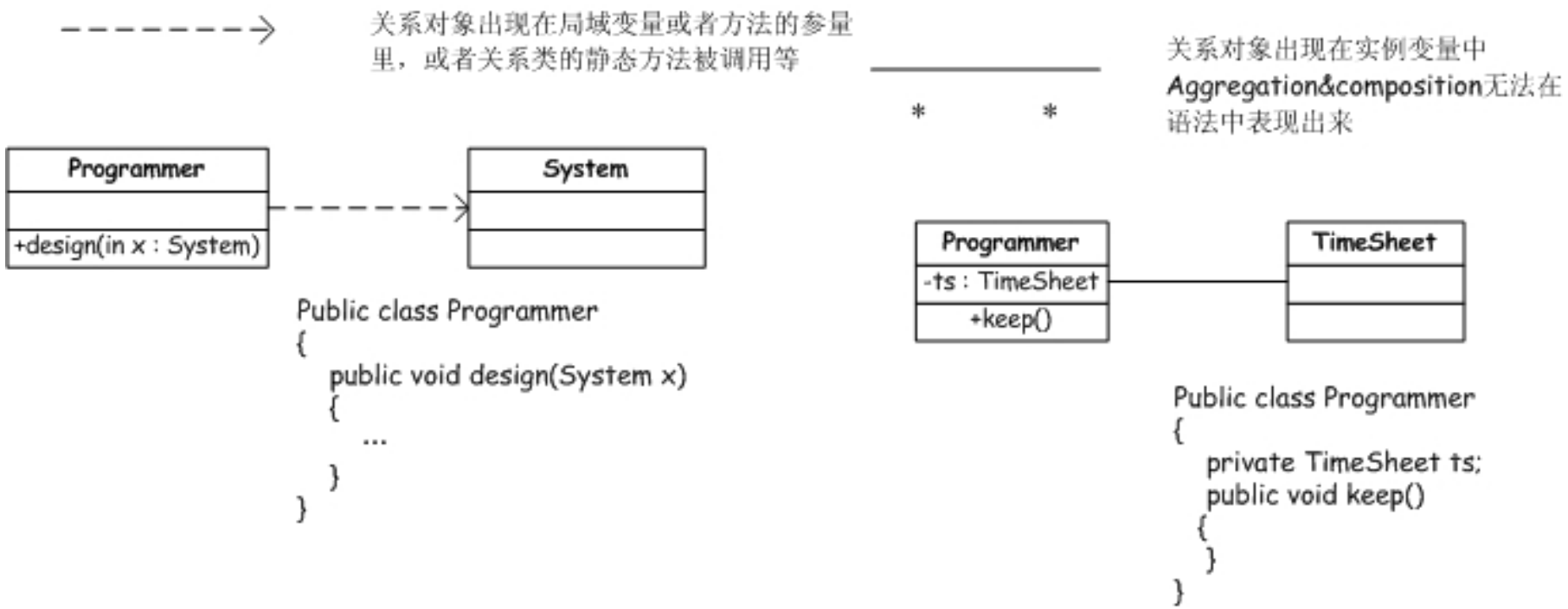

```
public class Programmer
{
   private int employeeId;
   public void increaseSalary()
   {
      ...
   }
}
```




```
public interface Myinterface
{
   public void MyMethod()
}
```

```
public class Programmer
{
    ...
}
public class JavaProgrammer
      extends Programmer
```

5. UML mapping Java (cont')



6. Other reference

- <UML Distilled > Martin Fowler
  - http://martinfowler.com
- <UML2 Toolkit> John.Wiley