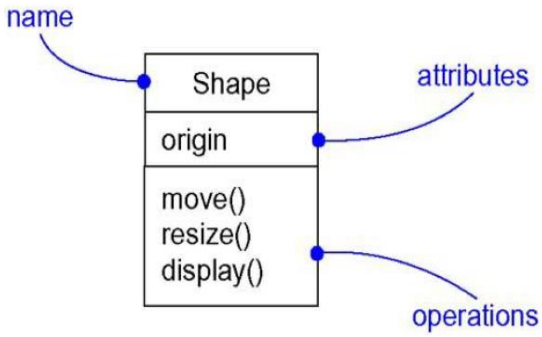
CLASS DIAGRAM

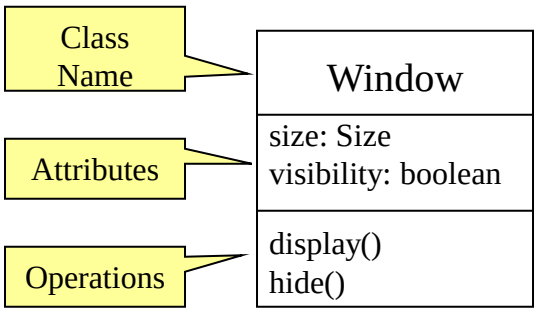
Introduction

- Most important building block
- Description of a set of objects that share the same attributes, operations, relationships and semantics
- Used to capture the vocabulary of the system
  - Include abstractions of the problem domain
  - Classes that make up the implementation
- Form a part of a balanced distribution of responsibilities across the system



Class

- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behavior)
  - Relationships with other classes
- Attributes and operations may
  - have their visibility marked:
  - "+" for *public*
  - "#" for *protected*
  - "–" for *private*
  - "~" for *package*

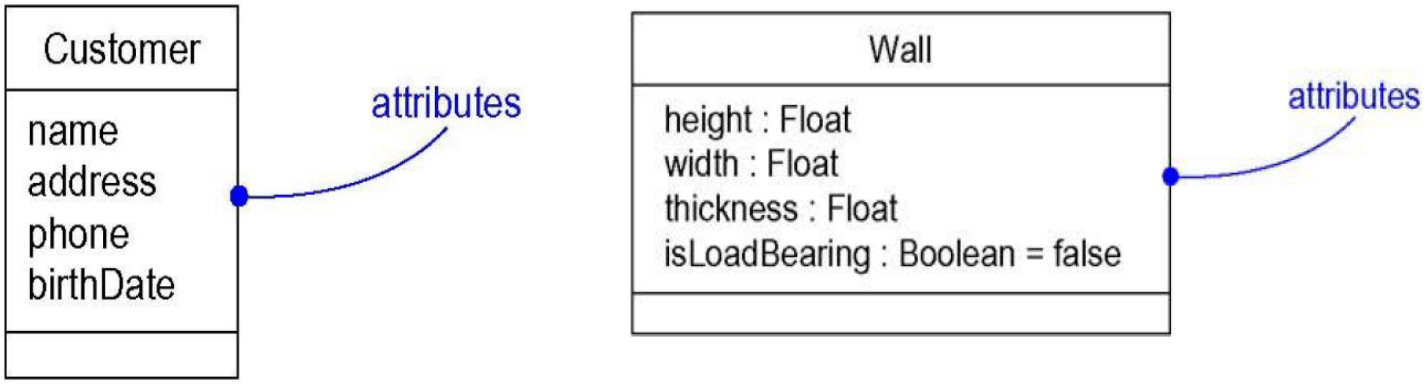


Class Names

- Name must be unique within its enclosing package
- Simple name and Path name
- Class names are noun phrases from the vocabulary of the system

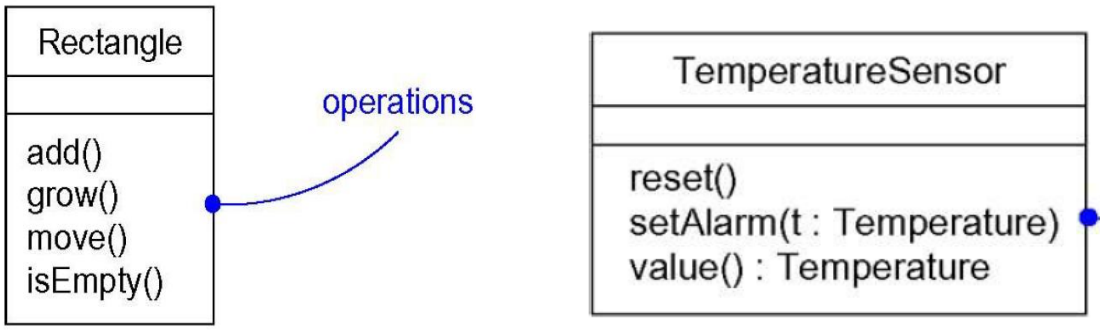
Attributes

- Named property of a class
  - Describes a range of values that instances of the property hold
  - Shared by all objects of that class
- An abstraction of the kind of data or state of an object
  - At a given moment, an object will have specific values for each of its attributes
- Noun phrase that represents some property



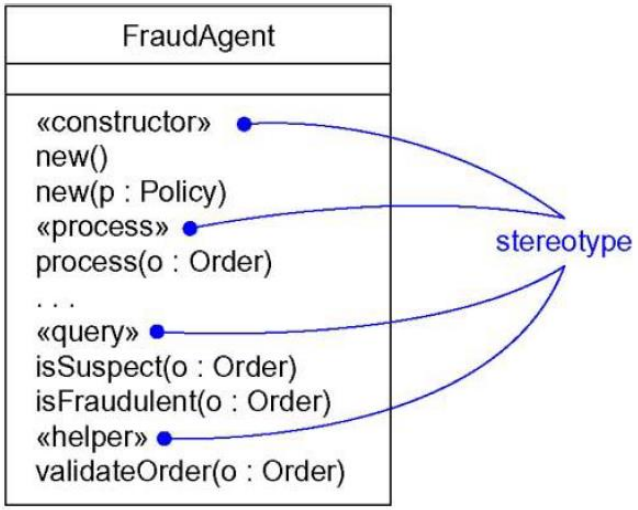
Operations

- A service that can be requested from any object of the class to affect behavior
- Invoking an operation changes the object's data or state
- Verb phrase representing some behavior of the class



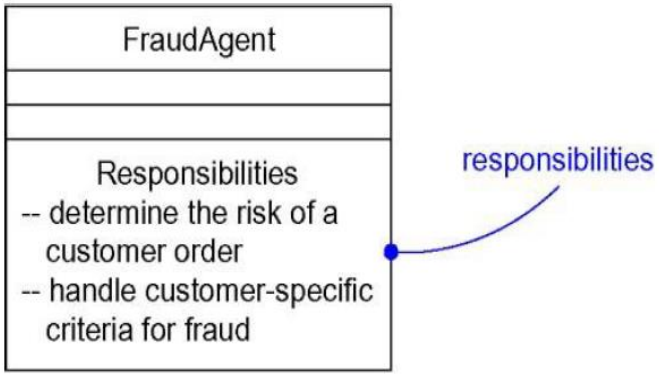
Organizing Attributes and Operations

- Don’t have to show every attribute and operation at once
- End each list with ellipsis (…)
- Can also prefix each group with a descriptive category using stereotypes



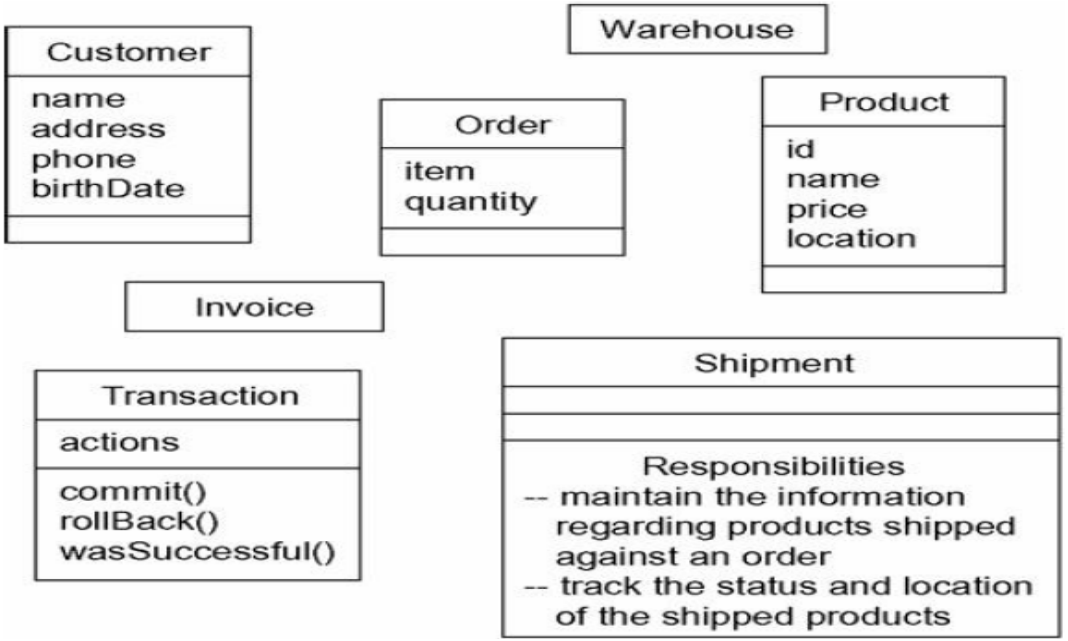
Responsibilities

- Is a contract or an obligation of a class
  - All objects of the class have the same kind of state and behavior
- Attributes and operations are the features that carry out the class's responsibility
  - `TemperatureSensor` class responsible for measuring temperature and raising an alarm
- Techniques used - CRC cards and use case based-analysis



Modeling the Vocabulary of a System

- Identify the things that describe the problem
- For each abstraction, identify a set of responsibilities
- Provide the attributes and operations needed to carry out those responsibilities



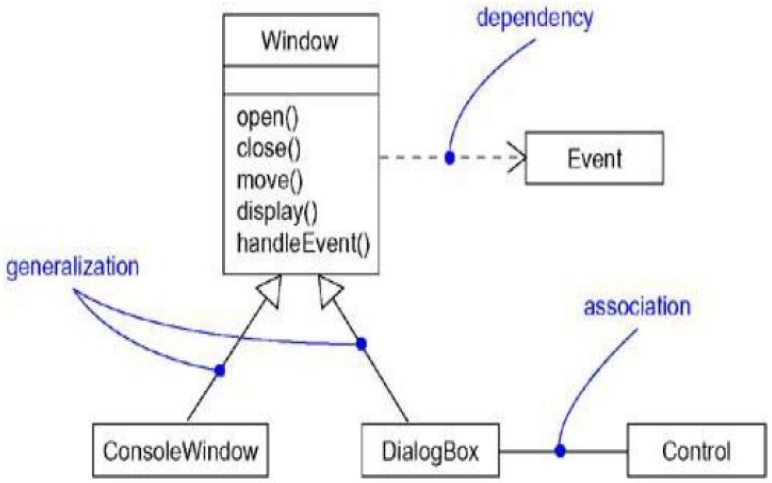
CLASS RELATIONSHIPS

Introduction

- Classes collaborate with other classes in a number of ways
- Three kinds of relationships in OO modeling
  - Dependencies
  - Generalizations
  - Associations
- Provide a way for combining your abstractions
- Building relationships is not like creating a balanced set of responsibilities

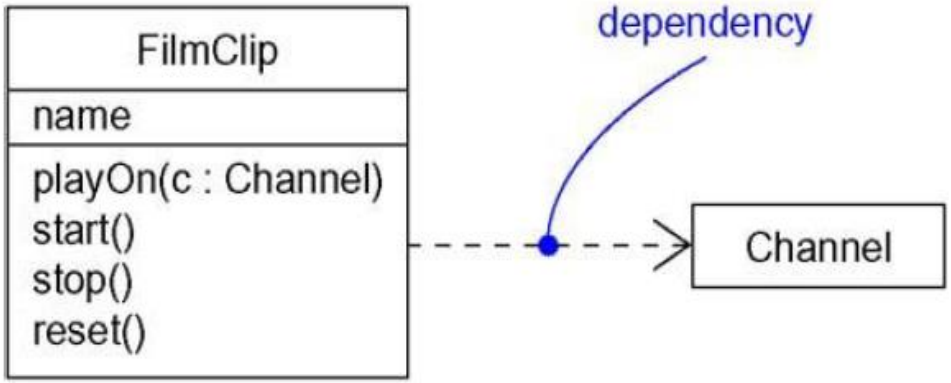
Introduction

- Relationship is a connection among things
- UML graphical notation
  - Permits to visualize relationships apart from any specific programming language
  - Emphasize the most important parts of a relationship
    - Name
    - Things it connects
    - Properties



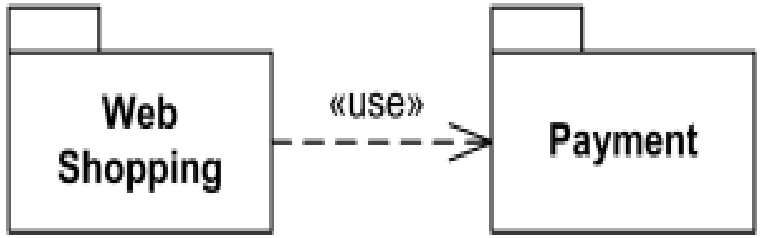
Dependency

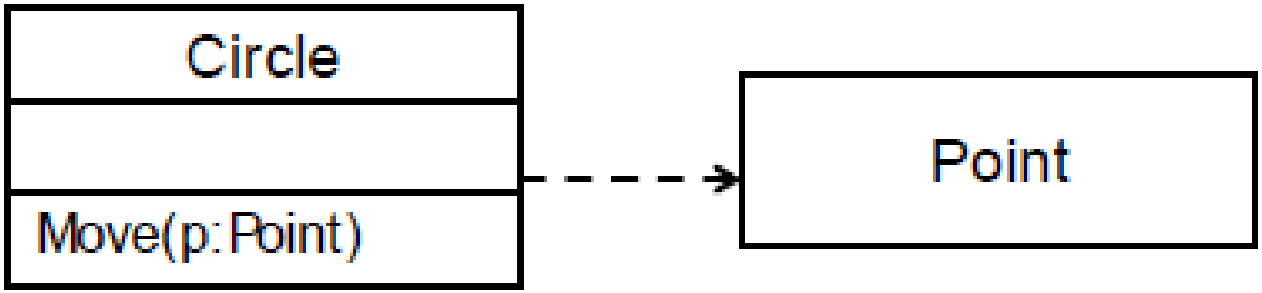
- *Using* relationship
  - States that a change in specification of one thing may affect another thing that uses it
- Used in the context of classes
  - Uses operations or variables/arguments typed by the other class
  - Shown as an argument in the signature of an operation
  - If used class changes, the operation of the other class may be affected



Dependency


CarFactory
«instantiate»
Car

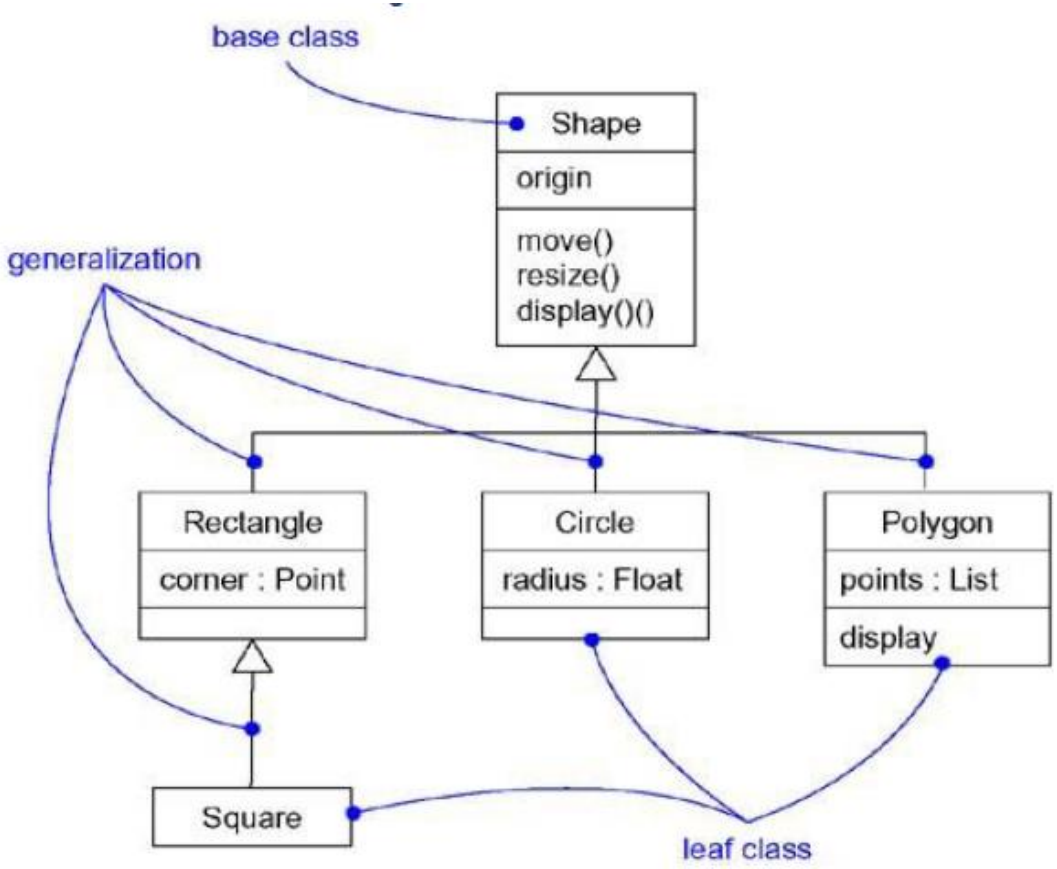

Web
Shopping
«use»
Payment


Circle
Move(p:Point)
Point

Generalization

- Relationship between a general thing (superclass) and a more specific kind of that thing (subclass)
- “Is-a-kind-of” relationship
- Child is substitutable for the parent
- Polymorphism
  - Operation of child has the same signature as an operation in the parent but overrides it

Generalization

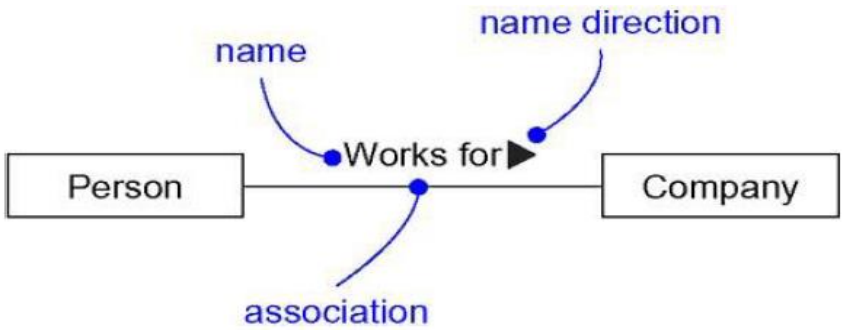


Association

- Structural relationship
  - Specifies that objects of one thing are connected to objects of another
- Can navigate from objects of one class to the other
- Self Association
  - Connects a class to itself
- Binary Association
  - Connects exactly two classes

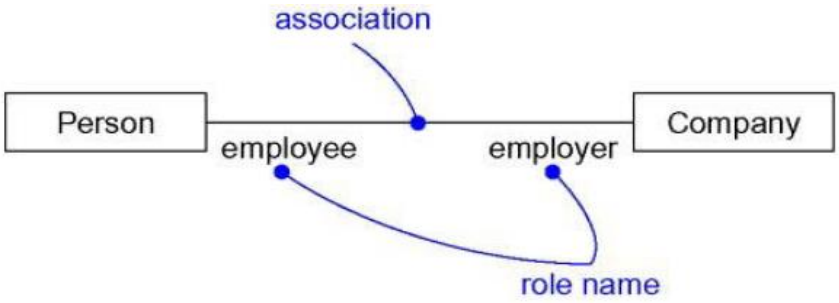
Association : Adornments

- Name
  - Use a name to describe the nature of the relationship
  - Can give a direction to the name



Association : Adornments

- Role
  - The face that the class at the far end of the association presents to the class at the near end
  - Explicitly name the role a class plays (*End name* or *Role name*)
  - Same class can play the same or different roles in other associations

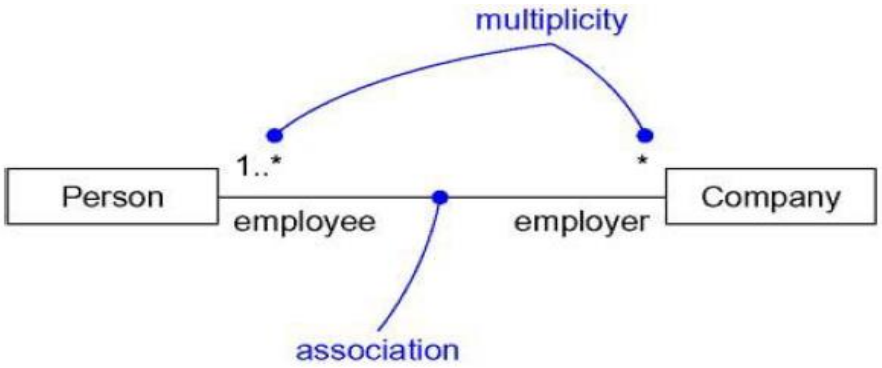


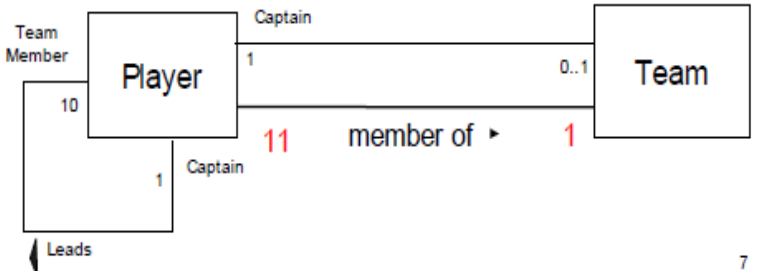
Association : Adornments

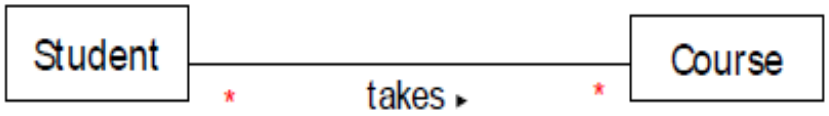
- Multiplicity
  - States how many objects may be connected across an instance of an association
  - An expression that evaluates to a range of values or an explicit value
  - Multiplicity at one end of an association means
    - For each object of the class at the opposite end, there must be that many objects at the near end
  - Show a multiplicity of
    - Exactly one (1)
    - Zero or one (0 . . 1)
    - Many (0 . . *)
    - One or more (1 . . *)
    - State an exact number (ex: 3)

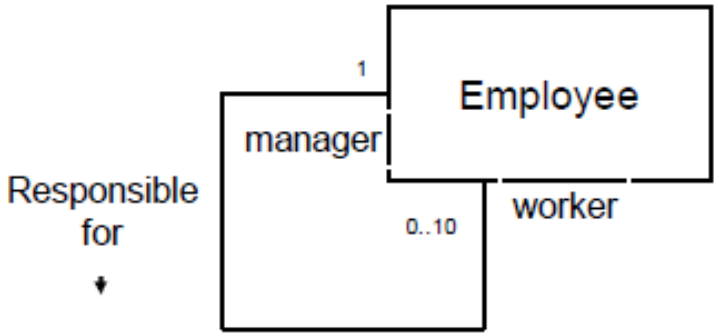
Association : Adornments

- Multiplicity
  - Specify complex multiplicities by using a list, 0..1, 3..4, 6..* (any number of objects other than 2 or 5)







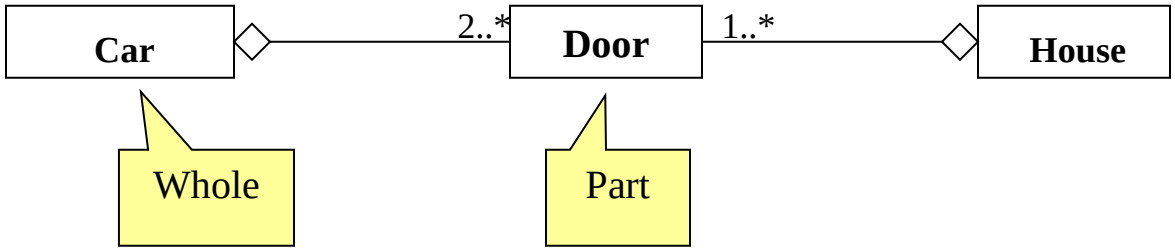


Association : Adornments

- Aggregation
  - Model a whole/part relationship
  - Represents a “has-a” relationship

Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a “is a part-part of” relationship.

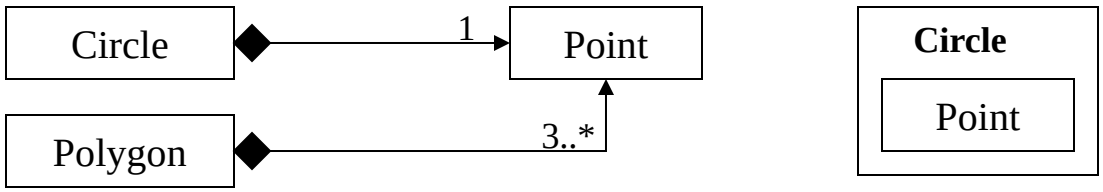


Aggregation (cont.)

- Aggregation tests:
  - Is the phrase “part of” used to describe the relationship?
    - A door is “part of” a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door **is** part of a car. A car **is not** part of a door.

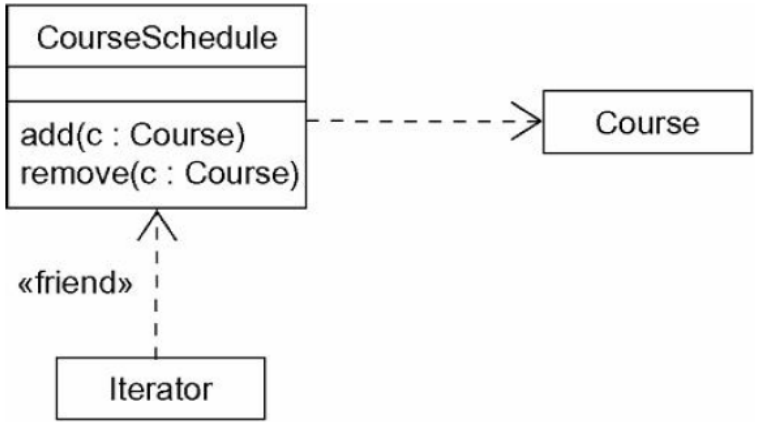
Composition

- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.



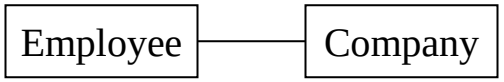
Modeling Simple Dependencies

- To model the using dependency
  - Create a dependency pointing from the class with the operation to the class used as a parameter in the operation

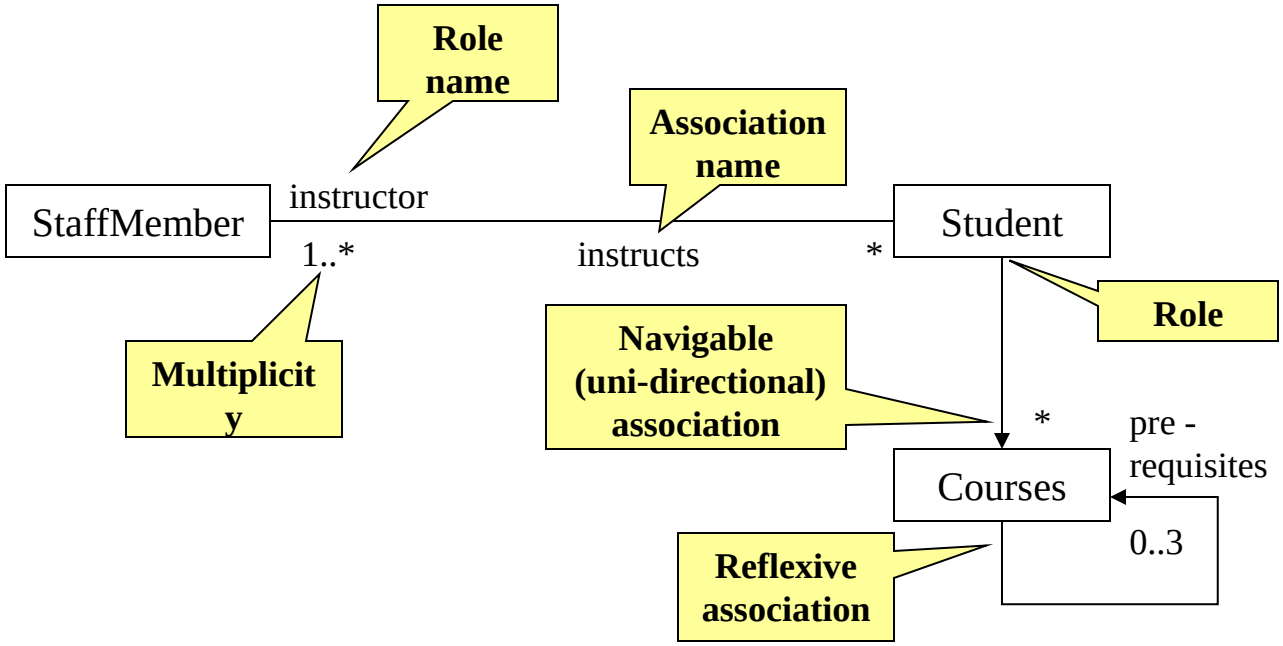


Associations

- An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.
- Example: "An Employee works for a Company"



Associations (cont.)



Associations (cont.)

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

Associations (cont.)

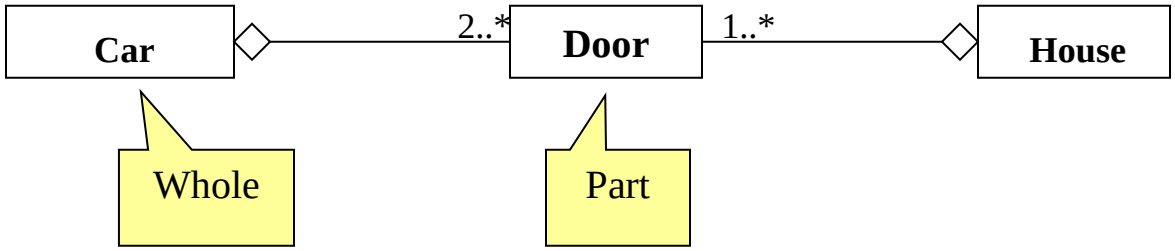
- Multiplicity
  - the number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

Multiplicity Indicators

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

Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a “is a part-part of” relationship.

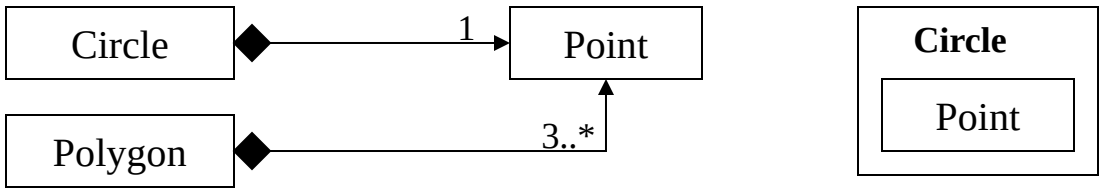


Aggregation (cont.)

- Aggregation tests:
  - Is the phrase “part of” used to describe the relationship?
    - A door is “part of” a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door **is** part of a car. A car **is not** part of a door.

Composition

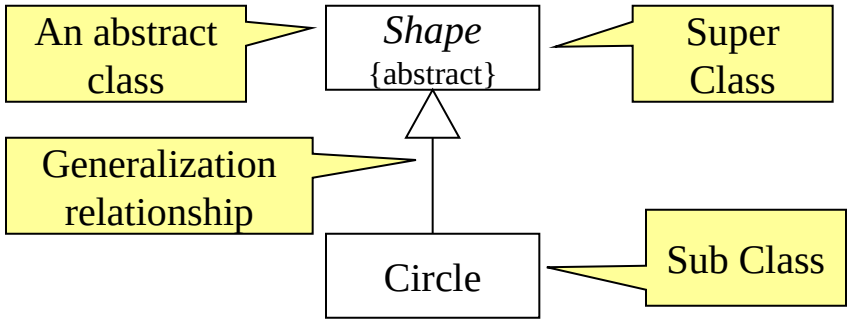
- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.



Generalization

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

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



Generalization

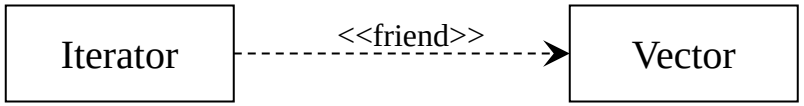
- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

Realization

- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.

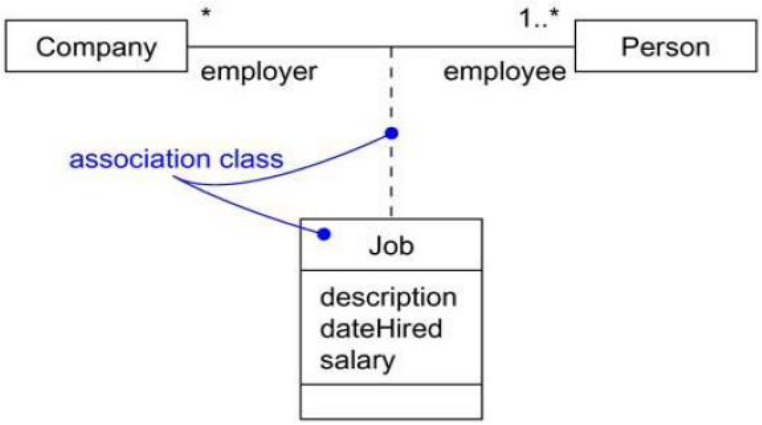
Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.
- This is different from an association, where an attribute of the dependent class is an instance of the independent class.

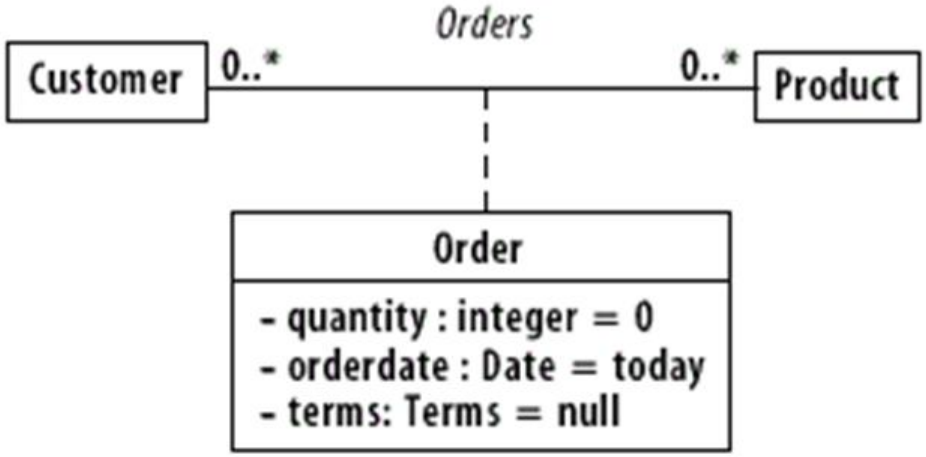


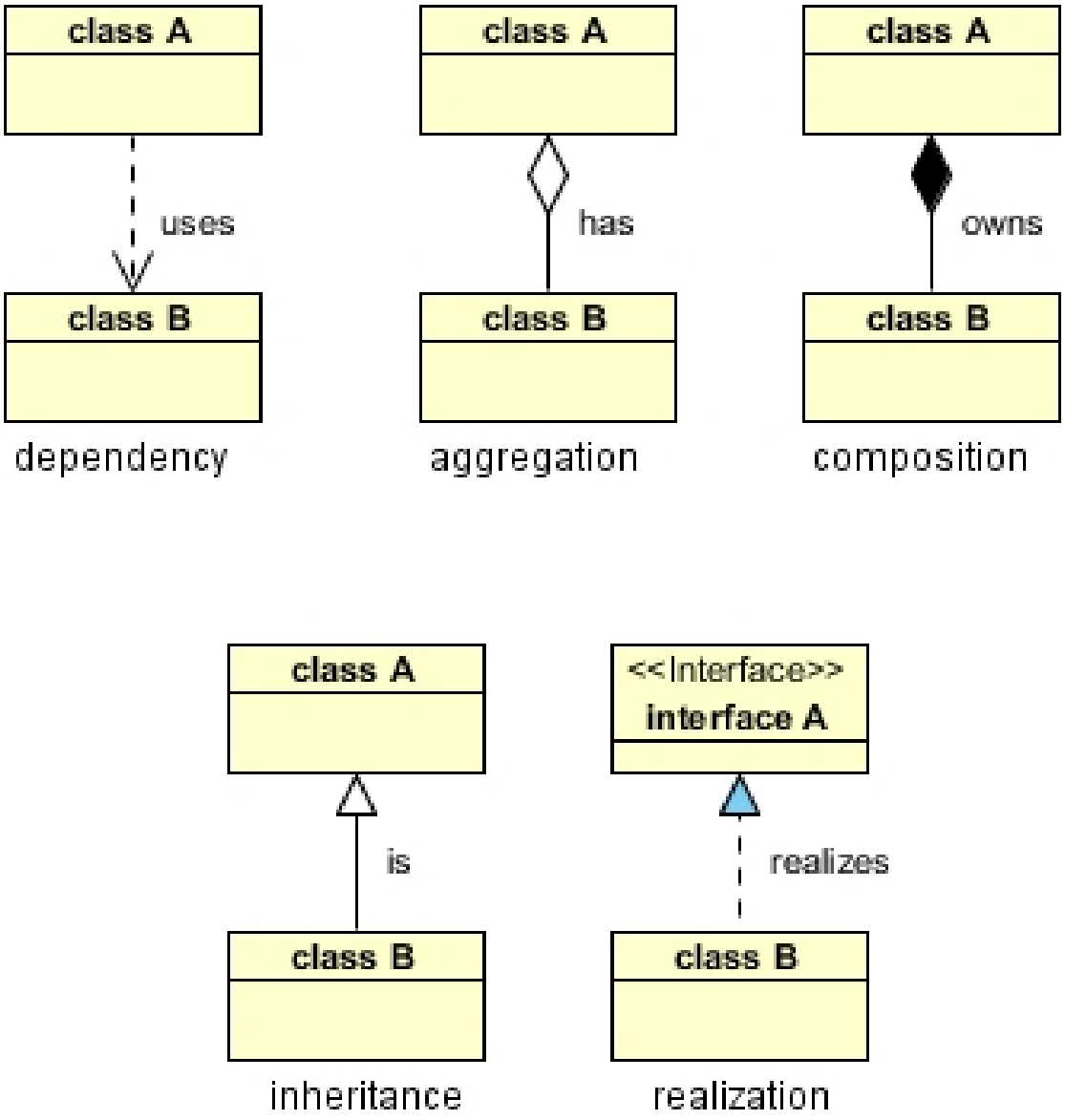
Association

- **Association Classes**
  - Association itself may have properties
  - Ex: In the relationship between a Company and a Person, there is Job that represent the properties of that relationship
    - Represents exactly one pairing of Person and Company
  - Modeling element that has both association and class properties
  - Can’t attach an association class to more than one association



Association Class

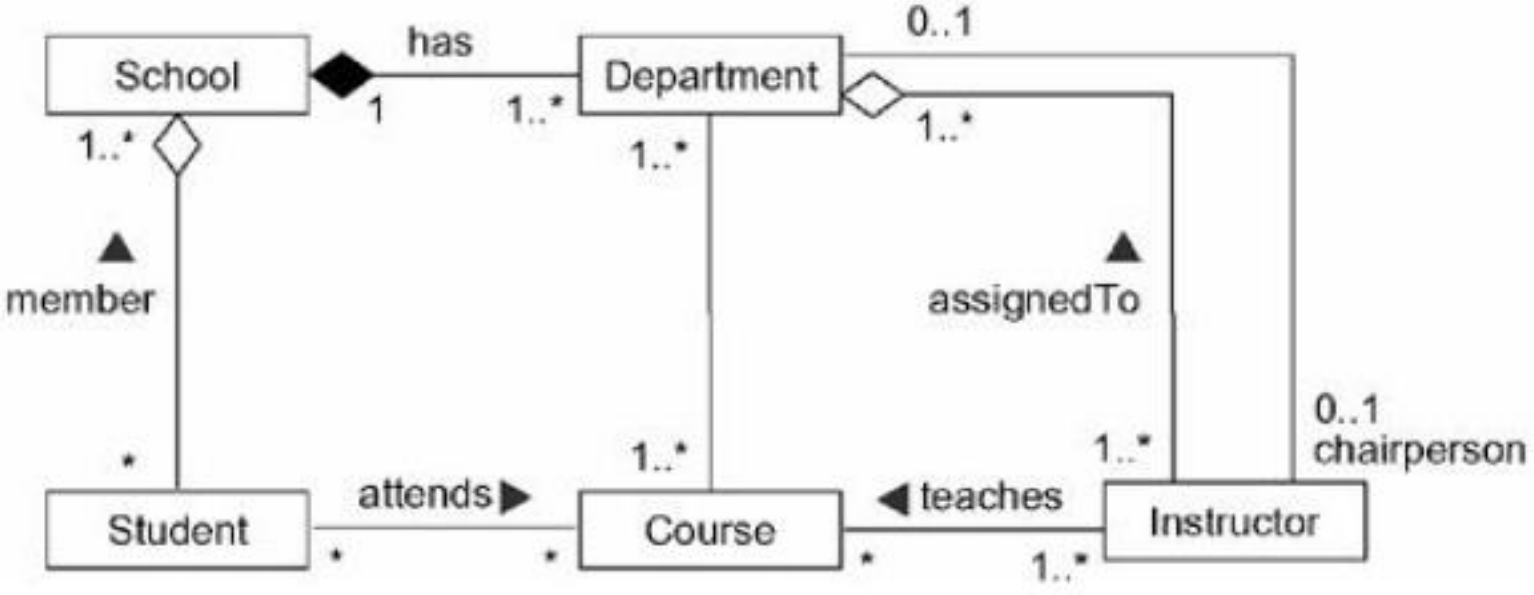



class A
uses
class B
dependency
class A
has
class B
aggregation
class A
owns
class B
composition
class A
is
class B
inheritance
<<Interface>>
interface A
realizes
class B
realization

Modeling Structural Relationships

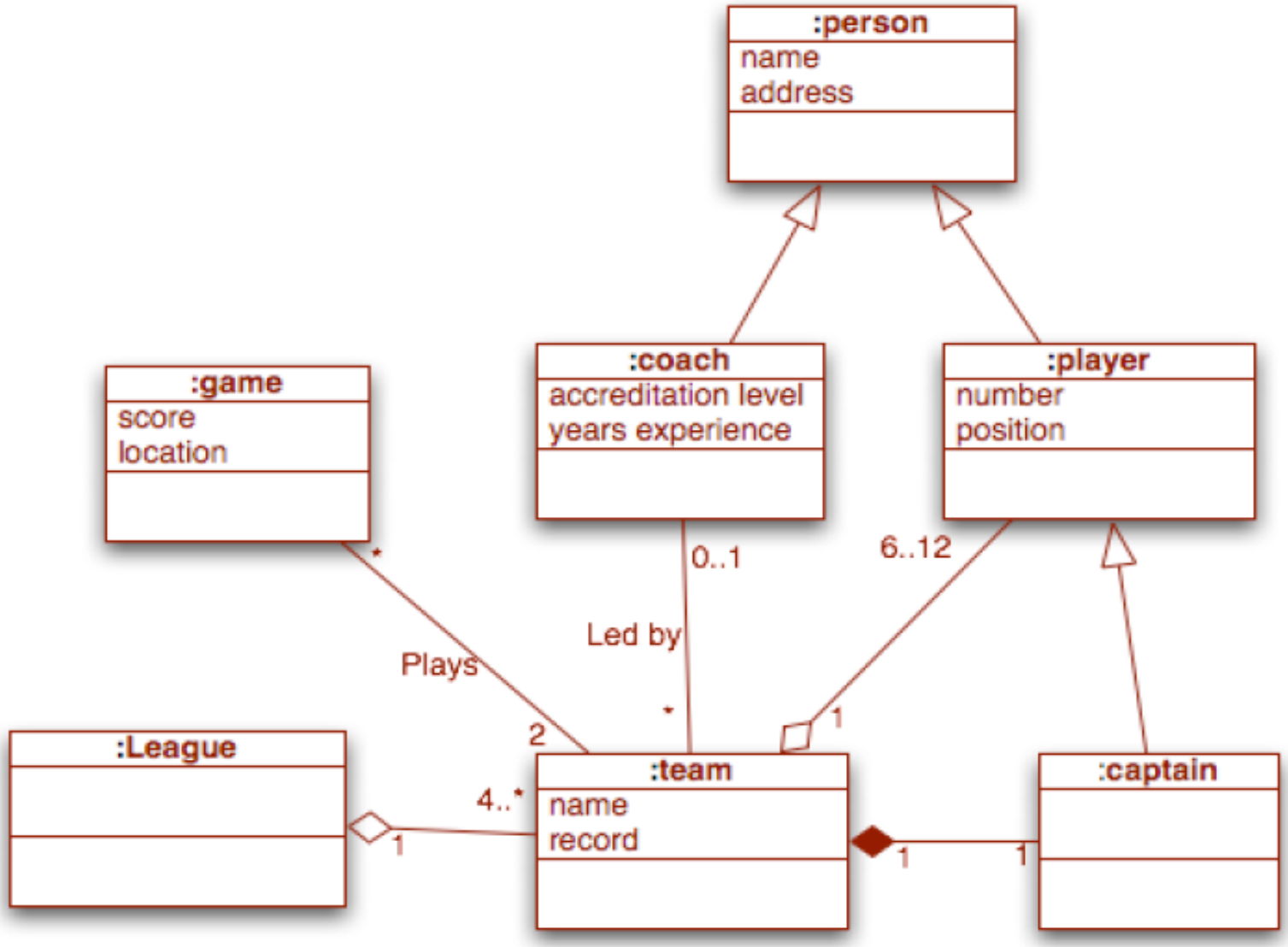
- Identify the association relationships among classes drawn from an information system for a school
  - School has one or more departments
  - Department offers one or more Courses
  - A particular Course will be offered by only one Department
  - Department has Instructors and Instructors can work for one or more Departments
  - Student can attend any number of Courses in a School
  - Every Course may have any number of Students
  - Instructors can teach zero or more Courses
  - Same Course can be taught by different Instructors
  - Students can be enrolled in more than one School
  - For every Department, there is exactly one Instructor who is the chairperson

Modeling Structural Relationships



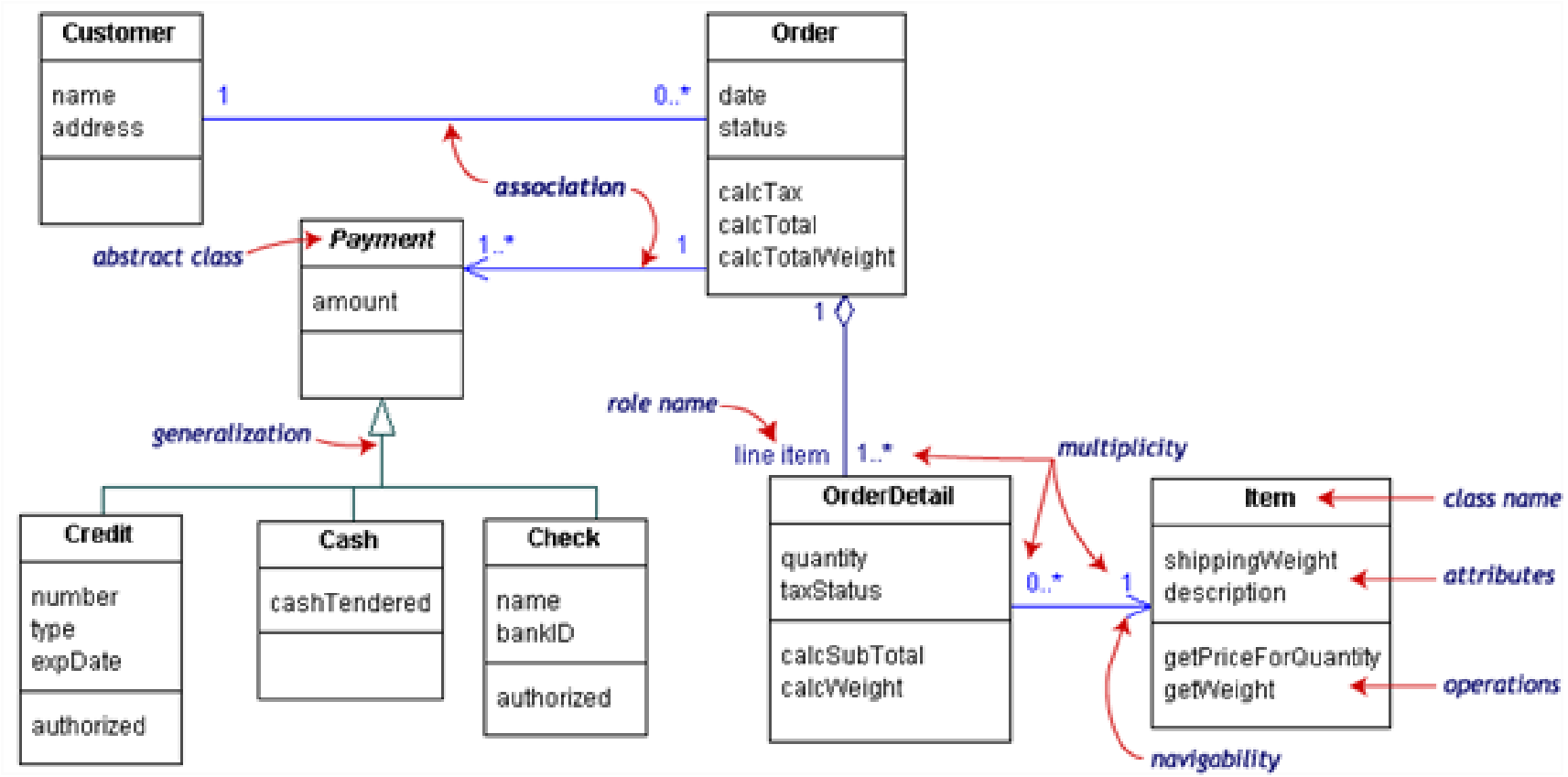
Class Diagram Example

A hockey league is made up of at least four hockey teams. Each hockey team is composed of six to twelve players, and one player captains the team. A team has a name and a record. Players have a number and a position. Hockey teams play games against each other. Each game has a score and a location. Teams are sometimes lead by a coach. A coach has a level of accreditation and a number of years of experience, and can coach multiple teams. Coaches and players are people, and people have names and addresses.


:person
name
address
:coach
accreditation level
years experience
:player
number
position
:game
score
location
:League
:team
name
record
:captain
0..1
6..12
Led by
Plays
*
*
2
1
4..*
1
1
1

Models a customer order from a retail catalog.

A customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.


Customer
name
address
1
0..*
Order
date
status
calcTax
calcTotal
calcTotalWeight
association
abstract class
Payment
amount
1..*
1
1
generalization
role name
line item
1..*
multiplicity
Credit
number
type
expDate
authorized
Cash
cashTendered
Check
name
bankID
authorized
OrderDetail
quantity
taxStatus
calcSubTotal
calcWeight
0..*
1
Item
shippingWeight
description
getPriceForQuantity
getWeight
class name
attributes
operations
navigability