UML (Class Diagrams)

Overview

- Class diagrams are for visualizing, specifying and documenting the system from a static perspective.
- Class diagrams indicate type of relationship between classes
- Class diagrams will have different levels of detail (abstraction) depending on where we are in the software development process.

- ✓ Identify Classes
- ✓ Class representation in class diagram
- Relationships among classes
  - ✓ Association
  - Generalization
  - Dependency
  - Realization

Generalization

- Generalization is a type of relationship where
  - One of the classes is the base (or parent) class; the other class is the derived (or child) class.
  - used to show a '**kind-of**' relationship.
  - another name for inheritance.
    - **graphically represented by a solid line with an open triangular arrowhead on the base class end.**
- The parent class has no knowledge of the child class.
- All OO programming languages support single inheritance; some (C++) also support multiple inheritance.

Generalization

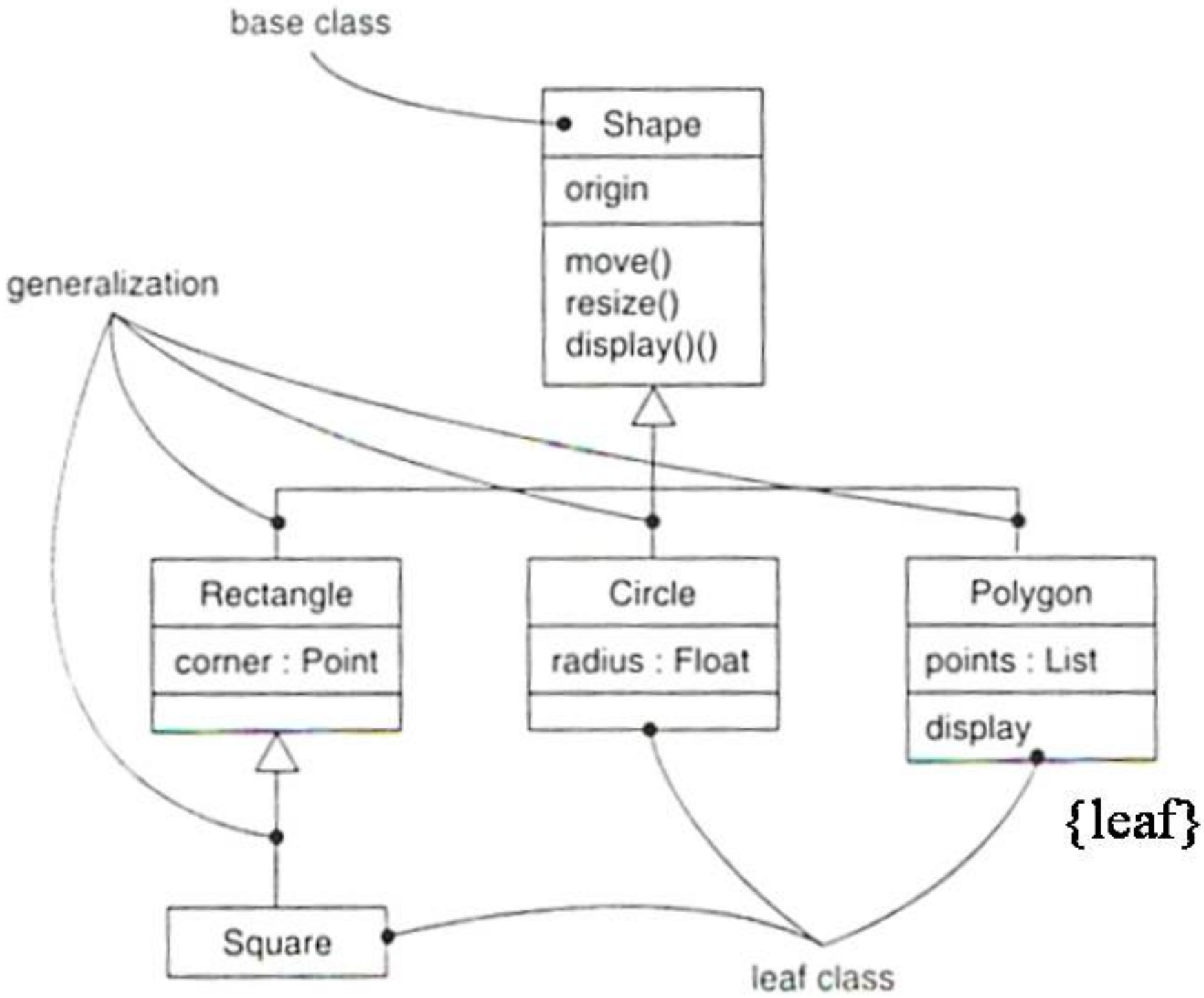
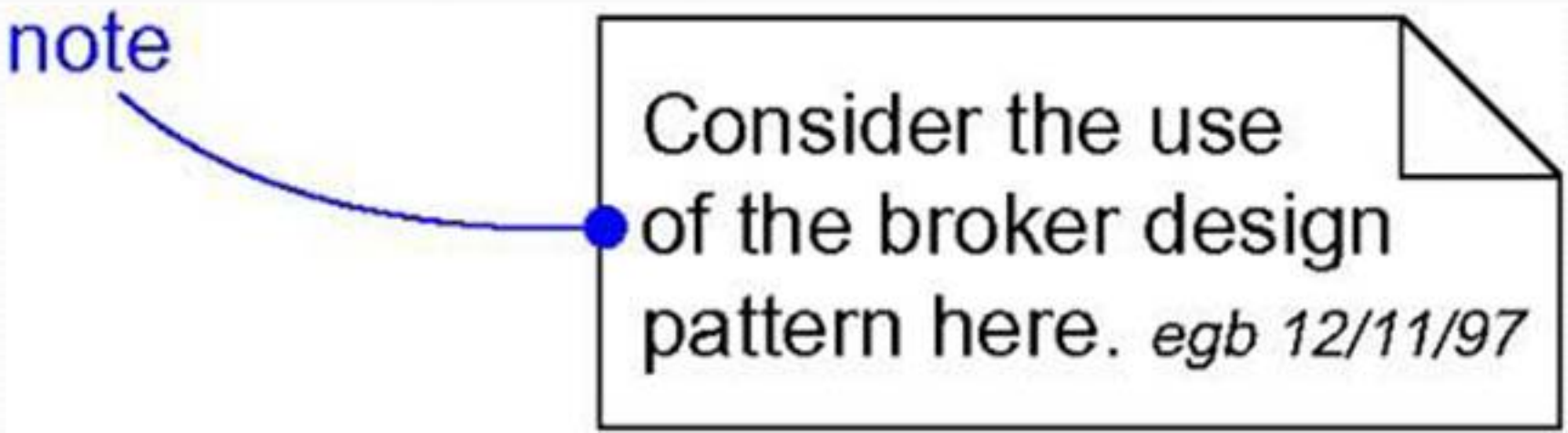


Figure 5-3: Generalization

Notes and Comments

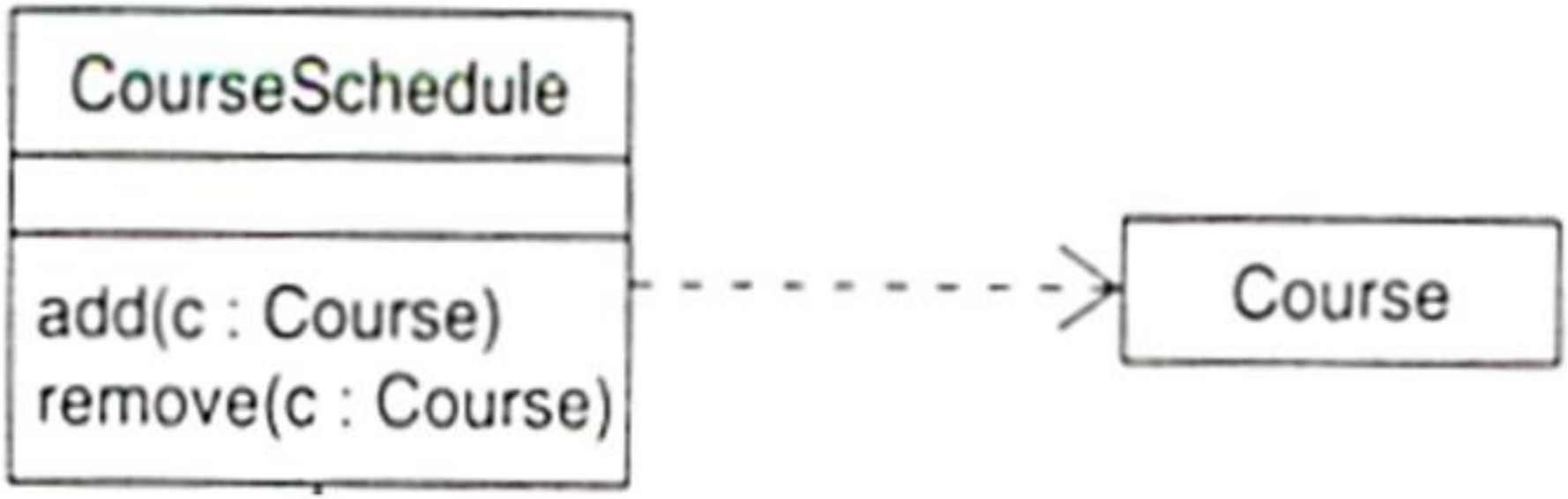
- Why comments in programming languages?
- Use *notes* to embed comments into UML diagrams.
- A *note* may stand alone or be attached to a graphical element.
  - A dashed line is used as the connector
- Use notes when a picture is not enough.

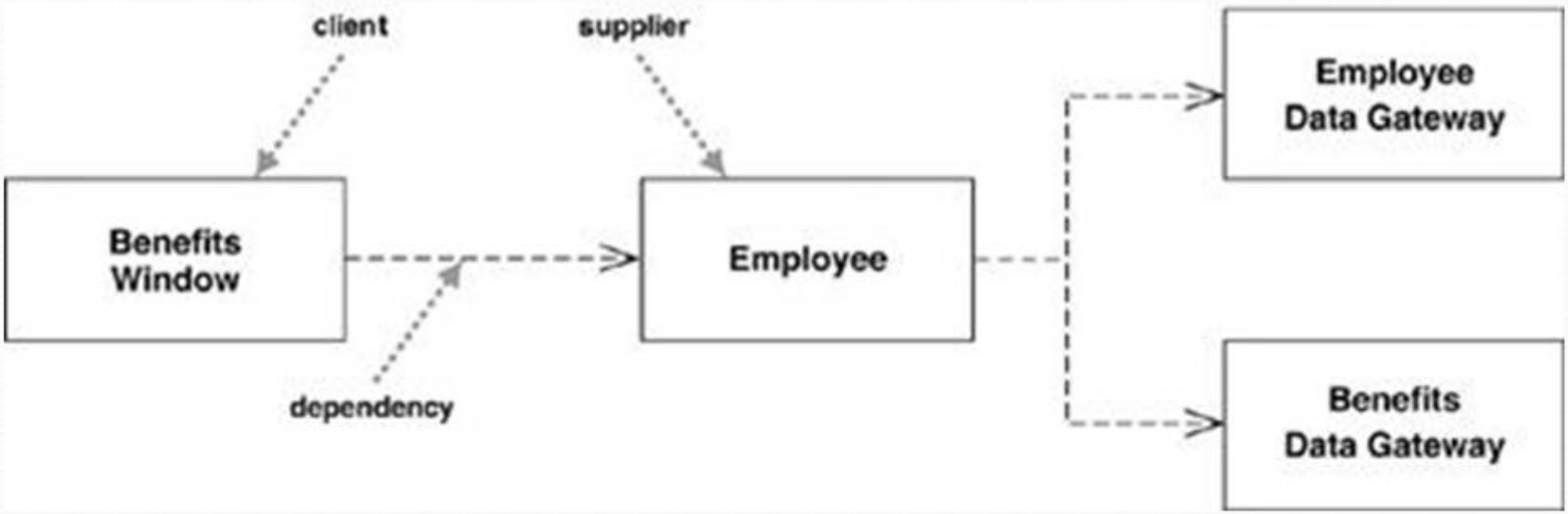


Dependency

- An indication that **one class 'uses' another class.**
- If the specifications of 'used' class changes it can have an impact on the 'using' class.
- Graphically represented as a dashed line with an open arrowhead on one end.
- Rarely labeled.
- The 'used' class is frequently an argument to one of the member functions of the 'using' class or ...
- The "used" class has no knowledge of the "using" class.

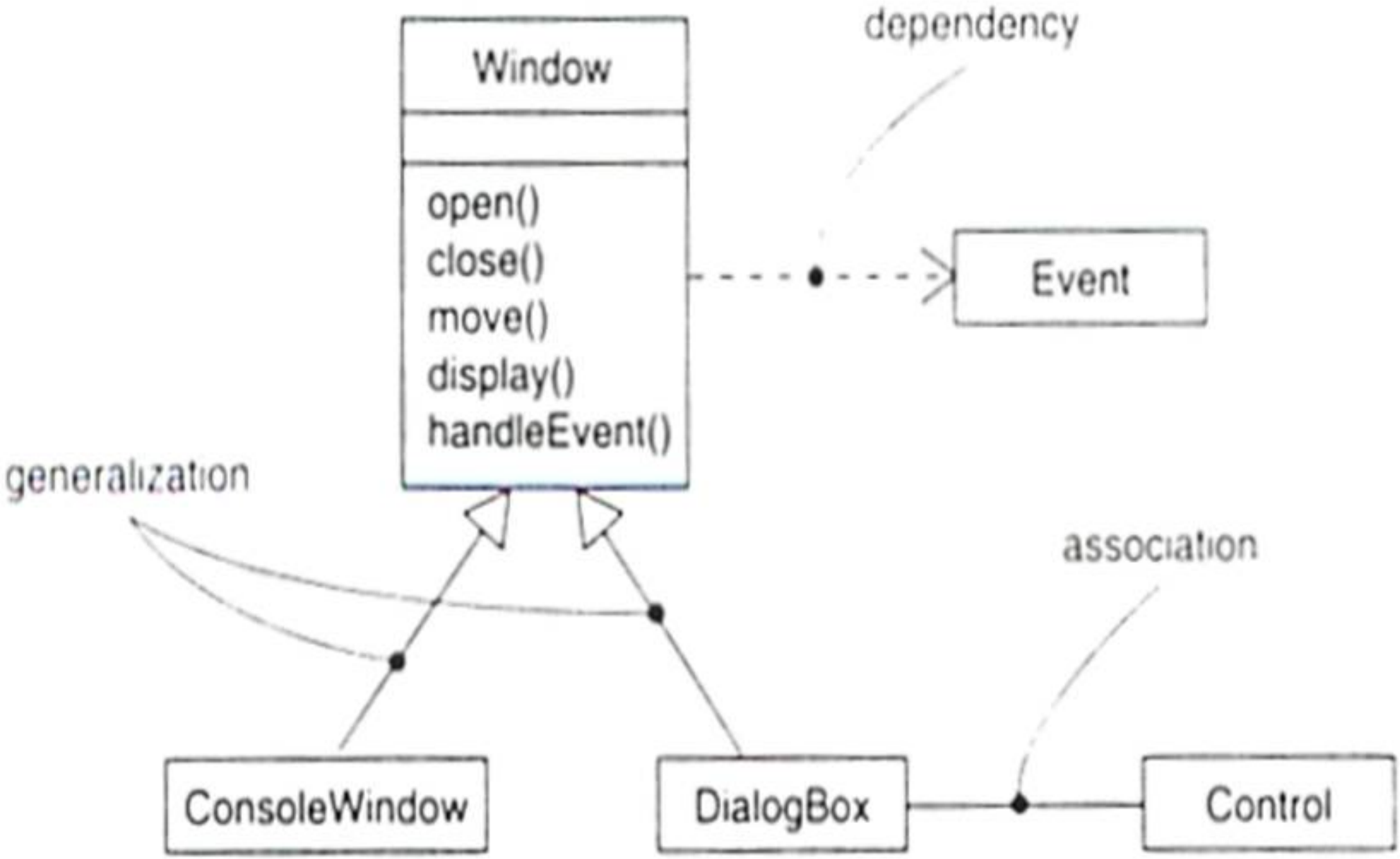
Dependency Relationships

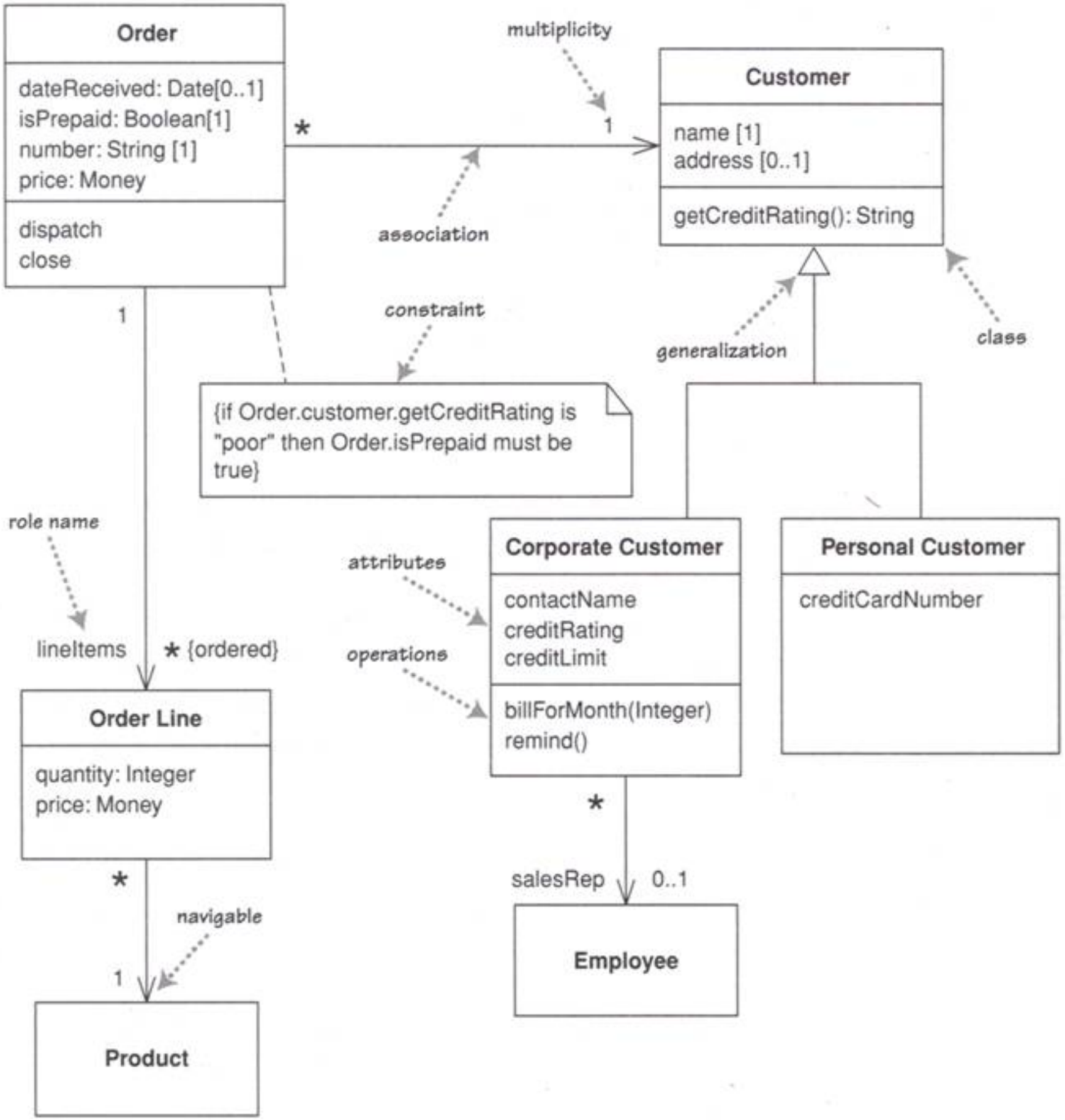




The general rule is to minimize dependencies

Common Class Relationships



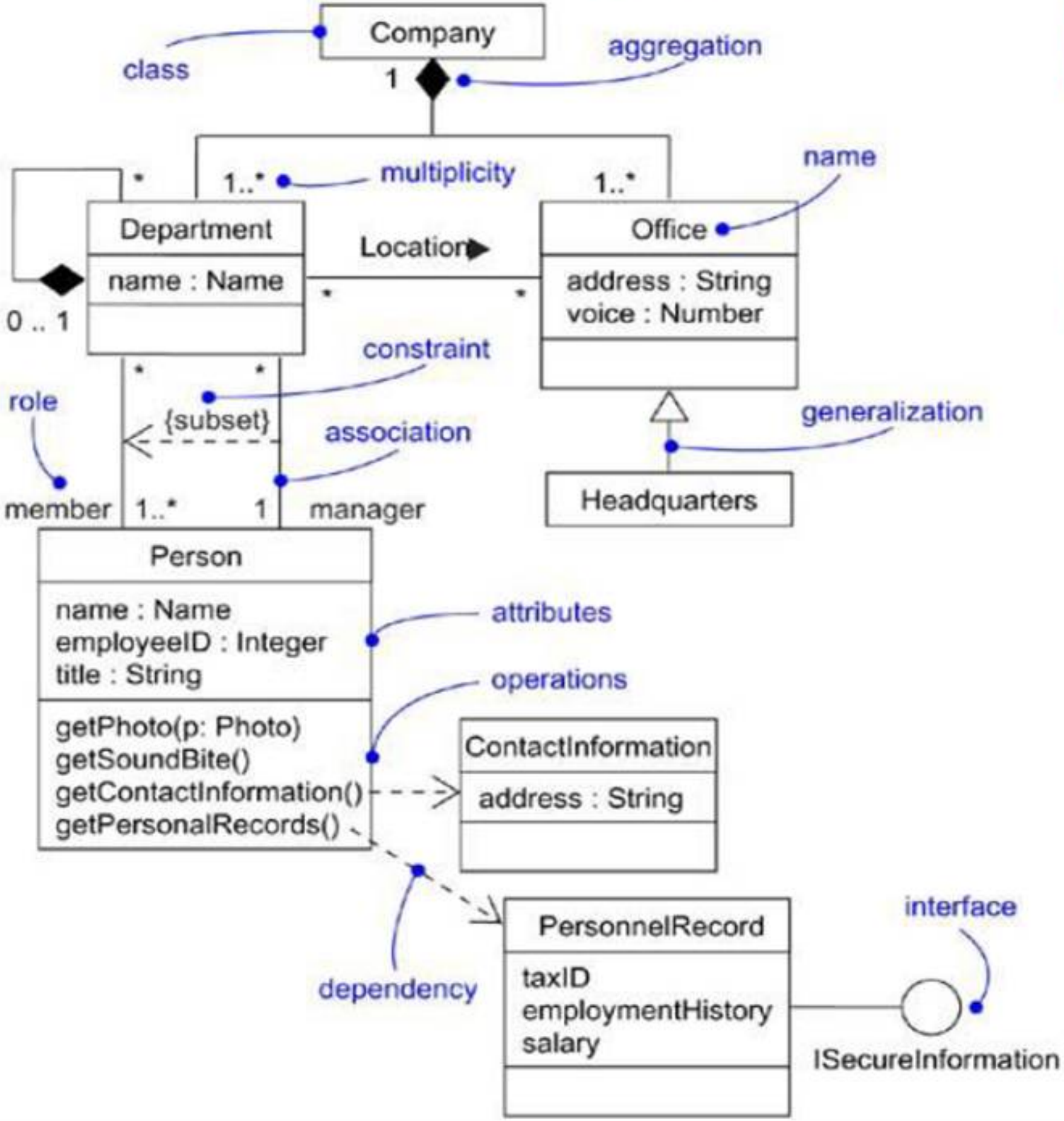


Simple Class Diagram

Common Modeling Techniques

- Modeling inheritance (generalization)
  - Make it a priority to look for possible inheritance relationships. This will reduce code redundancy.
    - Use the "ClassA 'is-a-kind-of' ClassB" test.
  - Look for classes that have similar attributes and behaviors and then factor those attributes and behaviors into a common base class.
  - Do not model relationships using multiple inheritance unless the implementation language supports multiple inheritance

A Class Diagram



Classifiers

- Modeling elements that can have instances are called classifiers
- A classifier generally has structural features (in the form of attributes), as well as behavioral features (in the form of operations)
- Every instance of a given classifier shares the same features.
- The most important kind of classifier in the UML is the class.

Classifiers

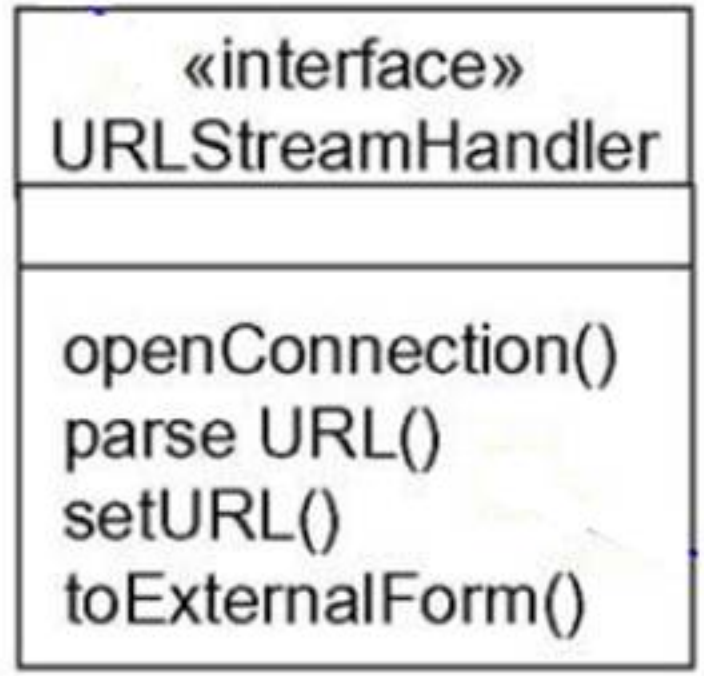
- UML provides a number of other classifiers also
  1. Interfaces
  2. Datatype
  3. Signal
  4. Component
  5. Node
  6. Use case
  7. Subsystem

Interfaces

- An interface is a named collection of operations used to specify a service of a class or of a component.
- An interface may not have attributes
- An interface may participate in generalization, association, and dependency relationships.
- An interface may participate in realization relationships.
- An interface specifies a contract for a class
- A class may realize many interfaces

Keywords

- One of the challenges of a graphical language is that you have to remember what each symbol mean.
- With too many of them, users find it difficult to remember what they mean.
  - Use keywords instead



- For example a UML **interface is a class that has only public** operations, with no method bodies.
- Because it's a special kind of class, it is shown using the class icon with the keyword «interface».

Abstract Classes

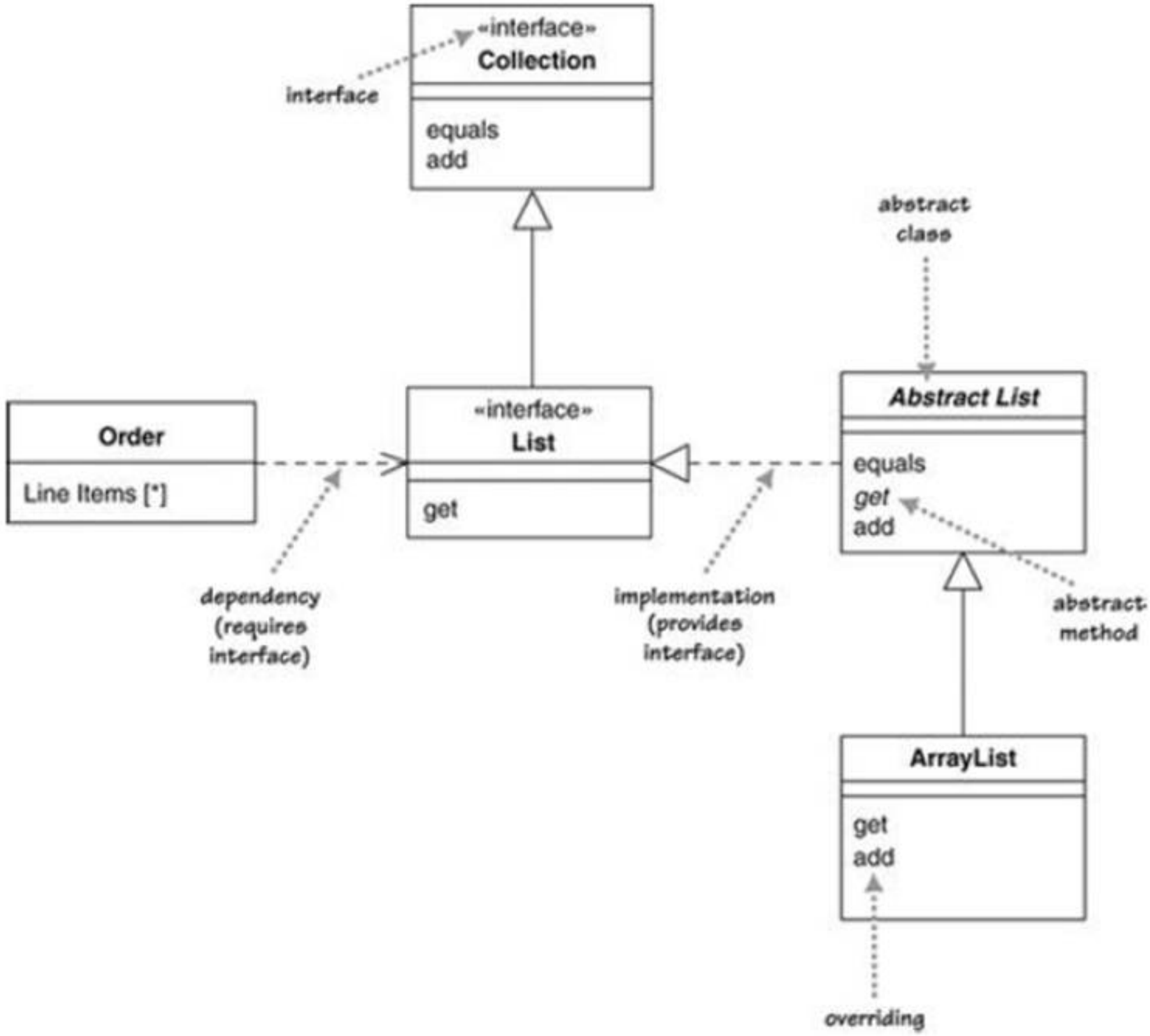
- An abstract class has one or more operations that are abstract.
- An **abstract operation has no implementation**
- An **abstract class is a class that cannot be directly instantiated.**
- Only occur in the context of an inheritance hierarchy.
- The most common way to indicate an abstract class or operation in the UML is to *italicize the name*.
- OR use the label: {abstract}.

Abstract Classes

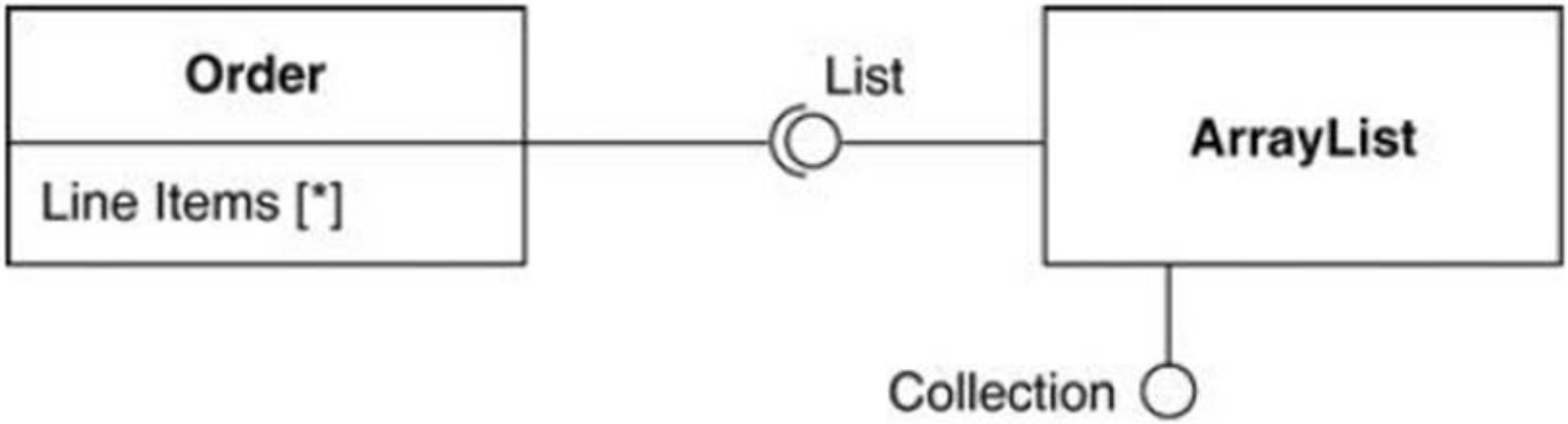
- Not all OO programming languages directly support abstract classes.
- In C++ abstract classes contain one or more pure virtual functions
  - virtual int myFunction(int x) = 0;
  - Pure virtual functions have no function body.
- When a virtual function is made pure, any derived class must provide its own definition

Interfaces

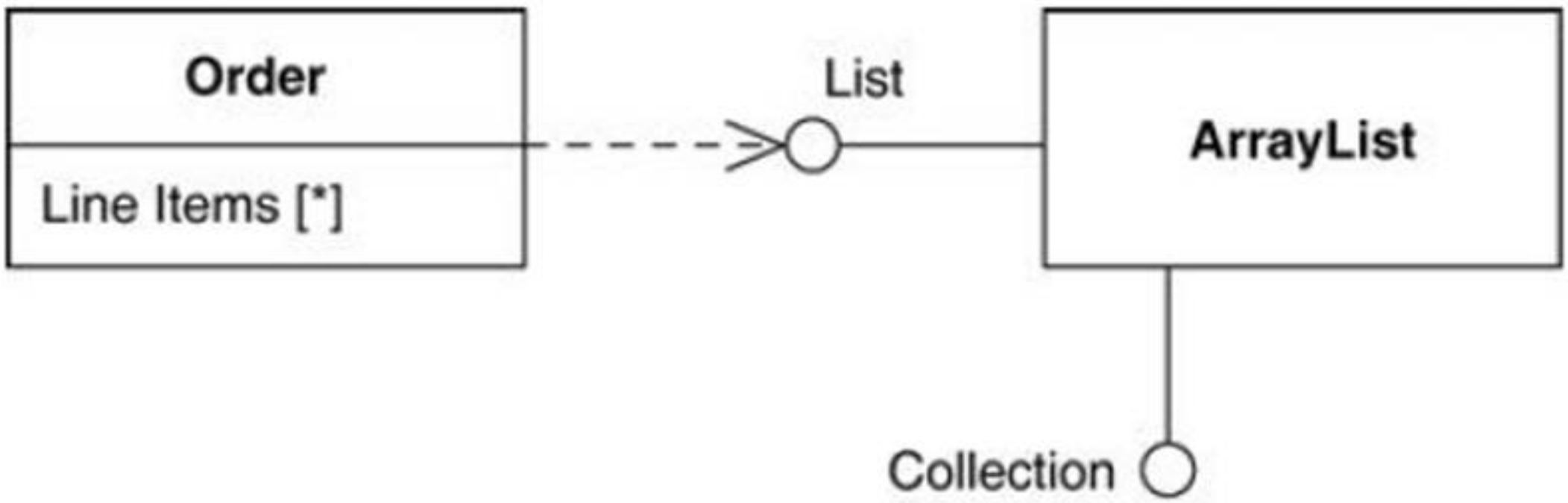
- An interface is a class that has no implementation
- Classes have two kinds of relationships with interfaces: providing and requiring.
- A class **provides an interface**
  - **A class can do that by** implementing the interface or implementing a subtype of the interface.
- A class **requires an interface if it needs an instance of that interface in order to work.**
  - This is having a dependency on the interface.


«interface»
Collection
equals
add
interface
abstract
class
Order
Line Items [*]
«interface»
List
get
Abstract List
equals
get
add
dependency
(requires
interface)
implementation
(provides
interface)
abstract
method
ArrayList
get
add
overriding

Ball and Socket Notation



Older lollipop Notation



Aggregation and Composition

- A plain association between two classes represents a structural relationship between peers
- Aggregation is ...
  - A strong form of association.
- Used to show a 'has-a' or a 'whole-part' relationship.
- Graphically represented as a solid line with an open diamond on the 'whole' end.

Aggregation and Composition

- Composition is ...
  - Used to show a 'contains-a' or a 'whole-part' relationship between two classes
  - Graphically represented as a solid line with a solid diamond on the 'whole' end.
- If the 'whole' object is destroyed, the part object will also be destroyed.
  - The whole and part have concurrent lifetimes.
  - Part objects may not be shared with other whole objects