

# Applying UML & Patterns (3<sup>rd</sup> ed.)

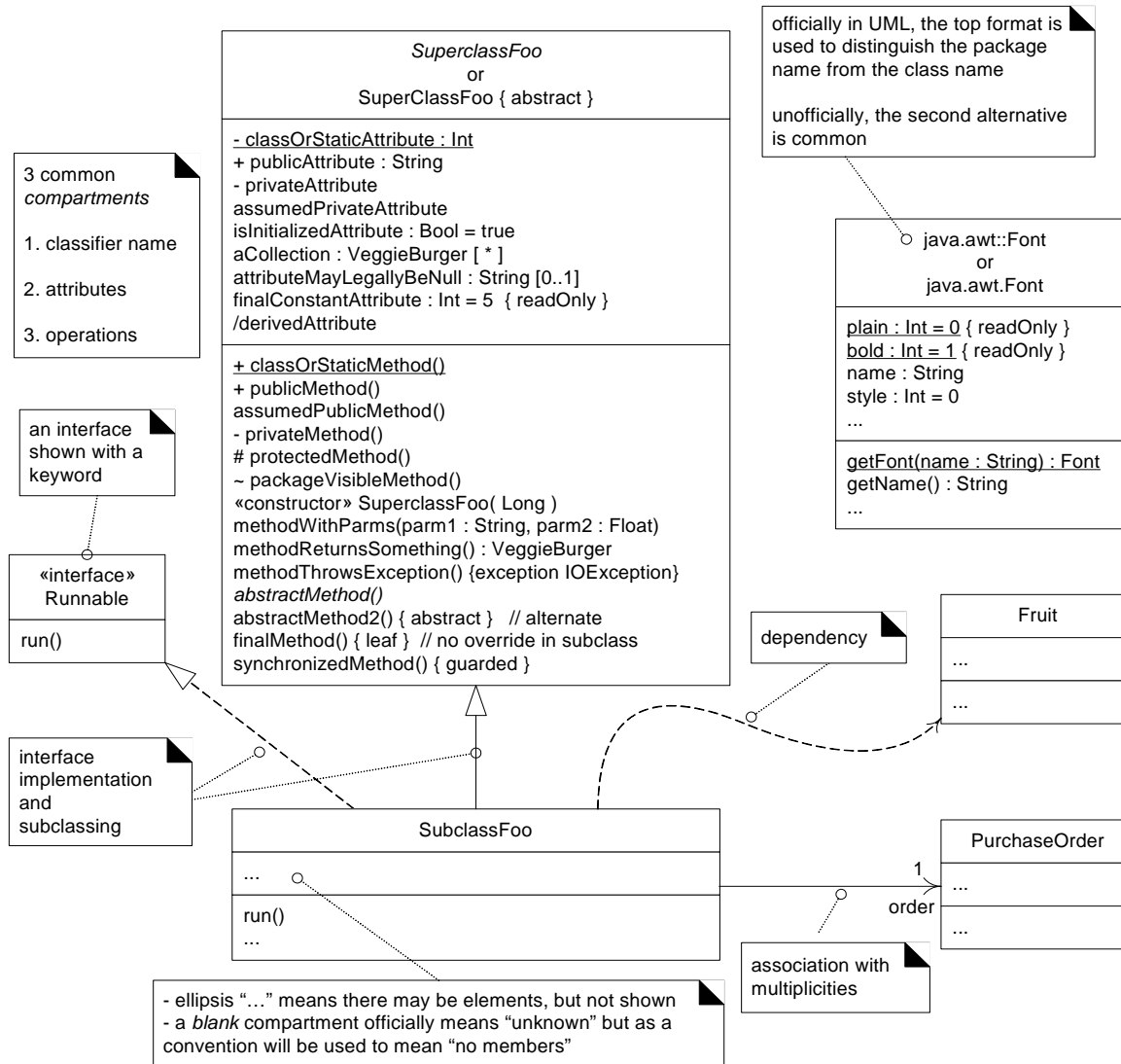
## Chapter 16

### UML CLASS DIAGRAMS

**This document may not be used or altered without the express permission of the author.**

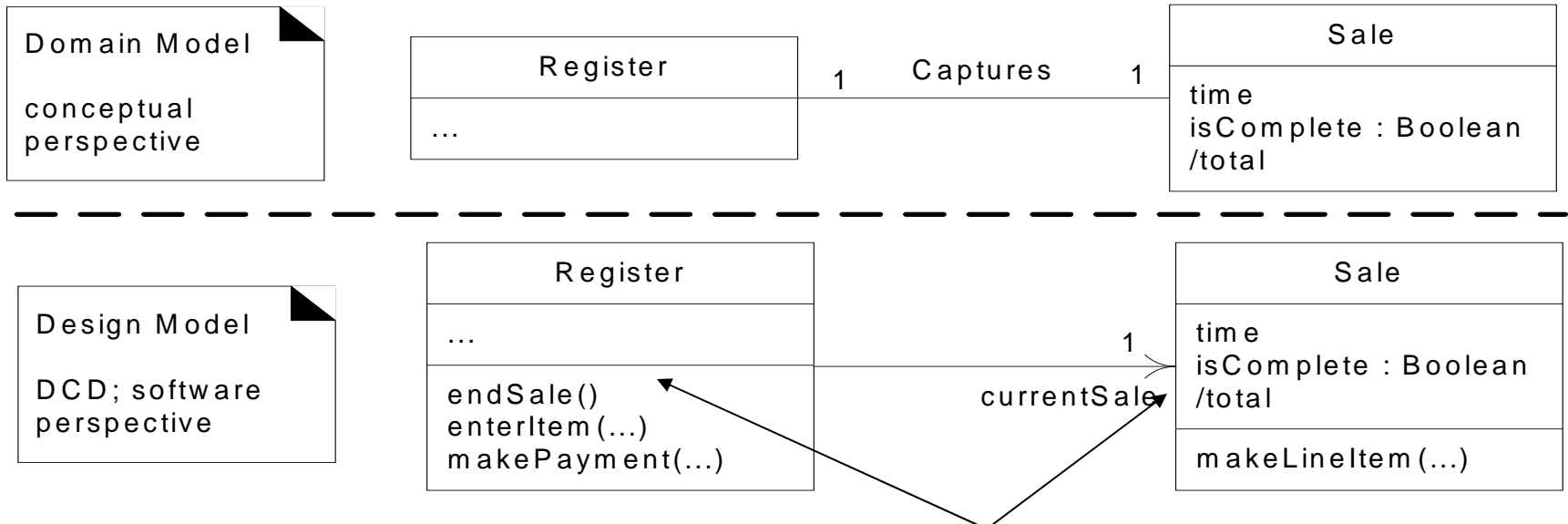
**Dr. Glenn L. Ray**  
School of Information Sciences  
University of Pittsburgh  
[gray@sis.pitt.edu](mailto:gray@sis.pitt.edu) 412-624-9470

# Fig. 16.1



# Fig. 16.2

Domain model is the analysis class diagram  
Don't show methods



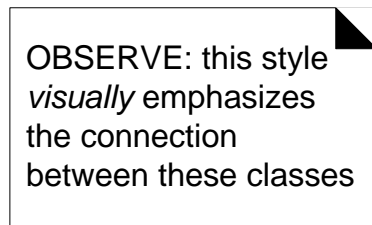
Avoid showing no-argument constructors & getters/setters

Design model shows methods and visibility (arrowhead on association)  
Register has reference to Sale; Sale does not have reference to Register

# Fig. 16.3

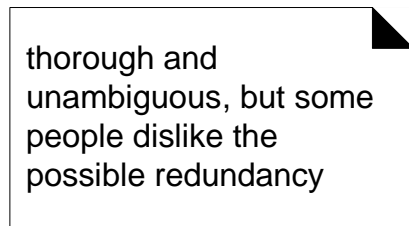
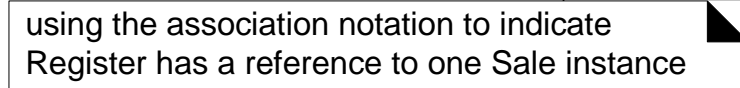


Avoid



Rolename is attribute name

Preferred



Avoid

# Class Diagrams

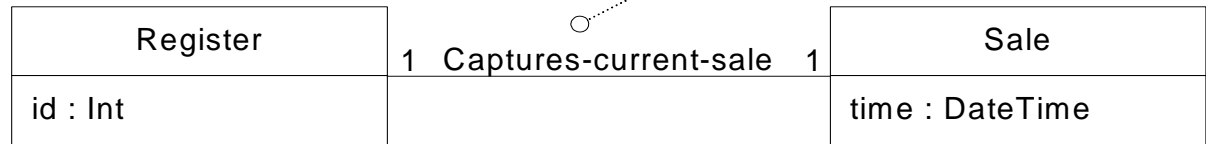
- UML notation for attributes
  - You don't have to use entire notation
  - Visibility
    - + (public)
    - - (private)
  - Assume attributes are private & methods public unless stated otherwise

visibility attributeName : type multiplicity = default {property-string}

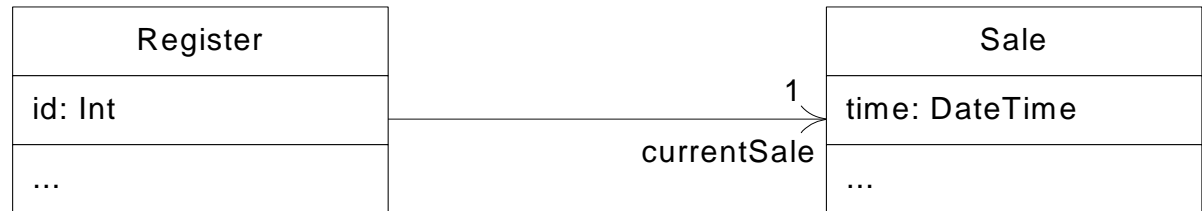
# Fig. 16.4

the association *name*, common when drawing a domain model, is often excluded (though still legal) when using class diagrams for a software perspective in a DCD

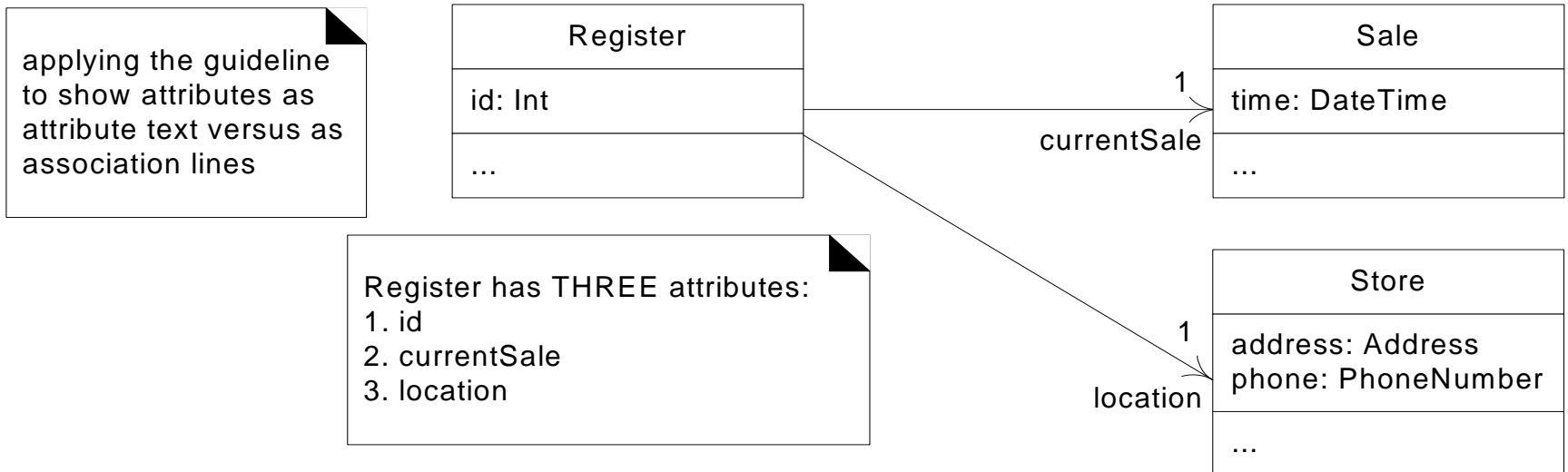
UP Domain Model  
conceptual perspective



UP Design Model  
DCD  
software perspective

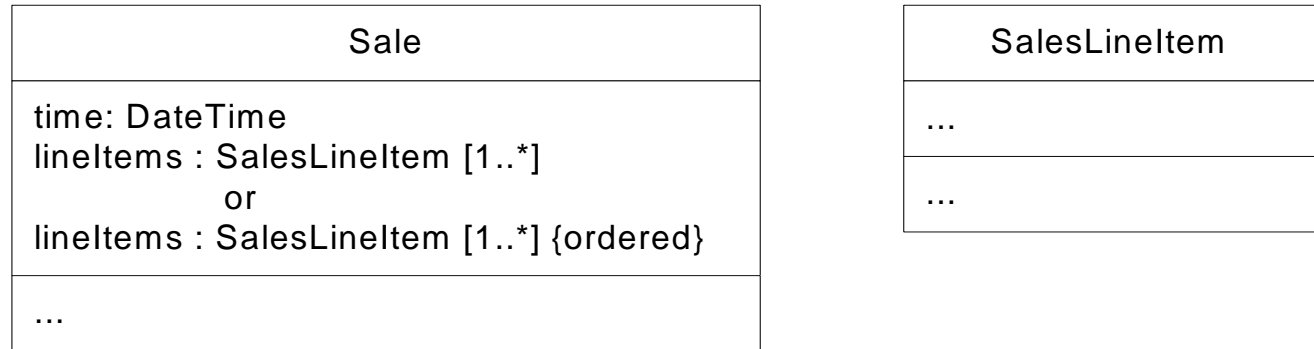


# Fig. 16.5



Showing Sale & Store attributes as associations visually emphasizes relationships important in object collaboration

# Fig. 16.6



Two ways to show a collection attribute



notice that an association end can optionally also have a property string such as {ordered, List}

Preferred, less visual clutter  
1:N association implies a collection attribute



# UML Operation Syntax

visibility operationName(parameter list) : return-type {property-string]

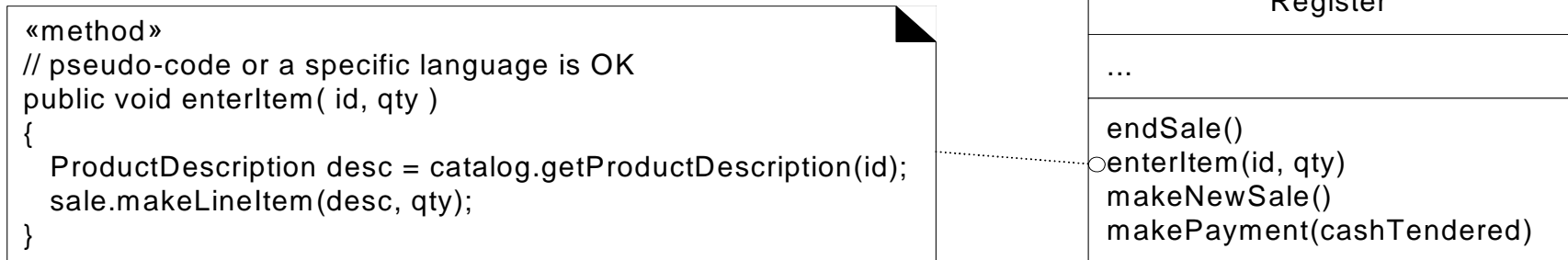
## Guidelines

- Show return types

- Assume operations are public unless otherwise depicted

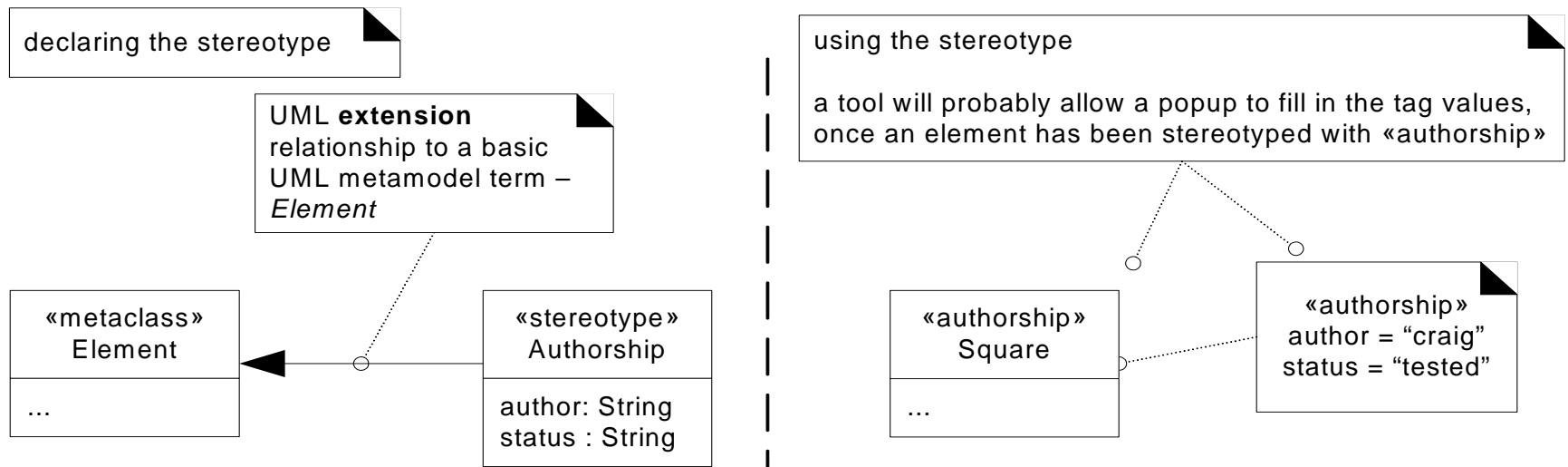
- Assume getters & setters, i.e., don't show them on model

# Fig. 16.7



UML note symbol showing a method body

# Fig. 16.8



Stereotypes allow you to refine (i.e., extend) model elements  
Groups of related model elements form a UML profile  
Can substitute <<..>> for «..»

# Class Diagrams

- More UML notation
  - Generalization
    - Same as inheritance when in design mode
    - Solid line with open arrow

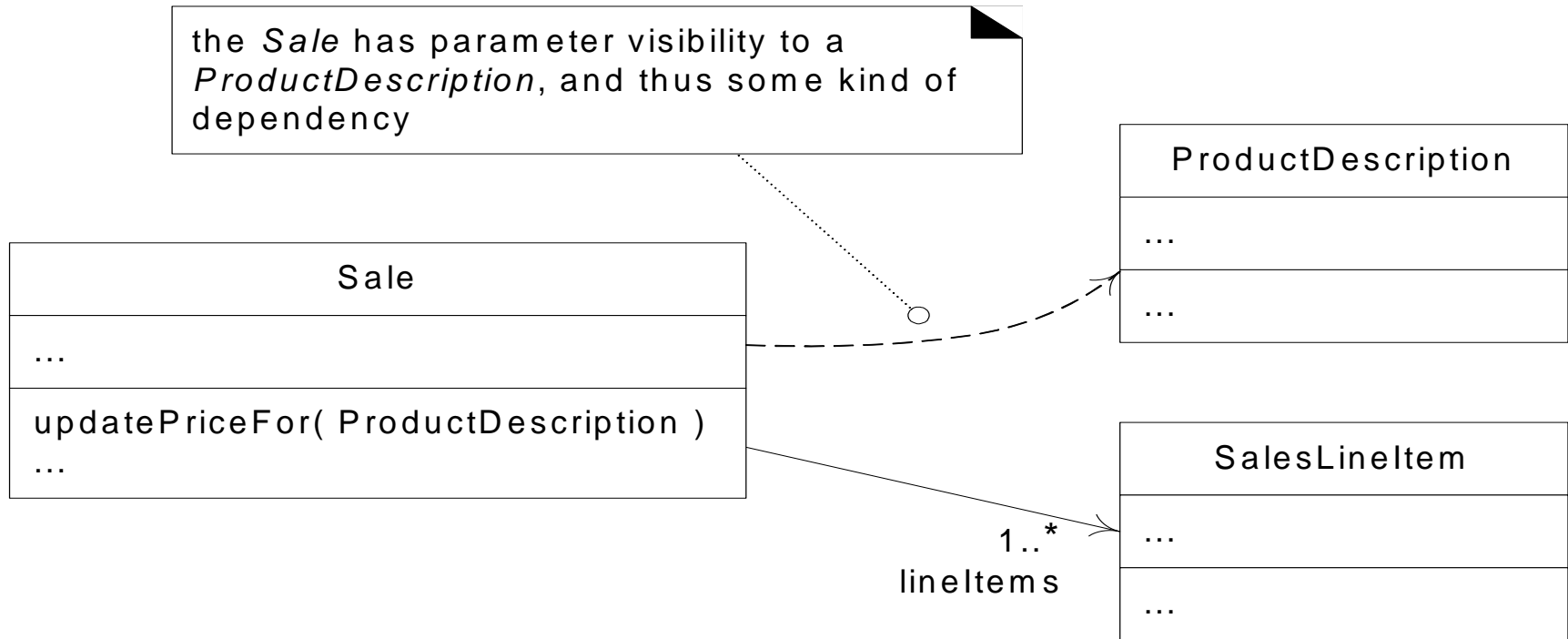


- To indicate *abstract*, use italics
- For final classes or operations, use *{leaf}*

# Class Diagrams

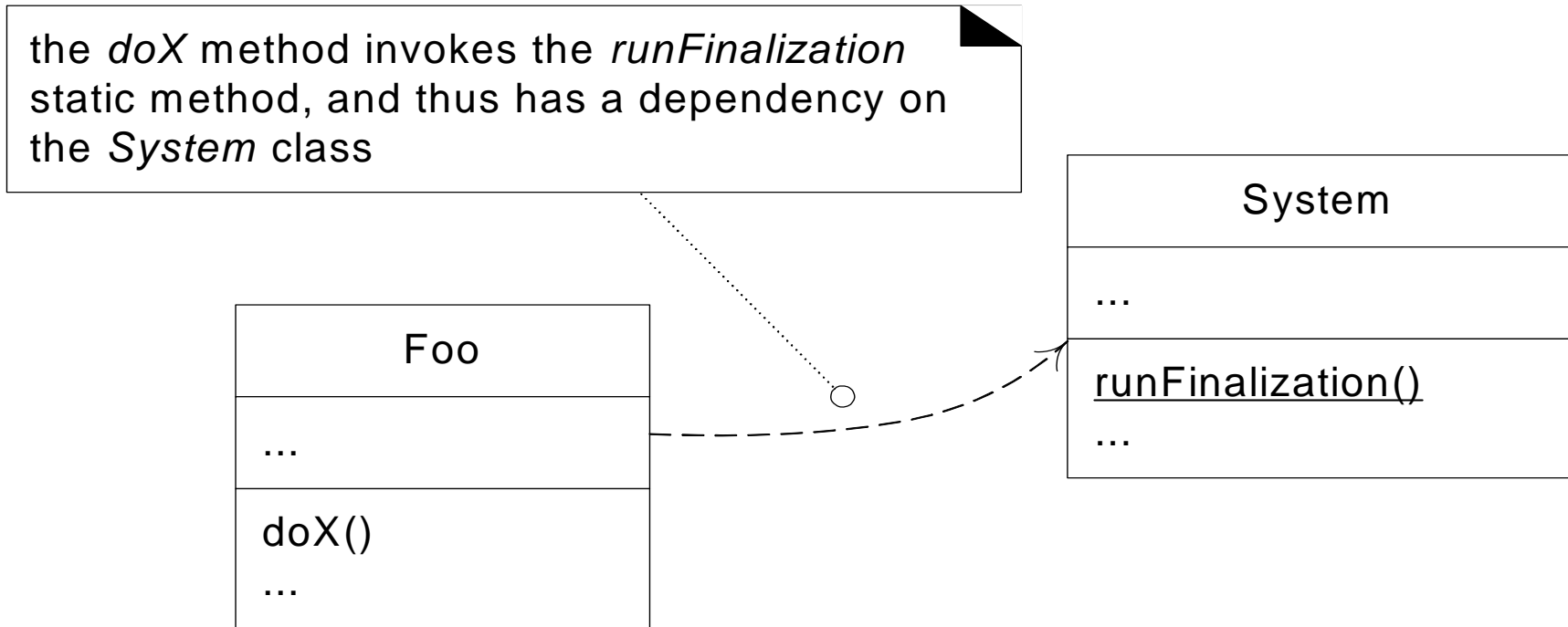
- Dependency
  - When a client has knowledge of a server
  - Changes in server affects client
  - Client and server are ‘coupled’
  - NOT a good thing! Leads to brittle design
  - Occurs anytime client has direct visibility to server
    - Can also be more subtle
    - We’ll cover types of visibility later

# Fig. 16.9



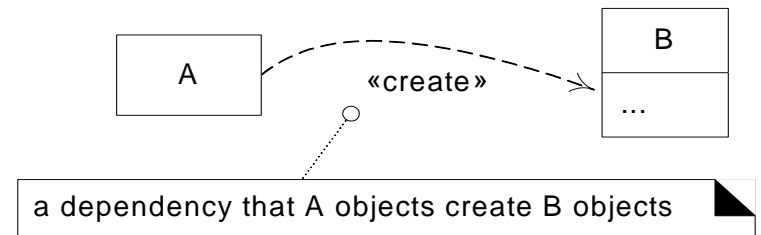
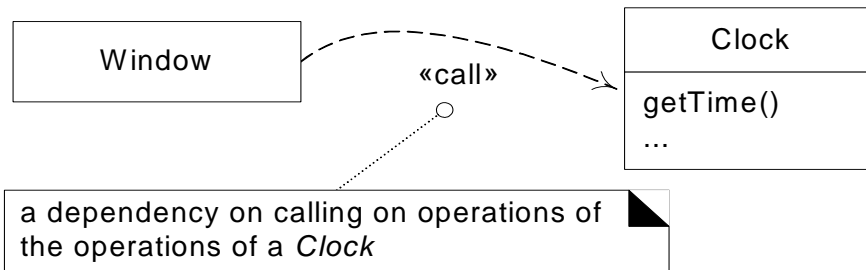
If *ProductDescription*'s public interface changes, then *Sale* may need to be modified

# Fig. 16.10



Dependency due to calling a static method

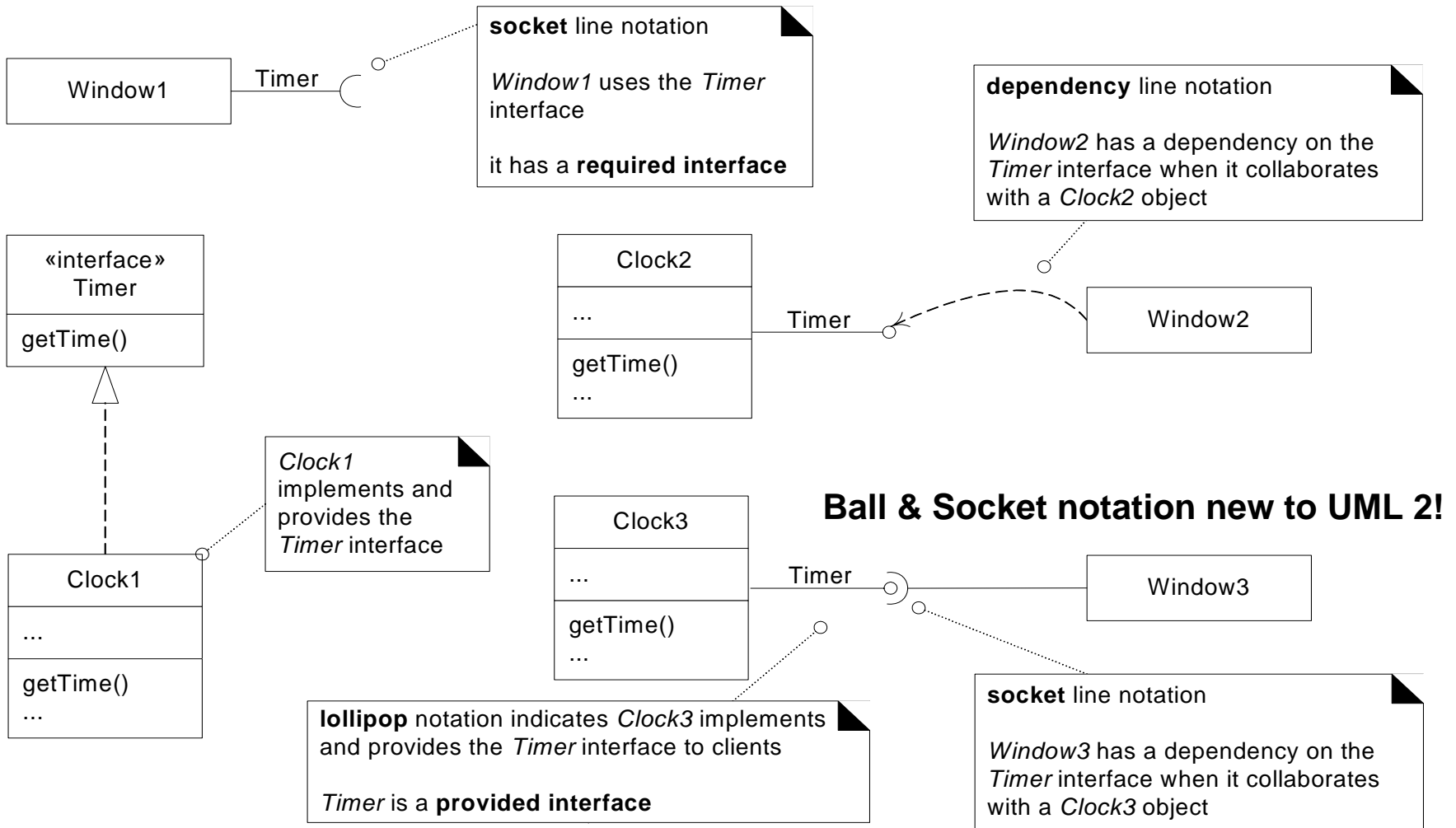
# Fig. 16.11



Using keywords or stereotypes to indicates type of dependency

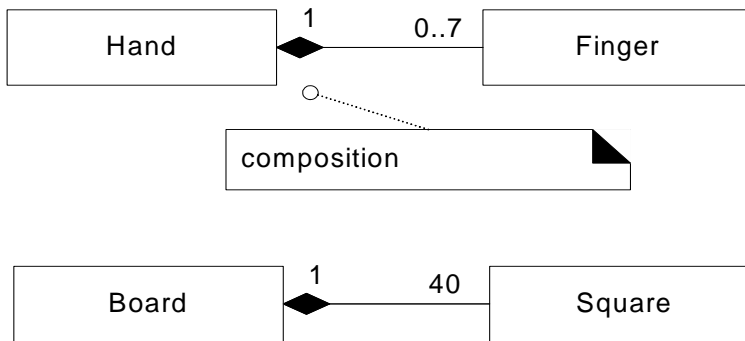


# Fig. 16.12



## Interface notation

# Fig. 16.13



composition means

-a part instance (*Square*) can only be part of one composite (*Board*) at a time

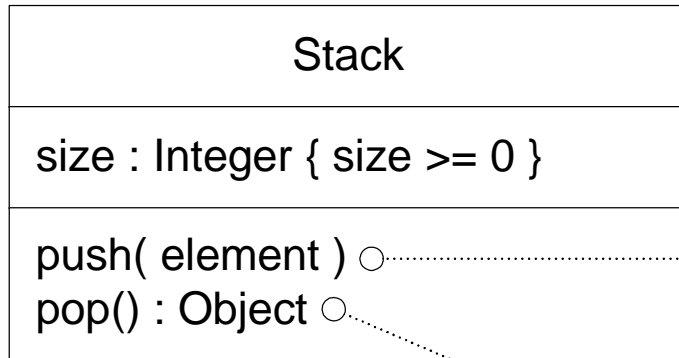
-the composite has sole responsibility for management of its parts, especially creation and deletion



- Aggregation is not a distinctive term (same as regular association)
- Use composition when appropriate
- Composition is the 'strong' form of aggregation

# Fig. 16.14

three ways to show UML constraints

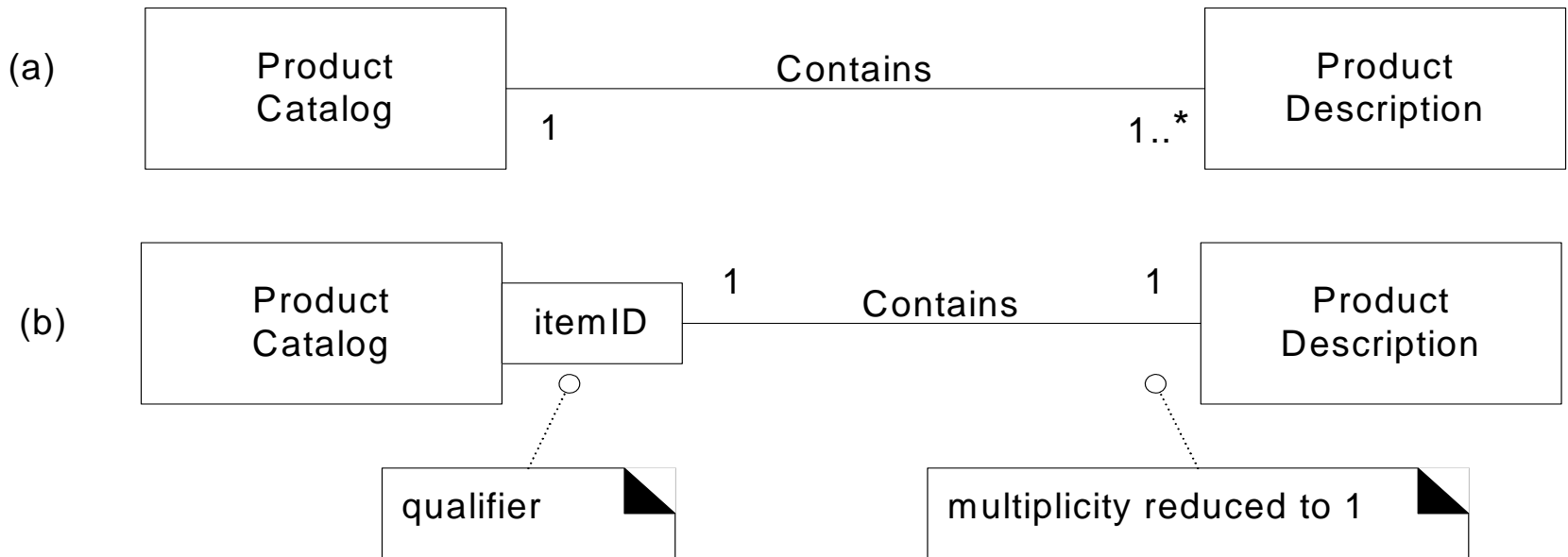


{ post condition: new size = old size + 1 }

```
{  
post condition: new size = old size - 1  
}
```

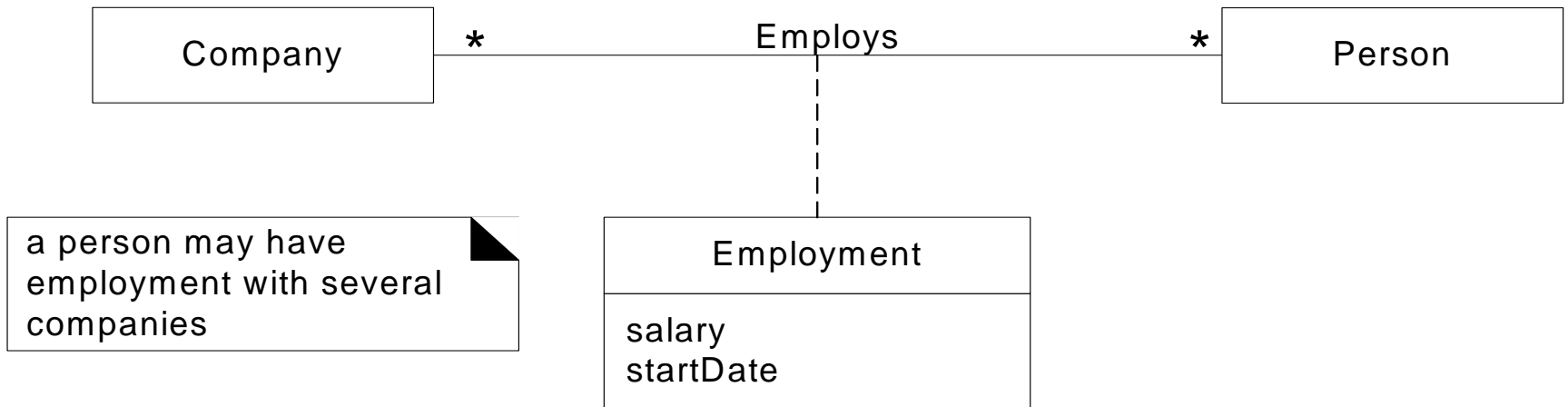
## Constraint notation

# Fig. 16.15



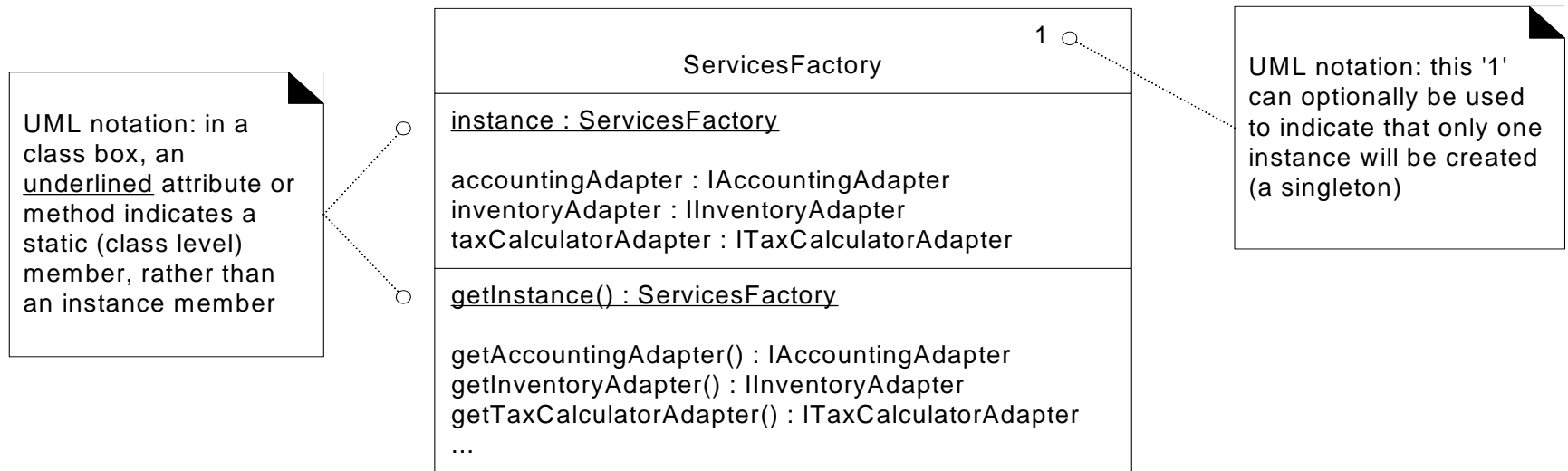
## Qualified associations

# Fig. 16.16



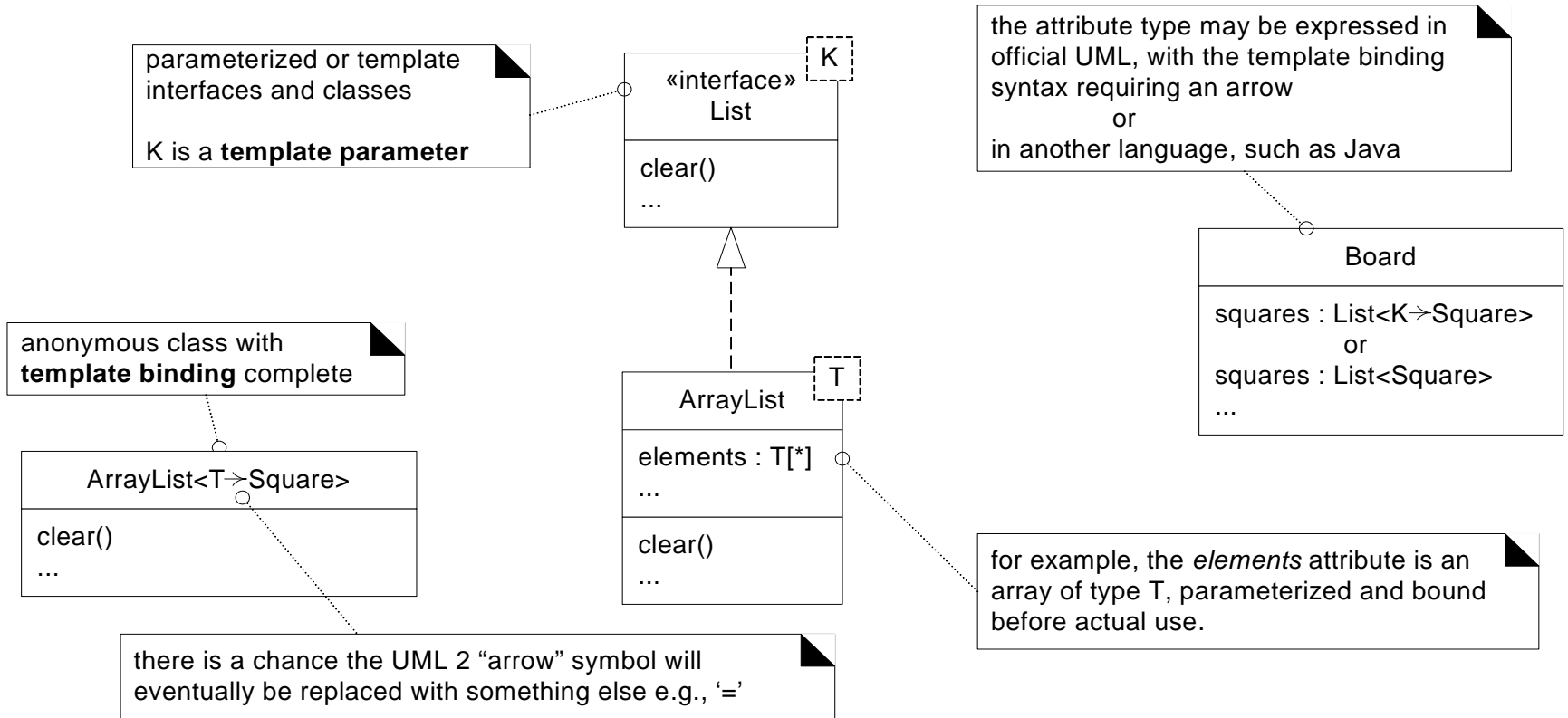
Association class simplifies many-to-many multiplicity

# Fig. 16.17



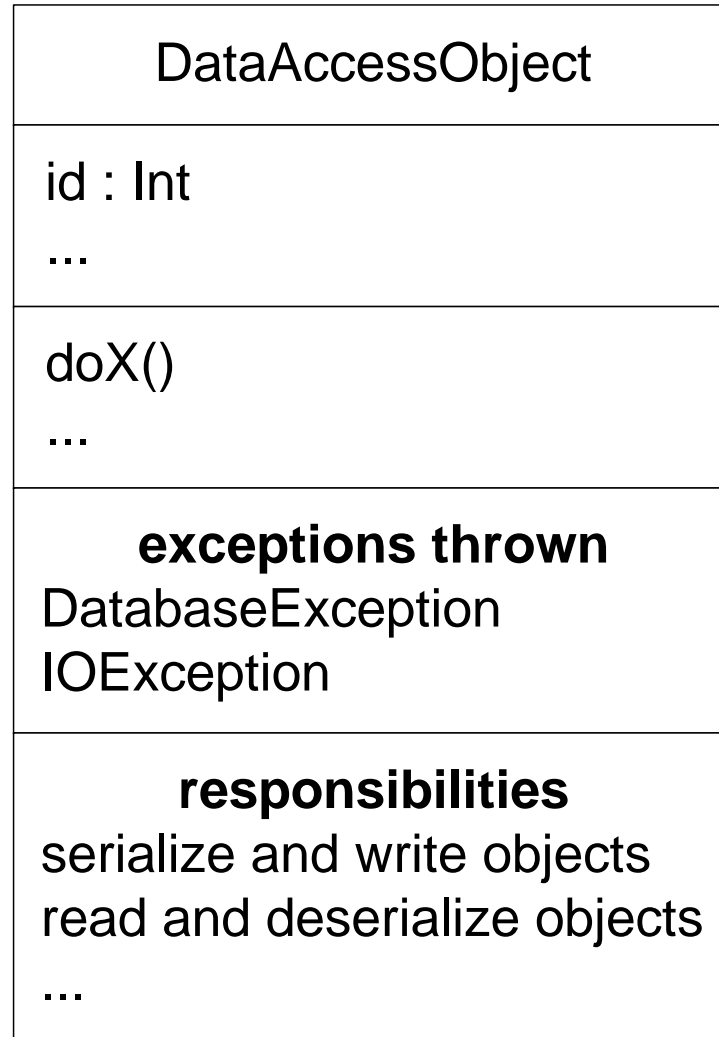
Singleton class: Only 1 instance can exist

# Fig. 16.18



## Template types are new to Java 1.5

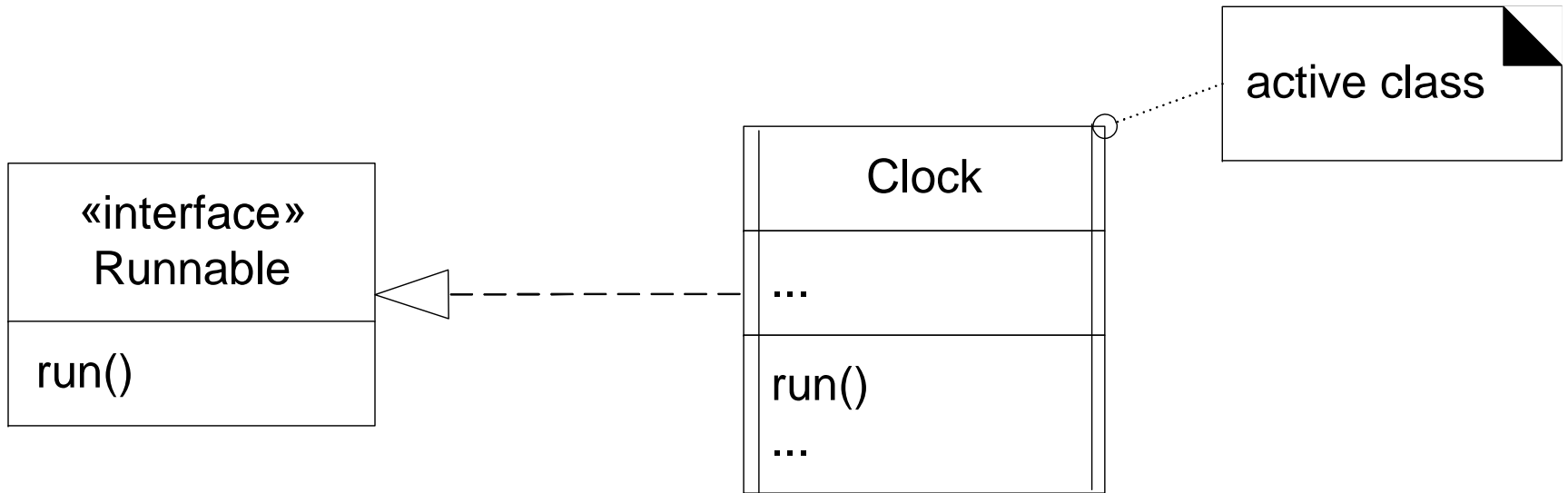
# Fig. 16.19



Additional compartments  
for a class

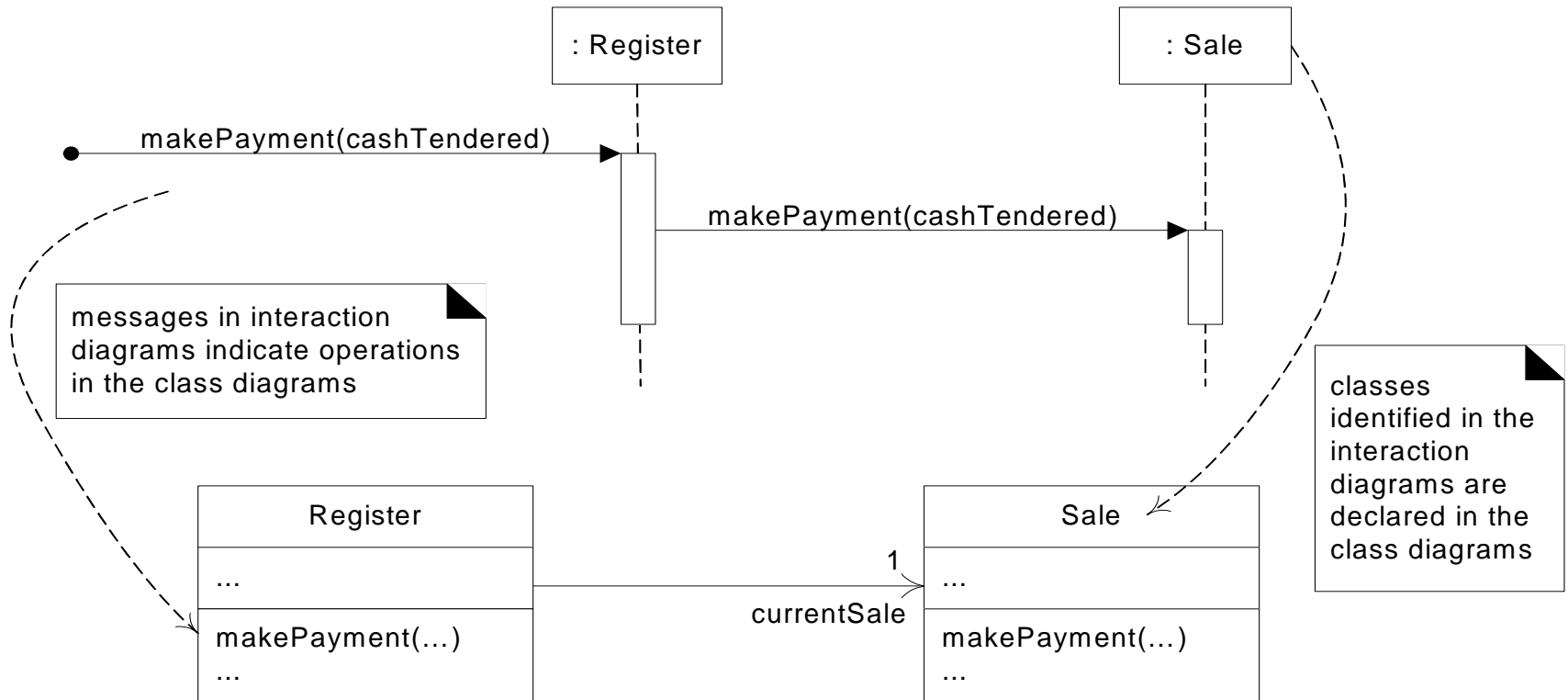


# Fig. 16.20



Active class executes in its own thread

# Fig. 16.21



Relationship between class & interaction diagrams  
Models must be consistent with each other!!