Applying UML and Patterns

**An Introduction to
Object-oriented Analysis
and Design
and Iterative Development**

Part III Elaboration Iteration I – Basic^2

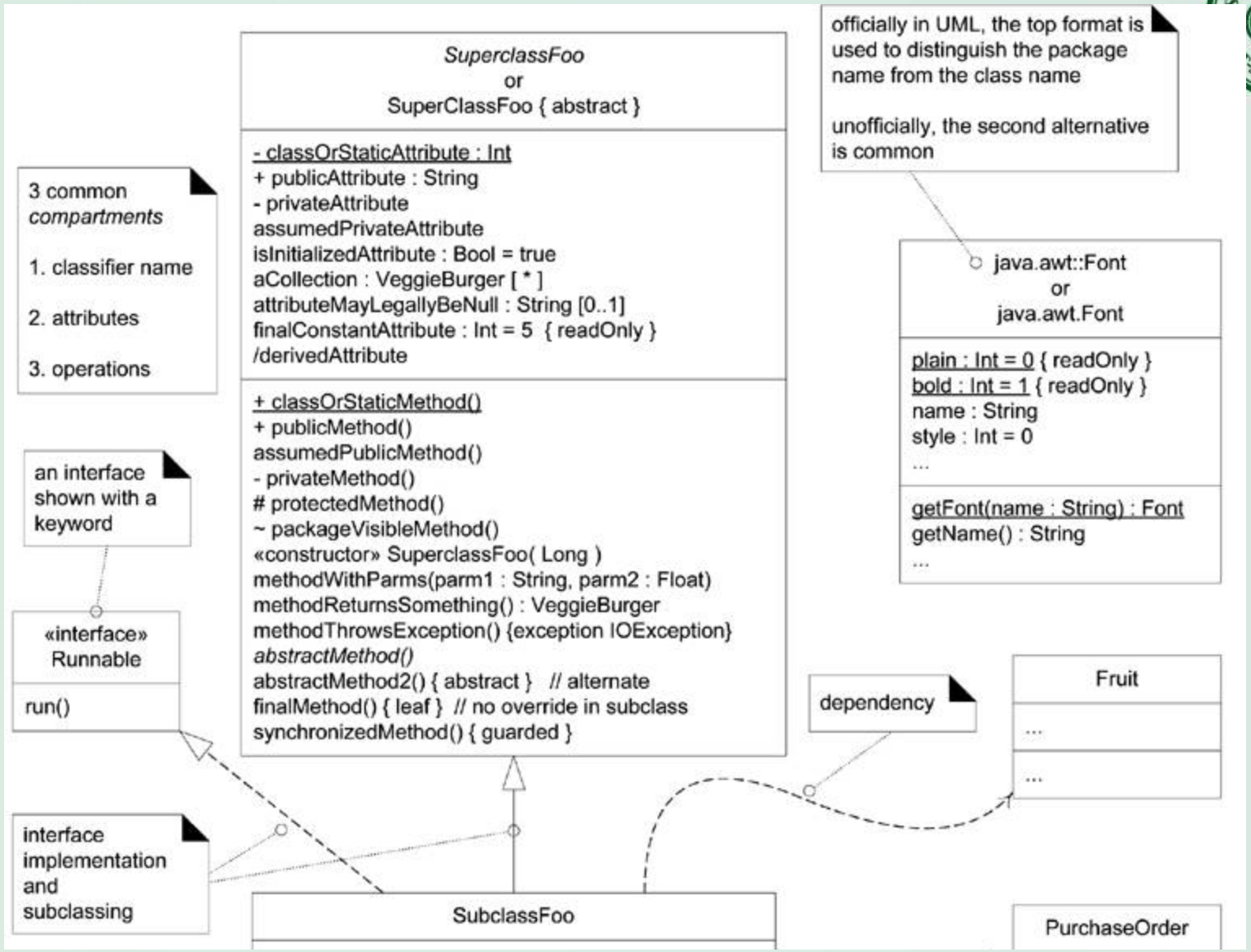
Chap 16
UML Class Diagrams

Introduction

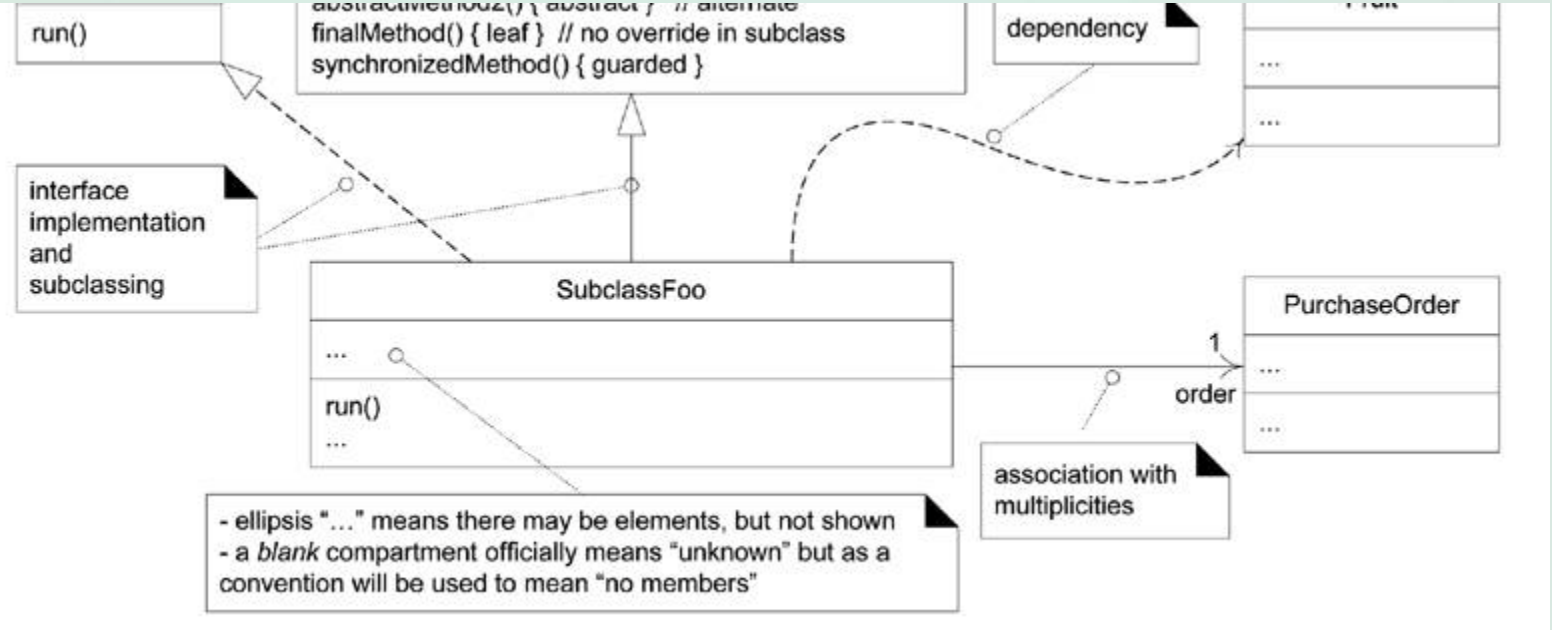
- Objective of this chapter
  - Provide a reference for frequently used UML class diagram notation

- The UML includes **class diagrams** to illustrate classes, interfaces, and their associations. They are used for **static object modeling**

Common UML class diagram notation 1

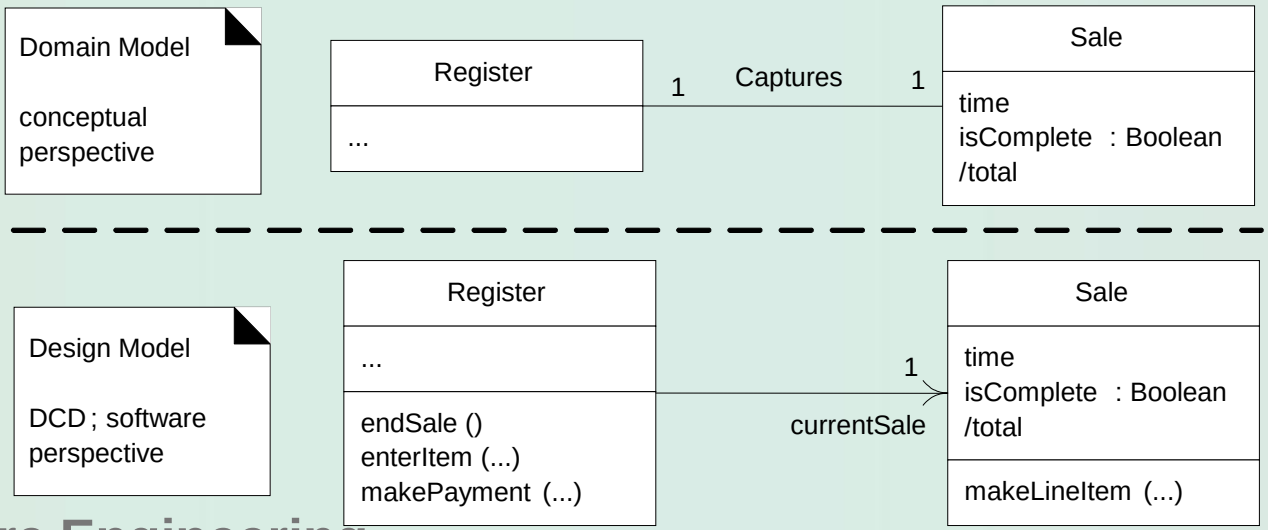


Common UML class diagram notation 2



Design Class Diagram (DCD)

- **The same UML diagram can be used in multiple perspectives**
  - In a conceptual perspective the class diagram can be used to visualize a domain model.
  - Class diagram is used in a software or design perspective, called *design class diagram (DCD)*



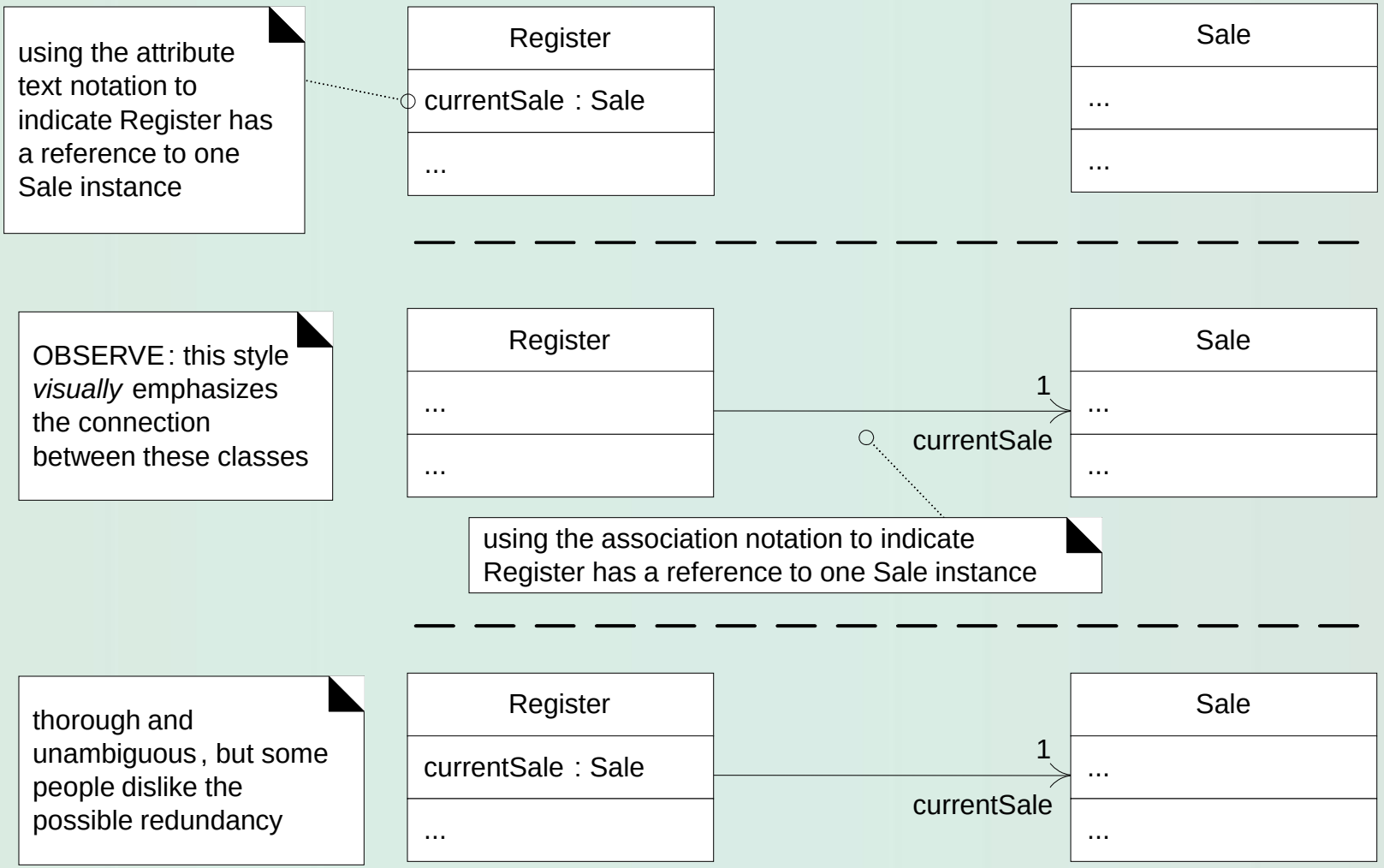
Classifier

- A UML classifier is "a model element that describes behavioral and structure features".
- Classifiers can also be specialized.
  - They are a generalization of many of the elements of the UML, including *classes*, *interfaces*, *use cases*, *and actors.*
  - In class diagrams, the two most common classifiers are regular **classes** and **interfaces**.

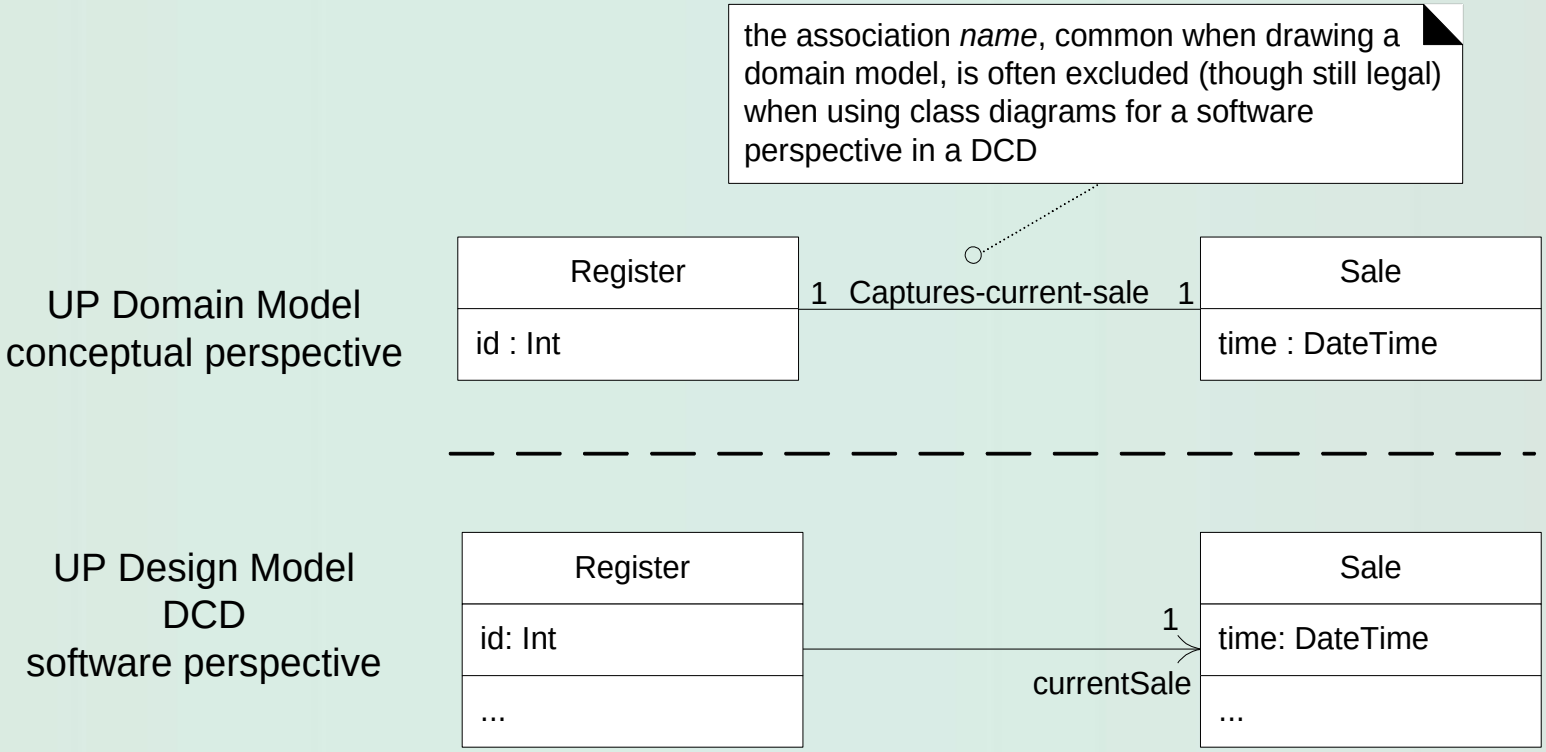
Ways to Show UML Attributes

- Ways to Show UML Attributes:
  - Attribute Text and Association Lines
- Attributes of a classifier are shown several ways:
  - attribute text notation, such as currentSale : Sale.
  - association line notation
  - both together
- The full format of the attribute text notation is:
  - *visibility name : type multiplicity = default {property-string}*
- ***Guideline:*** *Attributes are usually assumed private if no visibility is given*

Ways to Show UML Attributes 1



Ways to Show UML Attributes 2



Attribute text versus association line notation for a UML attribute

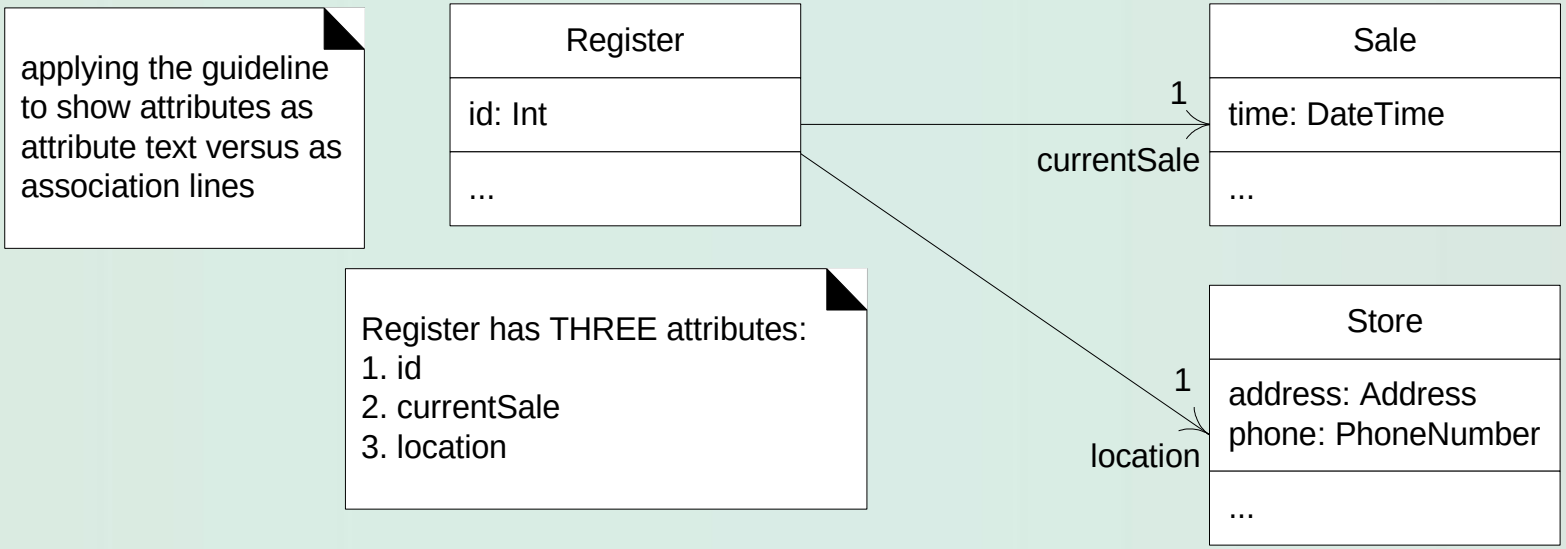
Ways to Show UML Attributes 3

- **Guideline**: When showing attributes-as-associations, follow the style in DCDs, which is suggested by the UML specification. (Fig 16.4 upper)
- **Guideline**: when using class diagrams for a domain model do show association names but avoid navigation arrows, as a domain model is not a software perspective.

Ways to Show UML Attributes 4

- **Guideline: When to Use Attribute Text versus Association Lines for Attributes**
  - Use the attribute text notation for **data type objects** and the association line notation for others.
  - Both are semantically equal, but showing an association line to another class box in the diagram (as in Figure 16.3) gives **visual emphasis -** it catches the eye, emphasizing the connection between the class of objects on the diagram.

Ways to Show UML Attributes 5



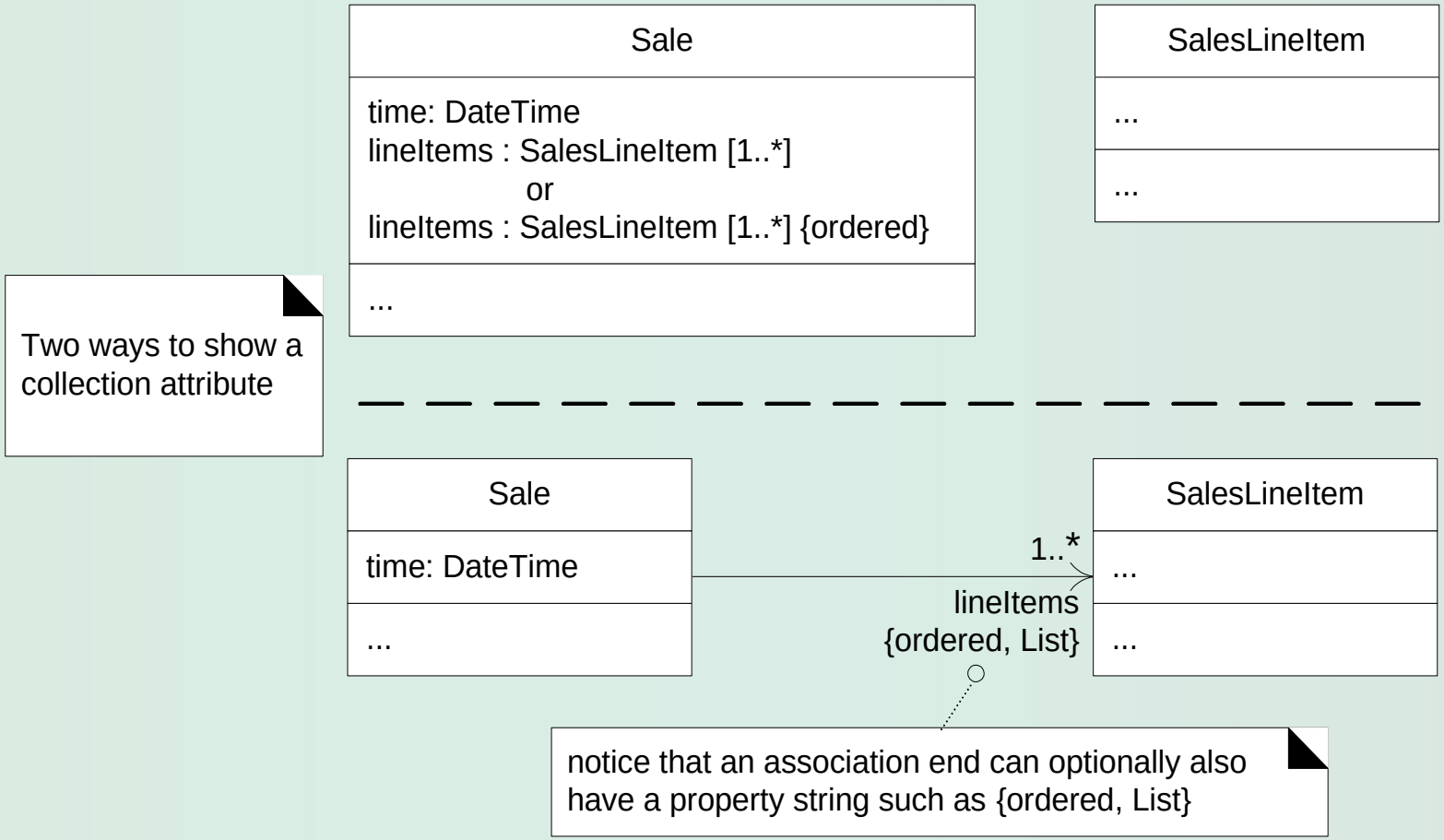

```
public class Register {
   private int id;
   private Sale currentSale;
   private Store location;
   // …
}
```

Ways to Show UML Attributes 6

- How to Show Collection Attributes with Attribute Text and Association Lines ?

```
public class Sale {
   private List<SalesLineItem> lineItems =
         new ArrayList<SalesLineItem>();
   // …
   }
}
```

Ways to Show UML Attributes 7



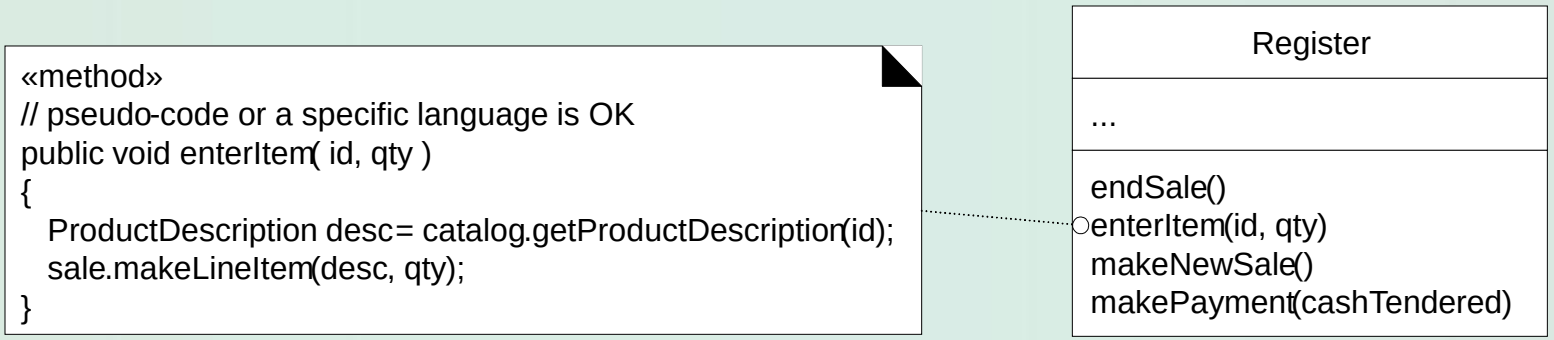
Operations and Methods 1

- Operations

  visibility name (parameter-list) : return-type {property-string}
- Guideline: Assume the version that includes a return type.
- Guideline: Operations are usually assumed public if no visibility is shown.
- Example
  - *+ getPlayer(name : String) : Player {exception IOException}*
  - *public Player getPlayer(String name) throws IOException*
- An operation is not a method.
  - A UML **operation** is a declaration, with a name, parameters, return type, exceptions list, and possibly a set of constraints of pre-and post-conditions.
  - But, it isn't an implementation - rather, methods are implementation

Operations and Methods 2

- How to Show Methods in Class Diagrams?
  - in interaction diagrams, by the details and sequence of messages
  - in class diagrams, with a UML note symbol stereotyped with «method»



Keywords 1

- A UML keyword is a textual adornment to categorize a model element.
  - For example, the keyword to categorize that a classifier box is an interface is «interface».
  - The «actor» keyword was used on p. 91 to replace the human stick-figure actor icon with a class box to model computer-system or robotic actors.
- Guideline: When sketching UML - when we want speed, ease, and creative flow - modelers often simplify keywords to something like '<interface>' or '<I>'.

Keywords 2

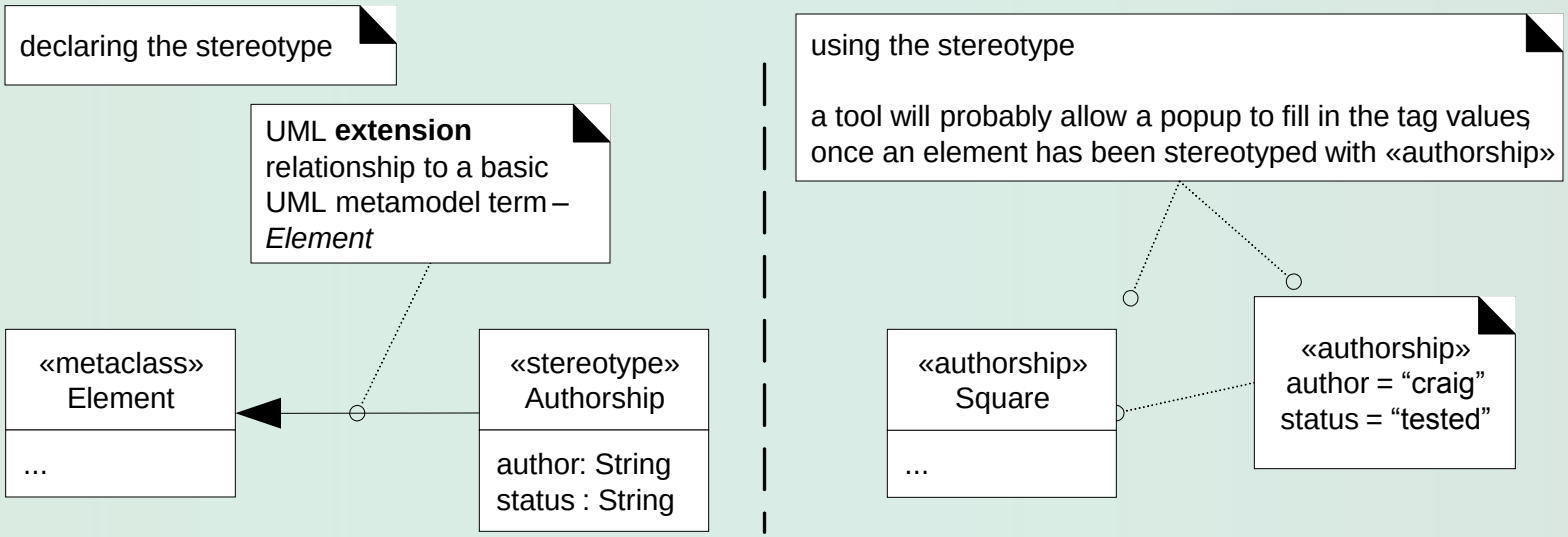
- Most keywords are shown in guillemet (« ») but some are shown in curly braces, such as {abstract}, which is a constraint containing the abstract keyword.
- In general, when a UML element says it can have a "property string“ - such as a UML operation and UML association end have - some of the property string terms will be keywords used in the curly brace format.

Keyword	Meaning	Example Usage
«actor»	classifier is an actor	in class diagram, above classifier name
«interface»	classifier is an interface	in class diagram, above classifier name
{abstract}	abstract element; can't be instantiated	in class diagrams, after classifier name or operation name
{ordered}	a set of objects have some imposed order	in class diagrams, at an association end

Stereotypes, Profiles, and Tags

- Stereotypes
  - are shown with guillemets symbols
  - represents a refinement of an existing modeling concept and is defined within a UML profile
  - The UML predefines many stereotypes, such as «destroy» (used on sequence diagrams), and also allows user-defined ones.
  - Thus, stereotypes provide an extension mechanism in the UML
- Profiles
  - a collection of related stereotypes, tags, and constraints to specialize the use of the UML for a specific domain or platform
  - For example, UML profile for project management or for data modeling.

Stereotypes



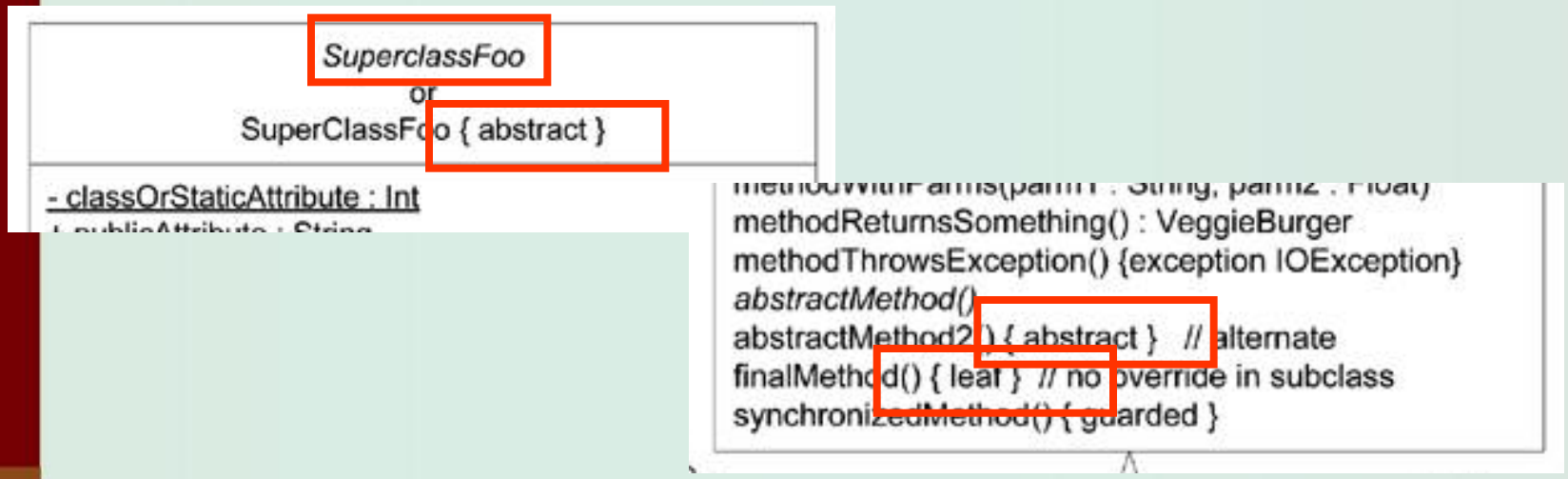
Stereotype declaration and use

Property and Property String

- In the UML, a property is *"a named value denoting a characteristic of an element. A property has semantic impact."*
  - Some properties are predefined in the UML, such as visibility - a property of an operation.
  - Others can be user-defined.
- Textual presentation approach
  - **UML property string** {name1=value1, name2=value2}
  - such as {abstract, visibility=public}.
  - Some properties are shown without a value, such as {abstract};

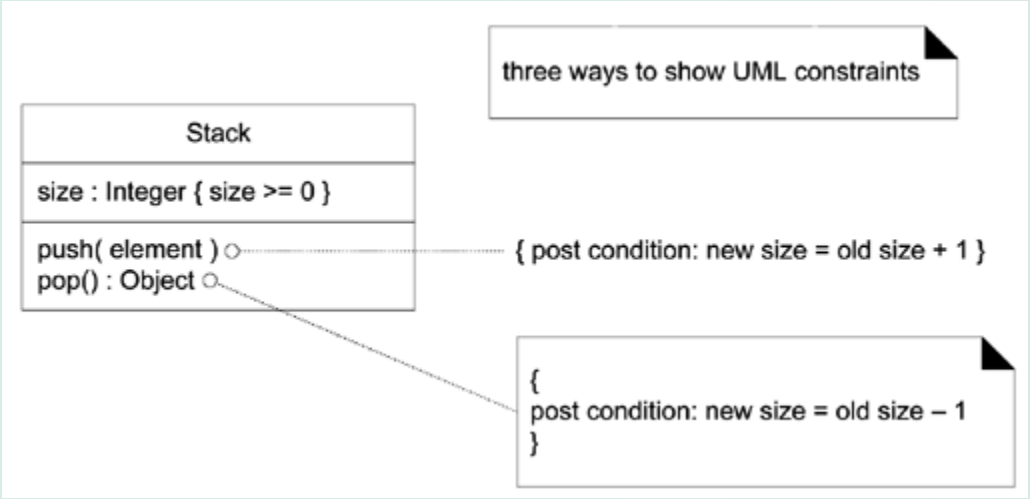
Abstract Classes and Abstract Operations

- Abstract classes and operations can be shown either with an {abstract} tag (useful when sketching UML) or by italicizing the name (easy to support in a UML tool).
- The opposite case, final classes and operations that can't be overridden in subclasses, are shown with the {leaf} tag.

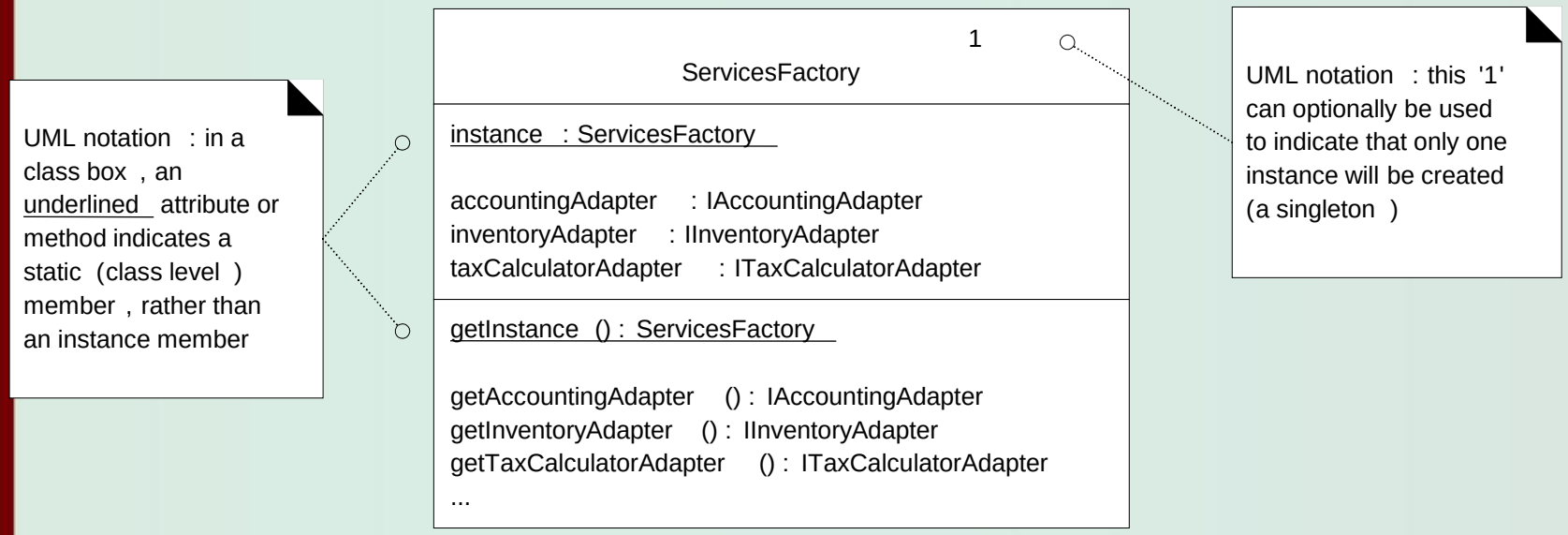


Constraint

- **Constraints**
  - Constraints may be used on most UML diagrams, but are especially common on class diagrams.
  - A UML **constraint** is a restriction or condition on a UML element.
  - It is visualized in text between braces;
    - for example: { size >= 0 }.
  - The text may be natural language or anything else, such as Object Constraint Language (OCL)

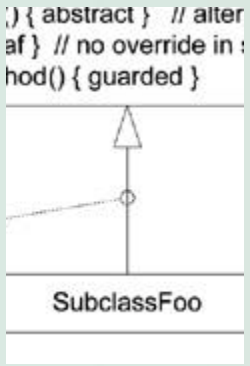


Singleton Classes



Generalization

- **Generalization** in the UML is shown with a solid line and fat triangular arrow from the subclass to superclass
  - A taxonomic relationship between a more general classifier and a more specific classifier.
  - Each instance of the specific classifier is also an indirect instance of the general classifier.
  - Thus, the specific classifier indirectly has features of the more general classifier.
- Generalization ➔ inheritance?
  - It depends. In a domain model conceptual-perspective class diagram, the answer is no.
  - In a DCD software-perspective class diagram, it implies OOPL inheritance from the superclass to subclass.



Dependency 1

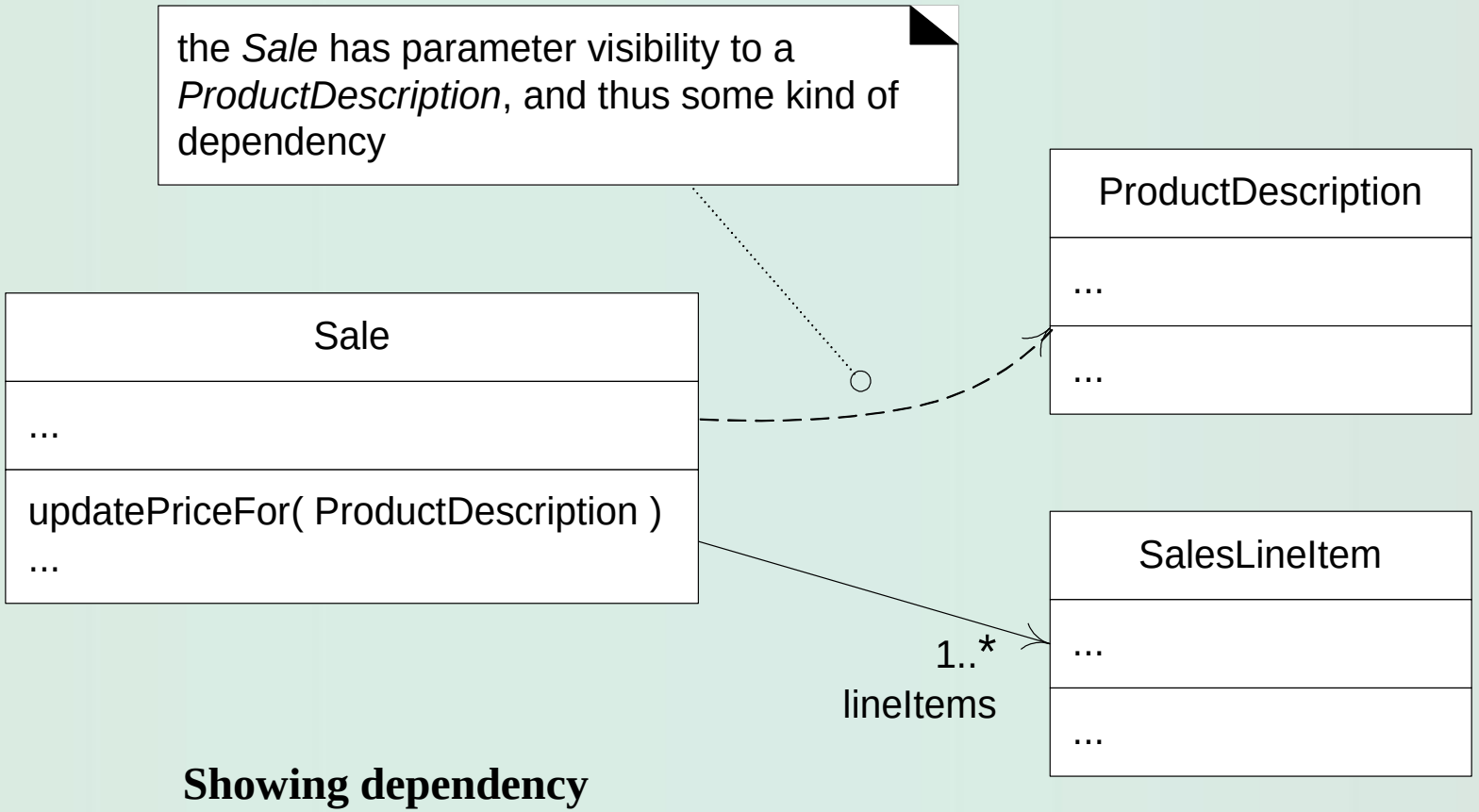
- Dependency lines may be used on any diagram, but are especially common on class and package diagrams.
- The UML includes a general dependency relationship that indicates that a client element (of any kind, including classes, packages, use cases, and so on) has knowledge of another supplier element and that a change in the supplier could affect the client.
- Dependency is illustrated with a dashed arrow line from the client to supplier.
- Dependency can be viewed as another version of **coupling**
- There are many kinds of dependency
  - having an attribute of the supplier type
  - sending a message to a supplier; the visibility to the supplier could be:
    - an attribute, a parameter variable, a local variable, a global variable, or class visibility (invoking static or class methods)
  - receiving a parameter of the supplier type
  - the supplier is a superclass or interface

Dependency 2

- All of these could be shown with a dependency line in the UML, but some of these types already have special lines that suggest the dependency
- When to show a dependency?
  - ***Guideline**: In class diagrams use the dependency line to depict global, parameter variable, local variable, and static-method (when a call is made to a static method of another class) dependency between objects.*

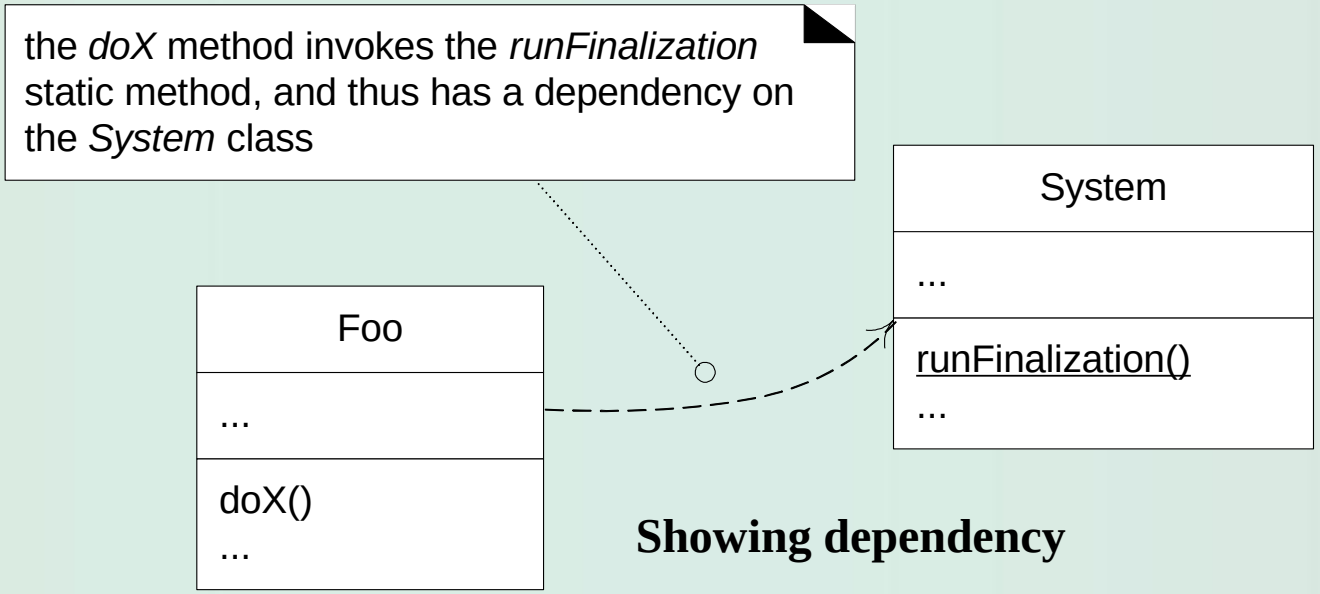
```
public class Sale
{
public void updatePriceFor( ProductDescription description )
{
   Money basePrice = description.getPrice();
   //…
}
// …
}
```

Dependency 2

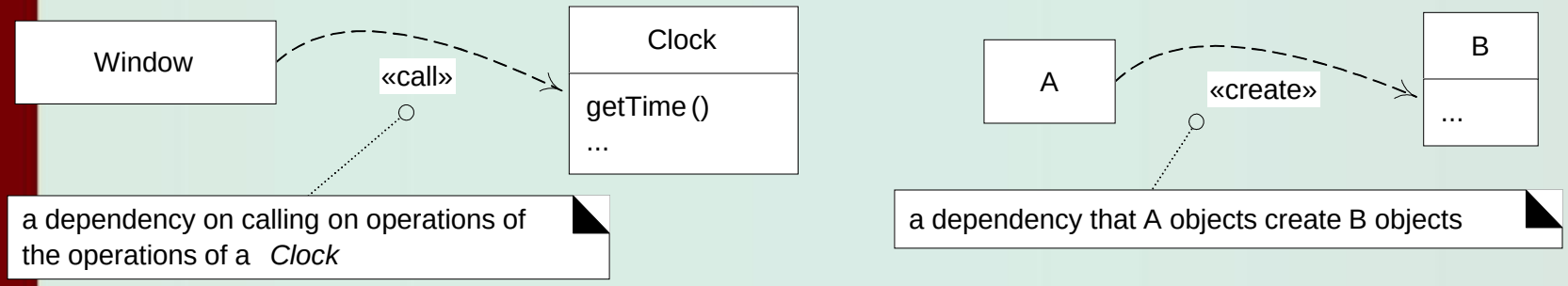


Showing dependency

Dependency 3

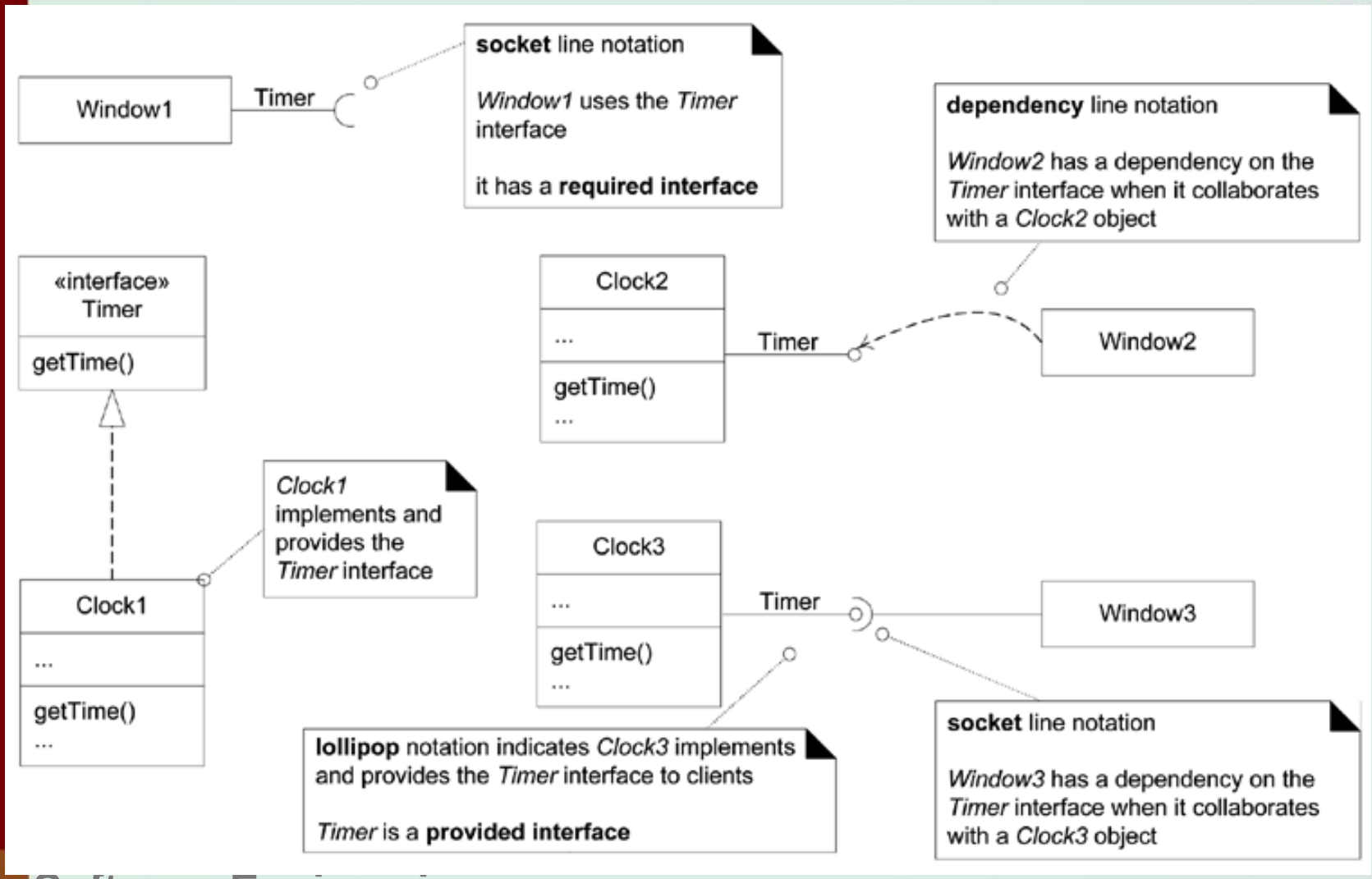


Showing dependency



Optional dependency labels in the UML

Interface


Window1
Timer
socket line notation
Window1 uses the Timer interface
it has a required interface
dependency line notation
Window2 has a dependency on the Timer interface when it collaborates with a Clock2 object
«interface»
Timer
getTime()
Clock2
...
getTime()
...
Timer
Window2
Clock1 implements and provides the Timer interface
Clock1
...
getTime()
...
Clock3
...
getTime()
...
Timer
Window3
lollipop notation indicates Clock3 implements and provides the Timer interface to clients
Timer is a provided interface
socket line notation
Window3 has a dependency on the Timer interface when it collaborates with a Clock3 object

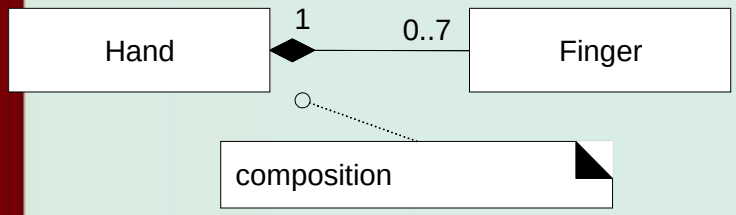
Composition Over Aggregation 1

- **Aggregation** is a vague kind of association in the UML that loosely suggests whole-part relationships
  - It has no meaningful distinct semantics in the UML versus a plain association,
  - but the term is defined in the UML. Why?
  - *In spite of the few semantics attached to aggregation, everybody thinks it is necessary (for different reasons). Think of it as a modeling placebo. [RJB04]*
- Guideline: Therefore, following the advice of UML creators, don't bother to use aggregation in the UML; rather, use composition when appropriate

Composition Over Aggregation 2

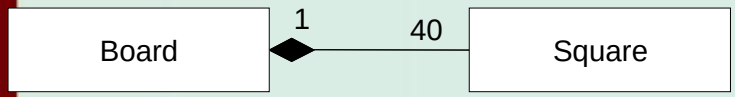
- **Composition**
  - also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models.
  - A composition relationship implies that
    - 1) an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time,
    - 2) the part must always belong to a composite (no free-floating Fingers), and
    - 3) the composite is responsible for the creation and deletion of its parts - either by itself creating/deleting the parts, or by collaborating with other objects.
- Guideline: The association name in composition is always implicitly some variation of "Has-part," therefore don't bother to explicitly name the association

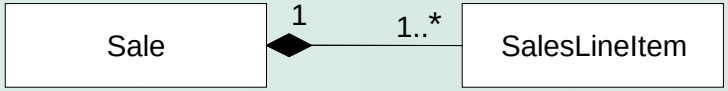
Composition Over Aggregation 3



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

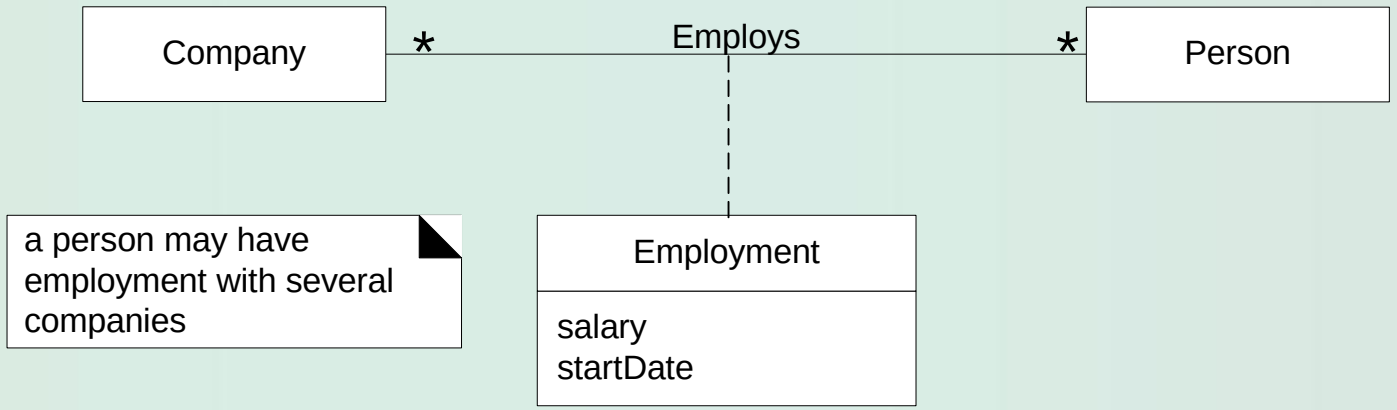
-the composite has sole responsibility for management of its parts, especially creation and deletion



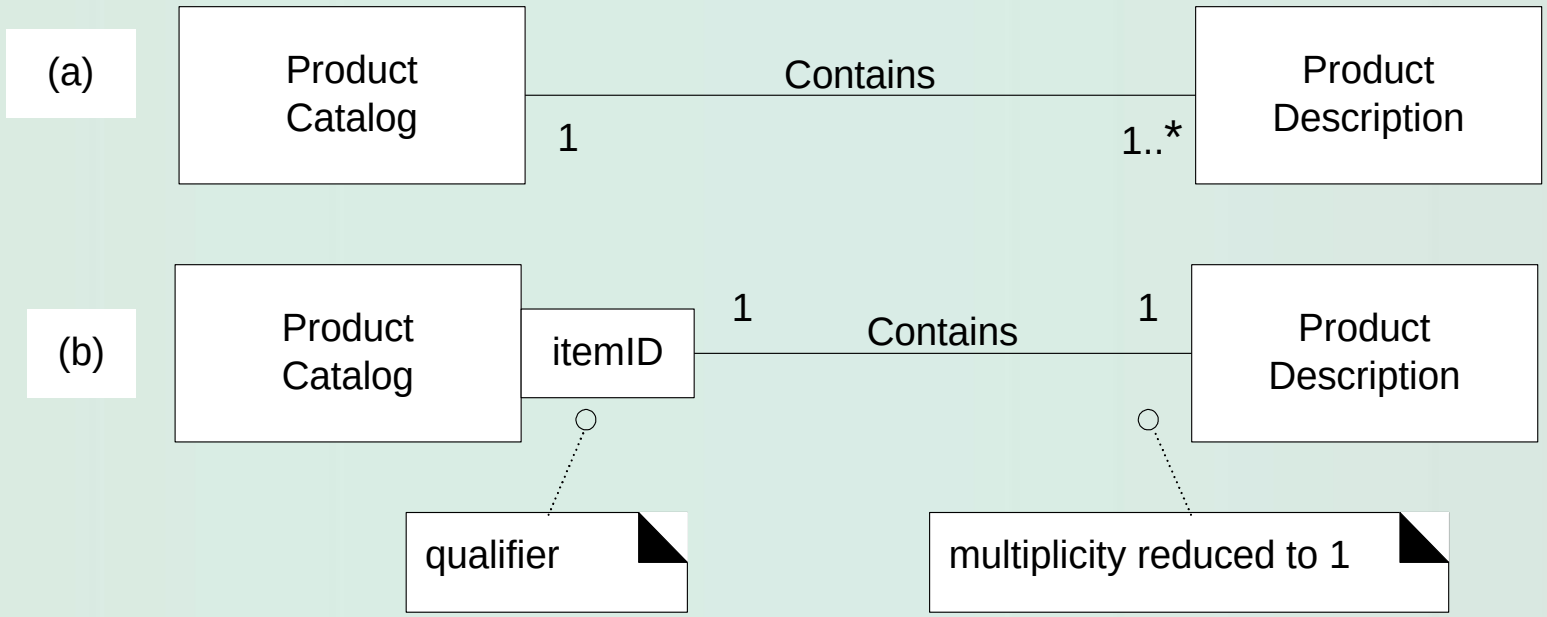


Association Class

- An **association class** allows you treat an association itself as a class, and model it with attributes, operations, and other features.
- For example, if a *Company* employs many *Persons*, modeled with an *Employs* association, you can model the association itself as the Employment class, with attributes such as *startDate*



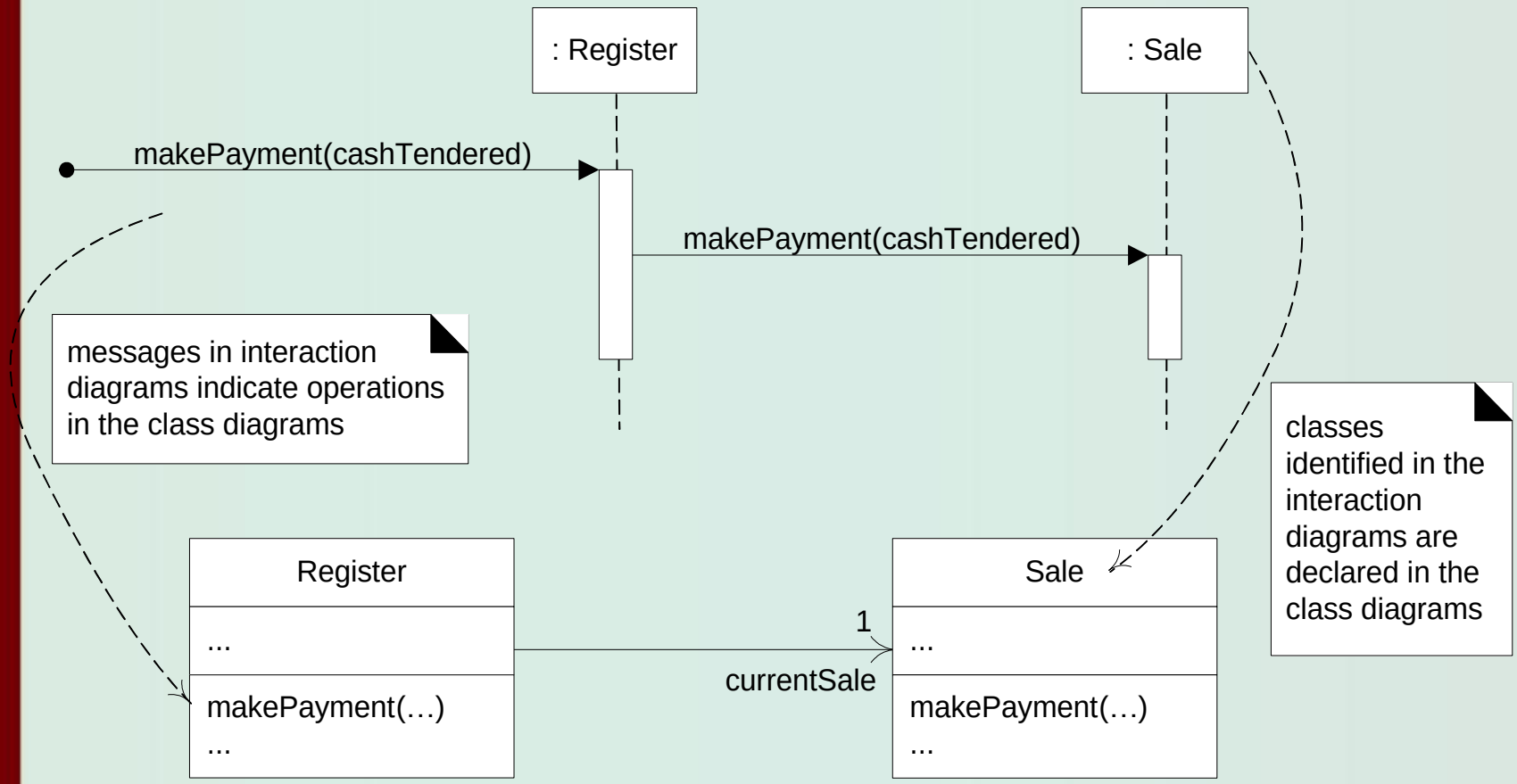
Qualified Association 1



Qualified Association 2

- A **qualified association** has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key.
- Informally, in a software perspective, it suggests looking things up by a key, such as objects in a *HashMap*. For example, if a ProductCatalog contains many ProductDescriptions, and each one can be selected by an itemID

Relationship Between Interaction and Class Diagrams

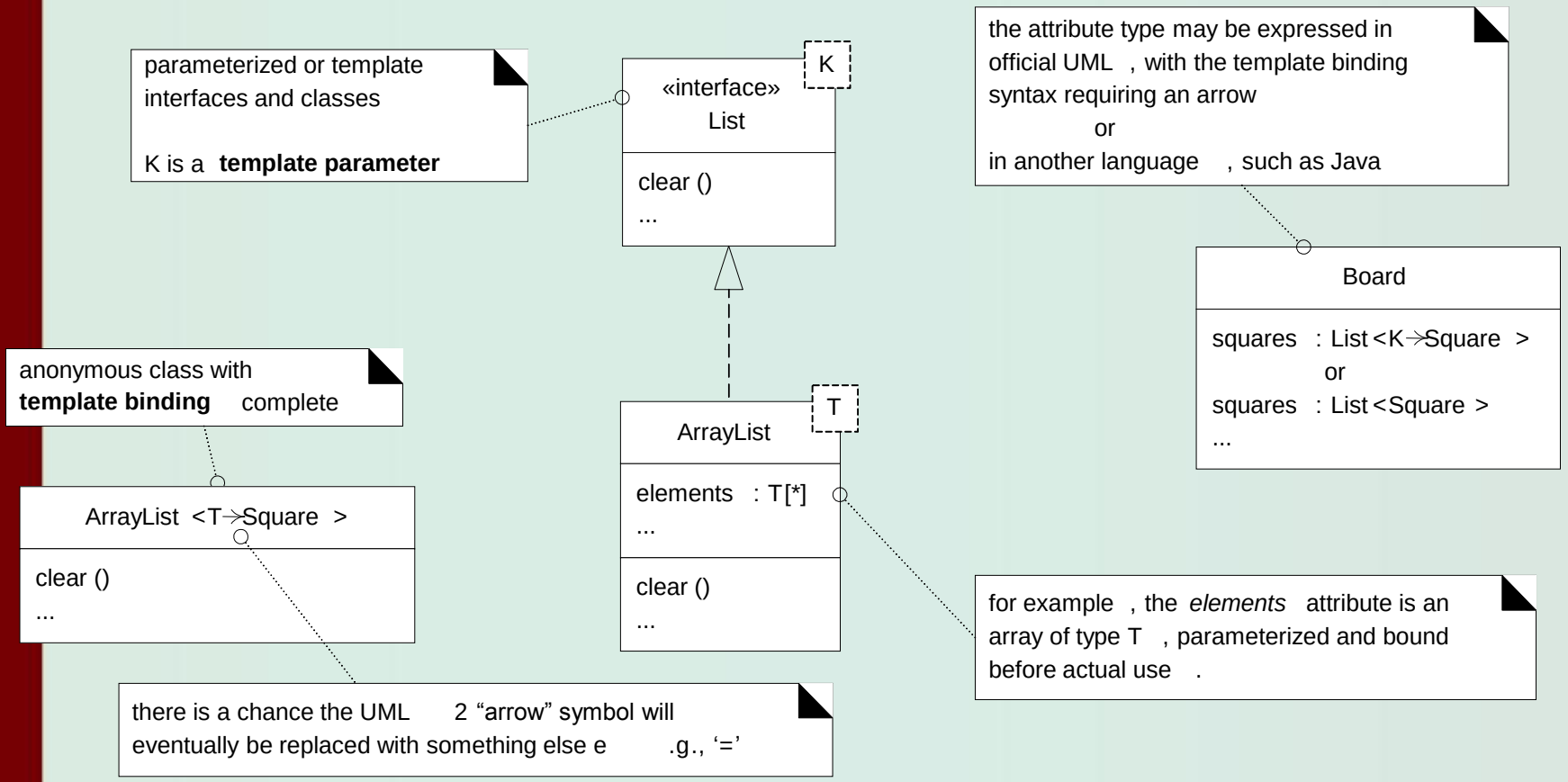


Template Classes and Interfaces 1

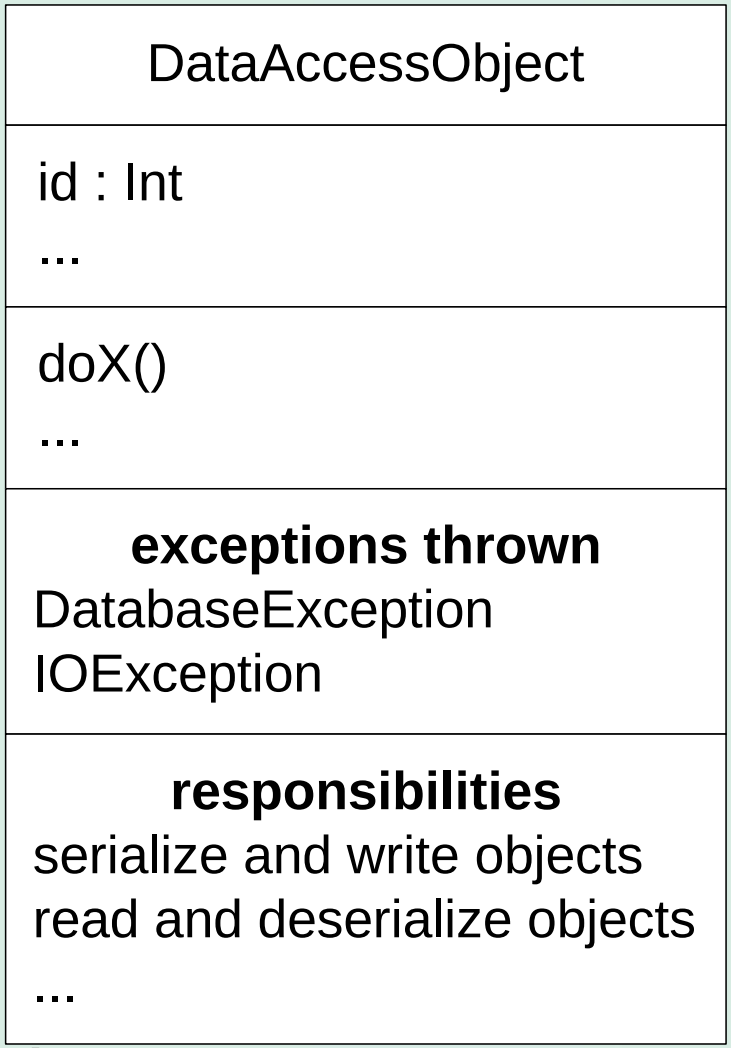
- Many languages (Java, C++, …) support *templatized* types, also known (with shades of variant meanings) as templates, parameterized types, and generics.
- They are most commonly used for the element type of collection classes, such as the elements of *lists* and *maps*. For example, in Java, suppose that a *Board* software object holds a *List* (an interface for a kind of collection) of many *Squares*. And, the concrete class that implements the List interface is an *ArrayList*:

```
public class Board {
    private List<Square> squares = new ArrayList<Square>();
    // ...
}
```

Template Classes and Interfaces 2



User-Defined Compartments



Active Class

- An **active object** runs on and controls its own thread of execution.
- The class of an active object is an active class