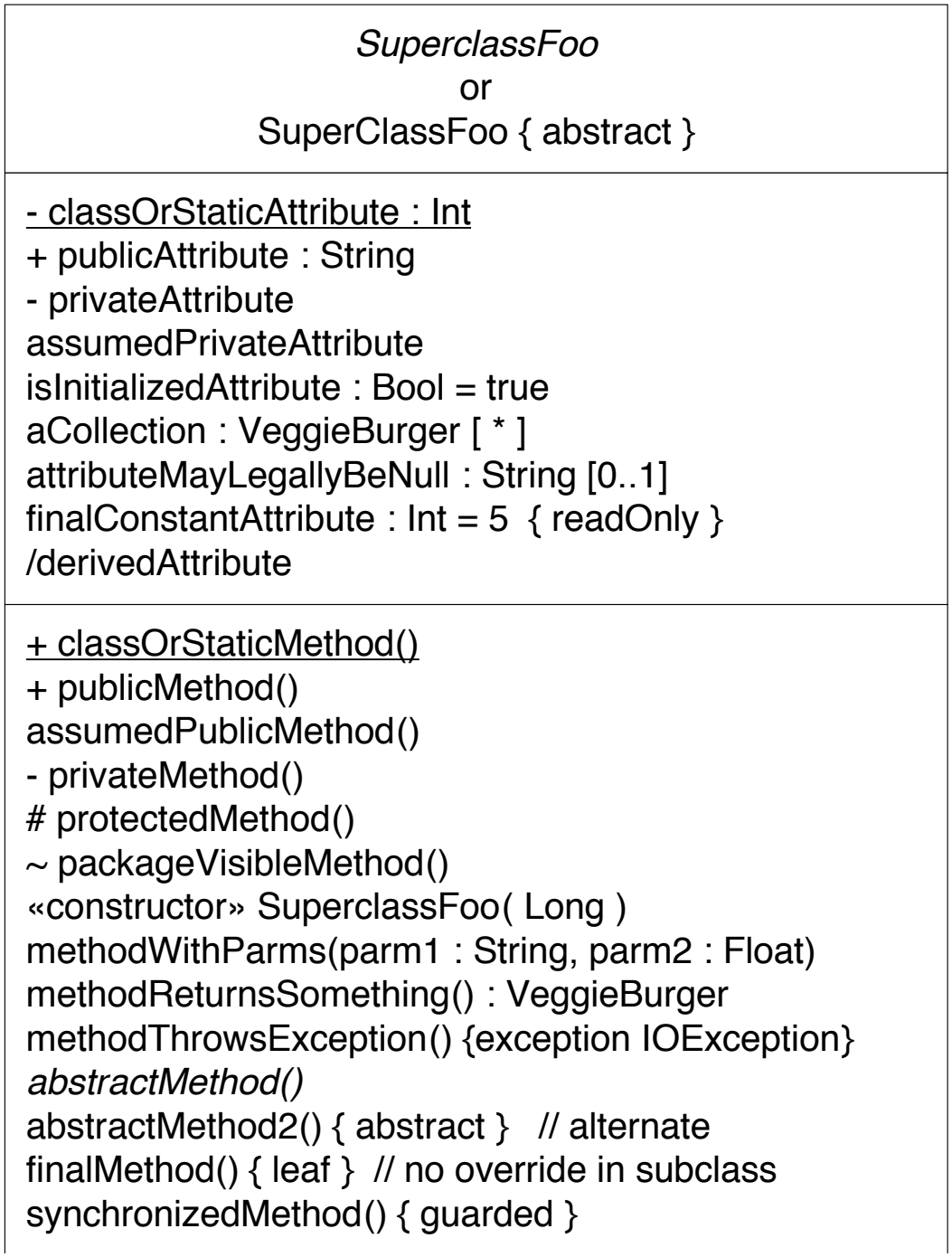
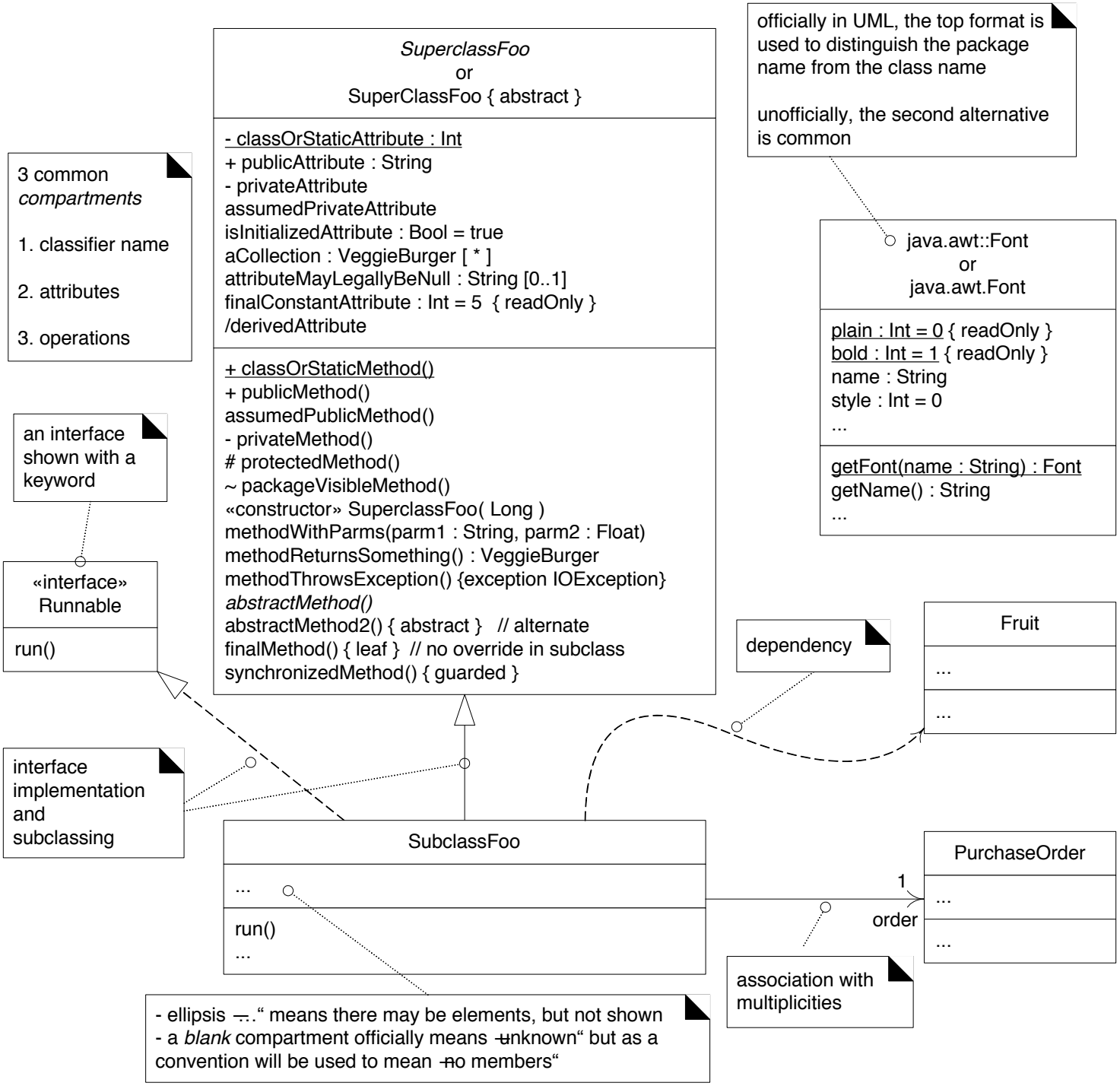
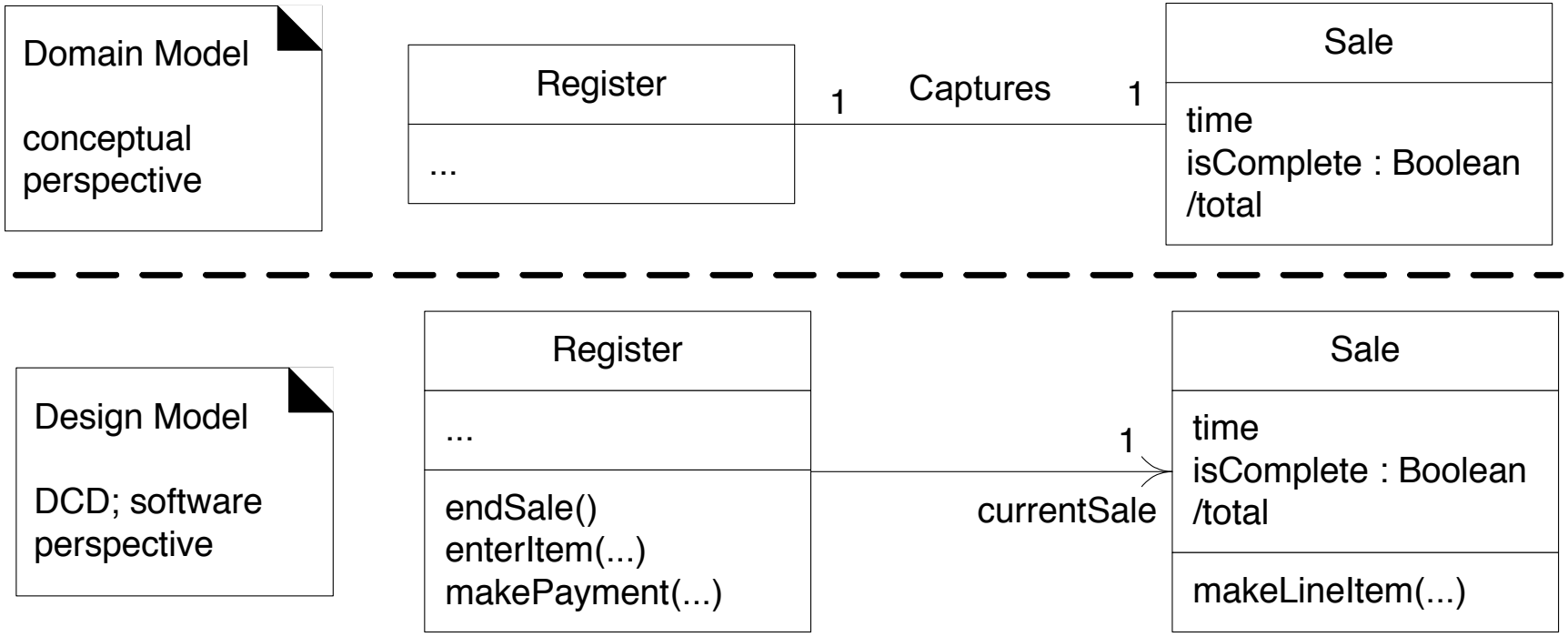
Chapter 16

UML Class Diagrams

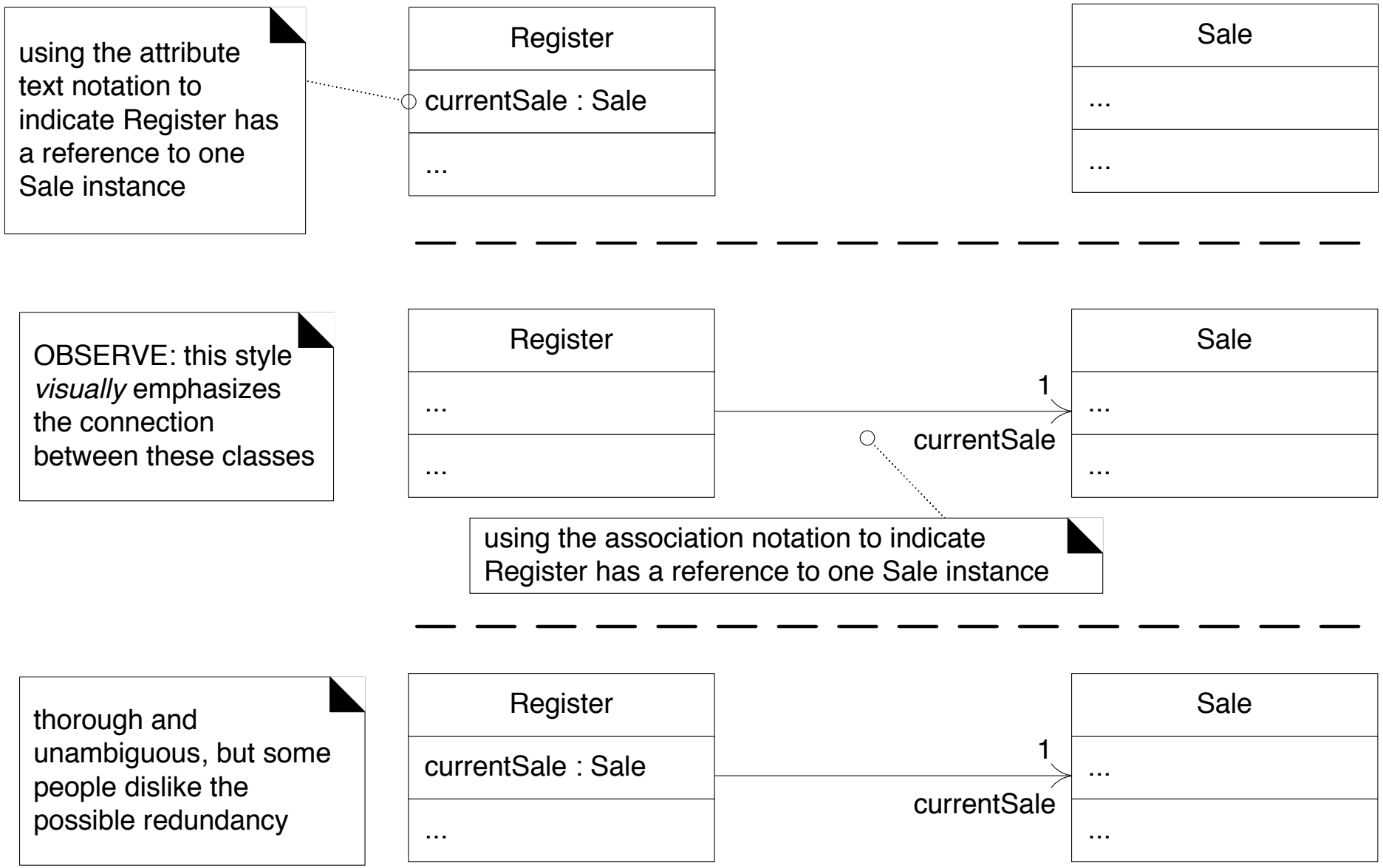

SuperclassFoo
or
SuperClassFoo { abstract }
- classOrStaticAttribute : Int
+ publicAttribute : String
- privateAttribute
assumedPrivateAttribute
isInitializedAttribute : Bool = true
aCollection : VeggieBurger [*]
attributeMayLegallyBeNull : String [0..1]
finalConstantAttribute : Int = 5 { readOnly }
/derivedAttribute
+ classOrStaticMethod()
+ publicMethod()
assumedPublicMethod()
- privateMethod()
protectedMethod()
~ packageVisibleMethod()
«constructor» SuperclassFoo(Long)
methodWithParms(parm1 : String, parm2 : Float)
methodReturnsSomething() : VeggieBurger
methodThrowsException() {exception IOException}
abstractMethod()
abstractMethod2() { abstract } // alternate
finalMethod() { leaf } // no override in subclass
synchronizedMethod() { guarded }


officially in UML, the top format is used to distinguish the package name from the class name
unofficially, the second alternative is common
3 common compartments
1. classifier name
2. attributes
3. operations
SuperclassFoo
or
SuperClassFoo { abstract }
- classOrStaticAttribute : Int
+ publicAttribute : String
- privateAttribute
assumedPrivateAttribute
isInitializedAttribute : Bool = true
aCollection : VeggieBurger [*]
attributeMayLegallyBeNull : String [0..1]
finalConstantAttribute : Int = 5 { readOnly }
/derivedAttribute
+ classOrStaticMethod()
+ publicMethod()
assumedPublicMethod()
- privateMethod()
protectedMethod()
~ packageVisibleMethod()
«constructor» SuperclassFoo(Long)
methodWithParms(parm1 : String, parm2 : Float)
methodReturnsSomething() : VeggieBurger
methodThrowsException() {exception IOException}
abstractMethod()
abstractMethod2() { abstract } // alternate
finalMethod() { leaf } // no override in subclass
synchronizedMethod() { guarded }
java.awt::Font
or
java.awt.Font
plain : Int = 0 { readOnly }
bold : Int = 1 { readOnly }
name : String
style : Int = 0
...
getFont(name : String) : Font
getName() : String
...
an interface shown with a keyword
«interface»
Runnable
run()
dependency
Fruit
...
...
interface implementation and subclassing
SubclassFoo
...
run()
...
PurchaseOrder
...
...
1
order
association with multiplicities
- ellipsis „...“ means there may be elements, but not shown
- a blank compartment officially means „unknown“ but as a convention will be used to mean „no members“

Domain Model vs. Design Class Diagram (DCD)

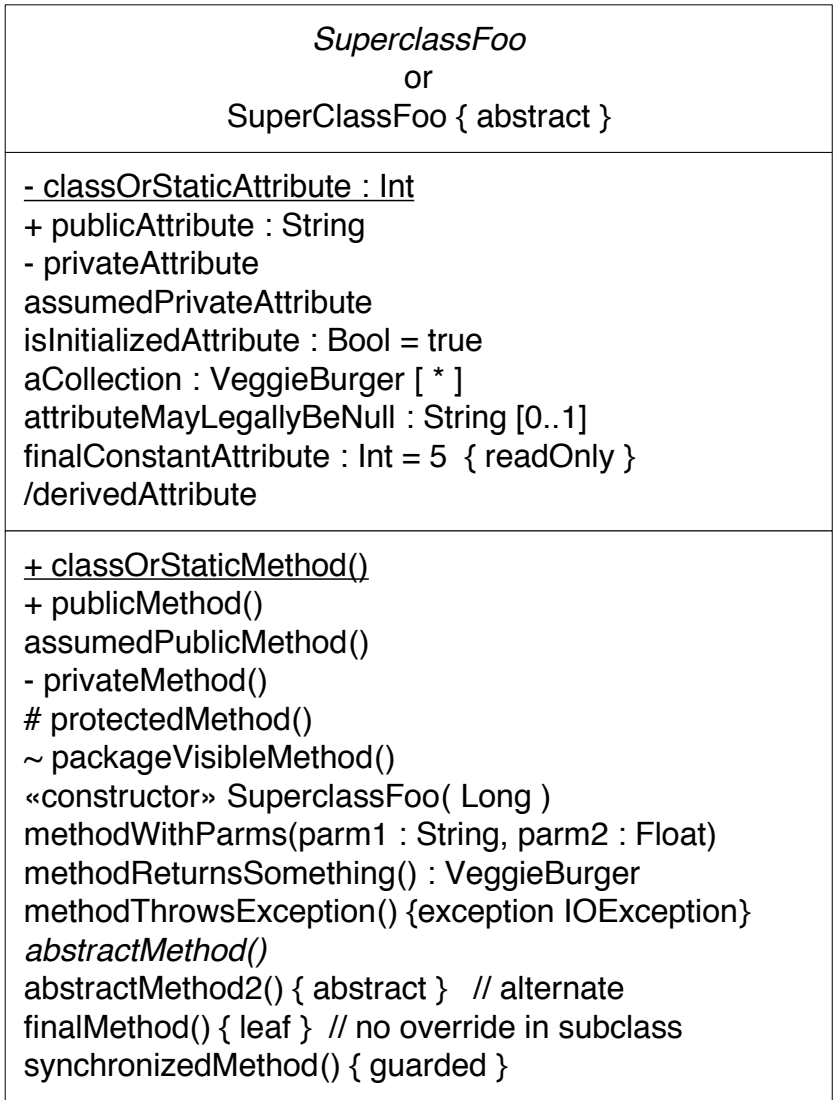


Attribute text vs. association lines

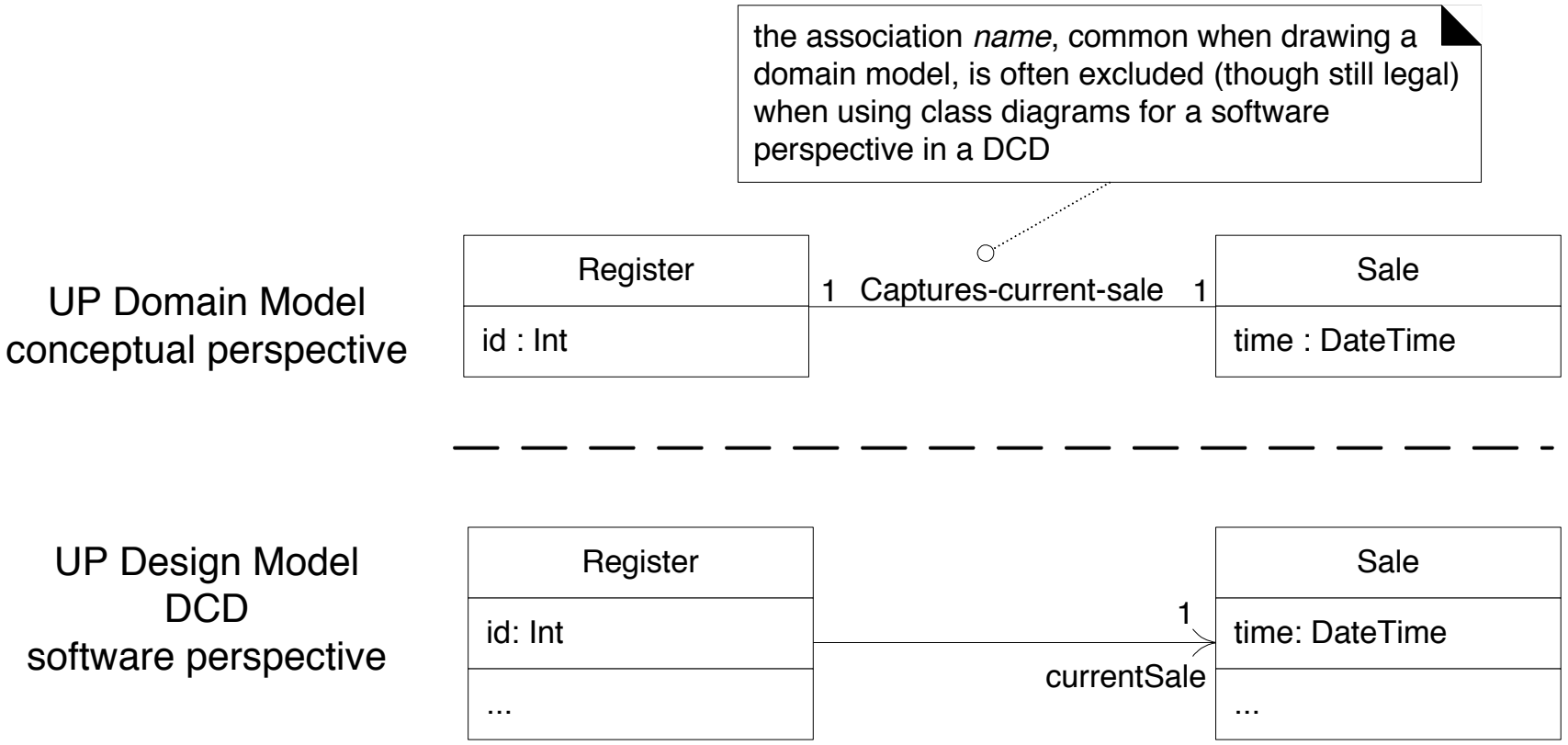


Full format attribute text notation

visibility name : type multiplicity = default {property string}

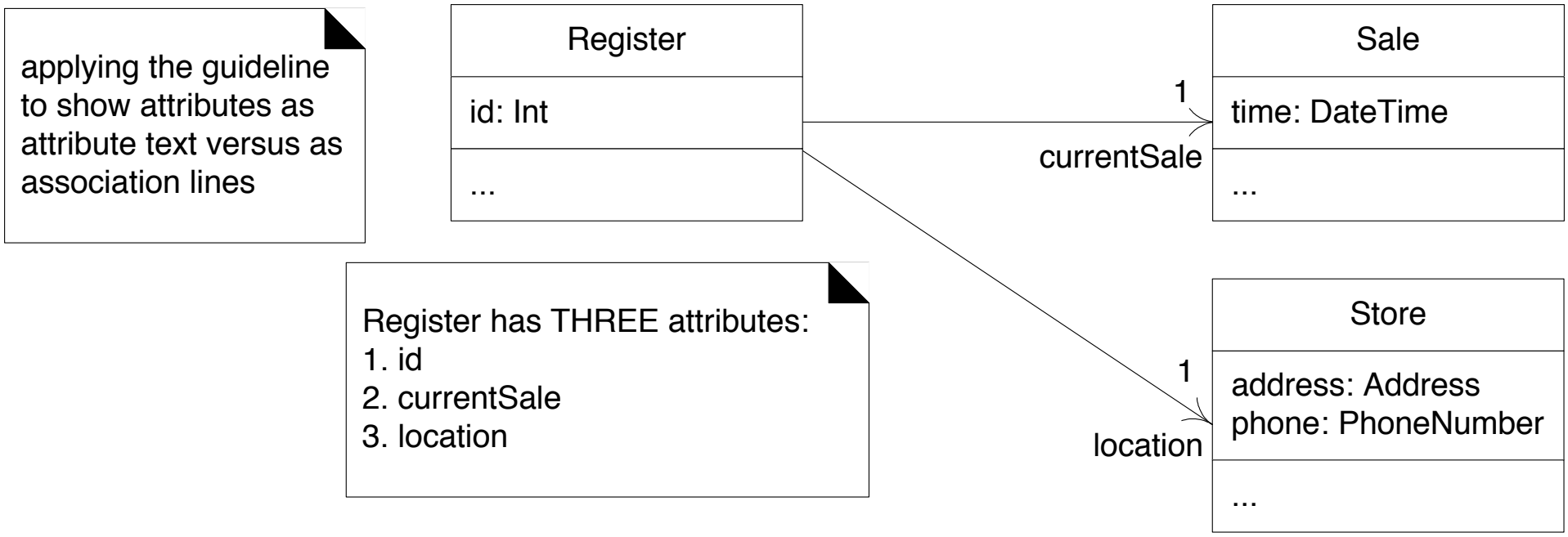


UML Class Diagram Conventions: Domain Models vs. DCDs



- Arrows:
  - Used in DCDs. Register has currentSale attribute
  - Not used in domain models.
- DCDs have multiplicities only at target end
- No role name for DCDs
- No association name for DCDs

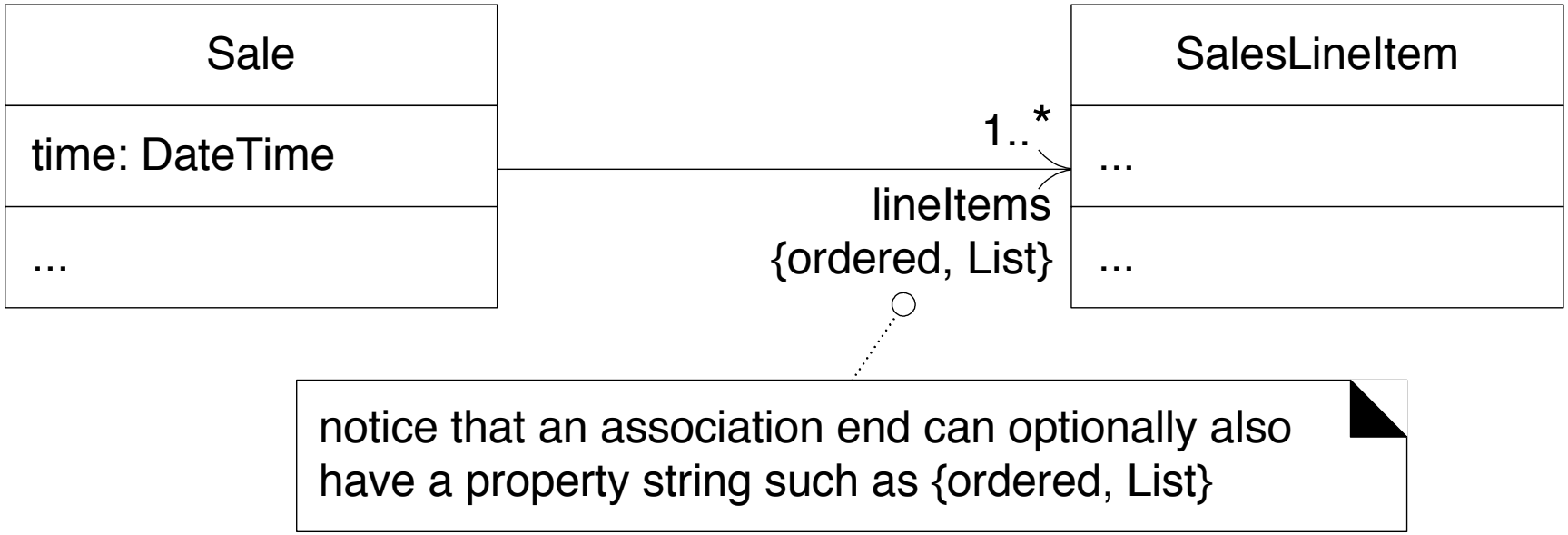
Attribute text vs. association lines



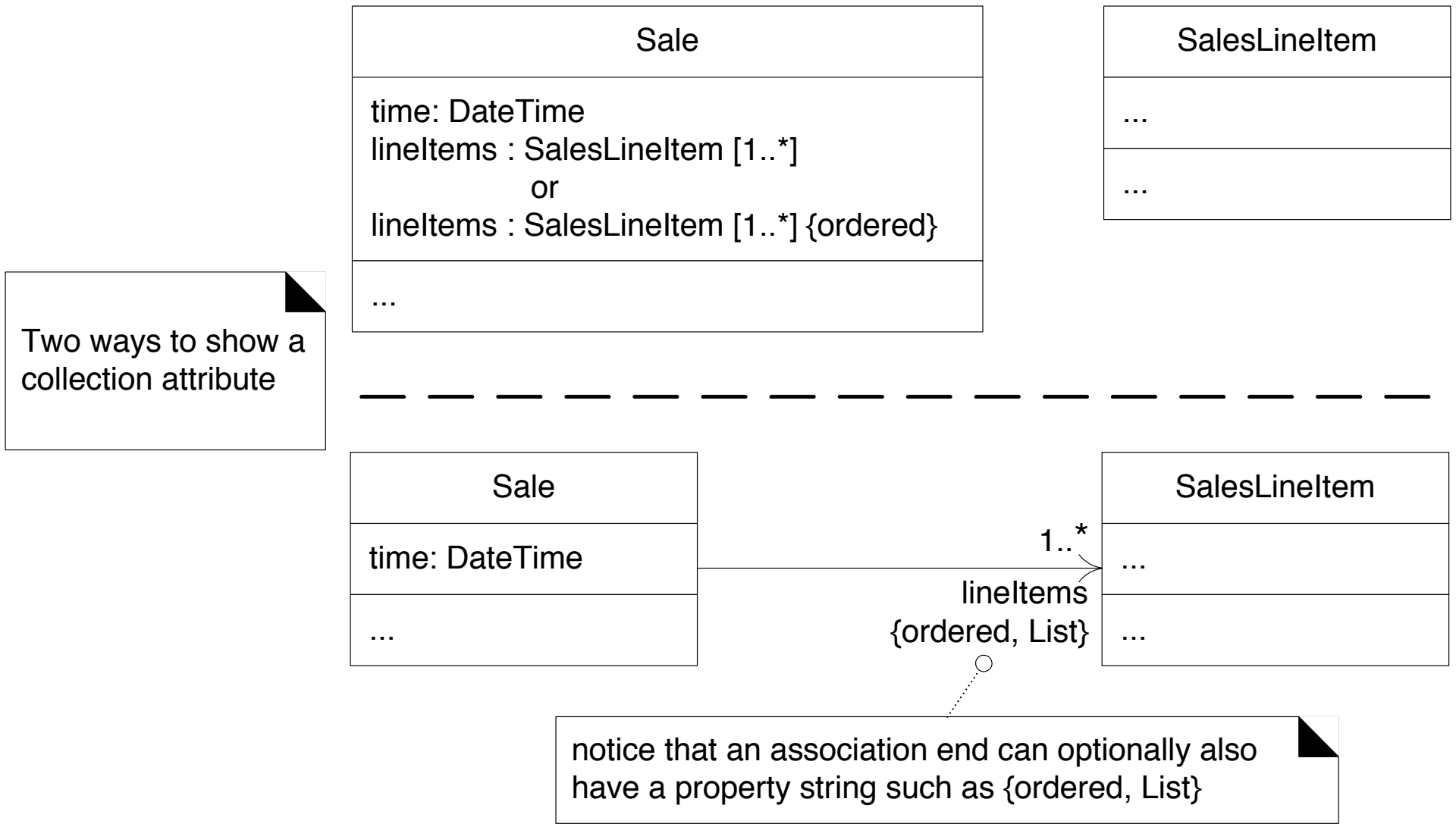
- Guideline: Attributes should be (simple, primitive) data types
  - Boolean, Date, Number (Integer, Real), String (Text), Time, Phone Number, ID Number, Postal Code, …
- Non-data type relationships (i.e., class relationships) should be expressed using associations, not attributes.
- Notational difference only.
  - As far as coding is concerned, they mean the same thing.

Notation at Association Ends

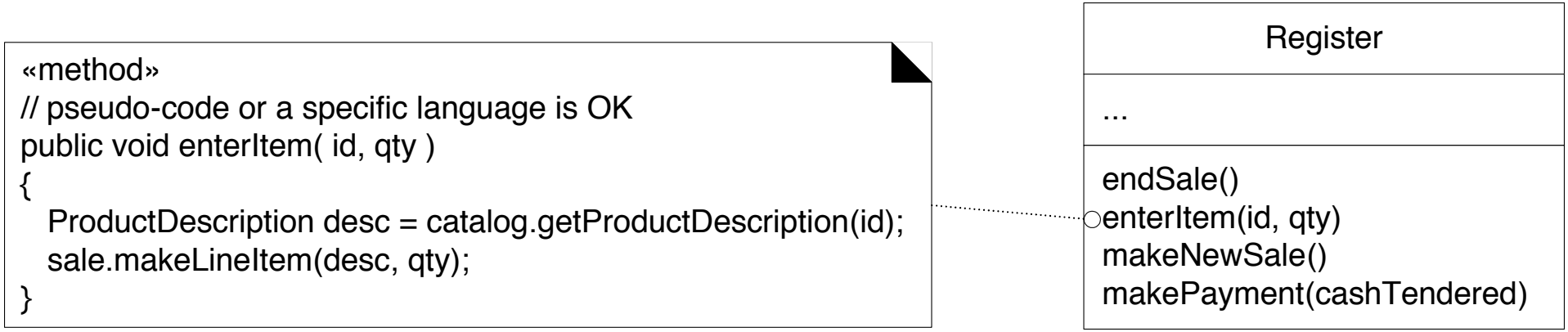
- Rolename: Association end name
- Multiplicity
- Property string



Collection Attributes



Note Symbols: Notes, Comments, Constraints and Method Bodies



Operations (Methods)

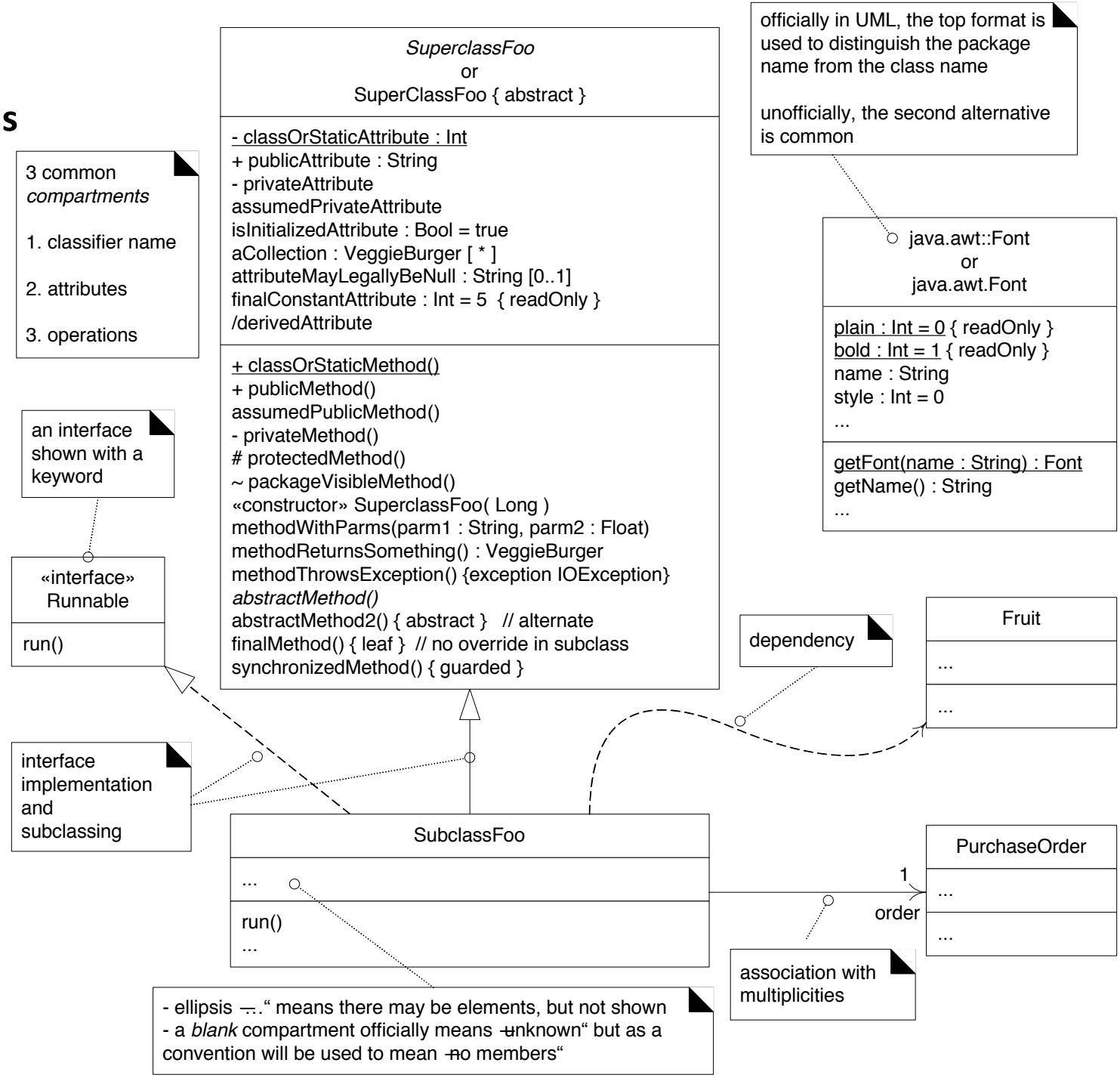
- Full official format
  visibility name (parameter-list) : return-type {property-string}
- Can use programming language notation if more convenient and understandable
- Operations assumed public unless indicated otherwise
- Property string: Arbitrary info
  - Example: Exceptions that may be raised
- Operation vs. method
  - Operation: A declaration, the specification of a method
  - Method: Implementation of an operation
- Access operations (set & get methods) usually excluded from DCD diagrams

UML Keywords

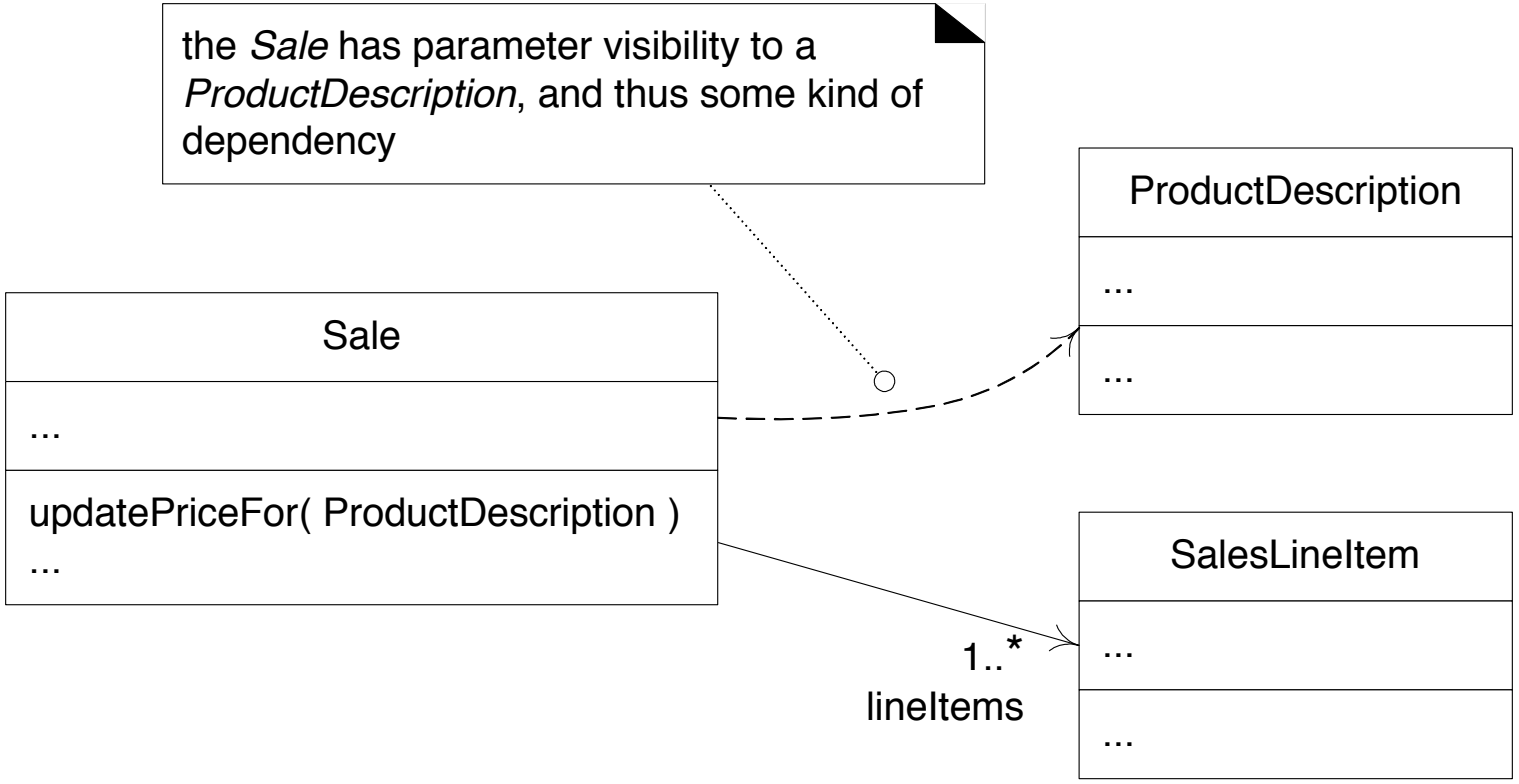
- <<actor>>
- <<interface>>
- {abstract}
- {ordered}

Generalization, Abstract Classes, Abstract Operations

- final class shown by {leaf}



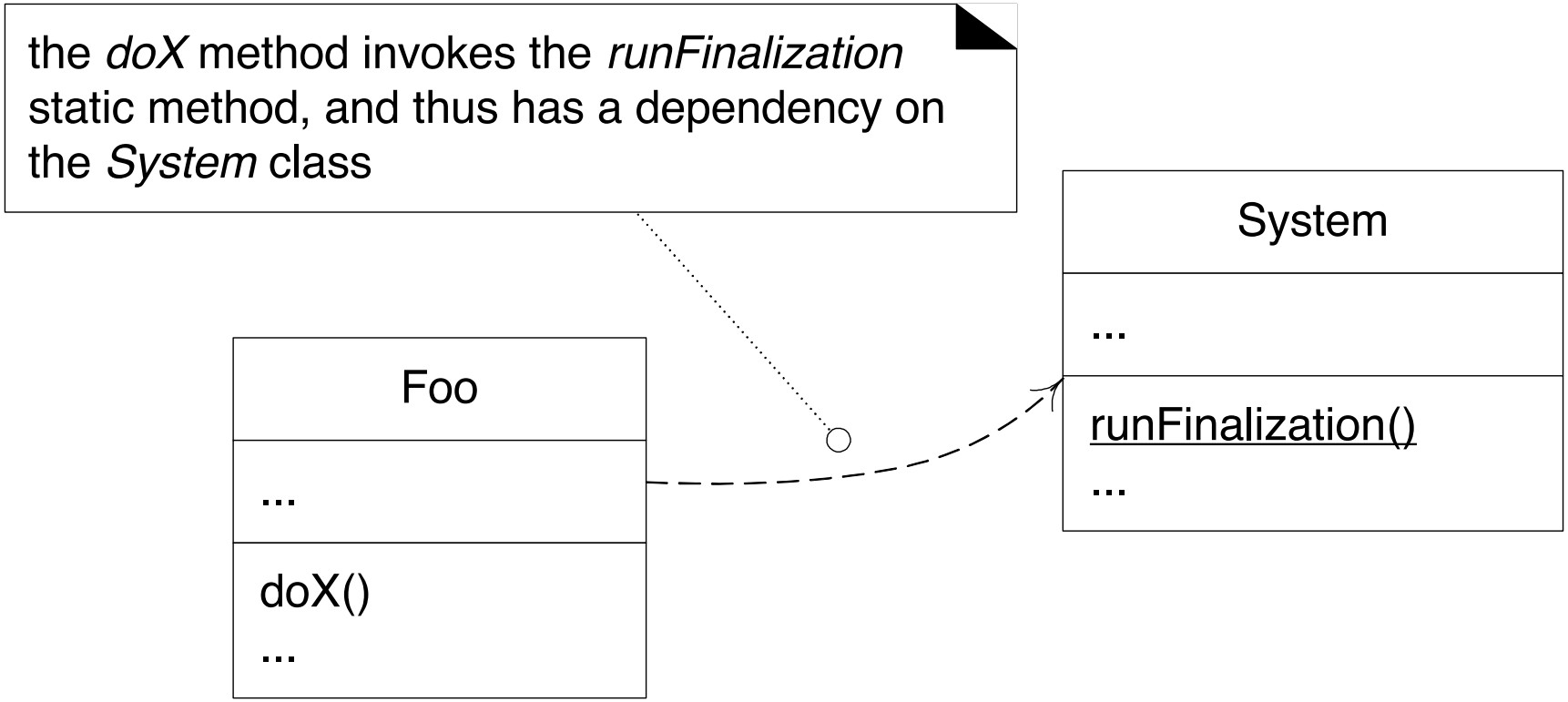
Dependency



- Reserve its use for purposes other than associations/attributes
  - Examples: Global, parameter variable, local variable and static method dependencies

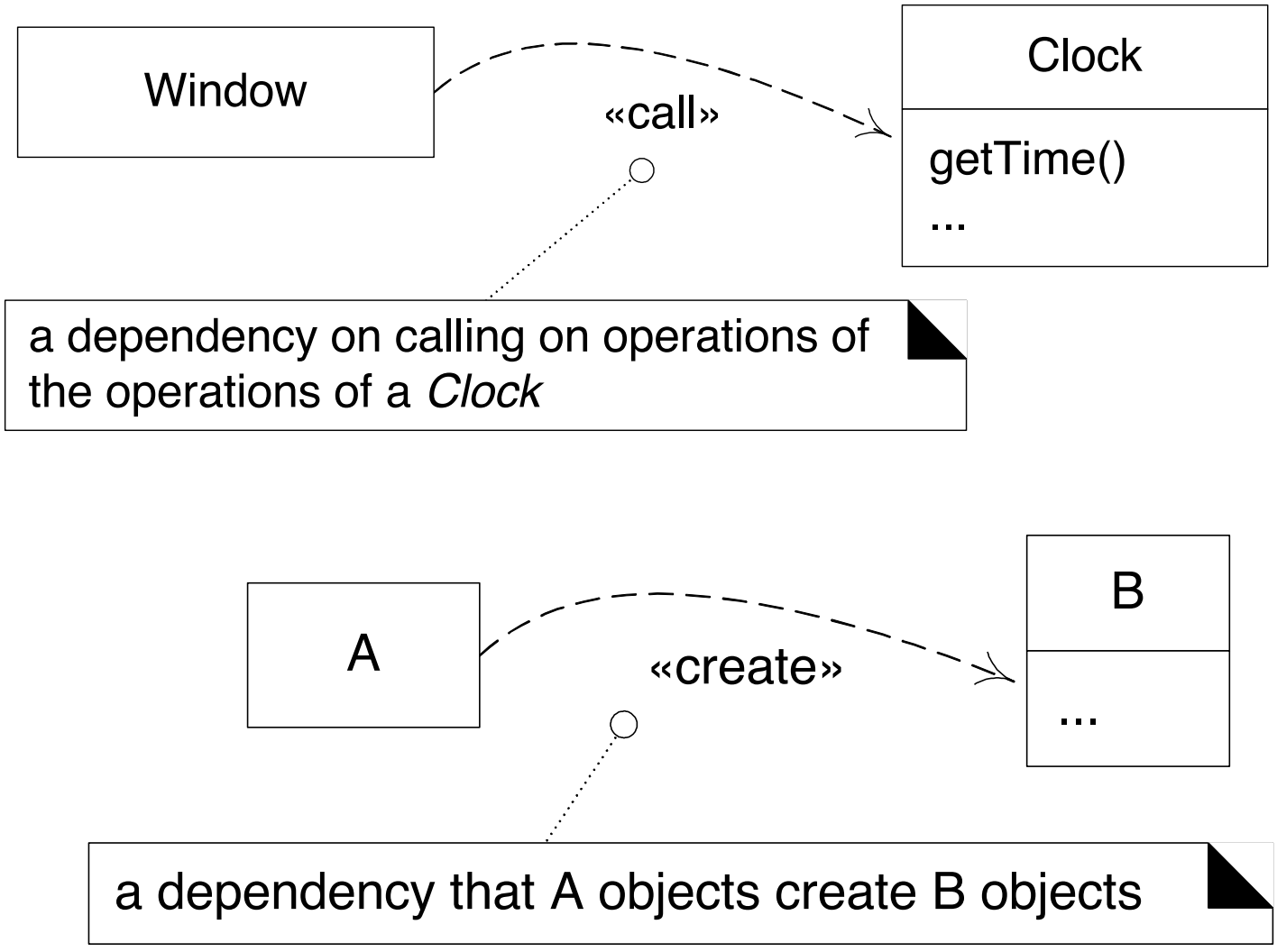
```
public class Sale {
   public void updatePriceFor(ProductDescription description) {
       Money basePrice = description.getPrice();
   }
}
```

Dependency Example

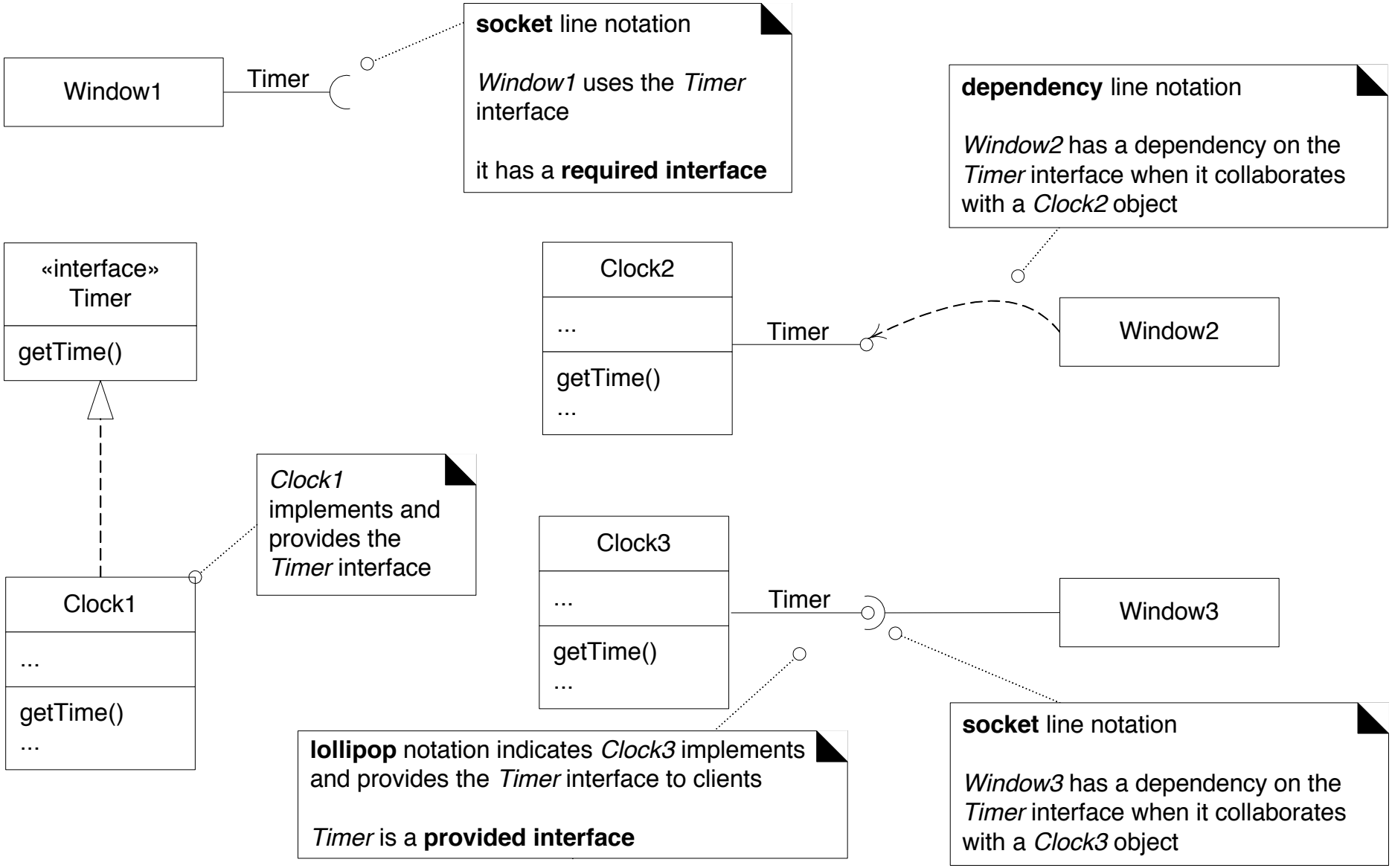



```
public class Foo {
    public void doX() {
        System.runFinalization();
        // ...
    }
}
```

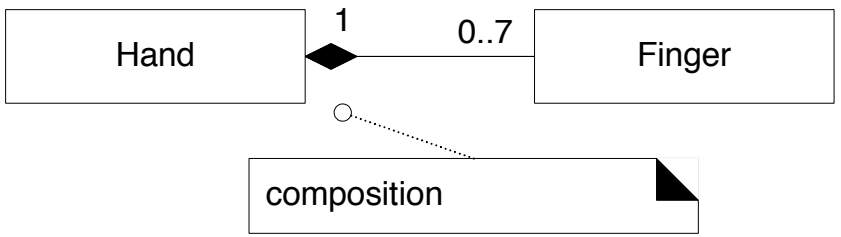
Dependency Labels



Interface Notations

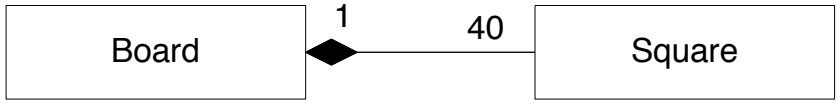


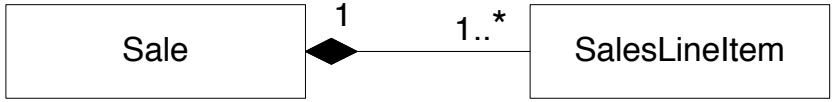
Composition



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

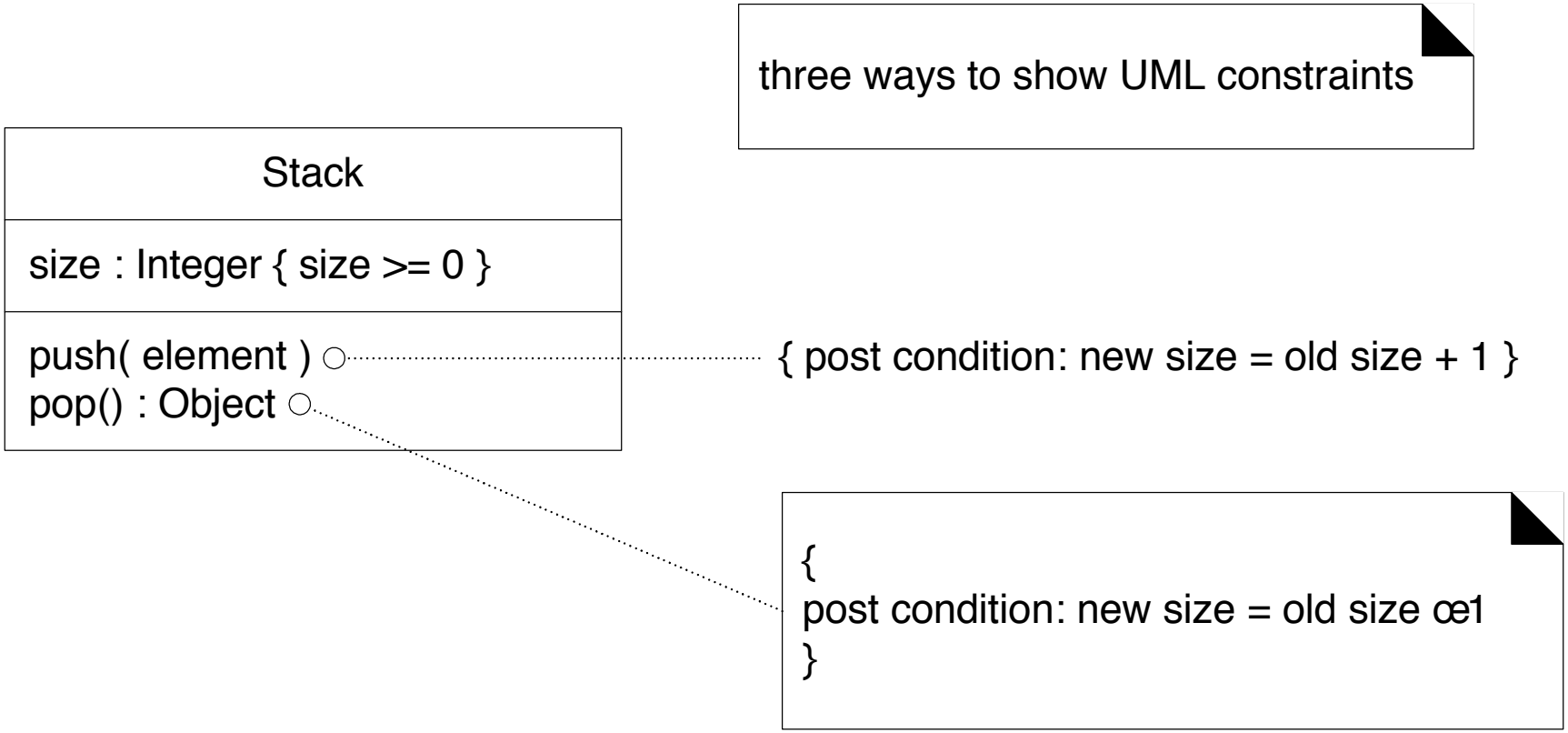
-the composite has sole responsibility for management of its parts, especially creation and deletion



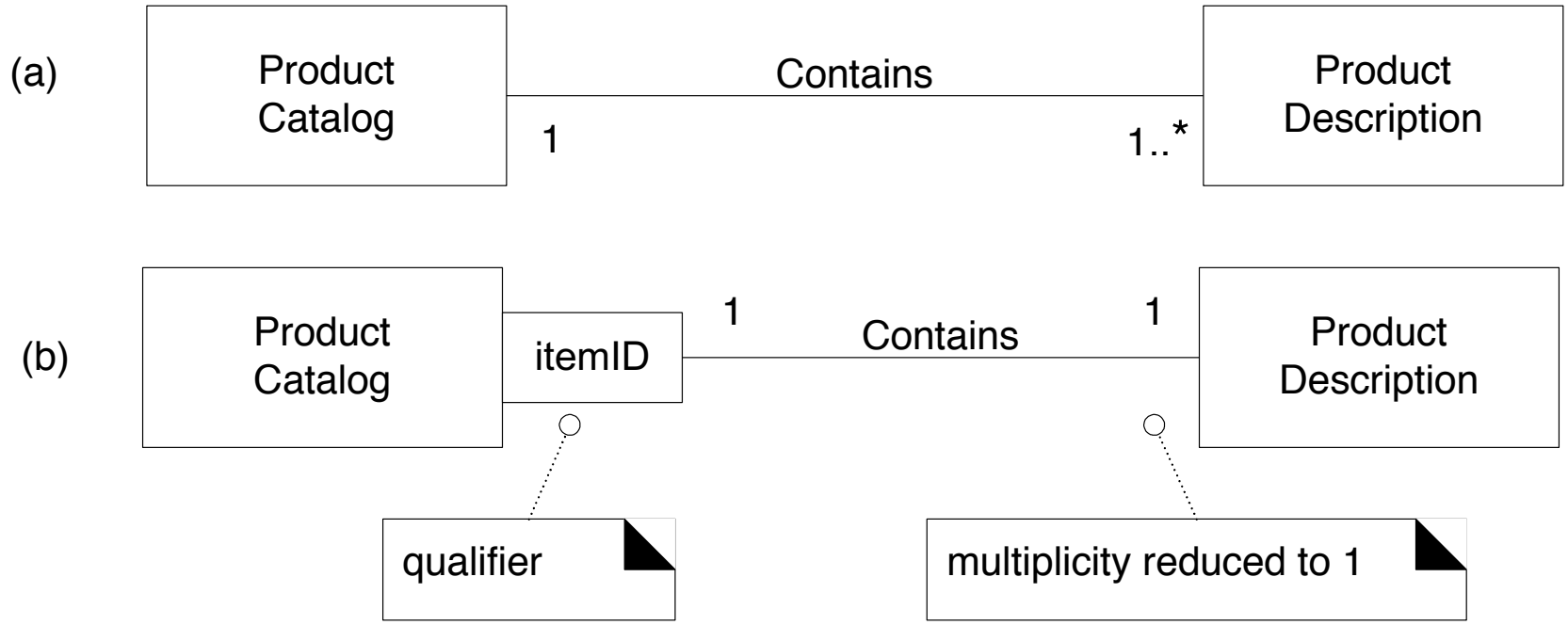


- Composition: A strong kind of association
  - An instance of Square (“the part”) belongs to only one Board (“the composite instance) at a given time
  - A Square object must always belong to a Board
    - No free-floating parts
  - The Board (composite) is responsible for creating and deleting the parts
  - If composite is destroyed, the parts must also be destroyed or attached to another composite.

Constraints Notation

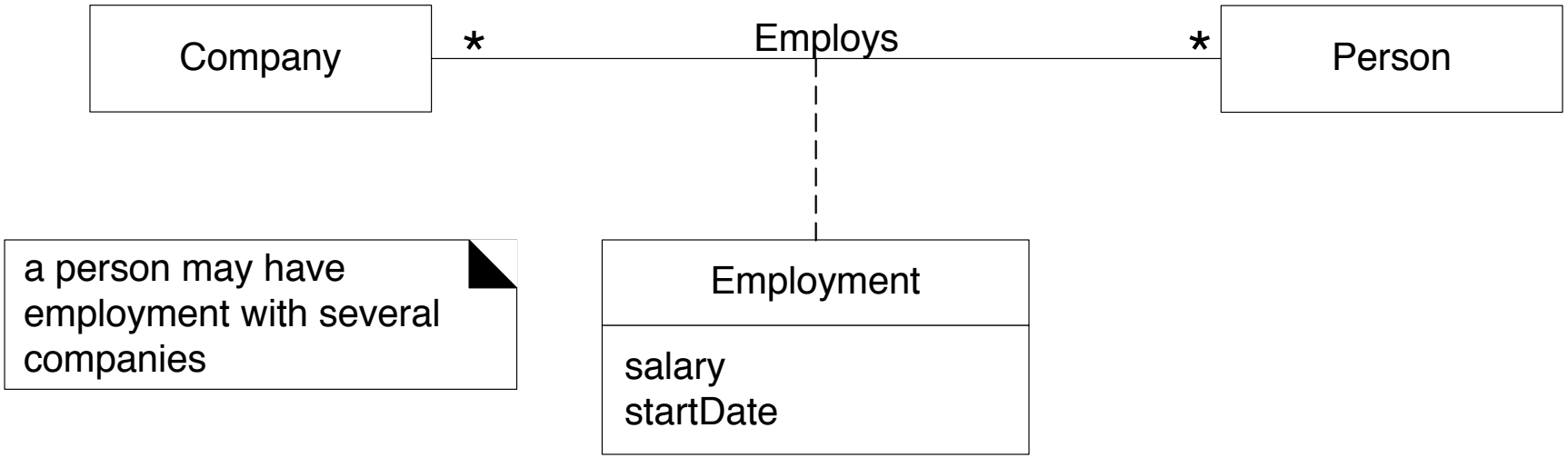


Qualified Associations



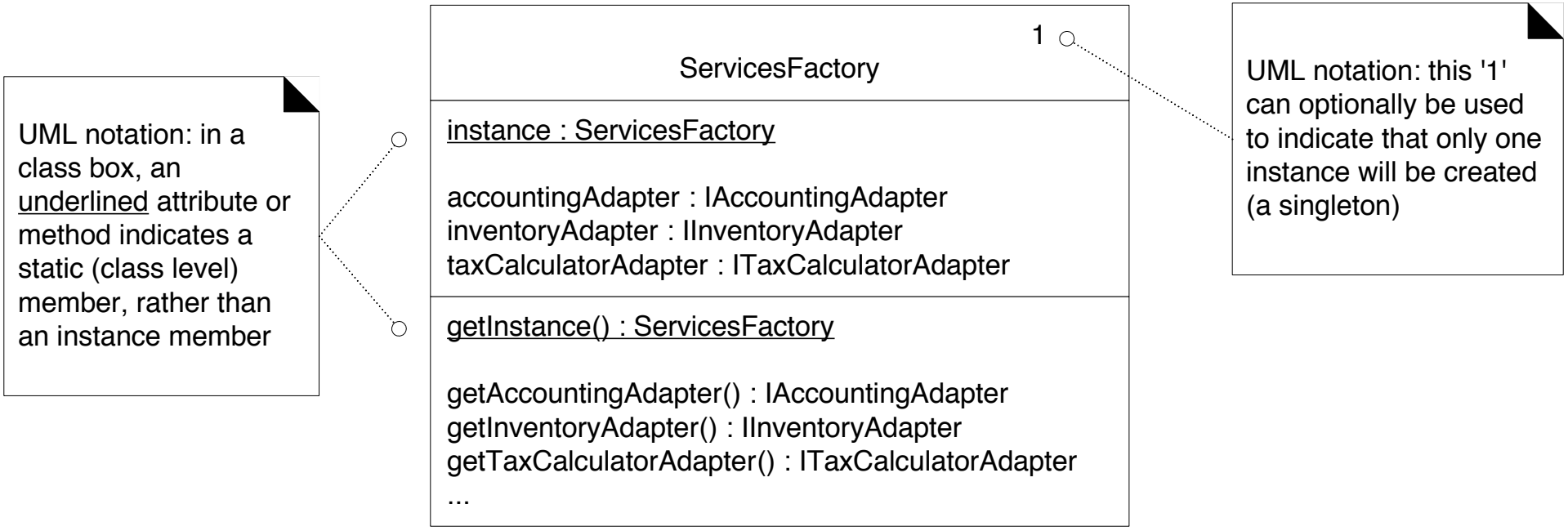
- Qualifier: Selects an object from a larger set of related objects
  - Based on the qualifier key
  - Suggests looking things up by key
    - e.g. in a hashtable
- Multiplicity on the association is changed from many to one.

Association Classes



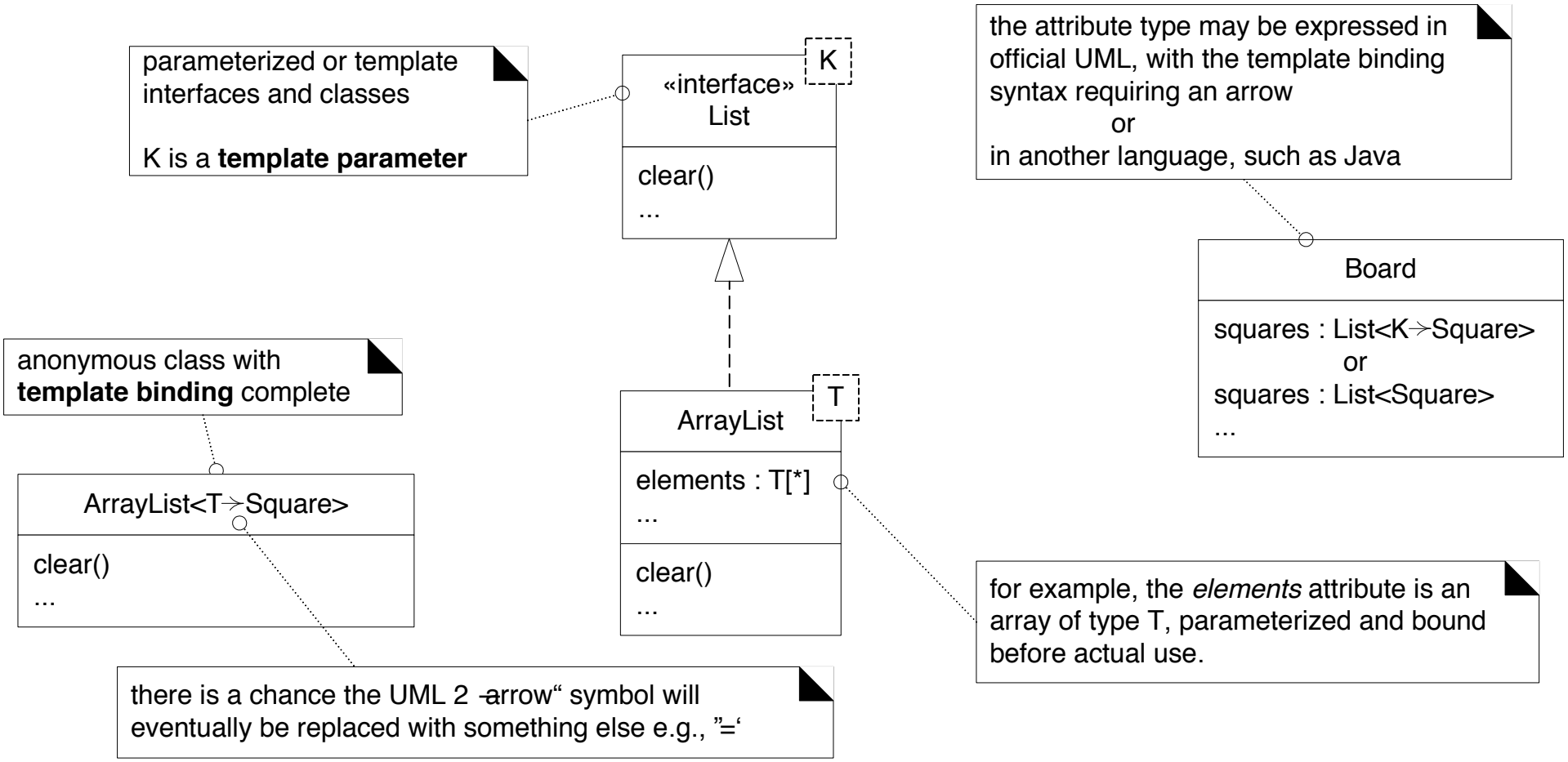
- The association itself is a class
  - Why: A lot of information needs to be remembered/stored about the association
    - attributes
    - operations

Singleton Classes

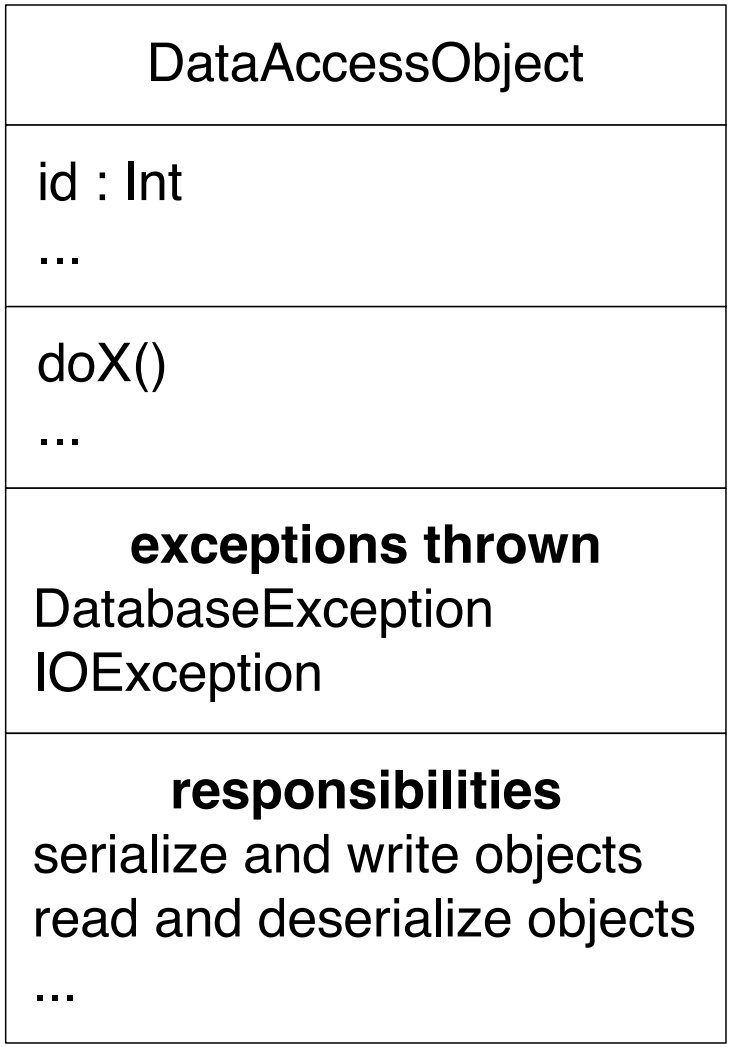


Template Classes and Interfaces

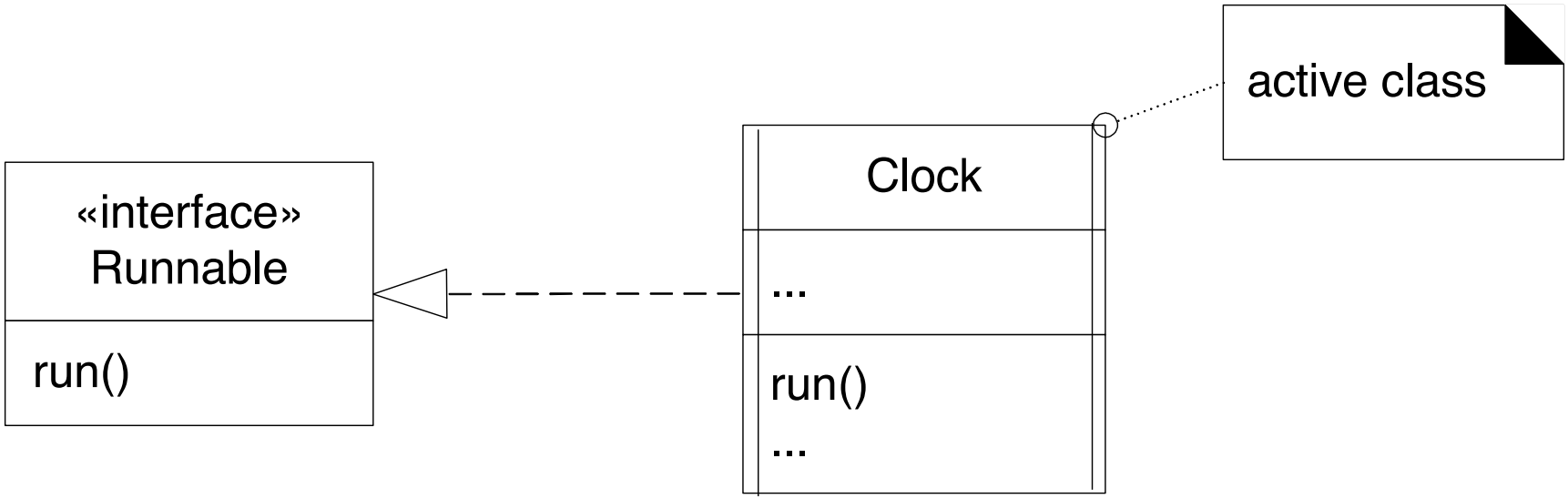
```
public class Board {
   private List<Square> squares = new ArrayList<Square>();
   // ...
}
```



User-Defined Compartments



Active Classes



- Active object: Runs on and controls its own thread of execution

Interaction diagrams help build and debug class Diagrams