Static Design Models

CSE 3311 & 5324

Christoph Csallner
University of Texas at Arlington (UTA)

UML Case Tool

- Should integrate with (or be part of) your IDE

- Should be able to parse your source code and generate UML diagrams from it
  - Overview, summary of your code
  - Reverse engineer parts of design from code
  - Class diagrams (common)
  - Sequence diagrams, etc. (less common)

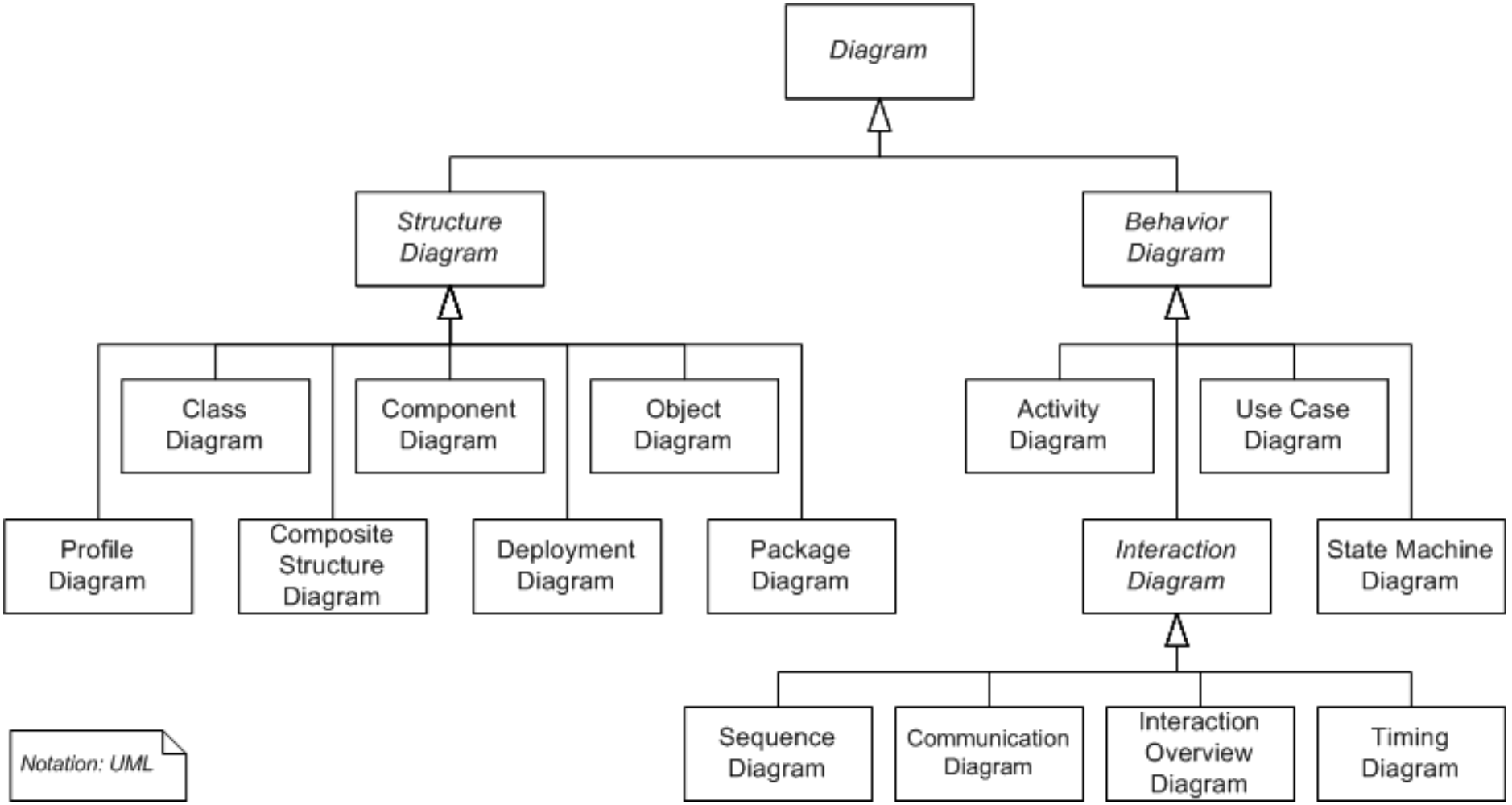
- What is your experience with UML tools?
  - UML → Code vs. Code → UML

Drawing UML vs. Coding

- Drawing UML alone is not enough
  - In the end, you get paid for code
  - Generating code from UML usually not worth the effort

- Spend about one day per three weeks on drawing
  - According to CL, page 215
  - “At the walls” = drawing UML on large whiteboards
  - Or drawing in a UML case tool

Taxonomy of UML Diagrams As Visualized by a UML Class Diagram



http://en.wikipedia.org/wiki/File:Uml_diagram2.png

Static vs. Dynamic Models

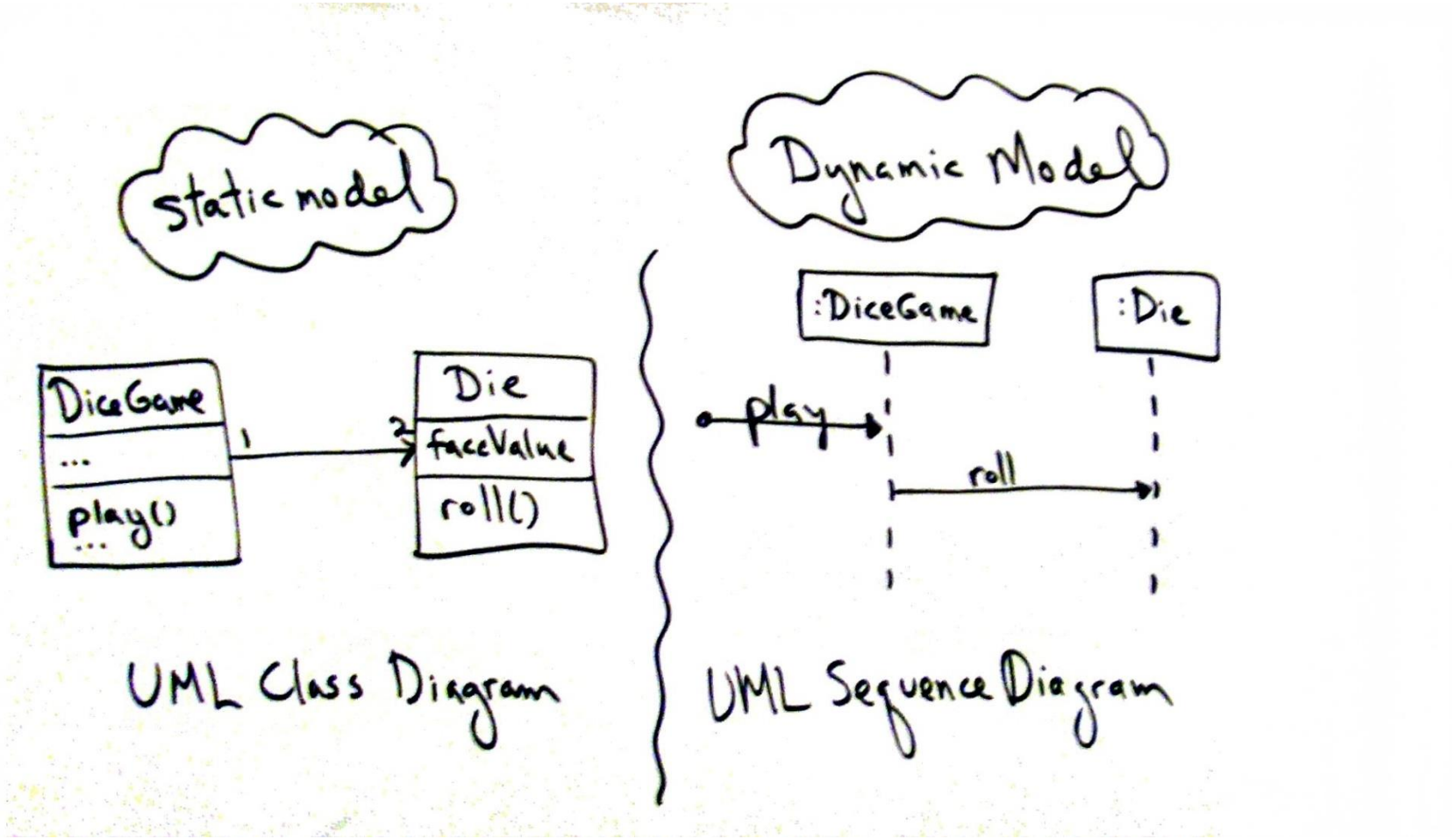
- Two categories of UML diagrams
- Static: Class diagram, …
  - Type definitions (including type dependencies)

- Dynamic: Sequence / Communication diagram, …
  - Show code execution, i.e., method calls
  - More interesting, difficult, important [CL, page 216]

- Guideline: Develop them in parallel

Static vs. Dynamic Models: Example

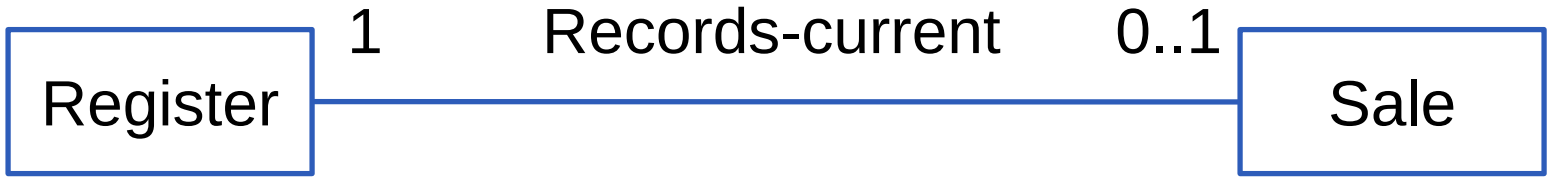
- Develop them in parallel:



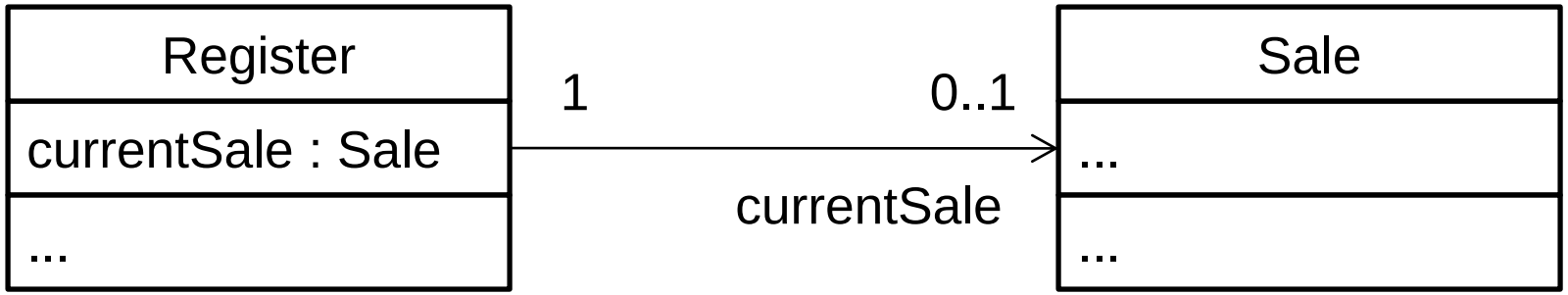
CL Figure 14.1

UML Class Diagram Notation

- Domain model: Part of requirements (the “What?”)

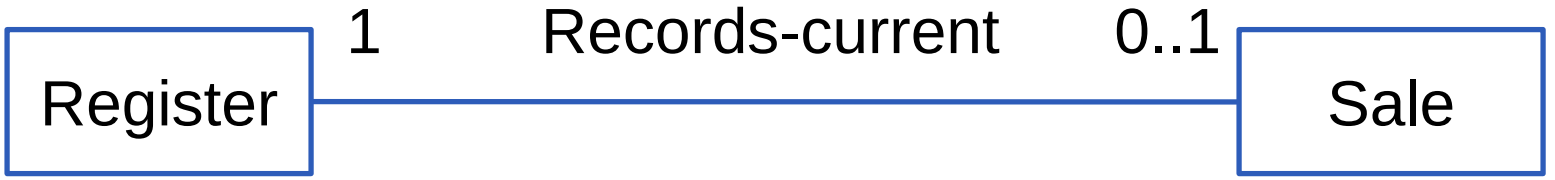


- Design class diagram (the “How?”)

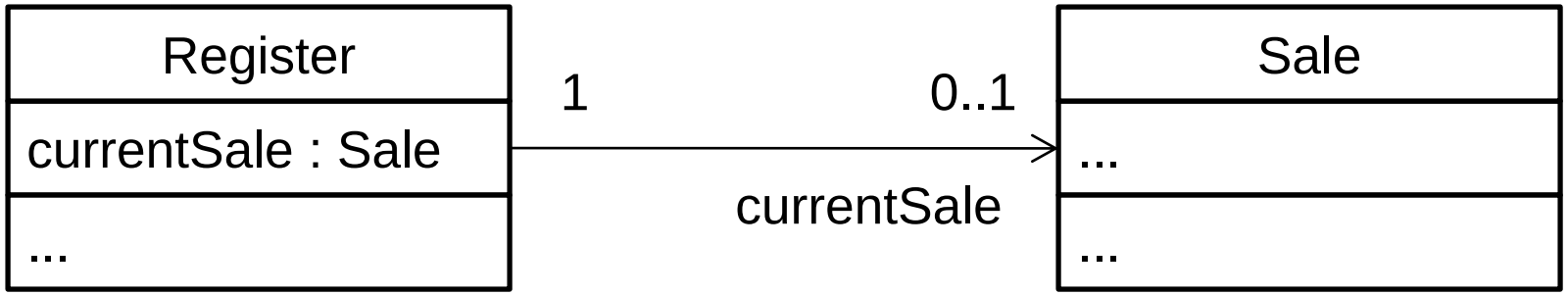


Box Named X Becomes Code Class X

- Domain model: Set of X items in the domain

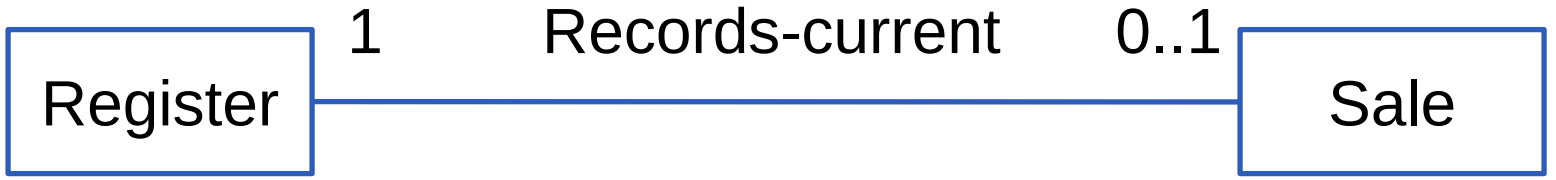


- Design class diagram:
  - Set of X objects = Instances (objects) of class X in code

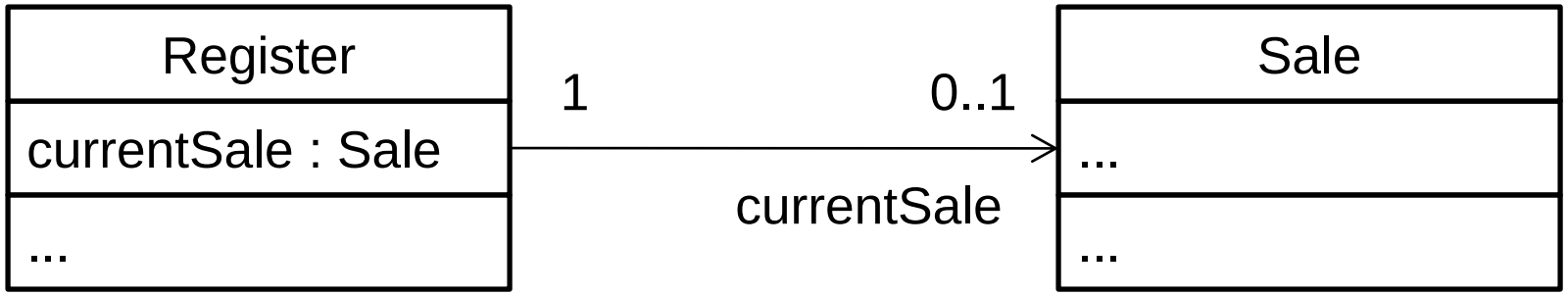


Connection = Relation on sets

- Domain model: Relation on domain entity sets
  - Register box represents all current registers in domain

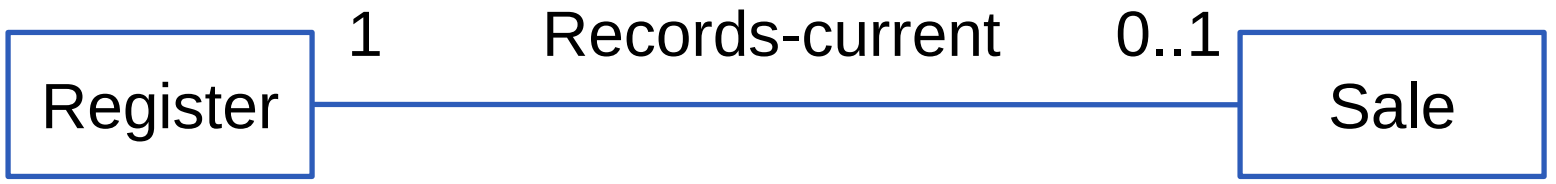


- Design: **Same relation! (Here on instance sets)**
  - Register box represents all current Register instances

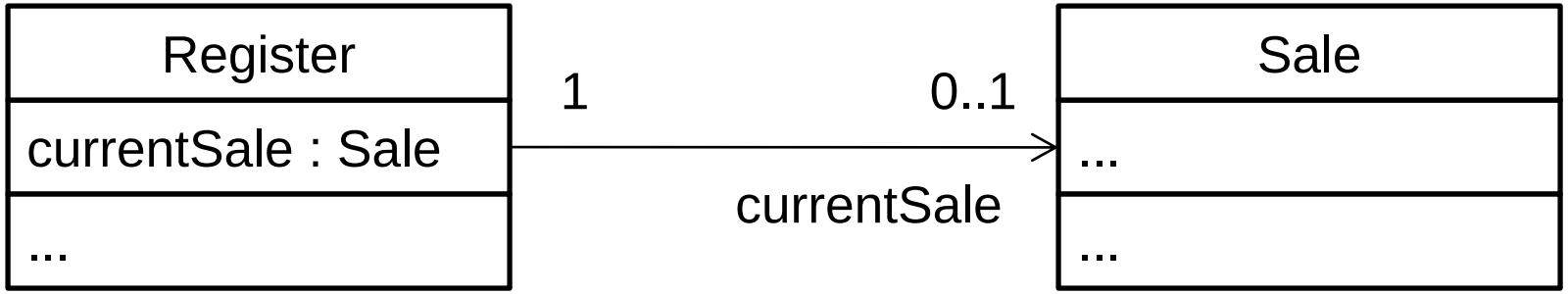


Connection in Code: Instance Field

- Constraints carry over to code: Multiplicity, ..

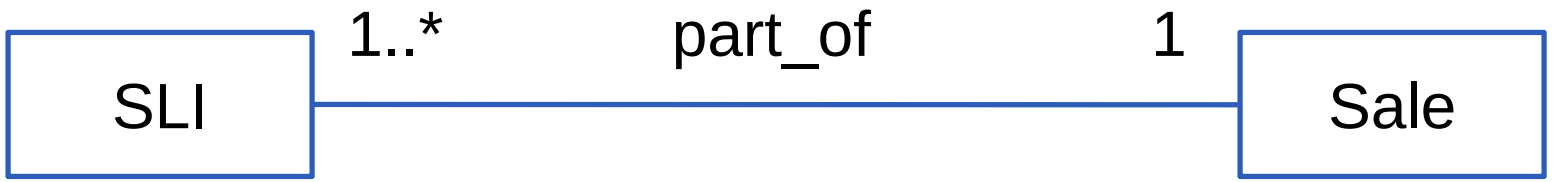


- Design class diagram: Attribute = Field
  - Think of an instance field as a Math. Function (next slide)

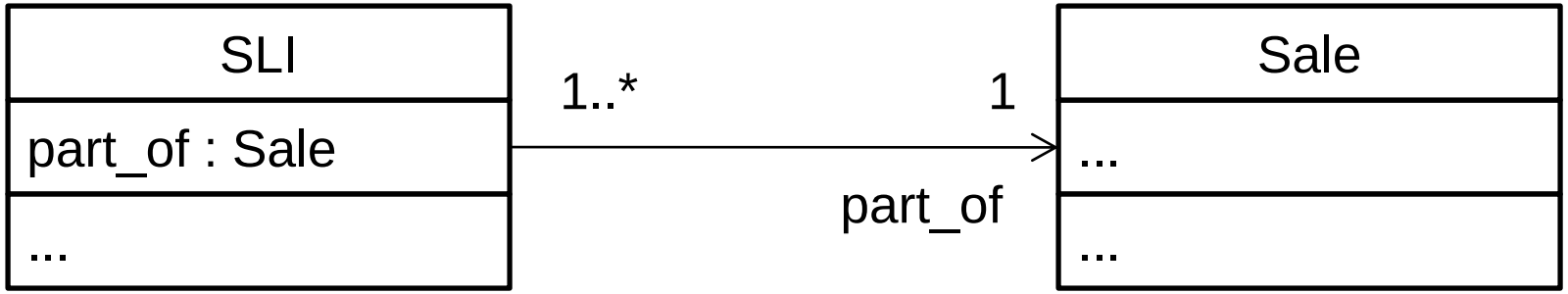


Example: SLI -- Sale

- SLI = Sales Line Item

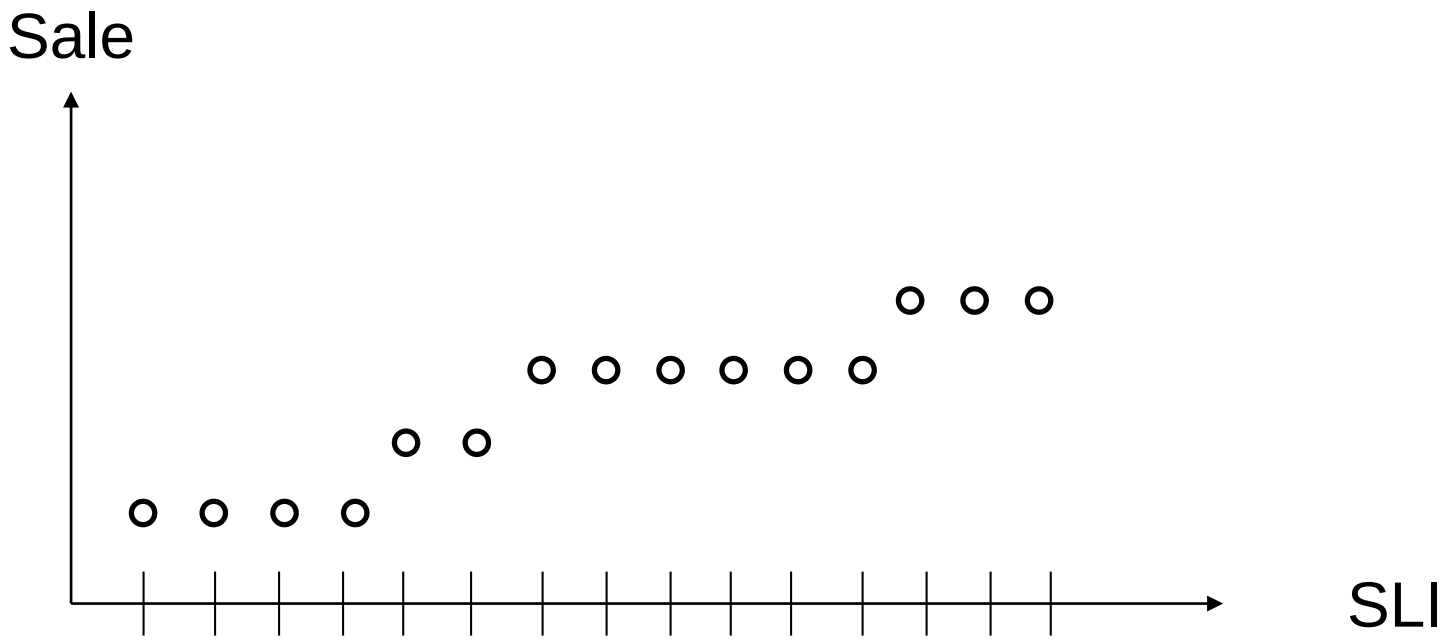


- Design class diagram: Attribute = Field
  - Think of an instance field as a Math. Function (next slide)



part_of as a x/y Plot

- Function: Each x-value maps to **up to one** y-value
  - In part_of example: Each x-value maps to exactly one y-value (the “1” annotation at Sale)



Instance Field as a Math. Function

- Map: a function (in Math sense, not method in OO)
- from an instance value to a field value, e.g.:

```
public class SLI {                          // SaleLineItem
    protected Sale partOf;                  // field of type Sale
    public SLI(Sale s) { partOf = s; }      // constructor
}
```

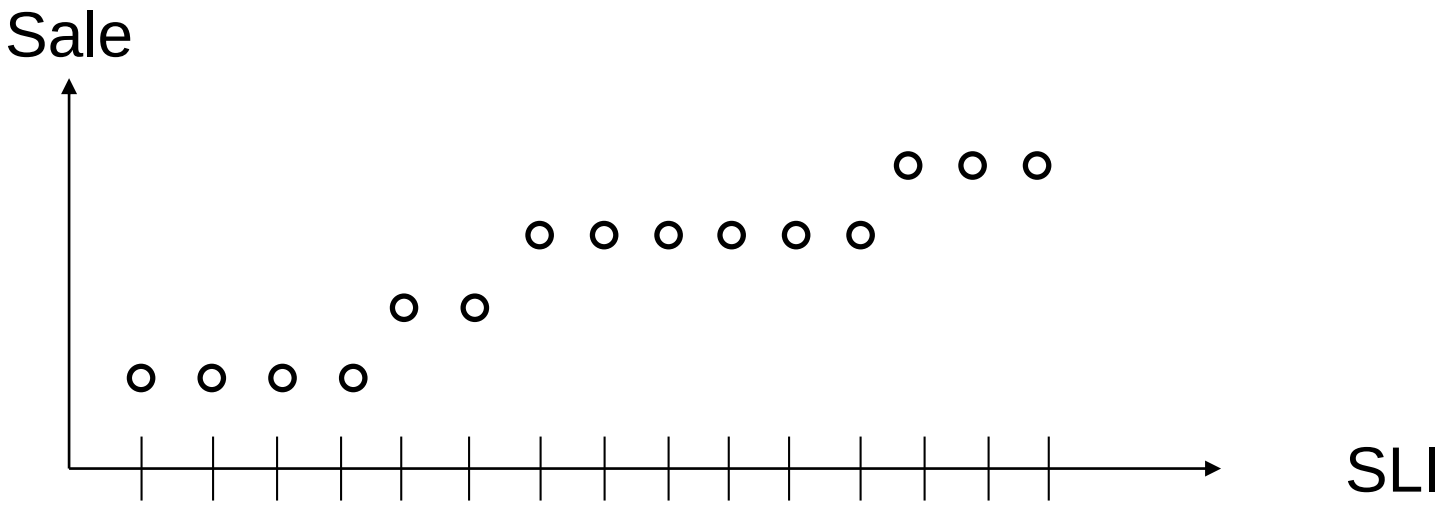
- partOf: SLI → Sale // map an SLI to a Sale
- Example usage:

```
void foo(SLI sli) {          // can read as function application:
  Sale s = sli.partOf;       // sli.partOf === partOf(sli)
}
```

Instance Field as a Math. Function

```
public class SLI {                          // SaleLineItem
    protected Sale partOf;                  /* … */
}
```

- part_of field of each SLI instance can point to zero or one Sale instances



Instance Field as “Function”: Example

- Function from instance value to field value, e.g.:

```
public class SLI {                          // SaleLineItem
    protected Sale partOf;                  // field of type Sale
    public SLI(Sale s) { partOf = s; }      // constructor
}
```

- partOf: SLI → Sale

```
{ (sli_1, sale_1),
  (sli_2, sale_1),
  (sli_3, sale_2) }
```

// in code:

```
sli1 = new SLI(sale_1);
sli2 = new SLI(sale_1);
sli3 = new SLI(sale_2);
```

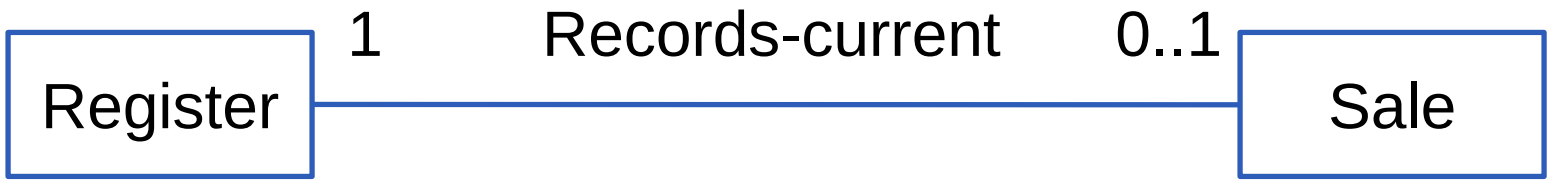
- Illegal:

```
{ (sli_4, sale_1),
  (sli_4, sale_2) }
```

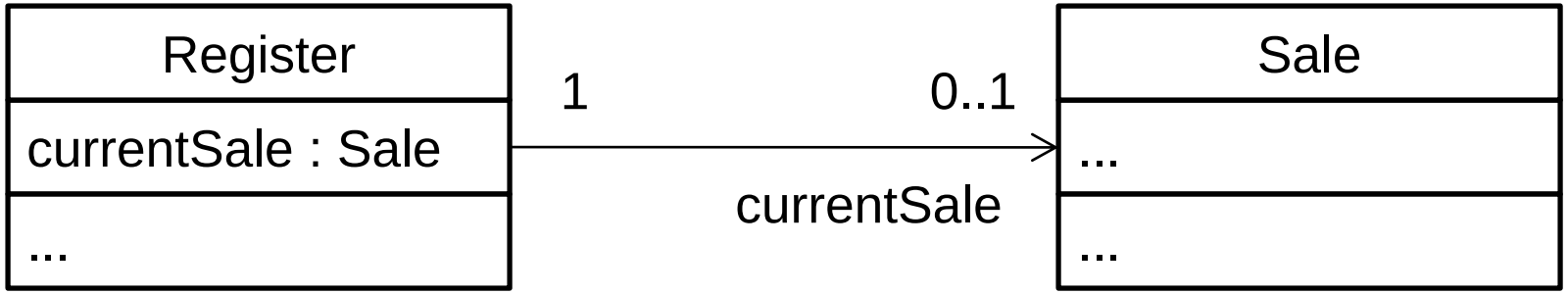
```
sli4 = new SLI(sale_1);
sli4.partOf = sale_2;
```

Another Example: Register -- Sale

- Constraints carry over to code: Multiplicity, ..

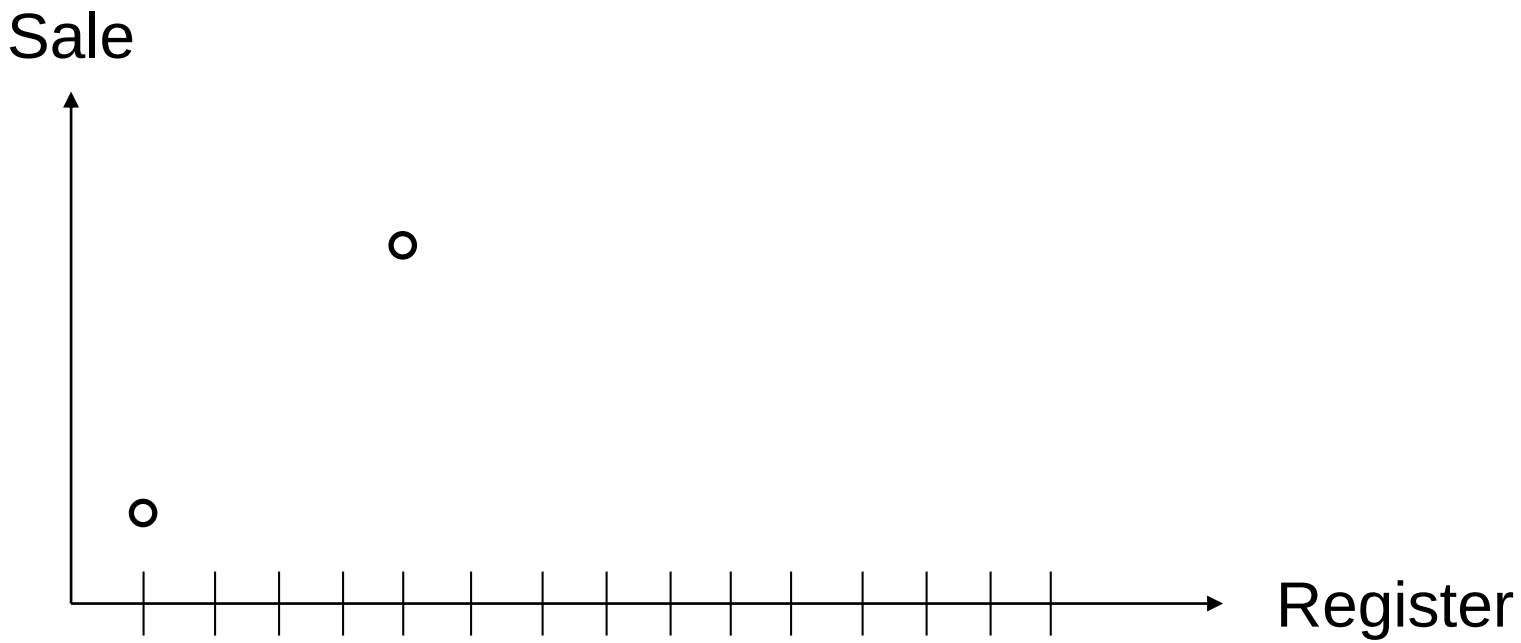


- Design class diagram: Attribute = Field
  - Think of an instance field as a Math. Function (next slide)

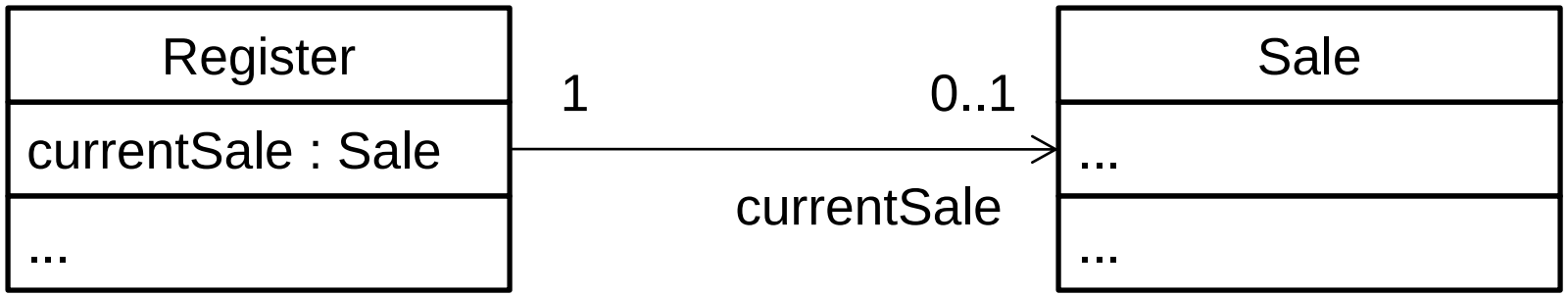


currentSale as a x/y Plot

- Function: Each x-value maps to up to one y-value
  - In currentSale example: “0..1” at Sale, so ok if a register maps to zero sales



Recap: currentSale in Domain Model

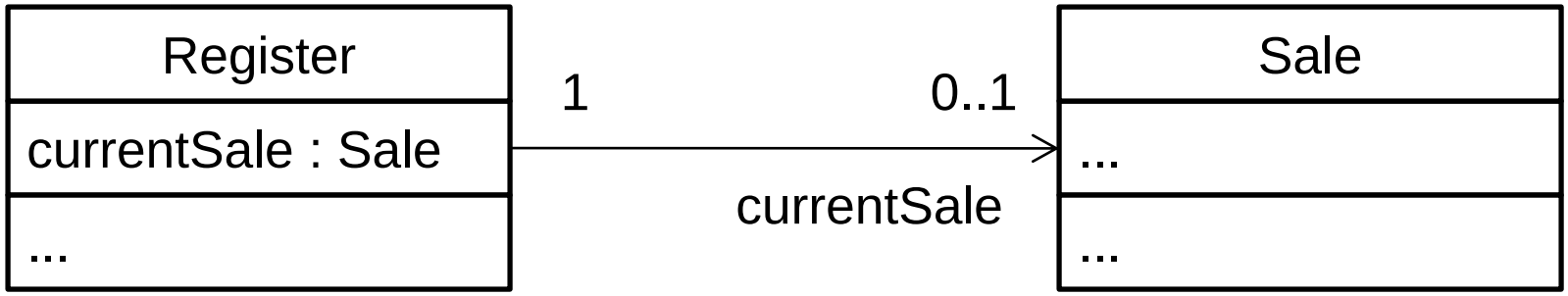


- "0..1" part of multiplicity constraint made currentSale a Math. function
  - Each register has up to one current sales

- Recall: Function is a relation where each left-side element maps to up to one right-side elements
  - Each left-side element appears in up to one tuples

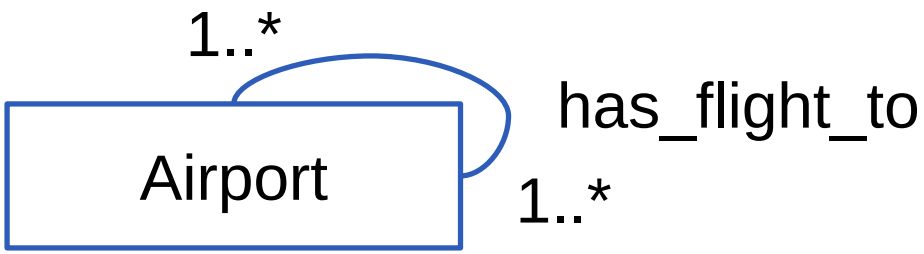
Recap: currentSale in OO Program

- Register object = Instance of the Register class
  - Each Register object has one “currentSale” pointer
- At any point in time: **Each Register object points to zero or one Sale objects**



How Can a Function Hold a Relation?

- Domain model: Association = **Relation** on sets



- **Important: A relation may not be a function**
- But we use instance field (a Math. Function)
- Trick: Simulate relation w/ function
- Use instance field that is a collection / array
  - "Option 1" on next slide

Storing a N:M Relation

- Option 1: As a field that has a collection type, e.g.:

```
public class Airport {
        private Set<Airport> connections;
        public Airport(Set<Airport> s) { connections = s; }
}
```

- Option 2: Explicitly in a single data structure, e.g.:

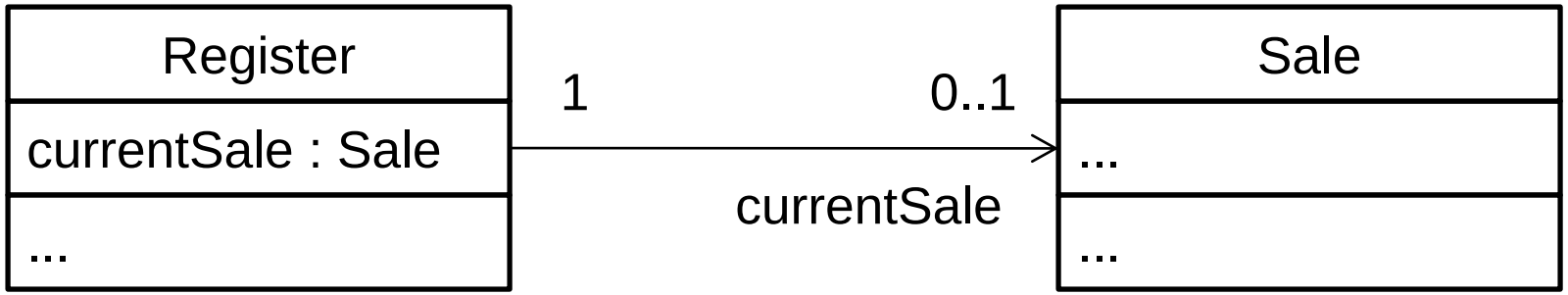
```
public class Connections {
        private Map<Airport, Set<Airport>> connections;
        public Connections(Map<Airport, Set<Airport>> m) {
                connections = m; }
        }
}
```

- Can replace set with other collection or array

Attribute Notation: Direction

- Direction
  - Stick arrowhead
  - “Navigation”, “navigability”

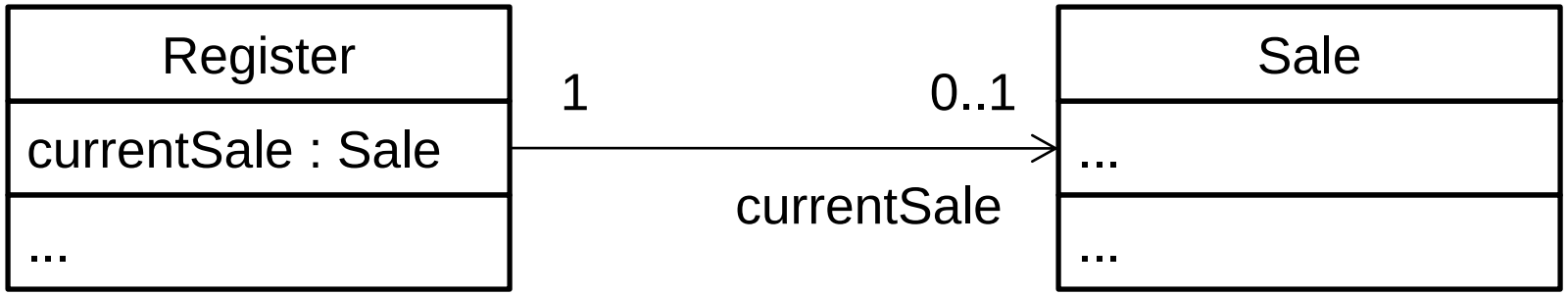
- From source class that contains the object reference field to the referred-to target class
  - Field access: source object → field value of source object



Attribute Notation in Class Diagram

- Text vs. line
  - Reference field: Use a line
  - Simple field (int, boolean, etc.): Use text

- When using a line, where show attribute name?
  - At target, but not at source

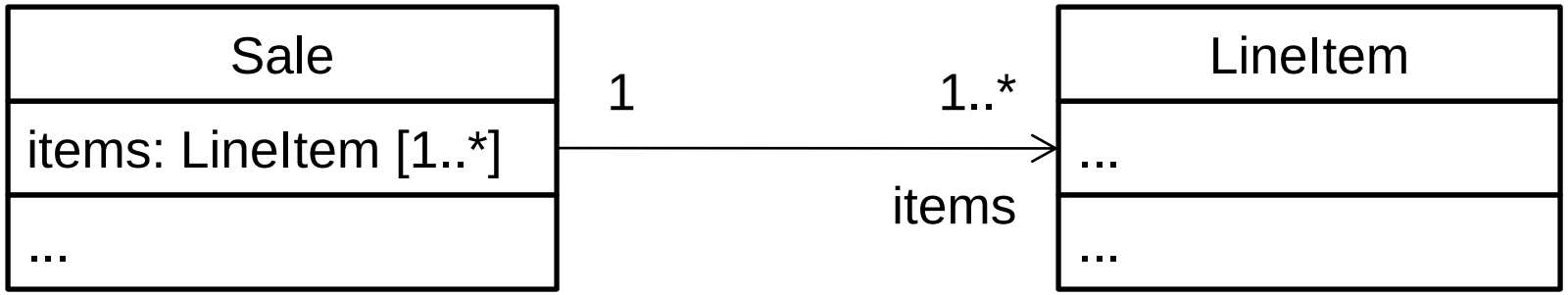


Collection Attribute Example

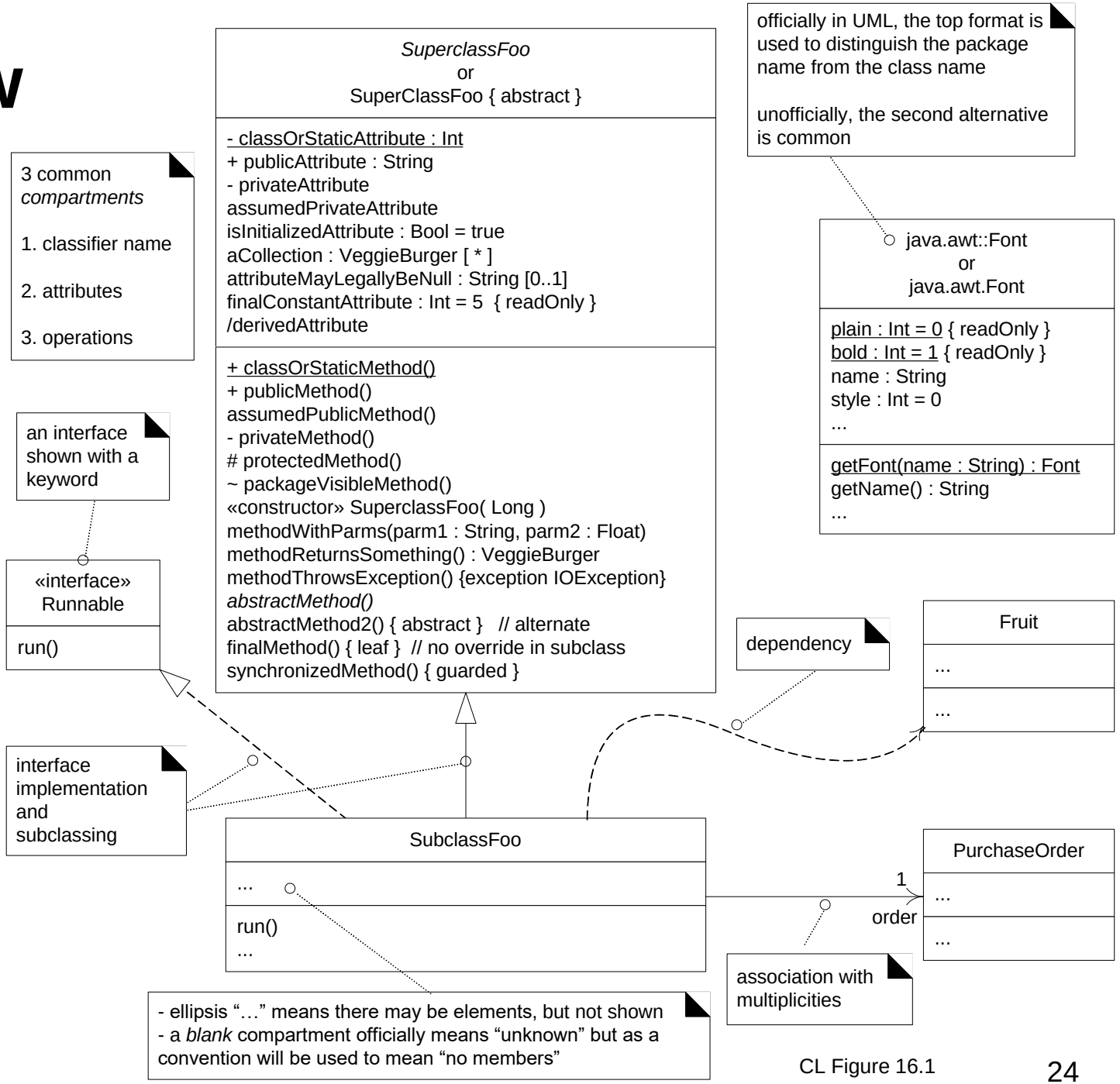
- Attribute = Field (Math. Function)
  - object → attribute(object)
  - Value may be a **collection** that contains element values
- Example: Map each sale to its line items:

```
class Sale {
   List<LineItem> items;  // sale → items(sale)
}
```

- But who enforces “1” multiplicity constraint?



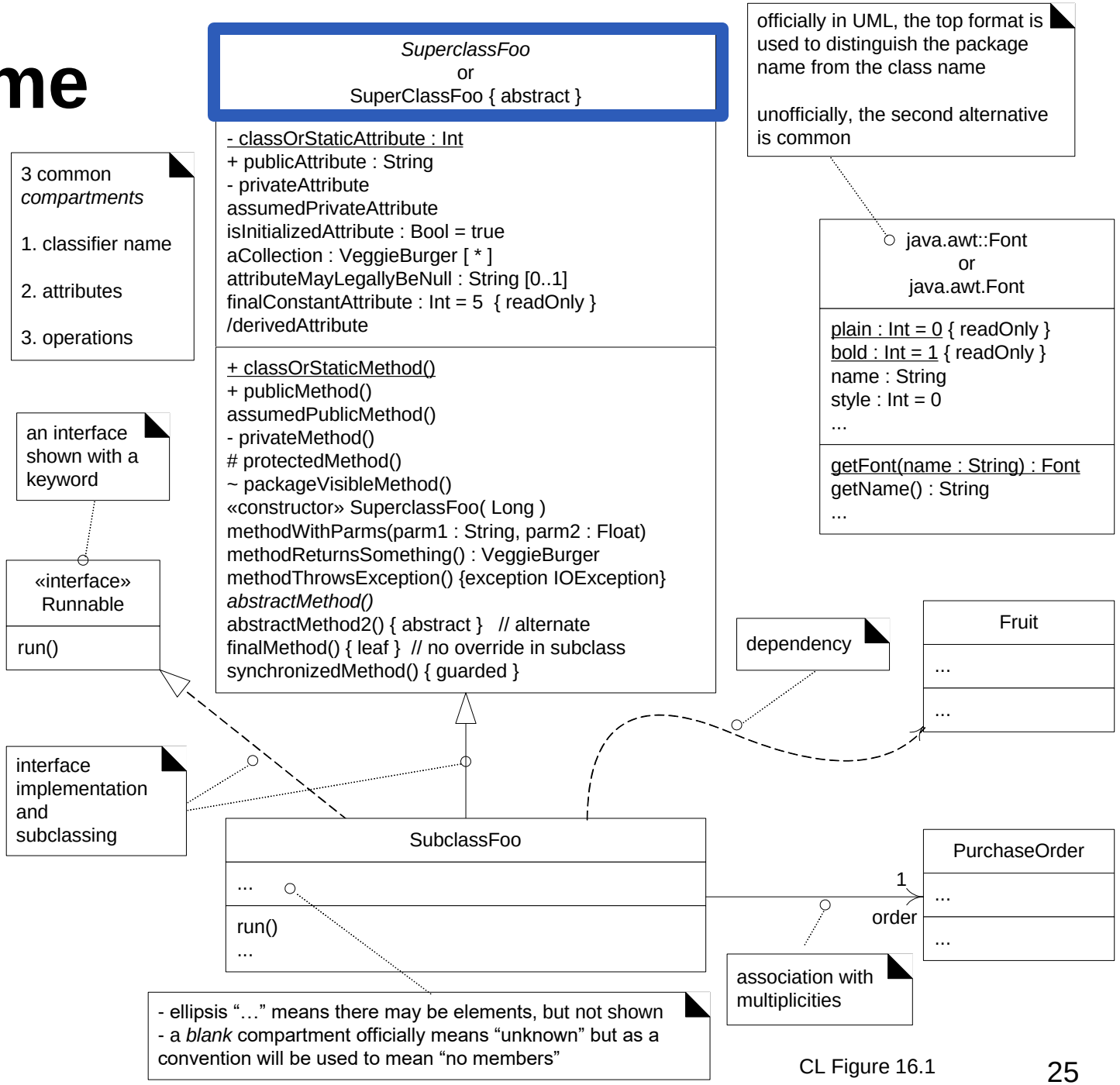
Overview



CL Figure 16.1

Type Name

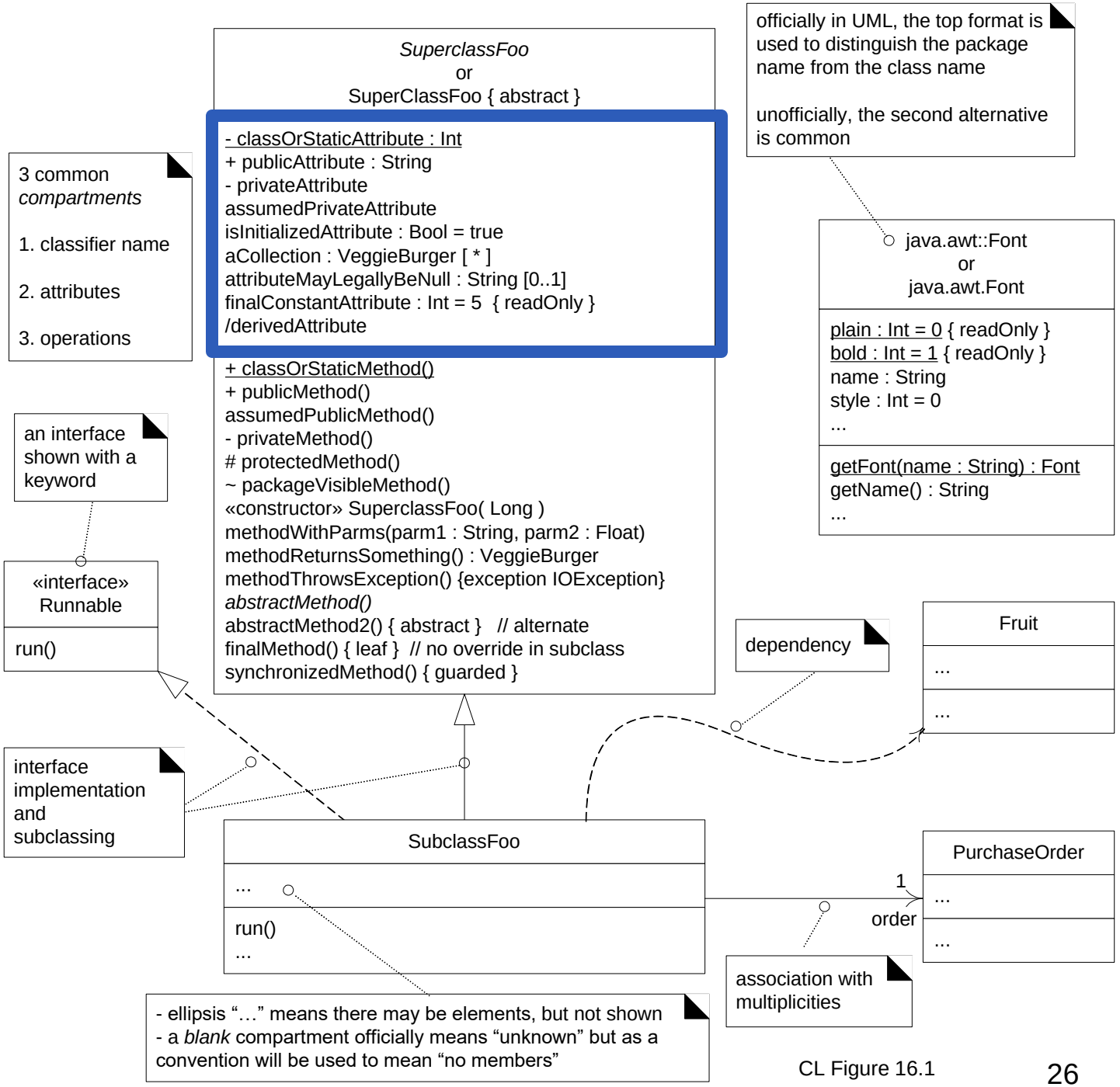
- Top box



CL Figure 16.1

Attribute

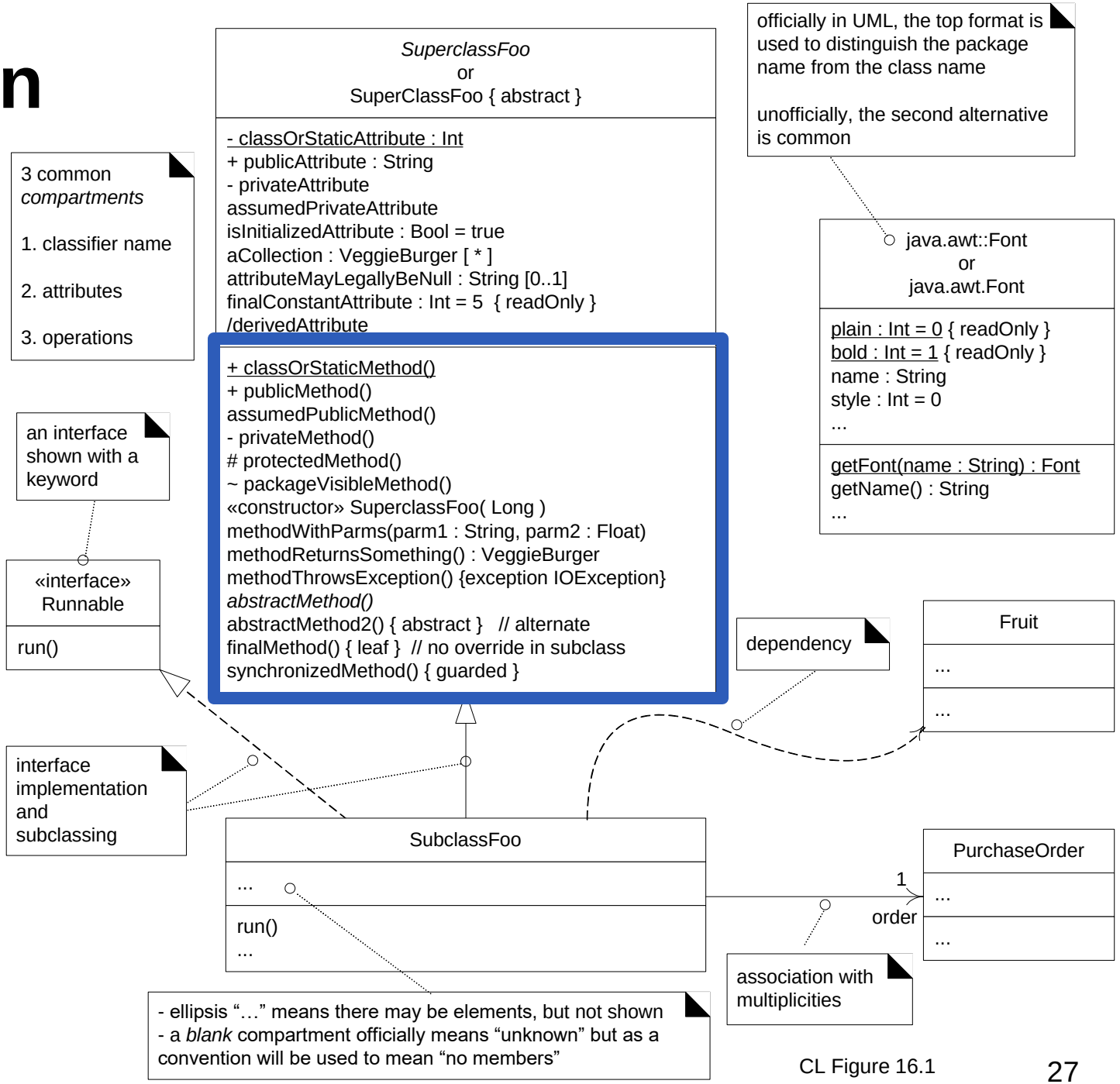
- Field
- Middle box
- Default: Private



CL Figure 16.1

Operation

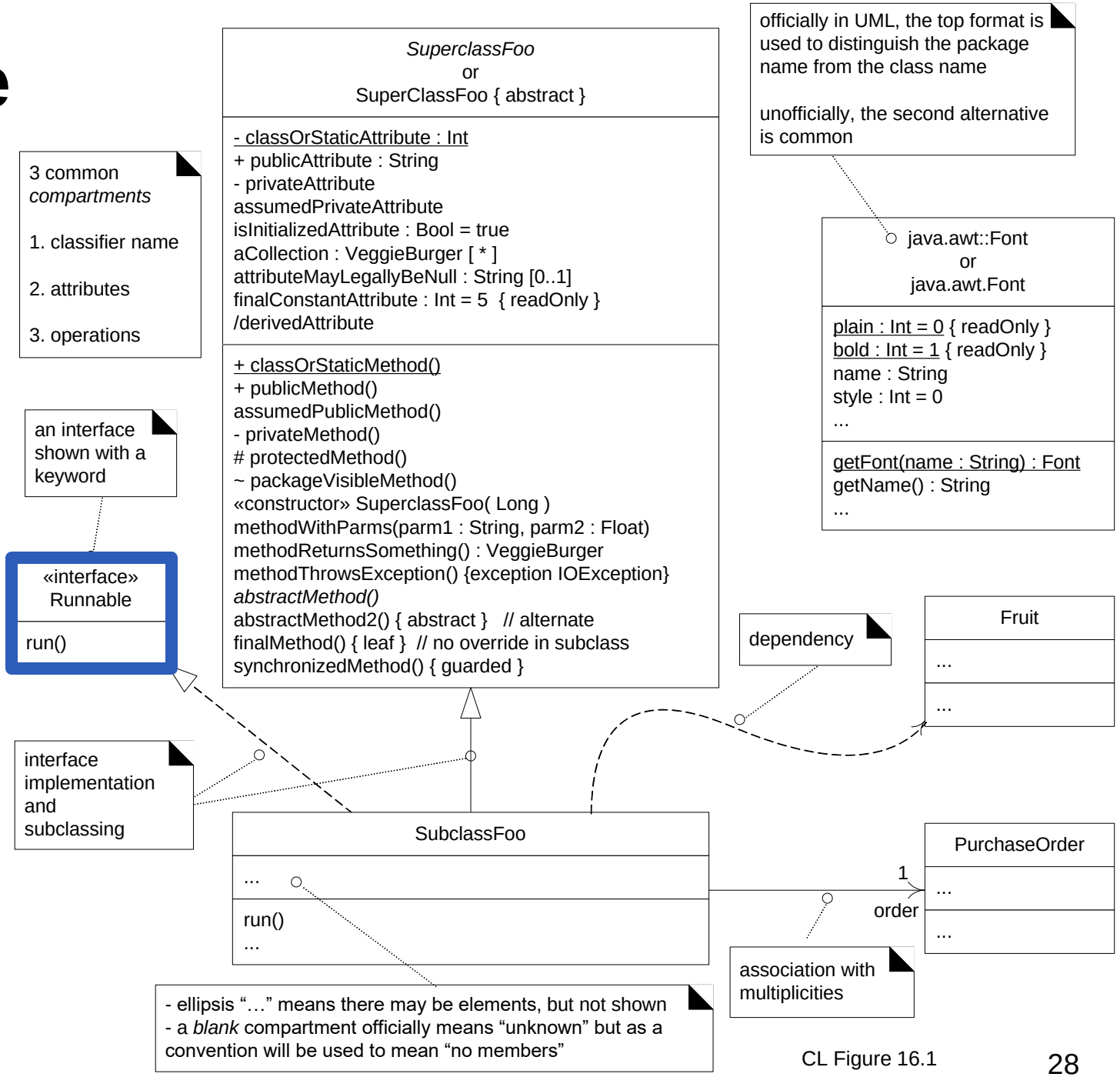
- Method signature
- Bottom box
- Default: Public
- Usually do not show getter or setter methods



CL Figure 16.1

Interface

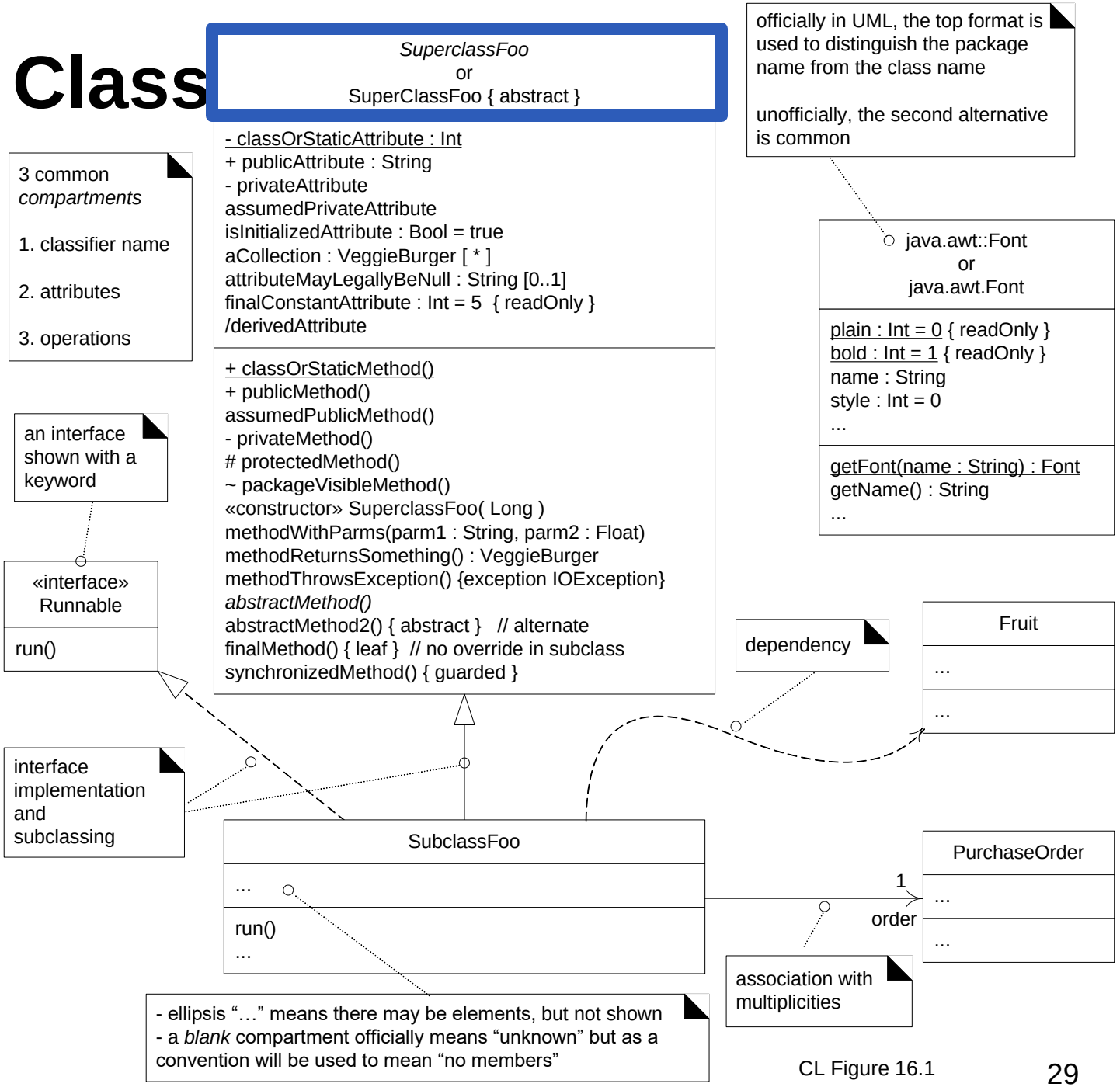
- Keyword
  «interface»



CL Figure 16.1

Abstract Class

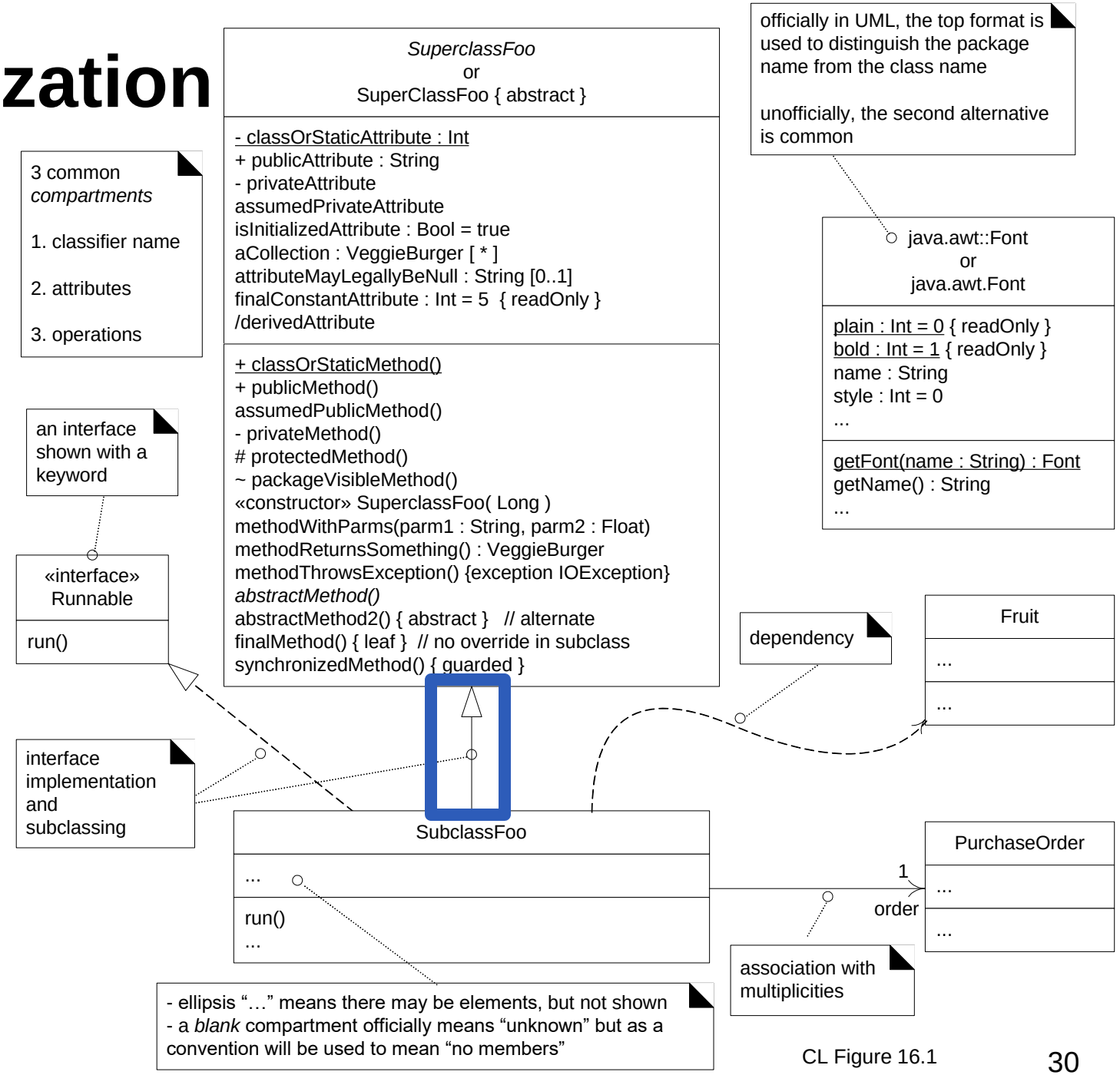
- Italic or {abstract}



CL Figure 16.1

Generalization

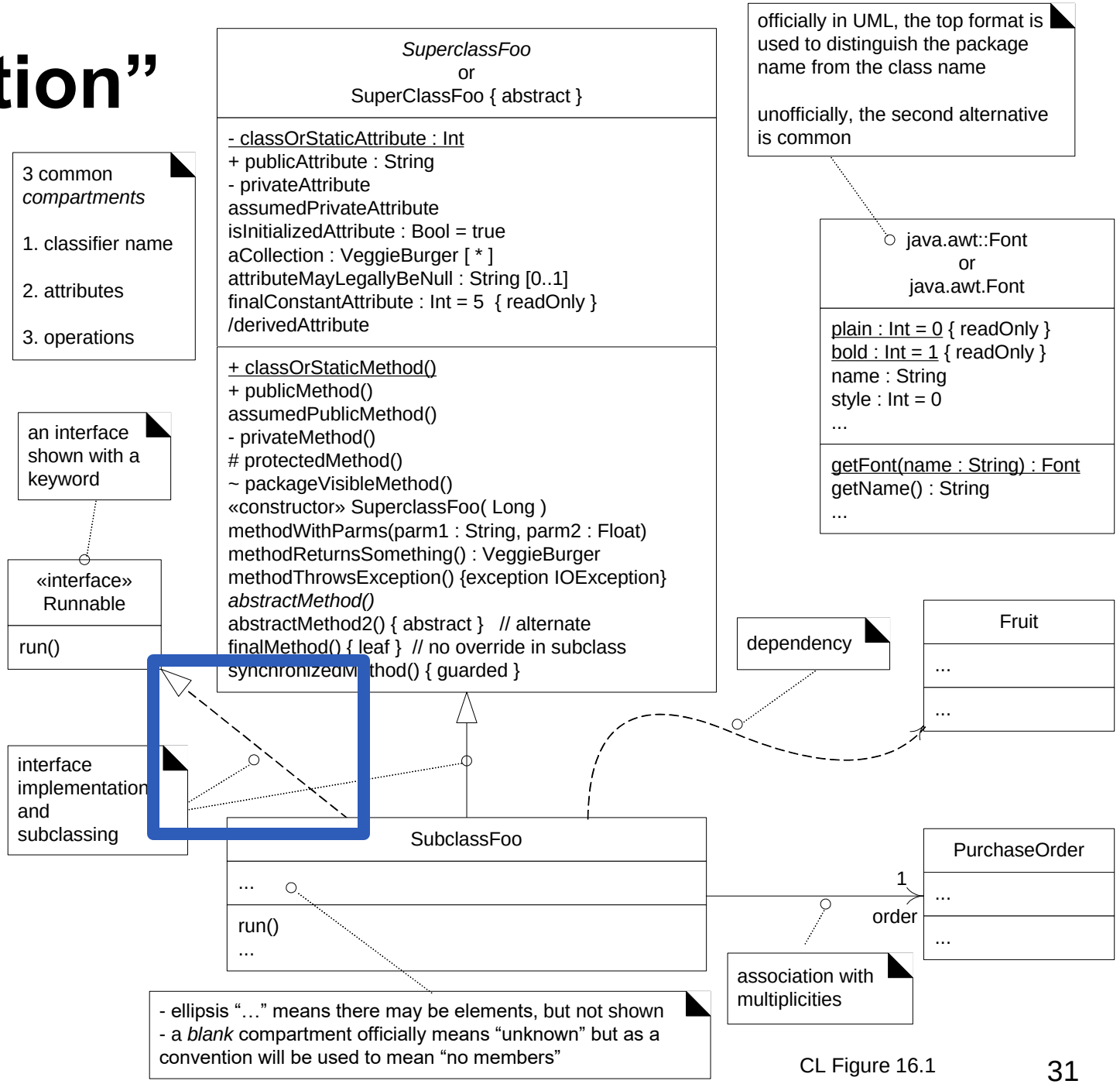
- Domain model: Subset relation
- From class to class
- Inherit code and fields
- Java: extends



CL Figure 16.1

“Realization”

- Domain model: Subset relation
- From class to interface
- Inherit only signature
- Java: implements



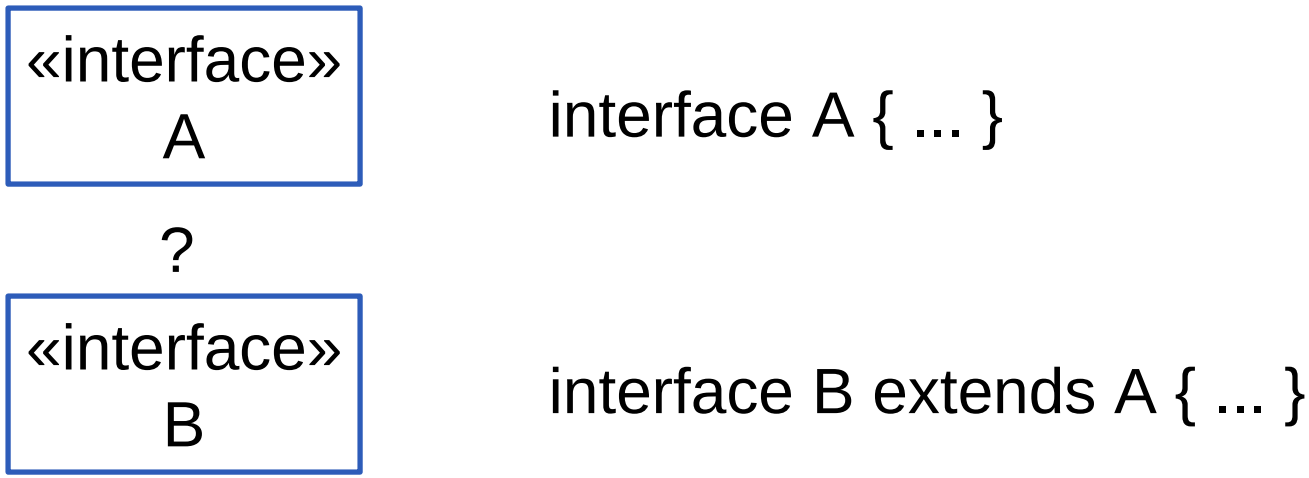
CL Figure 16.1

Two Subtype Relation Arrows

- Why do we need realization if we have generalization?
  - Not clear
  - Realization seems redundant
  - Maybe to visually distinguish arrow from class to interface

Subtype Relation Arrows: Example

- Which arrow should we use between two interfaces?



  - Not clear from CL or UMLUG2

Generalization vs. Realization

- Guideline: By default, use generalization
- Guideline: To visually highlight (interface → class) generalization arrows, use realization

IN-CLASS EXERCISE: STATIC DESIGN MODEL WITH MULTIPLICITY CONSTRAINTS

Domain Model → Code

- Get together with your team

- Convert your team project's domain model to code
  1. Locate or re-produce (a part of) your team project's domain model UML class diagram
  2. Convert your domain model into corresponding (object-oriented) class definitions
  3. Especially focus on 1:1, 1:N, N:M associations and how they map to fields

- Post your results in the main class Teams chat