Using UML to express Software Architecture

Outline

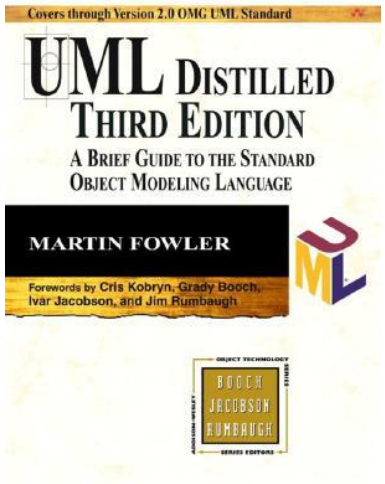
- UML overview
- UML class diagrams
- Class exercise – Ski school system
- [Fri] UML sequence diagrams

Readings

- Practical UML: A hands on introduction for developers

  http://dn.codegear.com/article/31863

- If you want to learn more about UML, there are a number of UML books (and web tutorials) available, including “UML Distilled”, by Martin Fowler.



UML

In an effort to promote Object Oriented designs, three leading object oriented programming researchers joined ranks to combine their languages:

- Grady Booch (BOOCH)
- Jim Rumbaugh (OML: object modeling technique)
- Ivar Jacobsen (OOSE: object oriented software eng)

and come up with an industry standard [mid 1990's].

UML – Unified Modeling Language

- The result is large (as one might expect)
  - Union of all Modeling Languages ☺
    - Use case diagrams
    - Class diagrams
    - Object diagrams
    - Sequence diagrams
    - Collaboration diagrams
    - Statechart diagrams
    - Activity diagrams
    - Component diagrams
    - Deployment diagrams
    - ….
  - But it's a nice standard that has been embraced by the industry.

UML class diagrams

UML class diagram:

a picture of the classes in an OO system,
their fields and methods, and
connections between the classes that
interact or inherit from each other

Does not include:

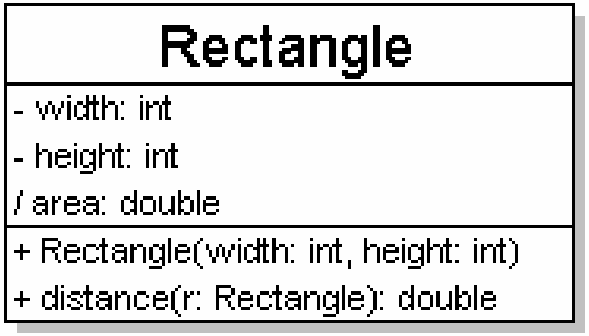
- details of *how* the classes interact with each other
- algorithmic details
- how a particular behavior is implemented

Practice as we go

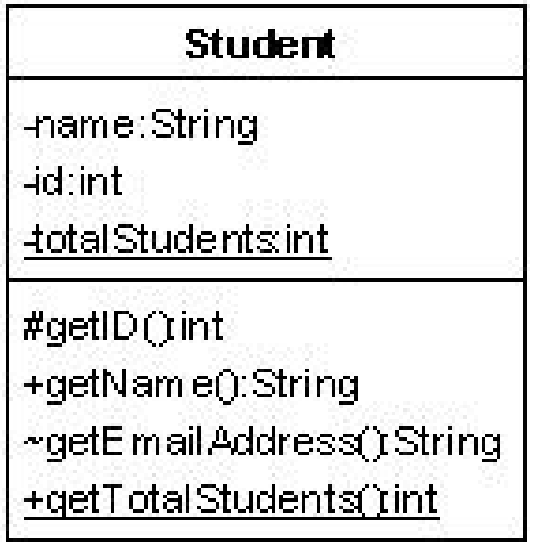
- Complete the templates for a
  - Square
    - Length
    - Perimeter
    - Area
  - Circle
    - Radius
    - Perimeter
    - Area

Diagram of one class

- class name in top of box
  - use *italics* for an *abstract class* name
  - + <<interface>> if an interface class



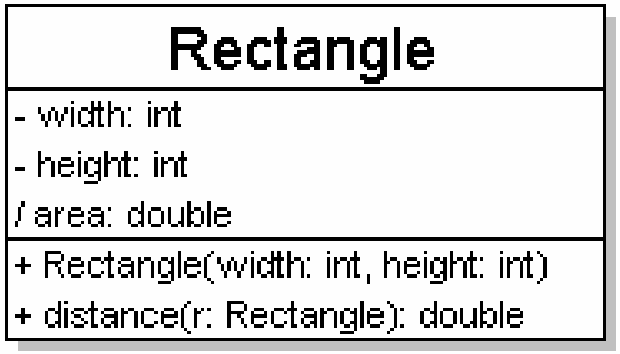
- attributes
  - should include all fields of the object

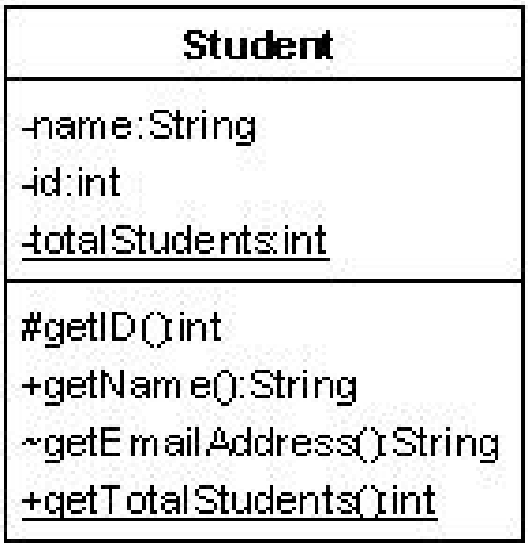


- operations / methods
  - may omit trivial (get/set) methods
  - should not include inherited methods

Class attributes

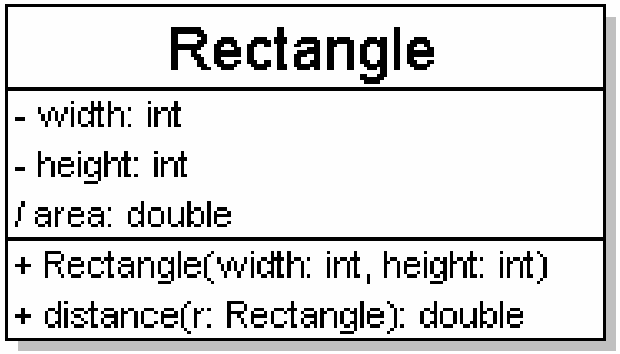
- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility: + public
    # protected
    - private
    ~ package (default)
    / derived
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values

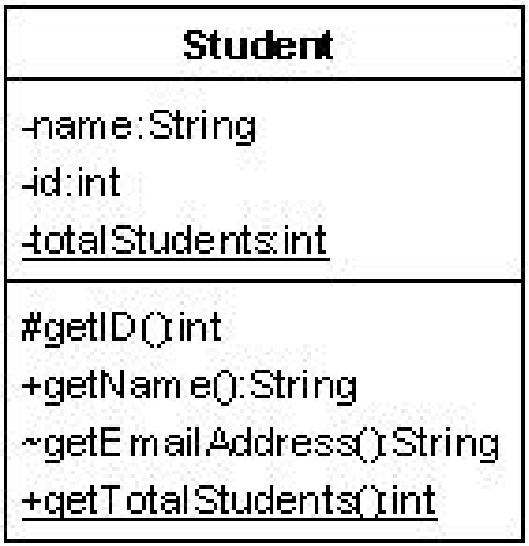




Class operations / methods

- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility: + public
    # protected
    - private
    ~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void





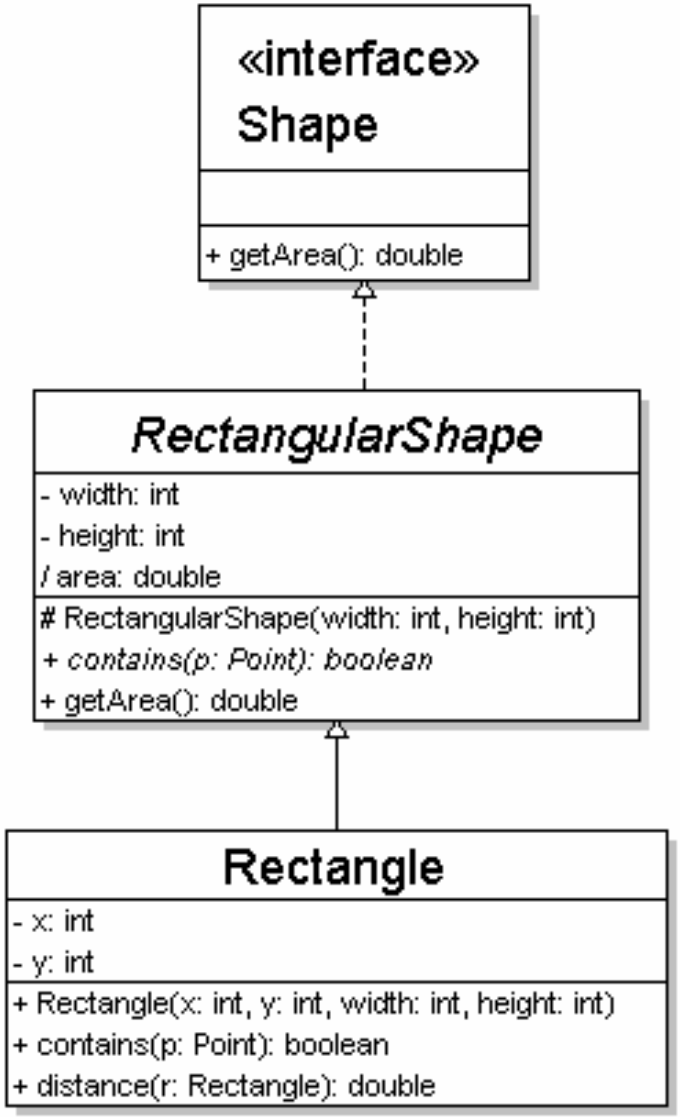
Relationships btwn. classes

- generalization: an inheritance relationship (isa)
  - inheritance between classes
  - interface implementation

- association: a usage relationship
  - dependency
  - aggregation (class is formed as a collection of others)
  - composition (variant of aggregation when a contained class will not exist without the container class)

Generalization relationships

- generalization (inheritance)
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - abstract class:
      solid line, white arrow
    - interface:
      dashed line, white arrow



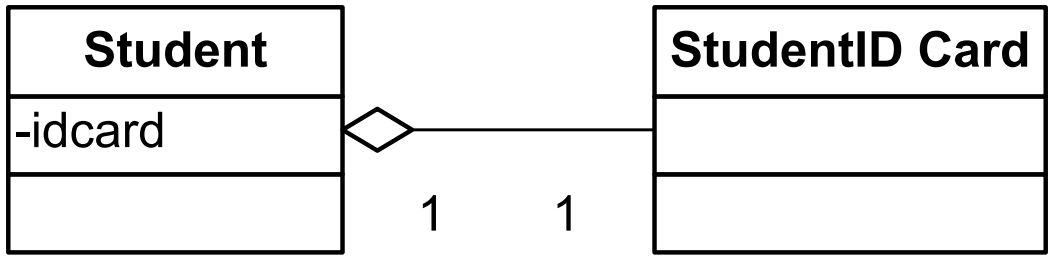
Association relationships

association: an instance of one class must know about the other in order to do its work

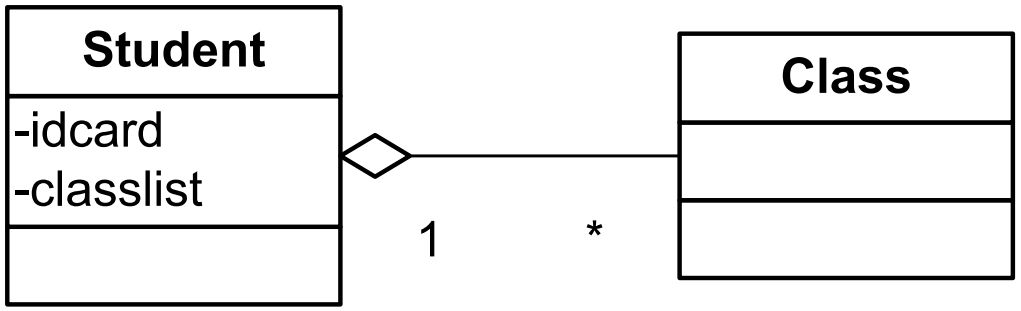
1. multiplicity
   - * ⇒ 0, 1, or more
   - 1 ⇒ 1 exactly
   - 2..4 ⇒ between 2 and 4, inclusive
   - 3..* ⇒ 3 or more
2. name (what relationship the objects have)
3. navigability (direction of a query, represented by a line between the objects no arrow if communication flows both ways)

Multiplicity of associations

- **one-to-one**
  - each student must carry exactly one ID card



- **one-to-many**
  - each student may have many classes



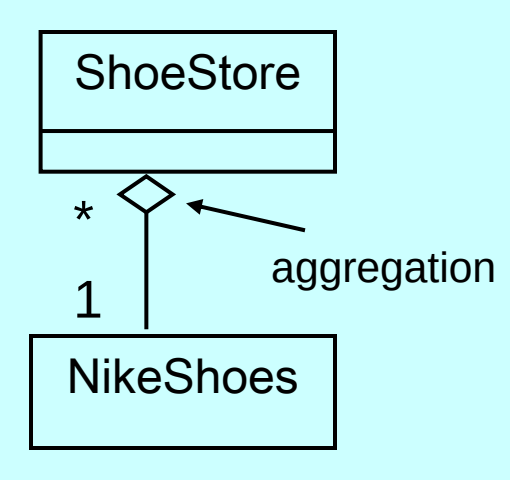
Back to our example

- Add a square-list class and associate it with a square

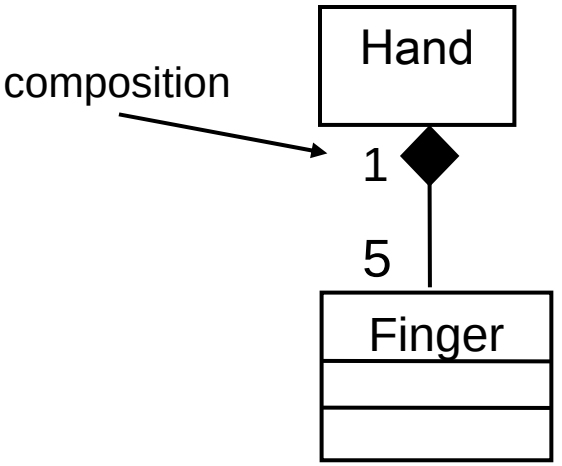
- Identify the multiplicity on the ends of the association
  - * $\Rightarrow$ 0, 1, or more
  - 1 $\Rightarrow$ 1 exactly
  - 2..4 $\Rightarrow$ between 2 and 4, inclusive
  - 3..* $\Rightarrow$ 3 or more

Association types

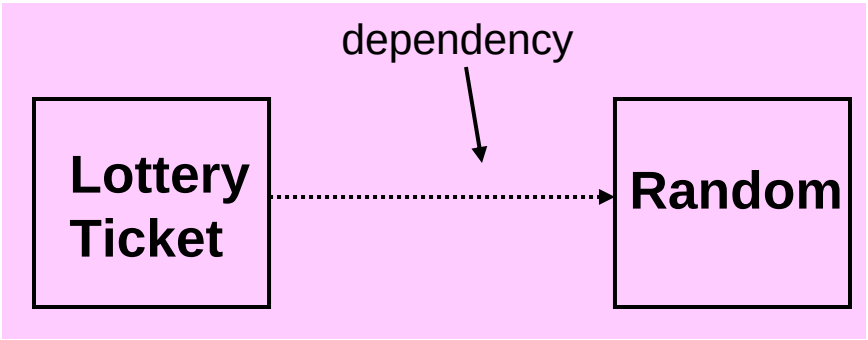
- **aggregation**: “contains”
  - symbolized by a clear white diamond pointing to *the class containing the other class*



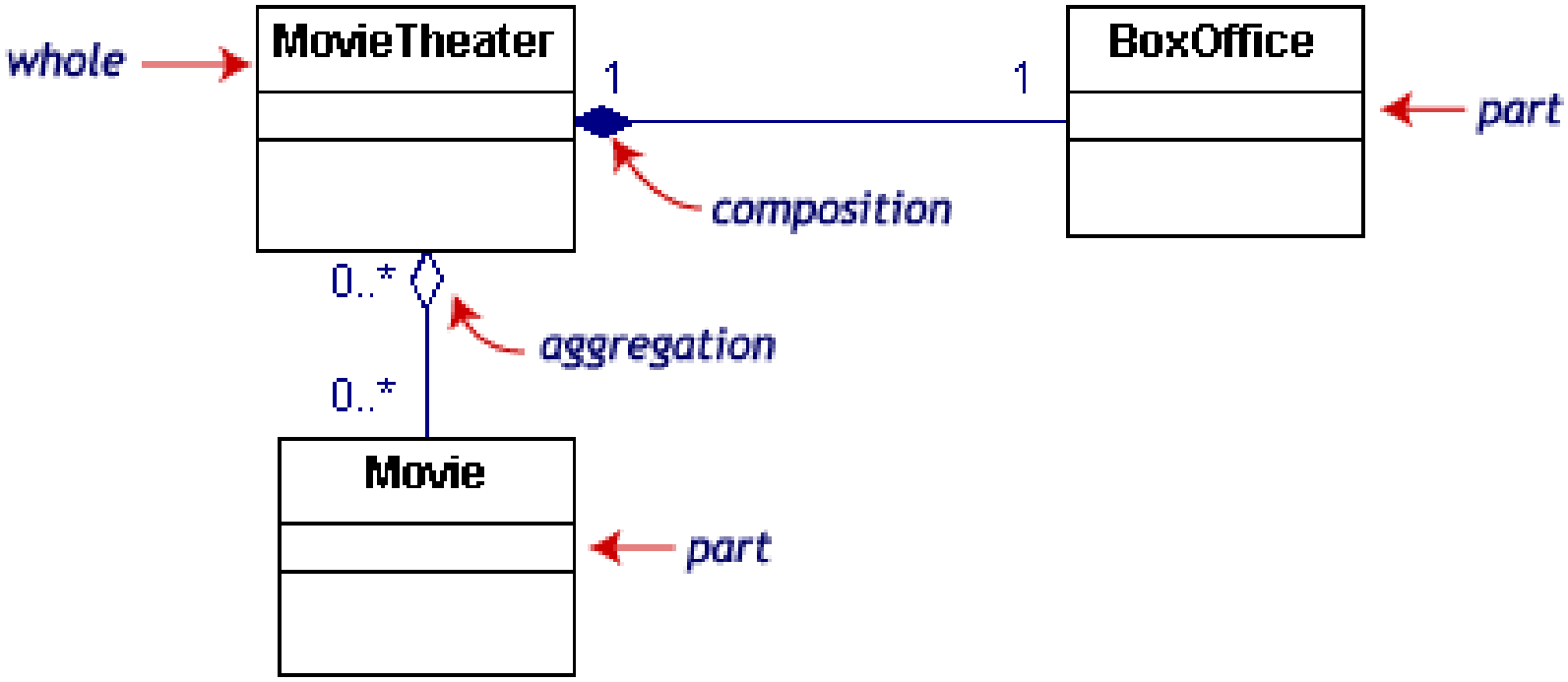
- **composition**: “contained for only this purpose”
  - *stronger version of aggregation*
  - the parts live and die with the whole
  - symbolized by a black diamond pointing to the containing class



- **dependency**: "uses temporarily"
  - symbolized by dotted line

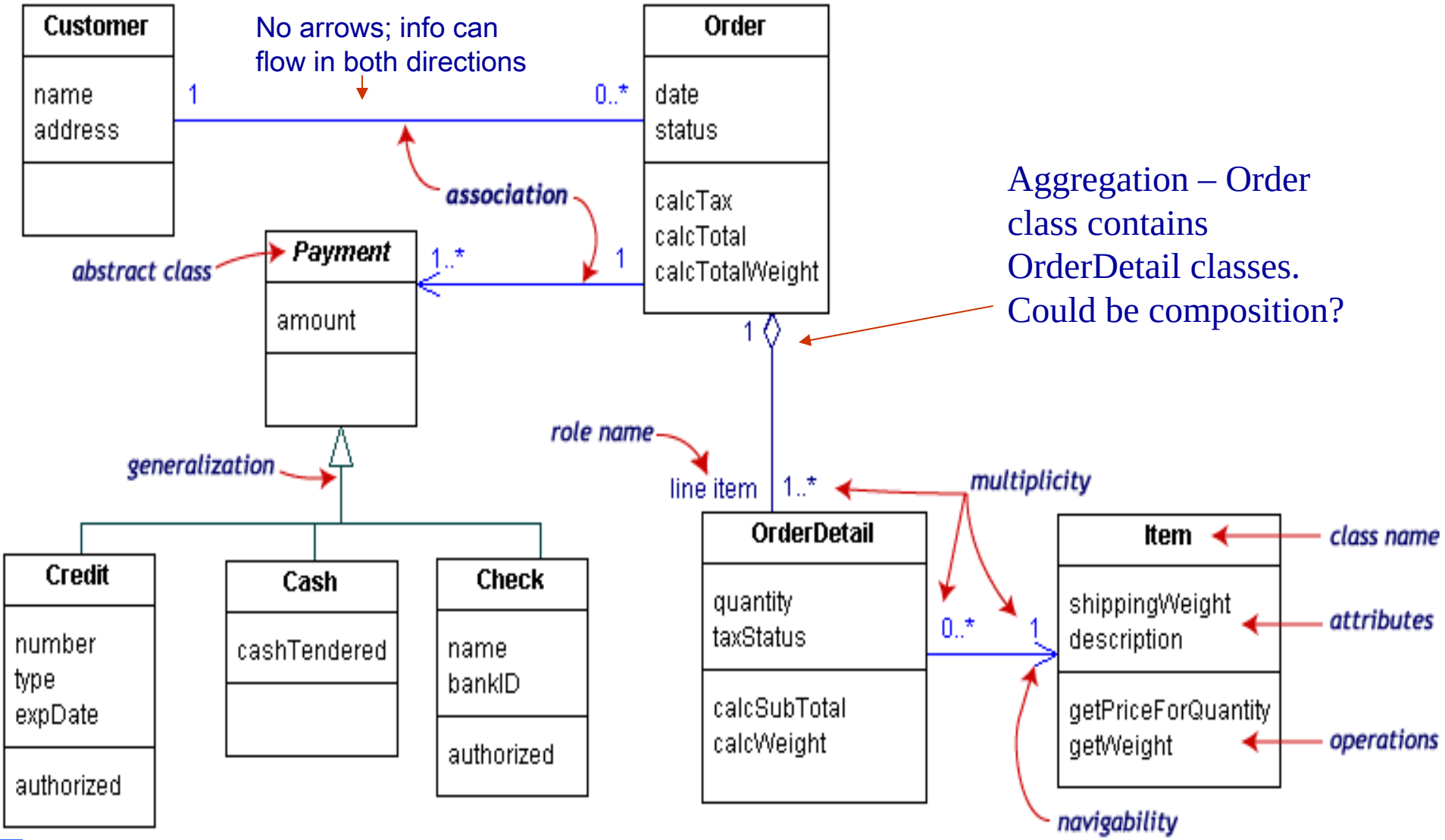


Composition/aggregation example

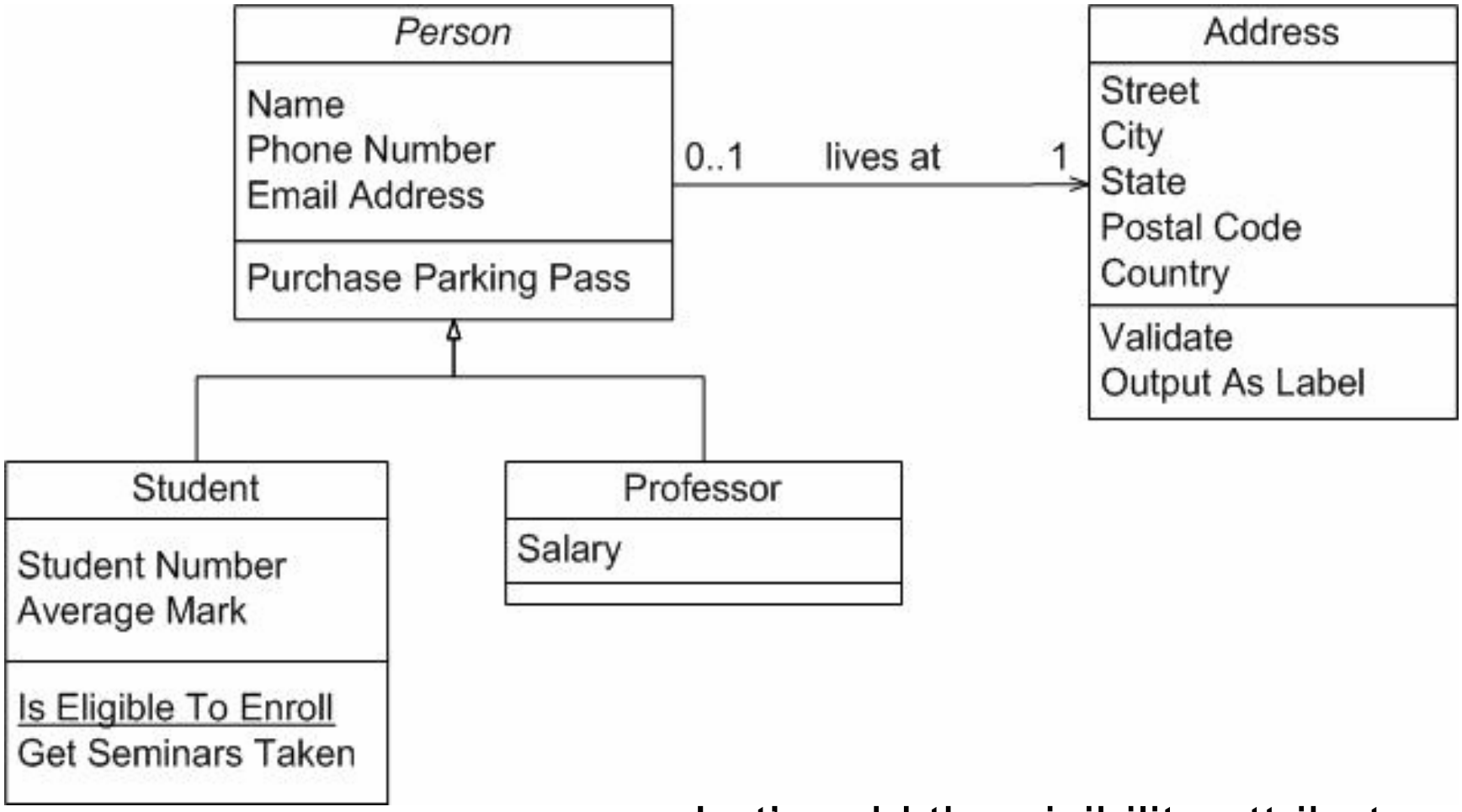


- If the movie theatre goes away
  - so does the box office => composition
  - but movies may still exist => aggregation

Class diagram example

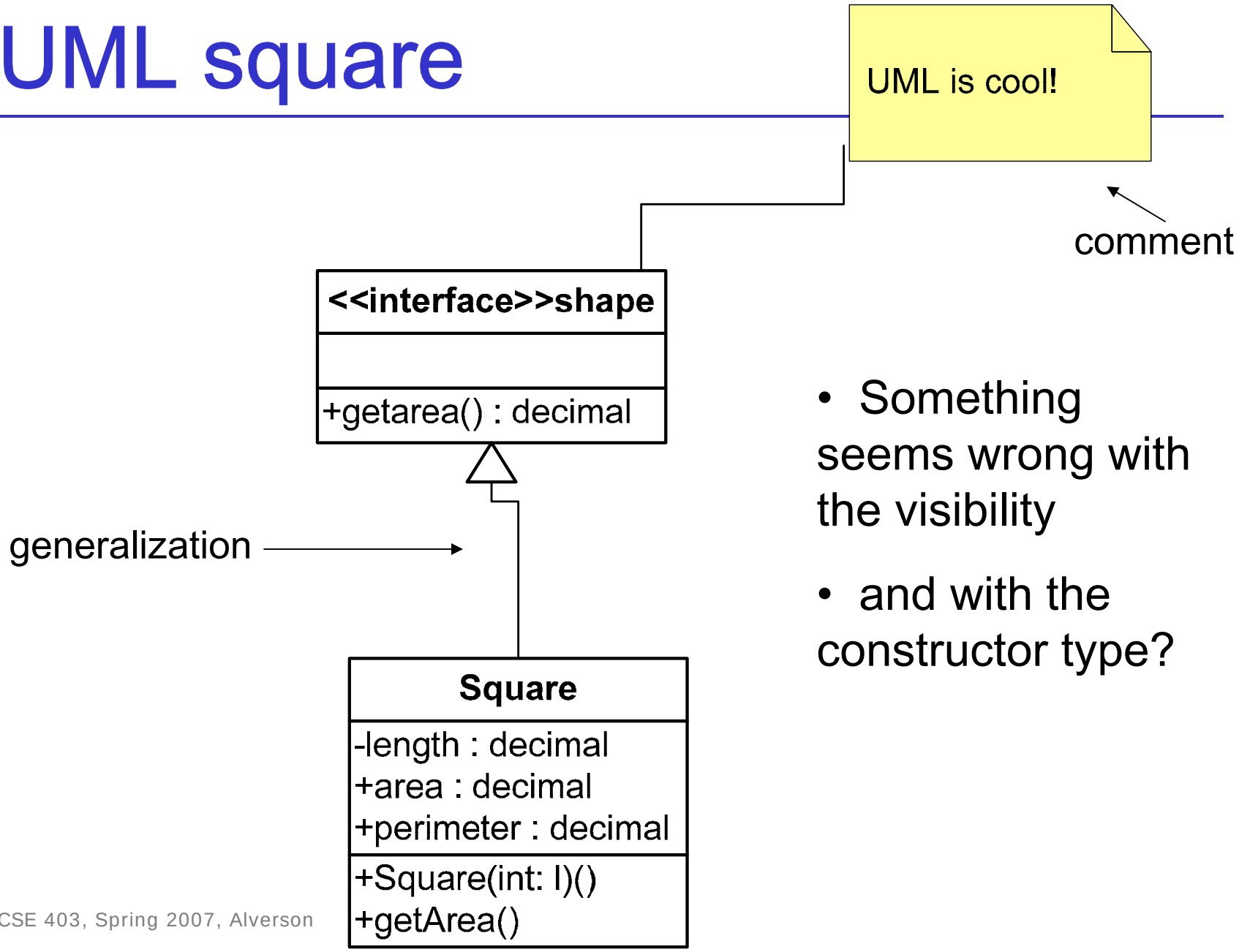


UML example #2



Let's add the visibility attributes

UML square



- Something seems wrong with the visibility
- and with the constructor type?

Shall we try a bigger design?

Crystal Ski School

- Crystal wants to implement a new ski school booking/payment mechanism
- They want to allow students to request particular instructors, and see their availability
- They want to allow instructors to retrieve their daily schedule
- They want to allow invoicing on a monthly basis

What would be our main…

- Objects

- Connections