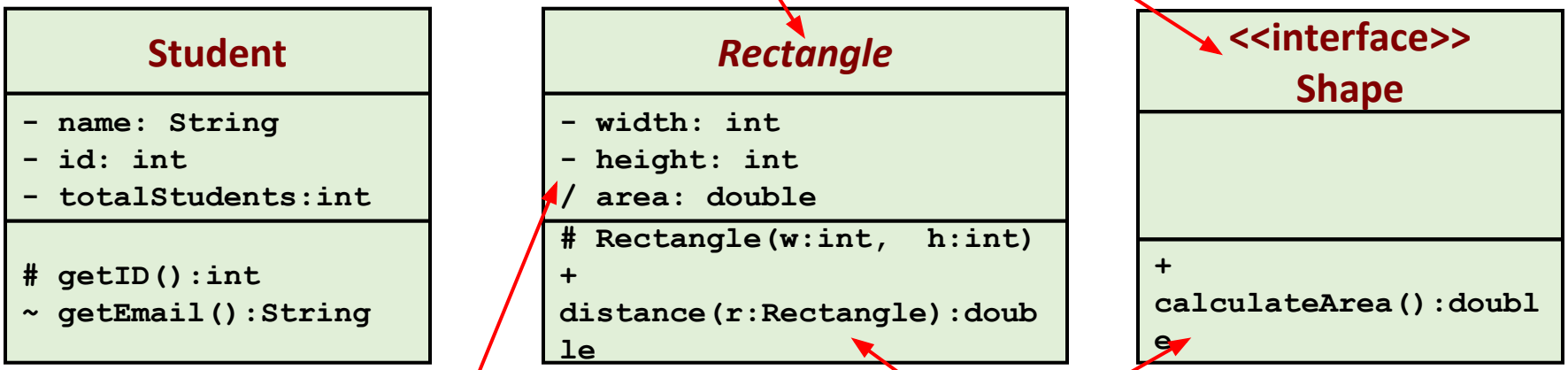
What is a UML class diagram?

- A UML class diagram is a picture of the classes in an OO system
  - their fields and methods
  - connections between the classes that interact or inherit from each other
- Not represented in a UML class diagram:
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

Diagram of a single class

Class name on top
- write «interface» on top of interfaces' names
- use *italics* for an abstract class name



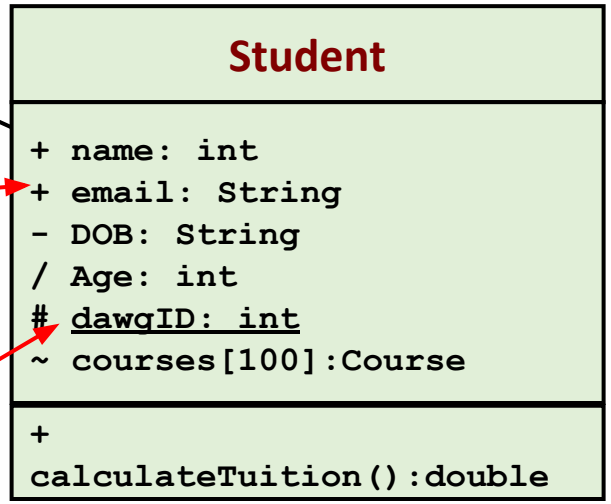
Attributes (optional)
- In the middle

Operations/ methods (optional)
- may omit trivial (get/set) methods
- but don't omit any methods from an interface!
- should not include inherited methods

Class attributes (fields, instance variables)

visibility name : type [count] = default_value

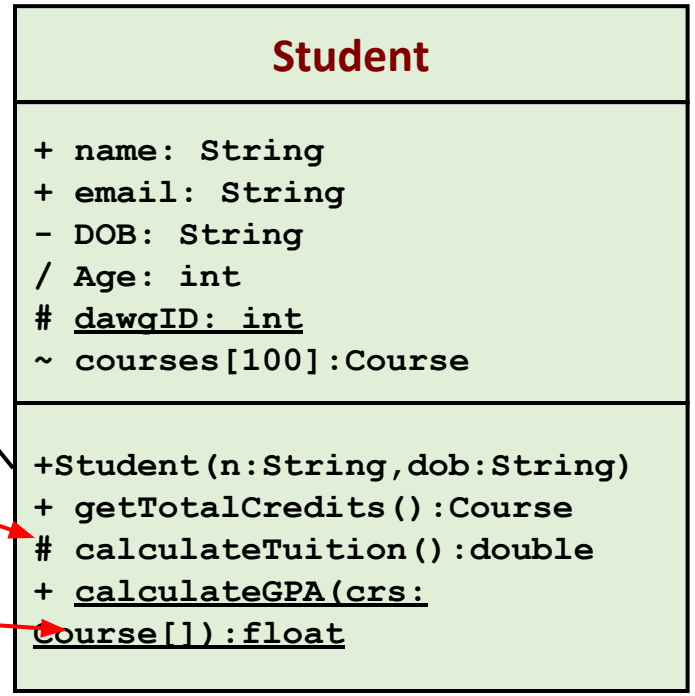
- Visibility
  - \+ public
  - # protected
  - - private
  - ~ package (default)
  - / derived
- <u>underline static attributes</u>



Class operations / methods

visibility name(parameters) : return_type

- Visibility
  - \+ public
  - # protected
  - - private
  - ~ package (default)
- underline static methods
- Parameters listed as name:type
- Omit return_type on constructors and when return type is void

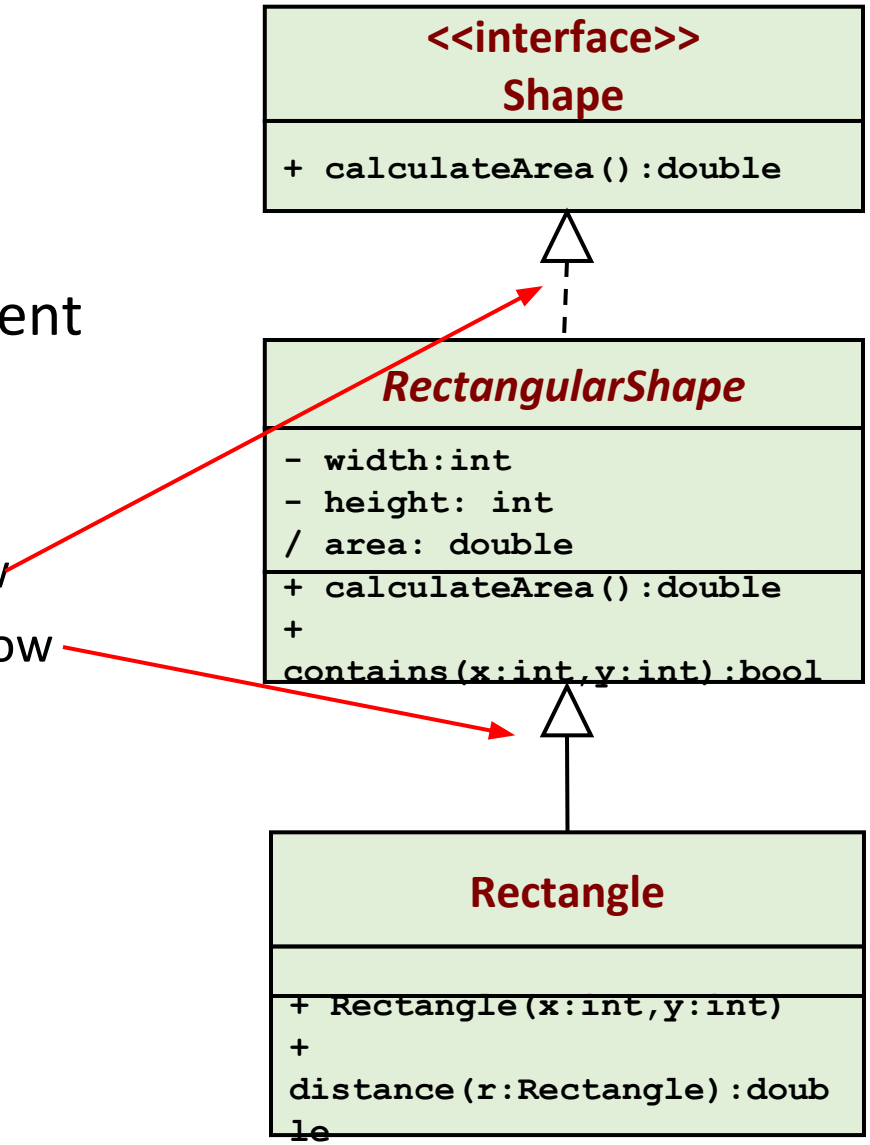


Relationships between class

- **Generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation
- A**ssociation**: a usage relationship
  - dependency
  - aggregation
  - composition

Generalization relationships

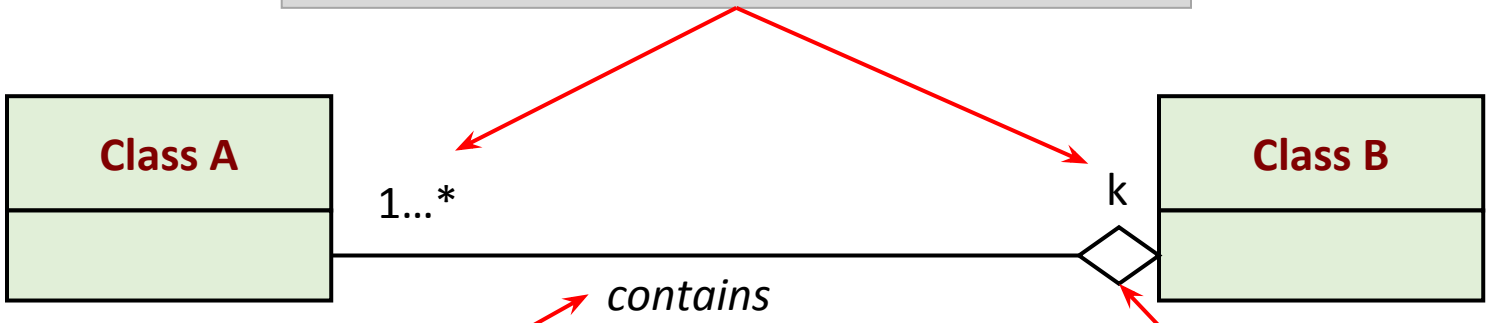
- Hierarchies drawn top-down
- Arrows point upward to parent
- Line/arrow styles indicate if parent is a(n):
  - **class**: solid line, black arrow
  - **interface**: dashed line, white arrow
  - **abstract class**: solid line, white arrow



Association (usage) relationships

Multiplicity (how many are used)

- * (zero or more)
- 1 (exactly one)
- 2..4 (between 2 and 4, inclusive)
- 3..* (3 or more, * may be omitted)



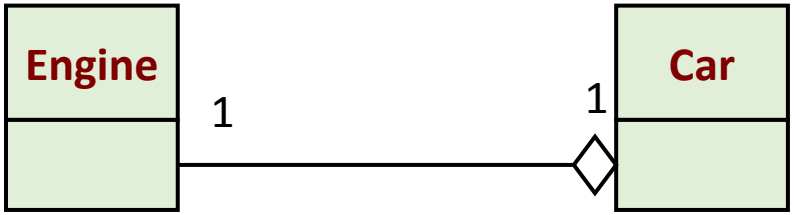
Name (what relationship the objects have)

Navigability (direction)

Association multiplicities

One to one

- Each car has exactly one engine
- Each engine belongs to exactly one car



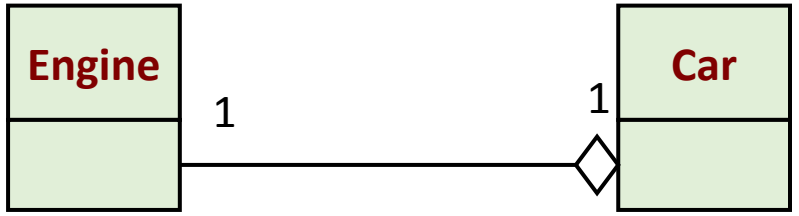
One to many

- Each book has many pages
- Each page belongs to exactly one book

Association types

Aggregation: “is part of”

- symbolized by a clear white diamond

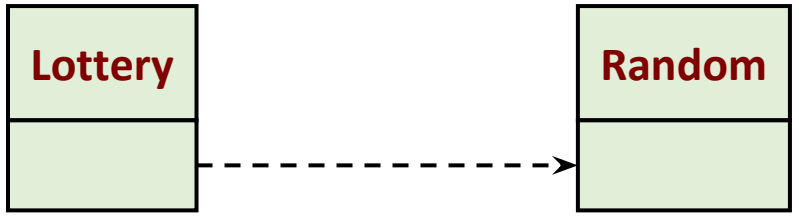


Composition: “is entirely made of”

- stronger version of aggregation
- the parts live and die with the whole
- symbolized by a black diamond

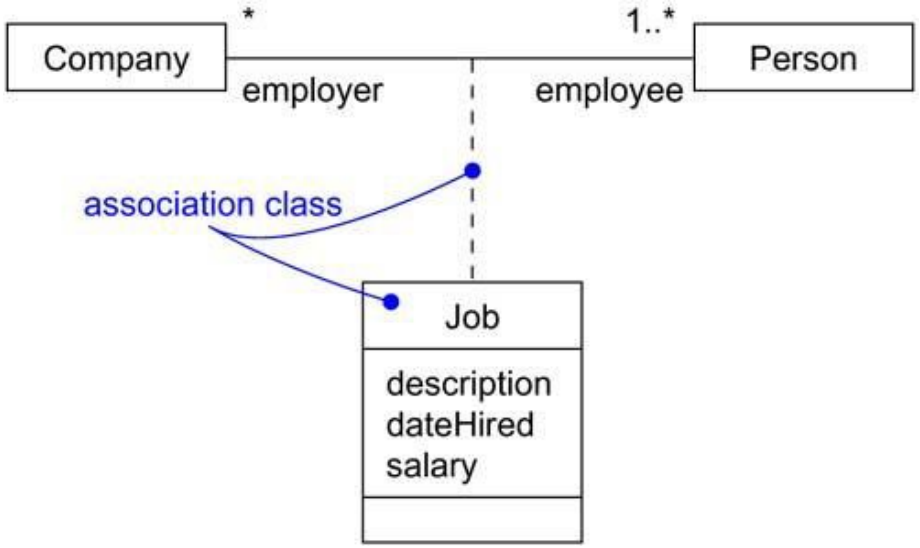
Dependency: “uses temporarily”

- symbolized by dotted line
- often is an implementation detail, not an intrinsic part of the object's state

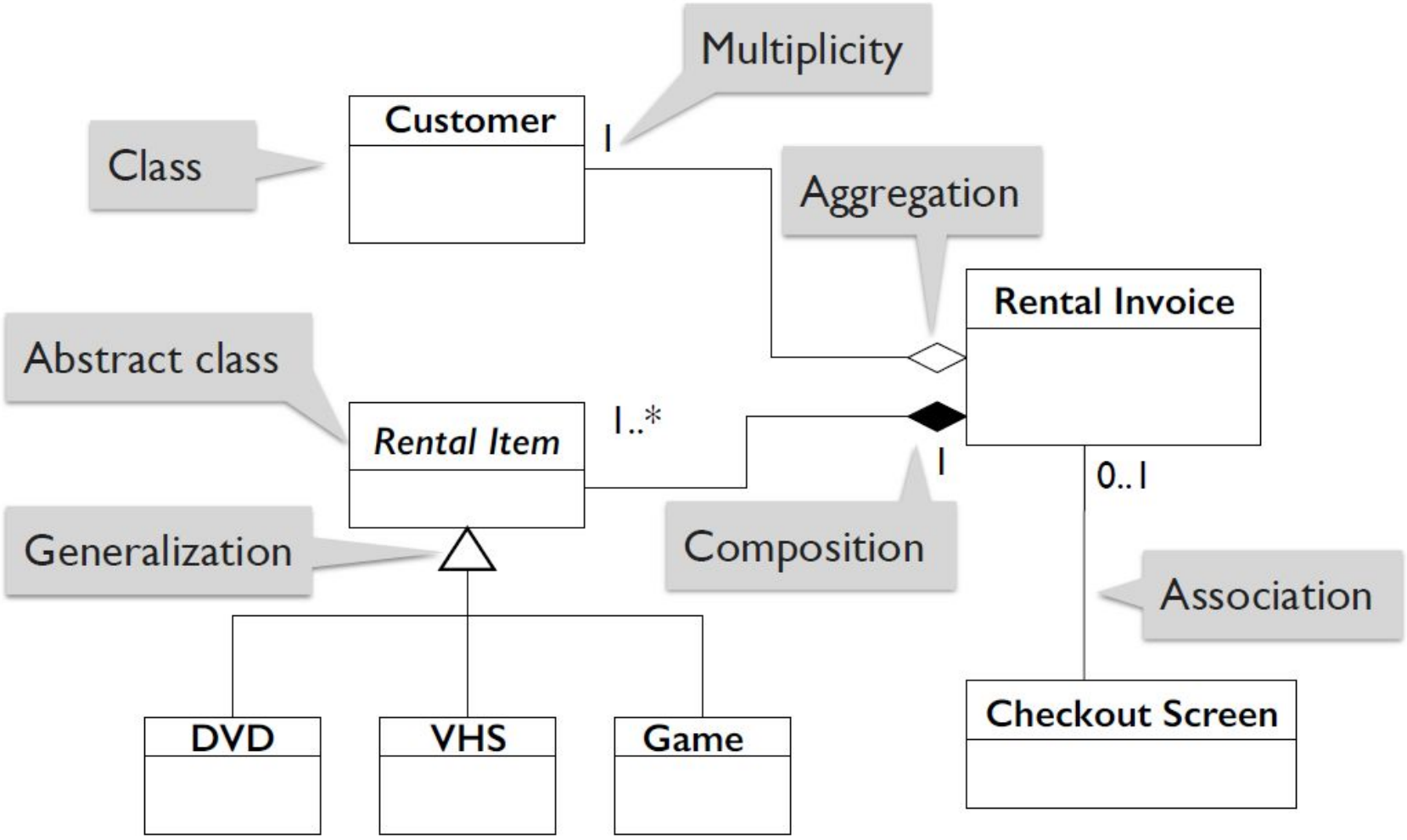


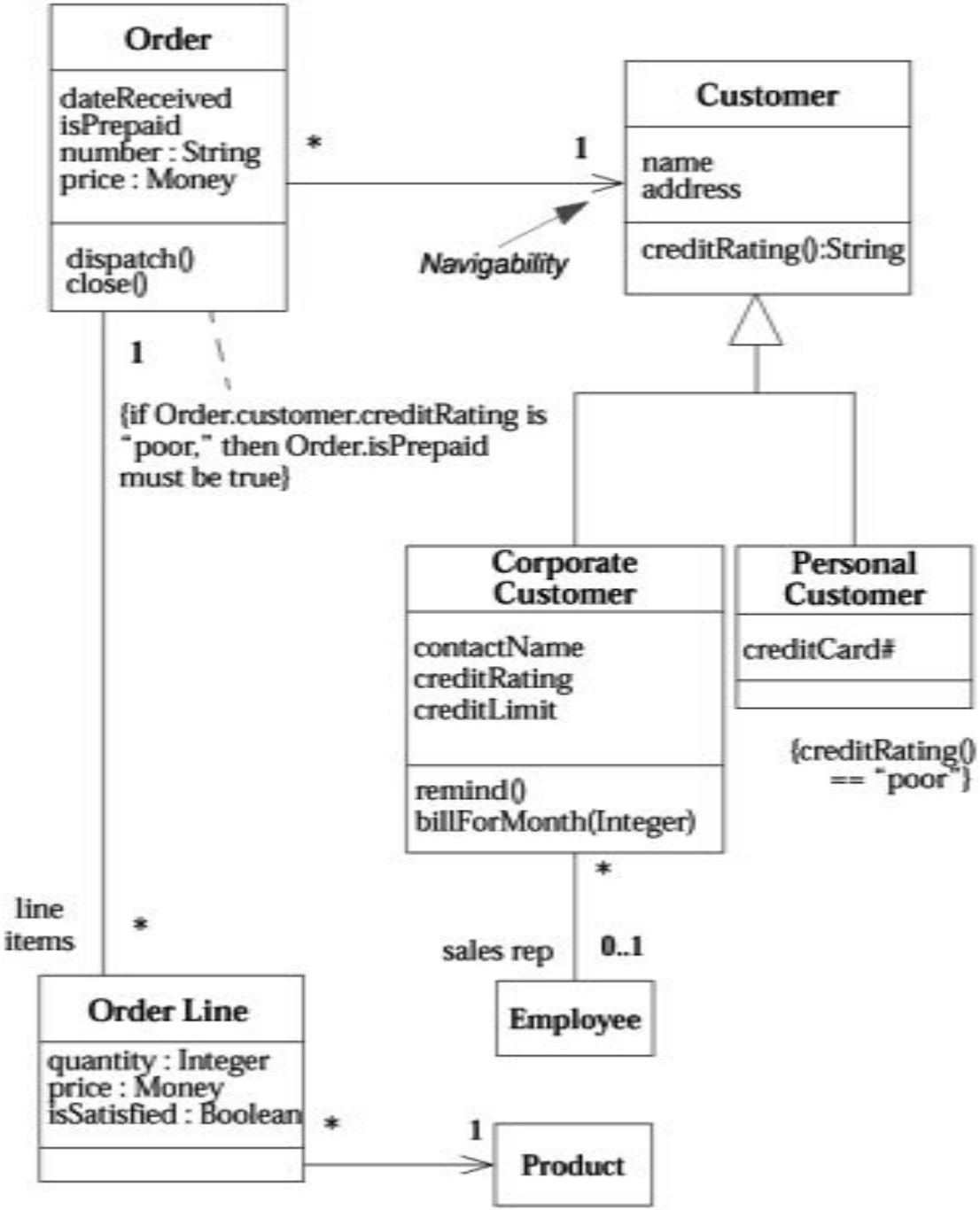
Association Class

- In an association between two classes, the association itself might have properties.
- For example, in an employer/employee relationship between a Company and a Person, there is a Job that represents the properties of that relationship that apply to exactly one pairing of the Person and Company.
- It wouldn't be appropriate to model this situation with a Company to Job association together with a Job to Person association. That wouldn't tie a specific instance of the Job to the specific pairing of Company and Person.

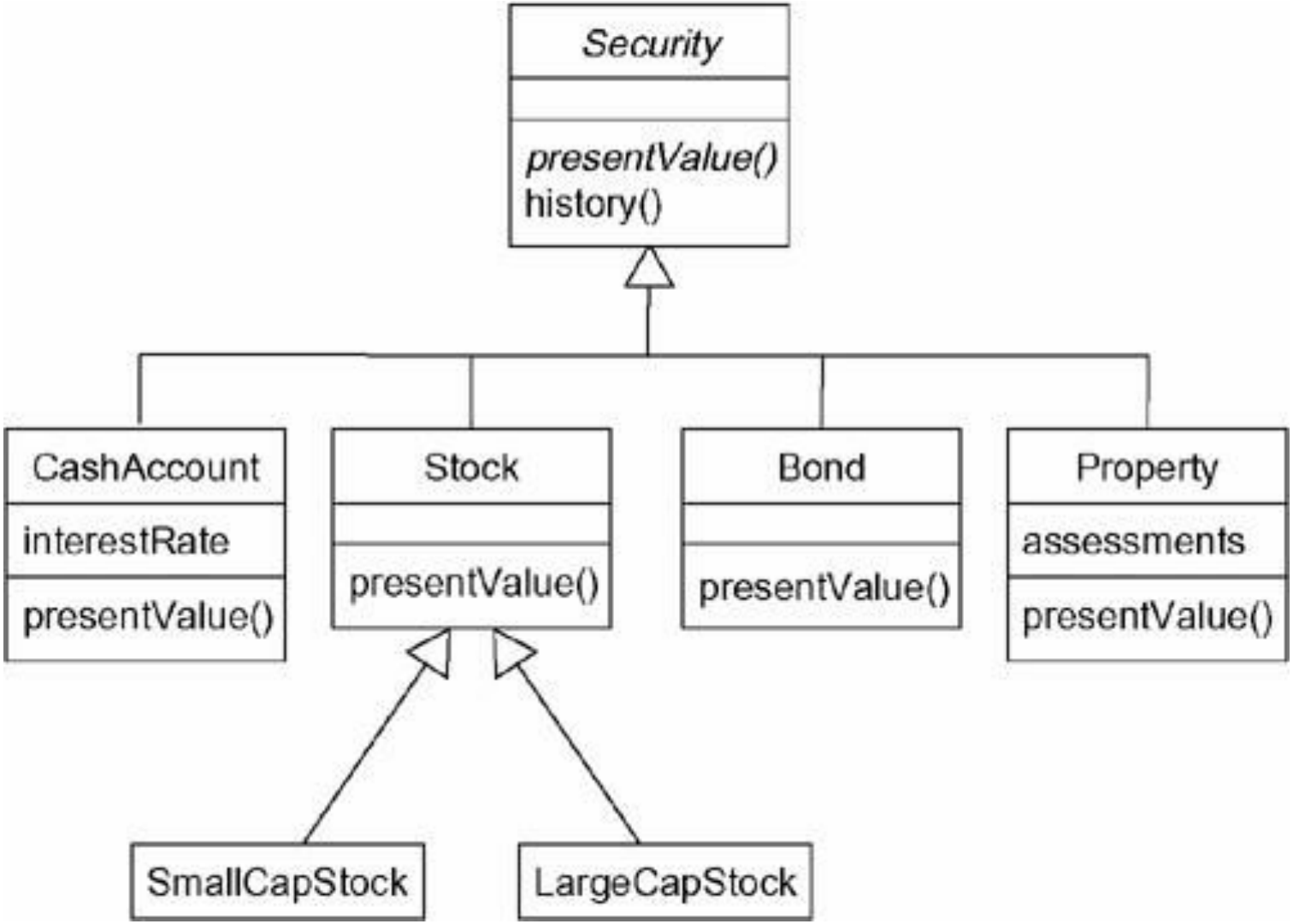


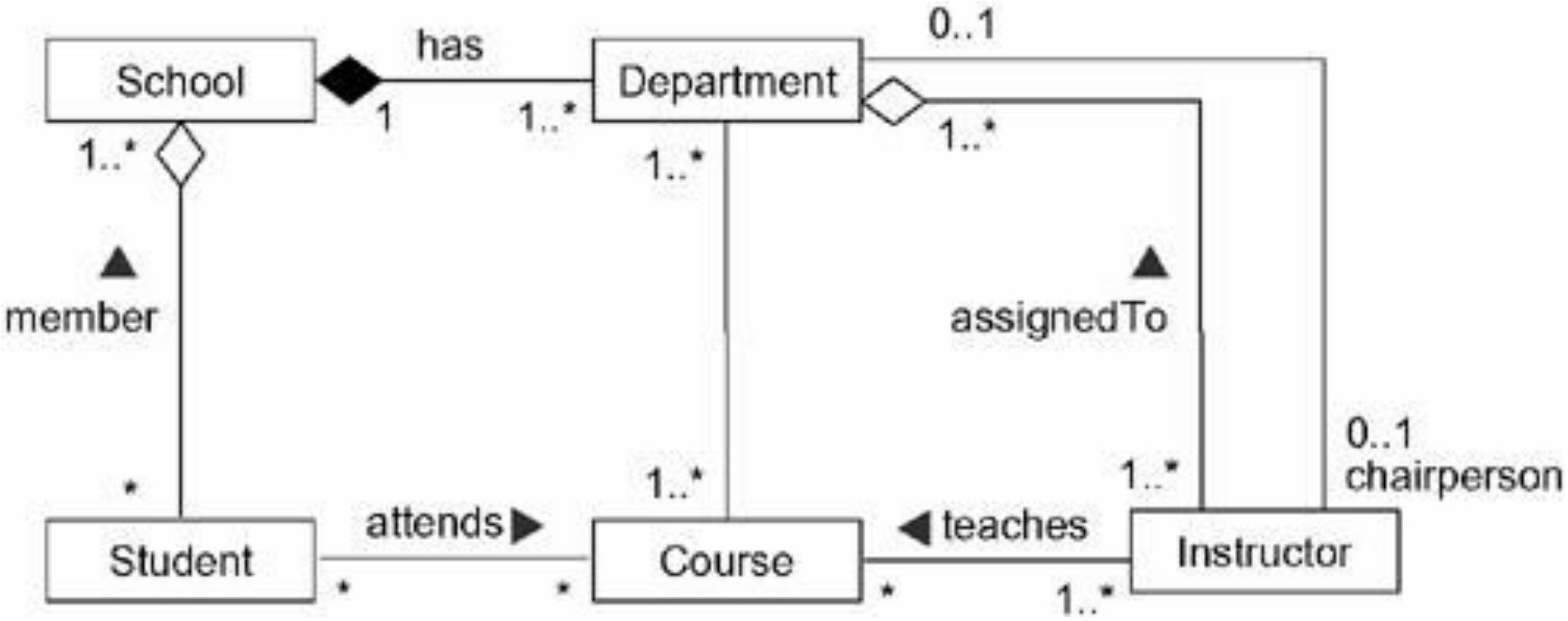
Example: Video rental store




Order
dateReceived
isPrepaid
number : String
price : Money
dispatch()
close()
*
1
Customer
name
address
creditRating():String
Navigability
1
{if Order.customer.creditRating is "poor," then Order.isPrepaid must be true}
Corporate Customer
contactName
creditRating
creditLimit
remind()
billForMonth(Integer)
Personal Customer
creditCard#
{creditRating() == "poor"}
line items
*
*
sales rep
0..1
Order Line
quantity : Integer
price : Money
isSatisfied : Boolean
*
1
Employee
Product

More Example of Inheritance




School
has
Department
0..1
1
1..*
1..*
1..*
1..*
member
assignedTo
*
1..*
1..*
0..1
chairperson
Student
attends
Course
teaches
Instructor
*
*
*
1..*

Multiple Inheritance