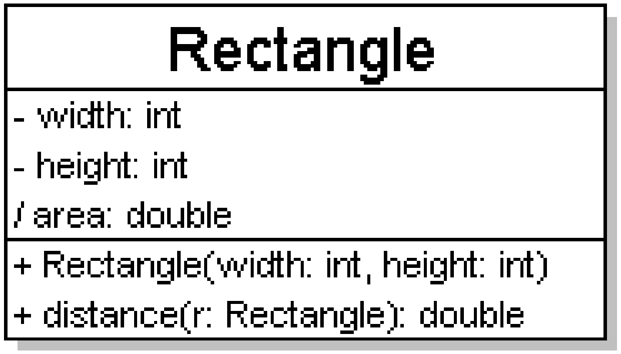
# UML class diagrams

- What is a UML class diagram?
  - **UML class diagram**: a picture of
    - the classes in an OO system
    - their fields and methods
    - connections between the classes
      - that interact or inherit from each other
- What are some things that are <u>not</u> 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 one class

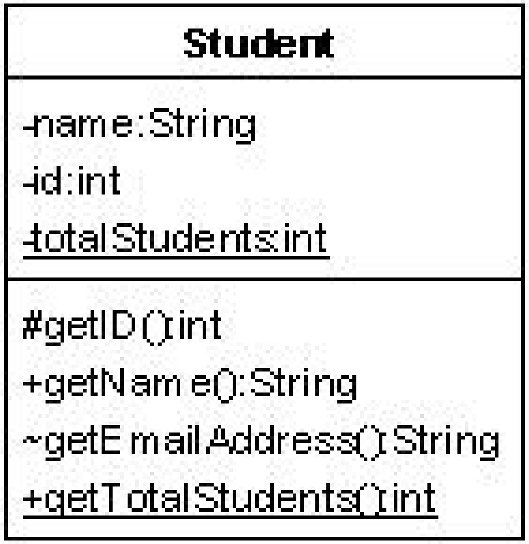
- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double



Student

-name:String
-id:int
-totalStudents:int

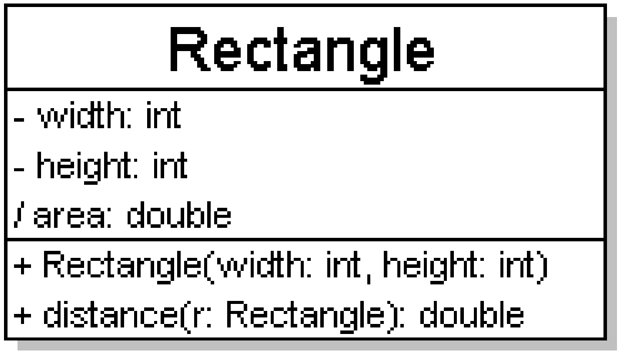
#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

# Class attributes

- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:

| | |
|---|---|
| + | public |
| # | protected |
| - | private |
| ~ | package (default) |
| / | derived |

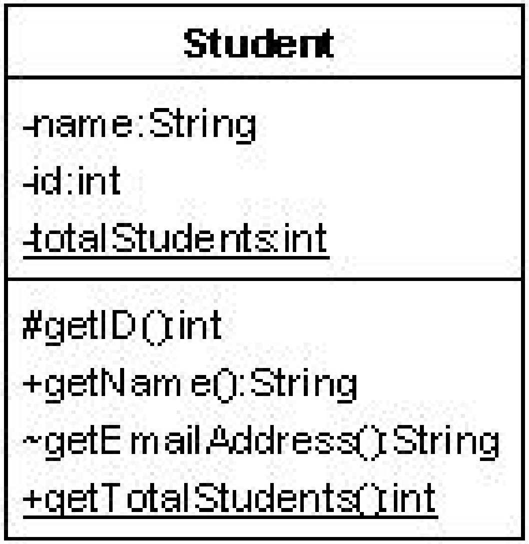
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
  - attribute example:
    - balance : double = 0.00



**Rectangle**

- width: int
- height: int
/ area: double

+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double



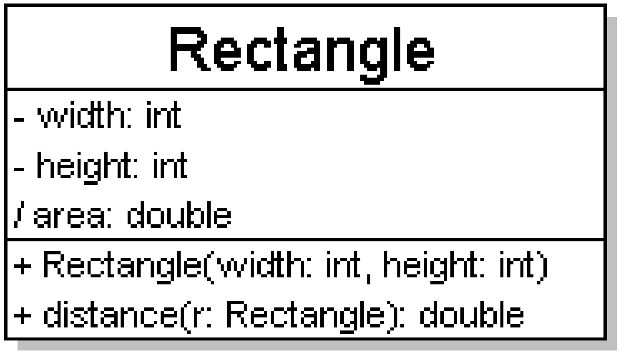
**Student**

-name:String
-id:int
-totalStudents:int

#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

# Class operations / methods

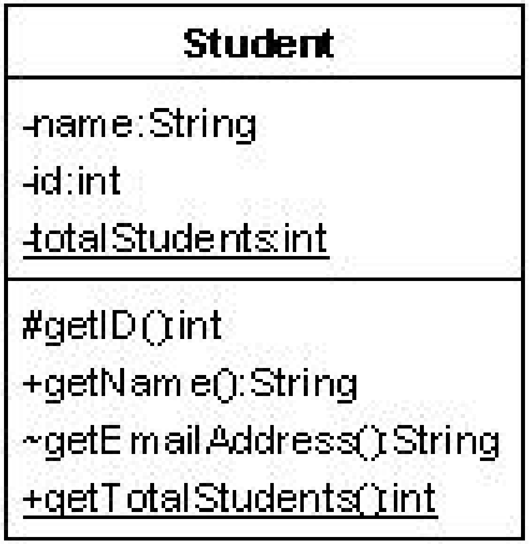
- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility: + public<br># protected<br>- private<br>~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    + distance(p1: Point, p2: Point): double



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double



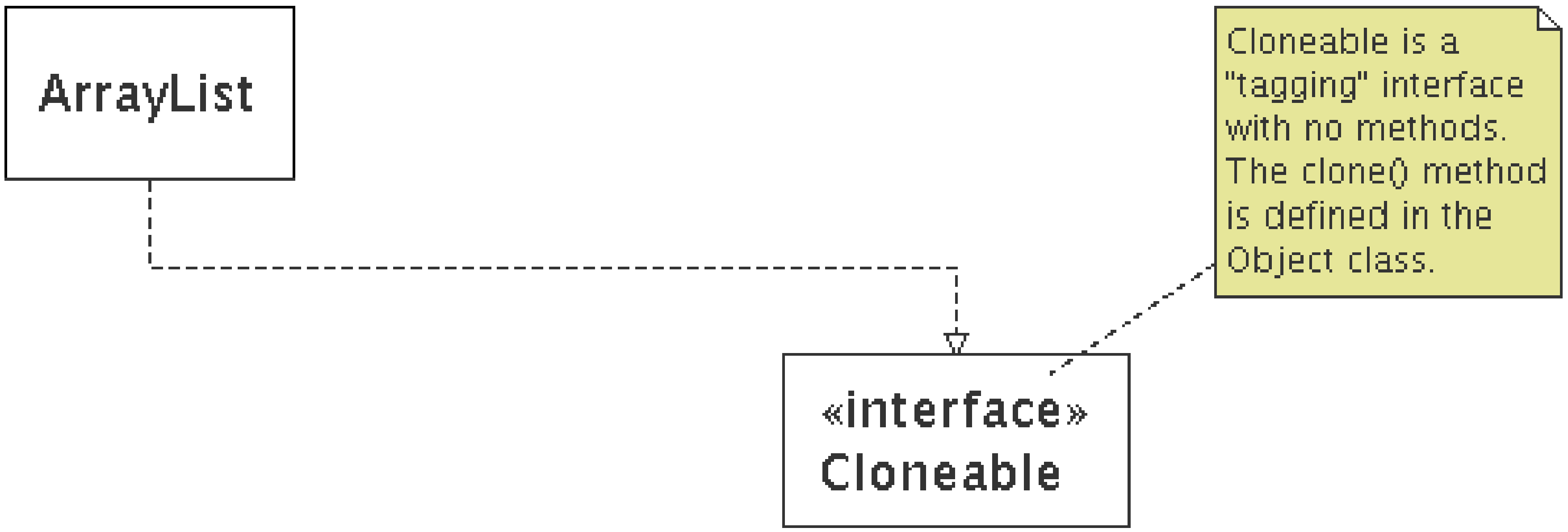
Student

-name:String
-id:int
-totalStudents:int

#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

# Comments

- represented as a folded note, attached to the appropriate class/method/etc by a dashed line


ArrayList
«interface»
Cloneable
Cloneable is a "tagging" interface with no methods. The clone() method is defined in the Object class.

# Relationships btwn. classes

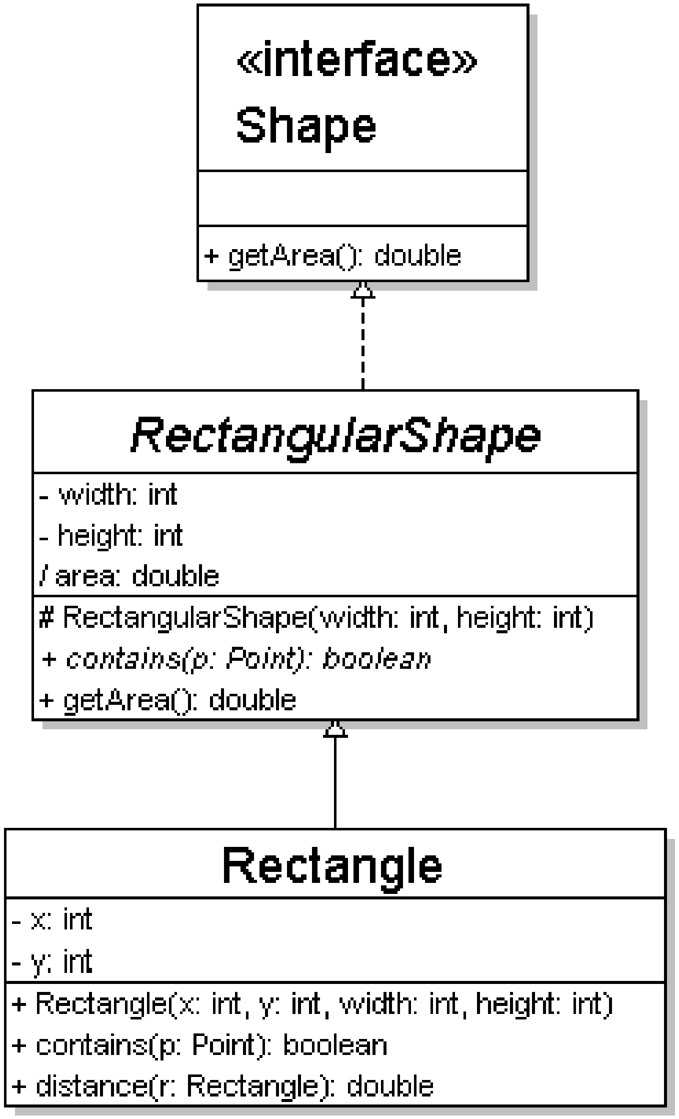
- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

# Generalization relationships

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

  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent

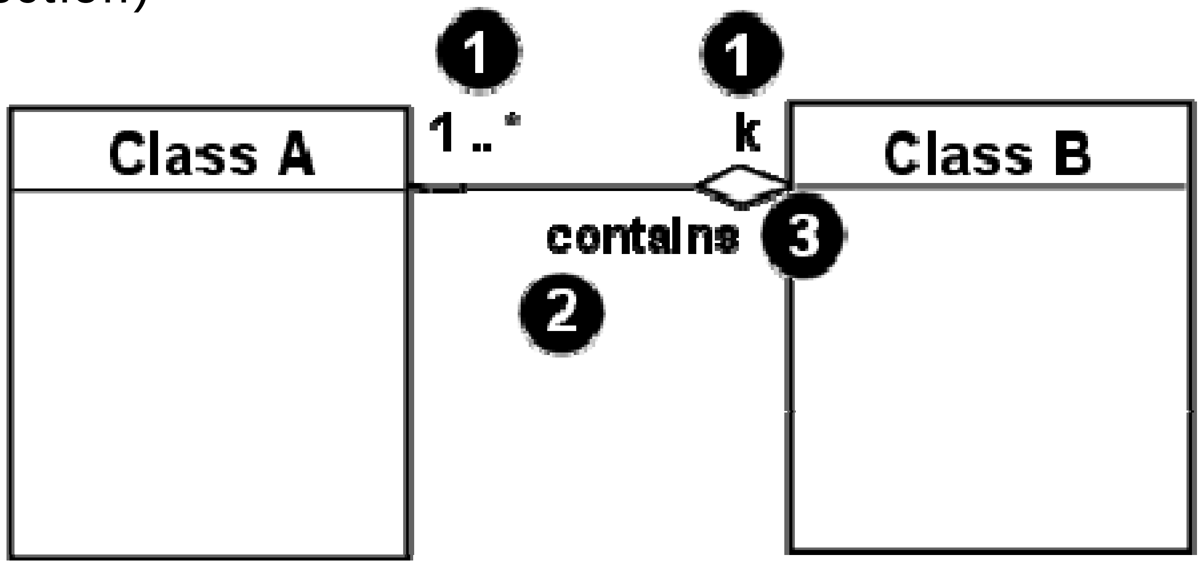


«interface»
Shape
+ getArea(): double
RectangularShape
- width: int
- height: int
/ area: double
# RectangularShape(width: int, height: int)
+ contains(p: Point): boolean
+ getArea(): double
Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double

# Associational relationships

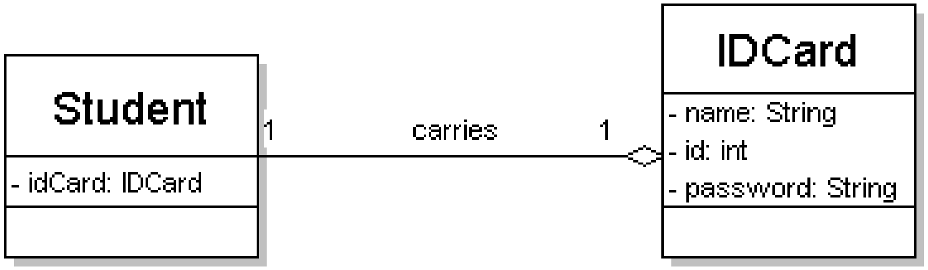
- associational (usage) relationships

1. multiplicity (how many are used)
    - * ⇒ 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)

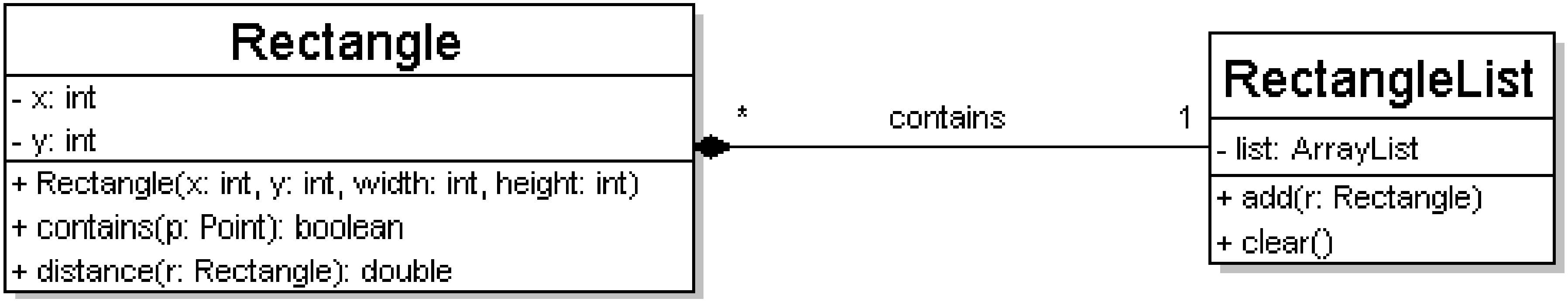


# Multiplicity of associations

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

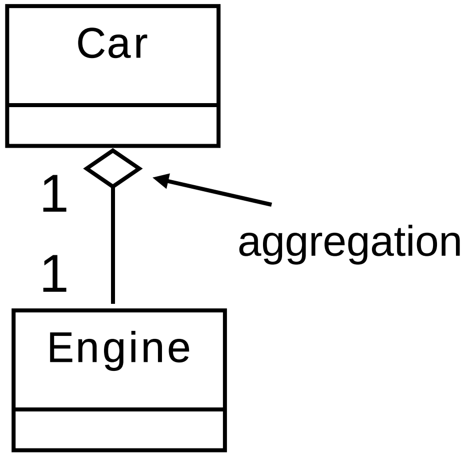


- one-to-many
  - one rectangle list can contain many rectangles

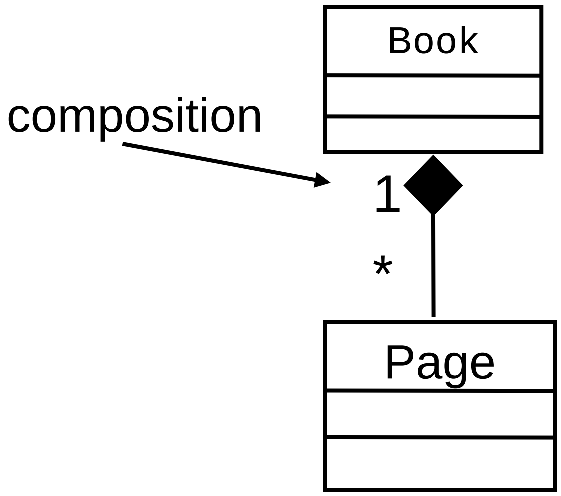

Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double
*
contains
1
RectangleList
- list: ArrayList
+ add(r: Rectangle)
+ clear()

# Association types

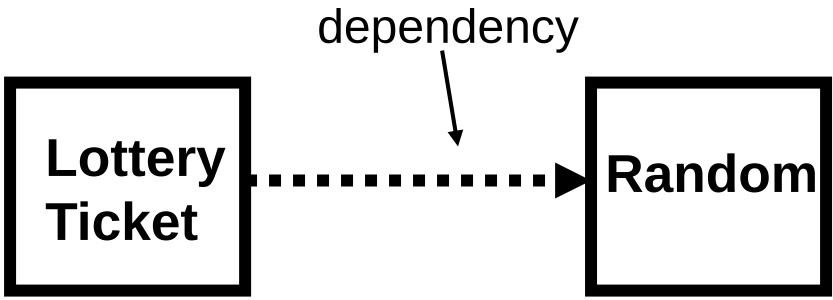
- **aggregation**: "is part of"
  - symbolized by a clear white diamond



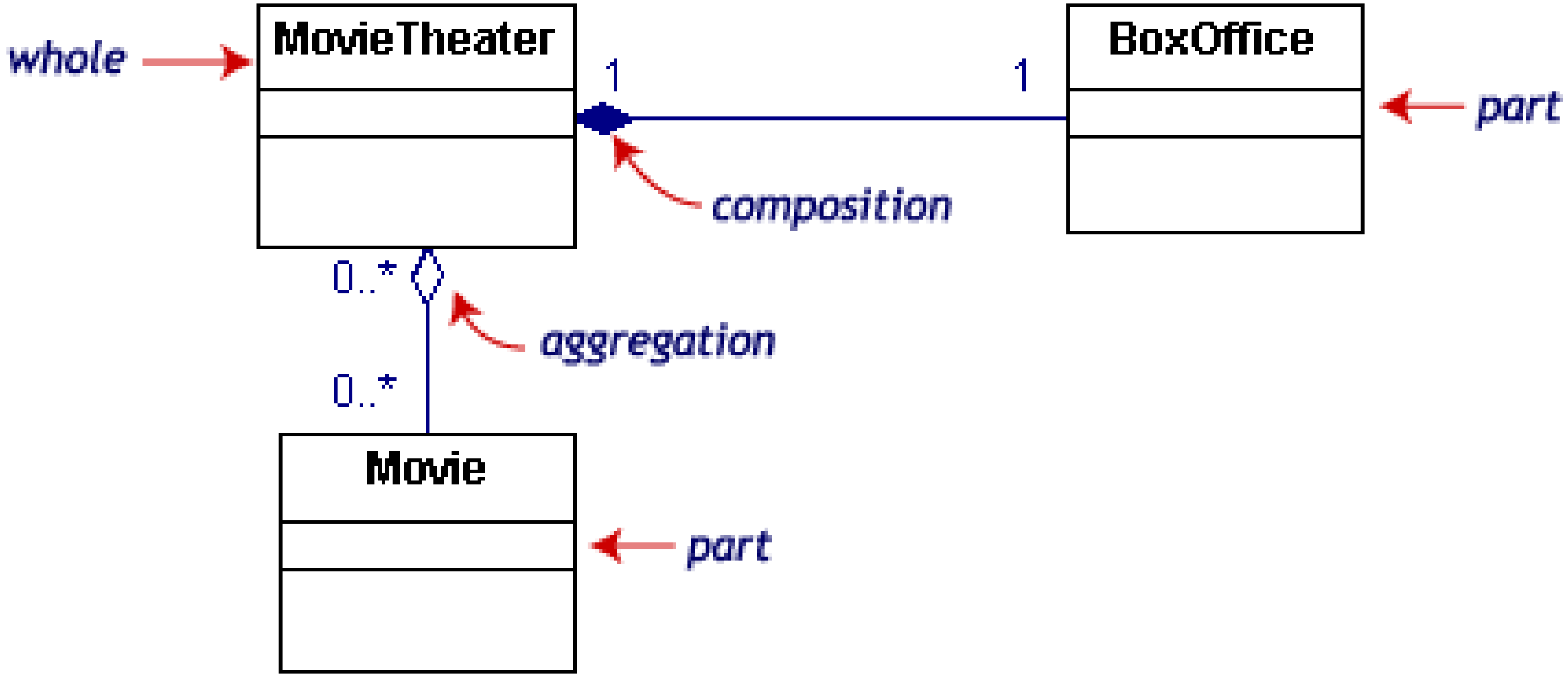
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond



- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



# Composition/aggregation example

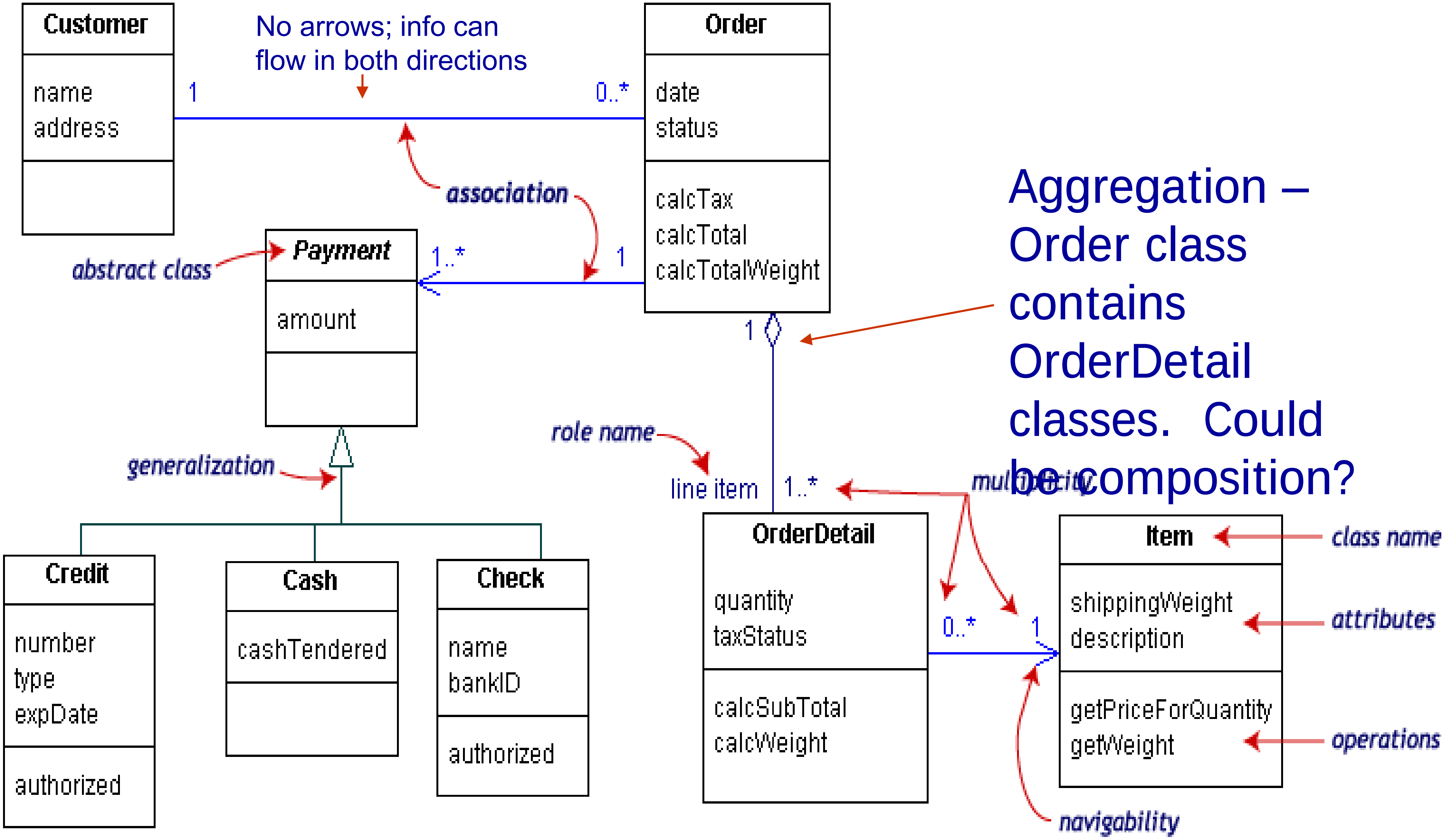


If the movie theatre goes away

so does the box office => composition

but movies may still exist => aggregation

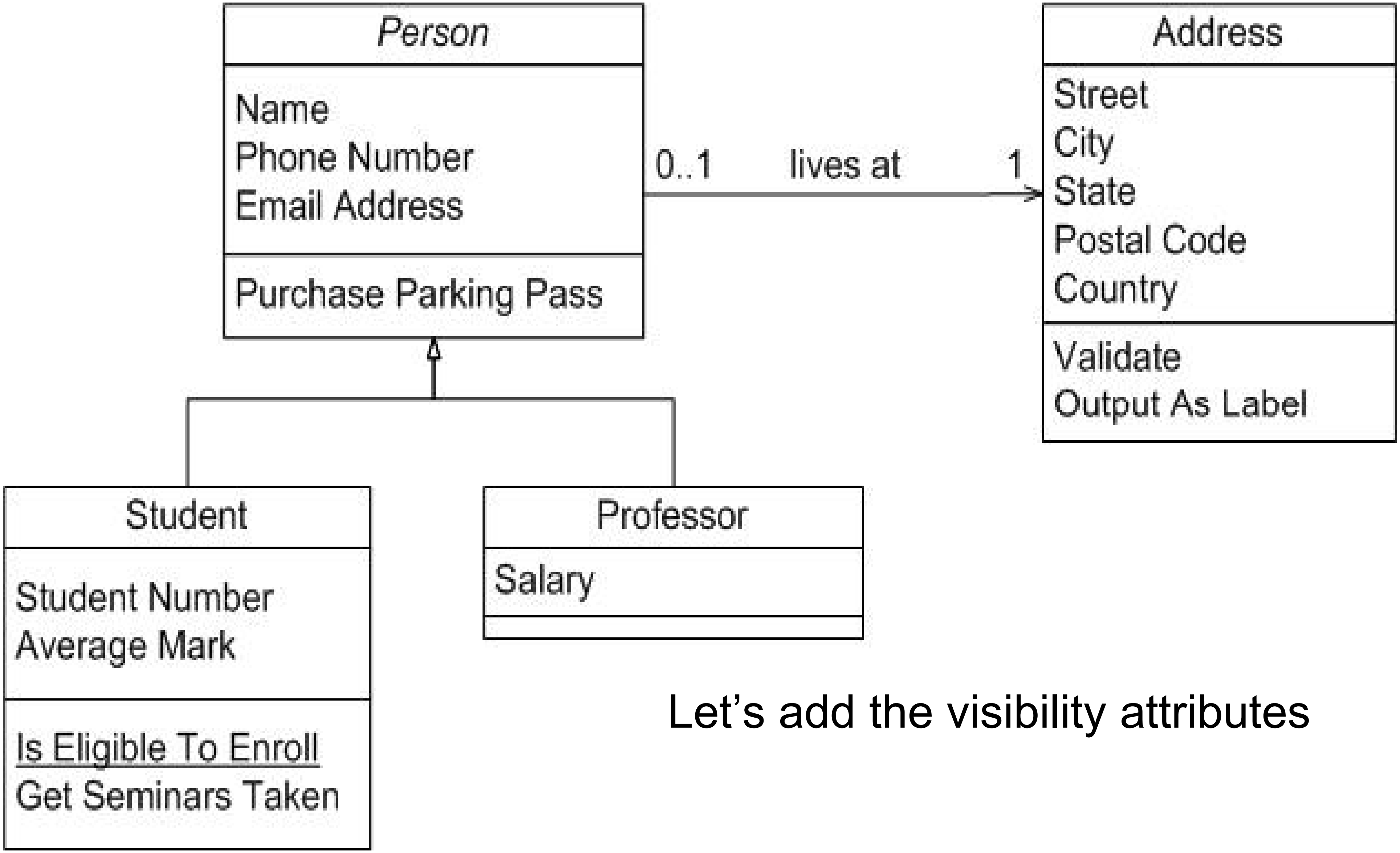
# Class diagram example



No arrows; info can flow in both directions

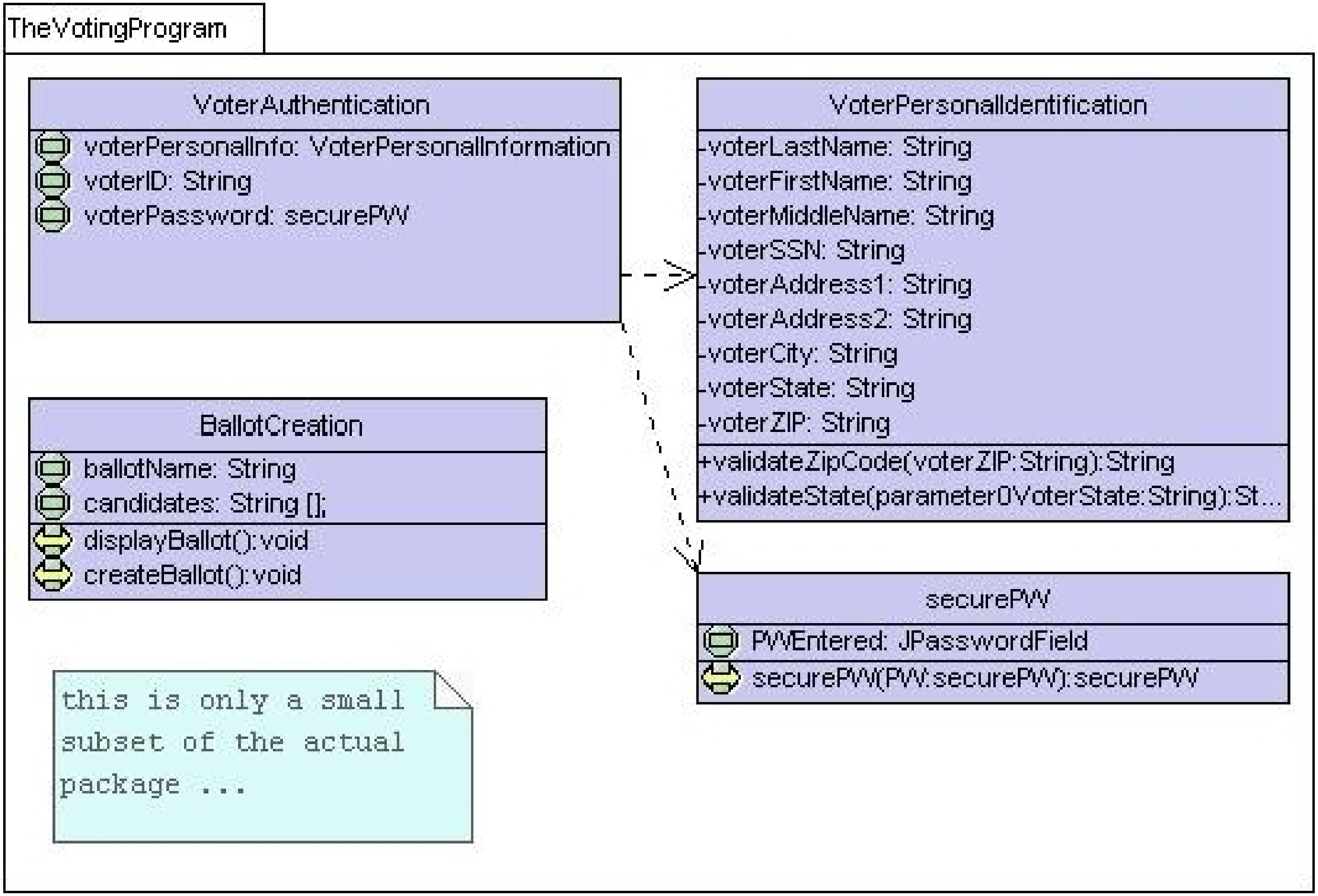
Aggregation – Order class contains OrderDetail classes. Could be composition?

# UML example:  people



Let’s add the visibility attributes

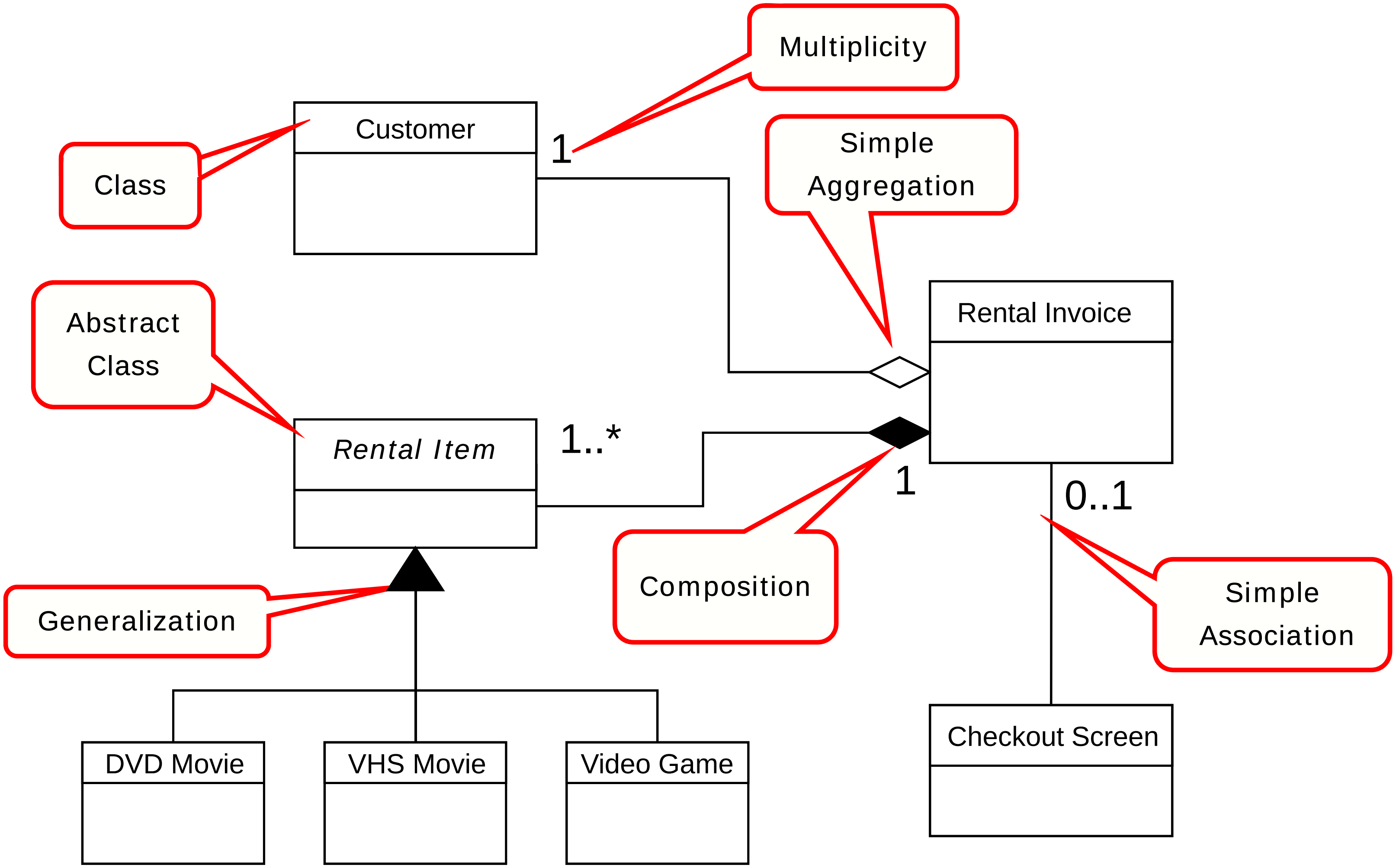
# Class diagram: voters



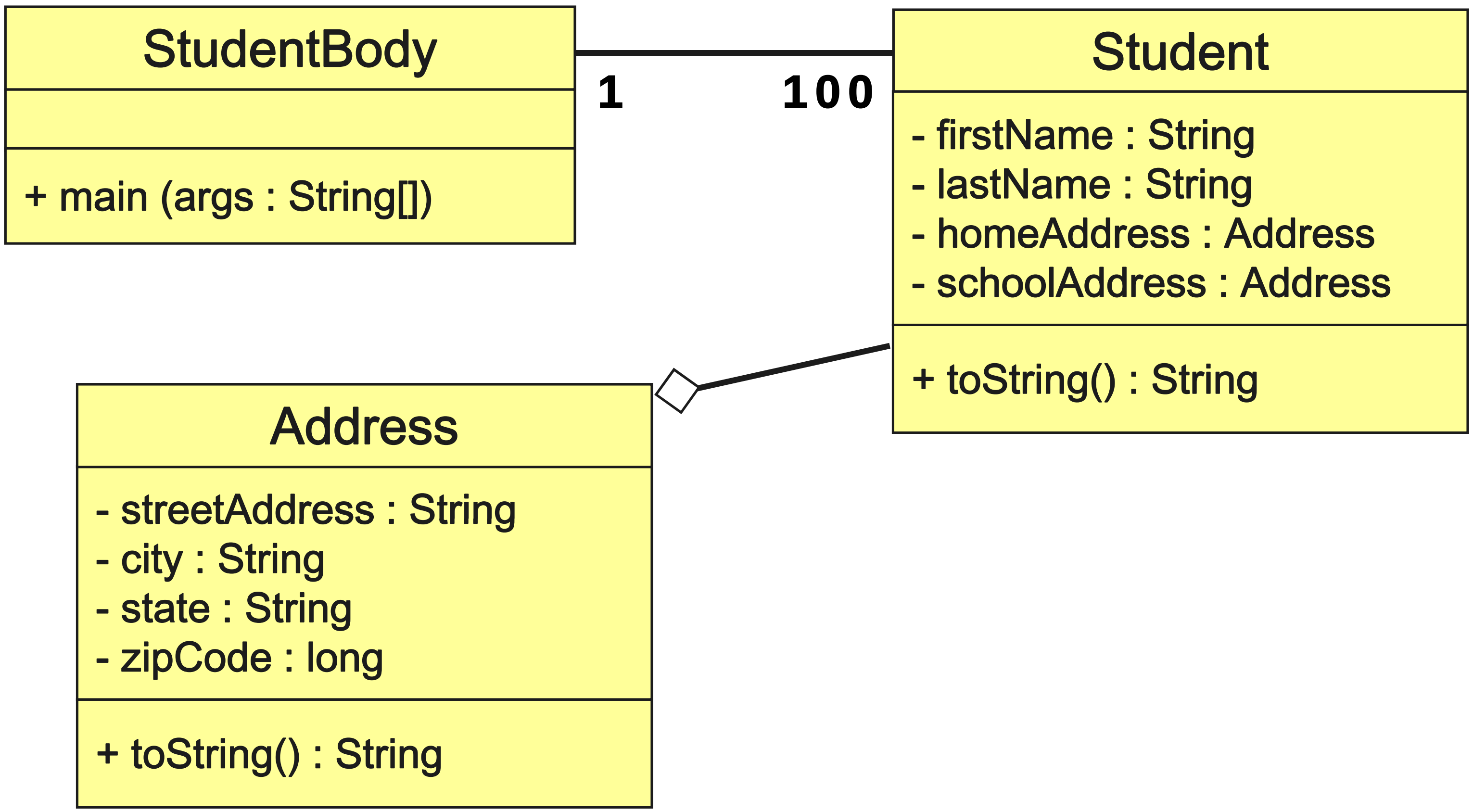
```
TheVotingProgram

VoterAuthentication
voterPersonalInfo: VoterPersonalInformation
voterID: String
voterPassword: securePW

VoterPersonalIdentification
-voterLastName: String
-voterFirstName: String
-voterMiddleName: String
-voterSSN: String
-voterAddress1: String
-voterAddress2: String
-voterCity: String
-voterState: String
-voterZIP: String
+validateZipCode(voterZIP:String):String
+validateState(parameter0VoterState:String):St...

BallotCreation
ballotName: String
candidates: String [];
displayBallot():void
createBallot():void

securePW
PWEntered: JPasswordField
securePW(PW:securePW):securePW
```

this is only a small subset of the actual package ...

# Class diagram example: video store



# Class diagram example: student



StudentBody

+ main (args : String[])

1

100

Student

- firstName : String
- lastName : String
- homeAddress : Address
- schoolAddress : Address

+ toString() : String

Address

- streetAddress : String
- city : String
- state : String
- zipCode : long

+ toString() : String

# Tools for creating UML diags.

- Violet (free)
  - http://horstmann.com/violet/
- Rational Rose
  - http://www.rational.com/
- Visual Paradigm UML Suite (trial)
  - http://www.visual-paradigm.com/
  - (nearly) direct download link: http://www.visual-paradigm.com/vp/download.jsp?product=vpuml&edition=ce

(there are many others, but most are commercial)