

SYSTEM DESIGN /MODELING: CLASS MODELS

User and system requirements

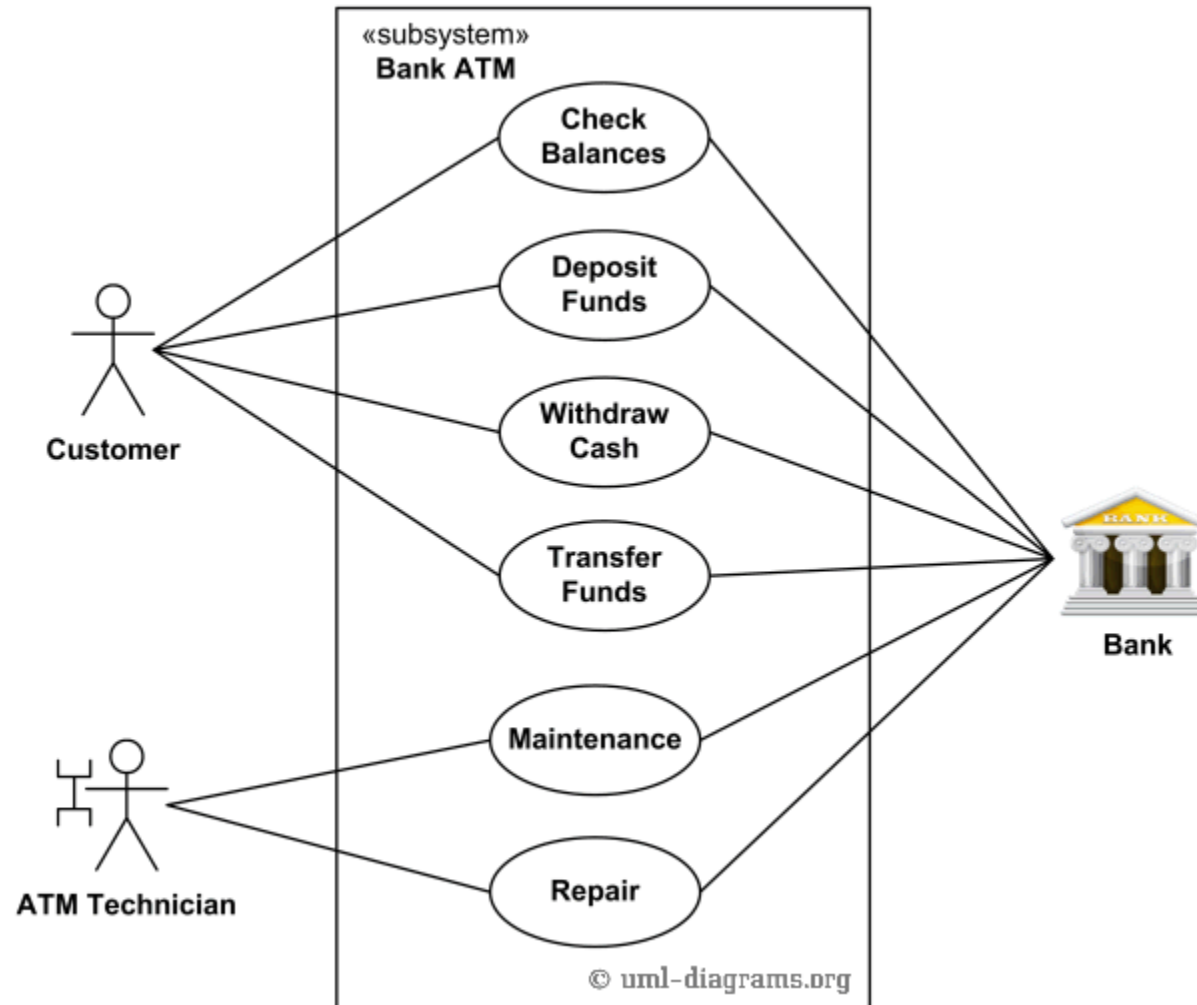
User requirements definition

- 1.** The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

- 1.1** On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2** The system shall generate the report for printing after 17.30 on the last working day of the month.
- 1.3** A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
- 1.4** If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
- 1.5** Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Use case Diagram



Use case diagrams

➤Pros

- To represent functional requirements
- Communication (stake holders and other software engineers)

➤Cons

- No information about hidden rules and assumptions
- No information about Data structures

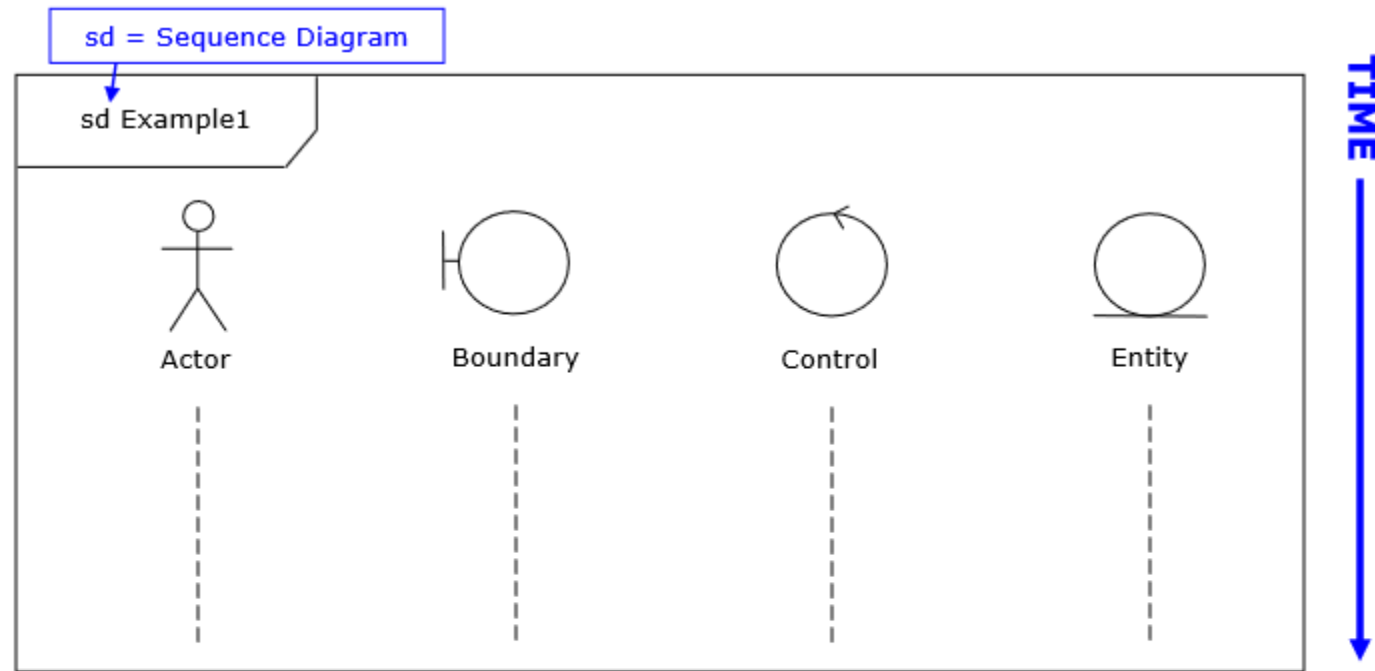
How can we Represent/Specify System Requirements ?

System Requirements

- Understanding: Whole system (as a product)
- Interactions in the system
 - ❖ Identify Components(on what basis)
 - ❖ How to define Components ?
 - ❖ Interactions between components ?

System Requirements :Sequence Diagram/ Prototype

- Understanding: Whole system (as a product)
- Interactions in the system
 - ❖ Identify Components(on what basis)
 - ❖ How to define Components ? (classes)
 - ❖ Interactions between components ? (Communications)



Actor: Stakeholder within a system

Boundary: Objects that interface with the actors. Examples would include MS Windows, Screens, or Menus.

Control: Mediate between Boundaries and Entities. It captures the control logic.

Entity: Objects representing system data. Can be thought of as the from the Entity-Relation perspective.

Class Diagrams/Models

- Components of Class Diagrams/Models
- Interactions
- Model Terminology and Concepts

Defining Class

- A **CLASS** is a template (specification, blueprint) for a collection of objects that share a common set of attributes and operations.
- Static Modeling

Class



Class Name: HealthClubMember

attributes

operations

Objects



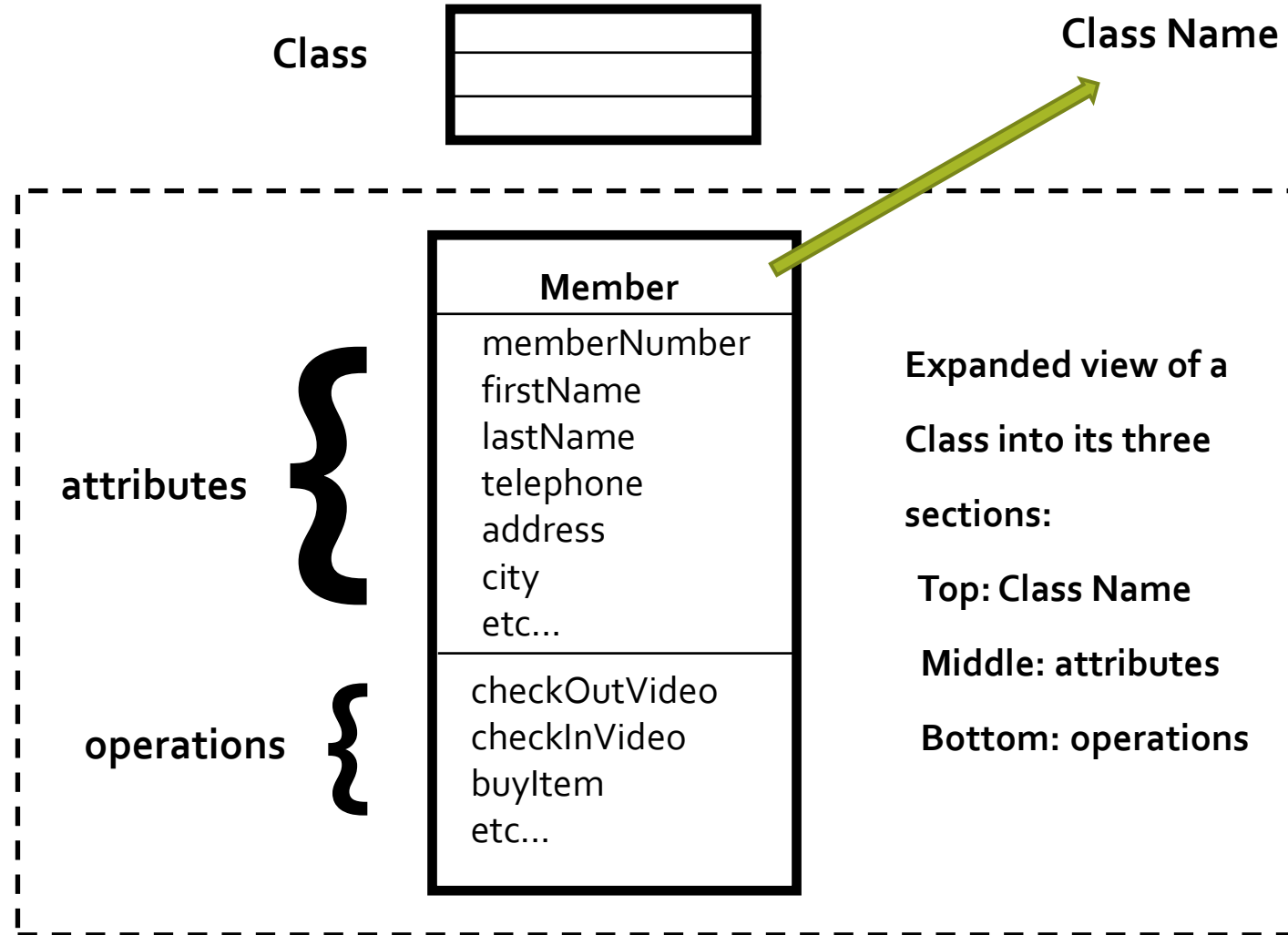
Developing Class Models [UML Models]

- Class diagrams used for different purposes during different times in the development life-cycle
 - Models that are close to code
vs.
 - Models that support earlier modeling:
 - For domain analysis
 - For requirements specification
- Class diagrams developed iteratively
 - Details added over time during lifecycle (sprint 1,2 3, ..etc.)
 - Initially: missing names, relations multiplicities, other details

More Abstract Perspectives/View points

- Some define particular perspectives for class models:
 - Conceptual (requirements)
 - Specification (design, UML)
 - Implementation (code, java)
- Conceptual perspective
 - Represents concepts in the domain
 - Drawn with no regard for implementation (language independent)
 - Used in requirements analysis
- Specification
 - Interfaces defined: a set of operations
 - Focus on interfaces not how implementation broken into classes
- Implementation
 - Direct code implementation of each class in the diagram
 - A blue-print for coding

UML Class Diagram Notation



How to Identify Classes ?

- Class object refer **nouns** in the given system[In general]
- **Information** about classes will be stored in a **file/data structure/data base**
- Are all the nouns are classes ?
 - No, not all the nouns are classes
- Then, how to identify the classes ?, how to differentiate between a class and an attribute ? (table technique as show below, rows &

Y: Table name

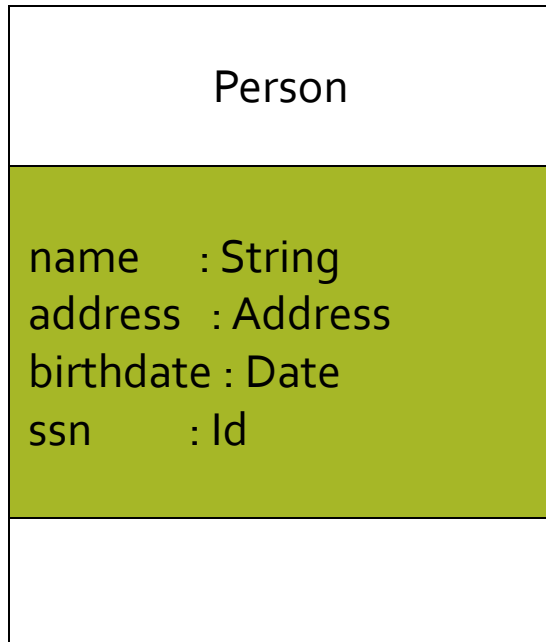
X1	X2	X3	X4

Examples

Class or attribute ? (every view & perception matters, let us know missing part of information)

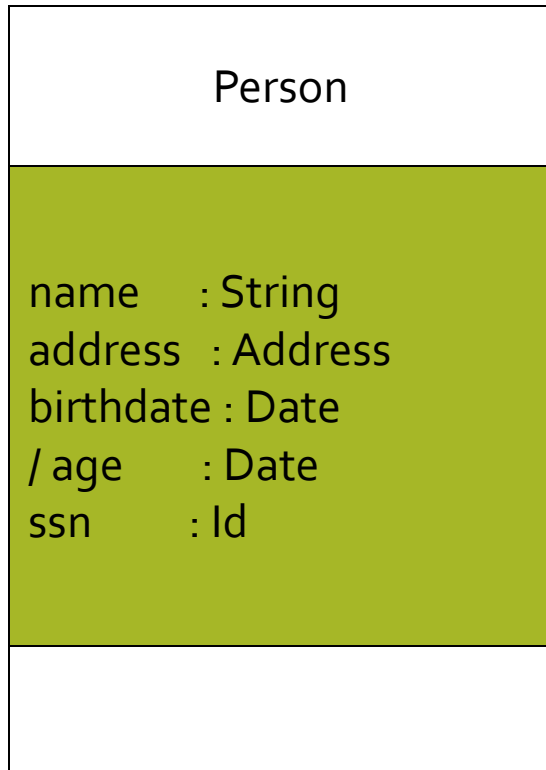
- Car
- Course
- Prerequisite
- International student
- Employee
- GPA

Class Attributes



An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.

Class Attributes (Cont'd)



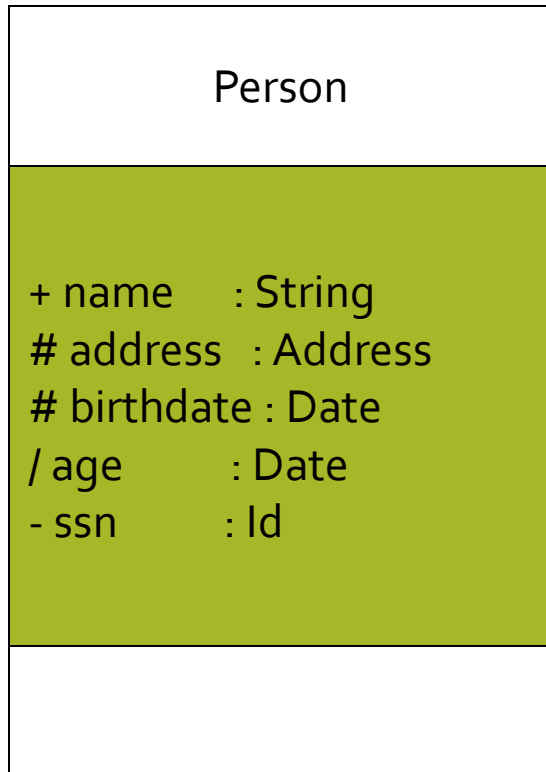
Attributes are usually listed in the form:

attributeName : Type

A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

/ age : Date

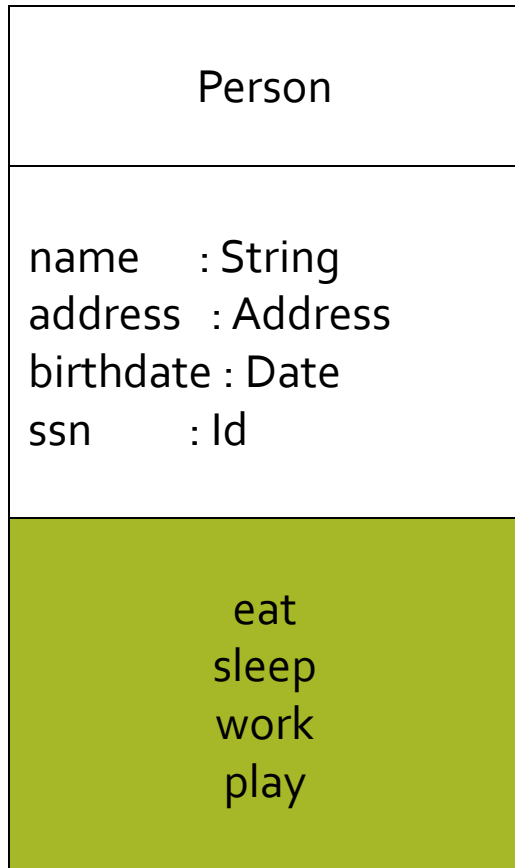
Class Attributes (Cont'd)



Attributes can be:

- + public
- # protected
- private
- / derived

Class Operations



Operations describe the class behavior and appear in the third compartment.

Why do **we** need Class Interactions or Relationship



• *Relationships*

A RELATIONSHIP is what a class or an object knows about another class or object.

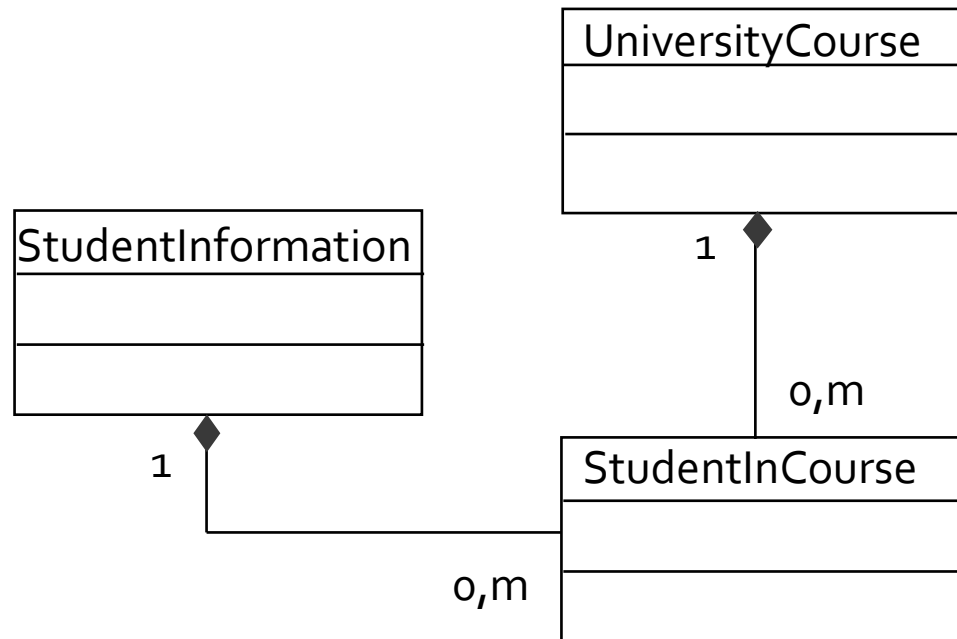
- ❑ *Associations*
- ❑ *Generalization* (*Superclass/Subclass*)
- ❑ *Aggregations*
- ❑ *Composition* (*Whole-Part*)

• Relationships

Exist to:

1) show relationships 2) enforce integrity 3) help produce results

In this example:



- *Removal of a University Course should also remove Students that are in the Course but not Student Information.*
- *Removal of a Student should also remove the Courses that the Student is in but not the University Course.*
- *Removal of a Student in a Course should not affect either University Course or Student Information.*

Class Interactions or Relationship



• *Relationships*

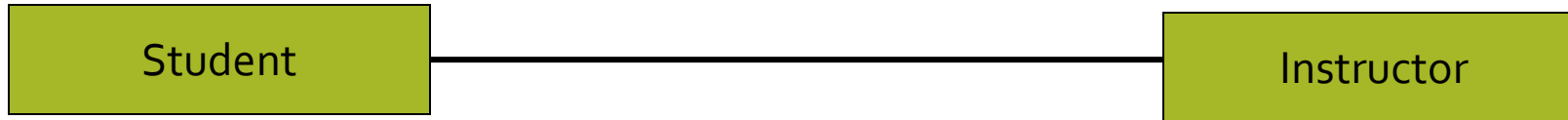
A RELATIONSHIP is what a class or an object knows about another class or object.

- ❑ *Associations*
- ❑ *Generalization* (*Superclass/Subclass*)
- ❑ *Aggregations*
- ❑ *Composition* (*Whole-Part*)

Association Relationships

If two classes in a model need to communicate with each other, there must be link between them.

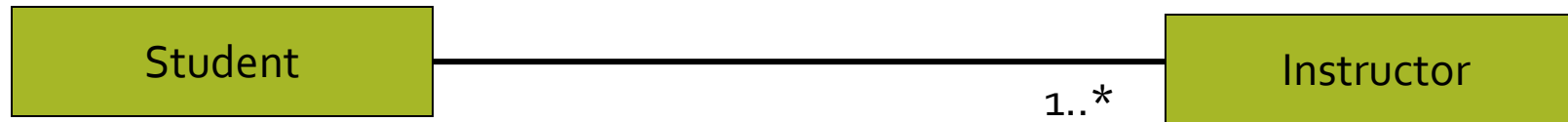
An *association* denotes that link.



Association Relationships (Cont'd)

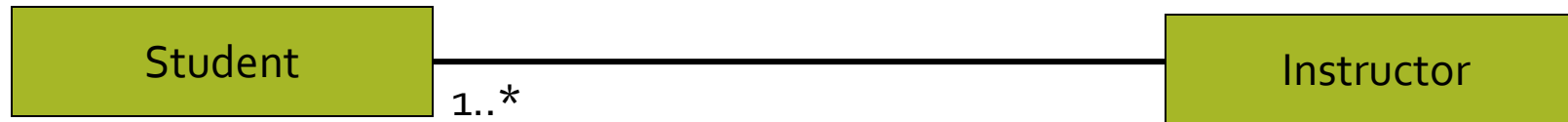
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:



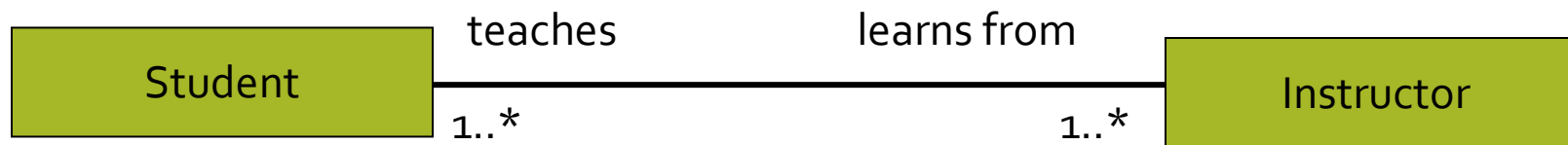
Association Relationships (Cont'd)

The example indicates that every *Instructor* has one or more *Students*:



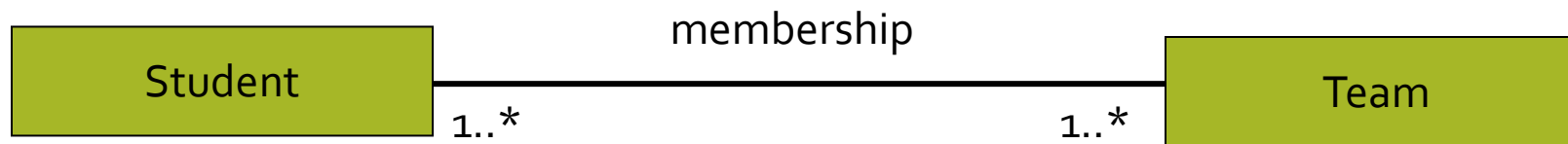
Association Relationships (Cont'd)

We can also indicate the behavior of an object in an association (*i.e.*, the *role* of an object) using *rolenames*.



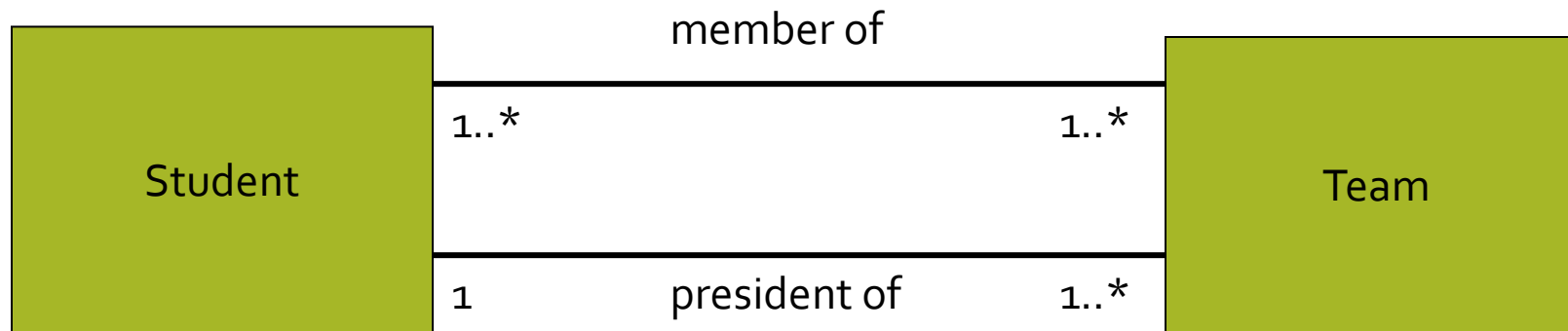
Association Relationships (Cont'd)

We can also name the association.



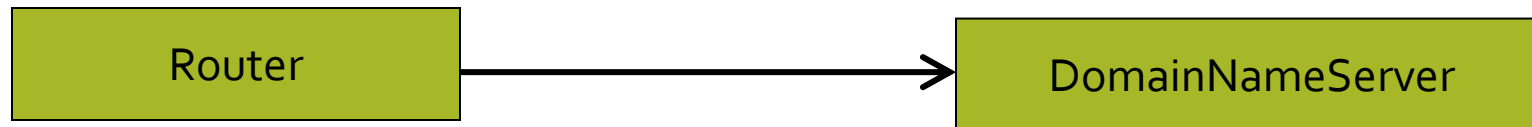
Association Relationships (Cont'd)

We can specify dual associations.



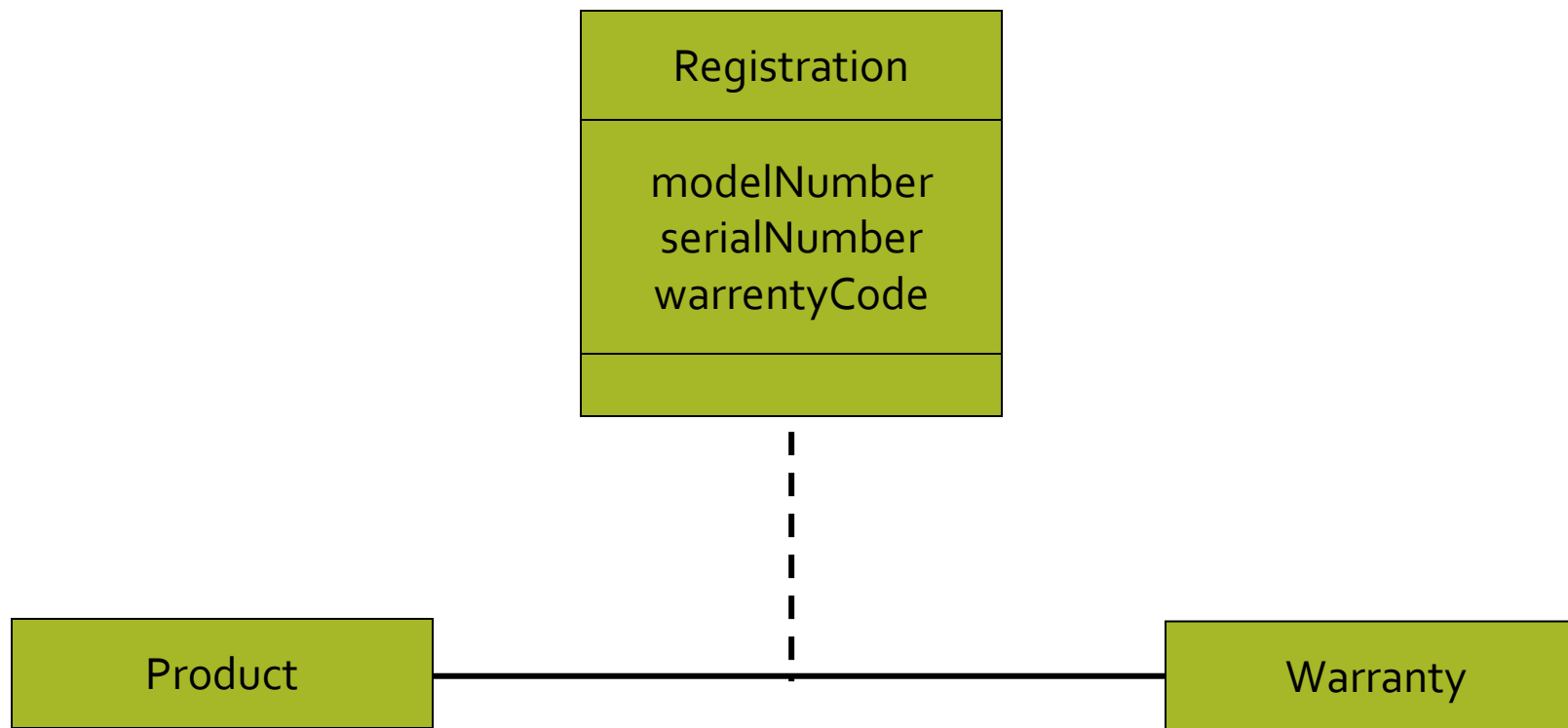
Association Relationships (Cont'd)

We can constrain the association relationship by defining the *navigability* of the association. Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of the *Router*.



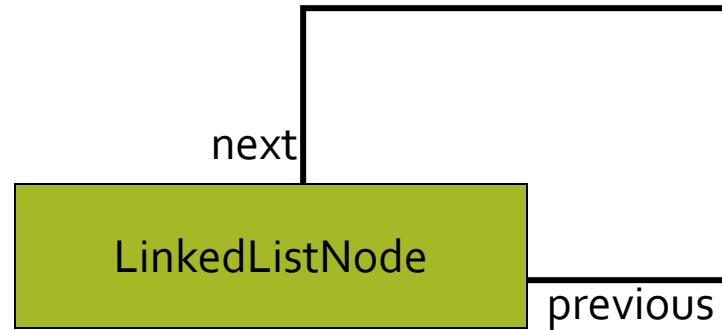
Association Relationships (Cont'd)

Associations can also be objects themselves, called *link classes* or an *association classes*.



Association Relationships (Cont'd)

A class can have a *self association*.



Examples : Out of box or unorthodox


Examples



Examples

Tarzan

2014 PG CC



★ ★ ★ ★ ★ 462 | **IMDb** 4.8/10

[Watch Trailer](#)

Tarzan must protect the jungle - and Jane in this animated family adventure.

Starring: Kellan Lutz, Spencer Locke
Runtime: 1 hour, 35 minutes
Available to watch on [supported devices](#).

amazonPrime

Watch for \$0.00 with a Prime membership

[Start your 30-day free trial and Watch Now](#)

Also available to Buy

[Buy HD \\$14.99](#) [Buy SD \\$12.99](#)

[Redeem a gift card or promotion code](#)

[Add to Watchlist](#)

[Send us Feedback](#)

Case study: Find out classes, attributes, and operations

- the architects assigned the task of constructing the design elements for a system that can be used to manage courses/classes for an organization that specializes in providing training. Let us name the system that we will be designing as the Courseware Management System. The organization offers a variety of courses in a variety of areas such as learning management techniques and understanding different software languages and technologies. Each course is made up of a set of topics. Tutors in the organization are assigned courses to teach according to the area that they specialize in and their availability. The organization publishes and maintains a calendar of the different courses and the assigned tutors every year. There is a group of course administrators in the organization who manage the courses including course content, assign courses to tutors, and define the course schedule. The training organization aims to use the Courseware Management System to get a better control and visibility to the management of courses as also to streamline the process of generating and managing the schedule of the different courses.

Noun Extraction

- Take some concise statement of the requirements
- Underline nouns or noun phrases that represent things
 - These are *candidate classes*
- Object or not?
 - Inside our system scope?
 - An event, states, time-periods?
 - An attribute of another object?
 - Synonyms?
- Again, looking for “things”

Case study 2: University Courses

- Some instructors are professors, while others have job title adjunct
- Departments offer many courses, but a course may be offered by >1 department
- Courses are taught by instructors, who may teach up to three courses
- Instructors are assigned to one (or more) departments
- One instructor also serves a department chair

Case study 2 : University Courses

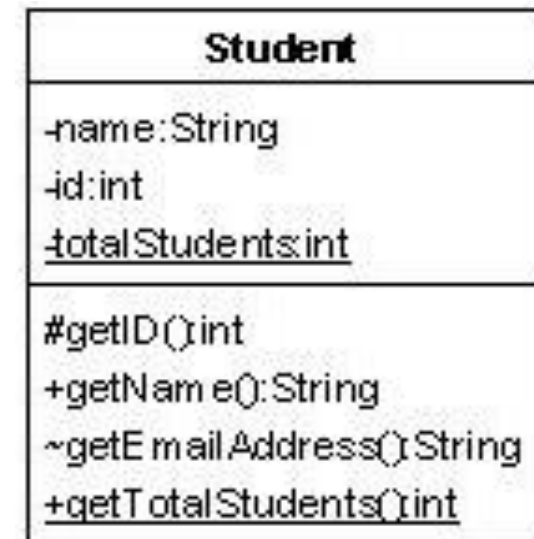
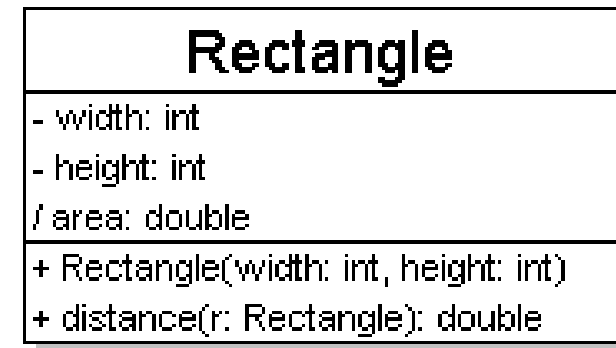
- Some **instructors** are **professors**, while others have job title **adjunct**
- **Departments** offer many courses, but a **course** may be offered by >1 department
- Courses are taught by instructors, who may teach up to three courses
- Instructors are assigned to one (or more) departments
- One instructor also serves a **department chair**

Case study 2: University Courses

- A course has title, description, name, credit hours and prerequisite.
- Furniture shop has different models for chairs, and tables. And each model have different colors, brands, prices, and measurements.
- Classes & Attributes ?

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



Class attributes

- attributes (fields, instance variables)
 - 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 <u>-totalStudents:int</u>
#getID()int +getName():String ~getEmailAdress()String <u>+getTotalStudents()int</u>

Class operations / methods

- operations / methods
 - *visibility name (parameters) : return_type*
- visibility:
 - + public
 - # protected
 - private
 - ~ package (default)
- 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 <u>-totalStudents:int</u>
#getID()int +getName():String ~getEmailAdress()String <u>+getTotalStudents()int</u>

Relationship between Classes

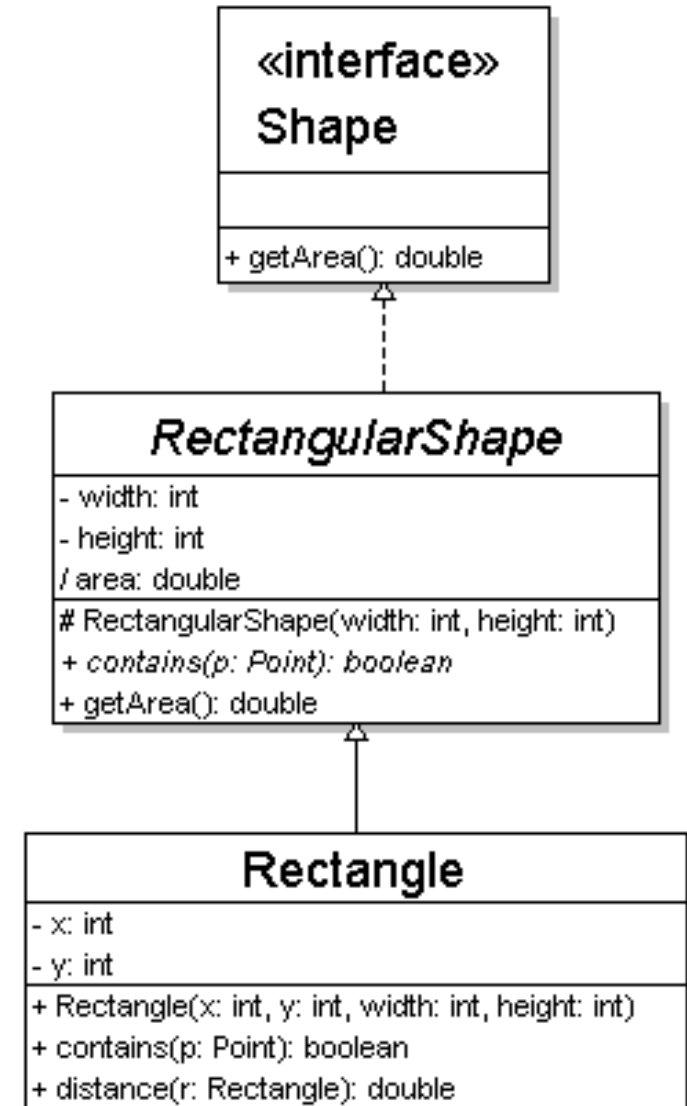
- **Generalization**: an inheritance relationship
 - inheritance between classes
 - interface implementation
- **Association**: a usage relationship
 - dependency
 - aggregation
 - composition

How to identify class relationships ?

- Search for verbs in textual description

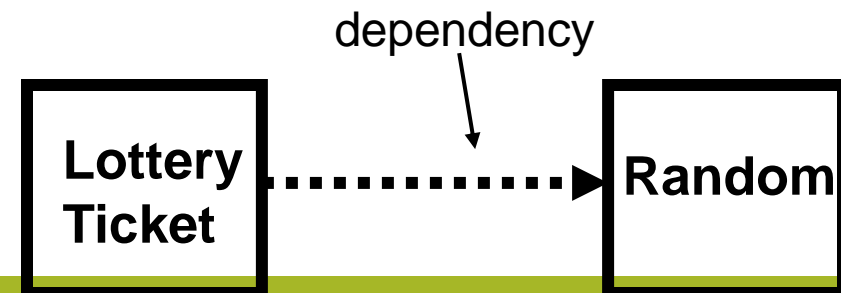
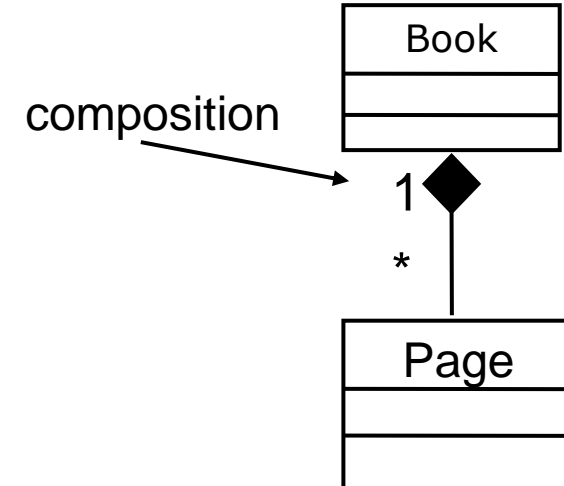
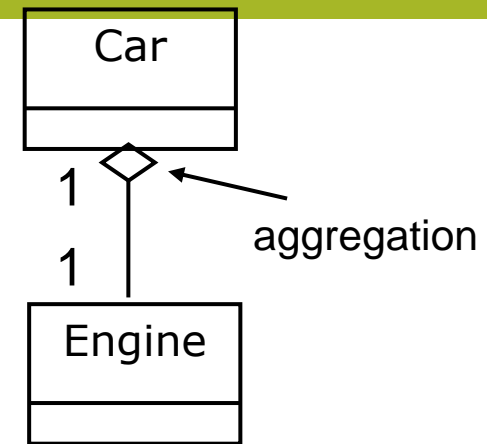
Generalization relationships

- generalization (inheritance) relationships
 - hierarchies drawn top-down with arrows pointing upward to parent
 - line/arrow styles differ, based on whether parent is a(n):
 - class:
solid line, black arrow
 - abstract class:
solid line, white arrow
 - interface:
dashed line, white arrow
- we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent




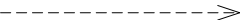


Association Types

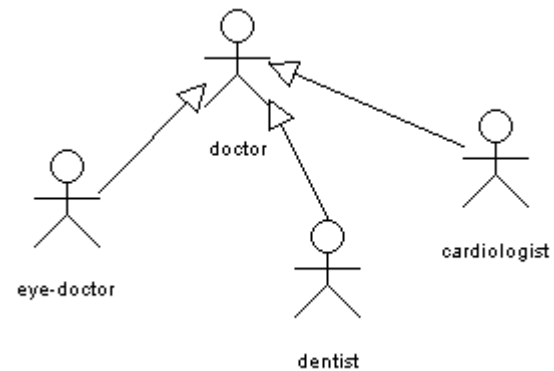
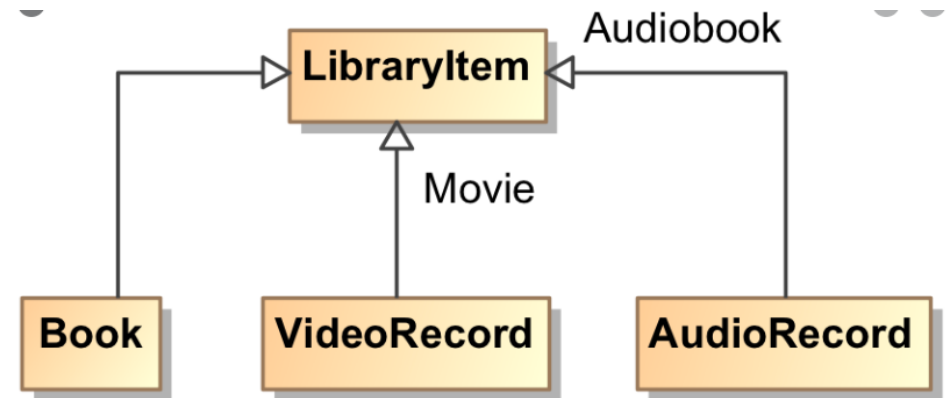
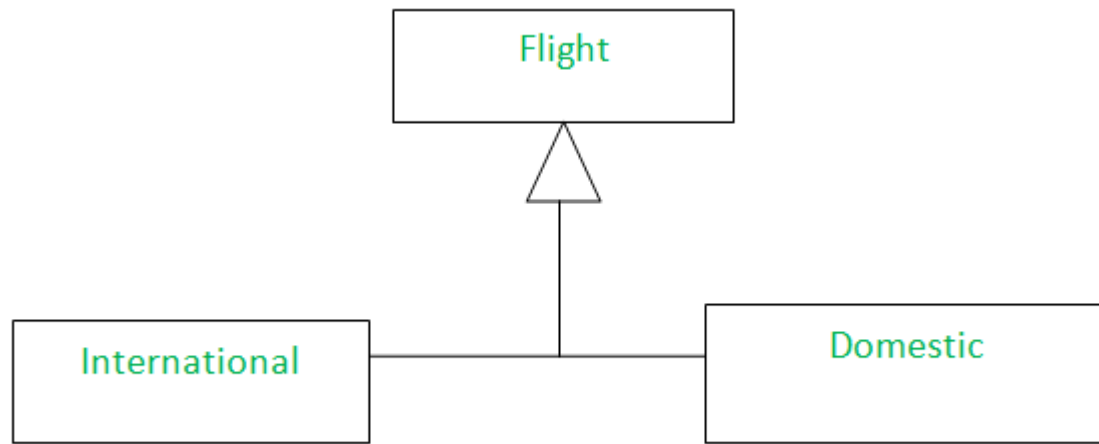
- **aggregation:** "is part of"
 - symbolized by a clear white diamond
- **composition:** "is entirely made of"
 - stronger version of aggregation
 - the parts live and die with the whole
 - symbolized by a black diamond
- **dependency:** "uses temporarily"
 - symbolized by dotted line
 - often is an implementation detail, not an intrinsic part of that object's state



Structural Modeling: Core Relationships

Construct	Description	Syntax
association	a relationship between two or more classifiers that involves connections among their instances.	
aggregation	A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.	
generalization	a taxonomic relationship between a more general and a more specific element.	
dependency	a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).	

Generalization



Associational relationships

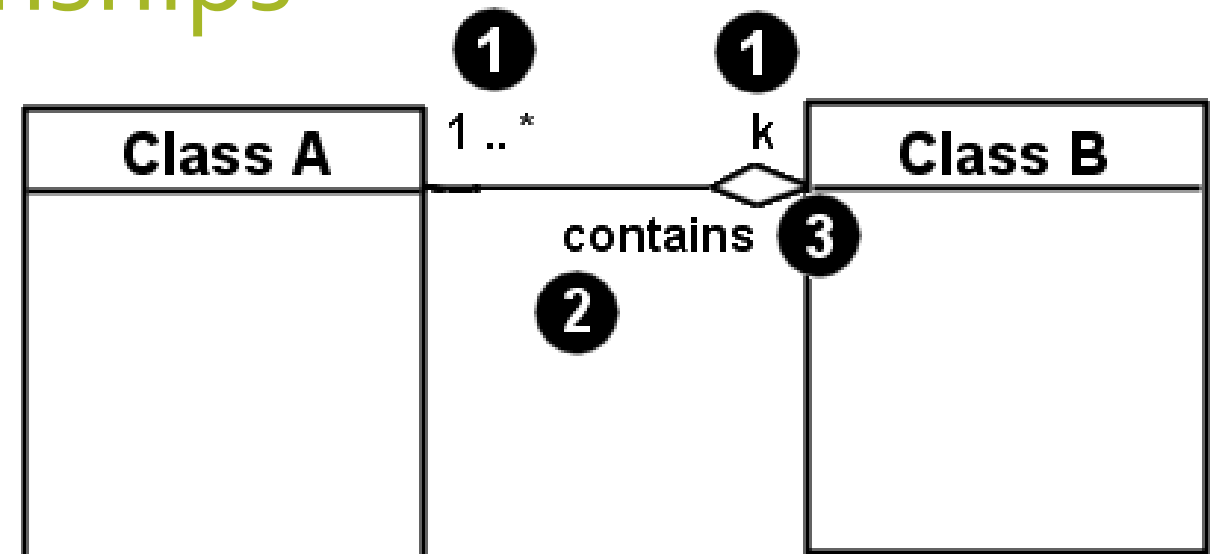
- associational (usage) relationships

1. multiplicity (how many are used)

- * \Rightarrow 0, 1, or more
- 1 \Rightarrow 1 exactly
- 2..4 \Rightarrow between 2 and 4, inclusive
- 3..* \Rightarrow 3 or more

2. name (what relationship the objects have)

3. navigability (direction)

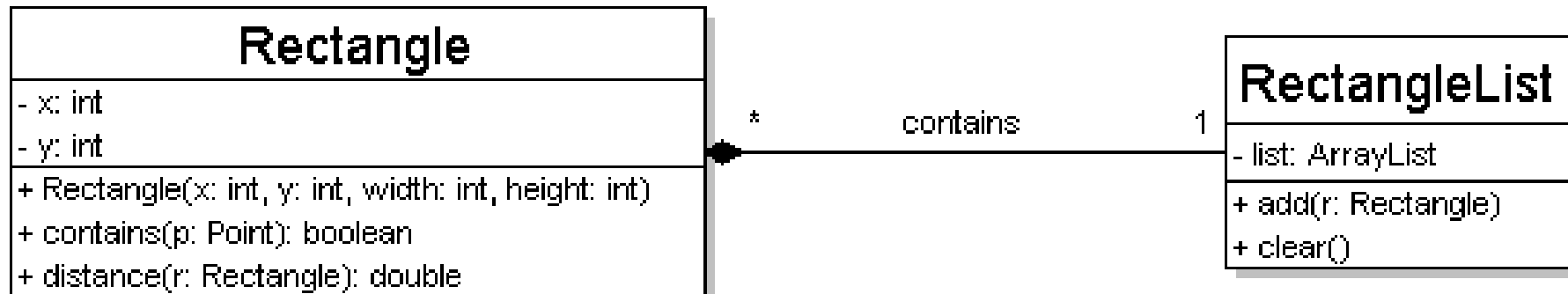


Multiplicity of associations

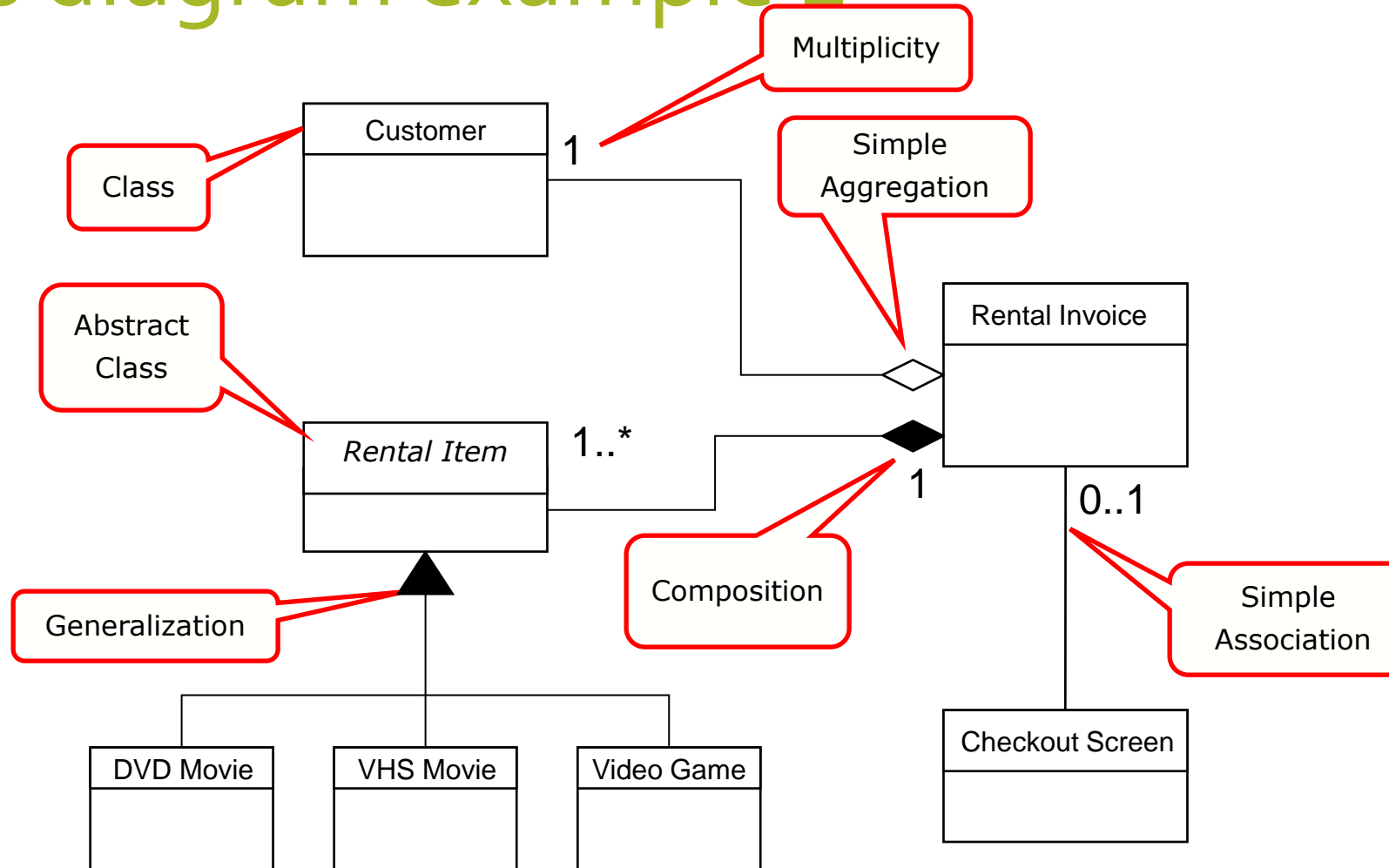
- one-to-one
 - each student must carry exactly one ID card



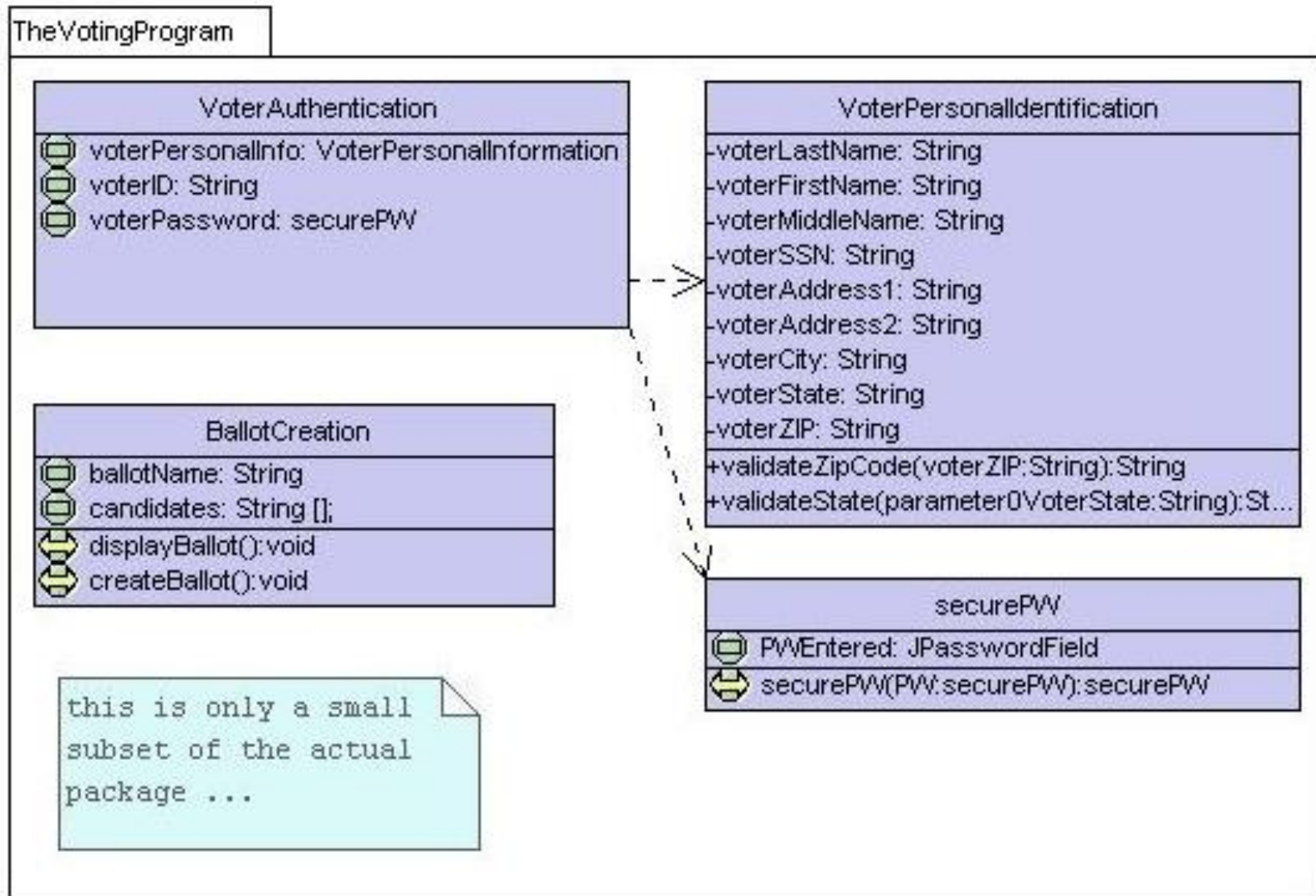
- one-to-many
 - one rectangle list can contain many rectangles



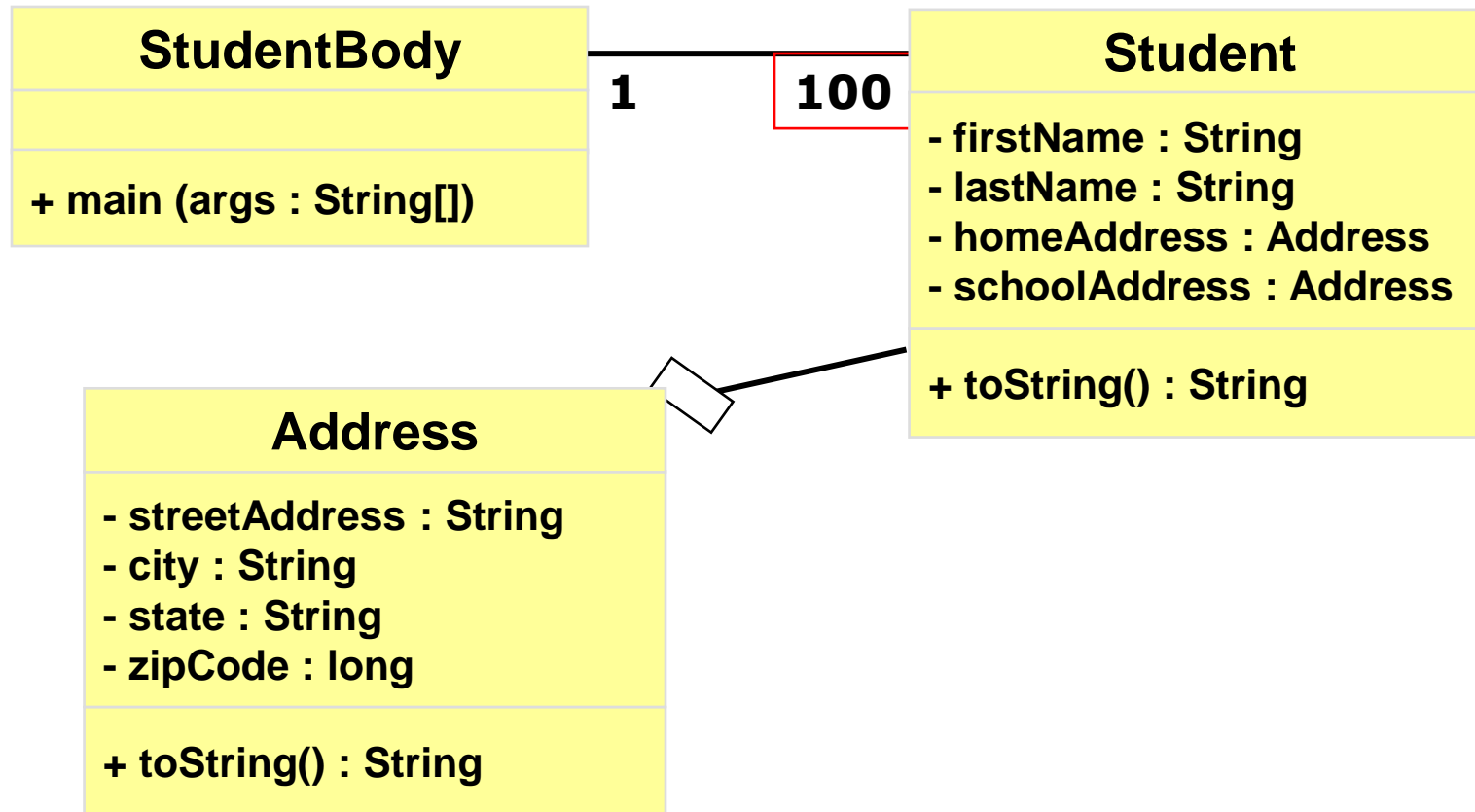
Class diagram example 2



Class diagram example 1



Class diagram example 3



Reference Links

- http://www.uml.org/HTB_Articulate_Class_Models_OMG.pdf
- <http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf>
- <http://www.uml-diagrams.org/class-diagrams-overview.html>
- [google images]
- Software Engineering, 9/10th Edition Author: Ian Sommerville