Topic 07: Requirement Modeling using Class Diagram

References

- All materials in these slides are from the slides of:
  - Dennis, Alan, et. al., System Analysis and Design with UML 3rd Edition, John Wiley & Sons, 2010.

Outline

- Structural Model
- Classes, Attributes & Operations
- Deriving Class Diagram from Use Case
- Class Responsibility & Collaboration Cards
- Class Diagram
- Creating CRC & Class Diagram

Objectives

- Understand the rules and style guidelines for creating CRC cards, class diagrams, and object diagrams.
- Understand the processes used to create CRC cards, class diagrams, and object diagrams.
- Be able to create CRC cards, class diagrams, and object diagrams.
- Understand the relationship between the structural and use case models.

Key Ideas

- Use-case models provide an external functional view of the system (what the system does)
- A structural or conceptual models describe the structure of the data that supports the business processes in an organization.
- The structure of data used in the system is represented through ***CRC cards***, ***class diagrams***, and ***object diagrams***.

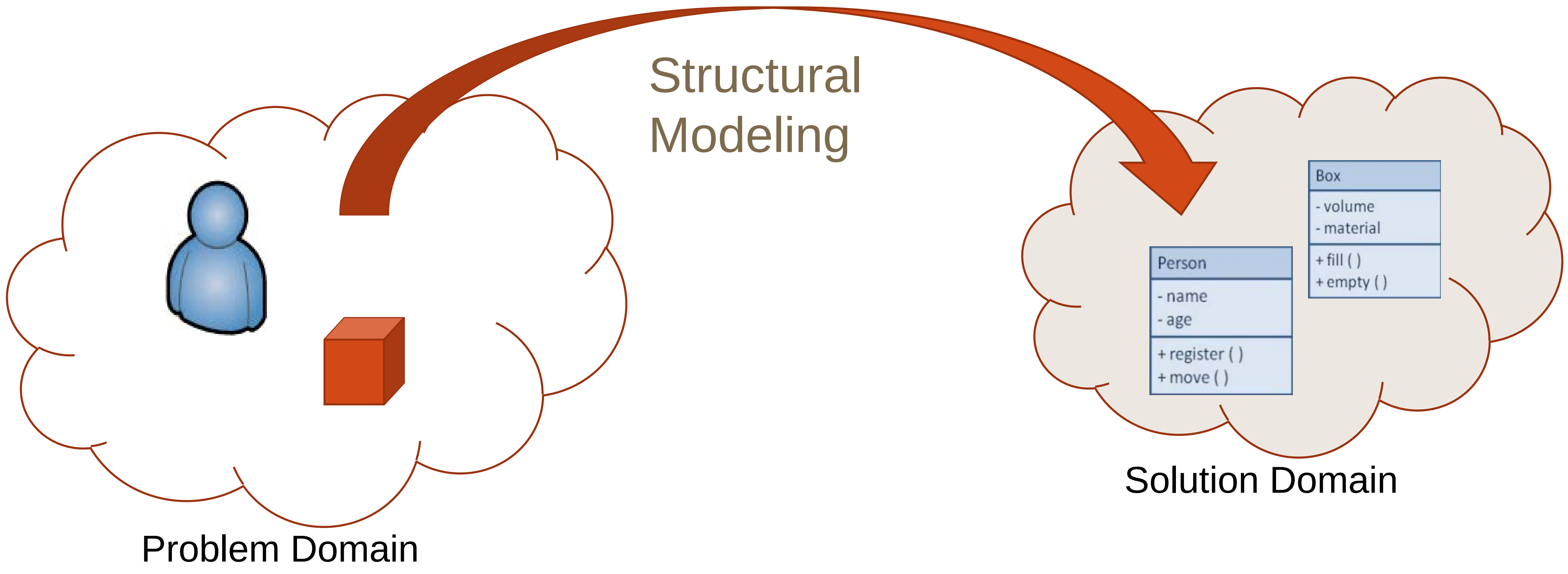
STRUCTURAL MODELS

Structural Model

- A structural model is a formal way of representing the objects (things, ideas, and concepts) that are used and created by a business system
- Reduce the “semantic gap” between the real world and the world of software
- Create a vocabulary for analysts and users
- Structural model also allow the representation of the relationships between the objects
- Drawn using an iterative process
  - First drawn in a conceptual, business-centric way
  - Then refined in a technology-centric way describing the actual databases and files

Structural Models

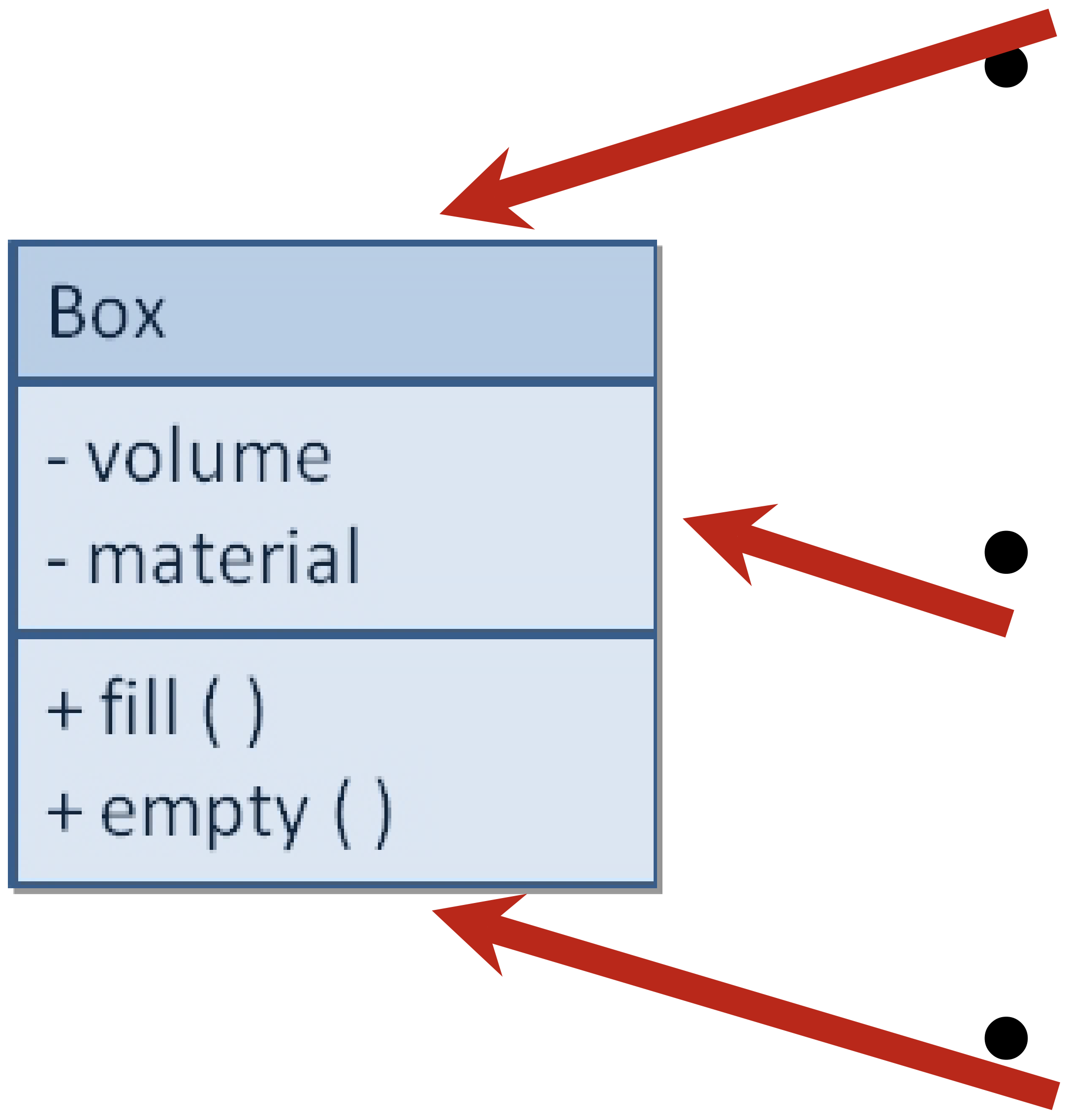
- **Main goal**: to discover the key data contained in the problem domain and to build a structural model of the objects



A Common Language

- Structural models create a well-defined vocabulary shared by users and analysts
  - Classes created during analysis are not the classes that programmers develop during implementation
  - This refinement comes later
- Typical structural models:
  - CRC cards
  - Class (and Object) diagrams

Classes, Attributes & Operations



- Classes
  - Templates for instances of people, places, or things
- Attributes
  - Properties that describe the state of an instance of a class (an object)
- Operations
  - Actions or functions that a class can perform

Attributes

- Units of information relevant to the description of the class
- Only attributes important to the task should be included
- include only those attributes that are relevant to the current application
  - "size" for a TV object is not necessary if the application is concerned only with connections between TV and other objects
- do not attempt to include every attribute at the beginning
  - more attributes can be added when the model is iterated
  - iteration of the model occurs almost at all times, particularly for large projects

Attributes (Cont)

- Avoid derivatives attributes; instead, create a function to derive each such attribute
  - “age” can be derived from “birthday” and “current-date”
  - “total” of all transactions can be dynamically computed when necessary, rather than storing it somewhere
  - “discount price” can be computed using current-price and discount factor

Relationships

- Describe how classes relate to one another
- Three basic types in UML

1. Generalization
   - ✴ Enables inheritance of attributes and operations
   - ✴ E.g.: A CUSTOMER class and an EMPLOYEE class can be generalized into a PERSON class by extracting the attributes and operations in common
2. Aggregation
   - ✴ Relates parts to wholes
   - ✴ E.g.: A door is a part of a car
3. Association
   - ✴ Miscellaneous relationships between classes
   - ✴ E.g.: A patient – an appointment

Generalization (bottom-up approach)

- Motivated by identifying similarities and common features among classes
  - “Part-time Instructor” derived from “Instructor” and “Student” while modeling a department
  - “User” derived from “Customer”, “Bank Manager” and “Teller” while modeling an ATM system
  - “Material” derived from “Book”, “Journal” and “Magazine” while modeling a library catalog system

Generalization (top-down approach)

- We can use “specialization” (top-down approach)
  - “Employee” become “ Secretary”, “Engineer”
  - “ Student” become “Full-time Student”, “Part-time Student”
  - “TV” become “Plasma TV”, “Flat Panel TV”

Identify as association if it is not clear whether it is association or aggregation

- “Mail” has “Address”
- “Mail” uses “Address” for delivery
- “Customer” has “Address”
- “Customer” resides at “Address”
- “TV” includes “Screen”
- “TV” sends information to “Screen”
- “Customer” is a “Users”

Identify as association if it is not clear whether it is association or aggregation

- ✤ “Mail” has “Address” (aggregation)
- ✤ “Mail” uses “Address” for delivery (association)
- ✤ “Customer” has “Address” (aggregation)
- ✤ “Customer” resides at “Address” (association)
- ✤ “TV” includes “Screen” (aggregation)
- ✤ “TV” sends information to “Screen” (association)
- ✤ “Customer” is a “Users” (generalization/specialization)

Deriving components of a class diagram from a use case diagram

- **Actors** are **potential candidates** for classes
  - Sometimes, an actor may not be modeled as a class
- Generalization or specialization between actors will end up in generalization or specialization relationship between the corresponding classes

Deriving components of a class diagram from a use case diagram

- Two actors will be related if they are connected through a series of use cases
  - The classes corresponding to these actors will thus have an association
  - The ATM example – the actor "User" is related to "Account" because of the use cases "deposit" and "update account"

Class Responsibility-Collaboration Cards

CRC CARDS

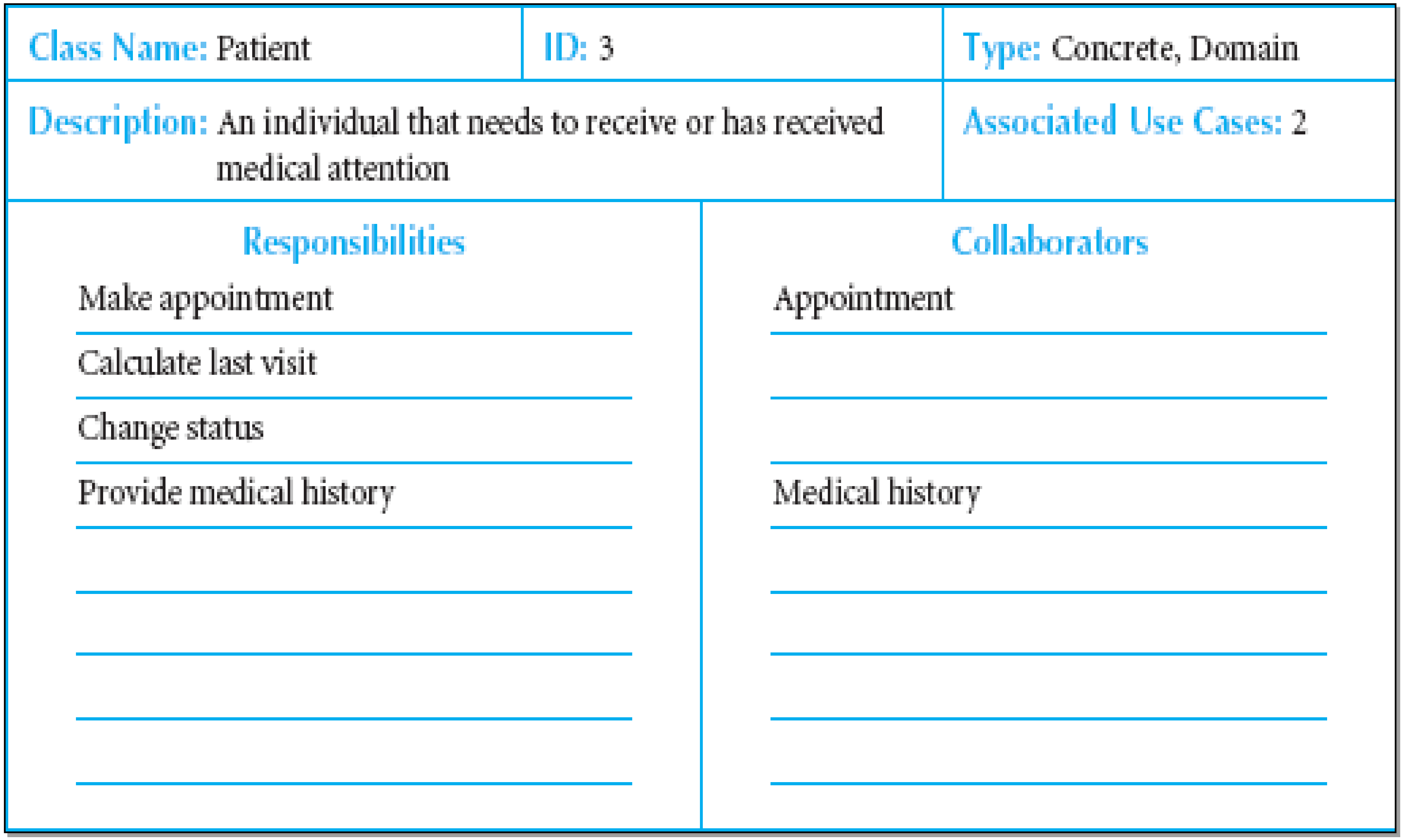
Responsibilities

- Knowing responsibilities are those things that an instance of a class must be capable of knowing.
  - An instance of a class typically knows the values of its attributes and its relationship
- Doing responsibilities are those things that an instance of a class must be capable of doing
  - An instance of a class can execute its operations

Collaboration

- Objects working together to service a request
- Collaborations allow the analyst to think in terms of clients, servers, and contracts
  - A *client* object is an instance of a class that sends a request to an instance of another class for an operation to be executed.
  - A *server* object is the instance that receives the request from the client object.
  - A *contract* formalizes the interactions between the client and server objects.
- For example, a patient makes an appointment with a doctor.
  - This sets up an obligation for both the patient and doctor to appear at the appointed time. Otherwise, consequences, such as billing the patient for the appointment regardless of whether the patient appears, can be dealt out. The contract should also spell out what the benefits of the contract will be, such as a treatment being prescribed for whatever ails the patient and a payment to the doctor for the services provided.

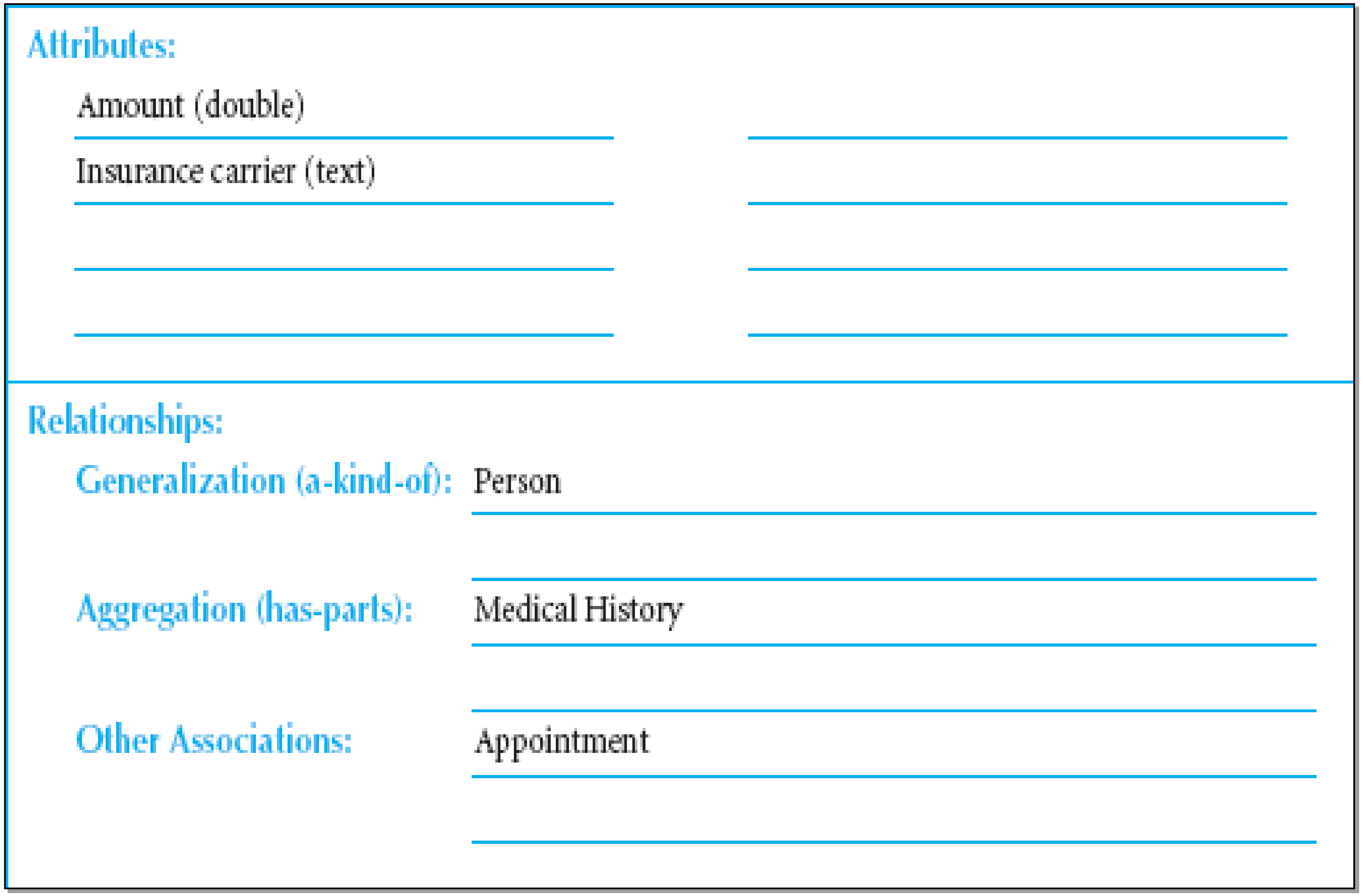
Front-Side of a CRC Card



Class Name: Patient	ID: 3	Type: Concrete, Domain
Description: An individual that needs to receive or has received medical attention		Associated Use Cases: 2

Responsibilities	Collaborators
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back-Side of a CRC Card



Attributes:

Amount (double)

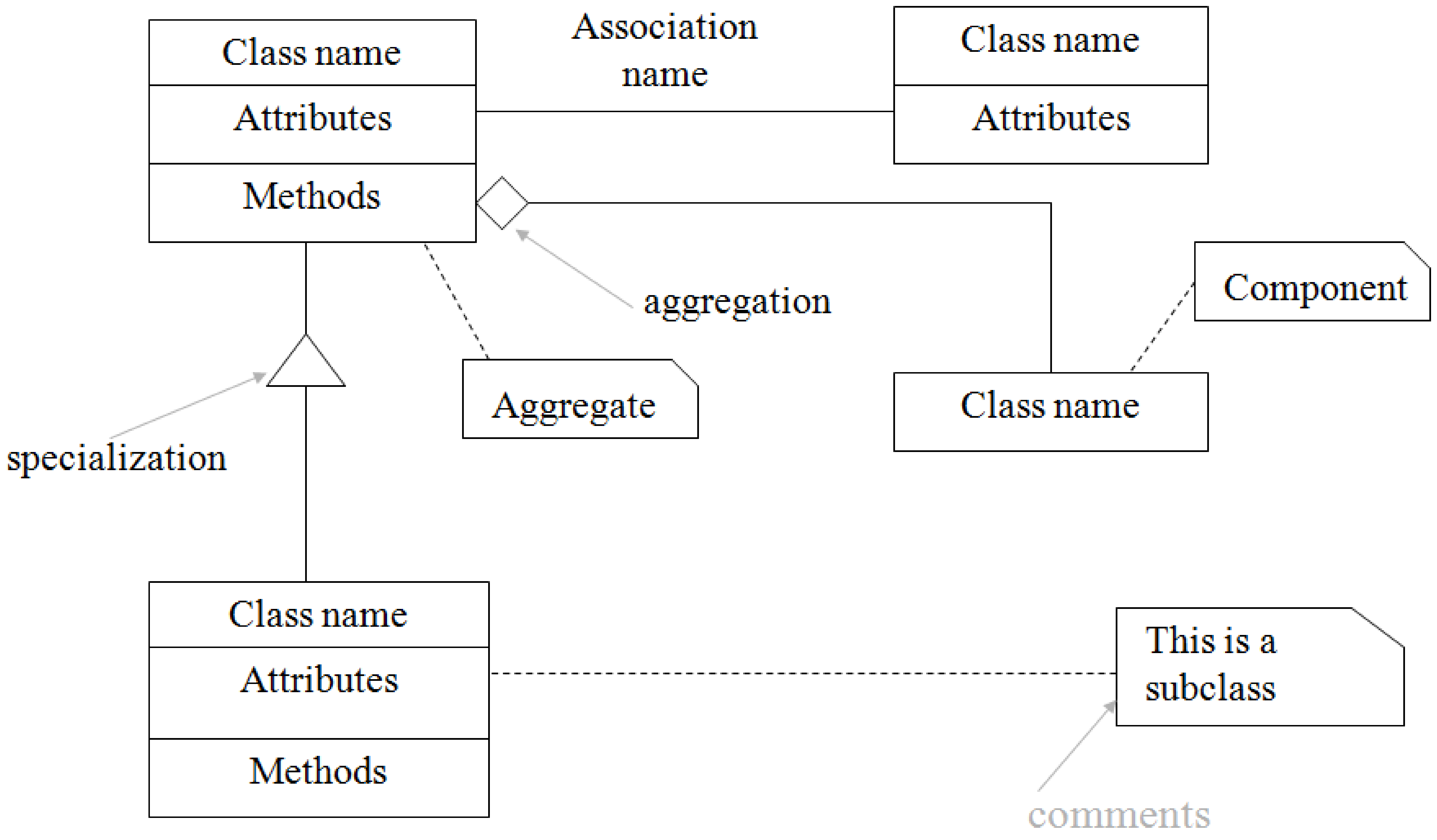
Insurance carrier (text)

Relationships:

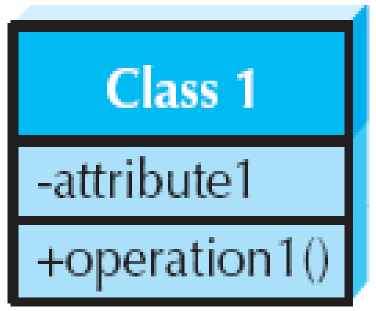
Generalization (a-kind-of):	Person
Aggregation (has-parts):	Medical History
Other Associations:	Appointment

CLASS DIAGRAMS

Class diagram – basic syntax



Elements of a Class Diagram

A class: • Represents a kind of person, place, or thing about which the system will need to capture and store information. • Has a name typed in bold and centered in its top compartment. • Has a list of attributes in its middle compartment. • Has a list of operations in its bottom compartment. • Does not explicitly show operations that are available to all classes.	
An attribute: • Represents properties that describe the state of an object. • Can be derived from other attributes, shown by placing a slash before the attribute's name.	attribute name /derived attribute name
An operation: • Represents the actions or functions that a class can perform. • Can be classified as a constructor, query, or update operation. • Includes parentheses that may contain parameters or information needed to perform the operation.	operation name ()

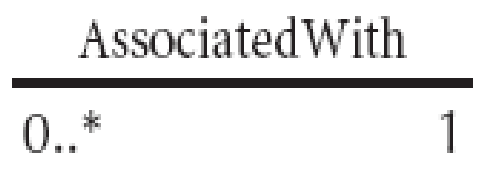
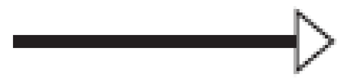
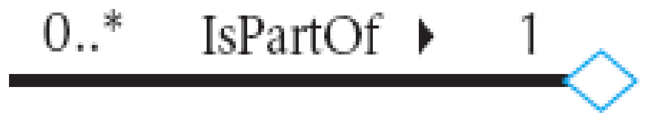
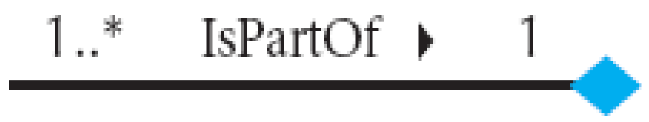
Attribute Visibility

- Attribute visibility can be specified in the class diagram
  - Public attributes (+) are visible to all classes
  - Private attributes (-) are visible only to an instance of the class in which they are defined
  - Protected attributes (#) are like private attributes, but are also visible to descendant classes
- Visibility helps restrict access to the attributes and thus ensure consistency and integrity

Operations

- Constructor
  - Creates object
- Query
  - Makes information about state available
- Update
  - Changes values of some or all attributes

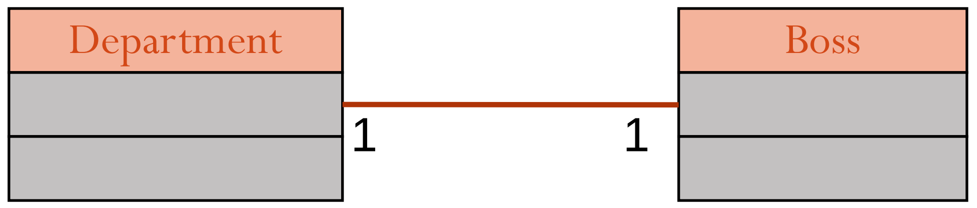
Relationships

An association: • Represents a relationship between multiple classes or a class and itself. • Is labeled using a verb phrase or a role name, whichever better represents the relationship. • Can exist between one or more classes. • Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	
A generalization: • Represents a-kind-of relationship between multiple classes.	
An aggregation: • Represents a logical a-part-of relationship between multiple classes or a class and itself. • Is a special form of an association.	
A composition: • Represents a physical a-part-of relationship between multiple classes or a class and itself • Is a special form of an association.	

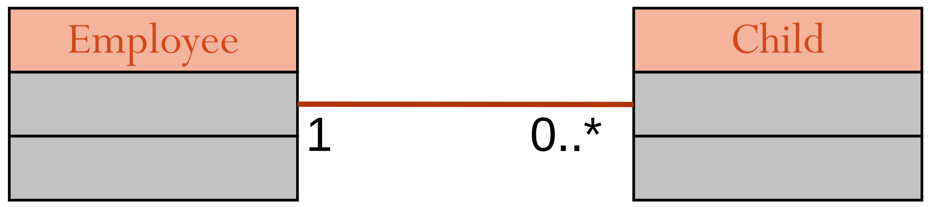
Multiplicities

Exactly one	1	$\begin{tikzcd} \boxed{\text{Department}} \arrow[r, dash, "1"'] & \boxed{\text{Boss}} \end{tikzcd}$	A department has one and only one boss.
Zero or more	0..*	$\begin{tikzcd} \boxed{\text{Employee}} \arrow[r, dash, "0..*"'] & \boxed{\text{Child}} \end{tikzcd}$	An employee has zero to many children.
One or more	1..*	$\begin{tikzcd} \boxed{\text{Boss}} \arrow[r, dash, "1..*"'] & \boxed{\text{Employee}} \end{tikzcd}$	A boss is responsible for one or more employees.
Zero or one	0..1	$\begin{tikzcd} \boxed{\text{Employee}} \arrow[r, dash, "0..1"'] & \boxed{\text{Spouse}} \end{tikzcd}$	An employee can be married to zero or one spouse.
Specified range	2..4	$\begin{tikzcd} \boxed{\text{Employee}} \arrow[r, dash, "2..4"'] & \boxed{\text{Vacation}} \end{tikzcd}$	An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5	$\begin{tikzcd} \boxed{\text{Employee}} \arrow[r, dash, "1..3{,}5"'] & \boxed{\text{Committee}} \end{tikzcd}$	An employee is a member of one to three or five committees.

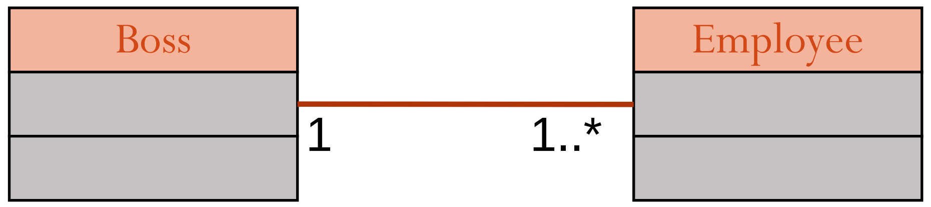
Association Relationship



Exactly one:
A department has one and only one boss

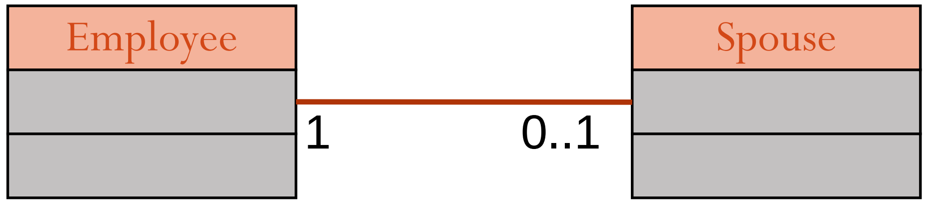


Zero or more:
An employee has zero to many children

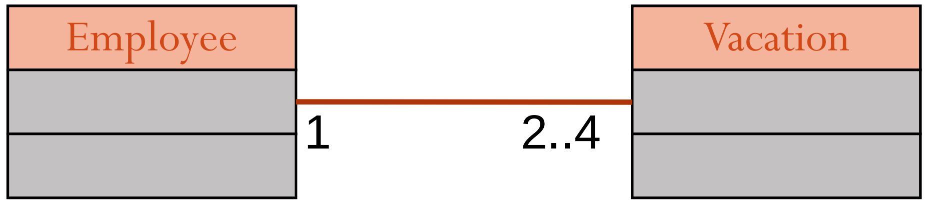


One or more:
A boss is responsible for one or more employees

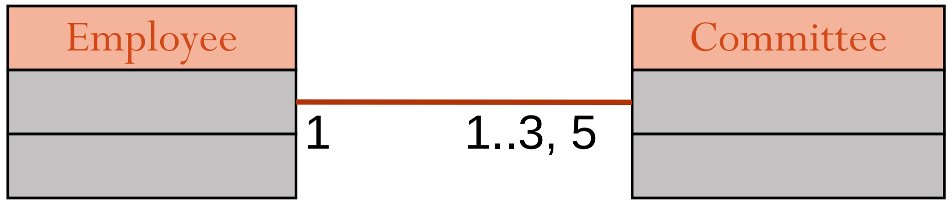
Association Relationship



Zero or one:
An employee can be married to 0 or 1 spouse



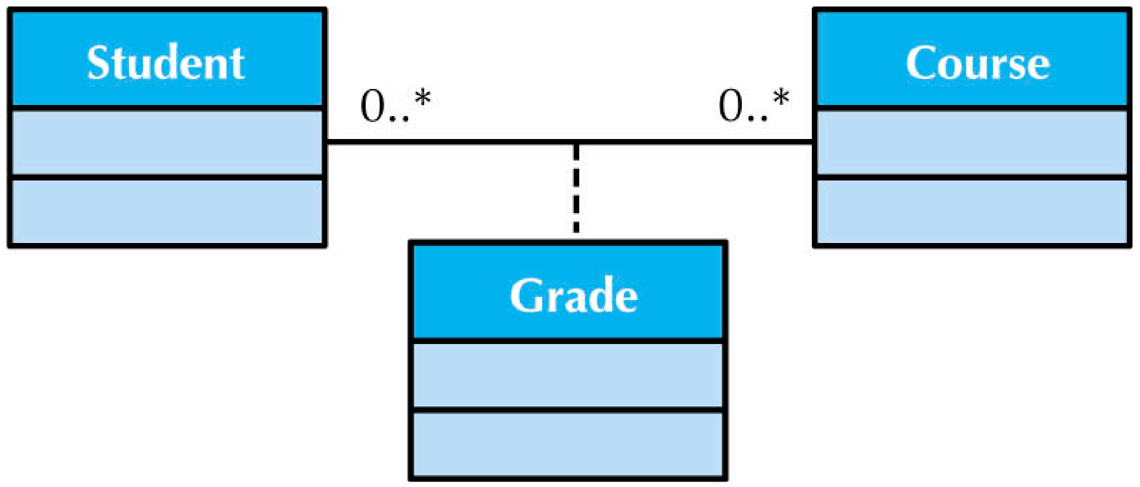
Specified range:
An employee can take 2 to 4 vacations each year

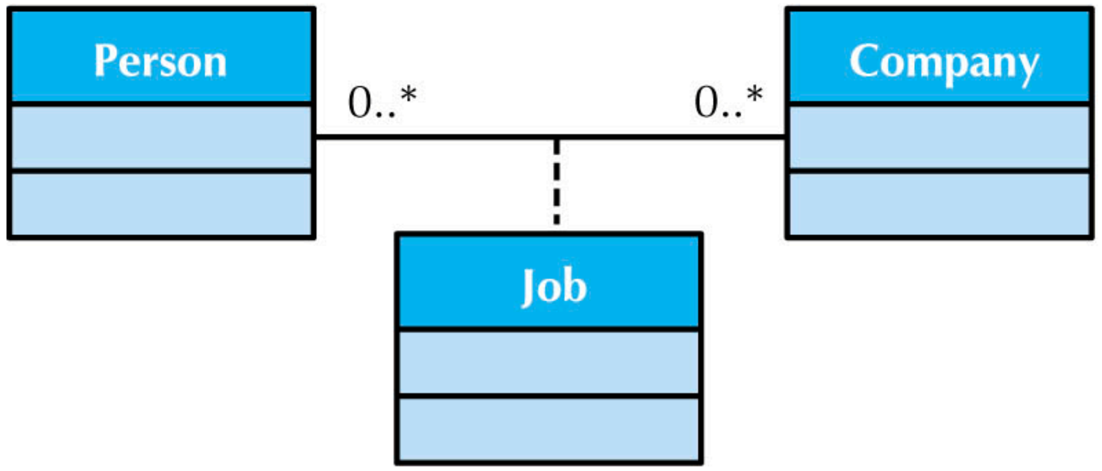


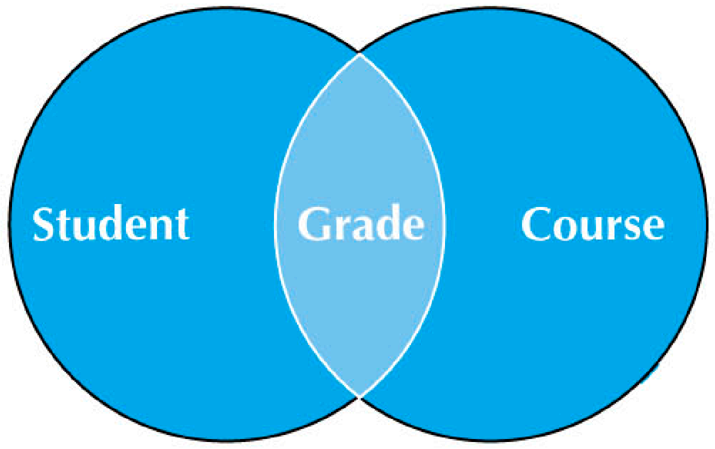
Multiple disjoint ranges:
An employee can be in 1 to 3 or 5 committees

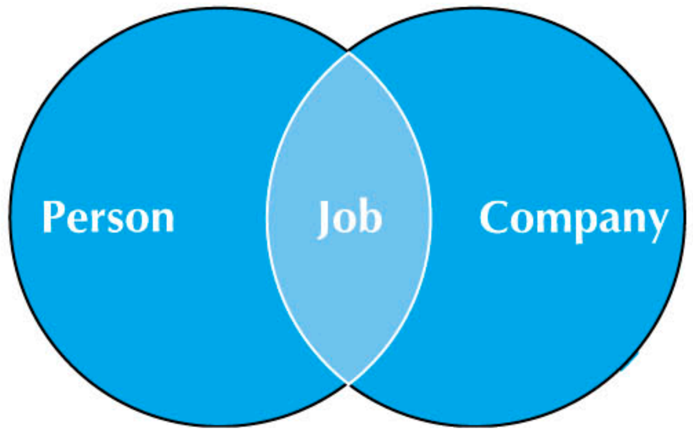
Association Class

When a relationship itself has associated properties, especially in many to many relationship, create an asscociation class



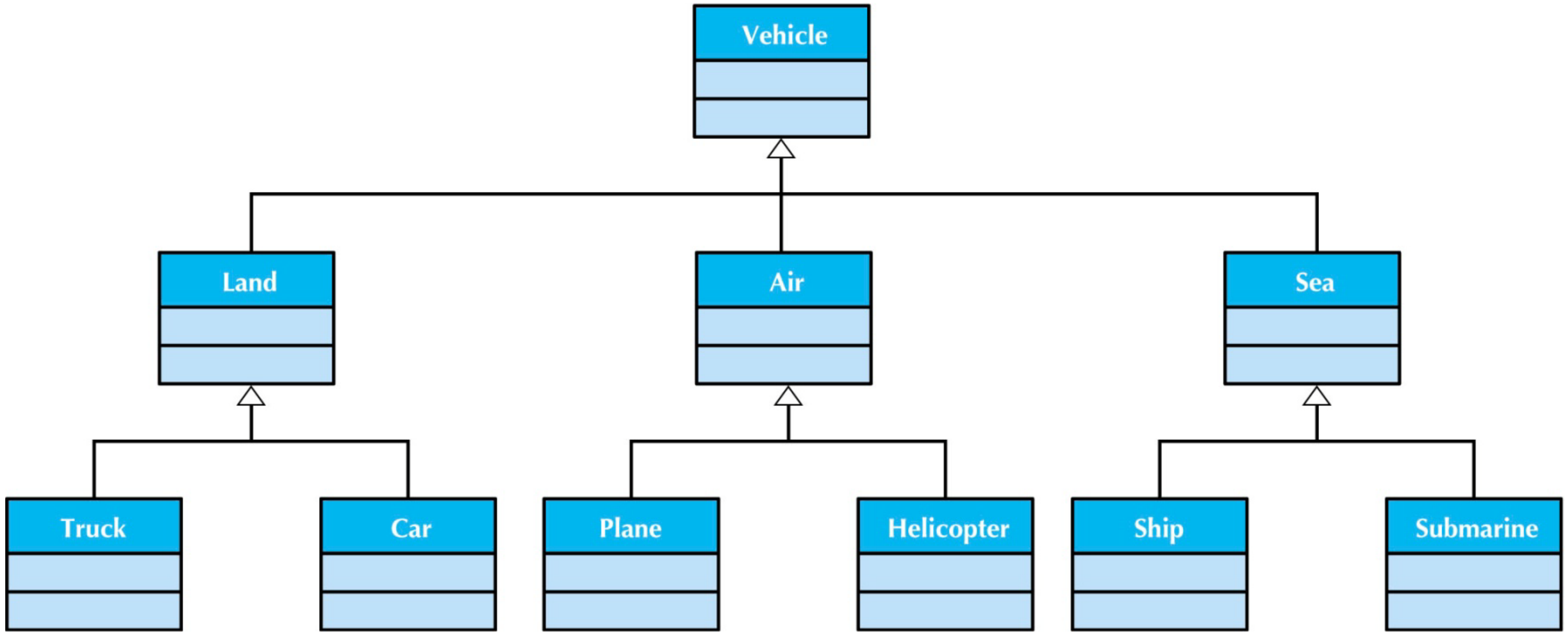




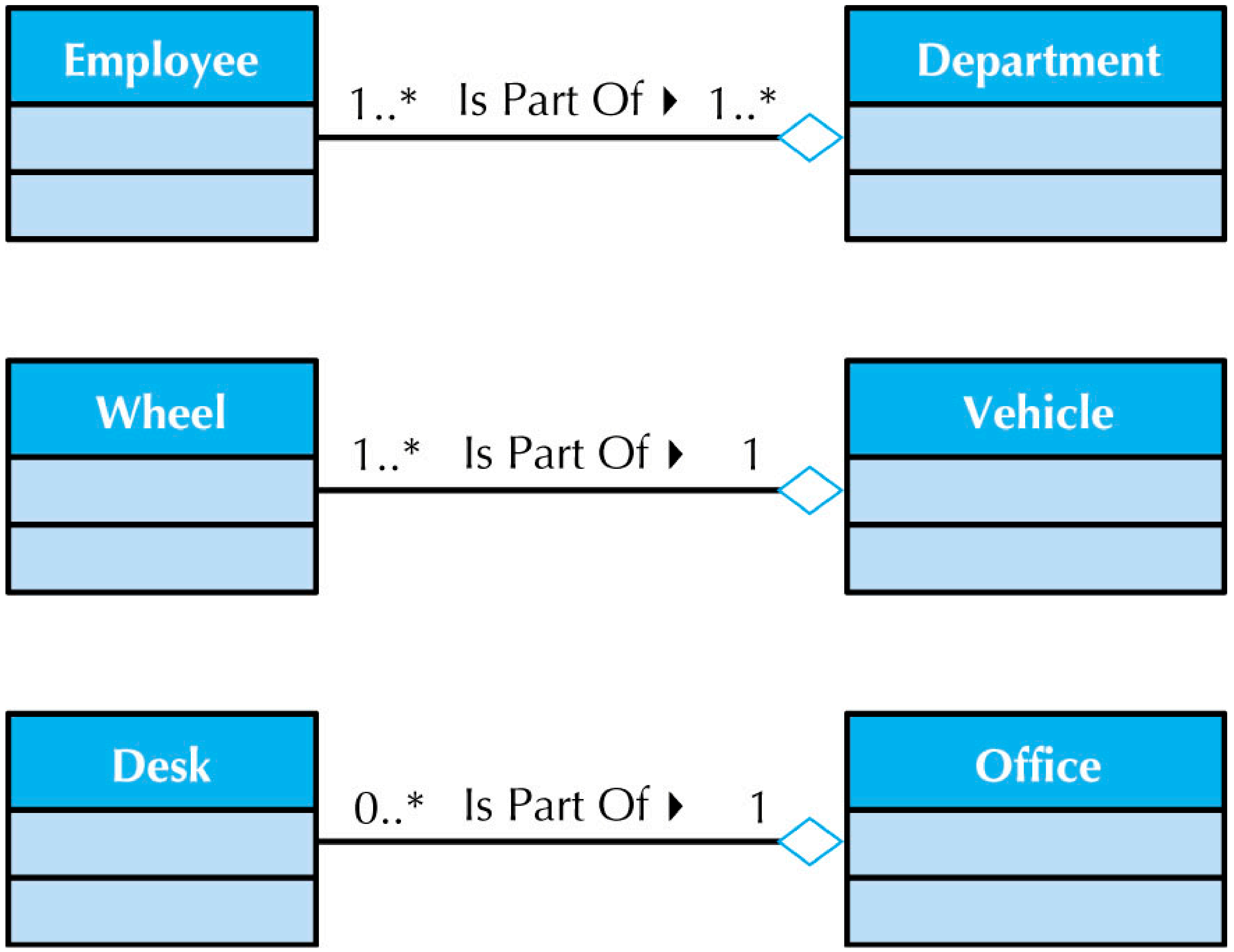


Generalization Relationship

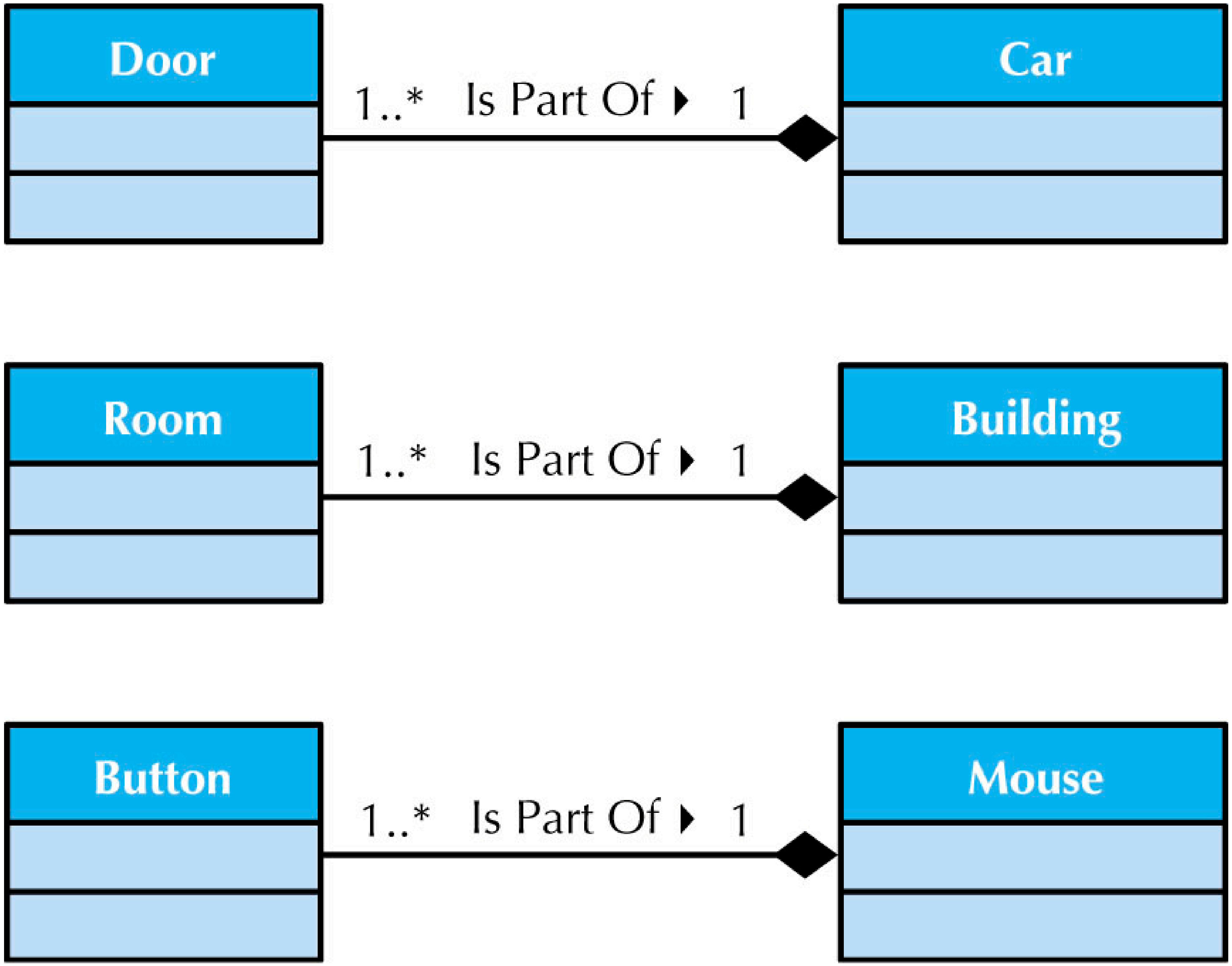
- *There are subclass and superclass. 'Is A' Relationship*
- *The properties and operations of superclass are also valid for objects of the subclass*
- *Subclass has specific properties or operations different from superclass*



Aggregation Relationship



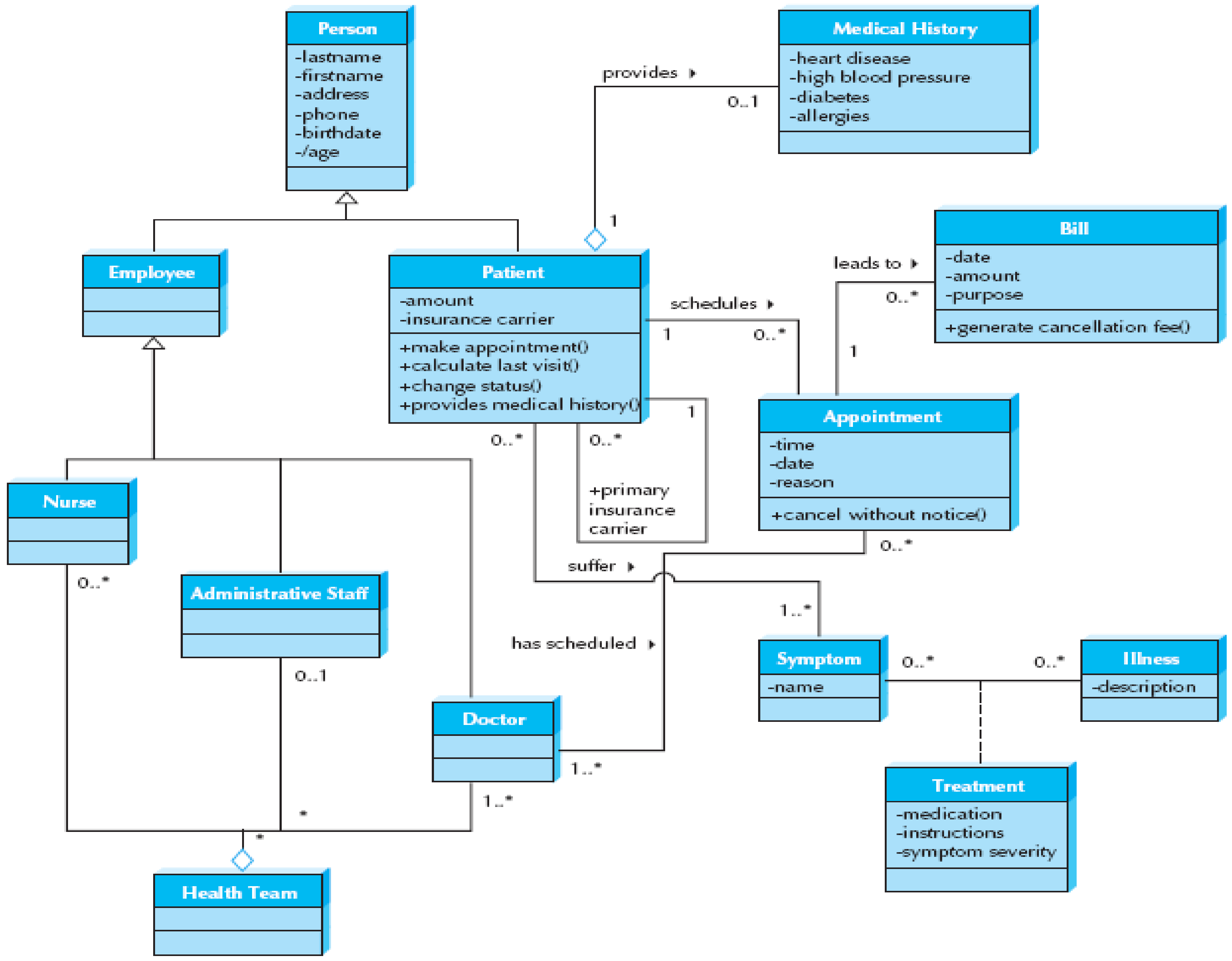
Compositition Relationship



Agregation vs Compositition

- **Composition is strong aggregation**
- **Aggregation** implies a relationship where the part can exist independently of the whole. Example: Departement (whole) and Employee (part). Delete the Department and the Employee still exist.
- **Composition** implies a relationship where the part cannot exist independent of the whole. Example: Building (whole) and Room (part). Rooms don't exist separate to a Building. If we delete the building, the room is also deleted.

Sample Class Diagram

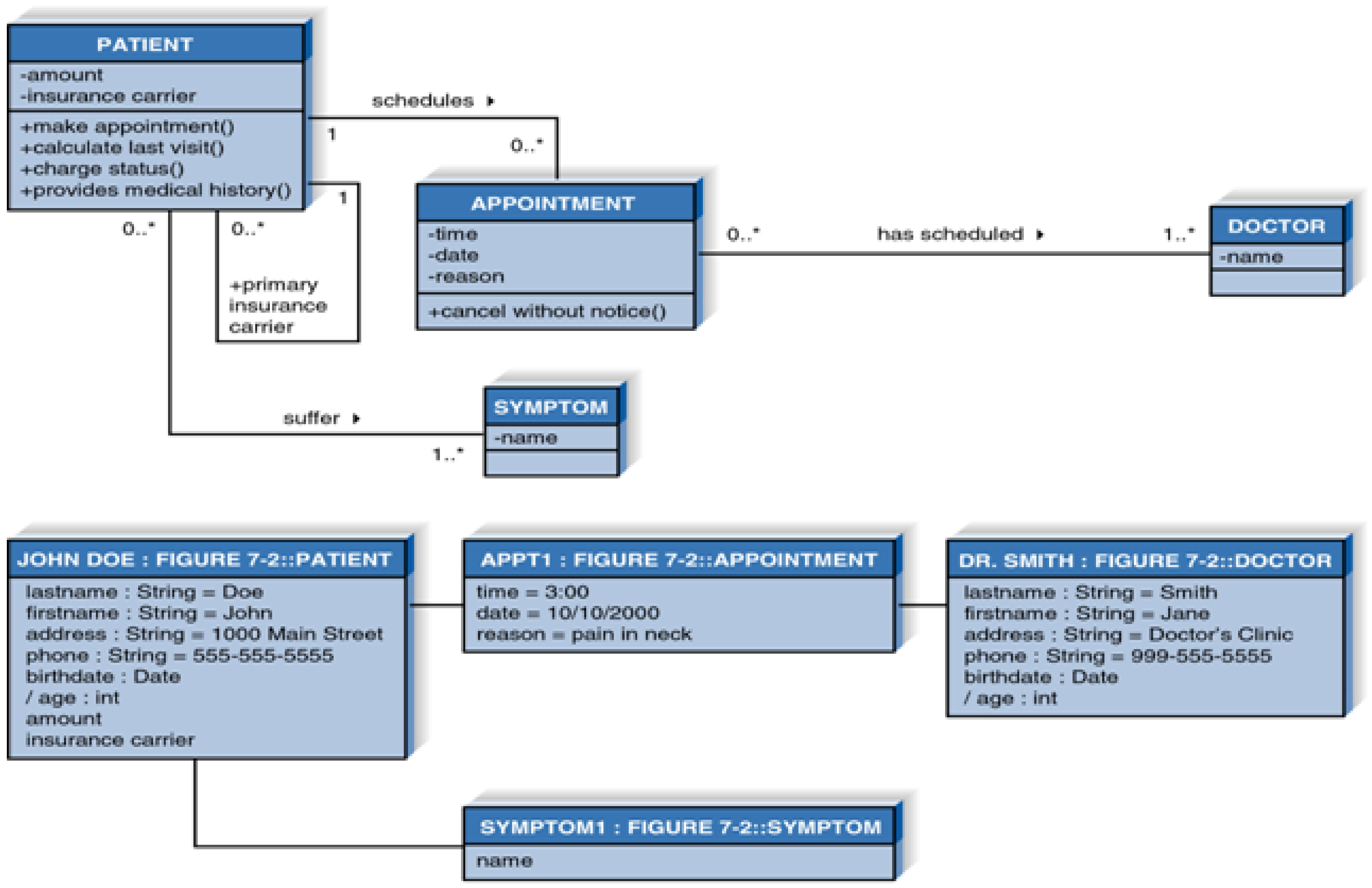


Simplifying Class Diagram

- Fully populated class diagrams of real-world system can be difficult to understand
- Common ways of simplifying class diagrams:
  - Show only concrete classes
  - The view mechanism shows a subset of classes
  - Packages show aggregations of classes (or any elements in UML)

Object Diagram

Create an instance of the class with a set of appropriate attribute values

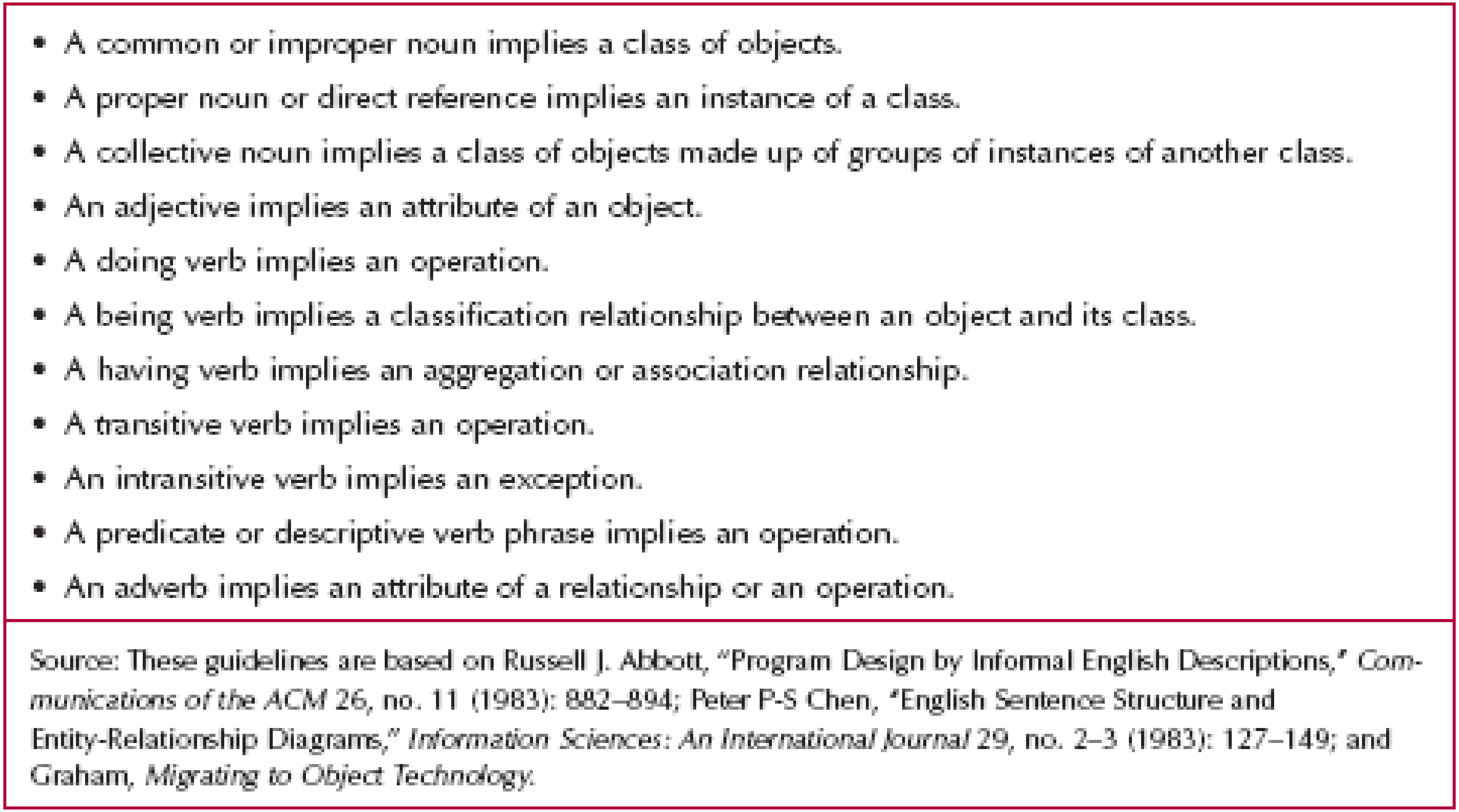


CREATING CRC CARDS AND CLASS DIAGRAMS

Object Identification

- Textual analysis of use-case information
  - Nouns suggest classes
  - Verbs suggest operations
- Creates a rough first cut
- Common object list
- Incidents
- Roles

Textual Analysis Guideline



- A common or improper noun implies a class of objects.
- A proper noun or direct reference implies an instance of a class.
- A collective noun implies a class of objects made up of groups of instances of another class.
- An adjective implies an attribute of an object.
- A doing verb implies an operation.
- A being verb implies a classification relationship between an object and its class.
- A having verb implies an aggregation or association relationship.
- A transitive verb implies an operation.
- An intransitive verb implies an exception.
- A predicate or descriptive verb phrase implies an operation.
- An adverb implies an attribute of a relationship or an operation.

Source: These guidelines are based on Russell J. Abbott, "Program Design by Informal English Descriptions," *Communications of the ACM* 26, no. 11 (1983): 882–894; Peter P-S Chen, "English Sentence Structure and Entity-Relationship Diagrams," *Information Sciences: An International Journal* 29, no. 2–3 (1983): 127–149; and Graham, *Migrating to Object Technology*.

Object Identification

2. Common object list

- Physical/tangible (books, chairs)
- Incidents (meetings, flight)
- Roles (doctor, nurse)
- Interactions (sales transaction)

Patterns

- Useful groupings of classes that recur in various situations
- Transactions
  - Transaction class
  - Transaction line item class
  - Item class
  - Location class
  - Participant class

7 Steps to Structural Models

1. Create CRC cards by performing textual analysis on the use-cases.
2. Brainstorm additional candidate classes, attributes, operations, and relationships by using the common object list approach.
3. Role-play each use-case using the CRC cards.
4. Create the class diagram based on the CRC cards.
5. Review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.
6. Incorporate useful patterns.
7. Review the structural model.

Summary

- Structural Models
- CRC Cards
- Class Diagrams
- Creating CRC Cards and Class Diagrams

Q & A