# Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

# Chapter ۵
# Structural Modeling

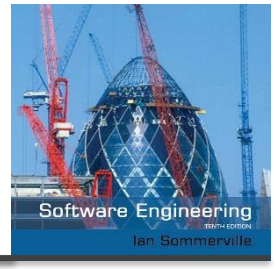
# Steps(I)

1. Preparing proposal
2. Requirements determination
   - User story
3. Abstract Business Process Modelling
4. Analysis
   - Functional Modelling
   - Structural Modelling
   - Behavioral Modelling

# Steps(II)

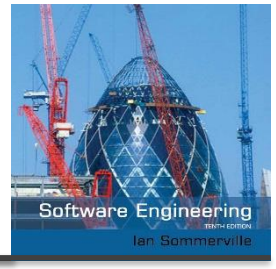
5. Design
   - Optimization
   - Database Management
   - User Interface
   - Physical Architecture

# System modeling



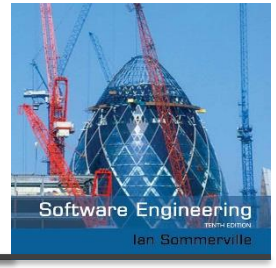

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

# System perspectives




- An external perspective, where you model the context or environment of the system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

# UML diagram types




- Use case diagrams, which show the interactions between a system and its environment.
- Activity diagrams, which show the activities involved in a process or in data processing .
- Class diagrams, which show the object classes in the system and the associations between these classes.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- State diagrams, which show how the system reacts to internal and external events.

# Structural modeling

- Supports the creation of an internal structural or static view of a business information system in that it shows how the system is structured to support the underlying business processes.
- A *structural model* is a formal way of representing the objects that are used and created by a business system. It illustrates people, places, or things about which information is captured and how they are related to one another.
- The structural model is drawn using an iterative process in which the model becomes more detailed over time.

# Structural modelling

- In analysis, analysts draw a *conceptual model*, which shows the logical organization of the objects without indicating how the objects are stored, created, or manipulated. Because this model is free from any implementation or technical details, the analysts can focus more easily on matching the model to the real business requirements of the system.
- In design, analysts evolve the conceptual structural model into a design model that reflects how the objects will be organized in databases and software. At this point, the model is checked for redundancy, and the analysts investigate ways to make the objects easy to retrieve.

# Structural Models

- The goal of the analyst is to discover the **key objects** contained in the problem domain and to build a structural model.
- Basic elements of structural models are classes, attributes, operations, and relationships.

# Class

- A *class* is a general template that we use to create specific instances, or *objects*, in the problem domain.
- All objects of a given class are identical in structure and behavior but contain different data in their attributes.
- There are two general kinds of classes of interest during analysis: concrete and abstract.
  - *Concrete classes* are used to create objects.
  - *Abstract classes* do not actually exist in the real world; they are simply useful abstractions.

# A second classification of classes

- Is the type of real-world thing that a class represents.
  - domain classes,
  - user-interface classes,
  - data structure classes,
  - file structure classes,
  - operating environment classes,
  - document classes,
  - ….

# Class

- An *attribute* of an analysis class represents a piece of information that is relevant to the description of the class within the application domain of the problem being investigated.
- The behavior of an analysis class is defined in an *operation* or service. In later phases, the operations are converted to *methods*.

# Object Identification

- **Textual Analysis**
- **Brainstorming**
- **Common Object Lists**
- **Patterns**

# CRC Cards

- *CRC (Class–Responsibility–Collaboration) cards* are used to document the responsibilities and collaborations of a class.
- *Responsibilities* of a class can be broken into two separate types: **knowing** and **doing**.
  - *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 relationships.
  - *Doing responsibilities* are those things that an instance of a class must be capable of doing. In this case, an instance of a class can execute its operations.

# CRC Cards(Cnt’d)

- *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.

# Elements of a CRC Card

- The front of the card contains the class's name, ID, type, description, associated use cases, responsibilities, and collaborators.
- The back of a CRC card contains the attributes and relationships of the class. The attributes of the class represent the knowing responsibilities that each instance of the class has to meet.

**Front:**

| **Class Name:** Old Patient | **ID:** 3 | **Type:** Concrete, Domain |
|---|---|---|
| **Description:** An individual who 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:**

**Attributes:**

| | |
|---|---|
| Amount (double) | |
| Insurance carrier (text) | |
| | |
| | |

**Relationships:**

**Generalization (a-kind-of):** Person

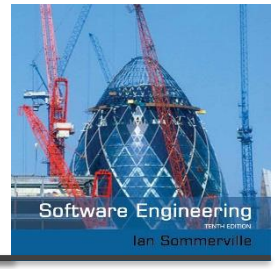
**Aggregation (has-parts):** Medical History

**Other Associations:** Appointment

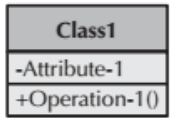
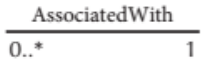
# Class Diagrams

- A *class diagram* is a *static model* that shows the classes and the relationships among classes that remain constant in the system over time.
- Elements of a Class Diagram
  - Class: The main building block of a class diagram is the class, which stores and manages information in the system. Visibility relates to the level of information hiding to be enforced for the attribute. Visibility of an attribute can be public (+), protected (#), or private (−).

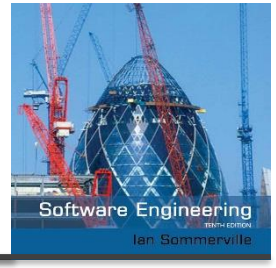
# Class diagrams

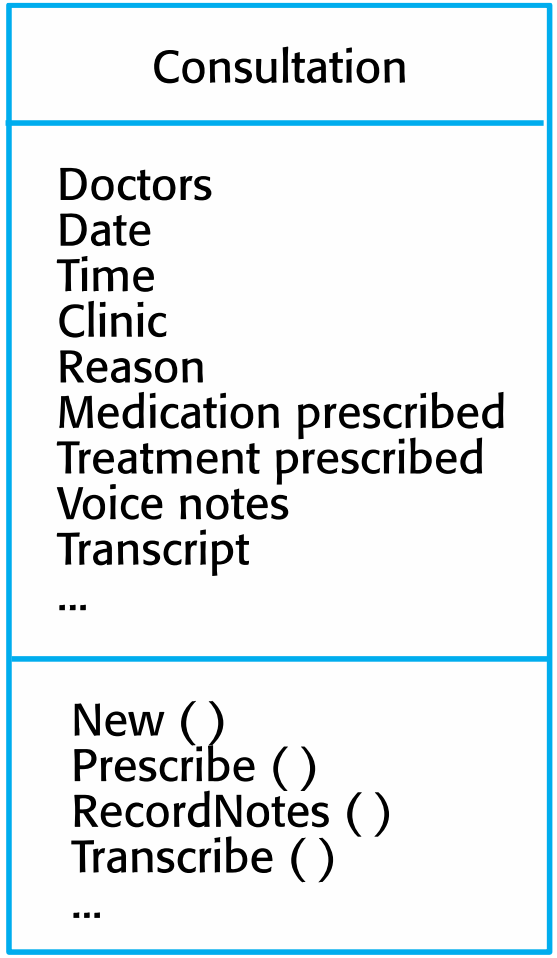



- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

| | |
|---|---|
| **A class:**<br>• Represents a kind of person, place, or thing about which the system will need to capture and store information.<br>• Has a name typed in bold and centered in its top compartment.<br>• Has a list of attributes in its middle compartment.<br>• Has a list of operations in its bottom compartment.<br>• Does not explicitly show operations that are available to all classes. |  |
| **An attribute:**<br>• Represents properties that describe the state of an object.<br>• Can be derived from other attributes, shown by placing a slash before the attribute's name. | attribute name<br>/derived attribute name |
| **An operation:**<br>• Represents the actions or functions that a class can perform.<br>• Can be classified as a constructor, query, or update operation.<br>• Includes parentheses that may contain parameters or information needed to perform the operation. | operation name () |
| **An association:**<br>• Represents a relationship between multiple classes or a class and itself.<br>• Is labeled using a verb phrase or a role name, whichever better represents the relationship.<br>• Can exist between one or more classes.<br>• Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance. |  |
| **A generalization:**<br>• Represents a-kind-of relationship between multiple classes. |  |
| **An aggregation:**<br>• Represents a logical a-part-of relationship between multiple classes or a class and itself.<br>• Is a special form of an association. |  |
| **A composition:**<br>• Represents a physical a-part-of relationship between multiple classes or a class and itself<br>• Is a special form of an association. |  |

# The Consultation class



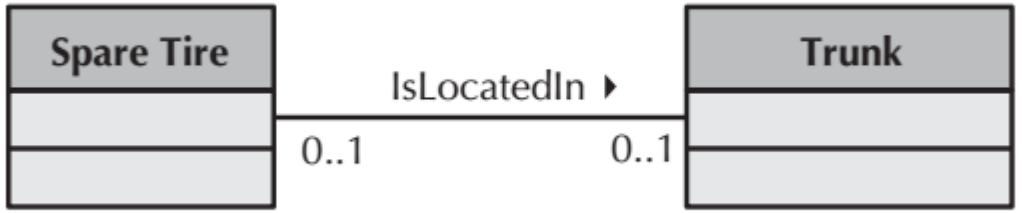



# Relationships

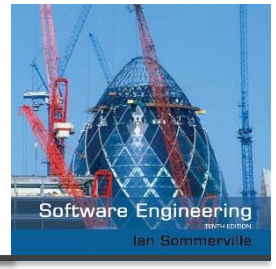
- **Generalization**
  - Enables the analyst to create classes that inherit attributes and operations of other classes. The subclasses inherit the attributes and operations of their superclass and can also contain attributes and operations that are unique just to them.
  - Is represented with the *a-kind-of* relationship, so that we say that an employee is a-kind-of person.
- **Aggregation**
  - Relate *parts* to *wholes.* For our purposes, we use the *a-part-of* or *has-parts* semantic relationship to represent the aggregation abstraction. For example, a door is a-part-of a car.

# Relationships(Cnt'd)

- **Association**
  - There are other types of relationships that do not fit neatly into a generalization (a-kind-of) or aggregation (a-part-of) framework.
  - Thus, they are simply considered to be *associations* between instances of classes.

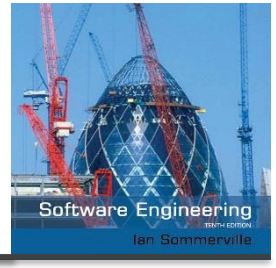


# Generalization



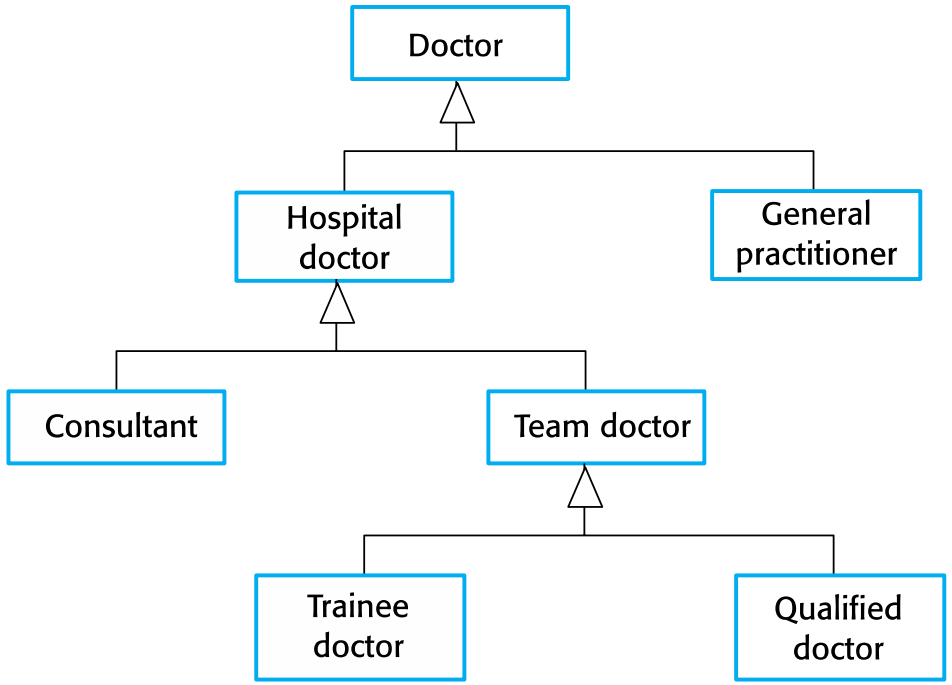

- ✧ Generalization is an everyday technique that we use to manage complexity.
- ✧ Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- ✧ This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents.

# Generalization

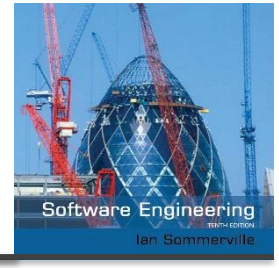


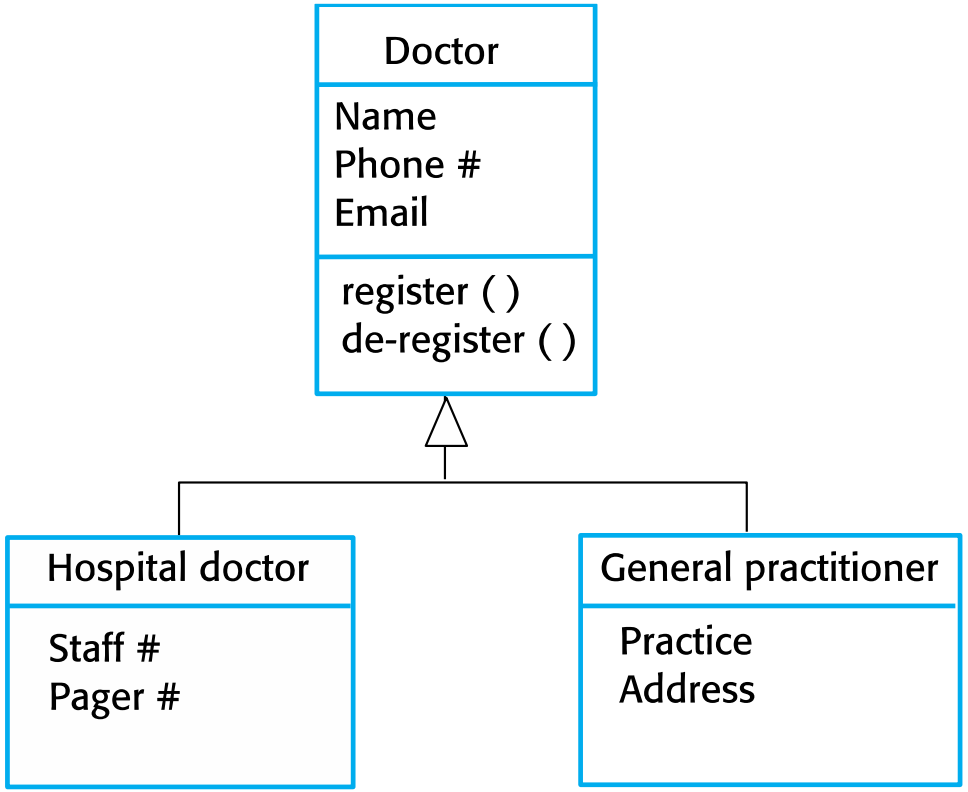

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language.
- In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.

# A generalization hierarchy

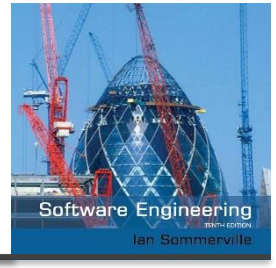


# A generalization hierarchy with added detail



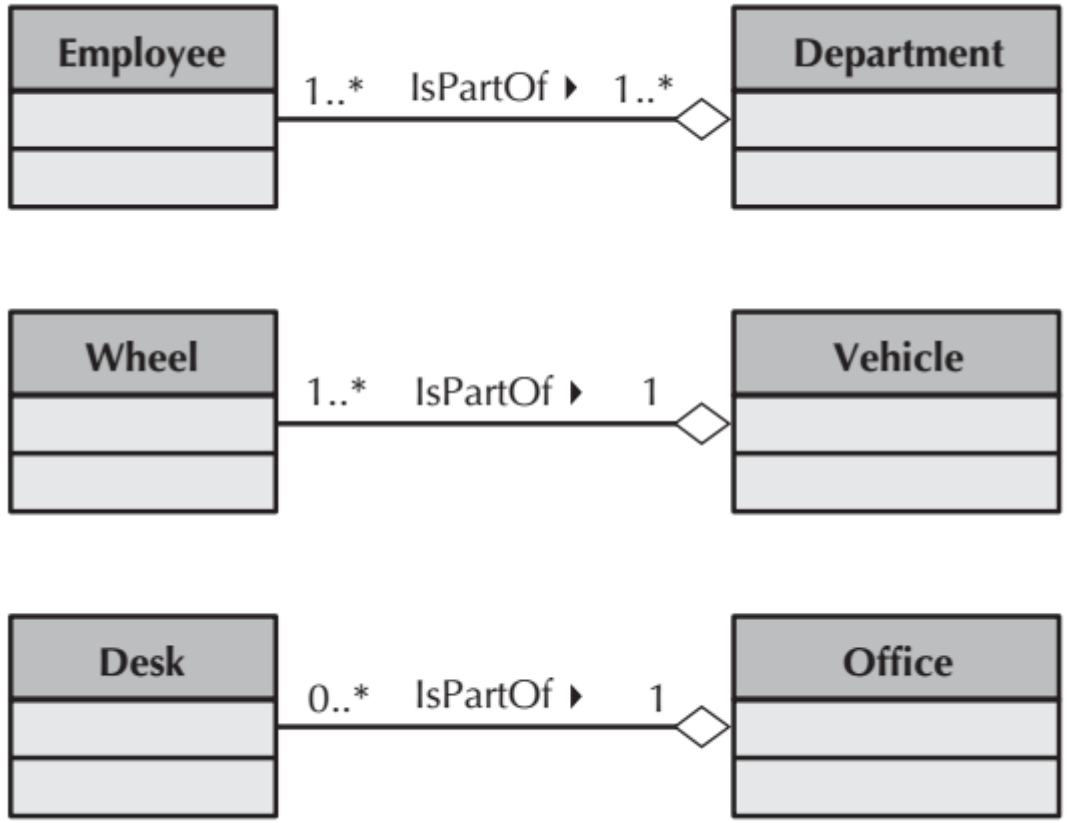



# Object class aggregation models

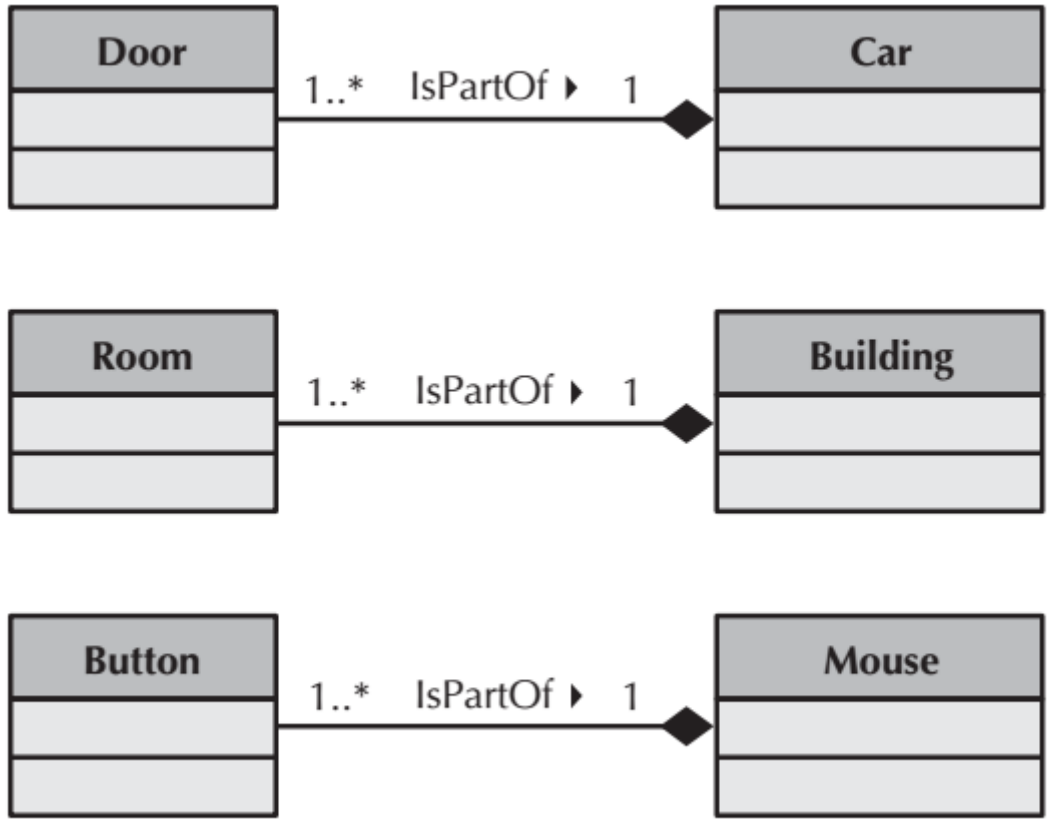


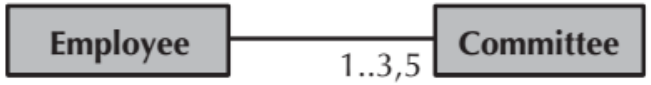

- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.

# Sample Aggregation Associations

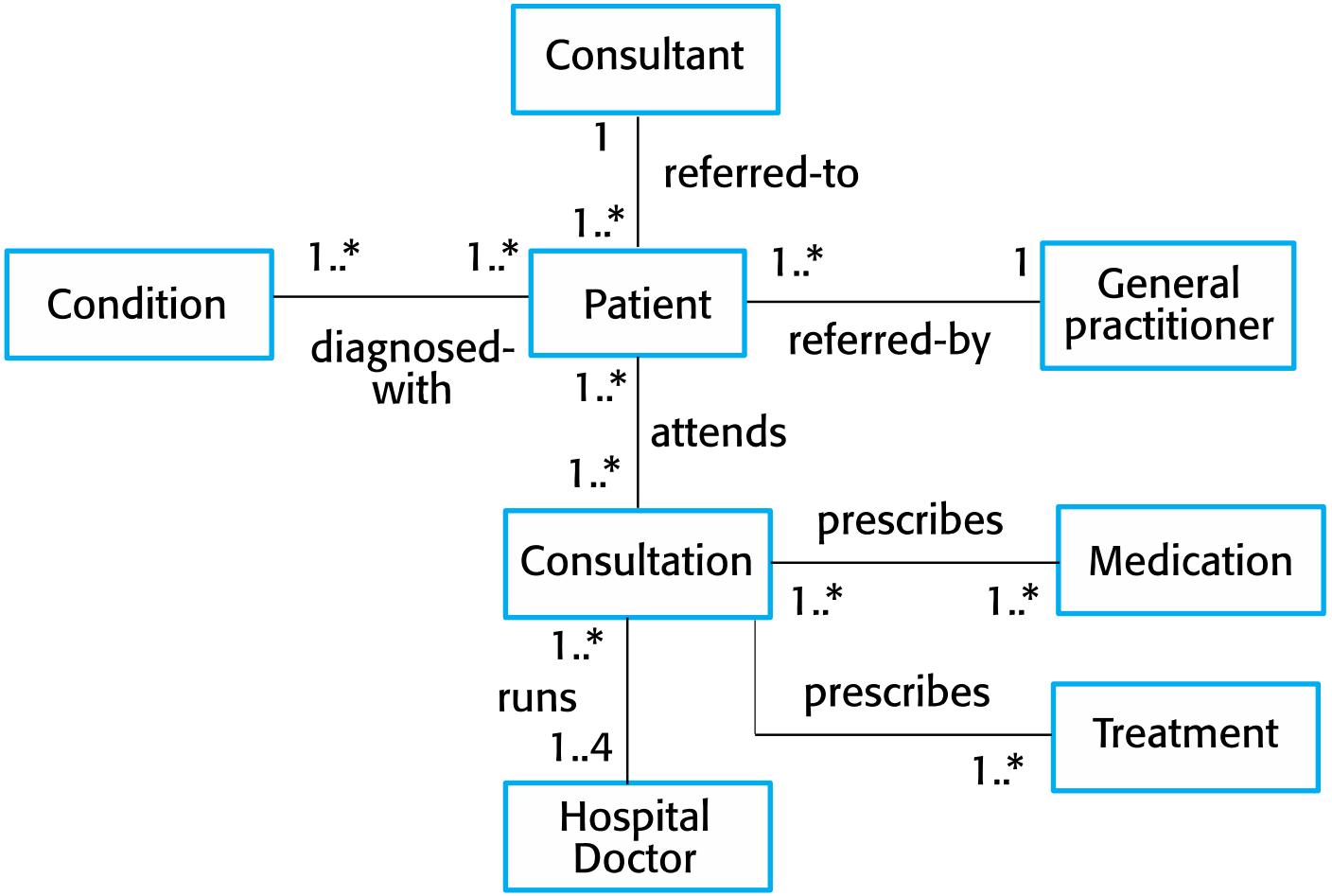


# Sample Composition Associations



| | | | |
|---|---|---|---|
| Exactly one | 1 |  | A department has one and only one boss. |
| Zero or more | 0..* |  | An employee has zero to many children. |
| One or more | 1..* |  | A boss is responsible for one or more employees. |
| Zero or one | 0..1 |  | An employee can be married to zero or one spouse. |
| Specified range | 2..4 |  | An employee can take from two to four vacations each year. |
| Multiple, disjoint ranges | 1..3,5 |  | An employee is a member of one to three or five committees. |

# Classes and associations in the MHC-PMS



# Verifying And Validating The Structural Model

- Accomplished during a formal review meeting using a walkthrough approach in which an analyst presents the model to a team of developers and users. The analyst walks through the model, explaining each part of the model and all the reasoning behind the decision to include each of the classes in the structural model.

# Verifying And Validating The Structural Model(Cnt’d)

- Test the consistency within the structural models.
  - First, every CRC card should be associated with a class on the class diagram, and vice versa.
  - Second, the responsibilities listed on the front of the CRC card must be included as operations in a class on a class diagram, and vice versa.
  - Third, collaborators on the front of the CRC card imply some type of relationship on the back of the CRC card and some type of association that is connected to the associated class on the class diagram.
  - Fourth, attributes listed on the back of the CRC card must be included as attributes in a class on a class diagram, and vice versa.
  - Fifth, the object type of the attributes listed on the back of the CRC card and with the attributes in the attribute list of the class on a class diagram implies an association from the class to the class of the object type.

# Verifying And Validating The Structural Model(Cnt’d)

- Test the consistency within the structural models.
  - Sixth, the relationships included on the back of the CRC card must be portrayed using the appropriate notation on the class diagram.
  - Seventh, an association class should be created only if there is indeed some unique characteristic (attribute, operation, or relationship) about the intersection of the connecting classes.

# What should you do for your project?

1. Create class diagram.

*We will work in the lab.*

# Reference


- **Dennis, Wixon, Tegarden**, "**System Analysis and Design, An Object Oriented Approach with UML", 5th Edition, 2015.**