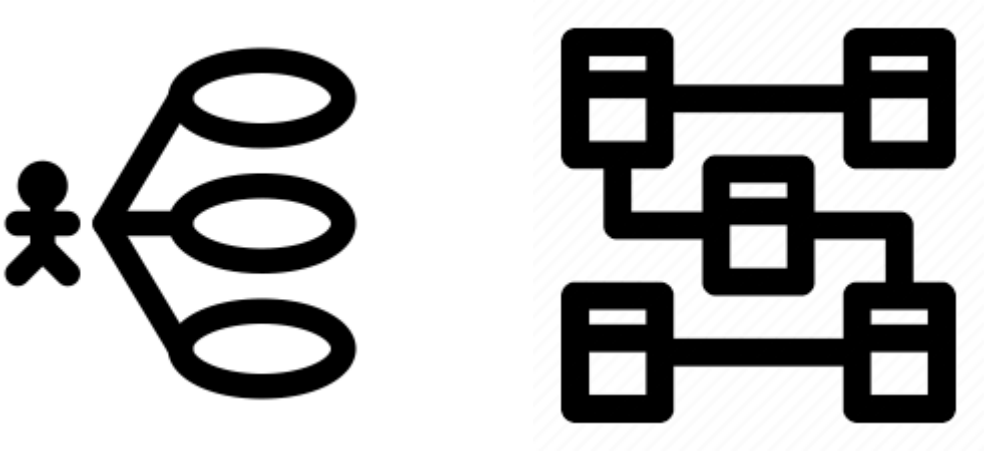
System Analysis and Design

System Modelling

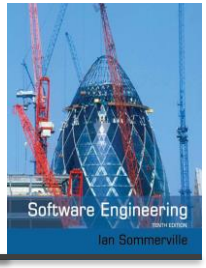


Prof. Rania Elgohary
rania.elgohary@cis.asu.edu.eg
Dr. Yasmine Afify
yasmine.afify@cis.asu.edu.eg

System modelling

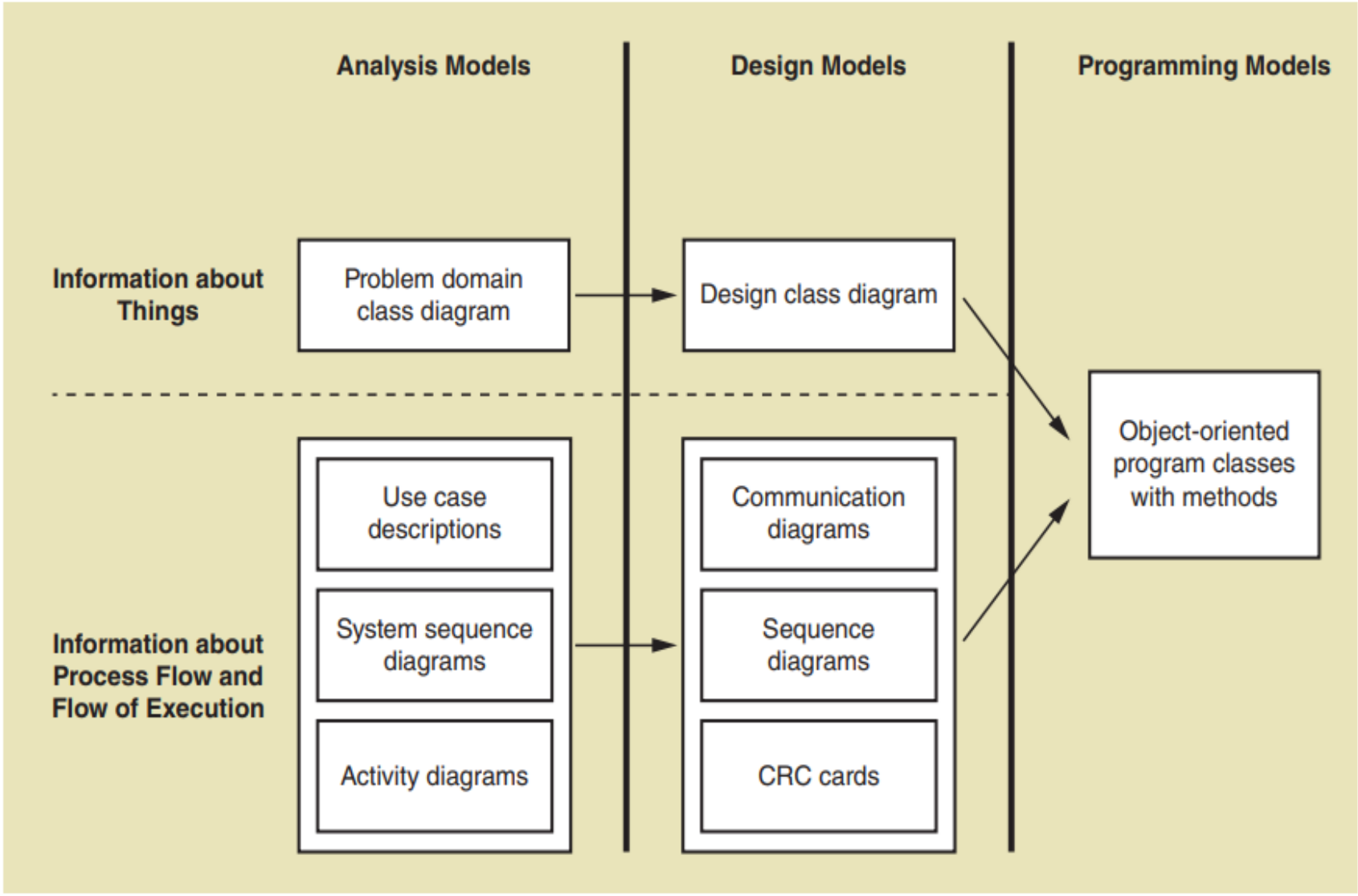
- System modelling is the process of developing abstract models of a system, with each model presenting a different perspective of that system.
- It is possible to generate a complete or partial system implementation from the system model.
- Unified Modelling Language (UML) is a standard language for specifying, modeling, visualizing, constructing, and documenting the software systems.

System Modelling Benefits

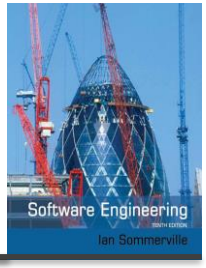


- Help project teams **communicate**
  - Communication between users, developers, analysts, testers, managers, and anyone else involved with a project
- **Explore** potential designs
- **Validate** the architectural design of the software
- Be **independent** of particular programming languages and development processes

FIGURE 12-3 *Analysis models to design models to programming models*



System perspectives



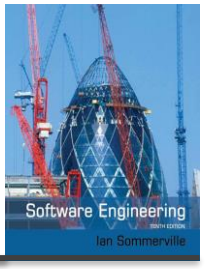
- An **external** perspective, where you model the context or environment of the system (context models)
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system (use case and sequence diagrams)
- A **structural** perspective, where you model the organization of a system (class diagram)
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events (state diagram)

UML diagram types

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

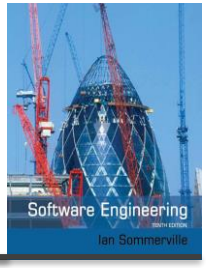
Use of graphical models

- As a means of facilitating discussion about a proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
  - Models should be an accurate representation of the system
  - May be incomplete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.



Interaction models

Interaction models

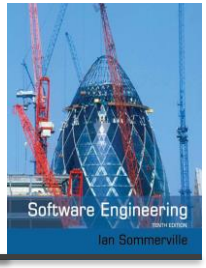


- Use case diagrams and sequence diagrams may be used for interaction modelling.
- Use case models and sequence diagrams present interaction at different levels of detail and so may be used together.
- The details of the interactions involved in a high-level use case may be documented in sequence diagrams.

Use case modelling

- Use cases were developed originally to support requirements elicitation from stakeholders who interact directly with the system.
- Each use case represents a discrete task that involves interaction with an external system.
- Use cases just focus on automated processes.
- Use Case diagrams aren't intended to show in which order the use cases are executed.
- Use case name should start with a verb.

Use cases



- Use case diagrams give a fairly simple overview of an interaction so you have to provide more detail to understand what is involved.
- Further details can either be:
  - a simple textual description
  - a structured description in a table
  - a sequence diagram

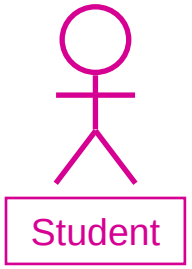
Diagrams in UML

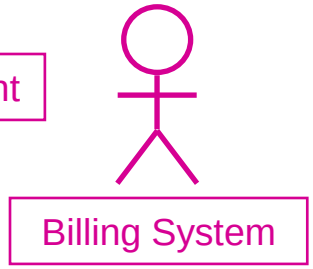
The university wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at login validation

Actors in Use Case Diagram

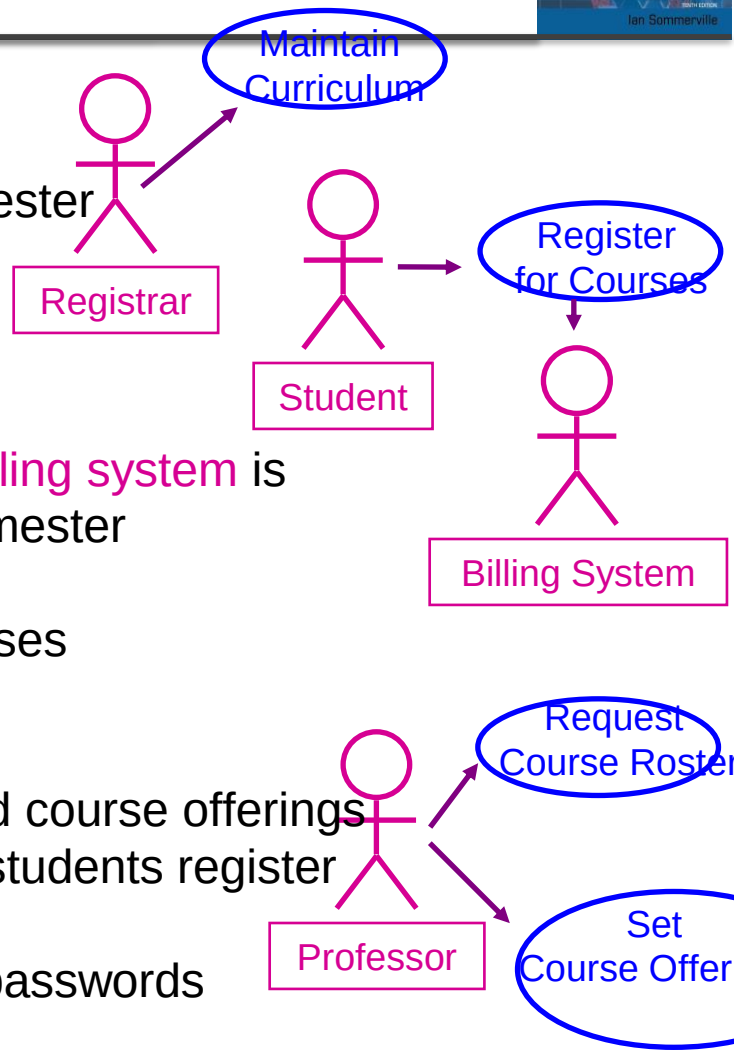
- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation



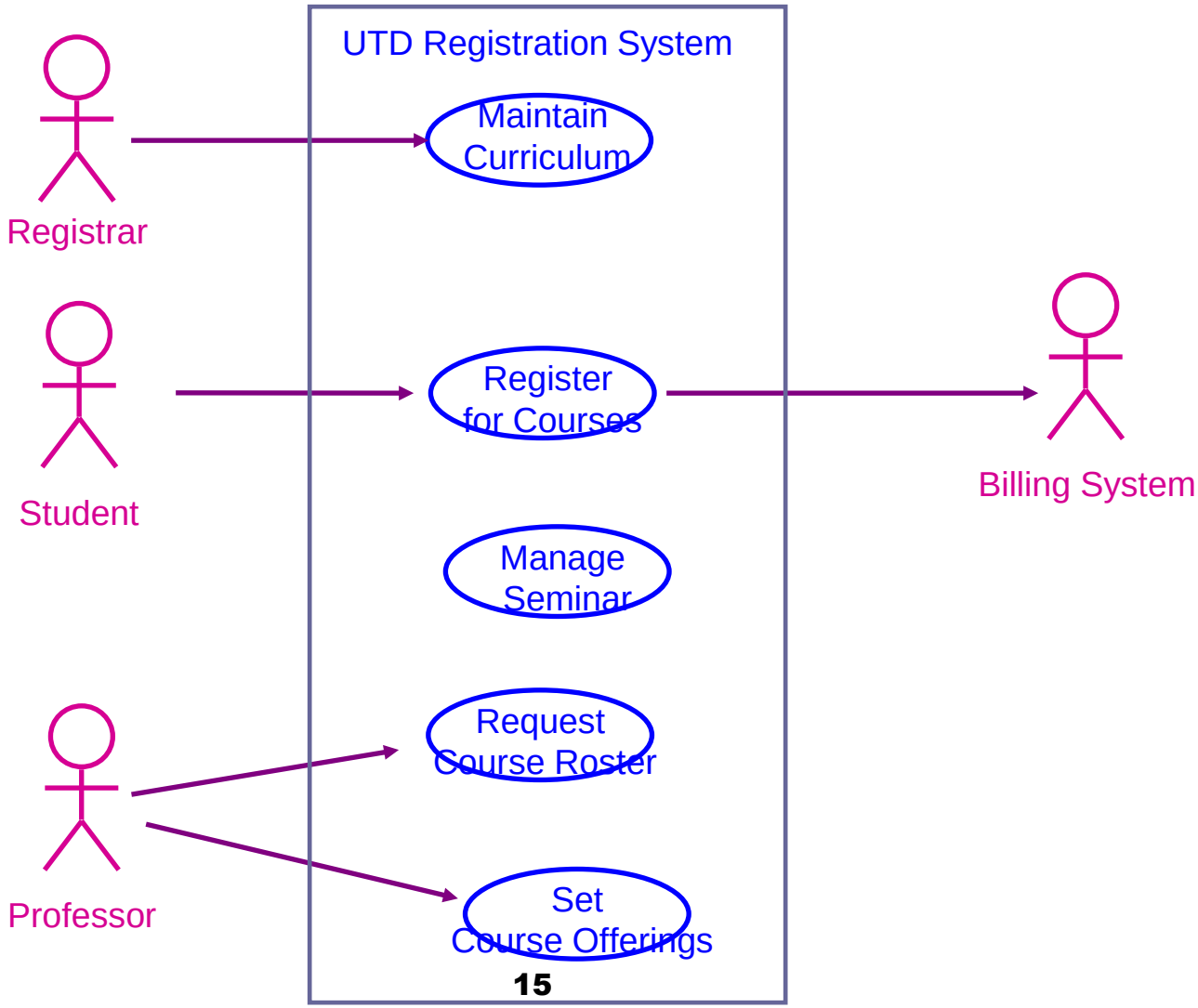


Actions in Use Case Diagram

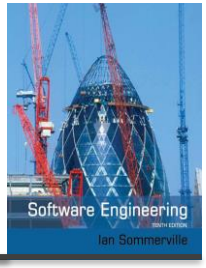
- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation



Use Case Diagram



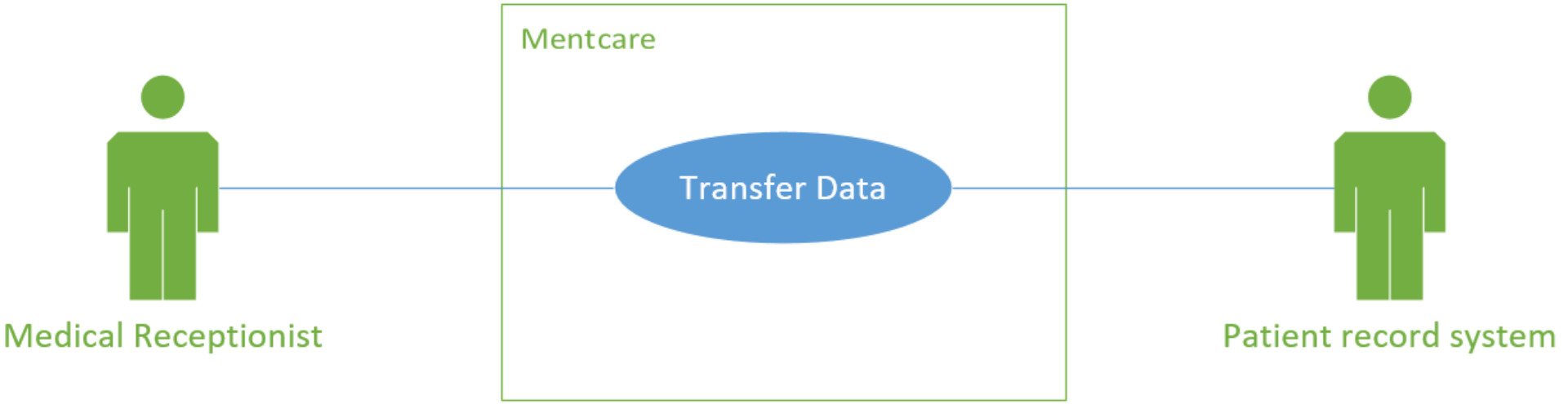
Use case actors



- There are three types of actors:
  - **Users** of the system (Physical person, or a user who will be directly using the system)
  - **Other systems** that will interact with the system being built
  - **Time** (Time becomes an actor when the passing of a certain amount of time triggers some event in the system (out of control))

Use case actor

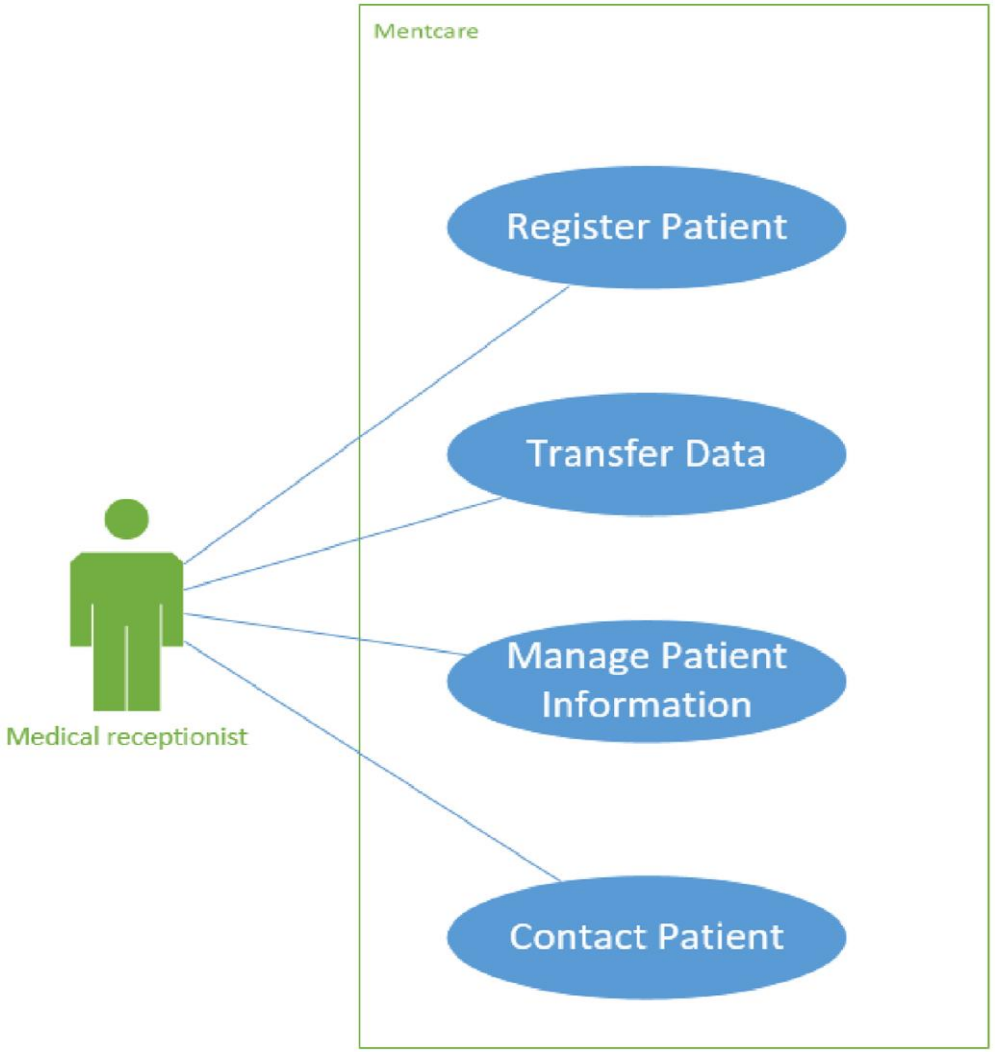
✧ A use case in the Mentcare system



Notice that there are two actors in this use case:
The operator who is transferring data and the patient record system.

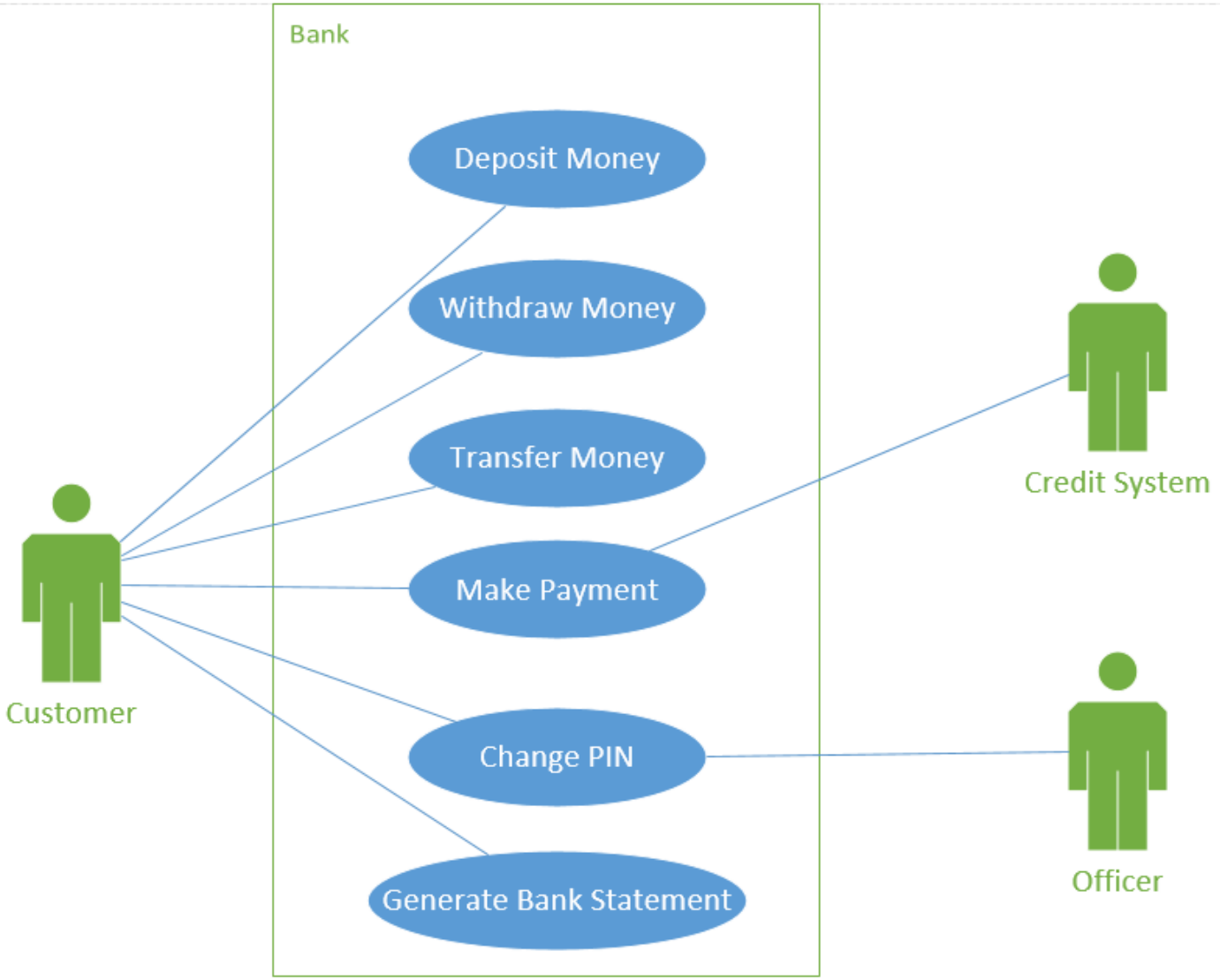
Use cases for actor ‘Medical Receptionist’

- The medical receptionist is responsible for patients registration, viewing their information and data transfer.
- In case of emergencies, he can contact the patients via sms.



Composite use case: Mentcare

Composite use case: Bank System



Relationships

- Association

the relationship between use case and an actor only

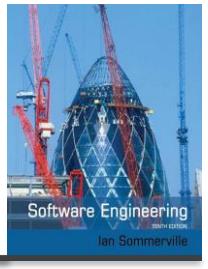
- Include
- Extend

only two types of relationships allowed between use cases

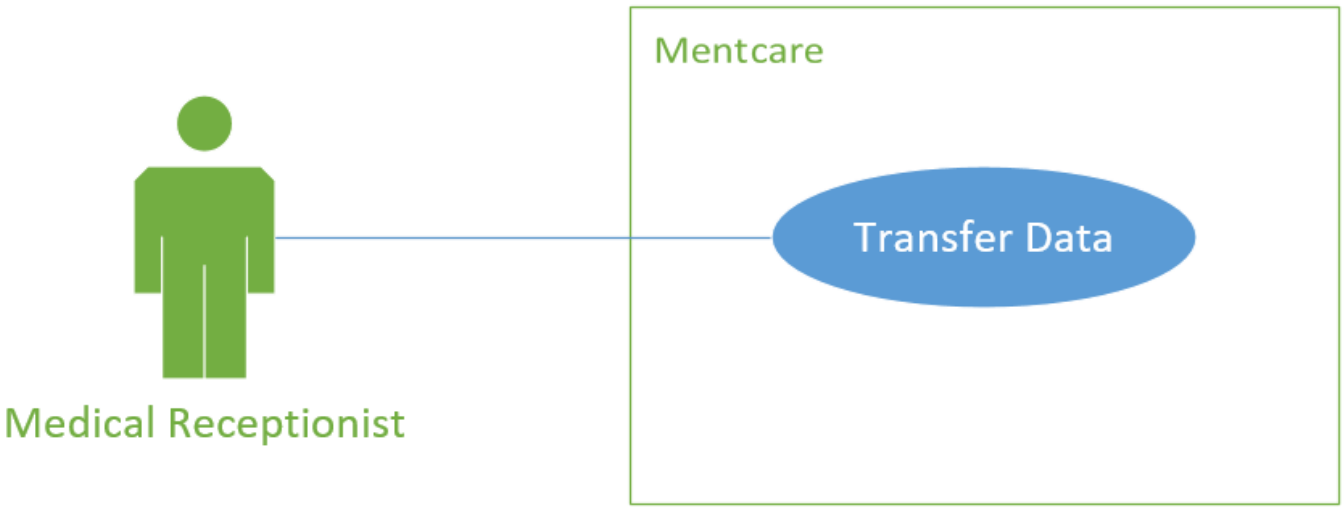
- Generalization

the only relationship allowed between actors

Association Relationship

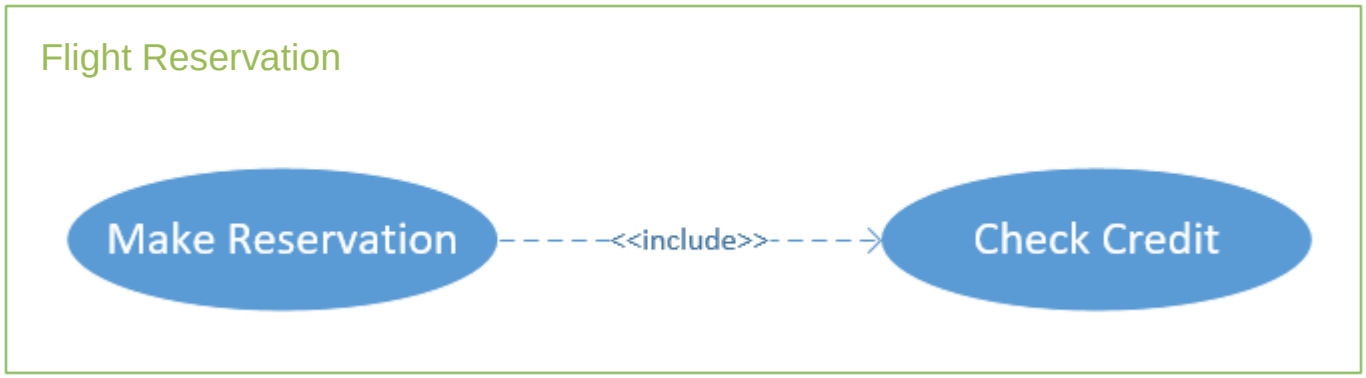


- Association relationship is used to show the relationship between a **use case** and **an actor only**
- Every use case **must be initiated** by an actor, with the **exception** of abstract use cases (in include and extend relationships)



Include Relationship

- Include relationship allows **one use case to use the functionality provided by another use case**
- This relationship can be used in one of two cases:
  - If two or more use cases have a **large piece of functionality that is identical**
  - A **single use case has an unusually large amount of functionality**
- An include relationship suggests that one use case **always** uses the functionality provided by another use case



While the “Make Reservation" use case is running, "Check Credit" must run.

Extend Relationship

- Extend relationship allows one use case **the option** to extend functionality provided by another use case

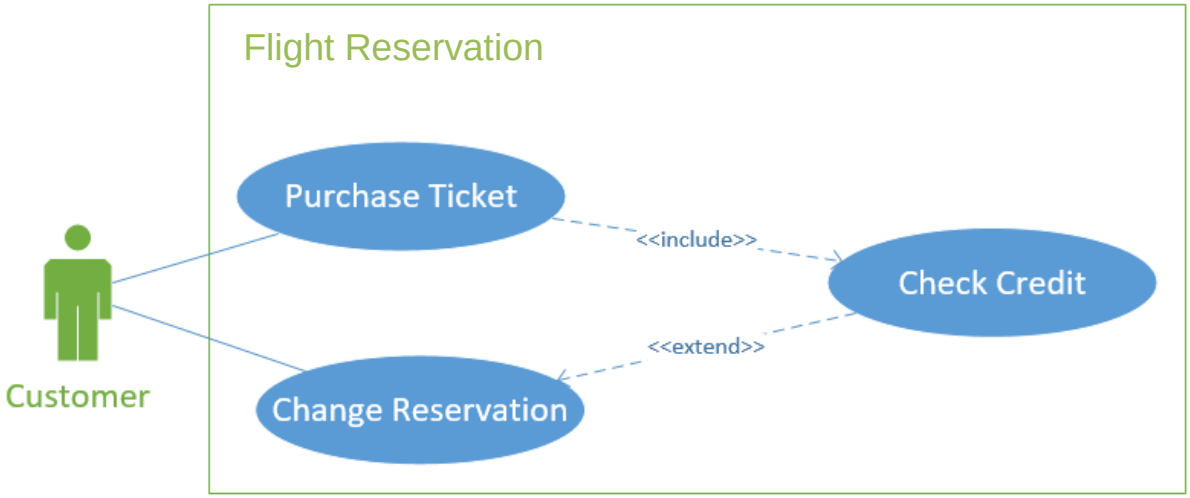
- While the "Change Reservation" use case is running, "Check Credit" runs if and only if the amount of the reservation has changed. If the amount has not changed, "Check Credit" does not need to run.
- The arrow is drawn:

from the use case that is optionally run ("Check Credit")

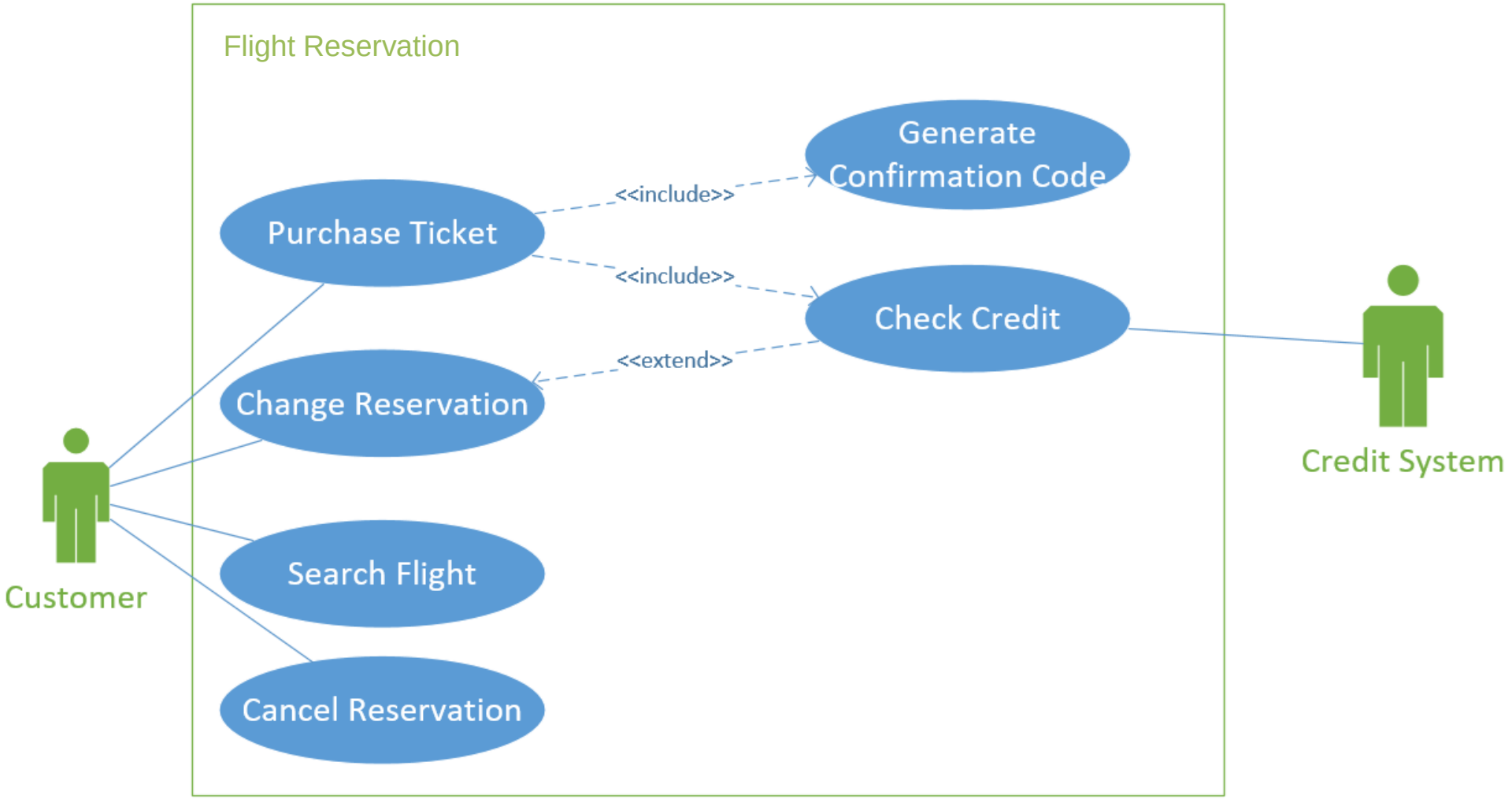
to the use case that is being extended ("Change Reservation")

Abstract use cases

- An abstract use case is **not started directly by an actor**
- Instead, an abstract use case provides **some additional functionality that can be used by other use cases**
- Abstract use cases are the use cases that **participate in an include or extend relationship**



Reservation system abstract use cases

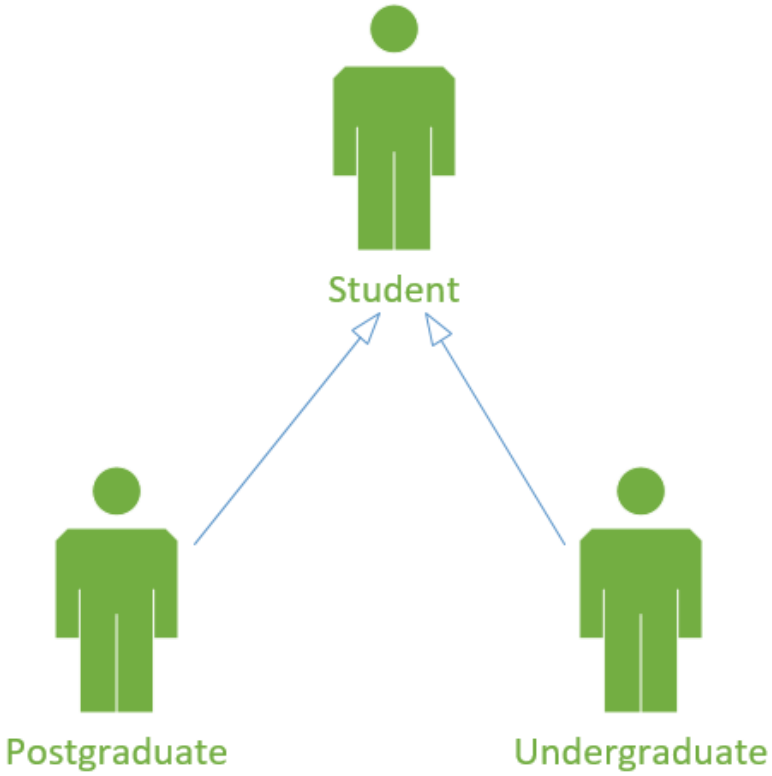


Generalization

- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes and learn the characteristics of these classes.
- The generalization is shown as an arrowhead pointing up to the more general class.
- The attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- Lower-level classes are sub classes, which inherit the attributes and operations from their super classes. These lower-level classes then add more specific attributes and operations.

Generalization

- For example, you may have two types of students.
- If the type A students will be initiating some use cases that type B students will not, it's probably worth including the actor generalizations.
- If both types of students use the same use cases, it's probably not necessary to show an actor generalization.



Functionalities

- Every **use case** should represent a **main functionality**

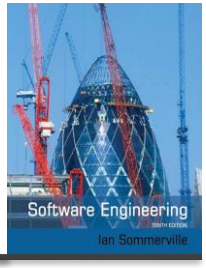
  So, we can't list "select seat", "enter user name" or "choose category" as use cases

    Because they are steps of a certain use cases ("Make Reservation", "Register" or "Search for Products") rather than being main functionalities

- We **can't create** a use case with a **vague name** like "enter data in database", "choose" , "save data" or "manage data"

- Don't create "Add course", "delete course" and "update course" as 3 use cases but create one use case called "Manage course data"

Relationships

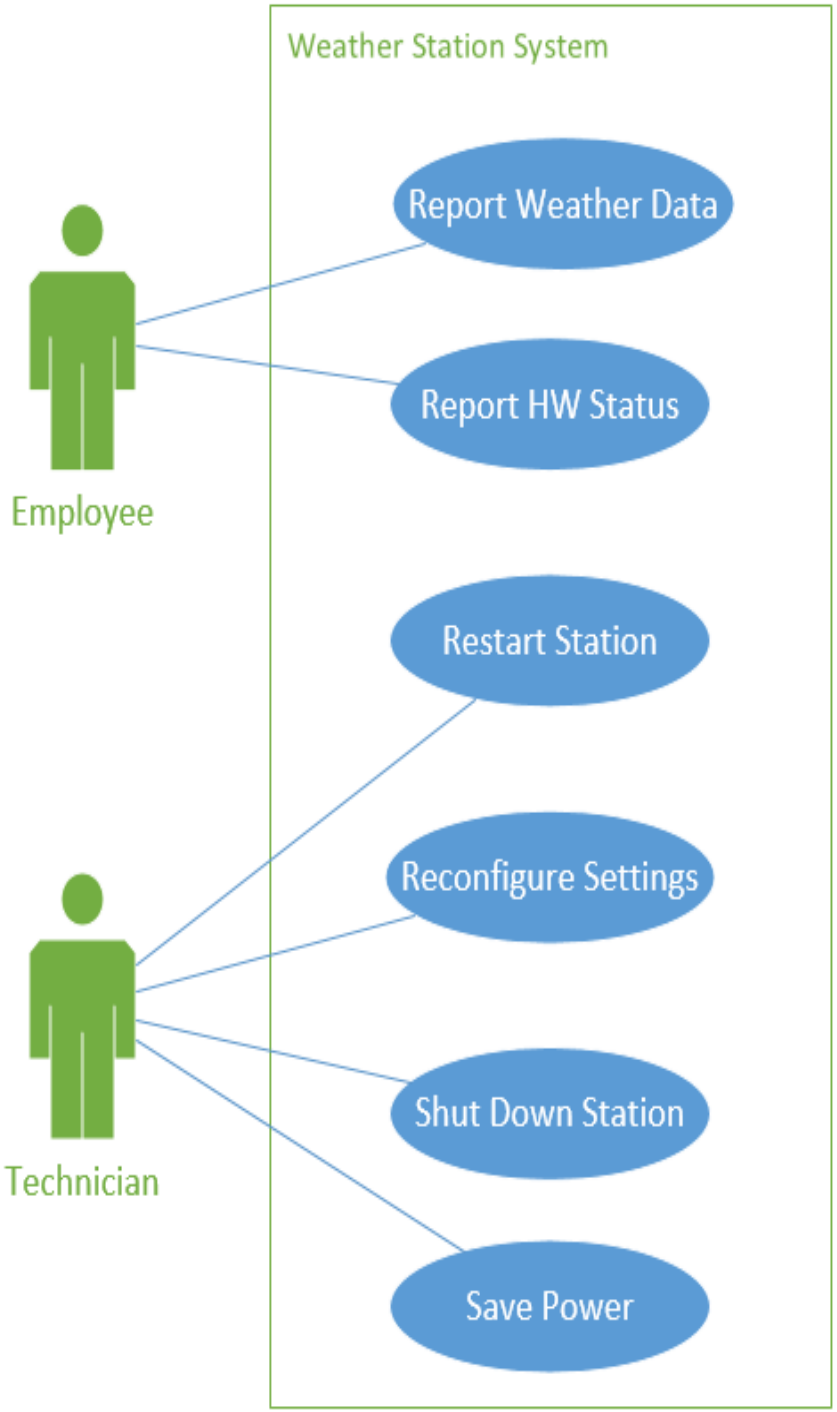


- When you have a use case that is processed by the system itself, you **cannot have** a **"system" actor.**
- **The solution is to** add it included in another use case that is initiated by another user.
- Example: When you create a new account, the system checks for user name availability, you cannot have a "system" actor, so, add a "create new account" use case that includes "check username availability" use case. Then, the actor is the user.

Exercise

The weather station system employee reports weather data and the status of the weather station hardware. Moreover, technicians can issue specific weather station control commands, such as: Restart, shutdown, power saving and reconfiguration.

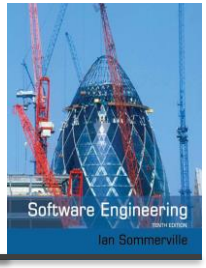
Draw use case diagram for weather station system.



Structural models

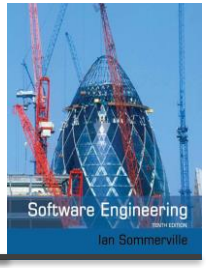
Chapter 5 System Modeling

Structural models



- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design.
- You create structural models of a system when you are discussing and designing the system architecture.

Class diagrams



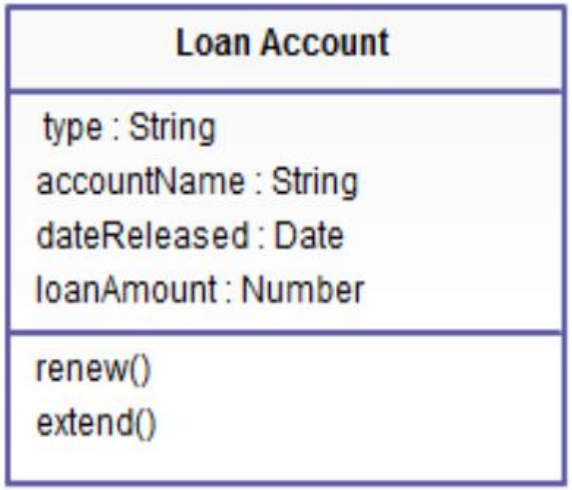
- Class diagrams are used **when** developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An **object class** can be thought of as a general definition of one kind of system object.
- An **association** is a link between classes that indicates 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.

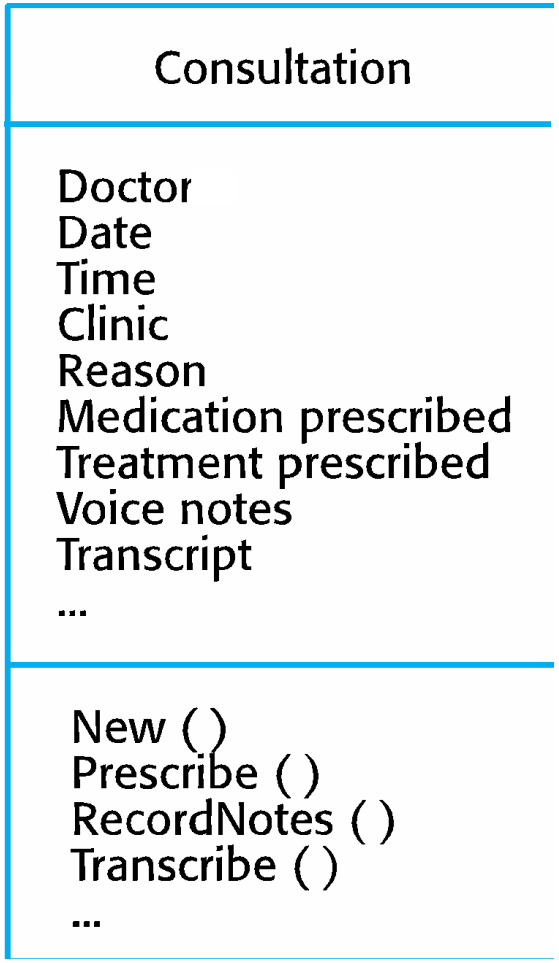
Class

- Each class has a class name, a set of attributes (optional), and a set of methods (optional).
- The scope of the attributes and methods can be specified (Private, Package, Protected, Public).

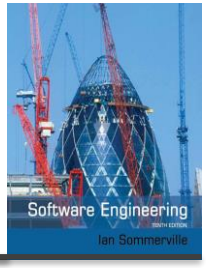
Class

Software Engineering
Ian Sommerville


Loan Account
type : String
accountName : String
dateReleased : Date
loanAmount : Number
renew()
extend()

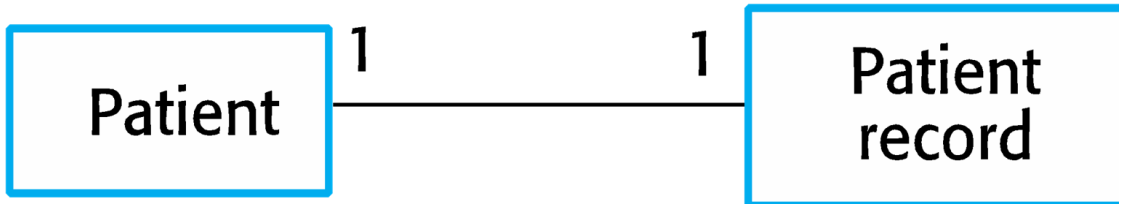

Consultation
Doctor
Date
Time
Clinic
Reason
Medication prescribed
Treatment prescribed
Voice notes
Transcript
...
New ( )
Prescribe ( )
RecordNotes ( )
Transcribe ( )
...

Association



- The association relationship describes a relationship between two classes where one class can use another class in some way.
- Multiplicity property of the classes must be specified.
- Relationship can be bidirectional (zero or two arrowheads) or unidirectional relationship (one arrow head).
- It can optionally be assigned a meaningful name to describe how the two classes are associated.
- A reflexive relationship describes relationship between one class and itself.

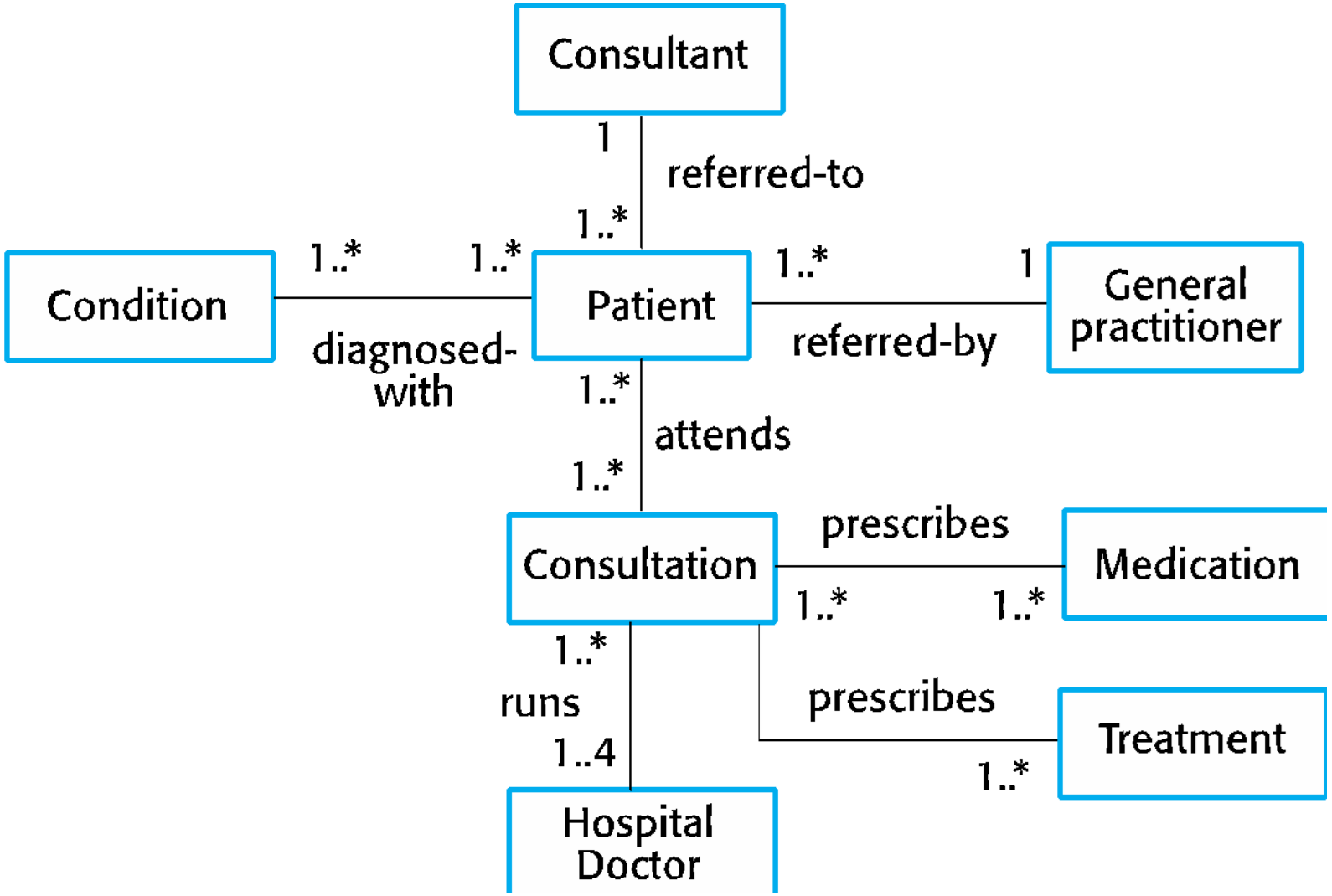
UML classes and association



Multiplicity Types

Symbol	Meaning
0	None
1	One
m	An integer value
m, n	m or n
0..1	Zero or one
m..n	At least *m, but not more than n*
*	Any nonnegative integer (zero or more)
0..*	Zero or more (identical to *)
1..*	One or more

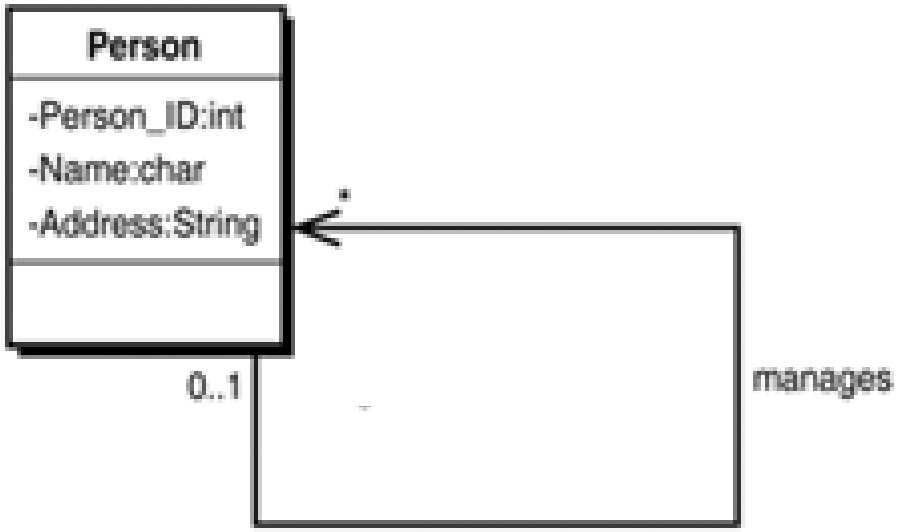
Classes and associations in the MHC-PMS



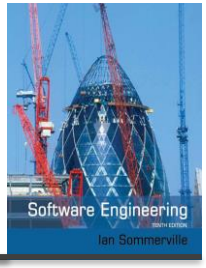
Reflexive association

A staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is supervised by the aviation engineer, there could be a “supervise” relationship in two instances of the same class.

Reflexive association

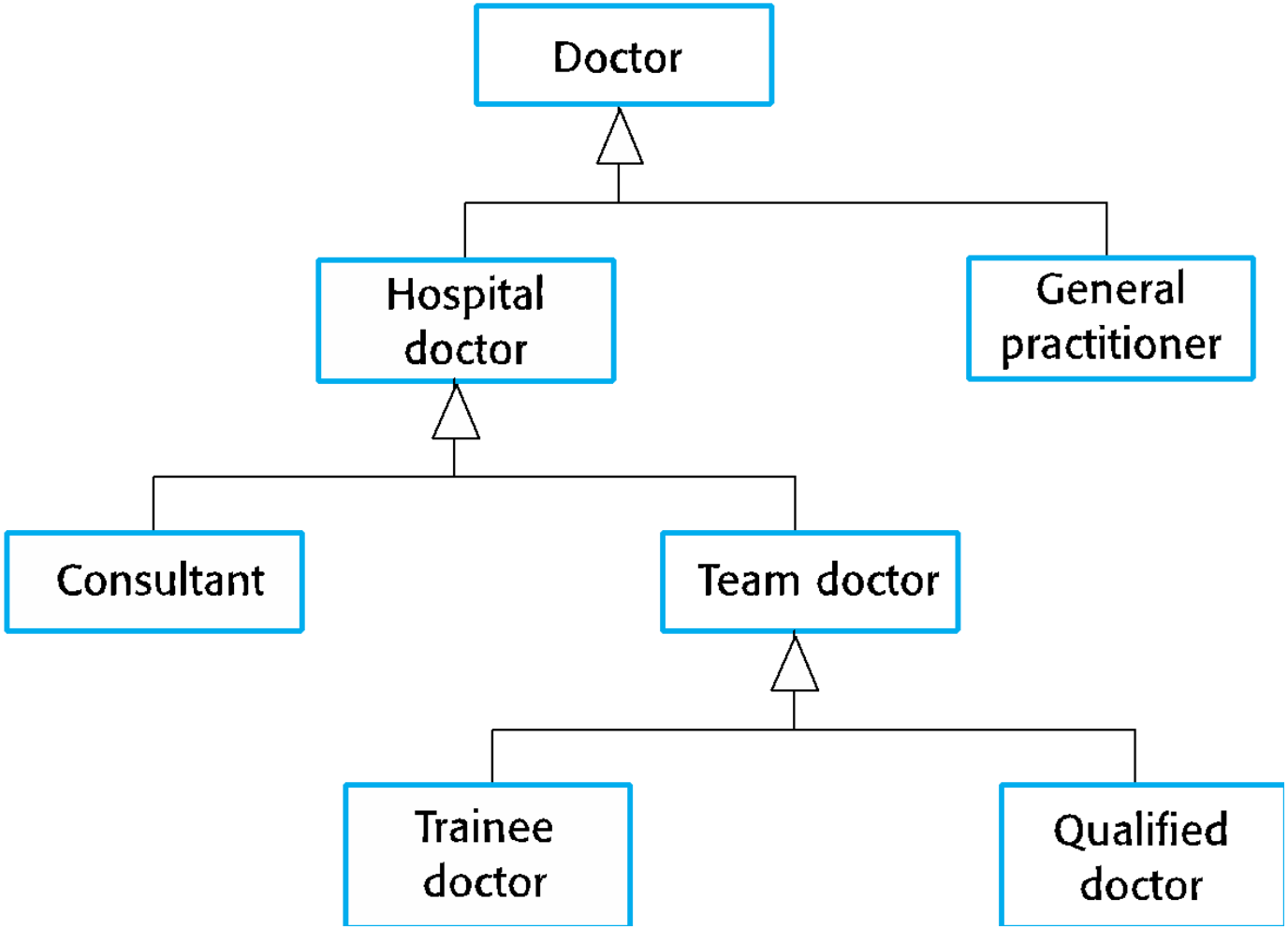


Generalization

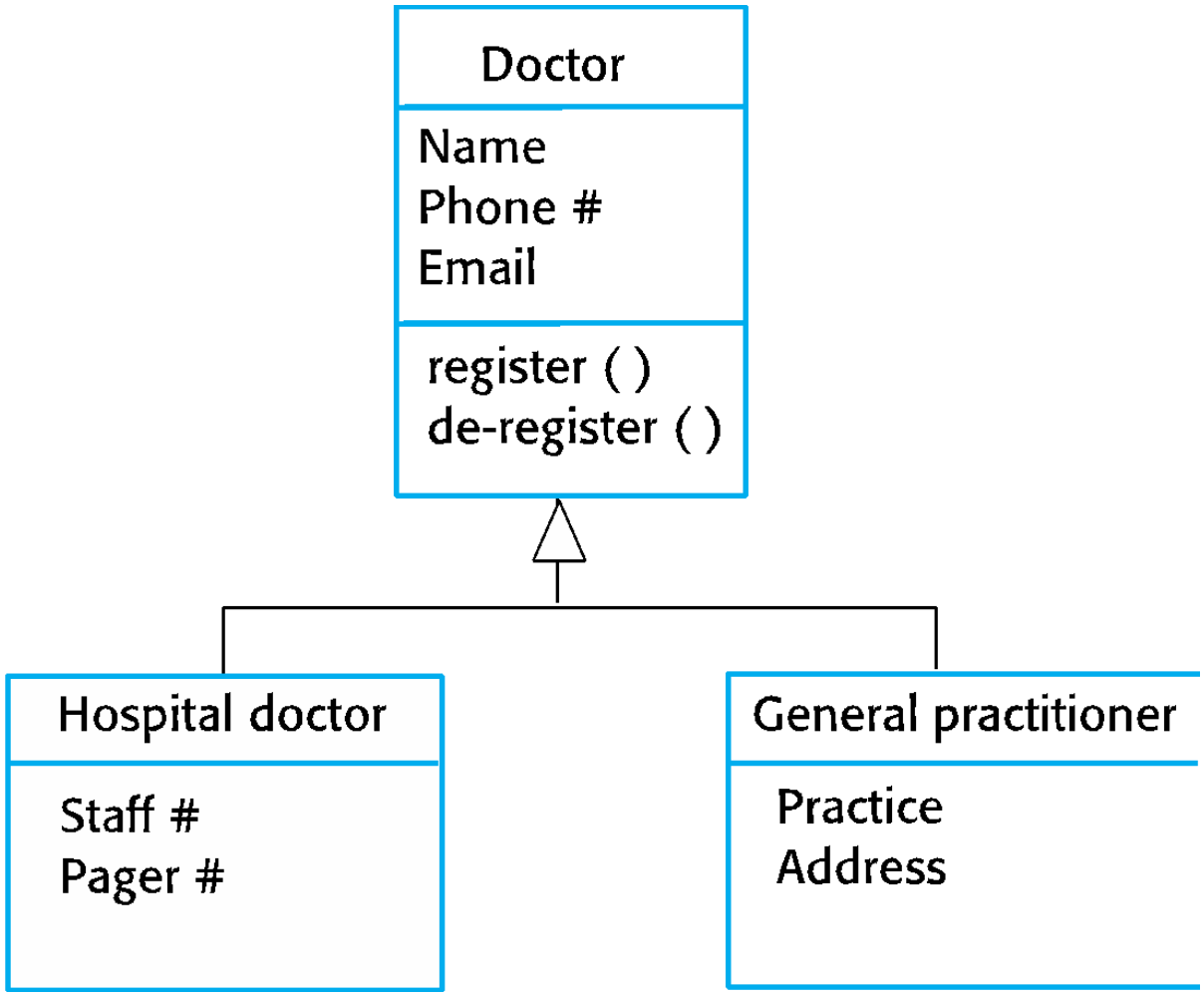


- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization.
- In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language.
- The lower-level classes are subclasses inherit the attributes and operations from their super classes. These lower-level classes then add more specific attributes and operations.
- This type of relationship is said to be a “IS-A” relationship.

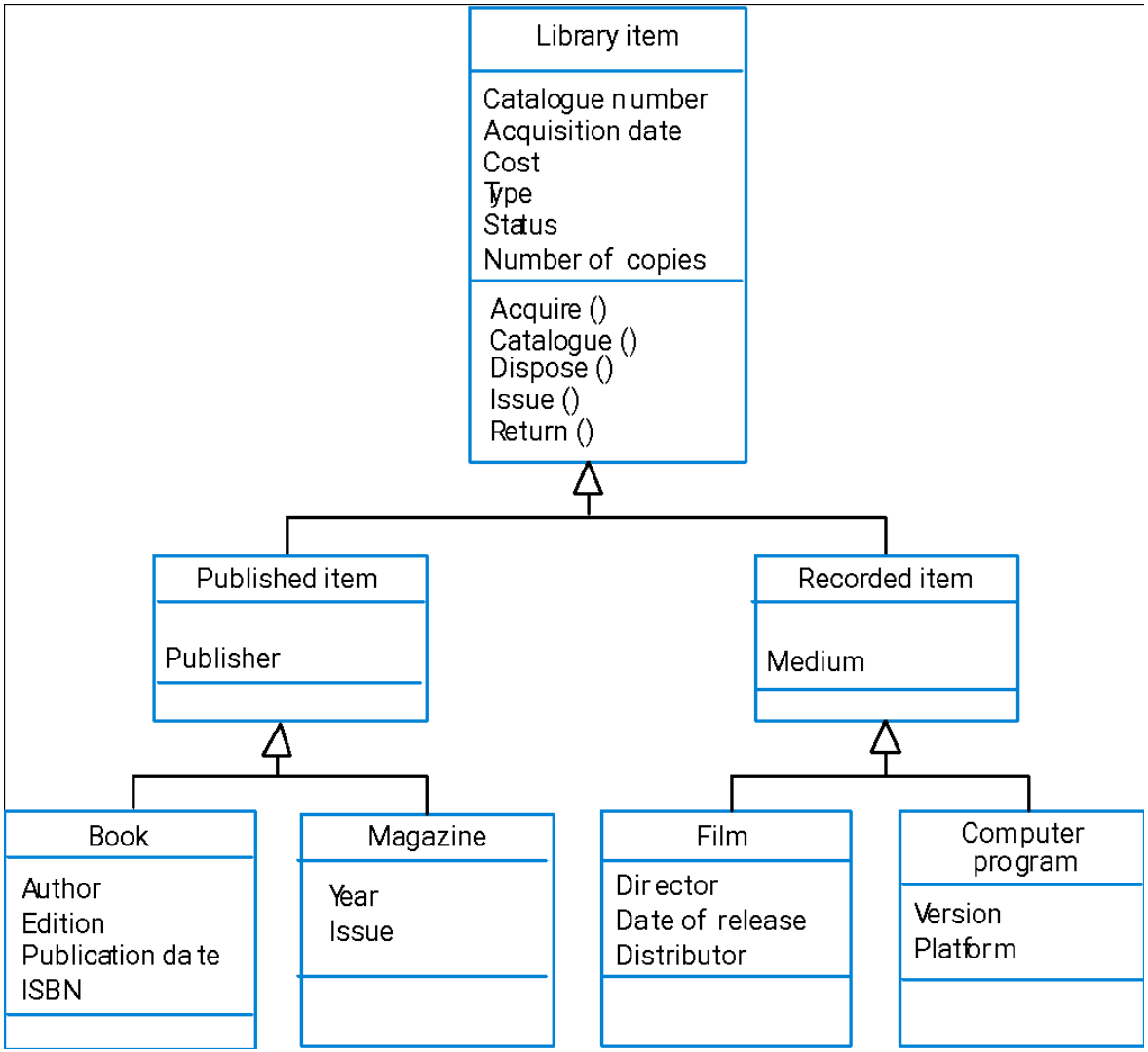
Generalization hierarchy



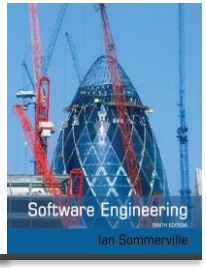
Generalization hierarchy with added detail



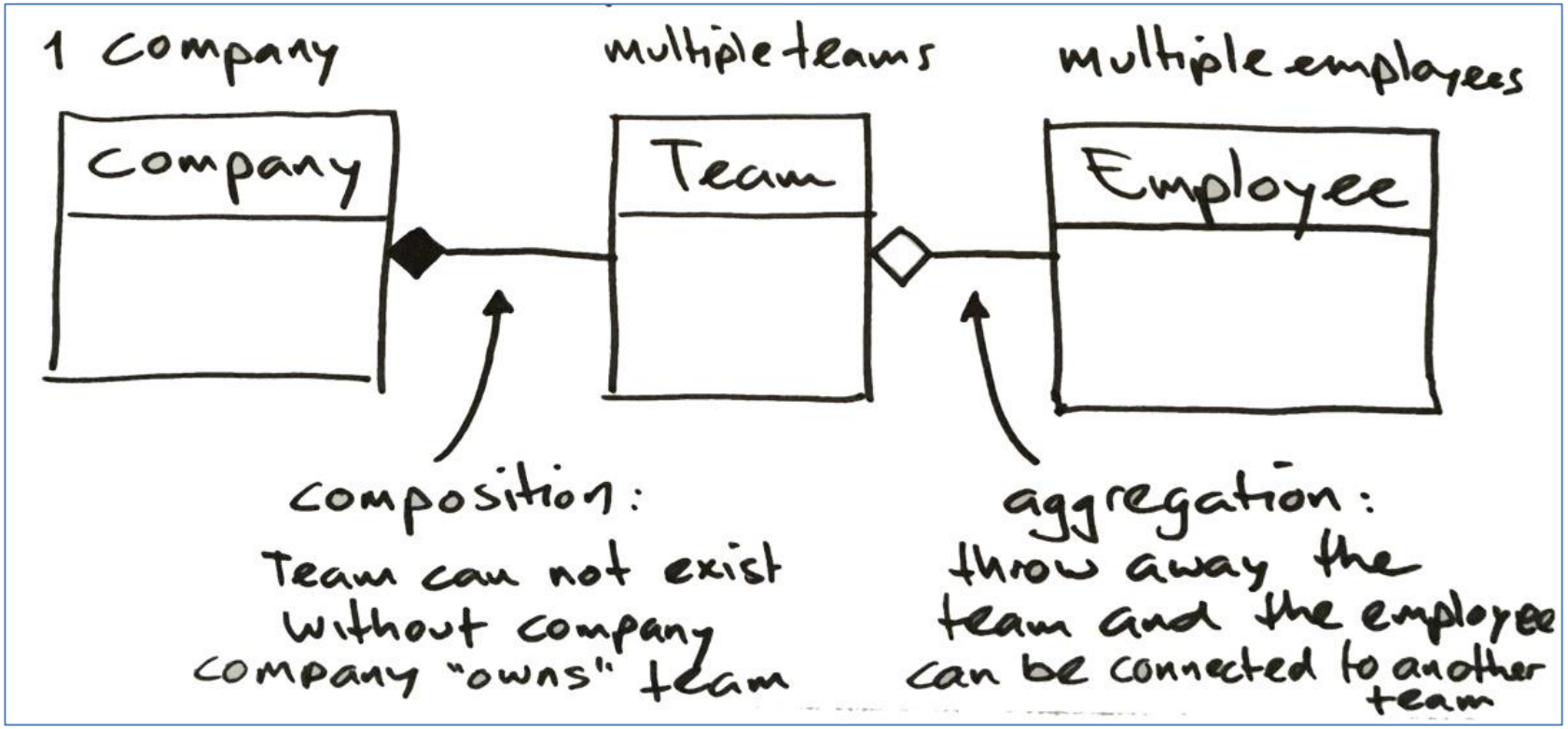
Generalization hierarchy



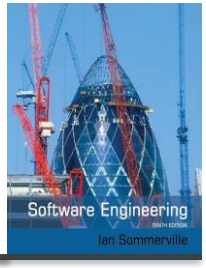
Object class aggregation models



- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation and composition associations.

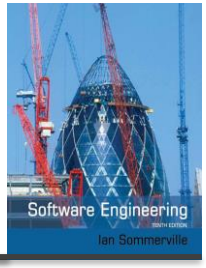

1 company
multiple teams
multiple employees
Company
Team
Employee
composition:
Team can not exist
without company
company "owns" team
aggregation:
throw away the
team and the employee
can be connected to another
team

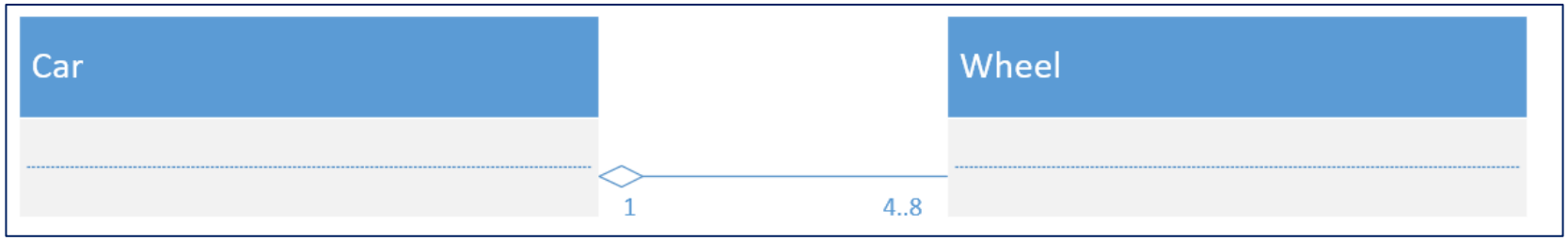
Aggregation association

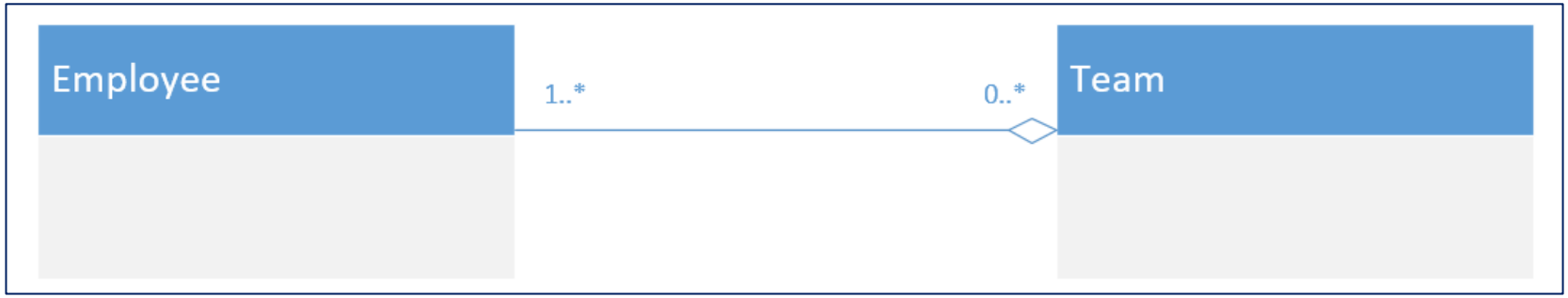


- The aggregation association is said to be a "HAS-A" relationship where the owner class owns the owned class
- **Aggregation** implies a relationship where the child can exist independently of the parent.
- The life time of the owned class is independent of the owner class.
- The whole and the part are created and destroyed at different times.

Aggregation association



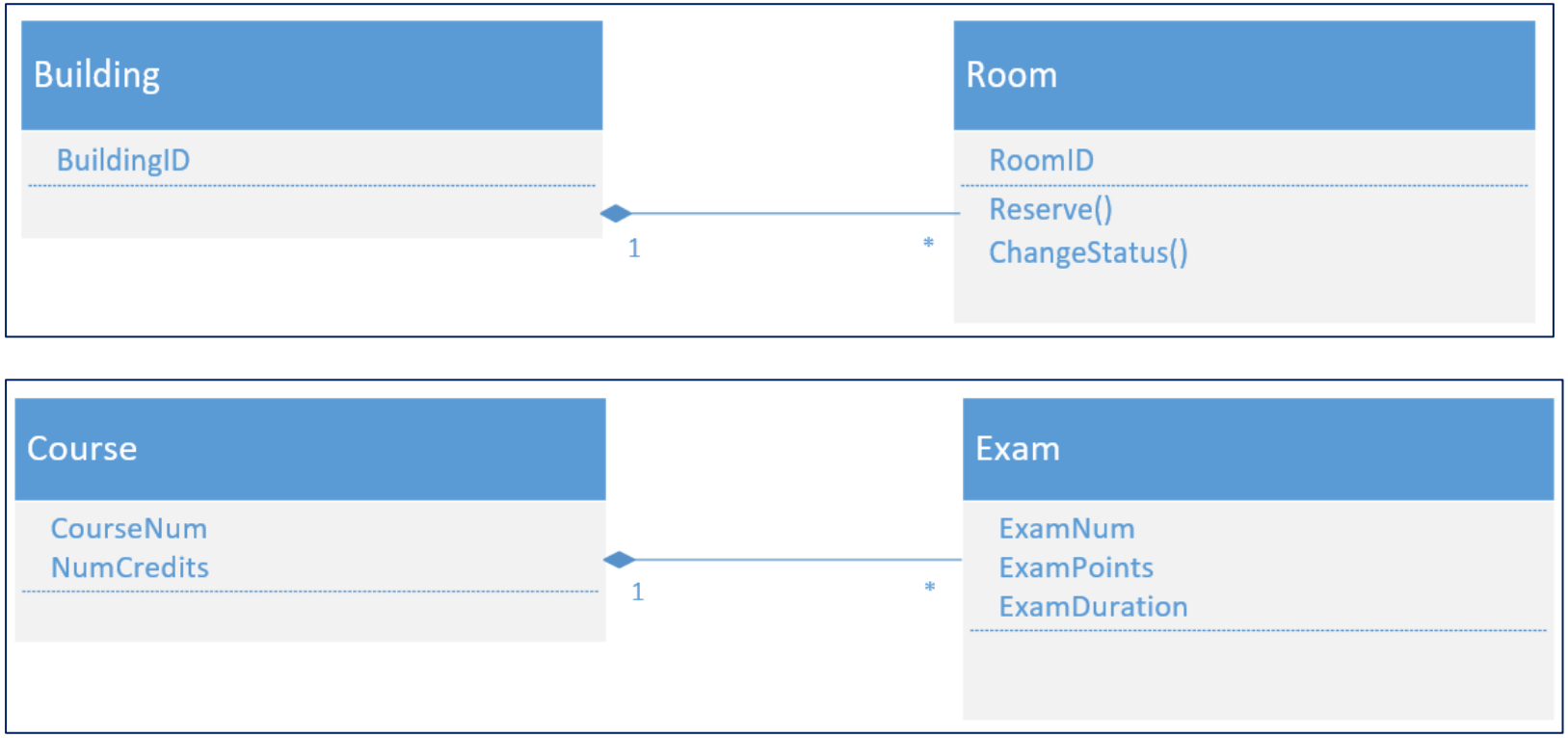




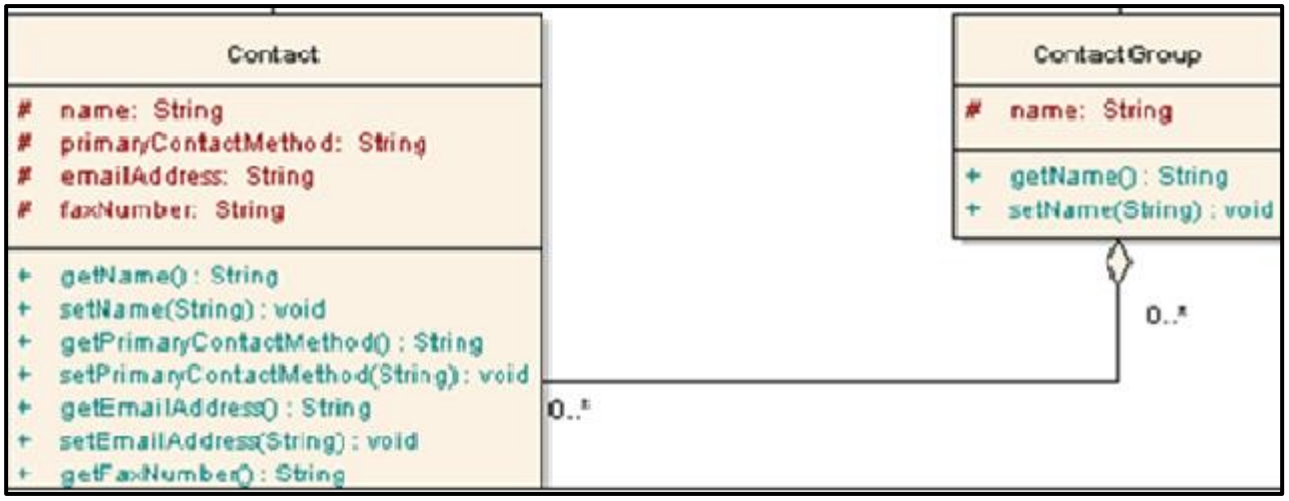
Composition association

- The composition association is said to be a "WHOLE-PART" relationship where the owner class owns the owned class
- **Composition** implies a relationship where the child cannot exist independent of the parent.
- The life time of the owned class is dependent on the owner class.
- The whole and the part are created and destroyed at the same time.

Composition association

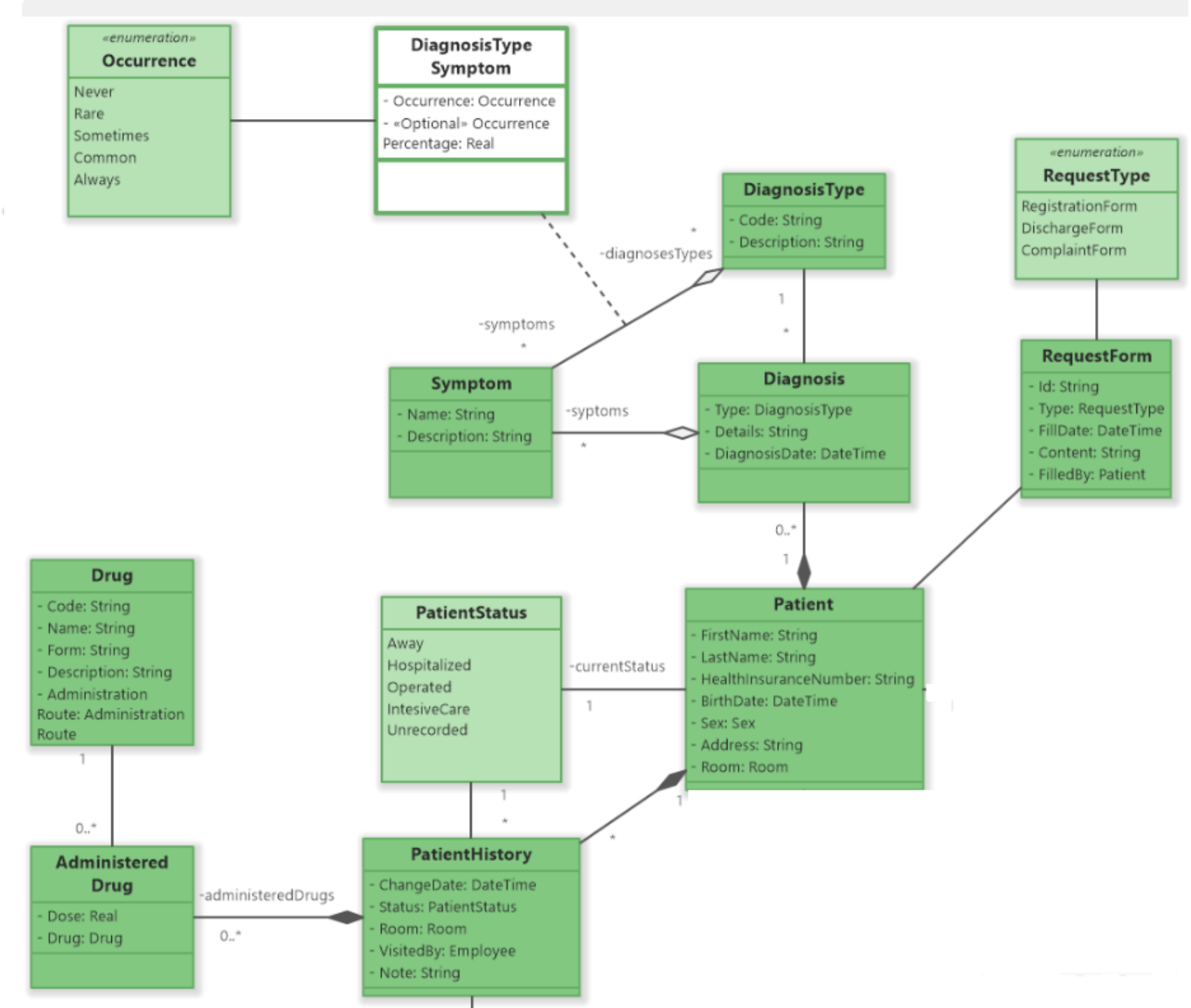


Aggregation and composition associations



Case Study

Hospital management system keeps information related to the doctors, patients, visits, and prescriptions. Doctor can visit the patients, examine them, add diagnoses, record symptoms, prescribe drugs, and schedule medical procedures, report the patient condition, and finally discharge the patient. System keeps information regarding symptoms, diagnosis details and prescription drug details. Patient can register himself, unregister, confirm the medical procedure, and fills request form for discharge. Patient history can be accessed by the doctor to make better judgements. Patient status can be displayed anytime, such as: hospitalized, intensive care, away and discharged.


«enumeration»
Occurrence
Never
Rare
Sometimes
Common
Always
DiagnosisType Symptom
- Occurrence: Occurrence
- «Optional» Occurrence
Percentage: Real
-diagnosesTypes
DiagnosisType
- Code: String
- Description: String
«enumeration»
RequestType
RegistrationForm
DischargeForm
ComplaintForm
-symptoms
Symptom
- Name: String
- Description: String
-syptoms
Diagnosis
- Type: DiagnosisType
- Details: String
- DiagnosisDate: DateTime
RequestForm
- Id: String
- Type: RequestType
- FillDate: DateTime
- Content: String
- FilledBy: Patient
Drug
- Code: String
- Name: String
- Form: String
- Description: String
- Administration Route: Administration Route
PatientStatus
Away
Hospitalized
Operated
IntesiveCare
Unrecorded
-currentStatus
Patient
- FirstName: String
- LastName: String
- HealthInsuranceNumber: String
- BirthDate: DateTime
- Sex: Sex
- Address: String
- Room: Room
Administered Drug
- Dose: Real
- Drug: Drug
-administeredDrugs
PatientHistory
- ChangeDate: DateTime
- Status: PatientStatus
- Room: Room
- VisitedBy: Employee
- Note: String
1
*
0..*

Thank You