SOFTWARE ENGINEERING

Chapter 2

Requirements analysis and specification

Requirements specification - 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: representing a system using some kind of graphical notation:
  - **Functional Decomposition Diagram (FDD)**
  - **Data Flow Diagram (DFD)**
  - **Entity-Relationship Diagram (ERD)**
  - **Data Dictionaries**
  - **Unified Modeling Language (UML): Use-case, Class Diagram**

Existing and planned system models

- **Models of the existing system**: can be used as a basis for discussing its strengths and weaknesses → requirements for the new system.
- **Models of the new system**: help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

System perspectives

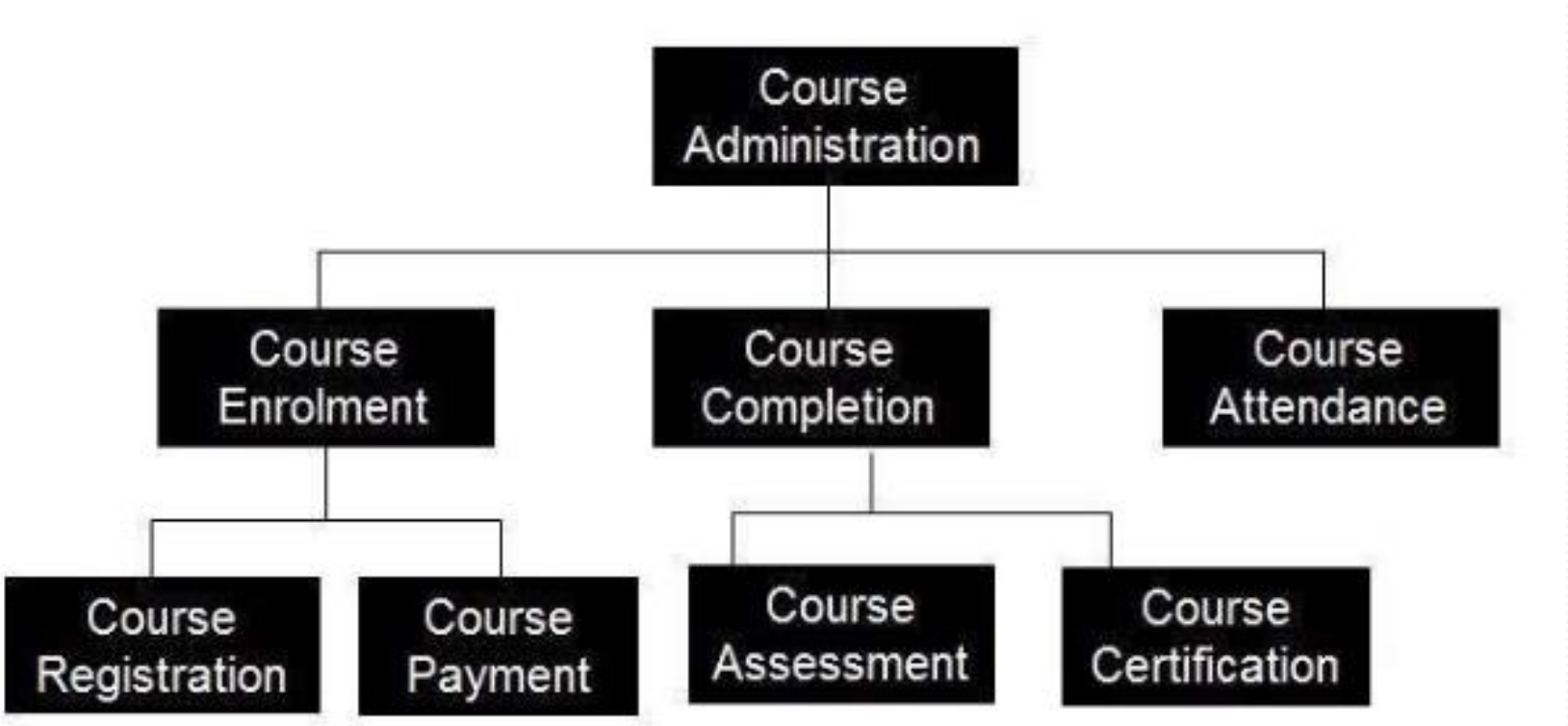
- Develop different models to represent the system from different perspectives.
  - External perspective: model the context or environment of the system.
  - Interaction perspective: model the interactions between a system and its environment, or between the components of a system.
  - Structural perspective: model the organization of a system or the structure of the data that is processed by the system.
  - Behavioral perspective: model the dynamic behavior of the system and how it responds to events.

Functional Decomposition Diagram (FDD)

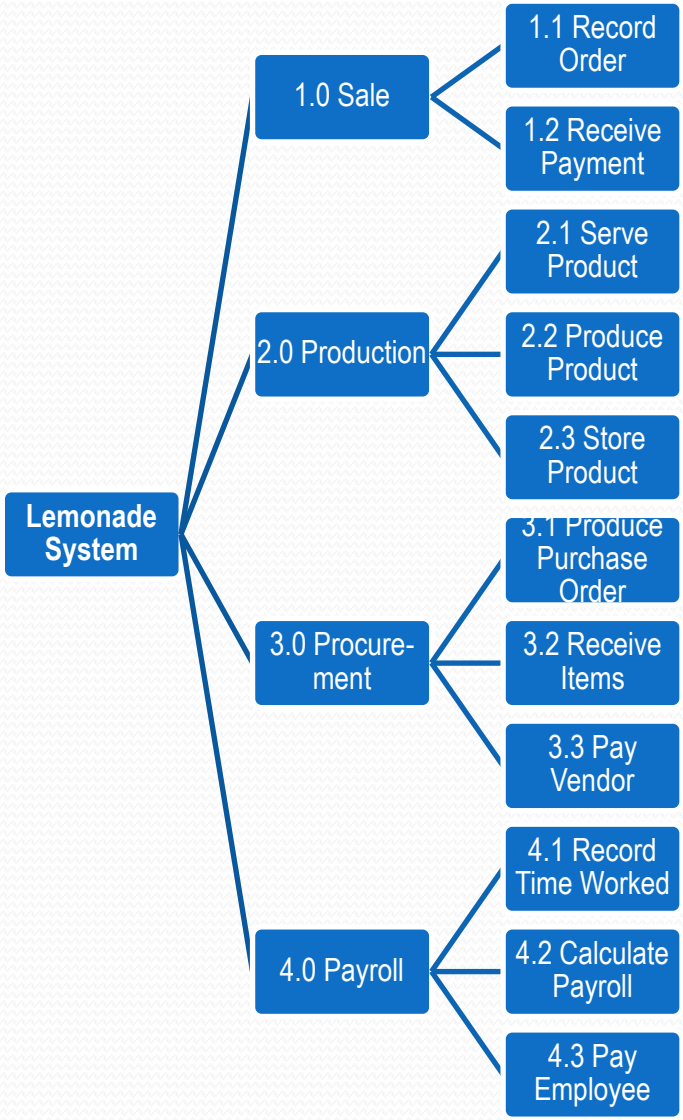
Functional Decomposition Diagram (FDD)

- Enables you to view your system functions in an hierarchical structure
- Breaks down a large, complex process into an array of smaller, simpler units or tasks
- Contains the overall function or task as well as the necessary sub-functions or tasks needed to achieve the overall objective.

FDD – Example 1



FDD – Example 2



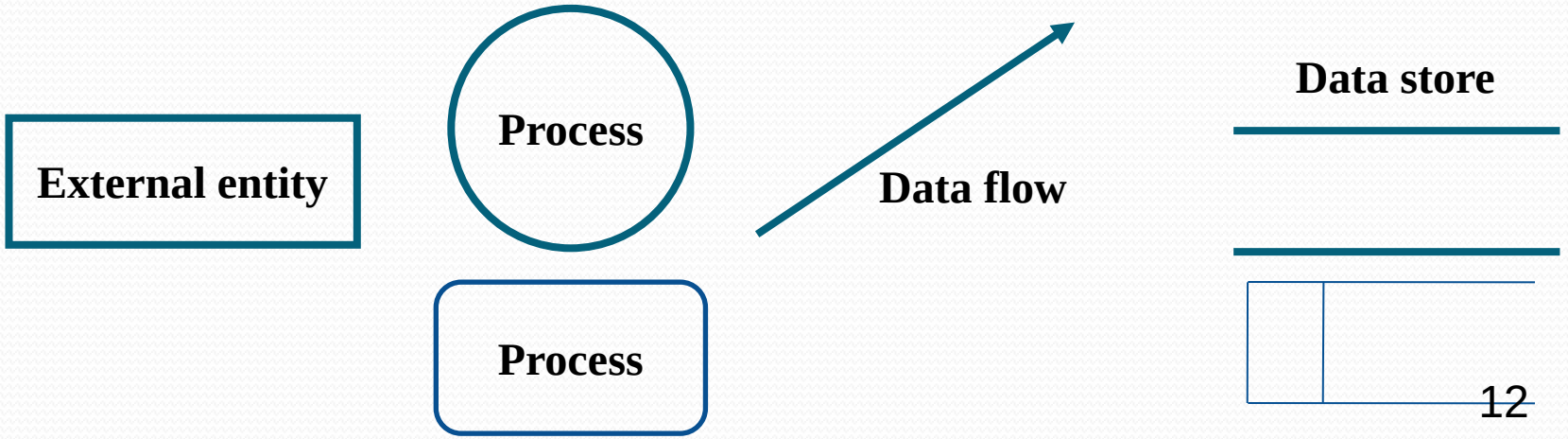
Data Flow Diagrams (DFD)

Data Flow Diagrams (DFD)

- DFD is a traditional visual representation of the information flows within a system.
- It shows how data enters and leaves the system, what changes the information, and where data is stored
- It works well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems

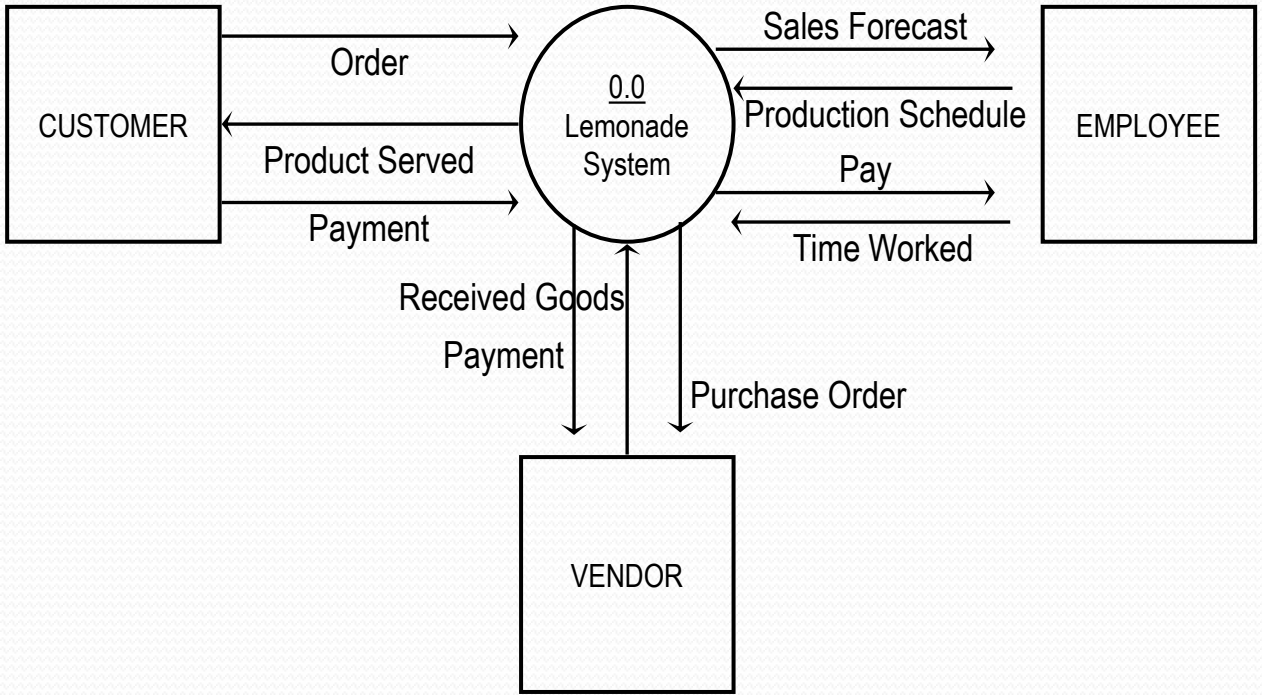
Standard symbols for DFDs

- **External entity**: an outside system that sends/receives data, communicating with the system; sources and destinations of information; outside organization or person
- **Process**: changes the data, producing an output.
- **Data store**: files or repositories that hold information for later use
- **Data flow**: shows the flow of data into or out of a process or data store.



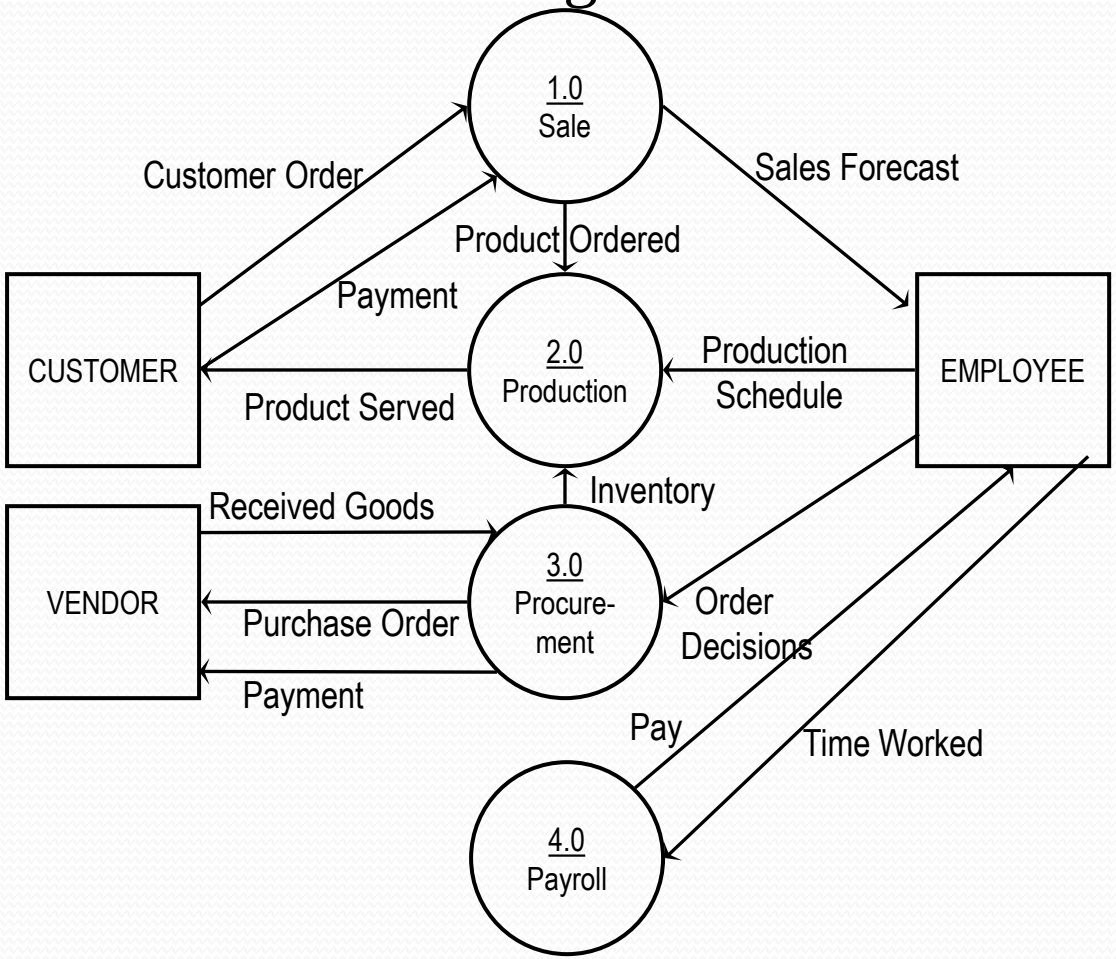
Levels in DFD

- DFDs may be partitioned into levels that represent increasing information flow and functional detail.
- Levels in DFD are numbered 0, 1, 2,…
- 0-level DFD (context diagram)
  - It's a basic overview of the whole system



Levels in DFD

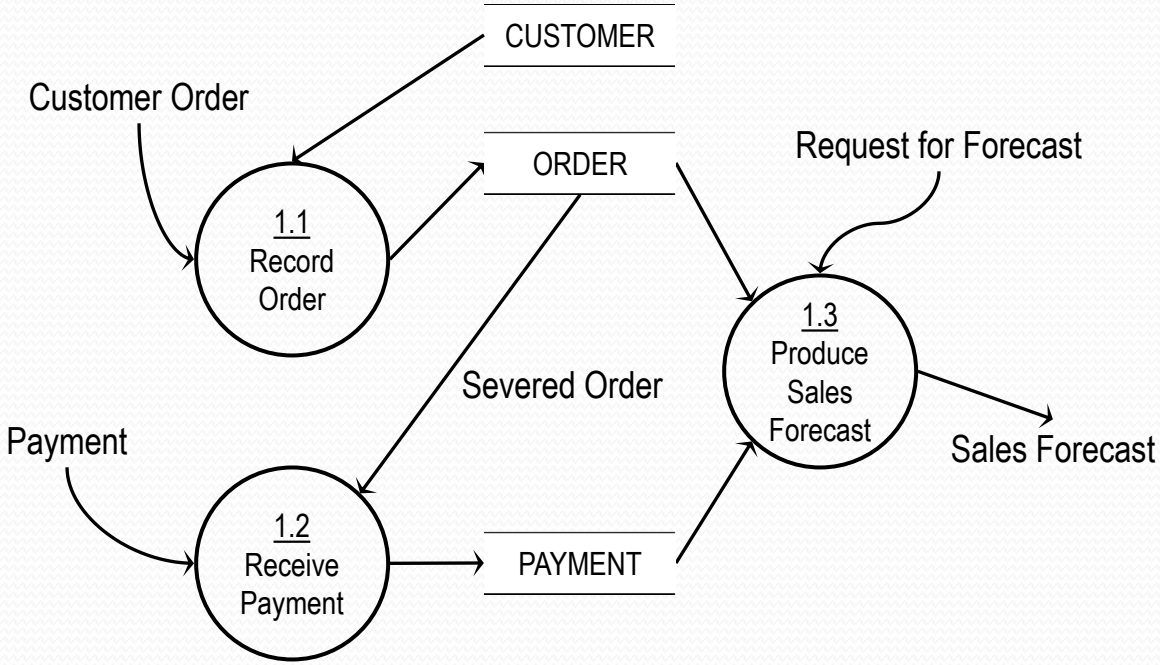
- 1-level DFD: provides a more detailed breakout of pieces of the Context Level Diagram.



Levels in DFD

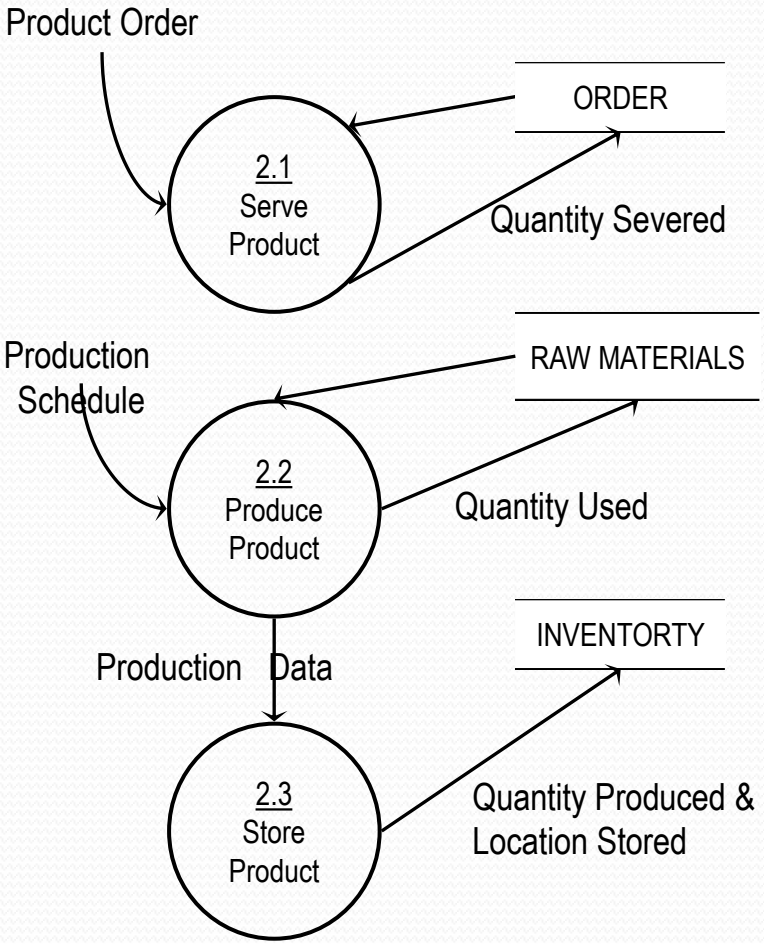
- 2-level DFD: goes one process deeper into parts of 1-level DFD

1.0 Sale



Levels in DFD

2.0 Production



Entity-Relationship Diagram (ERD)

Entity-Relationship Diagram (ERD)

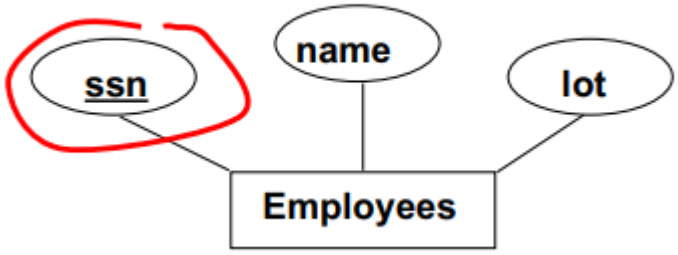
- ER-modeling is a data modeling method used to produce a conceptual data model of an information system
- Purpose of ERD
  - The database analyst gains a better understanding of the data to be contained in the database
  - ERD is used to connect the logical structure of the database to users.
- Components of an ERD:
  1. Entity
  2. Attributes
  3. Relationships

1. Entity

- A Real-world object, distinguishable from other objects
- Described using a set of attributes.
- Has its own identity and represents just one thing
- Exp: students, teachers, classes, and courses

Entity Set

- A collection of similar entities.
- All entities in an entity set have the same set of attributes
- Each entity set has a key (underlined).
- Exp: a Student set may contain all the students of a school



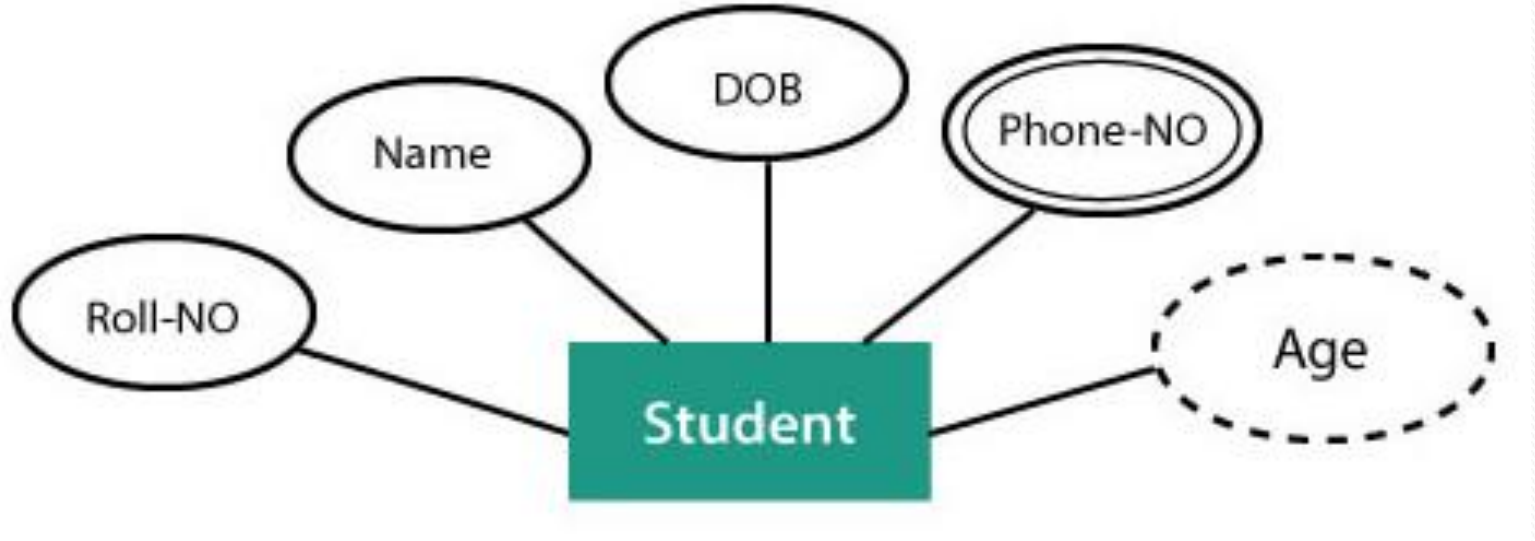
2. Attributes

- Attributes are simple values, e.g. integers or character strings, not structs, sets, etc.
- Exp: a student entity may have name, class, and age as attributes.

Types of Attributes:

- Key: uniquely identifies an entity among the entity set
- Composite: combination of other attributes.
- Single-valued: contain a single value
- Multi-valued: have more than one value
- Derived: attribute that does not exist in the physical database, but their values are derived from other attributes present in the database

2. Attributes



3. Relationships

- The association among entities is known as relationship.

Relationship set

- A set of relationships of a similar type. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

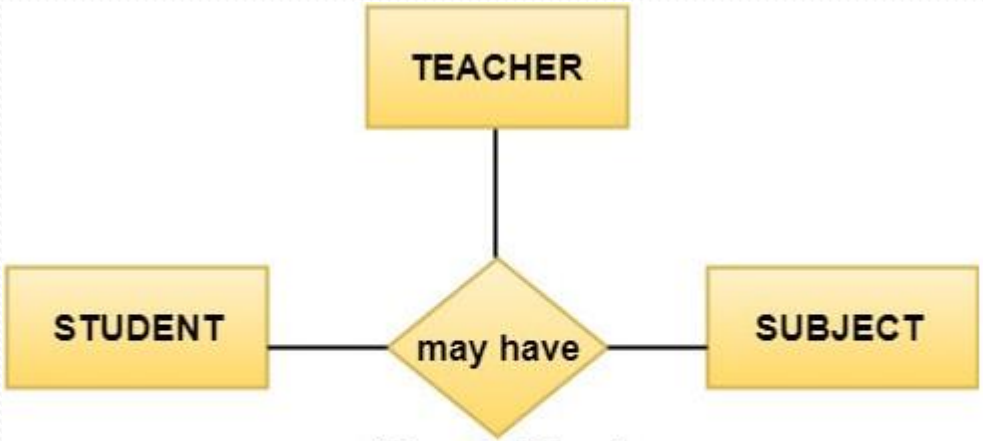
Degree of a relationship set

- The number of participating entities in a relationship describes the degree of the relationship.
  - Unary (degree 1)
  - Binary (degree 2)
  - Ternary (degree 3)

3. Relationships

Fig: Unary Relationship

Fig: Binary Relationship



(Many to Many)

Fig: Ternary Relationship

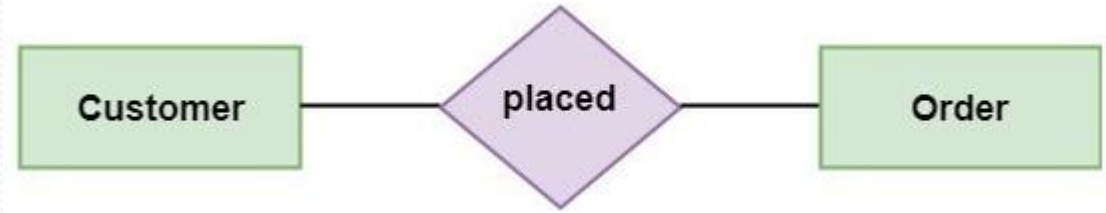
3. Relationships

Cardinality

- Describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.
- **Types of Cardinalities**
  - One to One
  - One to many
  - Many to One
  - Many to Many

3. Relationships

One to one



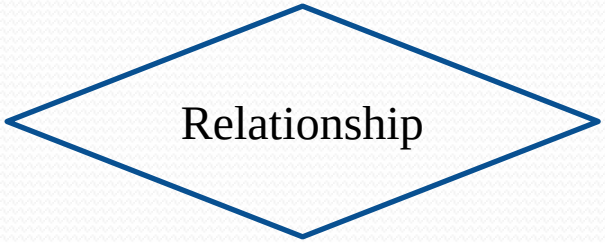
One to many

Many to one

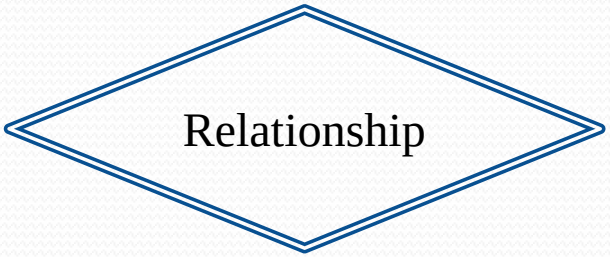
Many to many

Symbols of ERD

The existence of a weak entity is dependent upon another entity called the owner entity

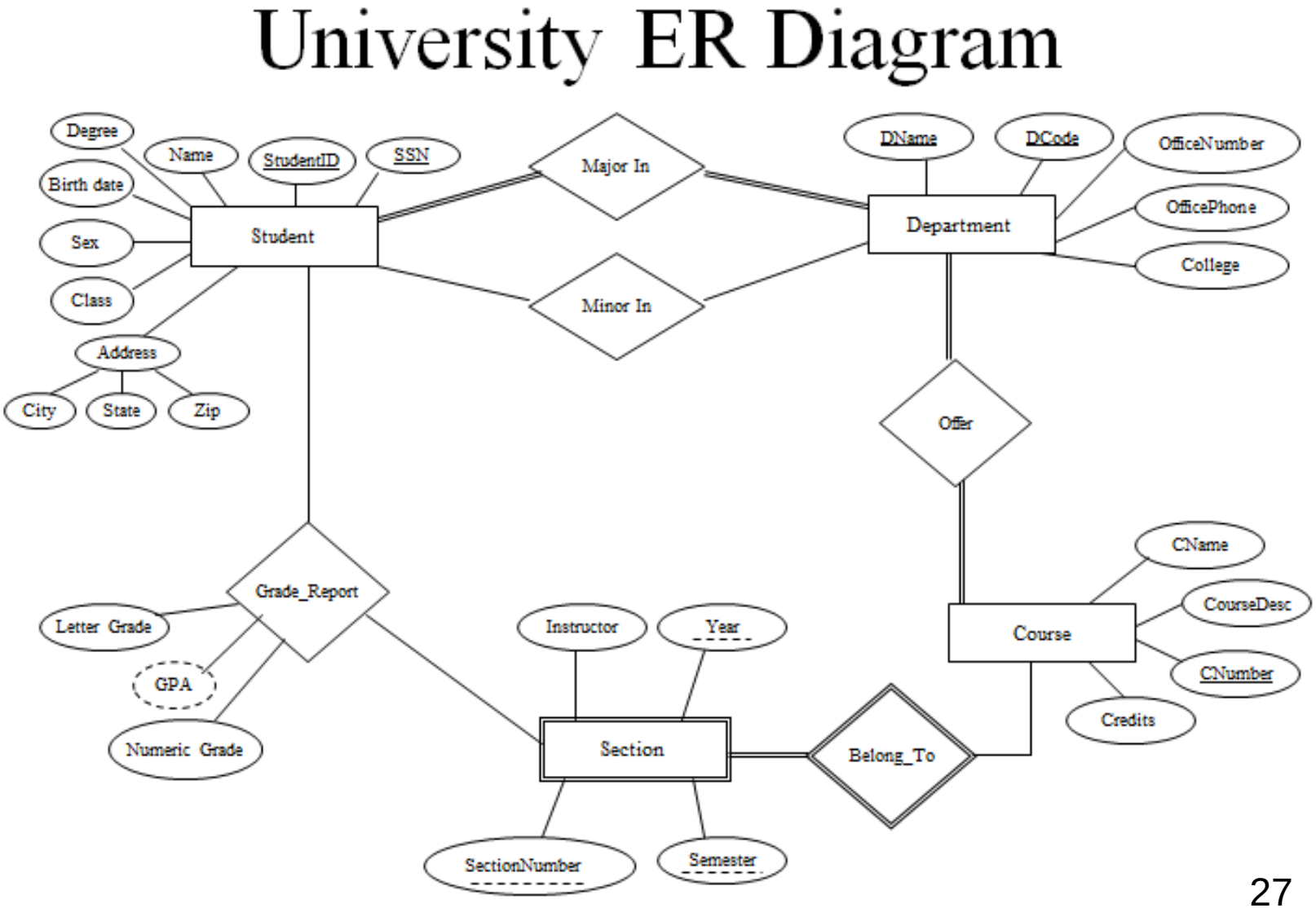


A relationship where entity is existence-independent of other entities



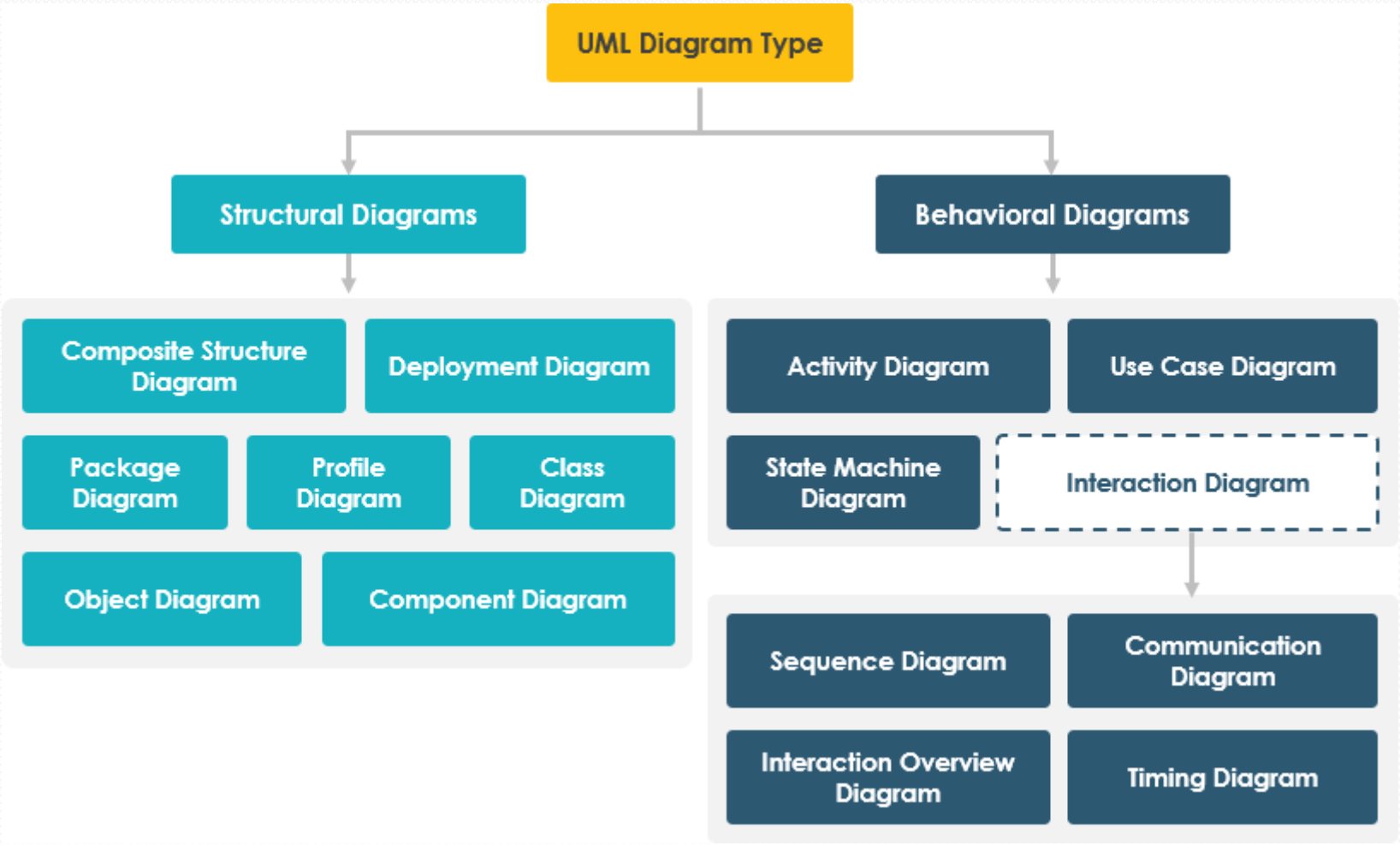
A relationship where Child entity is existence dependent on parent entity

Example - ERD


University ER Diagram
Degree
Name
StudentID
SSN
Birth date
Sex
Class
Address
City
State
Zip
Student
Major In
Minor In
DName
DCode
OfficeNumber
OfficePhone
College
Department
Offer
CName
CourseDesc
CNumber
Credits
Course
Grade_Report
Letter Grade
GPA
Numeric Grade
Instructor
Year
Section
Belong_To
SectionNumber
Semester

Unified Modeling Language (UML)

UML diagram types



UML diagram types

- **Activity diagrams:** which show the 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 these classes.
- **State diagrams:** which show how the system reacts to internal and external events.

Use of graphical models

- As a means of facilitating discussion about an existing or proposed system
  - Incomplete 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 but need not be complete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.

Context Models

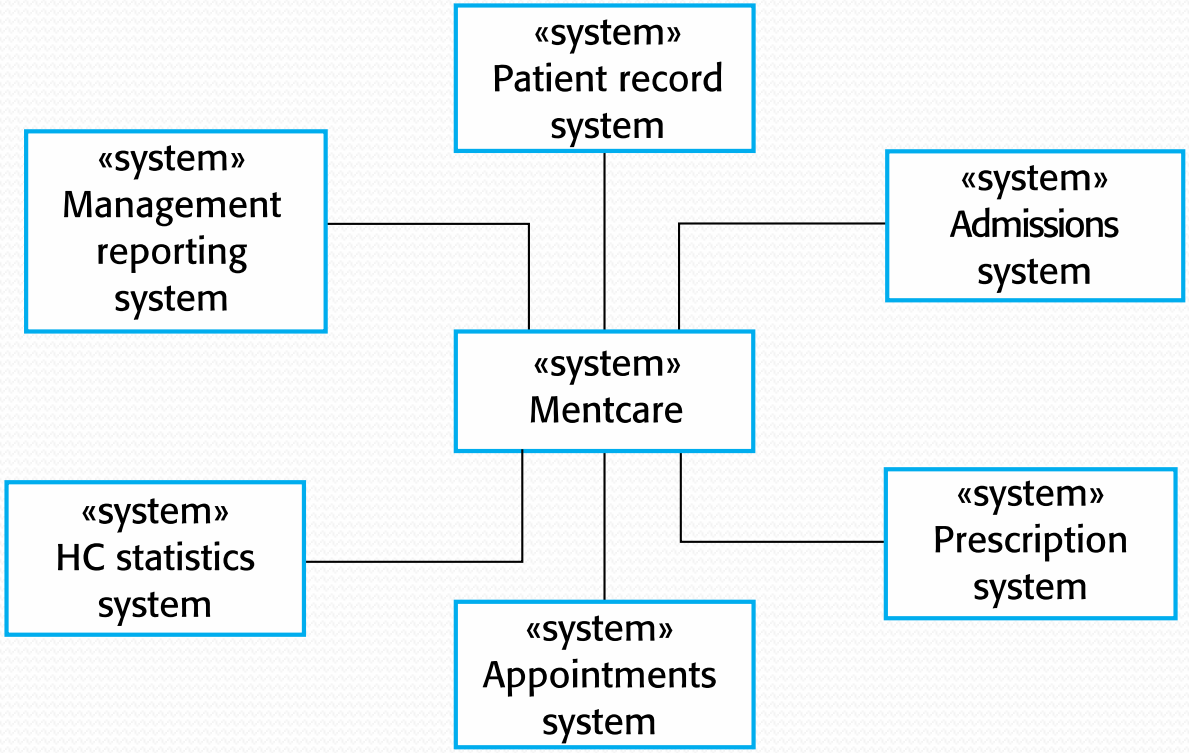
Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

System boundaries

- System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

The context of the Mentcare system



Interaction models

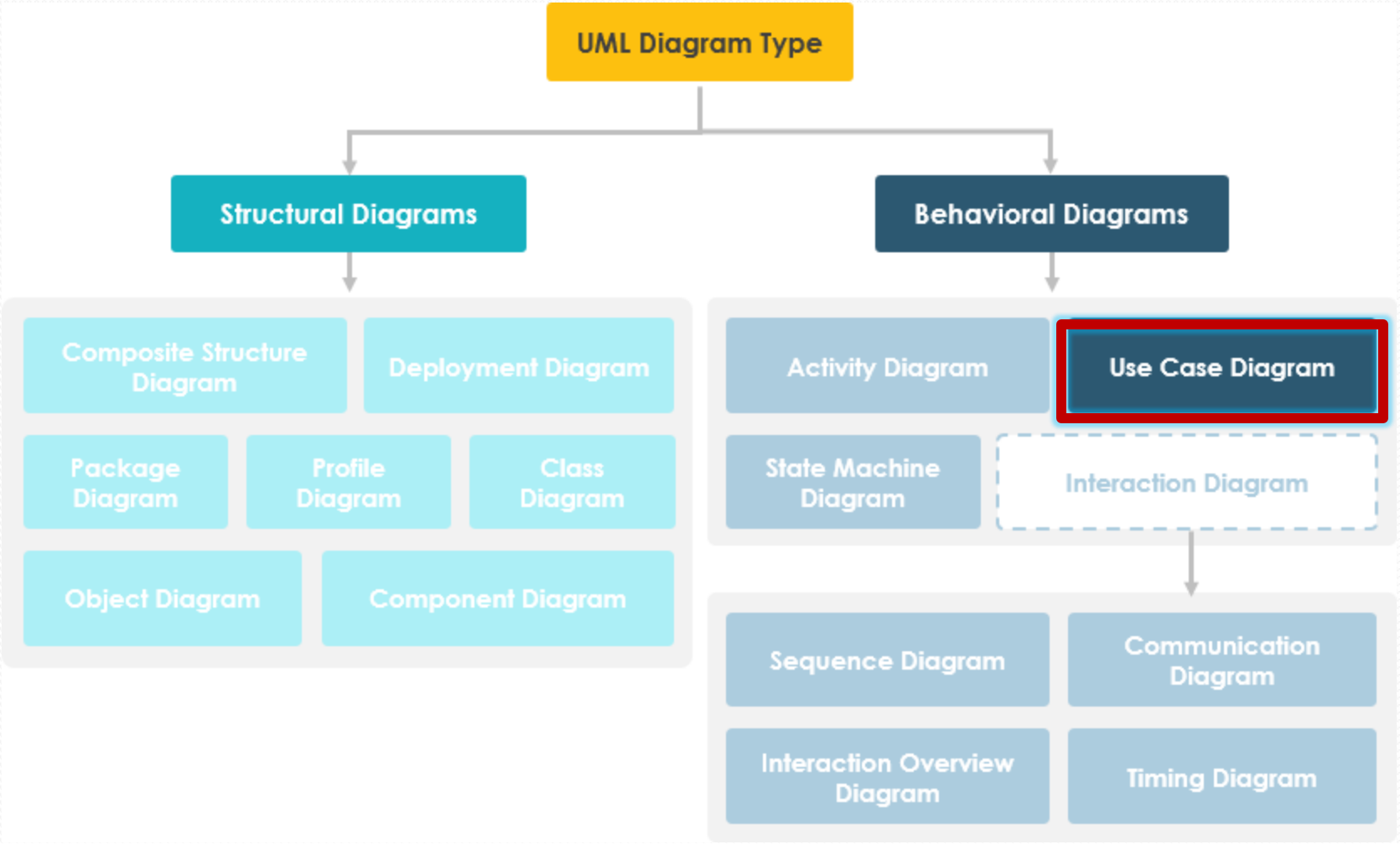
Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.

Use case modeling

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people or other systems.
- Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

Use case diagram

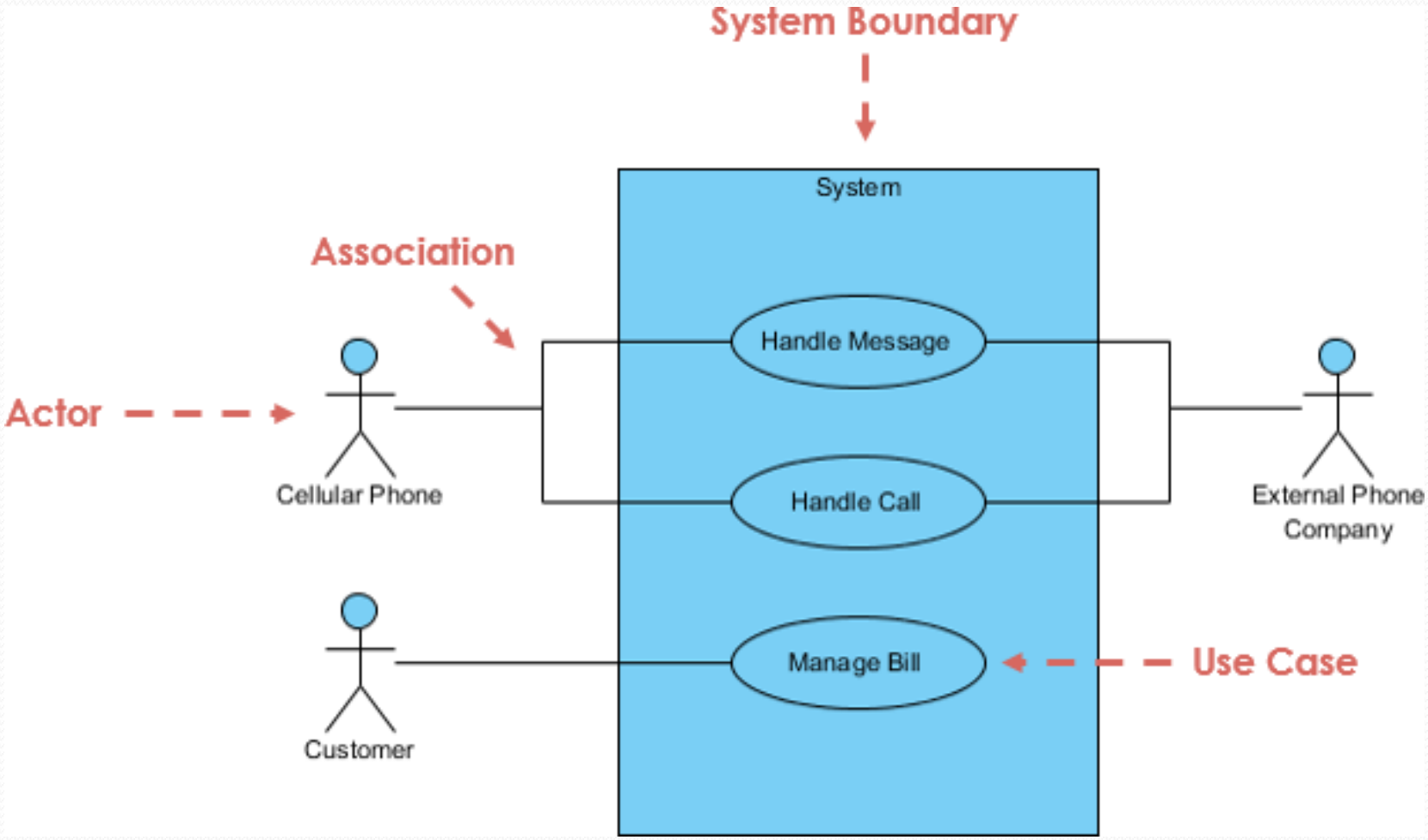


Use case diagram

Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

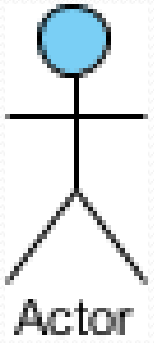
- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

Use case diagram



Actor

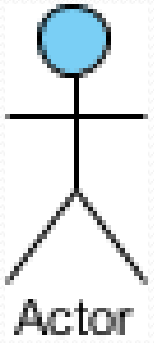
- Someone/**Something** interacts with use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
  - A prof. can be instructor and also researcher, plays 2 roles.
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).



Actors are always human???

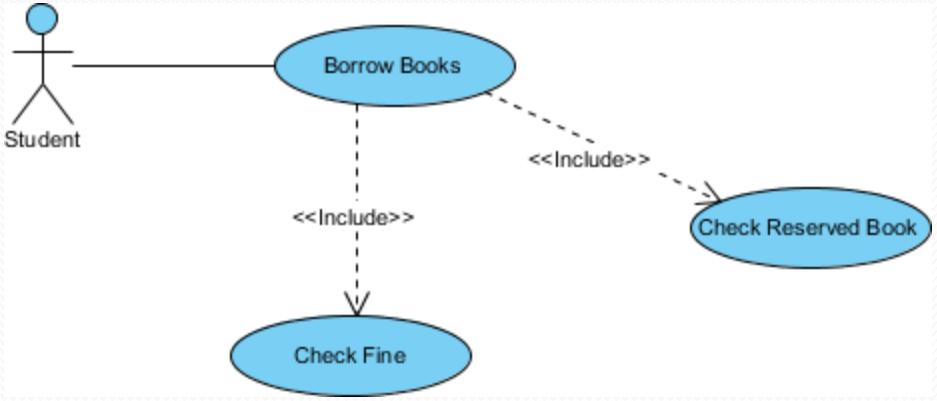
How to Identify Actor

- Often, people find it easiest to start the requirements elicitation process by identifying the actors. The following questions can help you identify the actors of your system (Schneider and Winters - 1998):
- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?
- Who provides information to the system?
- Does anything happen automatically at a present time?.



Use case

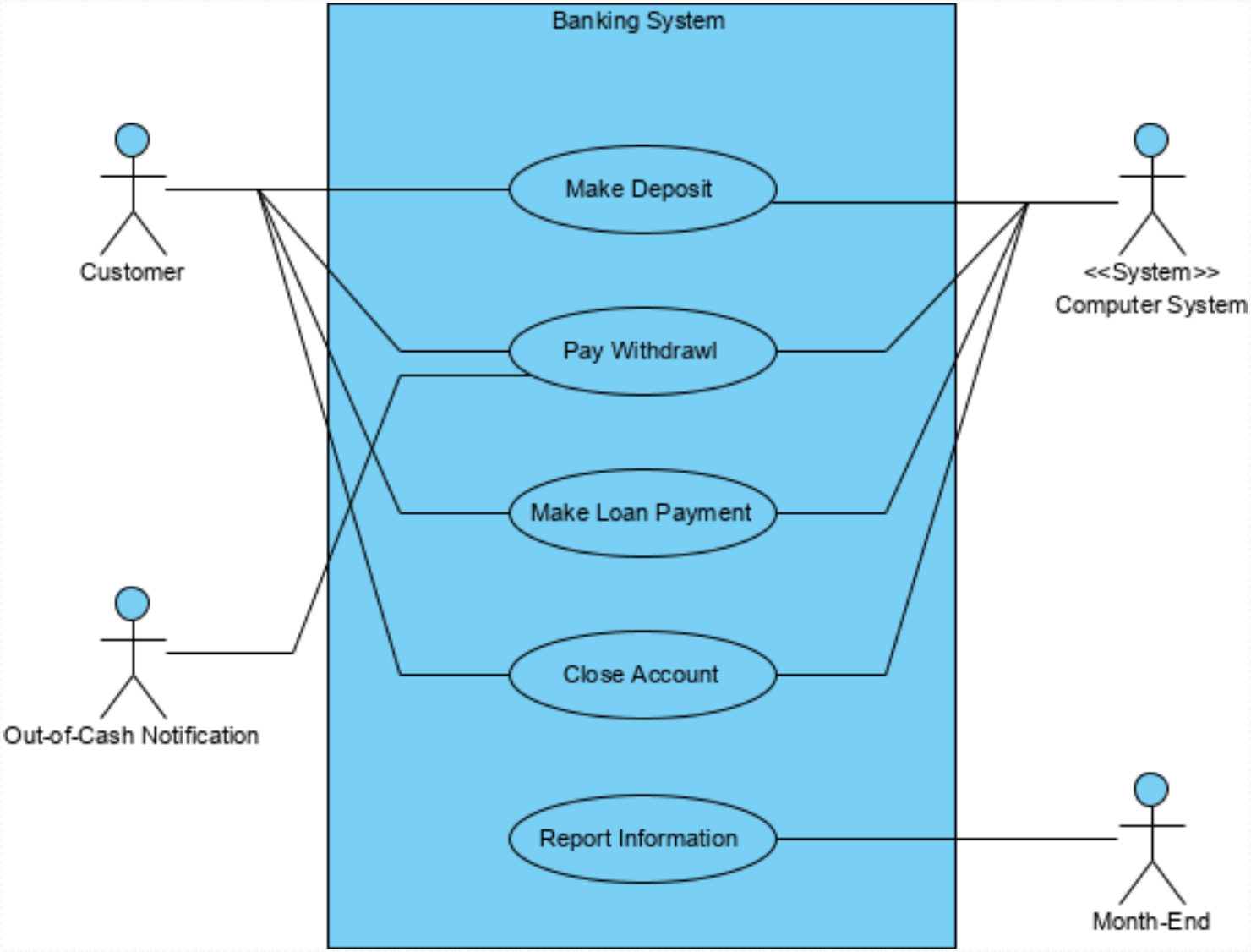
- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
  - i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.



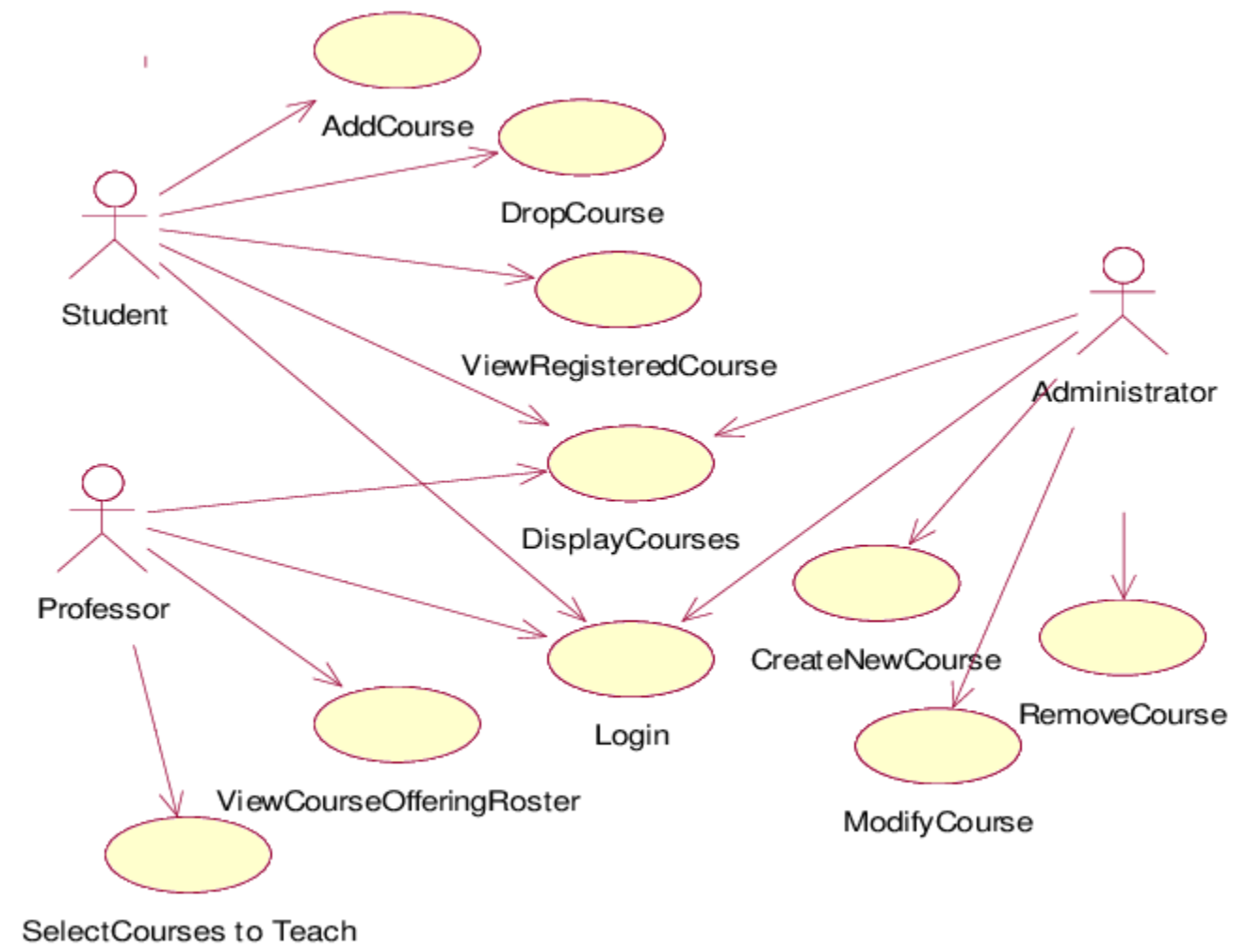
How to Identify Use Cases?

- Identifying the Use Cases, and then the scenario-based elicitation process carries on by asking what externally visible, observable value that each actor desires.
- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?.

Example


Banking System
Customer
Make Deposit
Pay Withdrawl
Make Loan Payment
Close Account
Report Information
<<System>>
Computer System
Out-of-Cash Notification
Month-End

Example

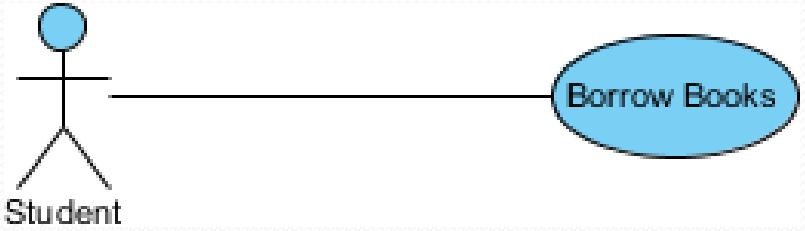

AddCourse
DropCourse
Student
ViewRegisteredCourse
Administrator
DisplayCourses
Professor
CreateNewCourse
RemoveCourse
Login
ViewCourseOfferingRoster
ModifyCourse
SelectCourses to Teach

Relationships

- Relationships are connections between model elements.
- **Types of relationships**:
  - Association
  - Generalization
  - Include
  - Extend
- Relationships between elements in use case diagram:
  - **Actor – Use case**: Association
  - **Actor – Actor**: Generalization
  - **Use case – Use case**: Generalization, Include, Extend

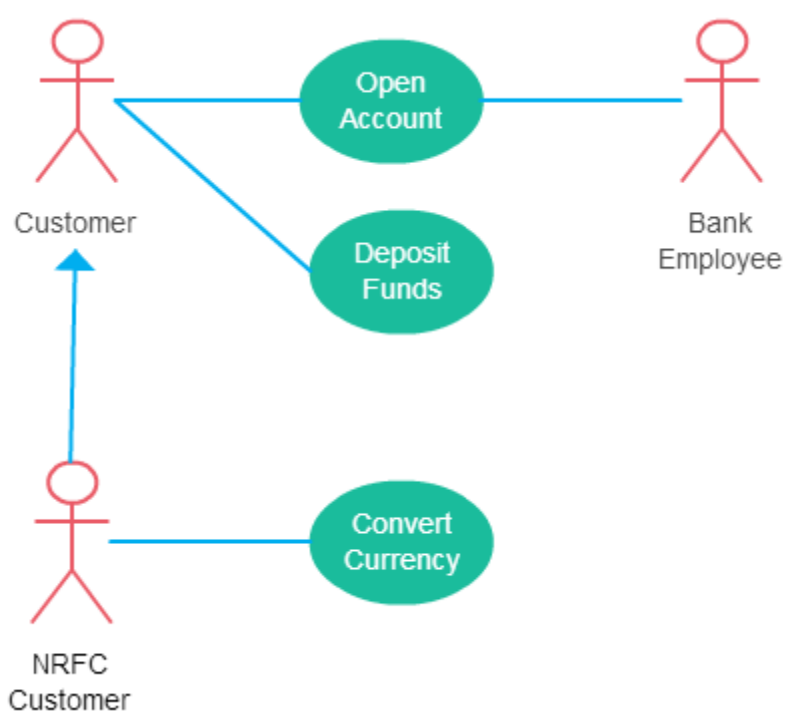
Association Relationship

- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

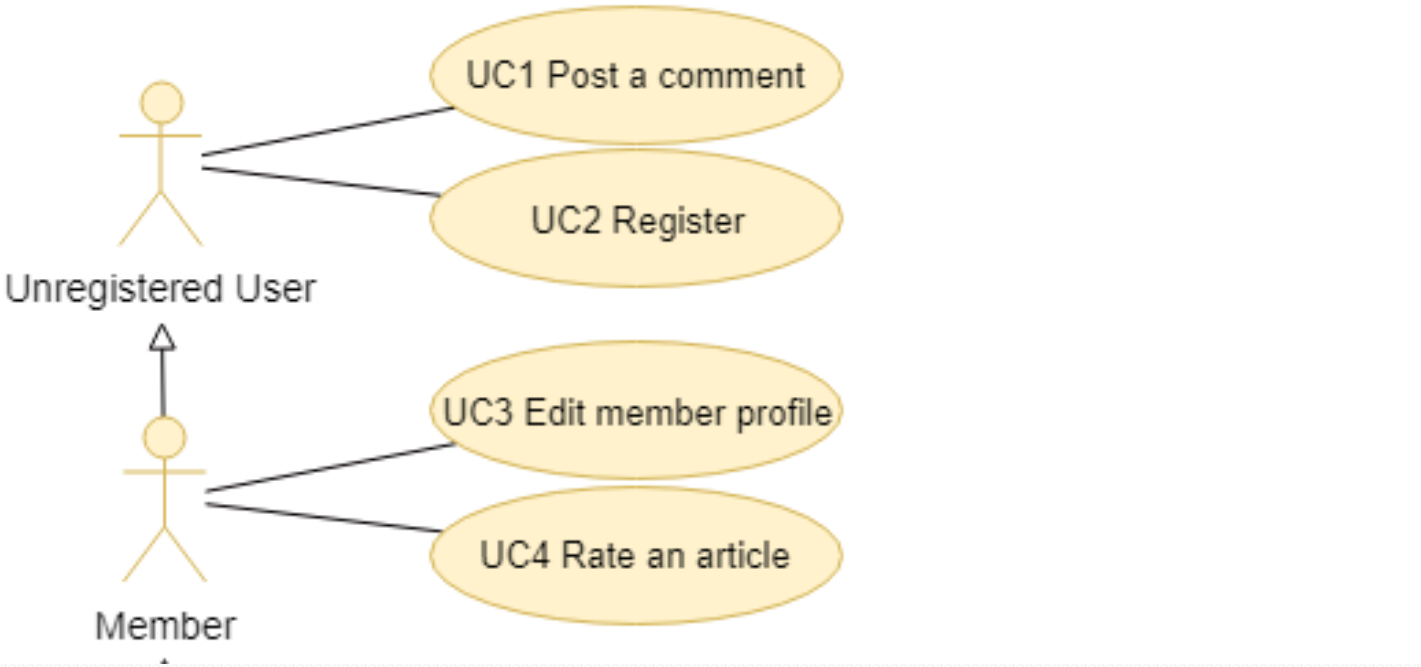


Actor Relationship

- Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role.



Actor Relationship

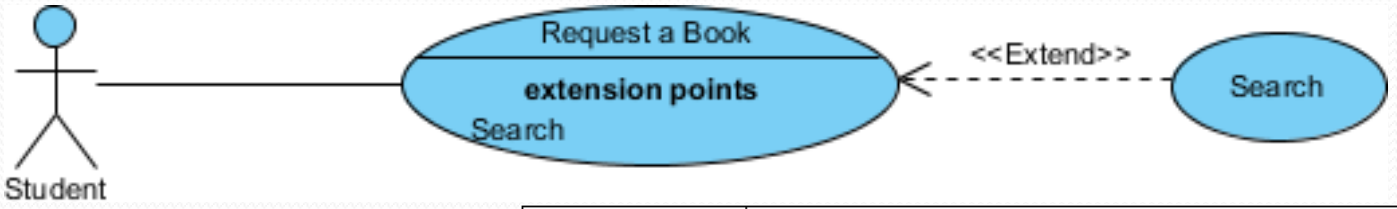


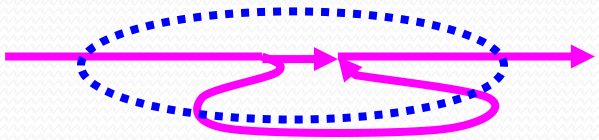
Can Member post a comment?

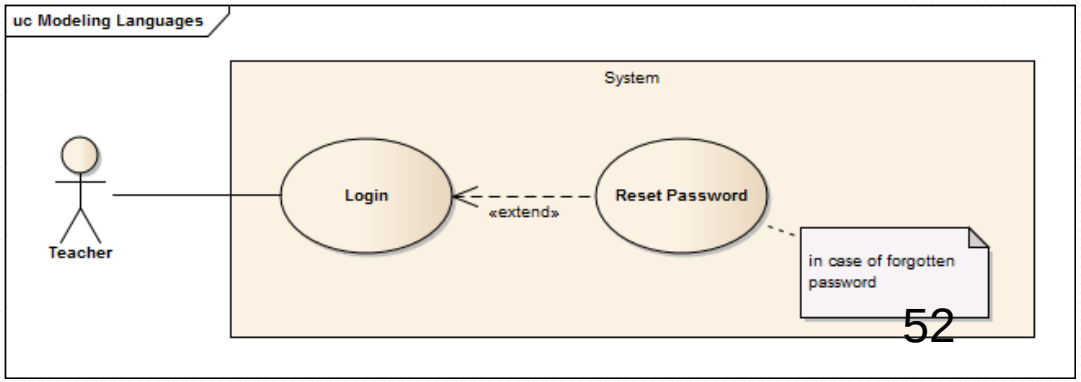
Use case Relationship

Extend

- Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
- The stereotype "<<extends>>" identifies as an extend relationship

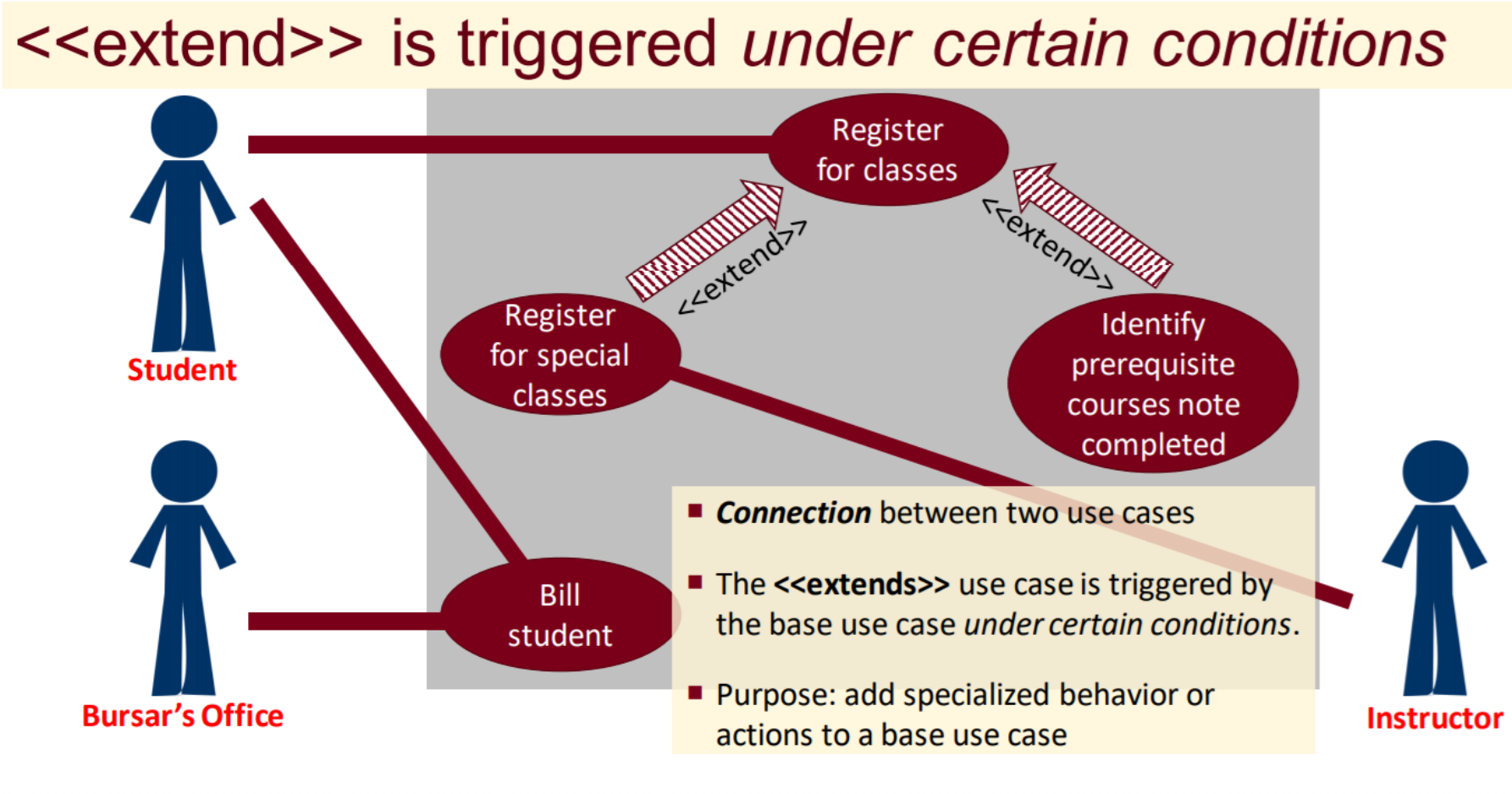






Use case Relationship

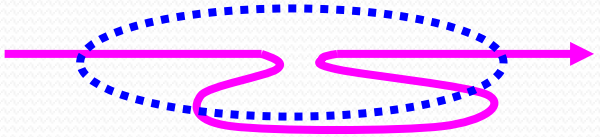
Extend



Use case Relationship

Include

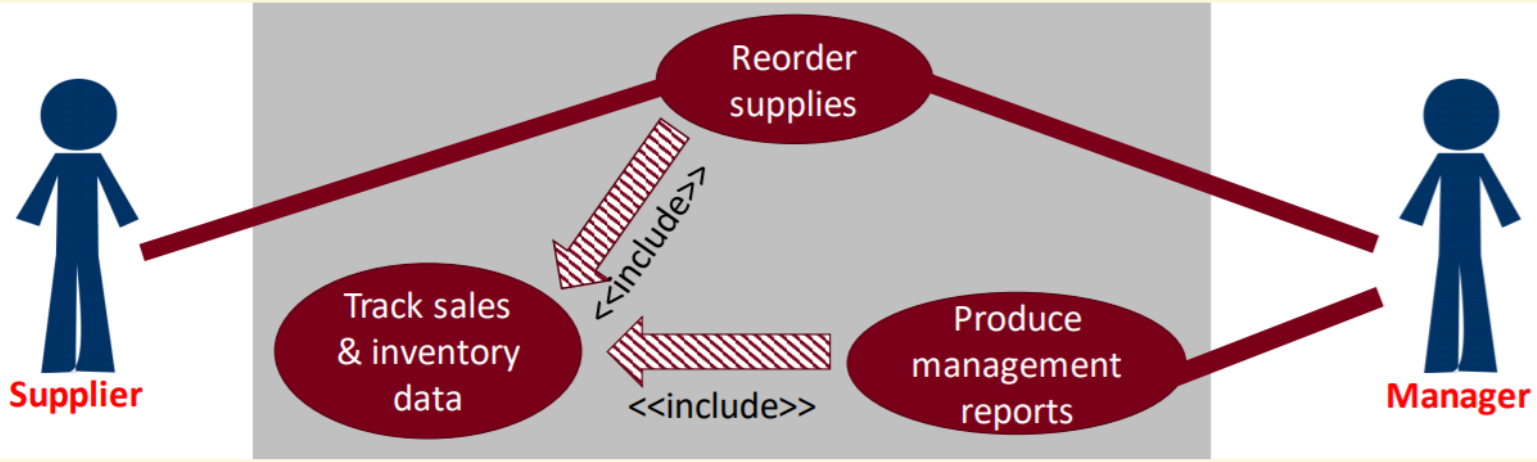
- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
- A use case includes the functionality described in another use case as a part of its business process flow.
- An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
- The stereotype "<<include>>" identifies the relationship as an include relationship.



Use case Relationship

Include

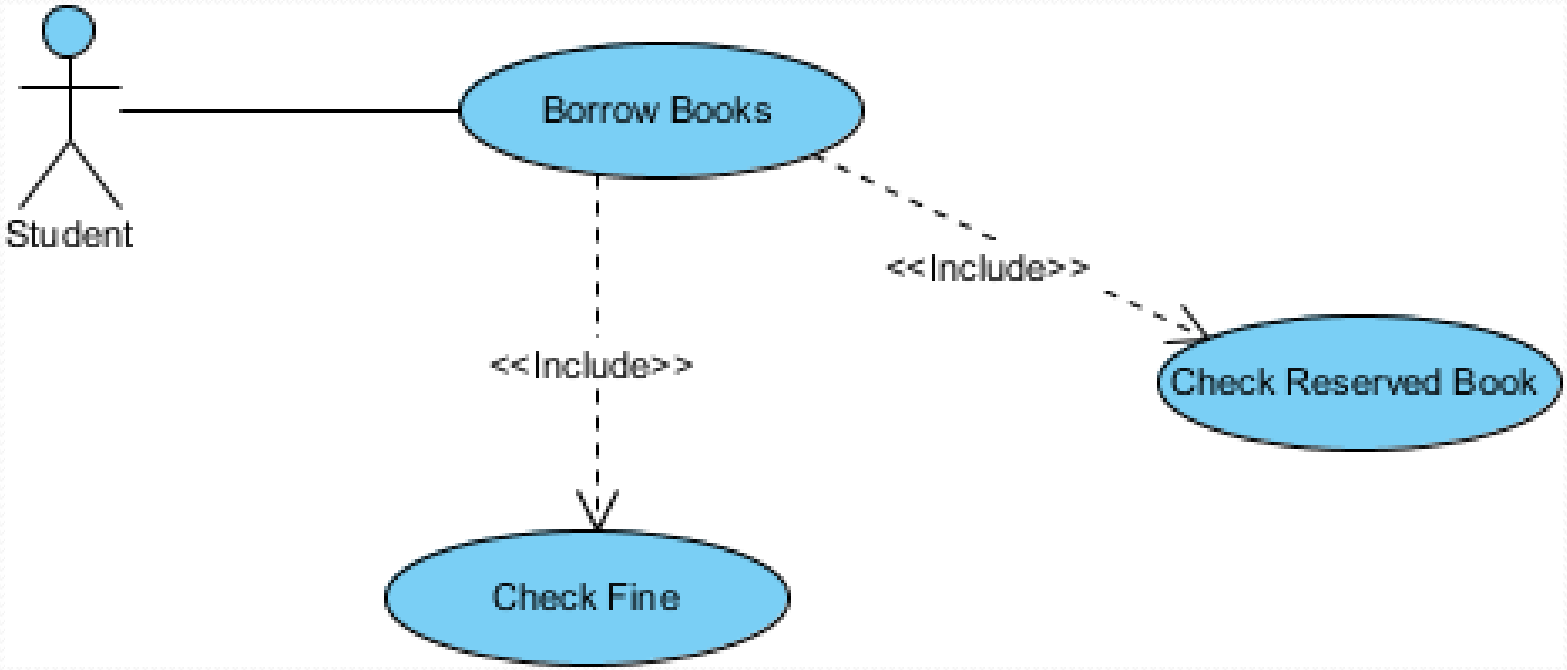
<<include>> is triggered *under all conditions*



- ***Connection*** between two use cases
- The **<<includes>>** use case is triggered by the base use case *under all conditions*.
- Purpose
  - Simplify the writing of the base use case (by carving out a new use case for complex logic), and/or
  - Allowing a specific function to be used by many use cases (simplifying diagram)
- [***Note***: **<<*include*>>** is called **<<*use*>>** in Visio]

Use case Relationship

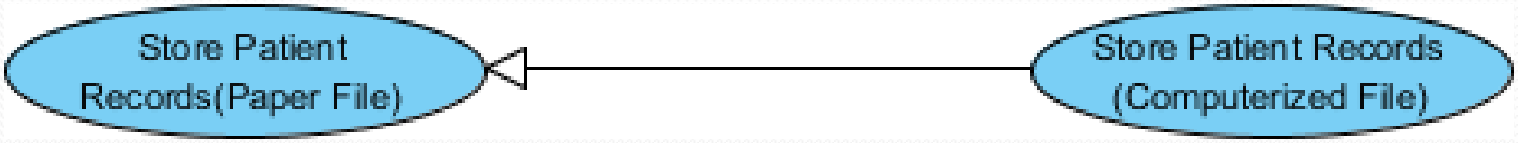
Include



Use case Relationship

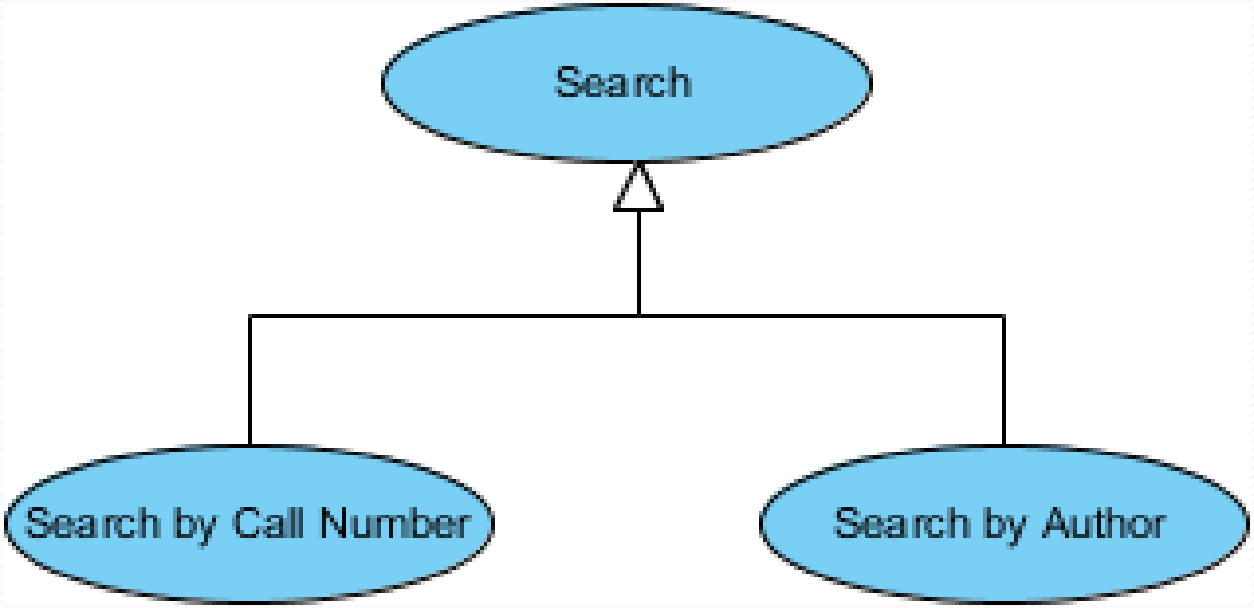
Generalization

- A generalization relationship is a parent-child relationship between use cases.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead.
- The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.



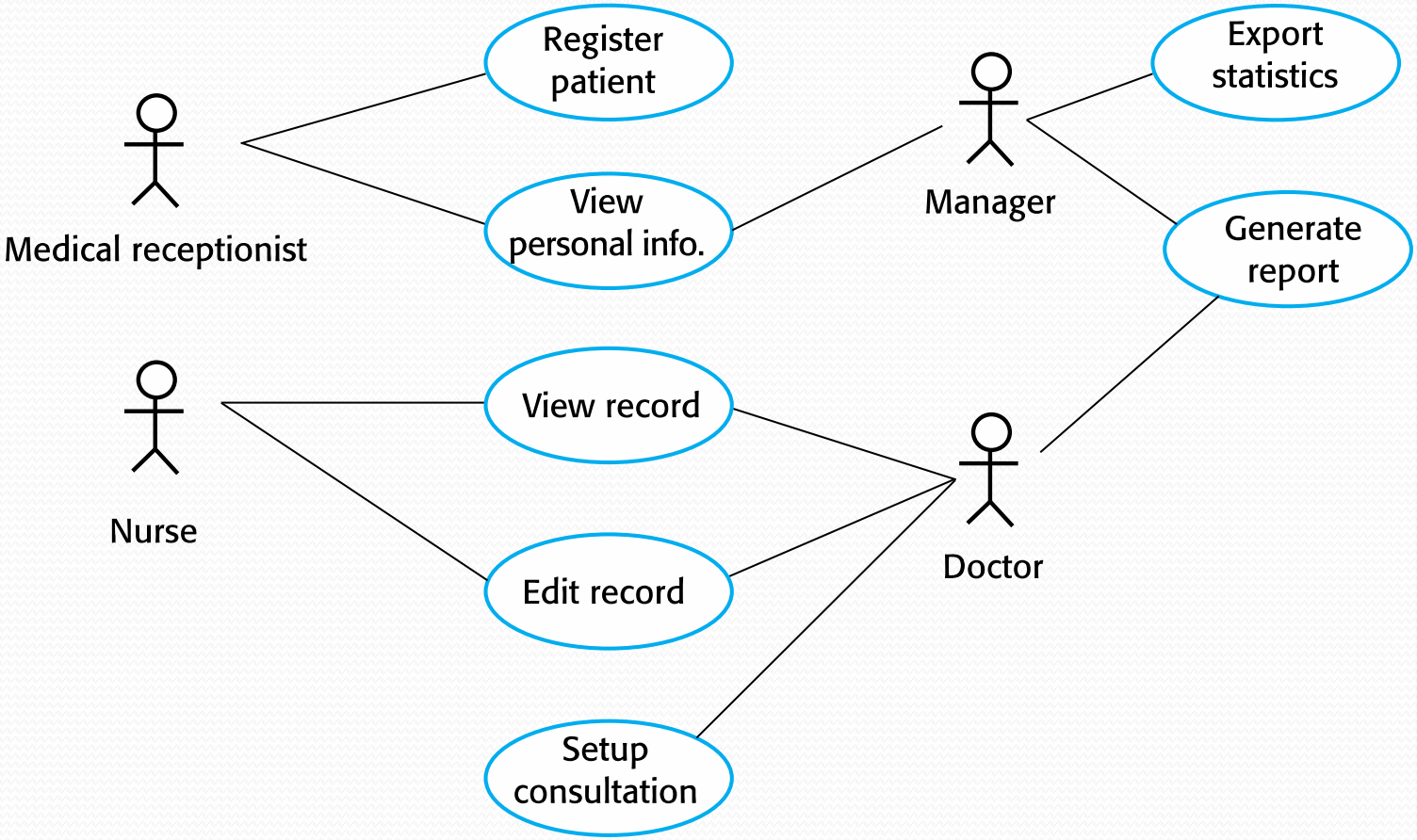
Use case Relationship

Generalization



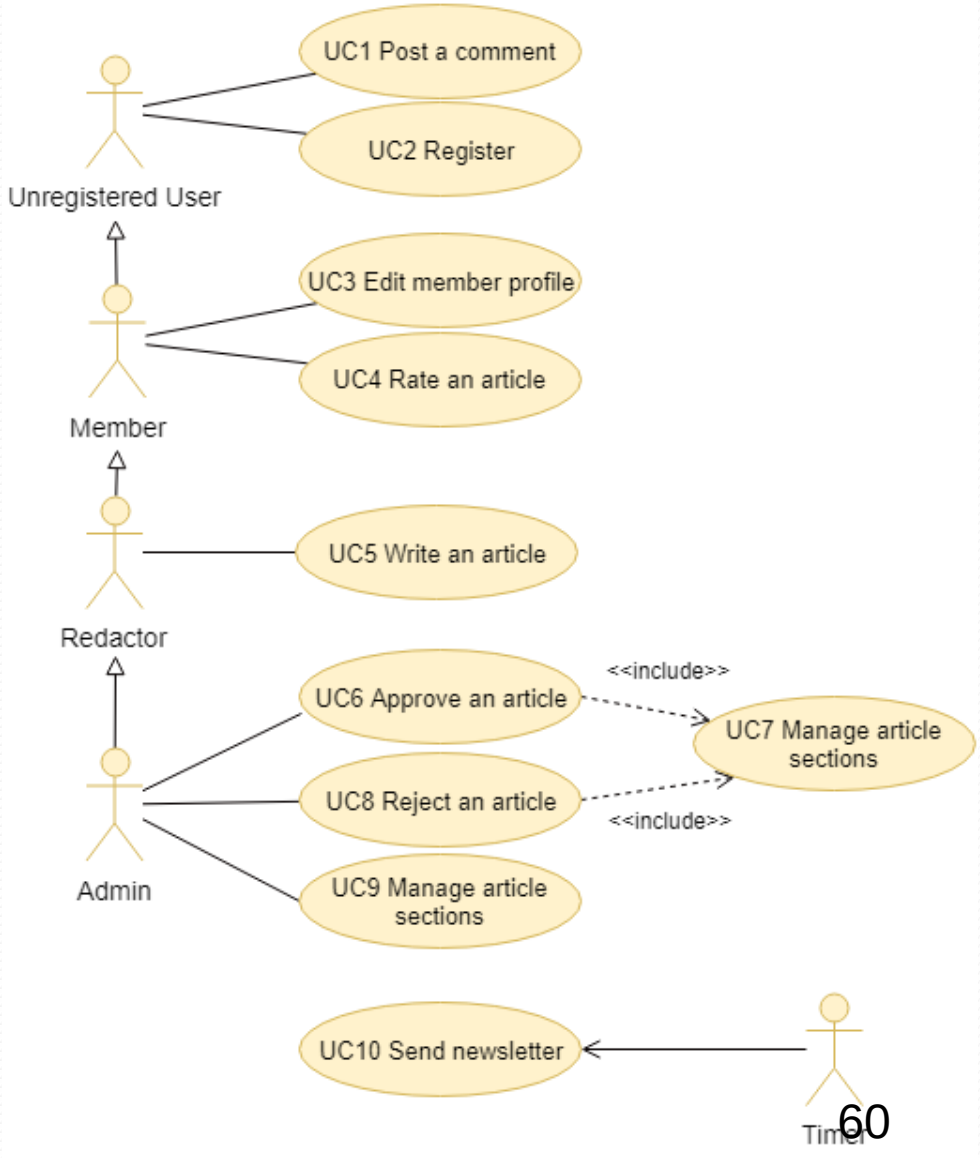
Use case Diagram

- A use case in the Mentcare system



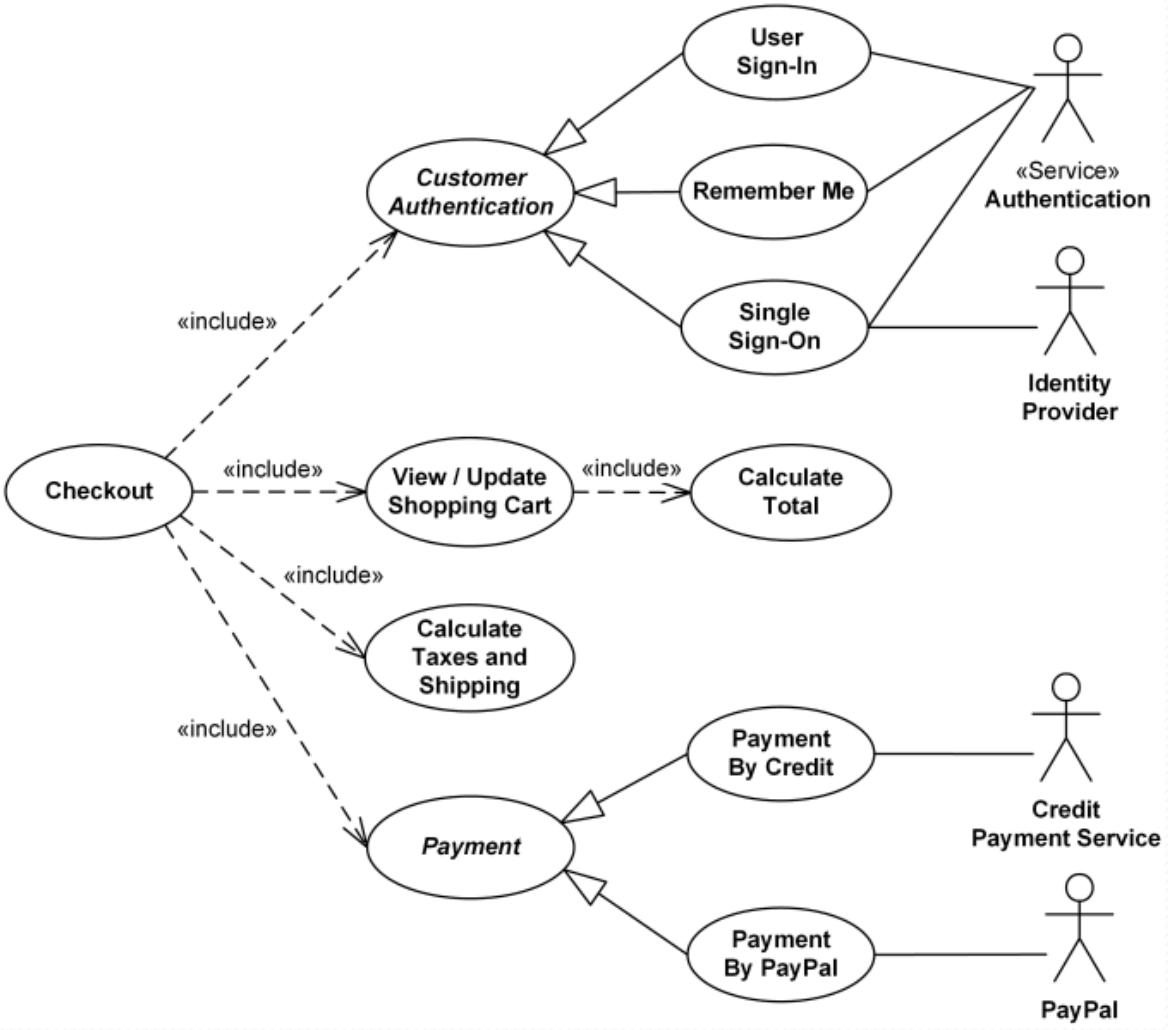
Use case Diagram

- A use case in a CMS



Use case Diagram

- A use case in a Online shopping

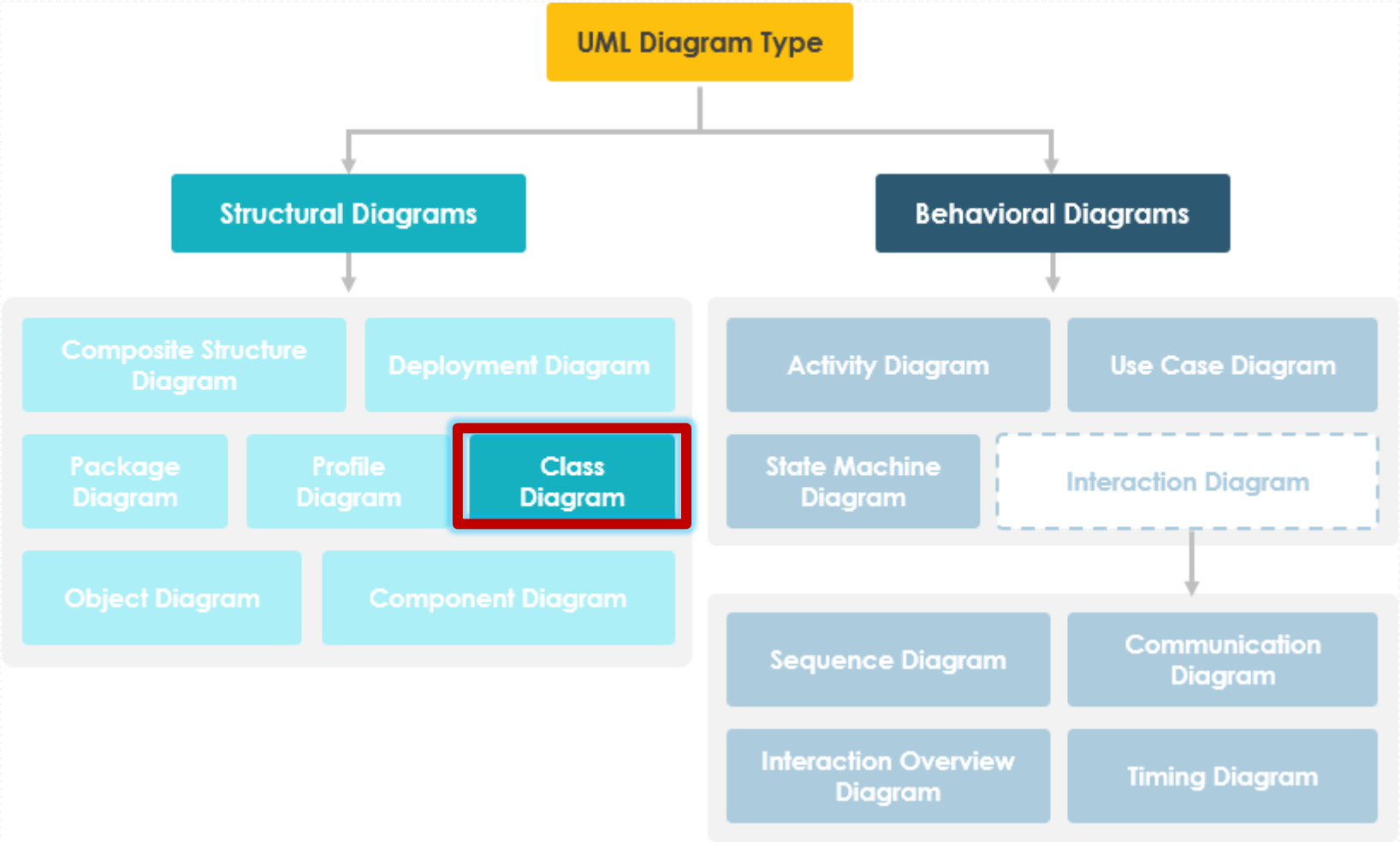


Structural models

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, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

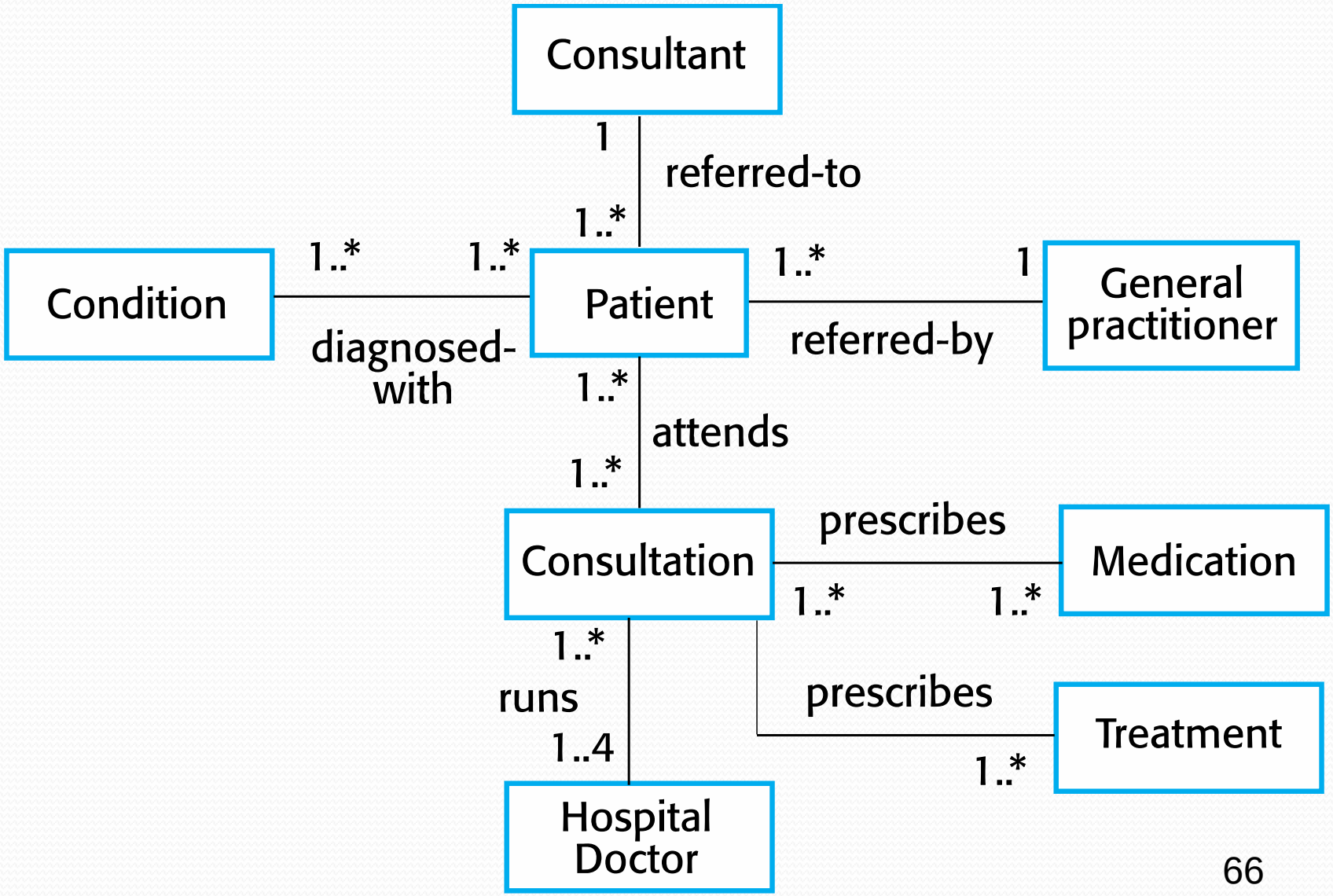
UML diagram types



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 SE process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Class diagrams

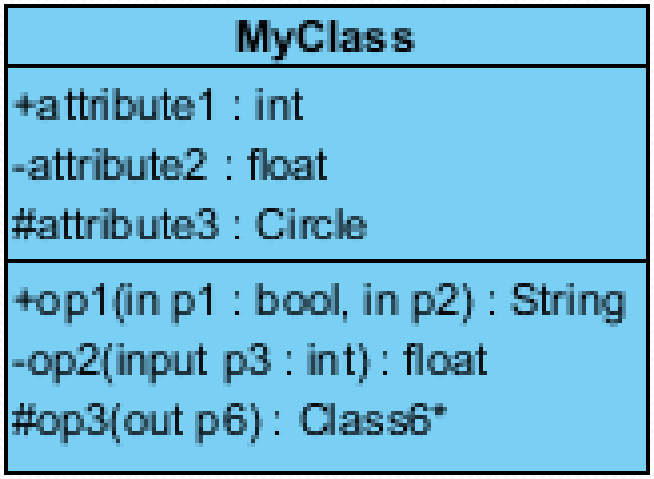


Class notation

A class notation consists of three parts:

- **Class Name**
  - The name of the class appears in the first partition.
- **Class Attributes**
  - Attributes are shown in the second partition.
  - The attribute type is shown after the colon.
  - Attributes map onto member variables (data members) in code.
- **Class Operations (Methods)**
  - Operations are shown in the third partition. They are services the class provides.
  - The return type of a method is shown after the colon at the end of the method signature.
  - The return type of method parameters is shown after the colon following the parameter name.
  - Operations map onto class methods in code

Class notation

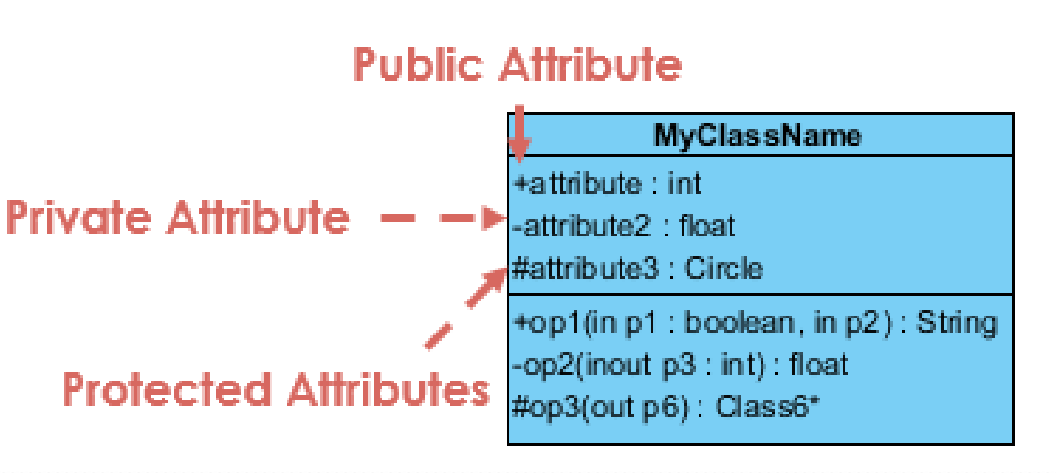


- What is the class name?
- How many attributes?
- How many methods?

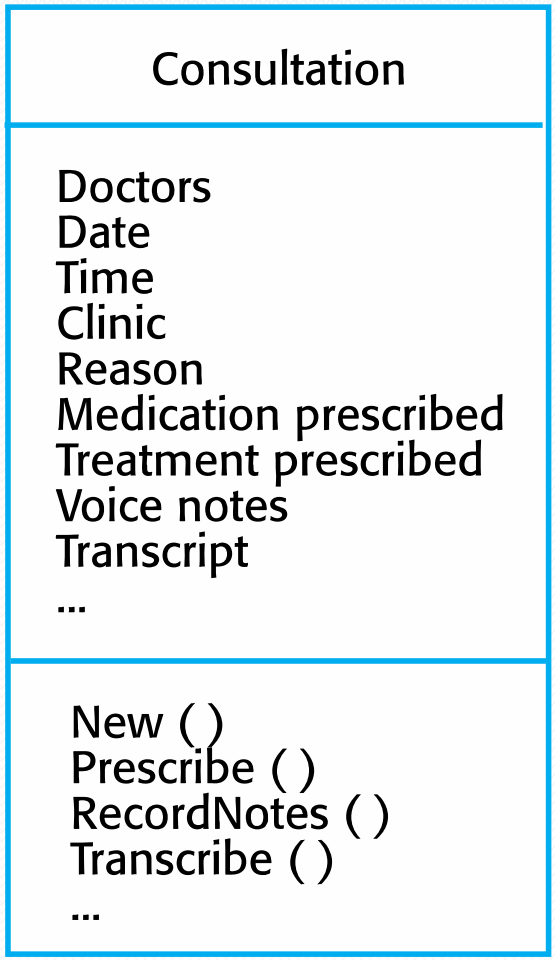
Class notation

Class Visibility

- The +, -, # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.



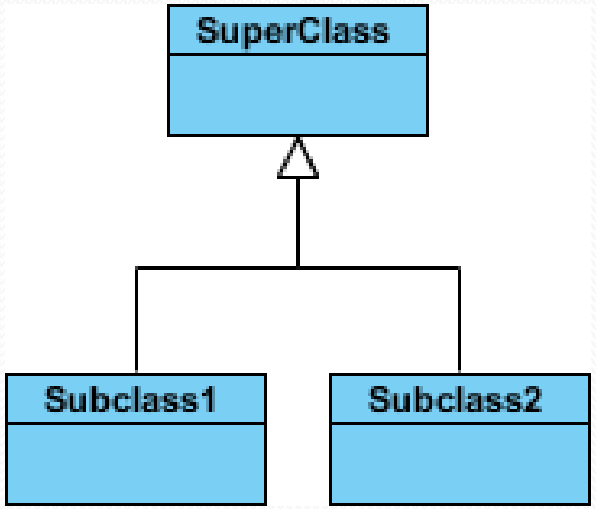
The Consultation class



Class Relationships

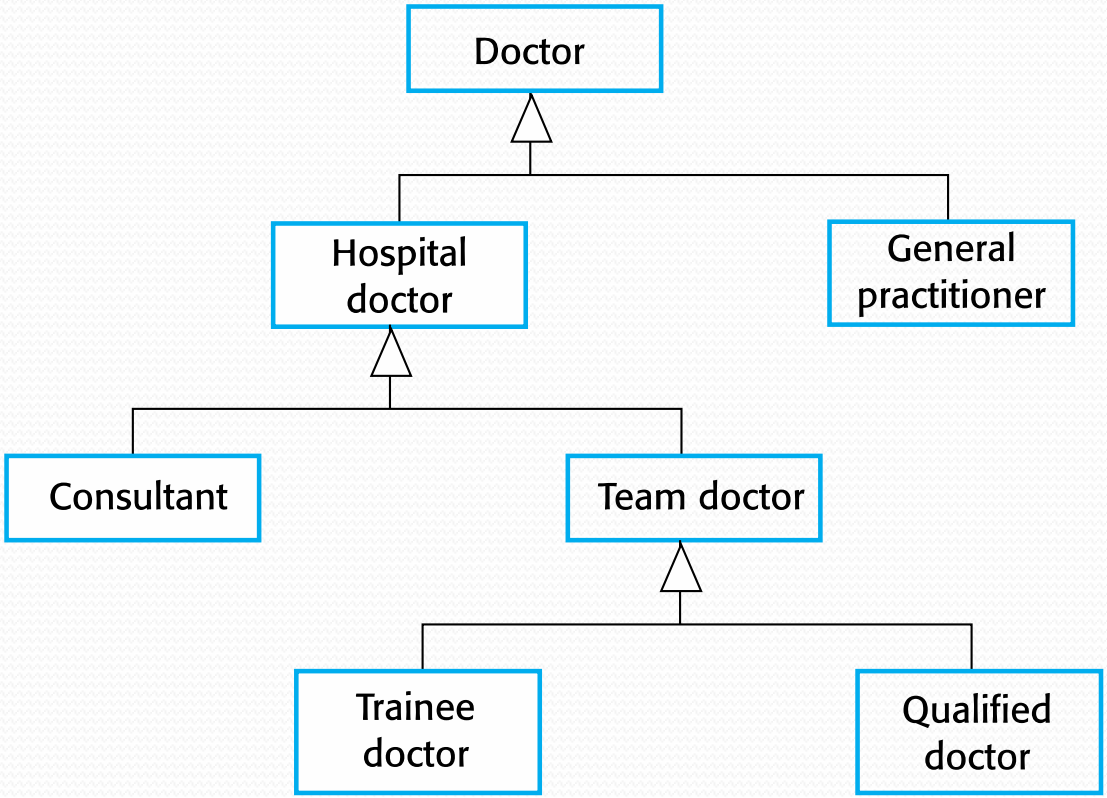
Inheritance (or Generalization):

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class



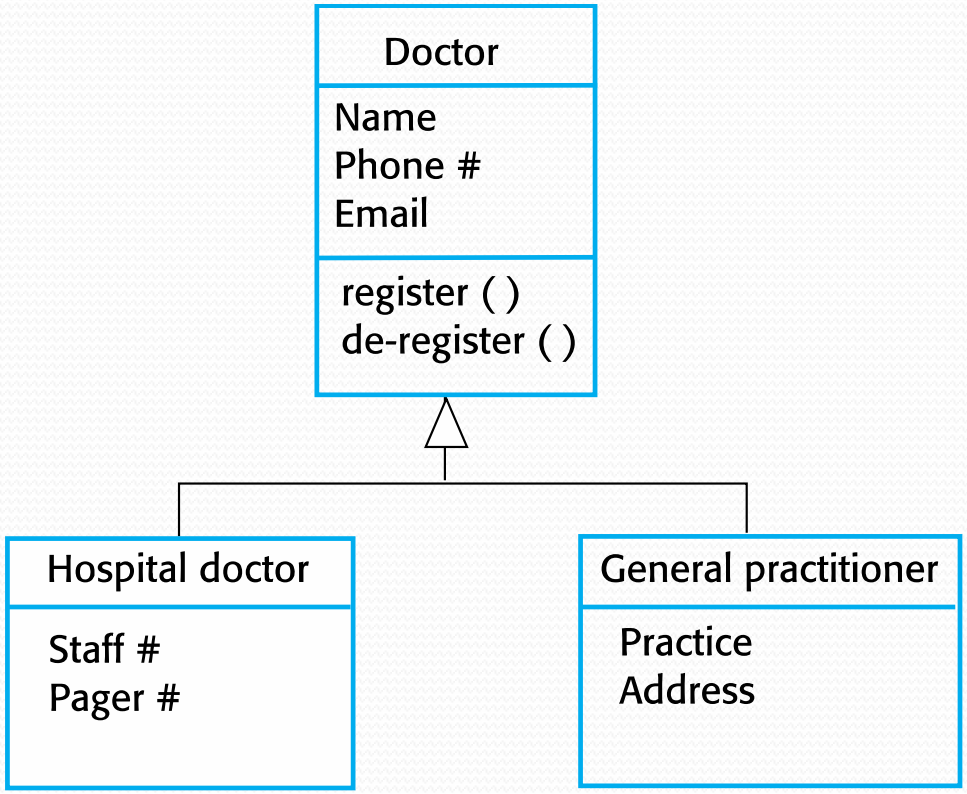
Class Relationships

Generalization



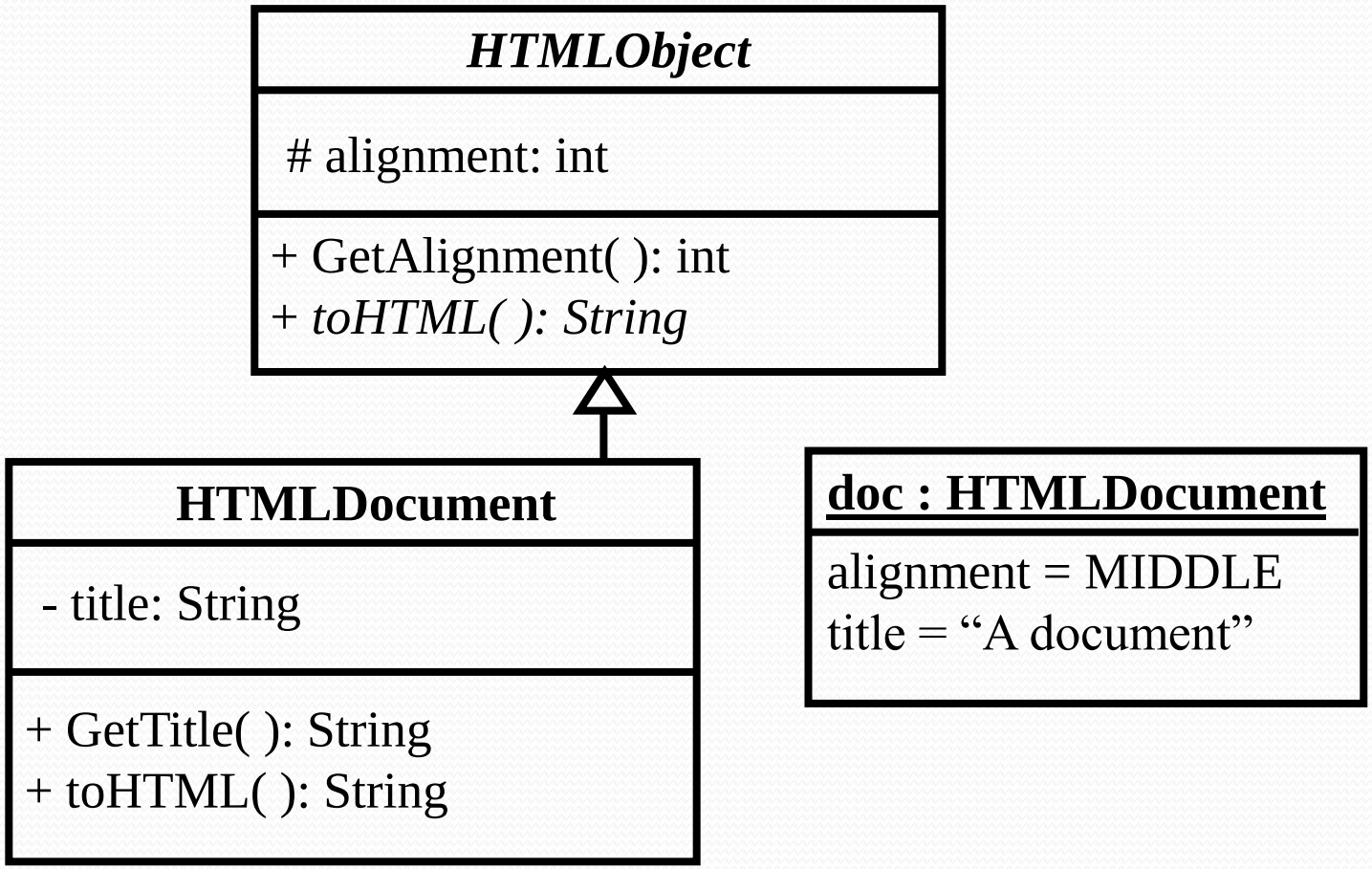
Class Relationships

Generalization



Class Relationships

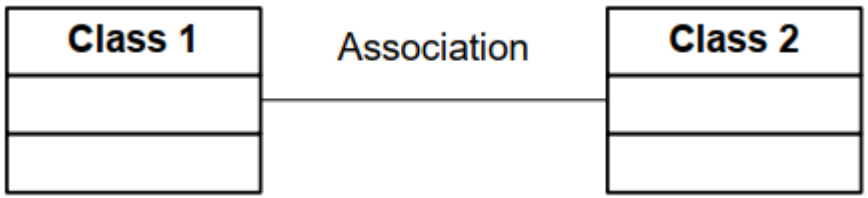
Generalization

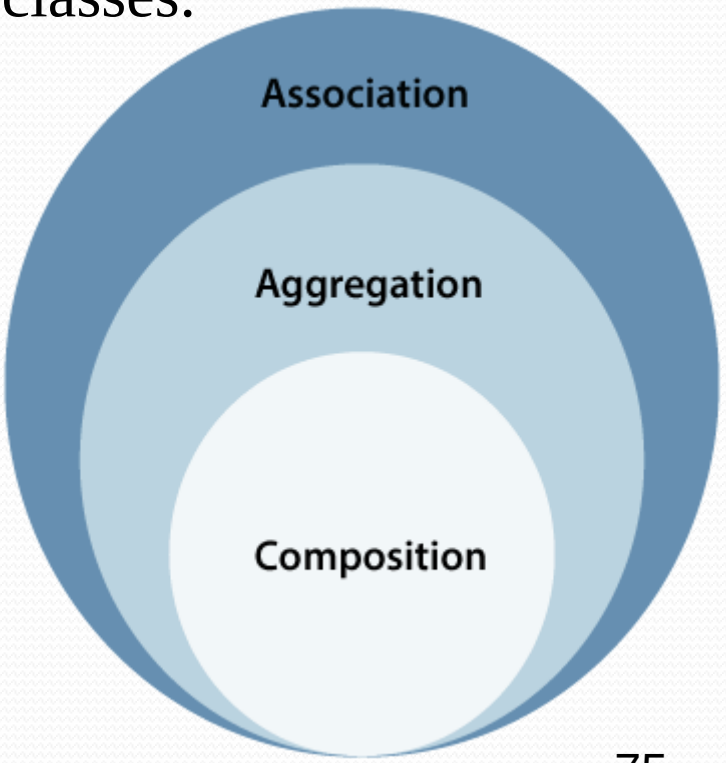


Class Relationships

Association:

- Association is a structural relationship in which different objects are linked within the system
- A structural link between two peer classes.

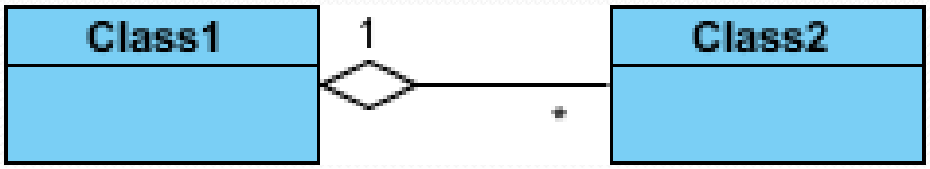




Class Relationships

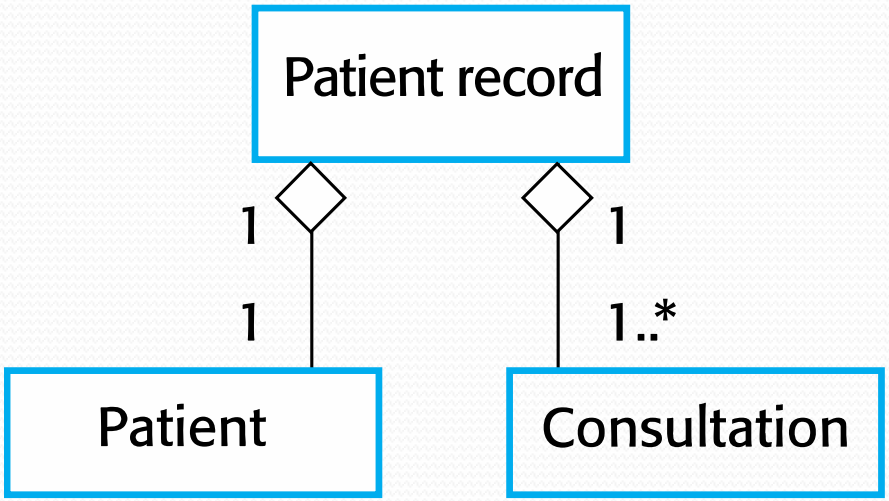
Aggregation:

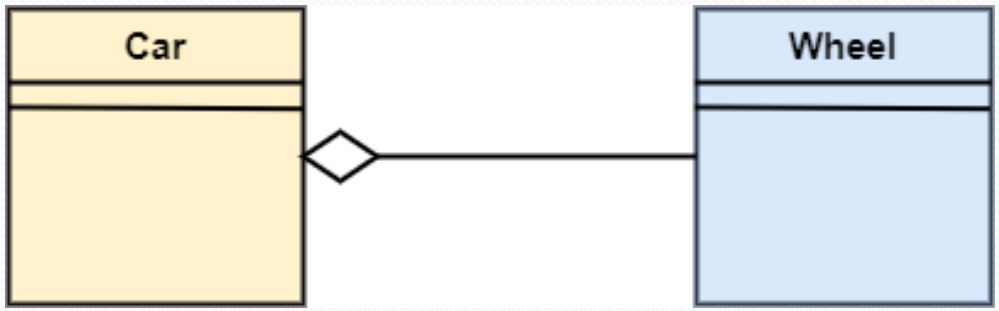
- A special type of association. It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite.



Class Relationships

Aggregation

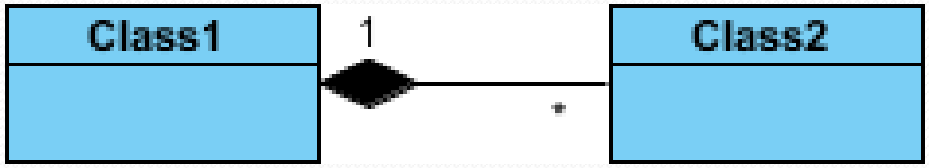




Class Relationships

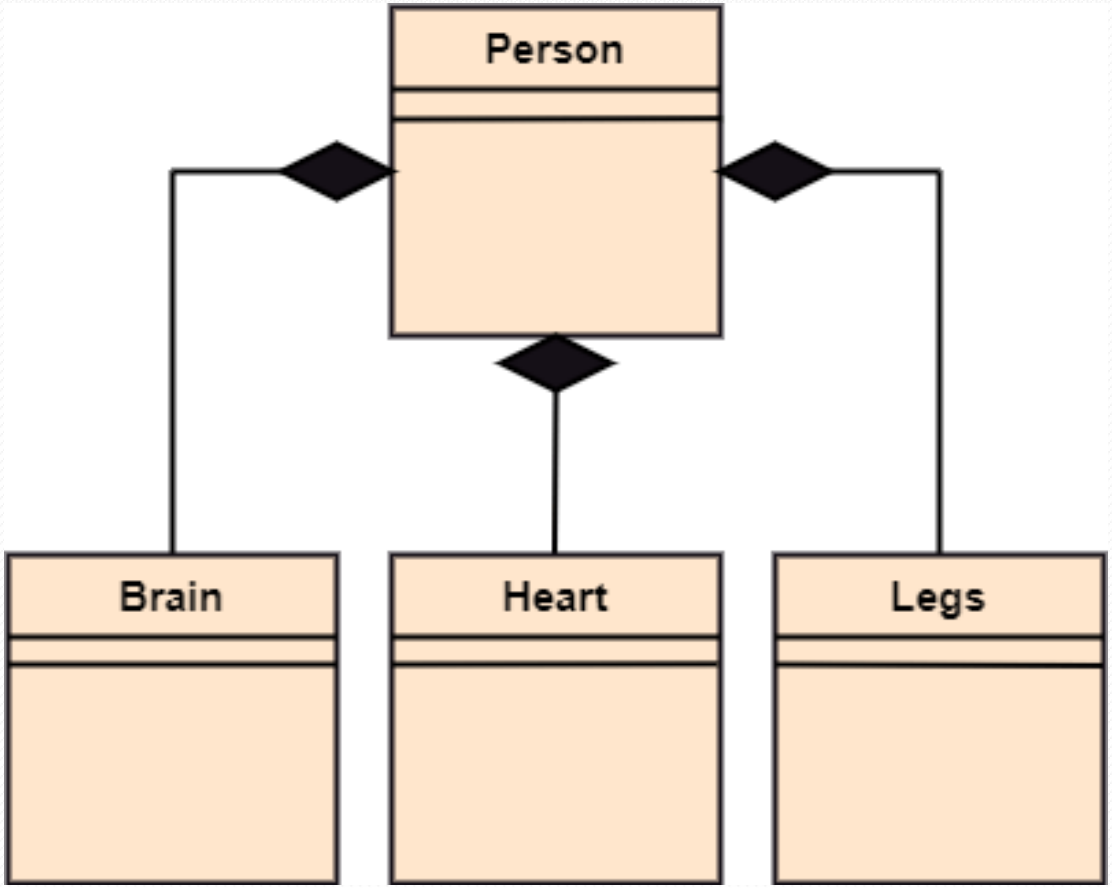
Composition:

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite



Class Relationships

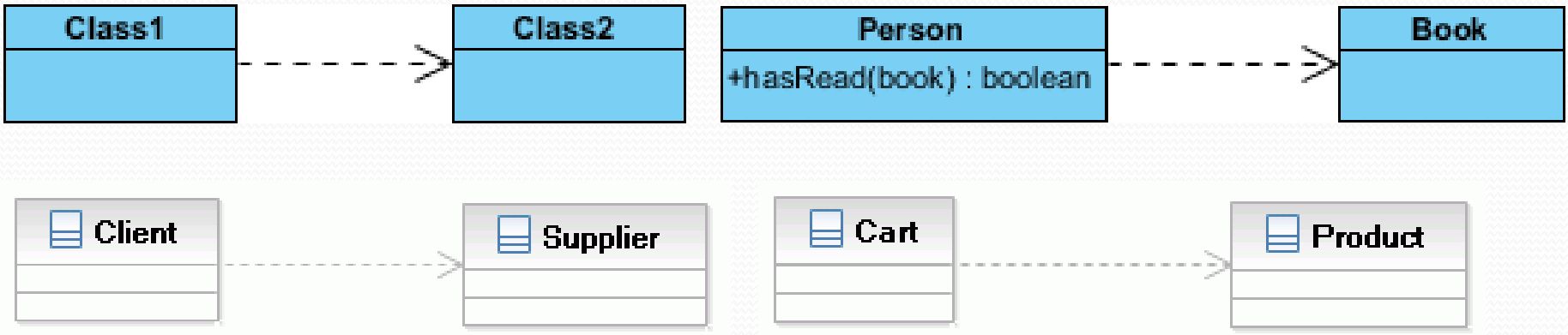
Composition:



Class Relationships

Dependency:

- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on (uses) Class2
- A dashed line with an open arrow

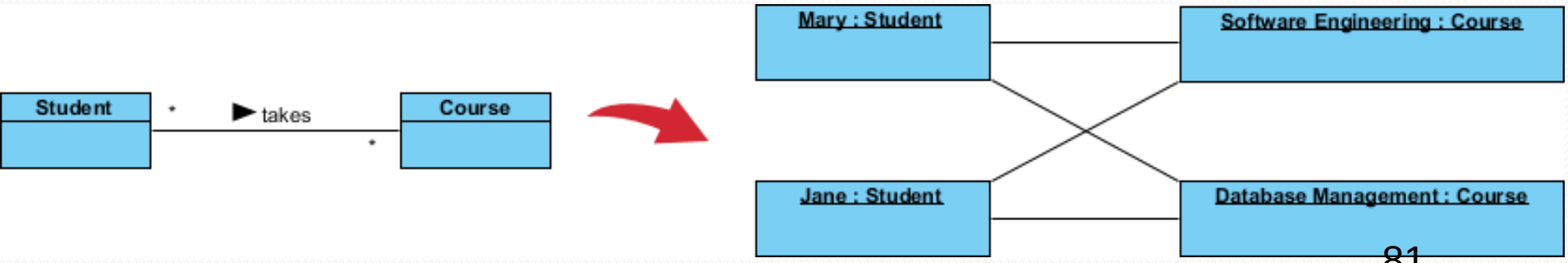


Class Relationships

Multiplicity

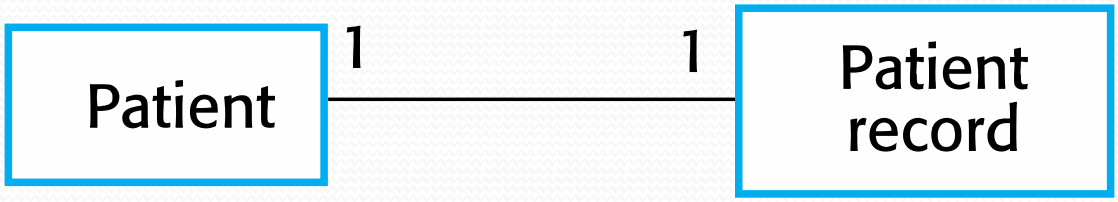
How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

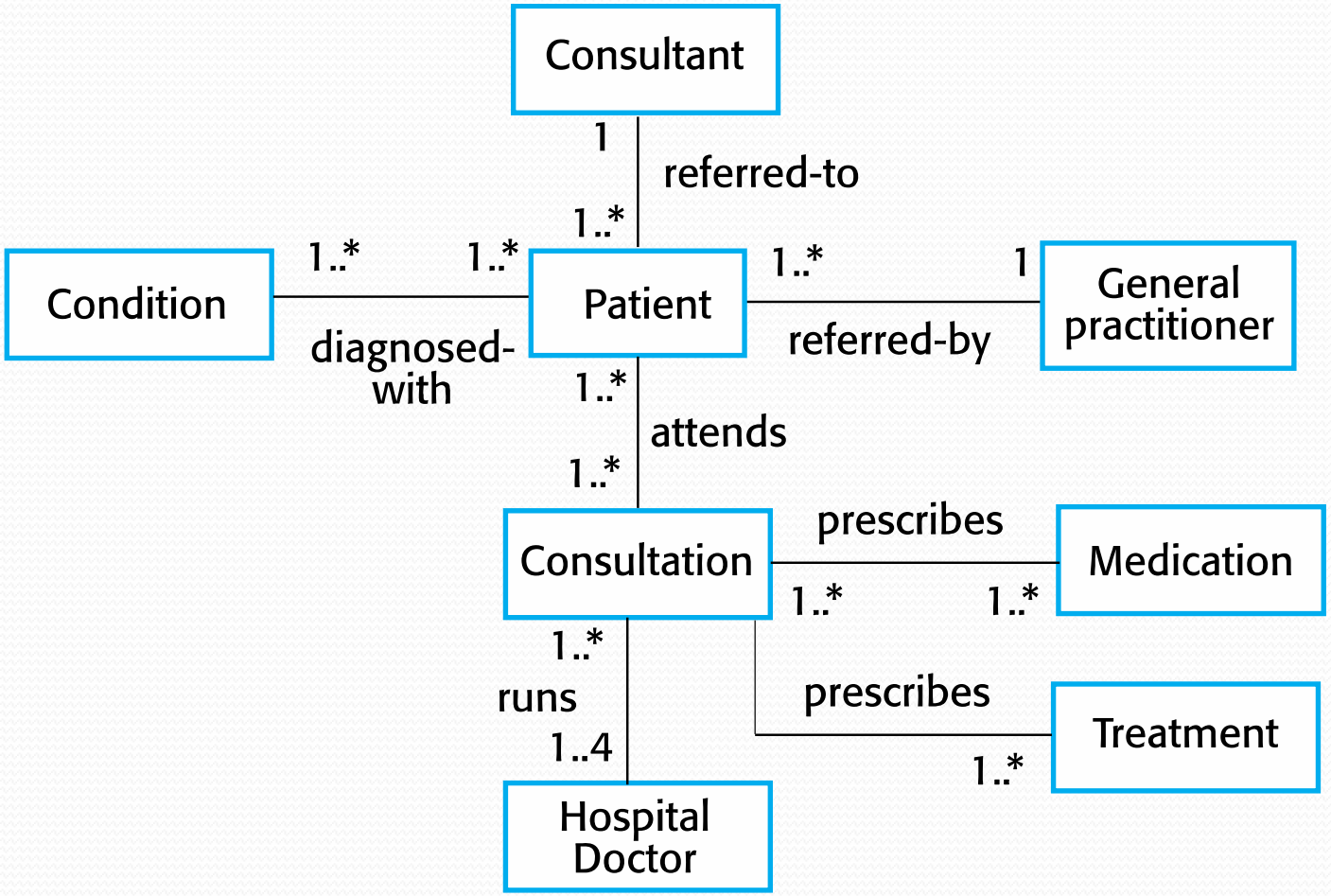


Class Relationships

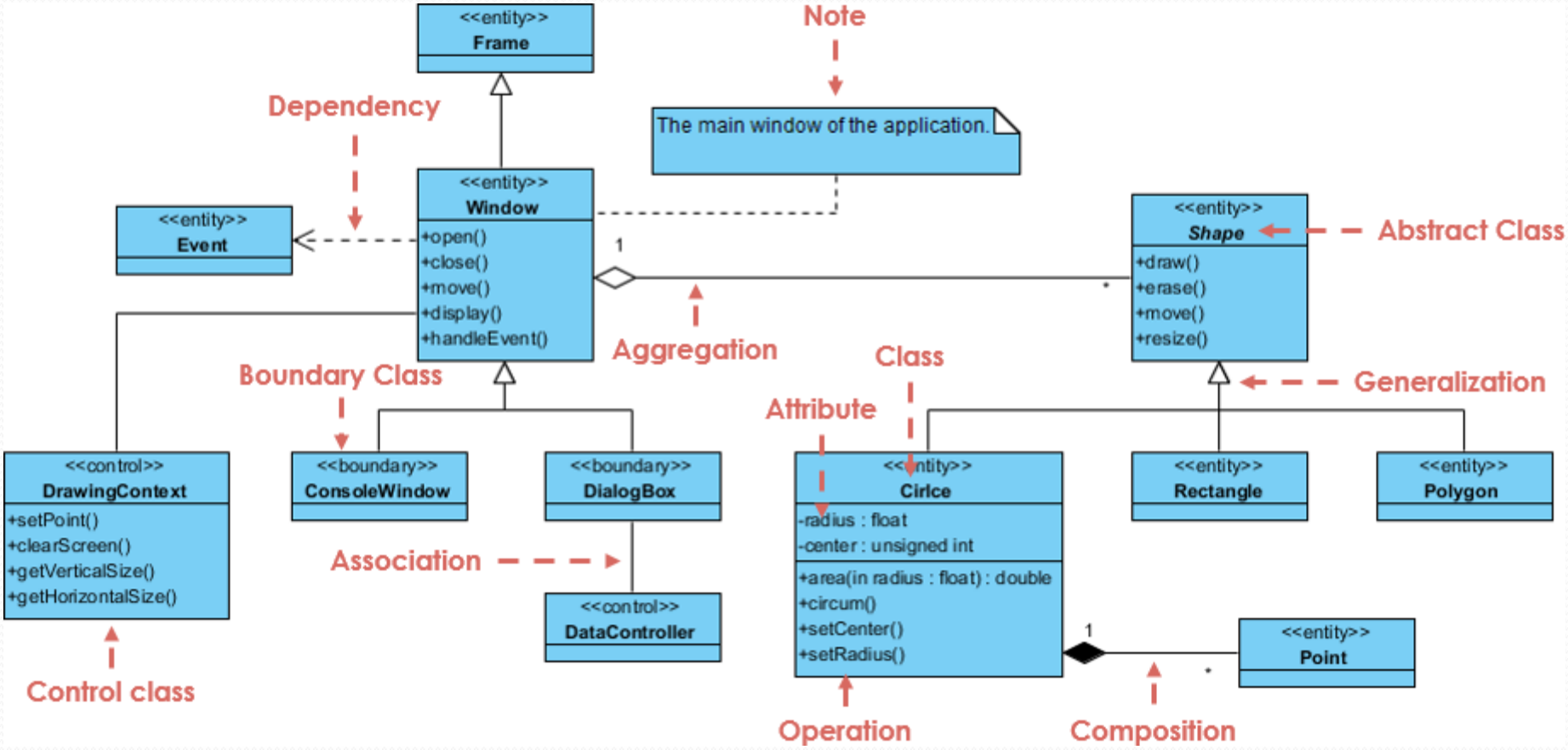
Multiplicity



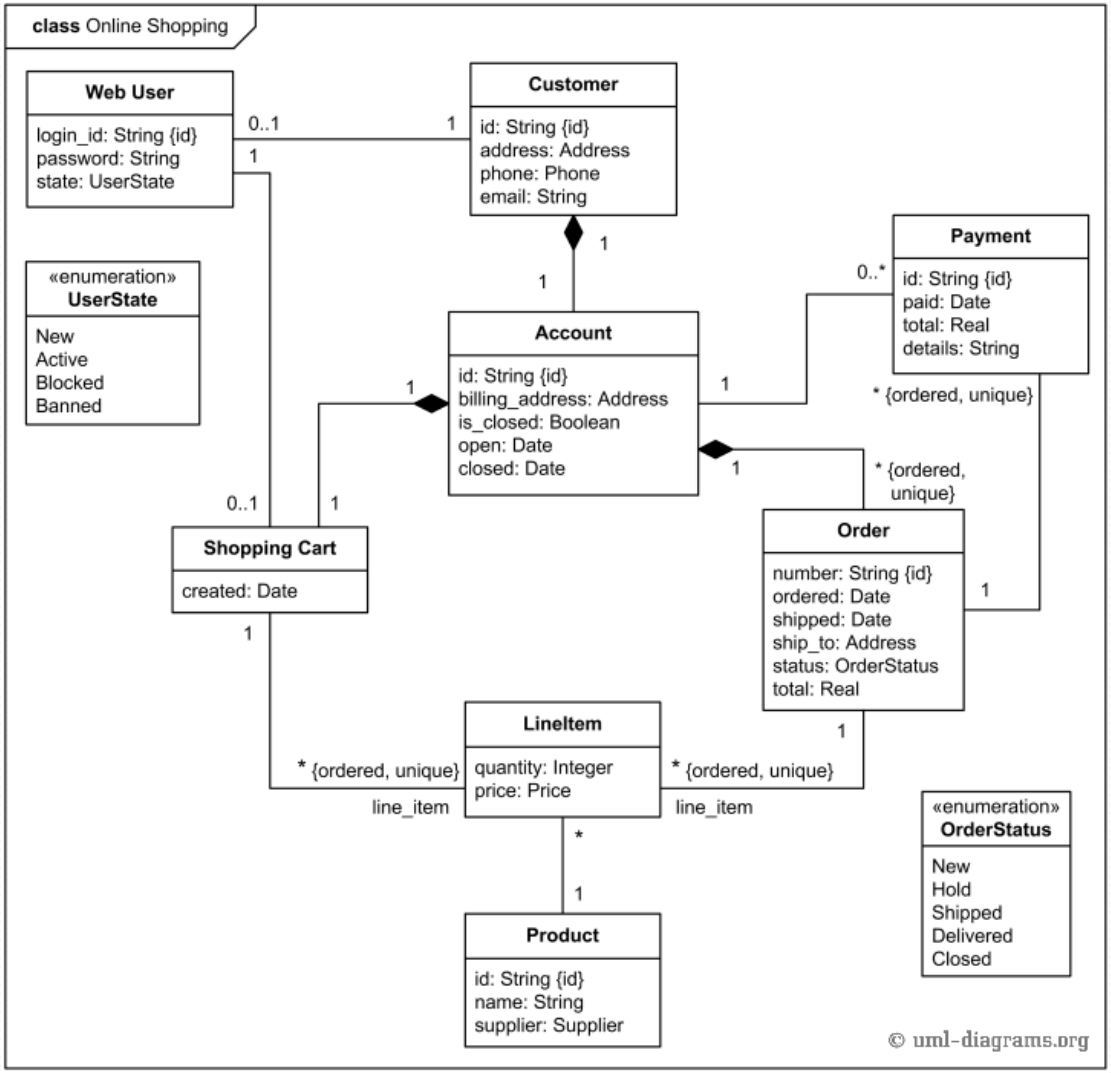
Classes and associations in the MHC-PMS



Class Diagram Example



Class Diagram Example



Key points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behaviour.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.

Key points

- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.