

Scribe Notes

Week-2

Section-A

Members:

Kannuru Dinesh(S20170010064)

Kartik Kapoor(S20170010066)

Riya Jain(S20170010126)

Saggurthi Jagadeesh Sumanth(S20170010135)

Bittu Kumar Ray(S20170010027)

UML class diagrams

Class diagrams are one of the most useful types of diagrams in UML as they clearly map out the structure of a particular system by modeling its classes, attributes, operations, and relationships between objects.

What is a class diagram in UML?

The Unified Modeling Language (UML) can help you model systems in various ways. One of the more popular types in UML is the class diagram. Popular among software engineers to document software architecture, class diagrams are a type of structure diagram because they describe what must be present in the system being modeled.

UML was set up as a standardized model to describe an object-oriented programming approach. Since classes are the building block of objects, class diagrams are the building blocks of UML. The various components in a class diagram can represent the classes that will actually be programmed, the main objects, or the interactions between classes and objects.

The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section expresses the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

Benefits of class diagrams

Class diagrams offer a number of benefits. UML class diagrams are used to:

- Illustrate data models for information systems, no matter how simple or complex.
- Better understand the general overview of the schematics of an application.
- Visually express any specific needs of a system and disseminate that information throughout the business.
- Create detailed charts that highlight any specific code needed to be programmed and implemented to the described structure.
- Provide an implementation-independent description of types used in a system that are later passed between its components.

Basic components of a class diagram

The standard class diagram is composed of three sections:

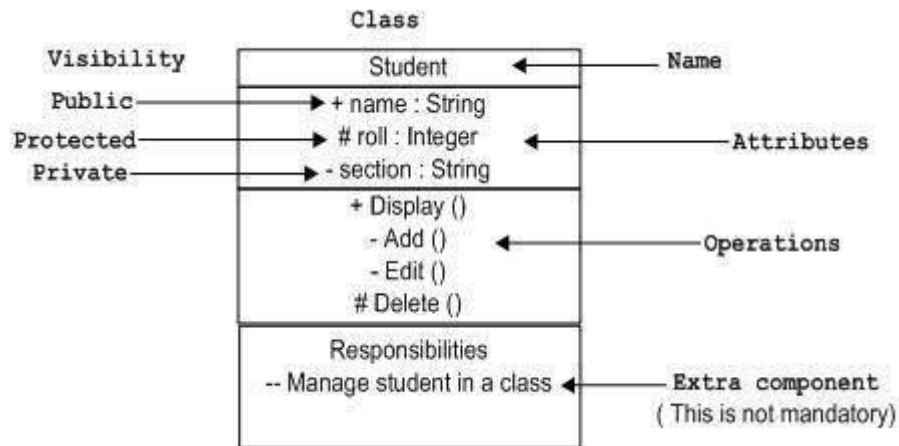
- **Upper section:** Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- **Middle section:** Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.
- **Bottom section:** Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data.

Notations:

1. UML Class Notation:

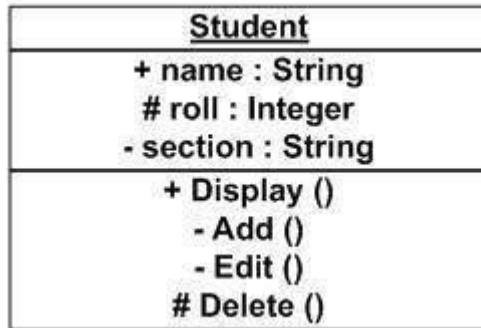
UML *class* is represented by the following figure. The diagram is divided into four parts.

- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.



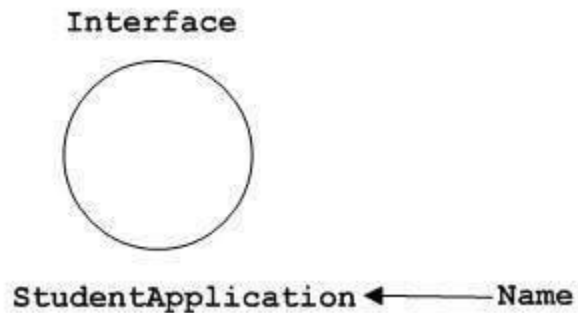
2. Object Notation:

The *object* is represented in the same way as the class. The only difference is the *name* which is underlined as shown in the following figure.



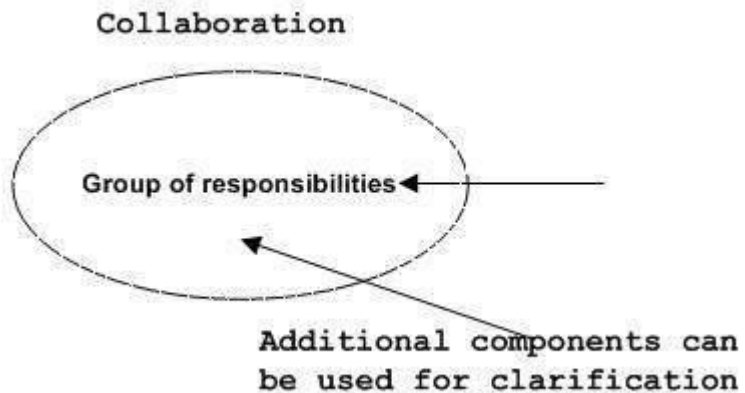
3. Interface Notation

Interface is represented by a circle as shown in the following figure. It has a name which is generally written below the circle.



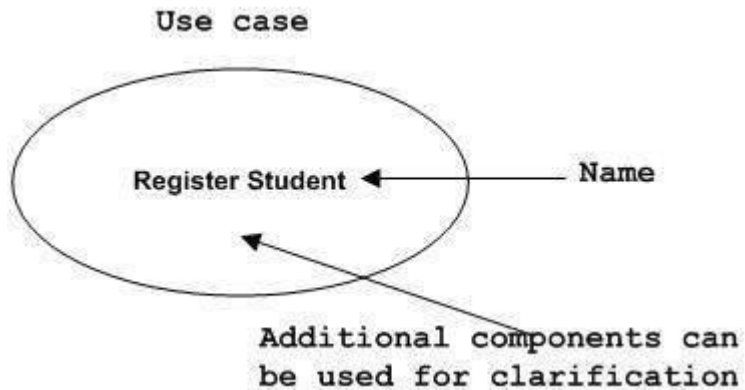
4. Collaboration Notation

Collaboration is represented by a dotted ellipse as shown in the following figure. It has a name written inside the ellipse.



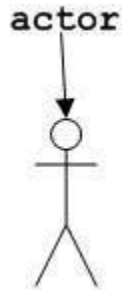
5. Use Case Notation

Use case is represented as an ellipse with a name inside it. It may contain additional responsibilities.



6. Actor Notation

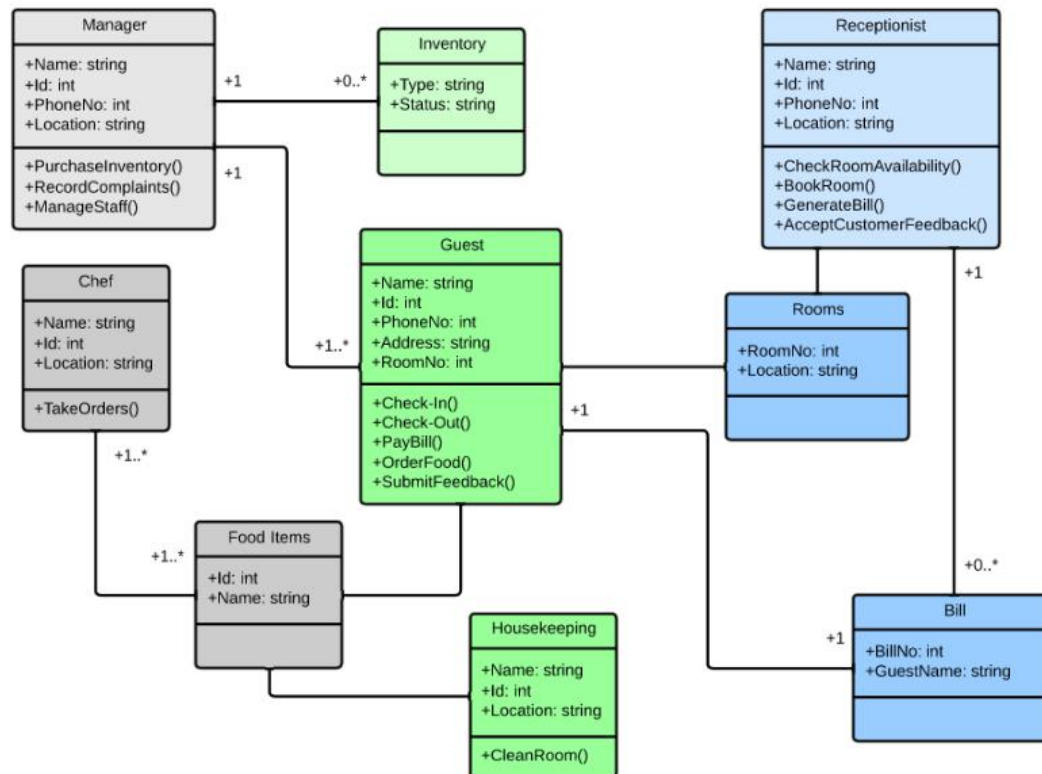
An actor can be defined as some internal or external entity that interacts with the system.



Class diagram examples

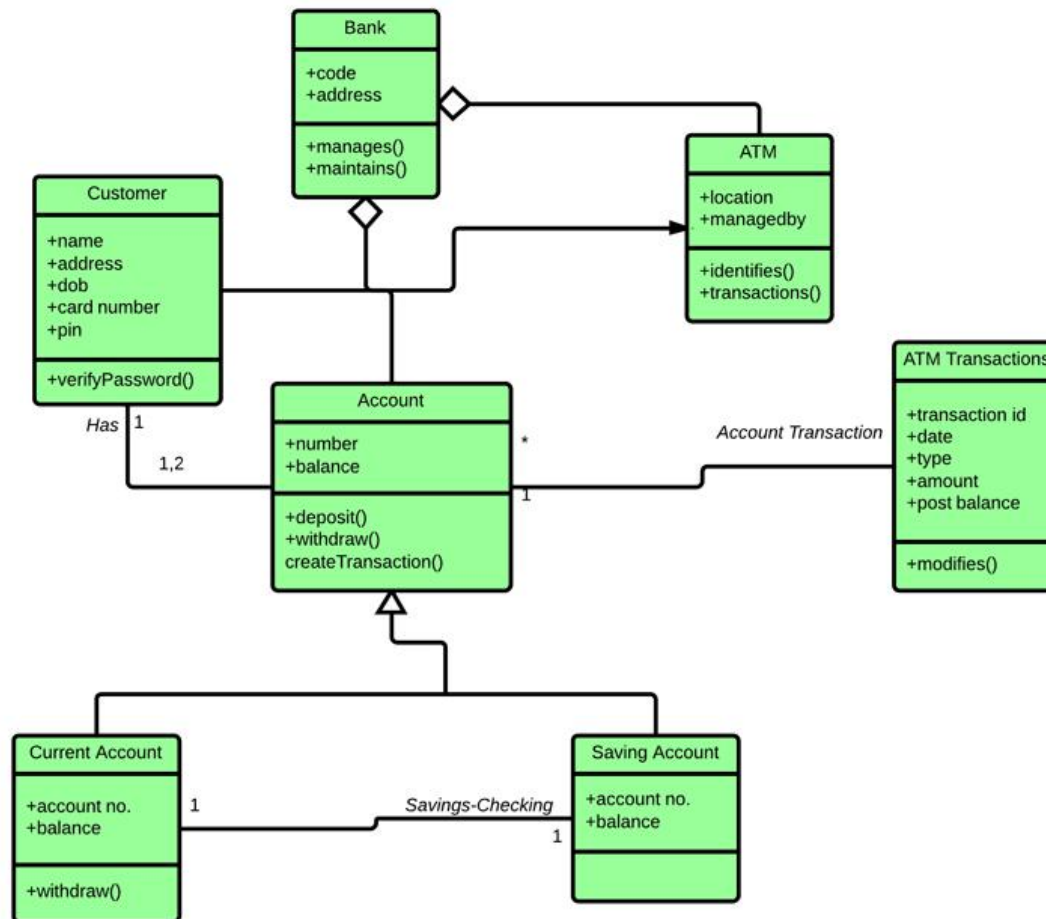
Class diagram for a hotel management system

A class diagram can show the relationships between each object in a hotel management system, including guest information, staff responsibilities, and room occupancy. The example below provides a useful overview of the hotel management system. Get started on a class diagram by clicking the template below.



Class diagram for an ATM system

ATMs are deceptively simple: although customers only need to press a few buttons to receive cash, there are many layers of security that a safe and effective ATM must pass through to prevent fraud and provide value for banking customers. The various human and inanimate parts of an ATM system are illustrated by this easy-to-read diagram—every class has its title, and the attributes are listed beneath.



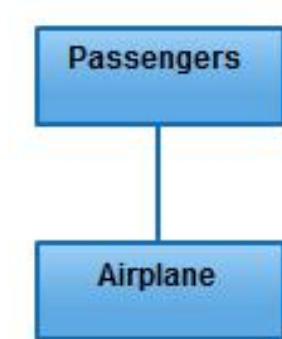
Relationships

Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections. The following are such types of logical connections that are possible in UML:

- Association
- Directed Association
- Reflexive Association
- Multiplicity
- Aggregation
- Composition
- Inheritance/Generalization
- Realization

Association

It is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as :



Directed Association

It refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.



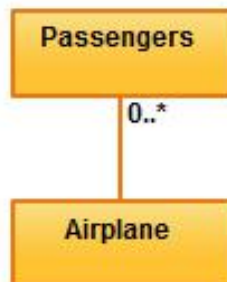
Reflexive Association

This occurs when a class may have multiple functions or responsibilities. For example, 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 managed by the aviation engineer there could be a managed by relationship in two instances of the same class.



Multiplicity

It is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.



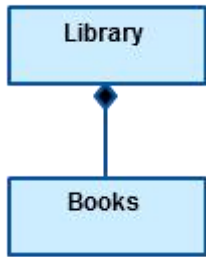
Aggregation

It refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.



Composition



The composition relationship is very similar to the aggregation relationship, with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag's side pocket will also cease to exist once the shoulder bag is destroyed.

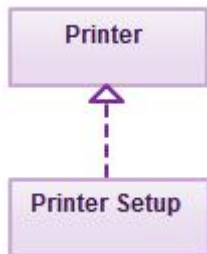
To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

Inheritance / Generalization

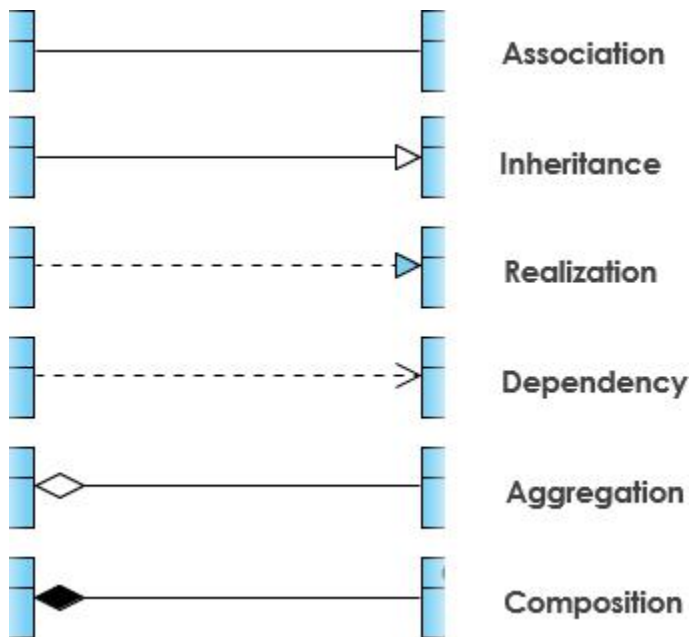


It refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.

Realization



It denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



Other alternative diagrammatic notations:

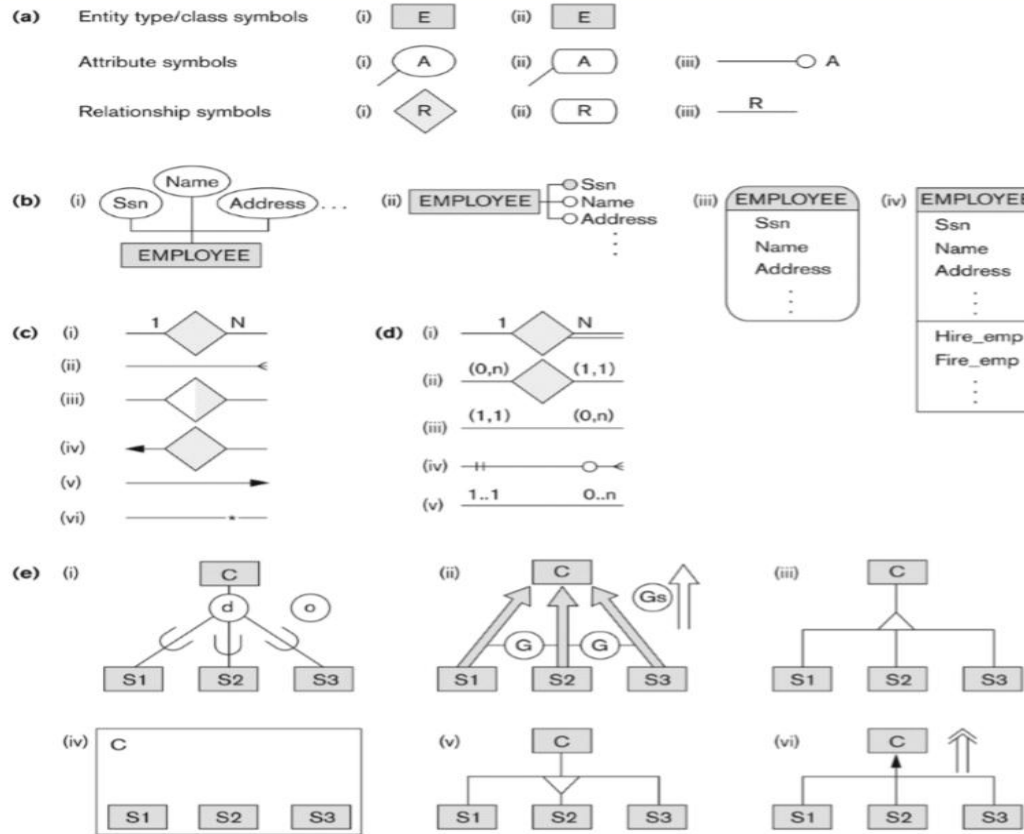


Figure A.1

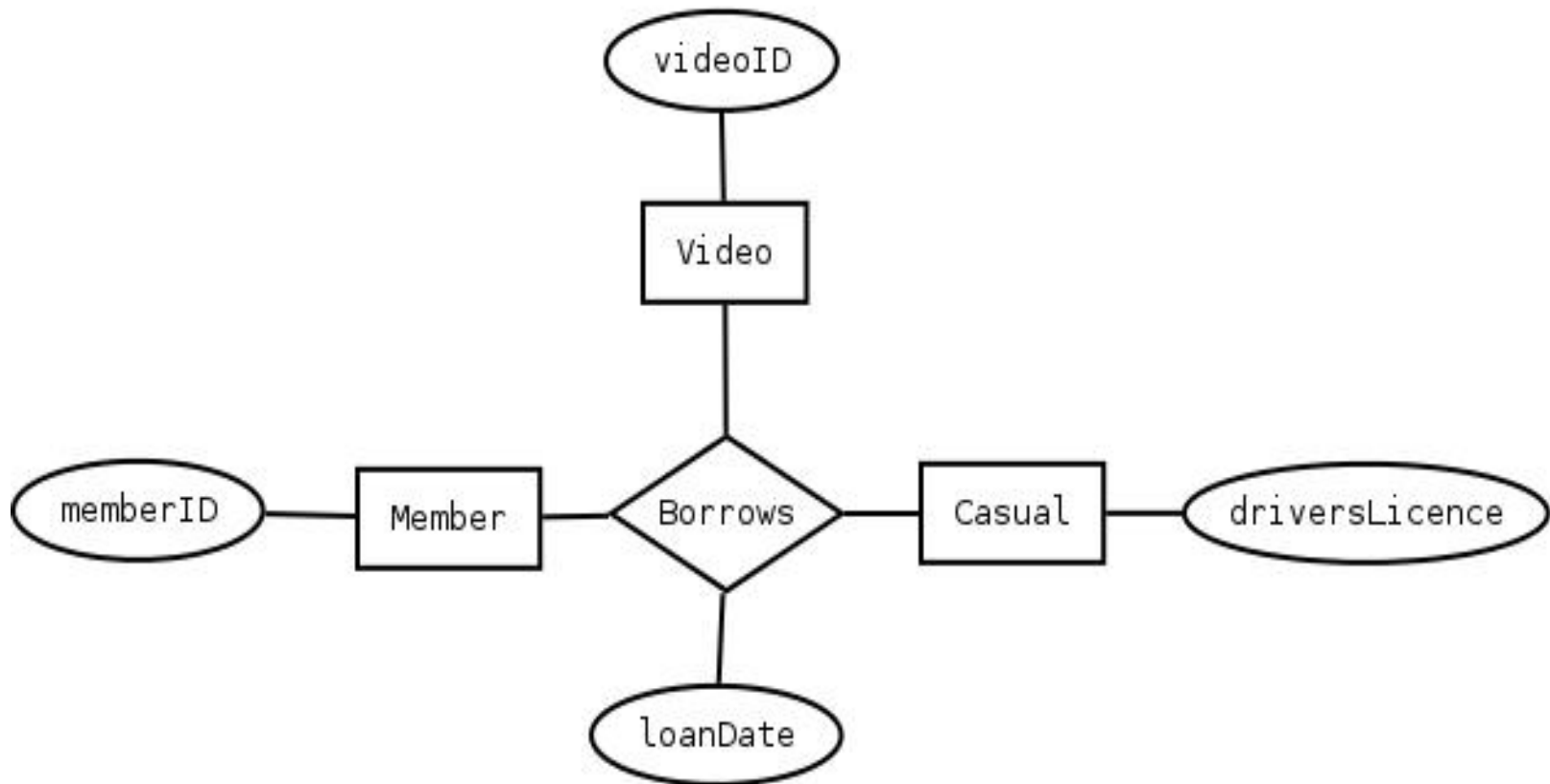
Relationships of Higher Degree

- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are called n -ary
- In general, an n -ary relationship is not equivalent to n binary relationships
- Constraints are harder to specify for higher-degree relationships ($n > 2$) than for binary relationships

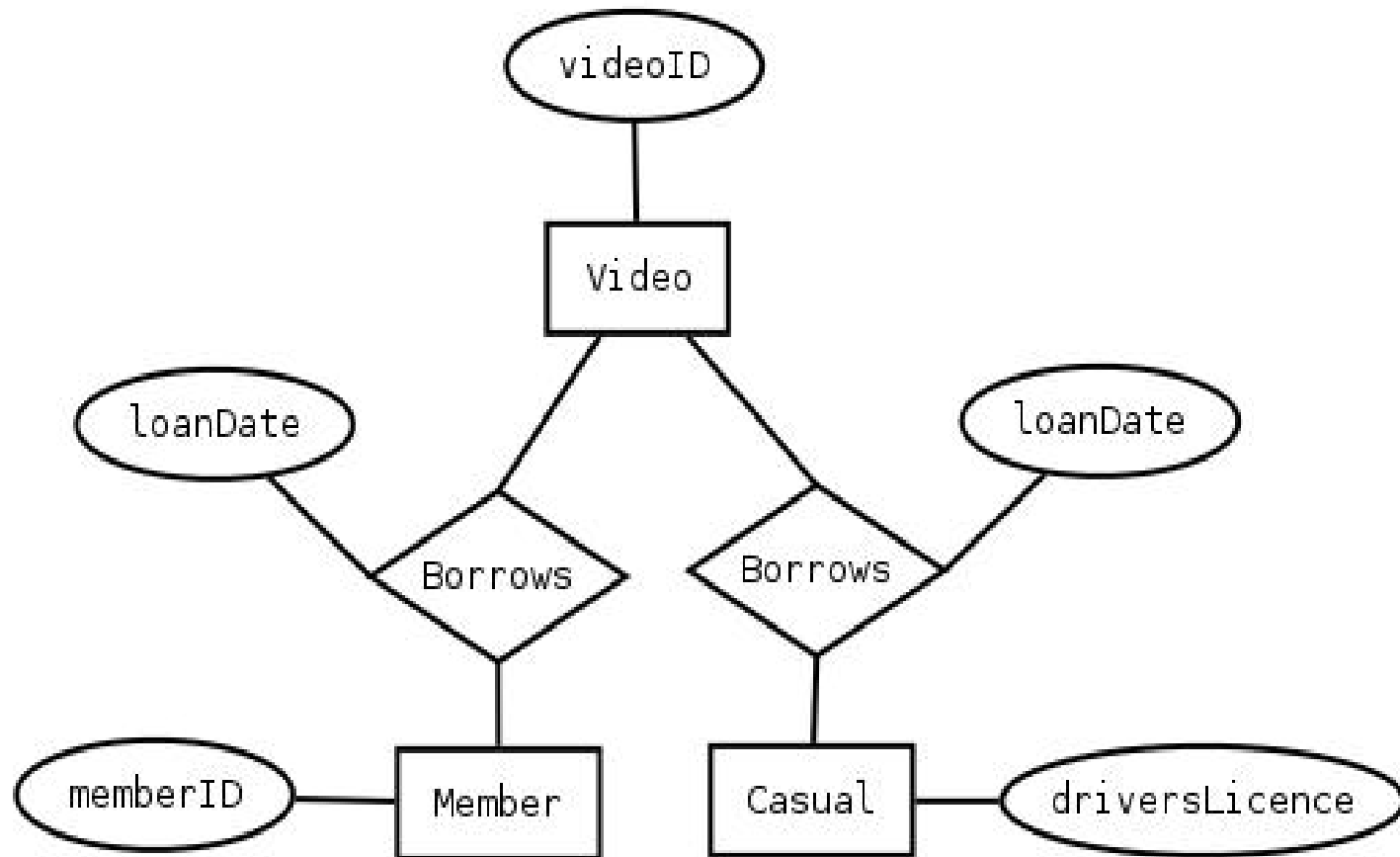
Discussion of n -ary relationships ($n > 2$)

- In general, 3 binary relationships can represent different information than a single ternary relationship
- If needed, the binary and n -ary relationships can all be included in the schema design
- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types)
- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant

One Ternary Relationship



Two Binary Relationship



Use case diagrams

What is a use case diagram?

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

When to apply use case diagrams

A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case.

UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.

UML use case diagrams are ideal for:

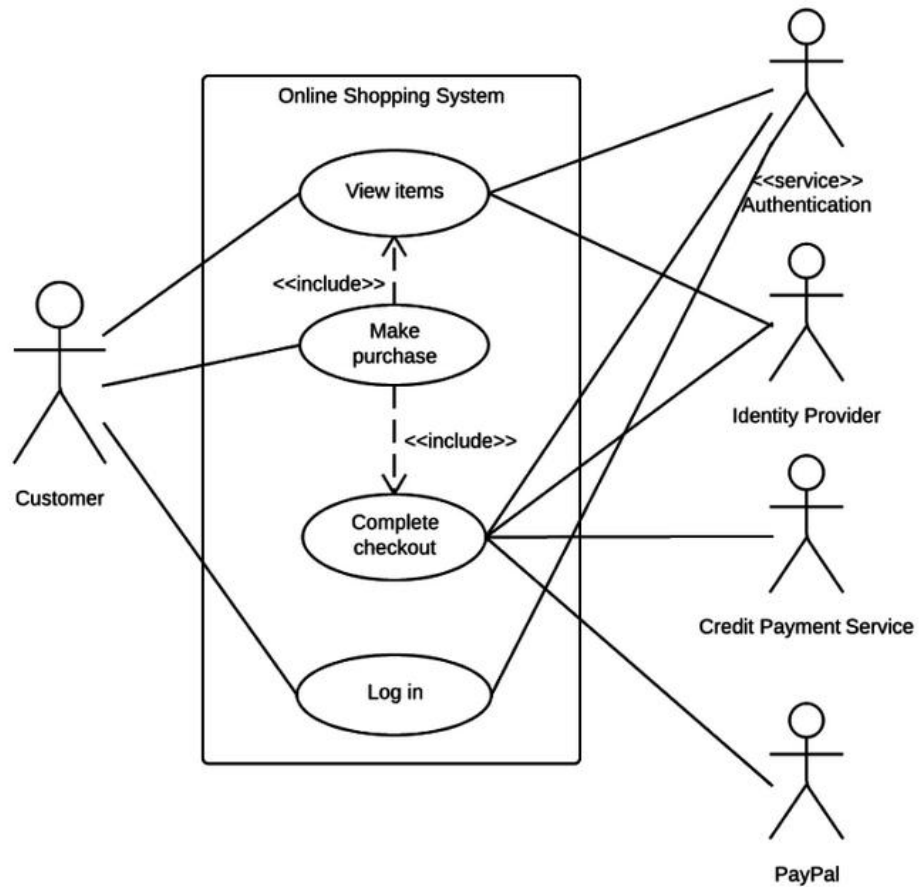
- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

Use case diagram components

Common components include:

- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Online shopping system use case diagram



Enhanced Entity Relationship Model (EER Model)

EER is a high-level data model that incorporates the extensions to the original ER model.

It is a diagrammatic technique for displaying the following concepts


- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation

These concepts are used when they come in EER schema and the resulting schema diagrams are called as EER Diagrams.

Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different entity types.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with  symbol.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.
For example: Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.
For example: Square, Circle, Triangle are the sub class of Shape super class.

B. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.

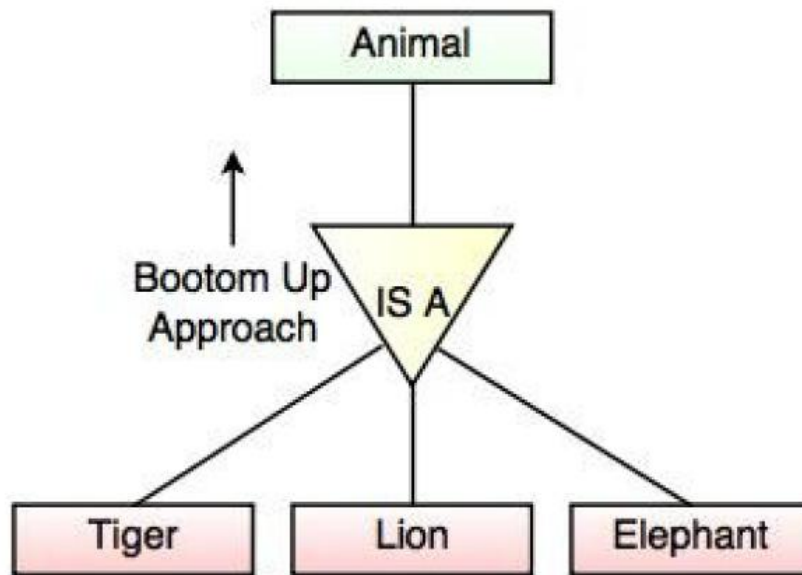
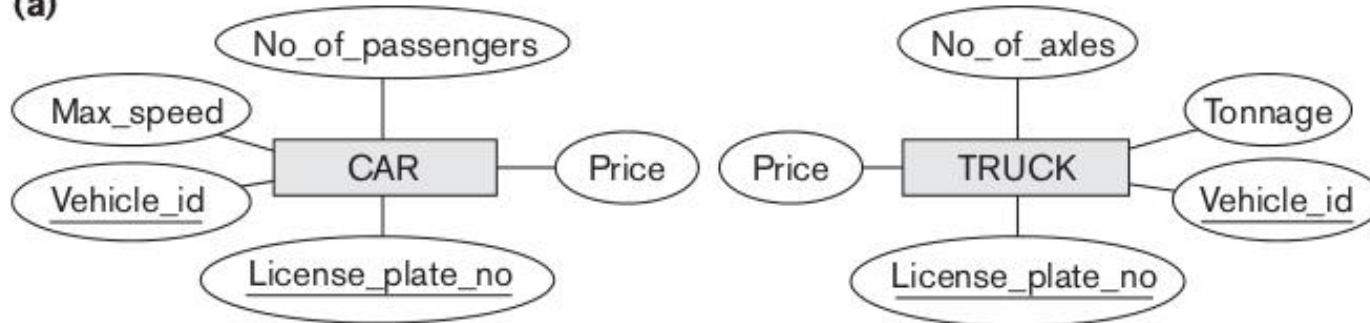


Fig. Generalization

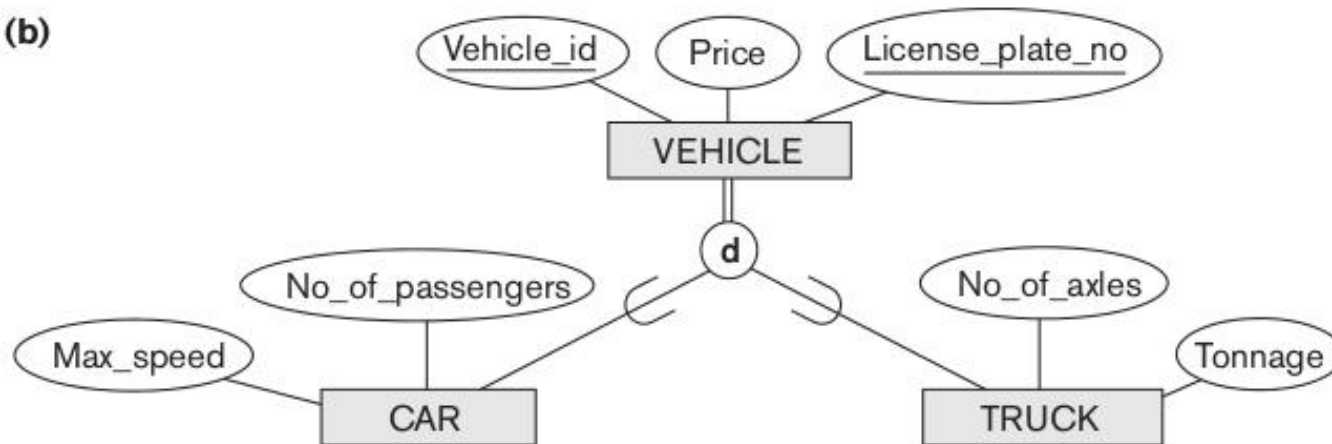
In the above example, Tiger, Lion, Elephant can all be generalized as Animals

Generalization example:

(a)



(b)



2. Specialization

- Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entity.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

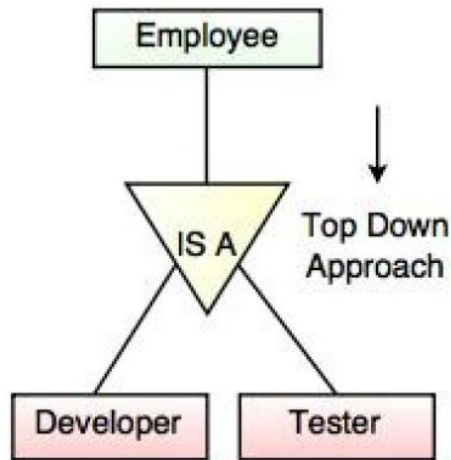


Fig. Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

Types of Specialization

- Predicate-defined (or condition-defined) : based on some predicate. E.g., based on value of an attribute, say, Job-type, or Age.
- Attribute-defined: shows the name of the attribute next to the line drawn from the superclass toward the subclasses (see Fig. 4.1)
- User-defined: membership is defined by the user on an entity by entity basis

Constraints on Specialization and Generalization

Two basic constraints can apply to a specialization/generalization:

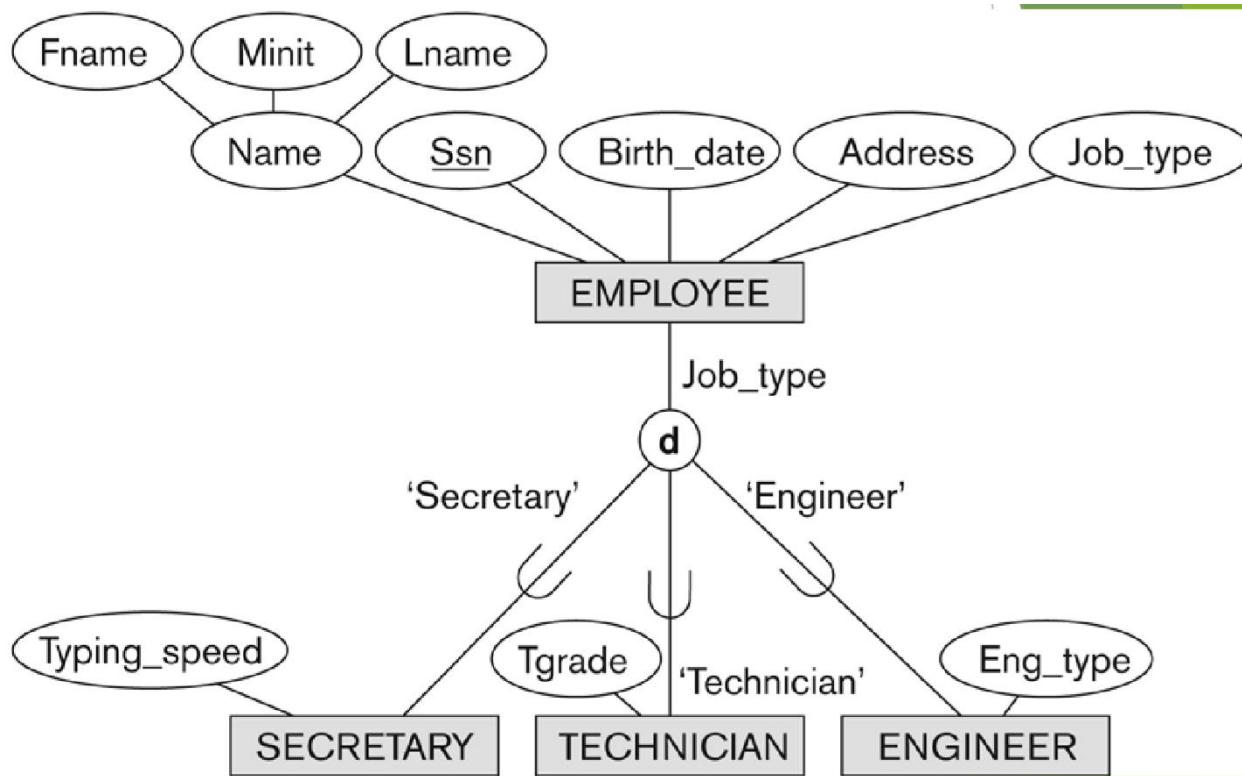
- Disjointness Constraint
- Completeness Constraint

- Disjointness Constraint:
 - Specifies that the subclasses of the specialization must be disjoint:
 - ◆ an entity can be a member of at most one of the subclasses of the specialization
 - Specified by d in EER diagram
 - If not disjoint, specialization is overlapping:
 - ◆ that is the same entity may be a member of more than one subclass of the specialization
 - Specified by o in EER diagram
- Completeness (Exhaustiveness) Constraint:
 - Total specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
 - Shown in EER diagrams by a double line
 - Partial allows an entity not to belong to any of the subclasses
 - Shown in EER diagrams by a single line

We have four types of specialization/generalization:

- Disjoint, total
- Disjoint, partial
- Overlapping, total
- Overlapping, partial

Example of disjoint partial Specialization:



Example of overlapping total Specialization:

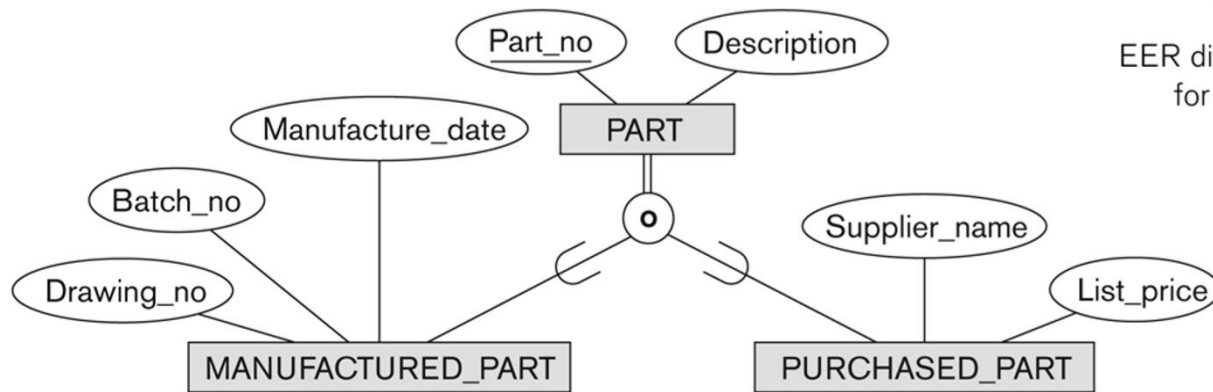
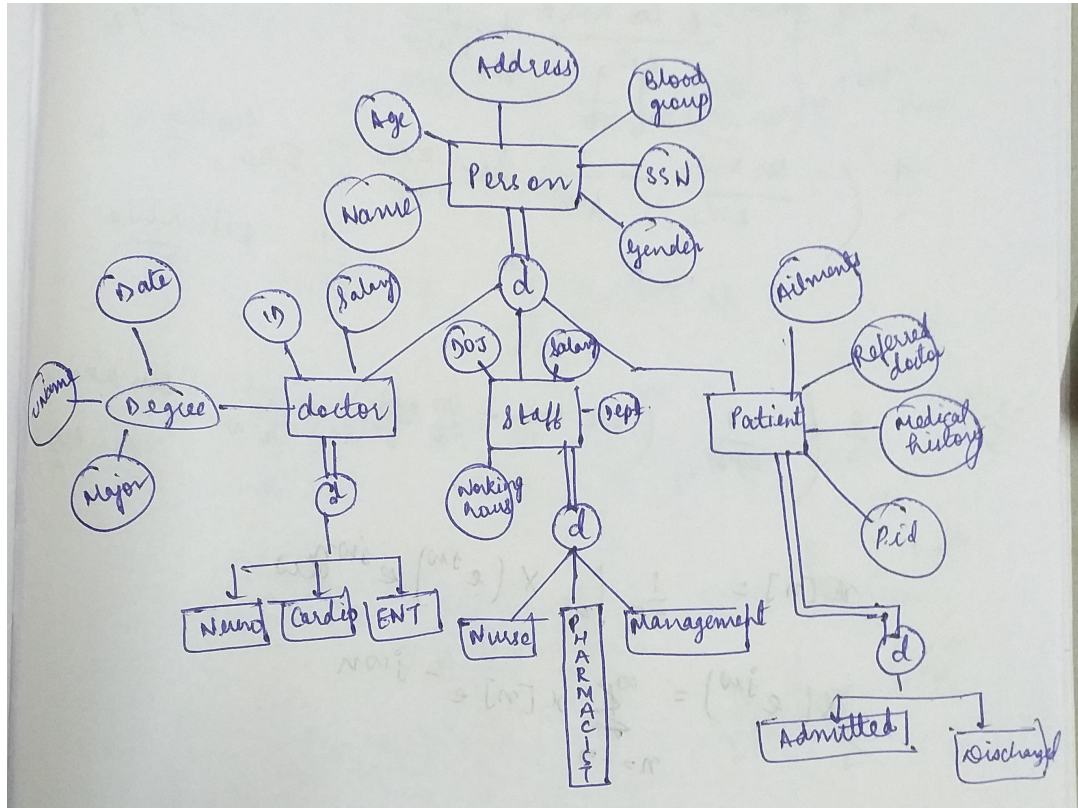


Figure 4.5

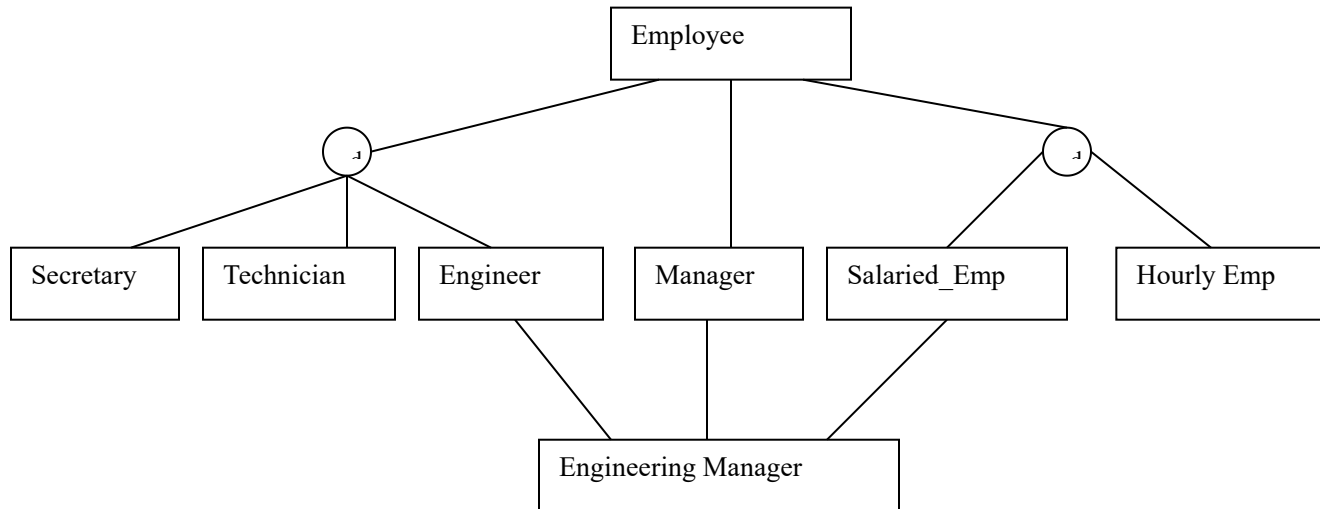
EER diagram notation
for an overlapping
(nondisjoint)
specialization.

The hospital Example:



Specialization and Generalization Lattices and Hierarchies

- A specialization may be a subclass, and also have a subclass specified on it.



- For example, in the figure above, Engineer is a subclass of Employee, but also a super class of Engineering Manager.
- This means that every Engineering Manager, must also be an Engineer.
- **Specialization Hierarchy** – has the constraint that every subclass participates as a subclass in only one class/subclass relationship, i.e. that each subclass has only one parent. This results in a tree structure.
- **Specialization Lattice** – has the constraint that a subclass can be a subclass of more than one class/subclass relationship. The figure shown above is a specialization lattice, because Engineering_Manager participates has more than one parent classes.
- In a lattice or hierarchy, the subclass inherits the attributes not only of the direct superclass, but also all of the predecessor super classes all the way to the root.
- A subclass with more than one super class is called a shared subclass. This leads to multiple inheritance, where a subclass inherits attributes from multiple classes.
- In a lattice, when a superclass inherits attributes from more than one superclass, and some attributes are inherited more than once via different paths (i.e. Engineer, Manager and Salaried Employee all inherit from Employee, that are then inherited by Engineering Manager).
- In this situation, the attributes are included only once in the subclass.

C. Category or Union

- Category represents a single super class or sub class relationship with more than one super class.
- It can be a total or partial participation.

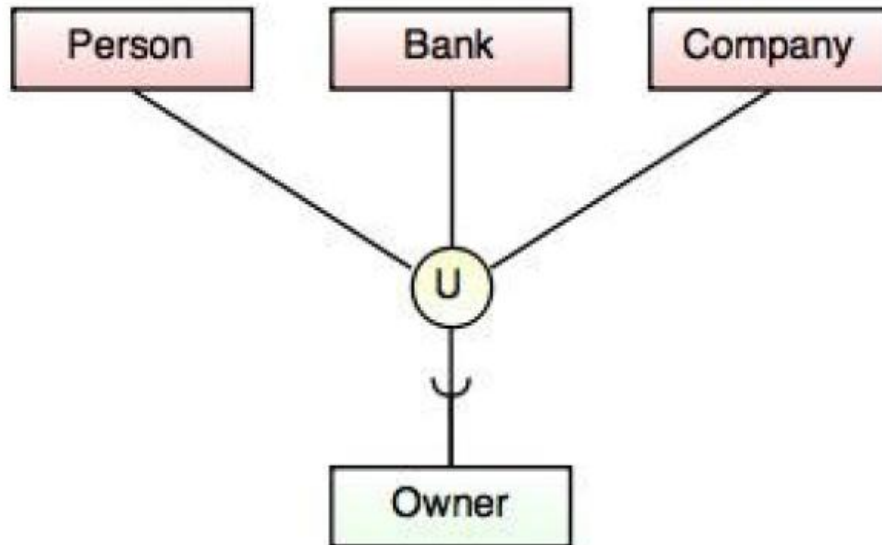


Fig. Categories (Union Type)

D. Aggregation

- Aggregation is a process that represent a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entity is treated as a single entity.

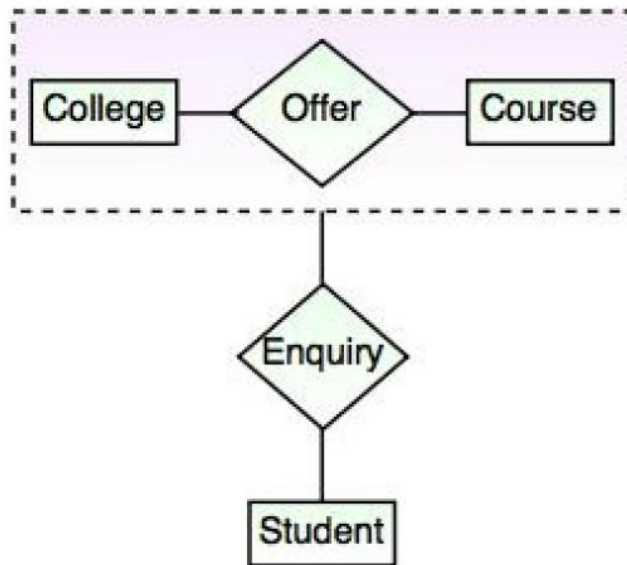


Fig. Aggregation

In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

- References -

Slides

Internet:

www.tutorialspoint.com

www.lucidchart.com

www.tutorialride.com

www.creately.com

www.visual-paradigm.com