MOHSIN IBNA HOSSAIN

AIUB, OOAD NOTES

OBJECT ORIENTED ANALLYSIS & DESIGN

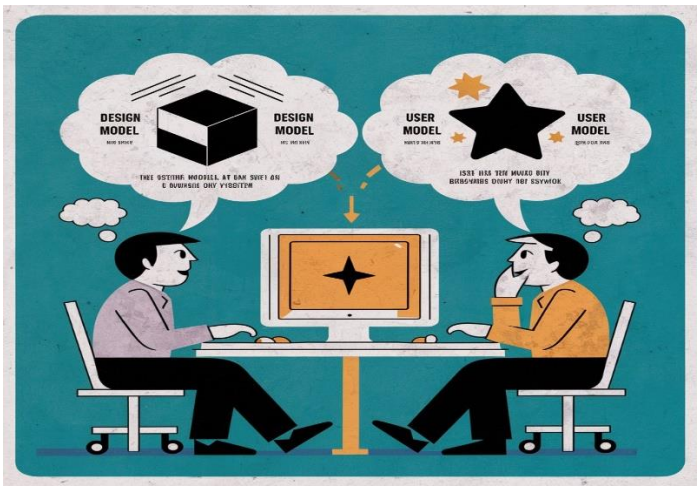
Table of Contents

Why do we model things?

Ch:01

WHY WE MODEL?

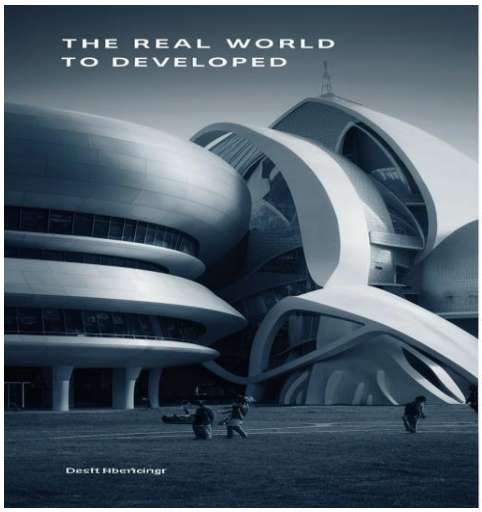
- A model is a simplification of reality
- A model provides the blueprints of a system.

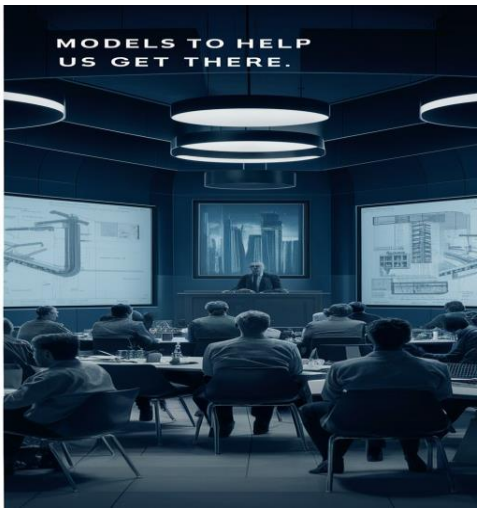


- The larger and more complex a system, the more crucial modeling becomes.

➢ We create models of complex systems because we can't understand them completely.

- Human understanding of complexity has limits, and modeling helps by allowing us to focus on one aspect at a time.





The Importance of Modeling in Software Development

- Modeling means **designing software applications before coding**.
- Essential for large software projects.
- Helpful for medium and small projects too.
- Proven and widely accepted engineering technique.
- Unsuccessful projects fail in different ways.
- Successful projects often share similarities.
- One common factor in successful projects is the use of modeling.
- Modeling helps us better understand the system we are developing.

Benefits of model

- Through modelling we achieve four aims:
  1. Models help us to **visualize a system** as it is or as we want it to be.
  2. Models permit us to specify the **structure or behaviour** of a system.
  3. Models give us **a template that guides us in constructing** a system.
  4. Models **document the decisions** we have made.

Unified Modeling Language (UML)

- **Unified Modeling Language (UML)** is a general-purpose modeling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. UML is **not a programming language**, it is rather **a visual language.**

What is UML?

- Unified Modeling Language (UML) is a standardized visual modeling language used in software engineering.
- Provides a general-purpose, developmental, and intuitive way to visualize system design.
- Helps **specify, visualize, construct, and document** software system artifacts.
- Uses UML diagrams to portray system **behavior** and **structure**.
- Assists software engineers, businessmen, and system architects with modeling, design, and analysis.
- Adopted as a standard by the Object Management Group (OMG) in 1997.
- Managed by OMG since adoption.
- Approved as a standard by the International Organization for Standardization (ISO) in 2005.

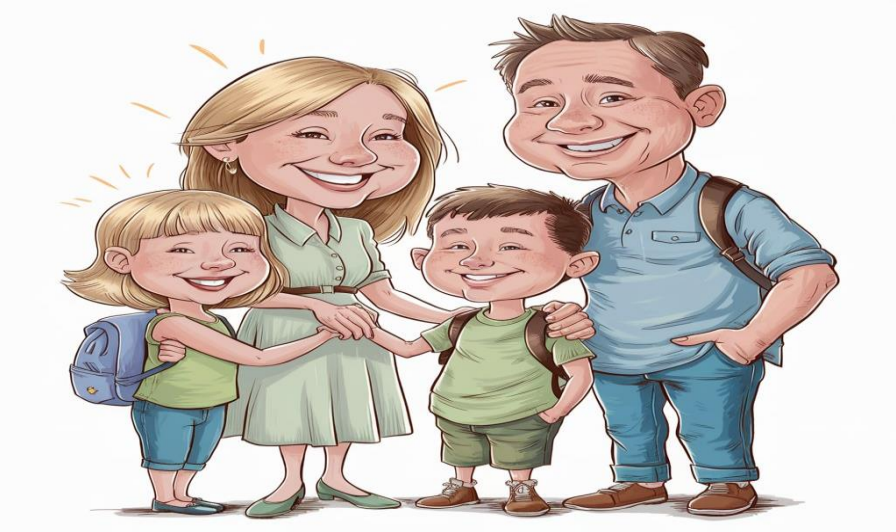
Why do we need UML?

- Complex applications need collaboration and planning from multiple teams.
- UML provides a clear way to communicate among teams.
- Non-programmers do not understand code; UML helps communicate with non-programmers.
- UML explains essential requirements, functionalities, and processes.
- Visualizing processes, user interactions, and system structure saves time.

Building blocks of UML

- The vocabulary of the UML includes three kinds of building blocks:

1. **Things** - abstractions that are primary concern in a model (John, Jesmin, Josheph, Jessy)
2. **Relationships** - tie these things together (Father, Mother, Daughter, Son)
3. **Diagrams** - group collections of things (Happy Family)



Things in UML

- **Things** are the most important building blocks of UML. Things can be

1. Structural Things
2. Behavioral Things
3. Grouping Things
4. Annotation Things

❑ Structural things

➢ **Structural things** define the static part of the model. They represent the physical and conceptual elements. There are **seven** kinds of structural things.

1. Class:

➔ Class represents a **set of objects** having **similar responsibilities**.

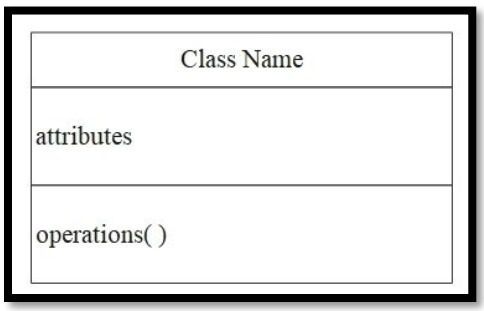


Fig:1

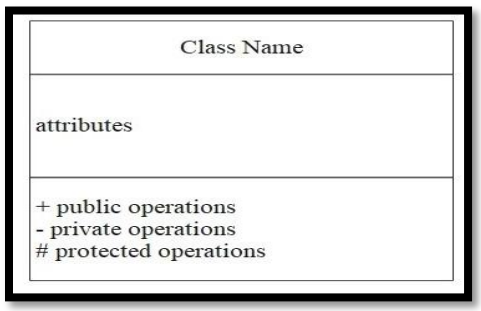


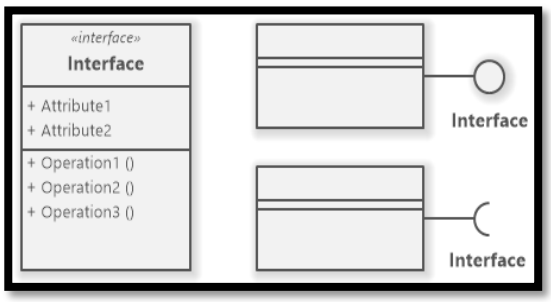
Fig:2

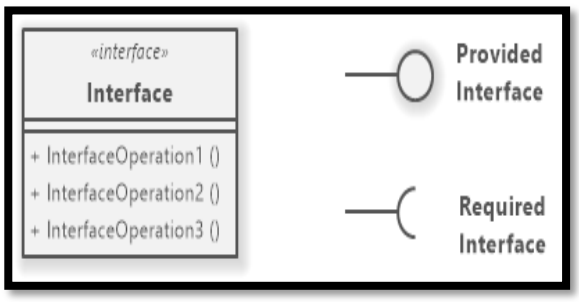
Visibility symbols are used to determine the accessibility of the information contained in classes.

Note: The "+" represents public operations, vice versa, "-" represents private operations. Plus, "#" is for the protected operations.

2. Interface:

➔ Interface defines a **set of operations**, which **specify the responsibility** of a class. An interface cannot be instantiated. UML interface can be depicted in the same way as the UML class or using "lollipop" notation.

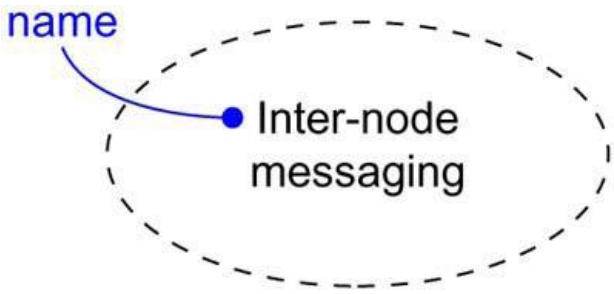




3. Collaboration:

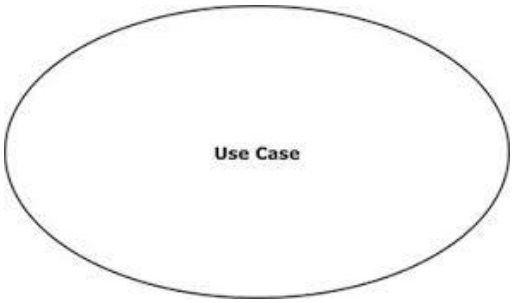
➔ **Collaboration** defines an **interaction between elements**.

- A collaboration is a society of classes, interfaces, and other elements that work together to provide some functionality within the system.
- Graphically, a collaboration is rendered as an ellipse with dashed lines.
- collaboration name must be unique within its enclosing package



4. Use case:

➔ Use case represents a **set of actions** performed by a system for a specific goal.



5. Active class:

➔ Active class looks similar **to a class with double lines** on the left and right or **a solid border**. Active class is generally used to describe the **concurrent behavior** of a system.

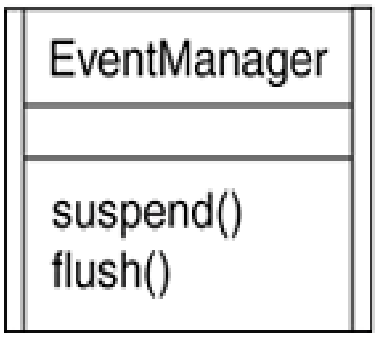


Fig: 1

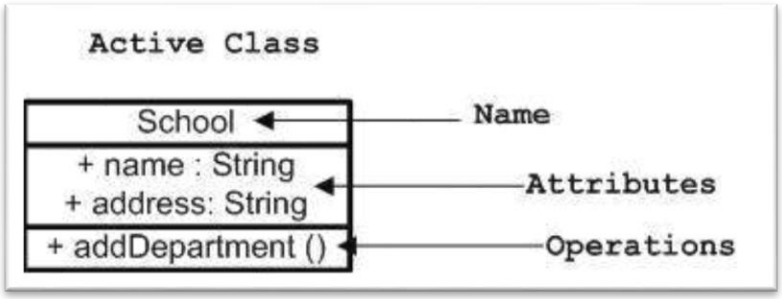
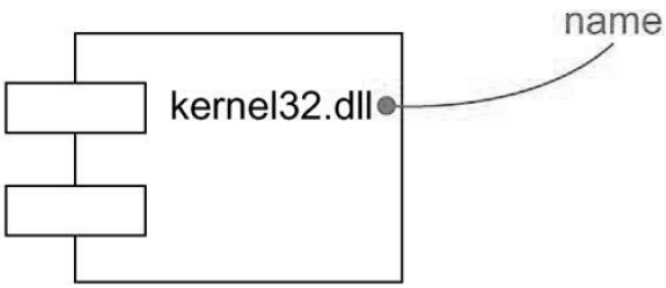


Fig: 2

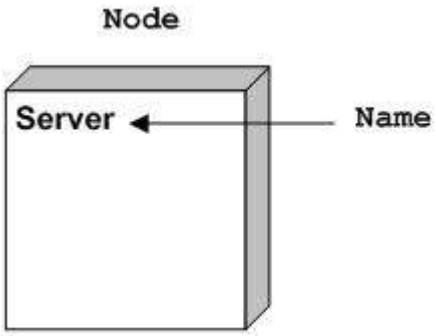
6. Component:

➔ Component describes the **physical part** of a system.



7. Node:

➔ A node can be defined as a physical element **that exists at run time.**



❑ Behavioral Things

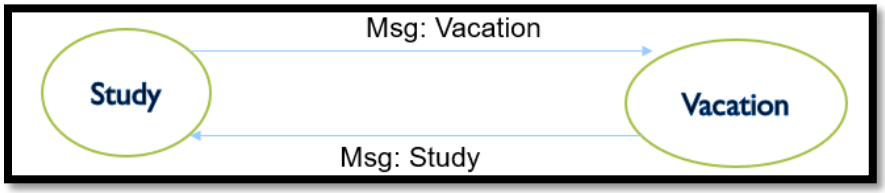
➔**A behavioral thing** consists of the **dynamic parts** of UML models. There are **two** kinds of behavioral things.

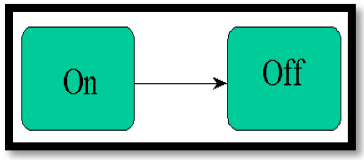
1. Interaction:

➔ Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

1. State machine:

➔ Specifies the sequence of states that an object or an interaction goes through during its lifetime in response to events.





❑ Grouping Things

➢ It is a method that together **binds the elements** of the UML model. In UML, the **package is the only thing**, which is used for grouping.

- **Package:** Package is the only thing that is available for grouping behavioral and structural things.

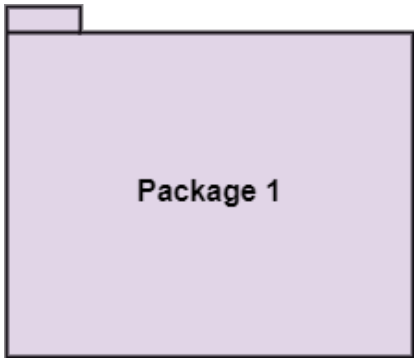


Fig:1

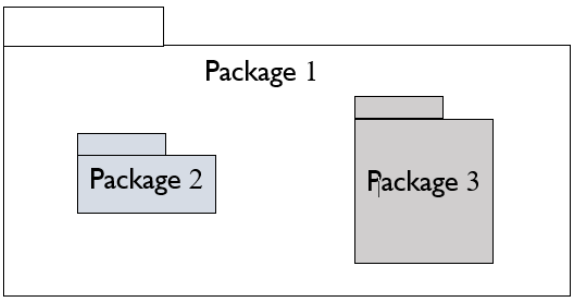


Fig:2

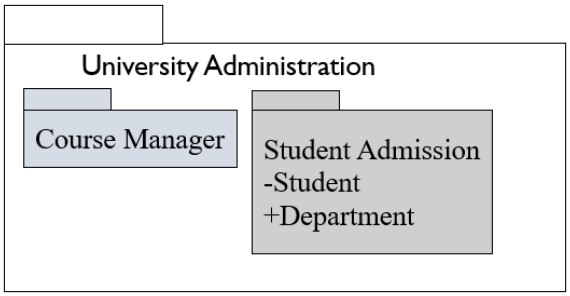


Fig:3

❑ Annotation Things

➢ It is a mechanism that captures the **remarks, descriptions, and comments** of UML model elements. In UML, a **note is the only Annotational thing**.

- **Note:** It is used to attach the constraints, comments, and rules to the elements of the model. It is a kind of yellow sticky note.

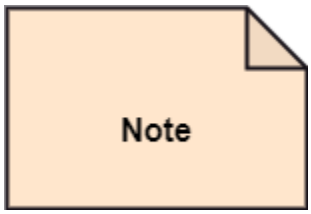


Fig:1

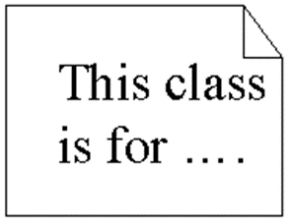


Fig:2

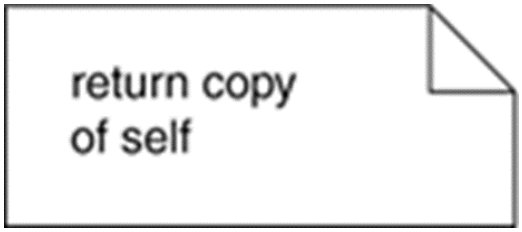


Fig:3

Relationships in UML

➔There are **four** kinds of relationships in the UML

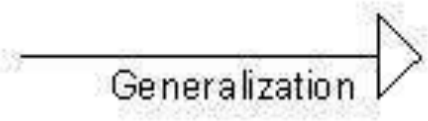
- Association
- Generalization
- Realization
- Dependency

1. Association:

- Association is basically a **set of links** that connects the elements of a UML model. It also describes **how many objects** are taking part in that relationship.

2. Generalization:

- Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the **inheritance relationship** in the world of objects.



Example:

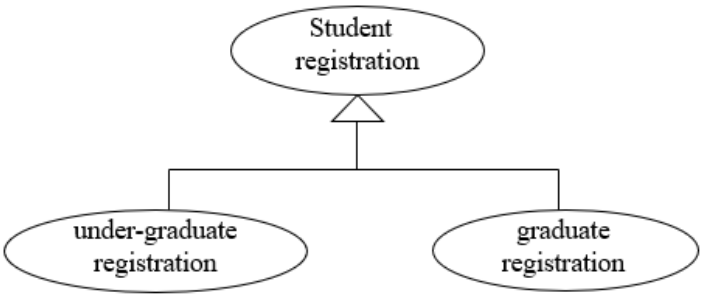


Fig:1

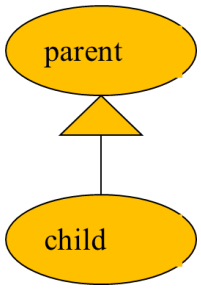
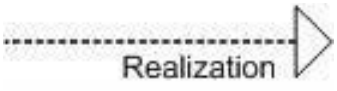


Fig:2

3. Realization:

- Realization can be defined as a relationship in which **two elements are connected**. One element describes **some responsibility, which is not implemented and the other one implements them**. This relationship exists in case of interfaces.



Example:

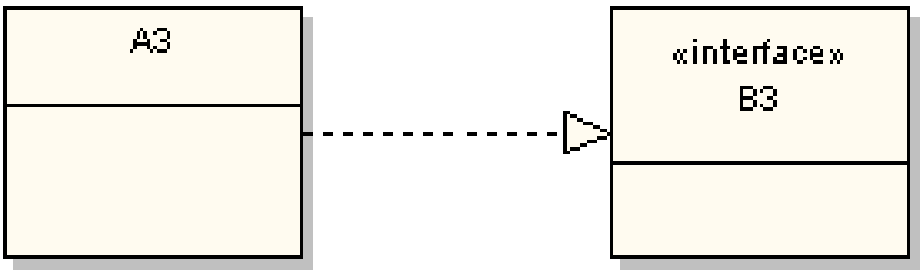
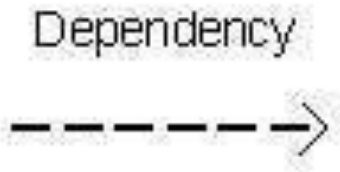


Fig: Class A3 implements B3

4. Dependency:

- Dependency is a relationship between **two things in which change in one element also affects the other**.



Example:

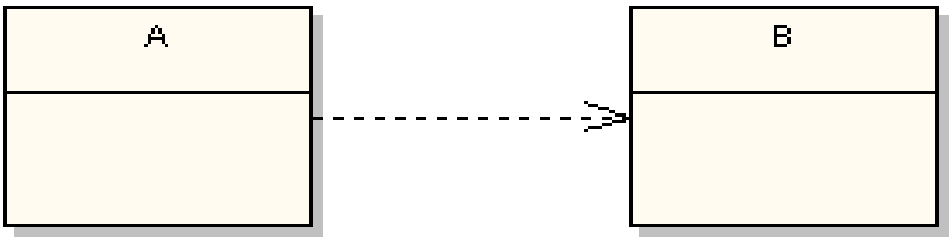


Fig: Class A depends on class B

Diagrams in UML

- UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

- The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

- UML includes the following **nine** diagrams

Structural Diagrams

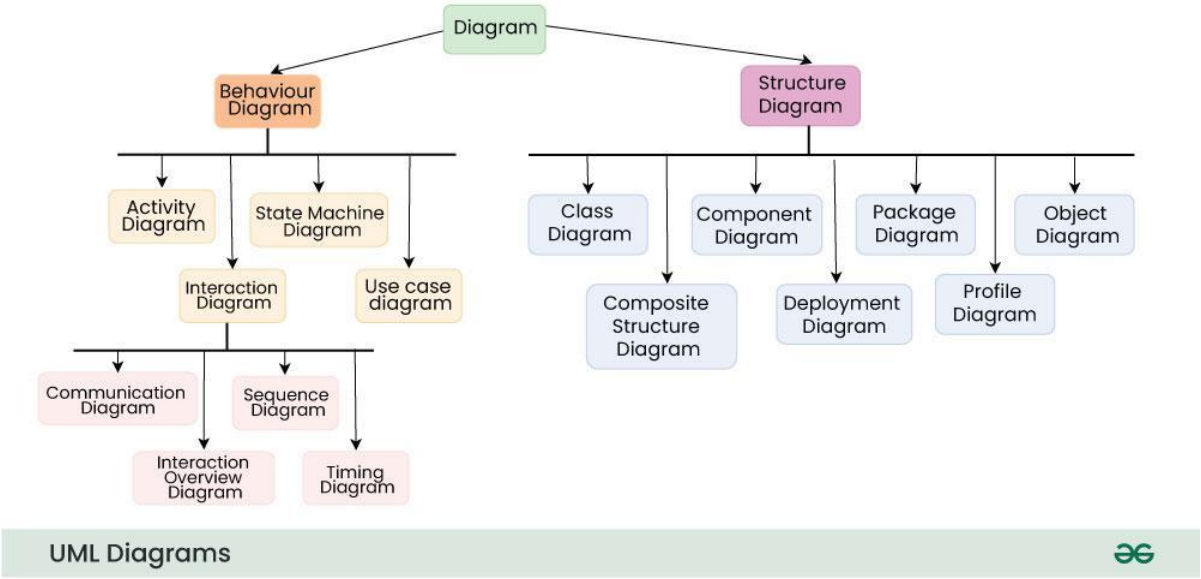
Represent the static aspects of a system.

- Class Diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

Behavioral Diagrams

Represent the dynamic aspects of a system.

- Use case Diagram
- Sequence Diagram (Interaction)
- Collaboration Diagram
- Statechart Diagram
- Activity Diagram



UML Diagrams

❑ Class diagram

- Shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams address the **static design view** of a system. Class diagram that includes active classes address the **static process view** of a system.

❑ Object diagram

- Shows a set of objects and their relationships. Object diagrams represent static snapshots on instances of the things found in class diagrams. These designs address the **static design or process view** of a system from the perspective of real or prototypical cases.

❑ Component diagram

- Shows an encapsulated class and its interfaces, ports, and internal structure consisting of nested components and connectors. Component diagrams address the **static design implementation view** of a system.

❑ Deployment diagram

- Shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the **static deployment view** of an architecture (networking). A node typically hosts one or more artifacts.

❑ Use case diagram

- Shows a set of use cases and actors and their relationships. Use case diagrams address the **static use case view** of a system.

❑ Interaction diagram

- Shows an interaction, consisting of a set of objects or roles, including the messages that may be dispatched among them. Interaction diagrams address the **dynamic view** of a system. Both **sequence diagrams** and **communication diagrams** are kinds of interaction diagrams.

  A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages.

  A communication diagram is an interaction diagram that emphasizes the structural organization of the objects or roles that send and receive messages.

❑ State diagram

- Shows a state machine, consisting of states, transitions, events, and activities. A state diagrams shows the **dynamic view** of an object.

❑ Activity diagram

- Shows the structure of a process or other computation as the flow of control and data from step to step within the computation. Activity diagrams address the **dynamic view** of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

❑ Deployment diagram

- Shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the **static deployment view** of an architecture (networking). A node typically hosts one or more artifacts.

❑ Artifact diagram

- Shows the physical constituents of a system on the computer. Artifacts include files, databases, and similar physical collections of bits. Artifacts are often used in conjunction with deployment diagrams.

❑ Package diagram

- Shows the decomposition of the model itself into organization units and their dependencies (package contains a set of diagrams and their dependency grouping).

❑ Timing diagram

- Is an interaction diagram that shows actual times across different objects or roles, as opposed to just relative sequences of messages?

❑ Interaction overview diagram

- Is a hybrid of an activity diagram and a sequence diagram.

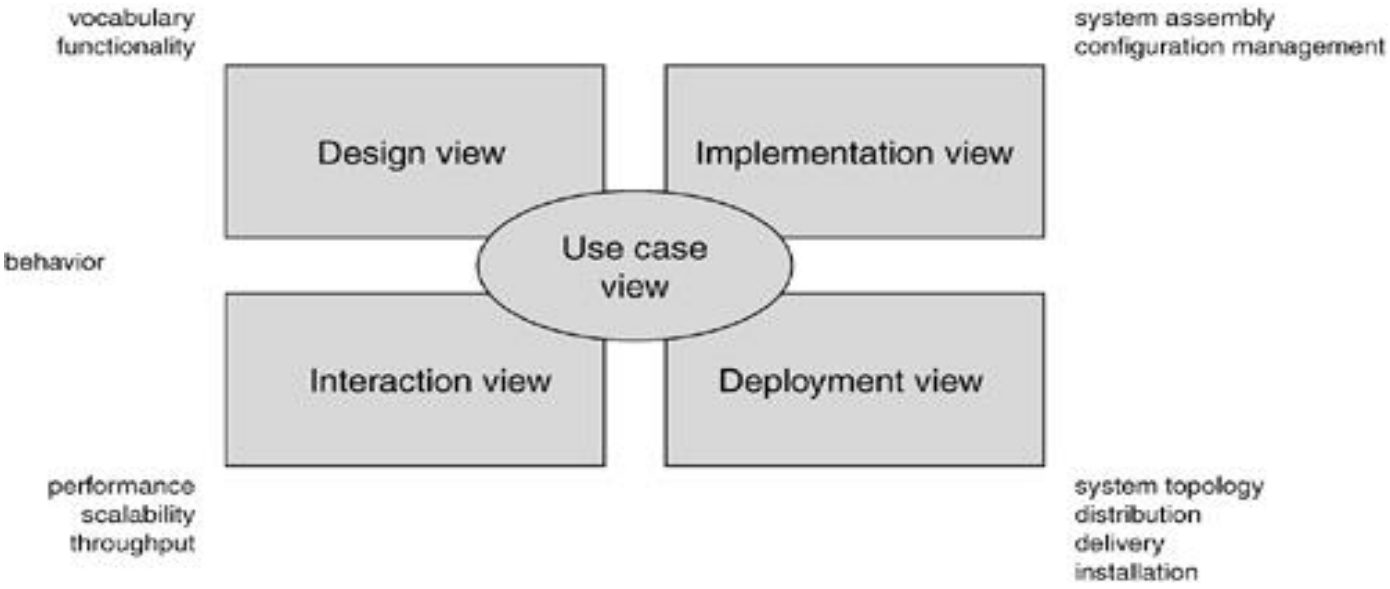
Architecture of OO Software Systemin UML

- To understand the architecture of object-oriented (OO) software system, one needs several complementary and interlocking views:

1. A use case view (exposing the requirements of the system),
2. A design view (capturing the vocabulary of the problem space and the solution space),
3. A process view (modelling the distribution of the system's processes and threads),
4. An implementation view (addressing the physical realization of the system),
5. A deployment view (focusing on system engineering view).

- Each of these views may have structural, as well as behavioural, aspects. Together these views represent the blueprints of Object-Oriented Software System.

System architecture modeling



UML Architecture

- Visualizing, specifying, constructing and documenting a software intensive system demands that the system be viewed from a number of perspectives (e.g., ATM system development design in several diagram drawing perspective).
- Different stakeholders- end users, analysts, developers, system integrators, testers, technical writers and project managers- each bring different agenda to a project, and each looks at that system in different ways at different times over the project's life.

1. The **use case view** of a system encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. With the UML, the static aspects of this view are captured in use case diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

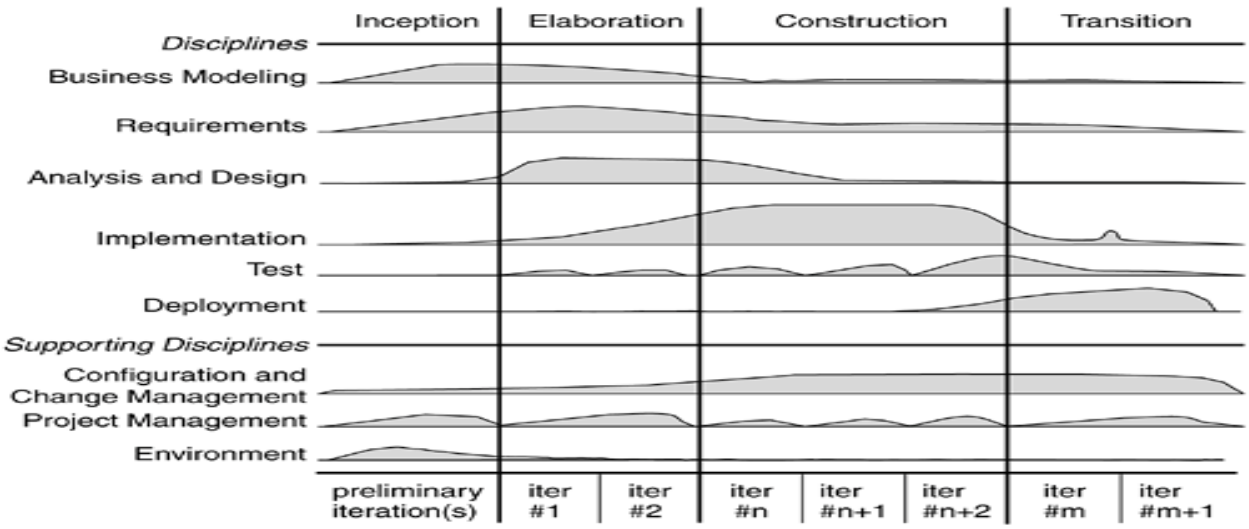
2. The **design view** of a system encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users. With the UML, the static aspects of this view are captured in class diagrams and object diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

3. The **interaction view** of a system shows the flow of control among its various parts, including possible concurrency and synchronization mechanisms. This view primarily addresses the performance, scalability, and throughput of the system. With the UML, the static and dynamic aspects of this view are captured in the same kinds of diagrams as for the design view, but with a focus on the active classes that control the system and the messages that flow between them.

4. The **implementation view** of a system encompasses the artifacts that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system's releases, made up of somewhat independent files that can be assembled in various ways to produce a running system. It is also concerned with the mapping from logical classes and components to physical artifacts. With the UML, the static aspects of this view are captured in artifact diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

5. The **deployment view** of a system encompasses the nodes that form the system's hardware topology on which the system executes. This view primarily addresses the distribution, delivery, and installation of the parts that make up the physical system. With the UML, the static aspects of this view are captured in deployment diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

SDLC – Rational Unified Model (RUP)



- **Inception** is the first phase of the process, when the seed idea for the development is brought up to the point of being at least internally sufficiently well-founded to warrant entering into the elaboration phase.
- **Elaboration** is the second phase of the process, when the product requirements and architecture are defined. In this phase, the requirements are articulated, prioritized, and baselined. A system's requirements may range from general vision statements to precise evaluation criteria, each specifying particular functional or nonfunctional behavior and each providing a basis for testing.
- **Construction** is the third phase of the process, when the software is brought from an executable architectural baseline to being ready to be transitioned to the user community. Here also, the system's requirements and especially its evaluation criteria are constantly reexamined against the business needs of the project, and resources are allocated as appropriate to actively attack risks to the project.
- **Transition** is the fourth phase of the process, when the software is delivered to the user community. Rarely does the software development process end here, for even during this phase, the system is continuously improved, bugs are eliminated, and features that didn't make an earlier release are added.
- **Iteration** is a distinct set of work tasks, with a baselined plan and evaluation criteria that results in an executable system that can be run, tested, and evaluated.

Use Case Diagram

Ch: 02

Introduction

- Use-cases are descriptions of the functionality of a system from a user perspective.

- Depict the behavior of the system, as it appears to an outside user.
- Describe the functionality and users (actors) of the system.
- Show the relationships between the actors that use the system, the use cases (functionality) they use, and the relationship between different use cases.
- Document the scope of the system.
- Illustrate the developer's understanding of the user's requirements.
- Use case diagrams contain **use cases**, **actors**, and their **relationships**.

What is a Use Case Diagram in UML?

- A use case diagram is used to represent the **dynamic behavior** of a system. It encapsulates the system's functionality by incorporating **use cases, actors, and their relationships**. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of 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:

1. It gathers the system's needs.
2. It depicts the **external view** of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Use Case Diagram Notations

- UML notations provide a visual language that enables software developers, designers, and other stakeholders to communicate and document system designs, architectures, and behaviors in a consistent and understandable manner.

1. USE Case:

- Use cases specify **desired behavior**
- The names of use cases are always written in the **form of a verb followed by an object**.
- Each sequence **represents an interaction of actors with the system**
- Each Actor must be **linked to a use case**, while **some use cases may not be linked to actors.**

2. Actor:

- An actor represents a set of roles that users of use case play when interacting with these use cases.
- Actors can be human or automated systems.
- Actors are entities
  - which require help from the system to perform their task, or
  - are needed to execute the system's functions.
- Actors are not part of the system.
- **A system can be an Actor of other systems**

3. Communication Link:

- 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.

Example:

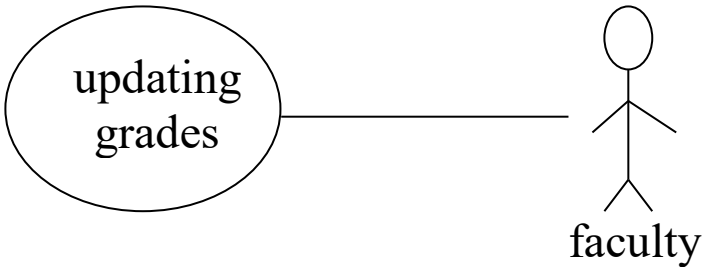


Fig: 1

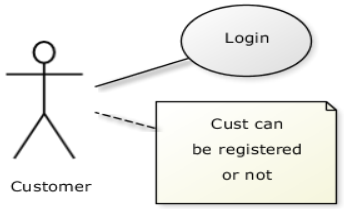
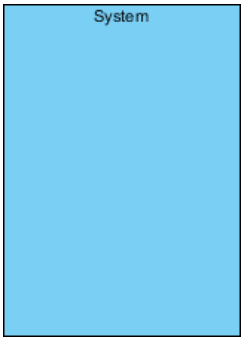


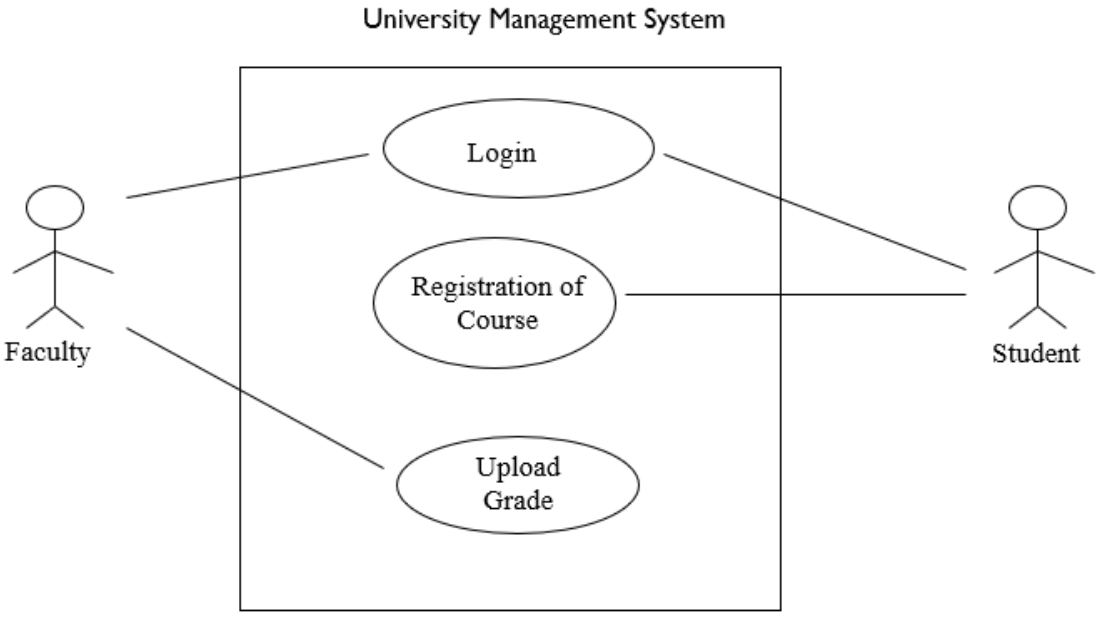
Fig: 2

4. Boundary of system:

- It is shown as a rectangle.
- **It helps to identify what is external versus internal, and what the responsibilities of the system are**.
- The external environment is represented only by actors.



Example of Use cases diagram



Relationships between Use cases

1. Include:

- When a use case uses the functionality of another use case.
- **Includes functionality of another use case** in the business process flow.
- Indicates the base use case will include behavior of the child use case.
- **They are shown as a dotted line with an open arrow and the key word "<< include>>"**
- Arrowhead points to the child use case; base of the arrow connects to the parent use case

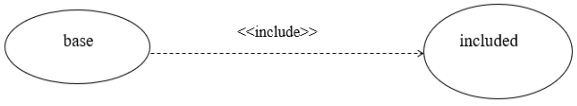


Fig: 1

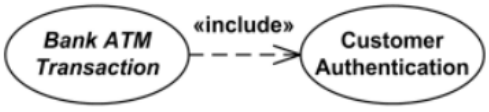


Fig: 2

2. Extends:

- Indicates that an **"Invalid Password"** use case may include (subject to specified in the extension) the behavior specified by base use case **"Login Account"**.
- 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**

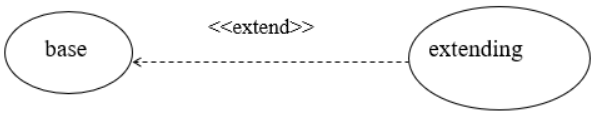


Fig: 1

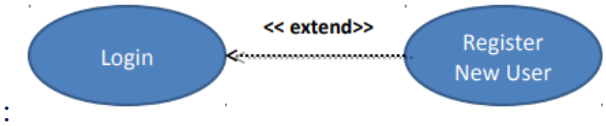


Fig: 2

3. Generalization:

- Generalization is a relationship between a general use case and a more specific use case that inherits and extends features to it
- use cases that are specialized versions of other use cases
- **It is shown as a solid line with a hollow arrow point**

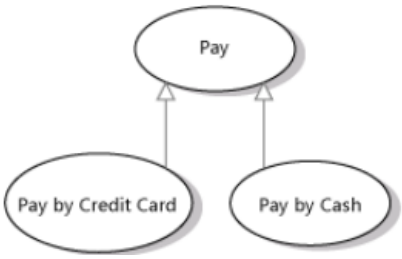
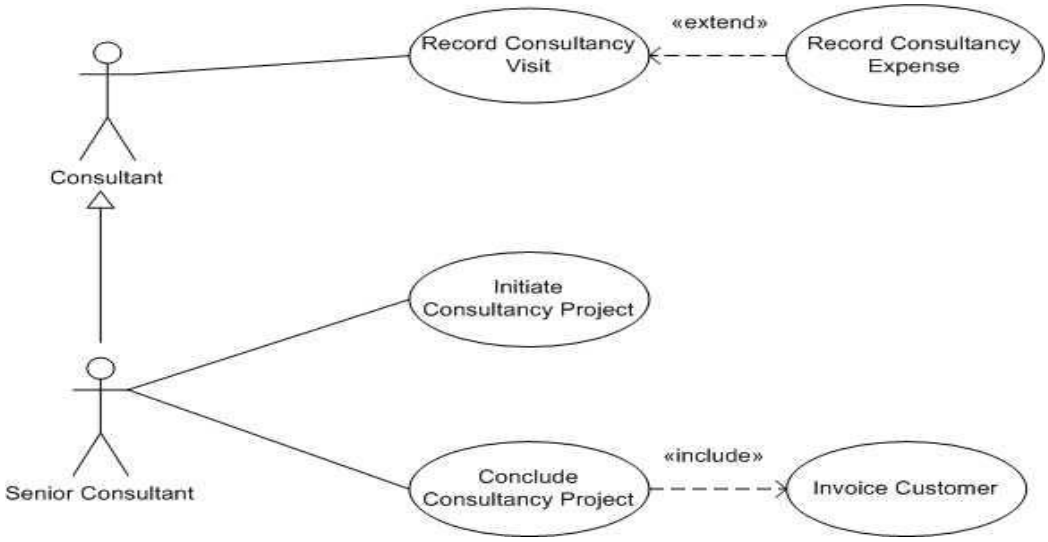
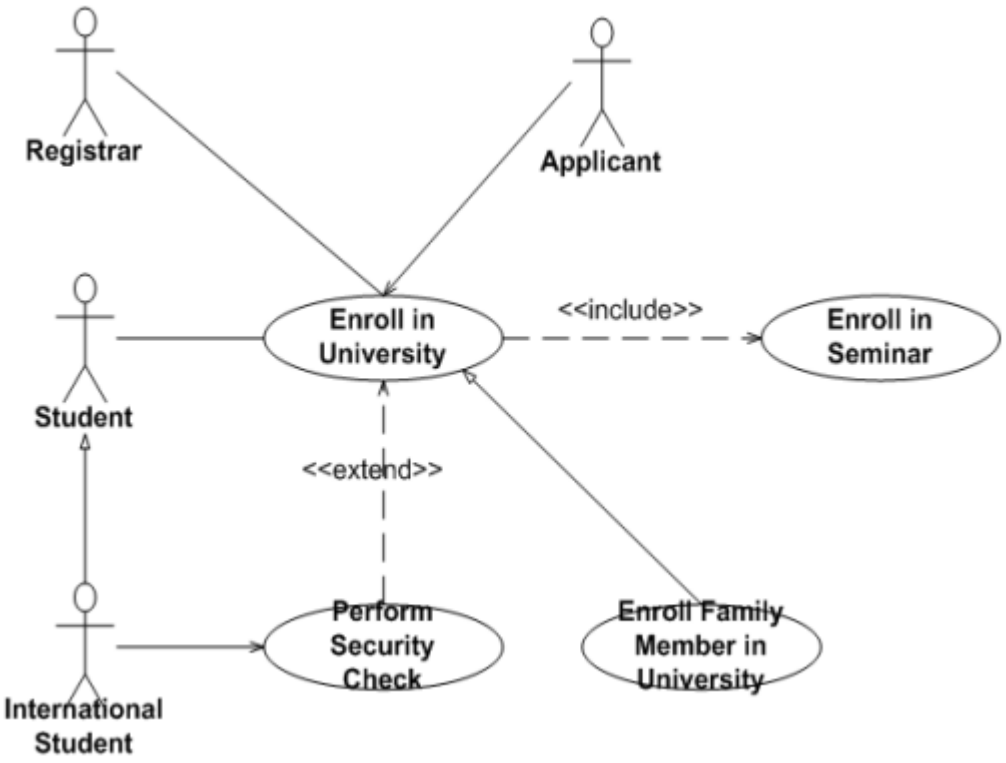


Fig: 1

Example of Relationships between Use cases



Extend vs Include



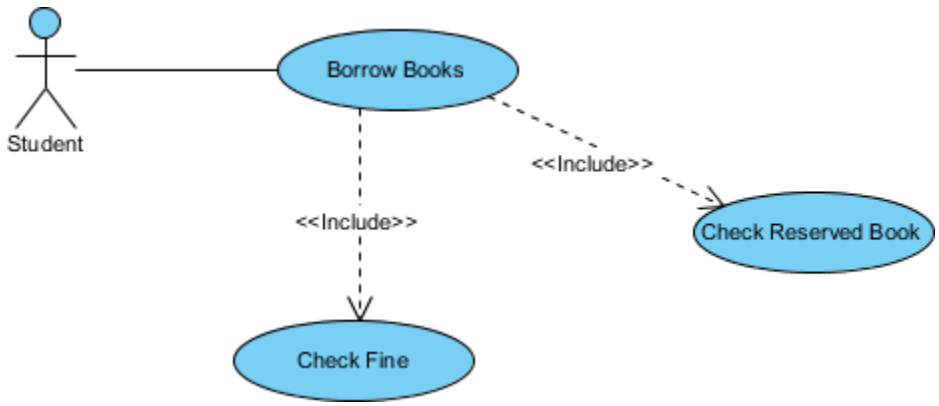
Use cases Example

1. Association Link:

- A Use Case diagram illustrates a set of use cases for a system, i.e. the actors and the relationships between the actors and use cases.

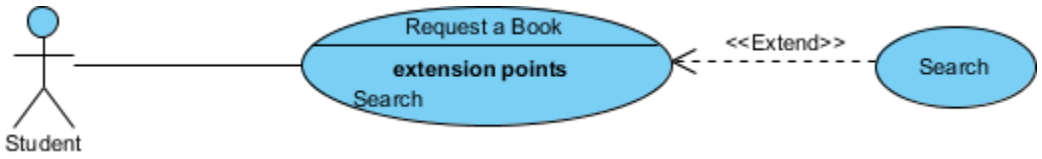
2. Include Relationship:

- The include relationship adds additional functionality not specified in the base use case. The <<Include>> relationship is used to include common behavior from an included use case into a base use case in order to support the reuse of common behavior.



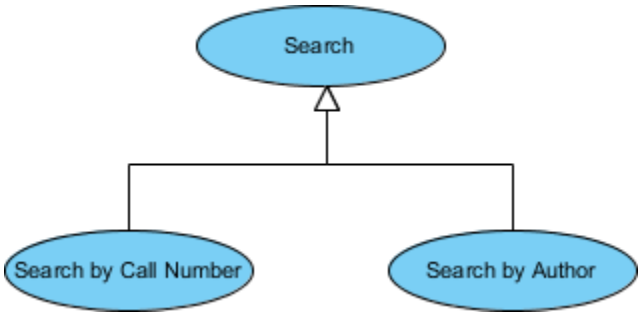
3. Extend Relationship:

- The extend relationships are important because they show optional functionality or system behavior. The <<extend>> relationship is used to include optional behavior from an extending use case in an extended use case. Take a look at the use case diagram example below. It shows an extend connector and an extension point "Search".



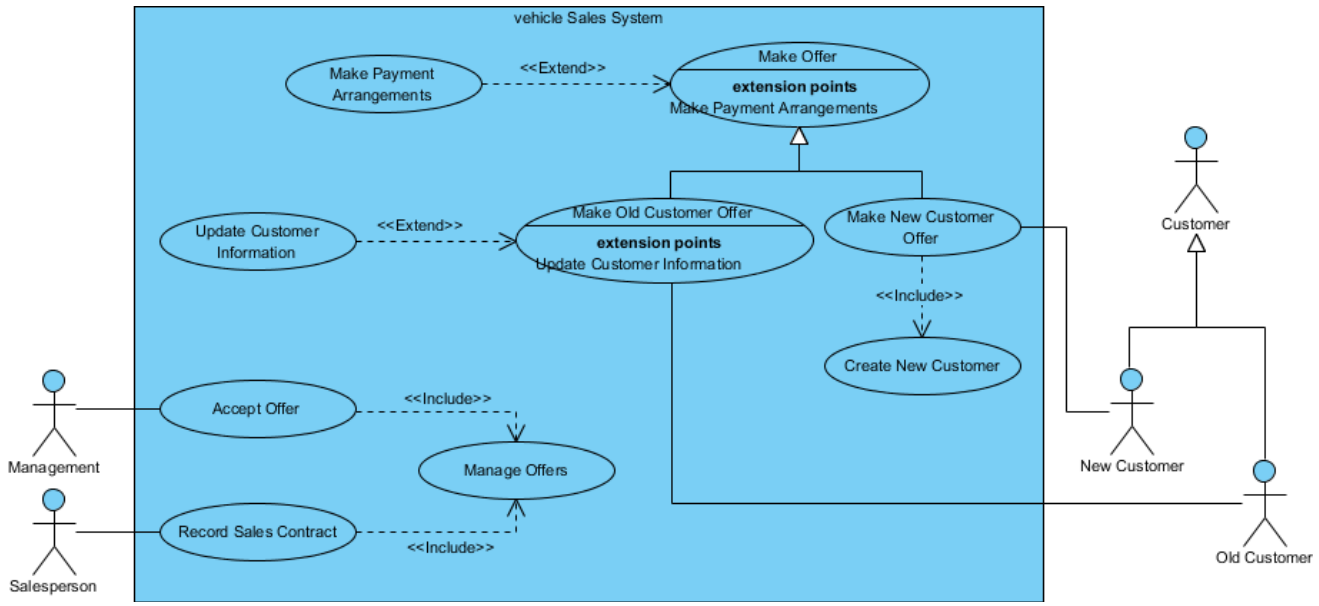
4. Generalization Relationship:

- A generalization relationship means that a child use case inherits the behavior and meaning of the parent use case. The child may add or override the behavior of the parent. The figure below provides a use case example by showing two generalization connectors that connect between the three use cases.



Use Case Diagram - Vehicle Sales Systems

- The figure below shows a use case diagram example for a vehicle system. As we can see even a system as big as a vehicle sales system contains not more than 10 use cases! That's the beauty of use case modeling.
- The use case model also shows the use of extend and include. Besides, there are associations that connect between actors and use cases.



Use cases description

- Title or Reference Name - meaningful name of the UC
- Author/Date - the author and creation date
- Modification/Date - last modification and its date
- Purpose - specifies the goal to be achieved
- Overview - short description of the processes
- Cross References - requirements references
- Actors - agents participating
- Pre-Conditions - must be true to allow execution
- Post Conditions - will be set when completes normally
- Normal flow of events - regular flow of activities
- Alternative flow of events - other flow of activities
- Exceptional flow of events - unusual situations
- Implementation issues - foreseen implementation problems

Use cases example – ATM money withdraw

- **Use case Title:** UC-1
- **Use Case:** Withdraw Money
- **Author:** Jack Lonagon
- **Created Date:** 1-OCT-2015
- **Modification Date:** 20-Apr-2017
- **Purpose:** To withdraw some cash from user's bank account
- **Overview:** The use case starts when the customer inserts his credit card into the system. The system requests the user PIN. The system validates the PIN. If the validation succeeded, the customer can choose the withdraw operation else alternative 1 – validation failure is executed. The customer enters the amount of cash to withdraw. The system checks the amount of cash in the user account, its credit limit. If the withdraw amount in the range between the current amount + credit limit, the system dispense the cash and prints a withdraw receipt, else alternative 2 – amount exceeded is executed.
- **Cross References:** UC-3

- **Actors**: Customer
- **Pre-Condition:**

  The ATM must be in a state ready to accept transactions

  The ATM must have at least some cash on hand that it can dispense

  The ATM must have enough paper to print a receipt for at least one transaction

- **Post Condition:**

  The current amount of cash in the user account is the amount before the withdraw minus the withdraw amount

  A receipt was printed on the withdraw amount

  The withdraw transaction was audit in the System log file

- **Typical Course of events:**

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM & PIN	3. System verifies the customer ID and PIN
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal (50k per day)
	9. System dispenses the cash
	10. System deduces the withdraw amount from account balance and set new balance
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

- **Alternative flow of events:**

  Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card. Some other thing may happen for example go to **step 2** try login again for maximum 3 times and after that the failed login attempt will seize and inform related authority of any suspicious activity in the ATM machine.

  Step 8: Customer has insufficient funds in its account. Display an error message and go to **step 6**.

  Step 8: Customer exceeds its legal amount. Display an error message and go to **step 4.**

- **Exceptional flow of events:**

  Power failure in the process of the transaction **before step 9,** cancel the transaction and eject the card

Identify Use cases

- **One method to identify use cases is actor-based:**
  - Identify the actors related to a system or organization.
  - For each actor, identify the processes they initiate or participate in.

- **A second method to identify use cases is event-based:**
  - Identify the external events that a system must respond to.
  - Relate the events (cash withdraw) to actors (customer) and use cases (ATM system).

- **The following questions may be used to help identify the use cases for a system:**
  - What are the tasks of each actor ?
  - Will any actor create, store, change, remove, or read information in the system ?
  - What use cases will create, store, change, remove, or read this information ?
  - Will any actor need to inform the system about sudden, external changes ?
  - Does any actor need to be informed about certain occurrences in the system ?

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.

- 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?

How to Identify Use Cases?

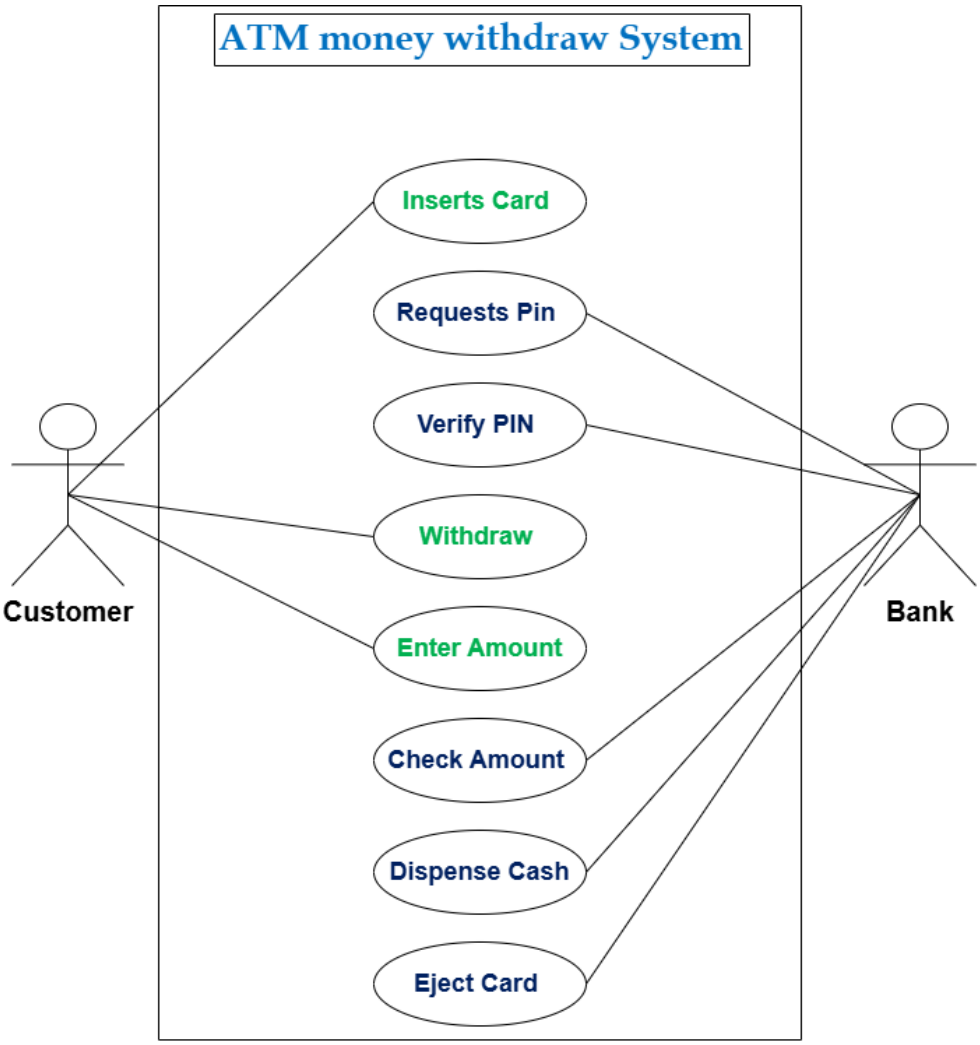
- Identifying the Use Cases, and then the scenario-based elicitation process carries on by asking what externally visible, observable value that each actor desires. The following questions can be asked to identify use cases, once your actors have been identified.

- 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?

Use Case Diagram – Case Studies

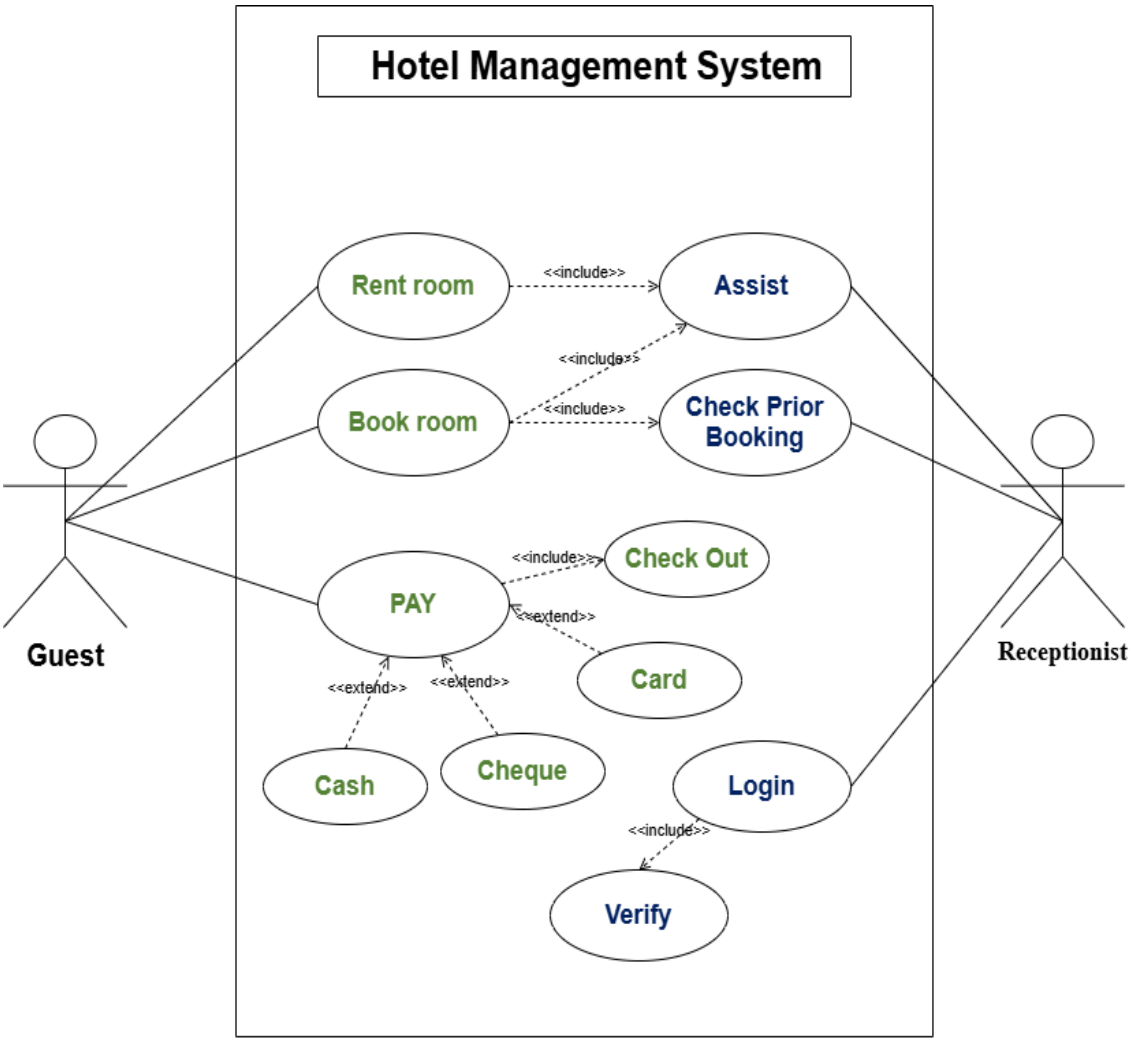
1. Case: 01

- The use case starts when the **customer** **inserts his credit** card into the **system** the system **requests the user PIN**. The system **validates the PIN**. If the validation succeeded, the customer chooses the **Withdraw operation**. The customer **enters the amount** of cash to withdraw. The system **checks the amount** of cash in the user account, its credit limit. If the withdraw amount is in the range between the current amount and credit limit, the system **dispense** the cash and prints a withdraw receipt and **eject the card.**



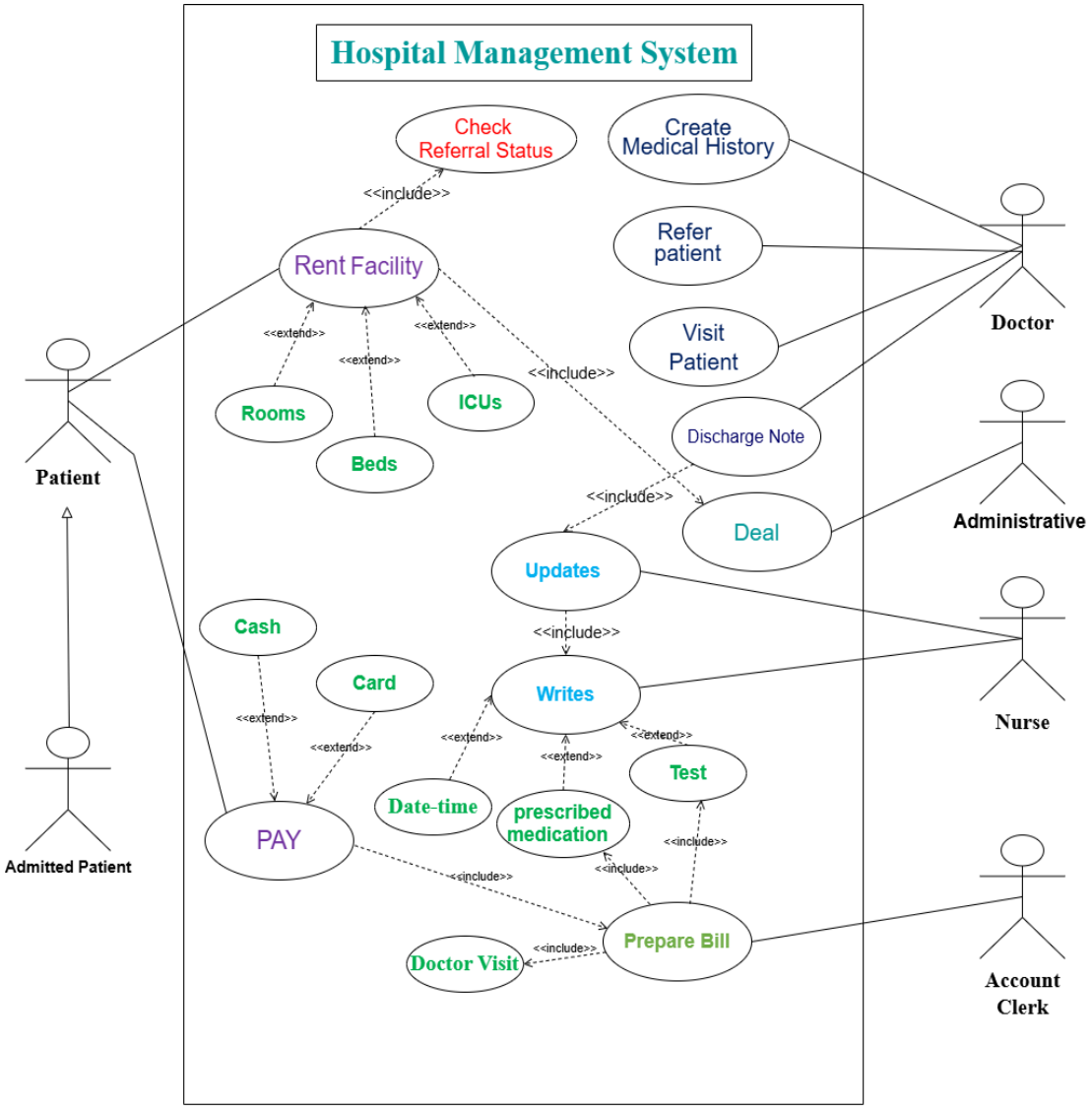
2. Case: 02

- In a hotel management system, a guest can **rent rooms**. Hotel receptionist uses the system **to assist in the renting**. A guest can also **book a room** for future renting with the **help of the receptionist**, but he has to **check** if the room has prior booking or not. A Guest **pays** for the he rooms when he **checks out**. He can pay by **cash, cheque, or credit card**. Receptionists has to **logon** to the system before they can use it, but to logon they have to go through username/password **verification.**



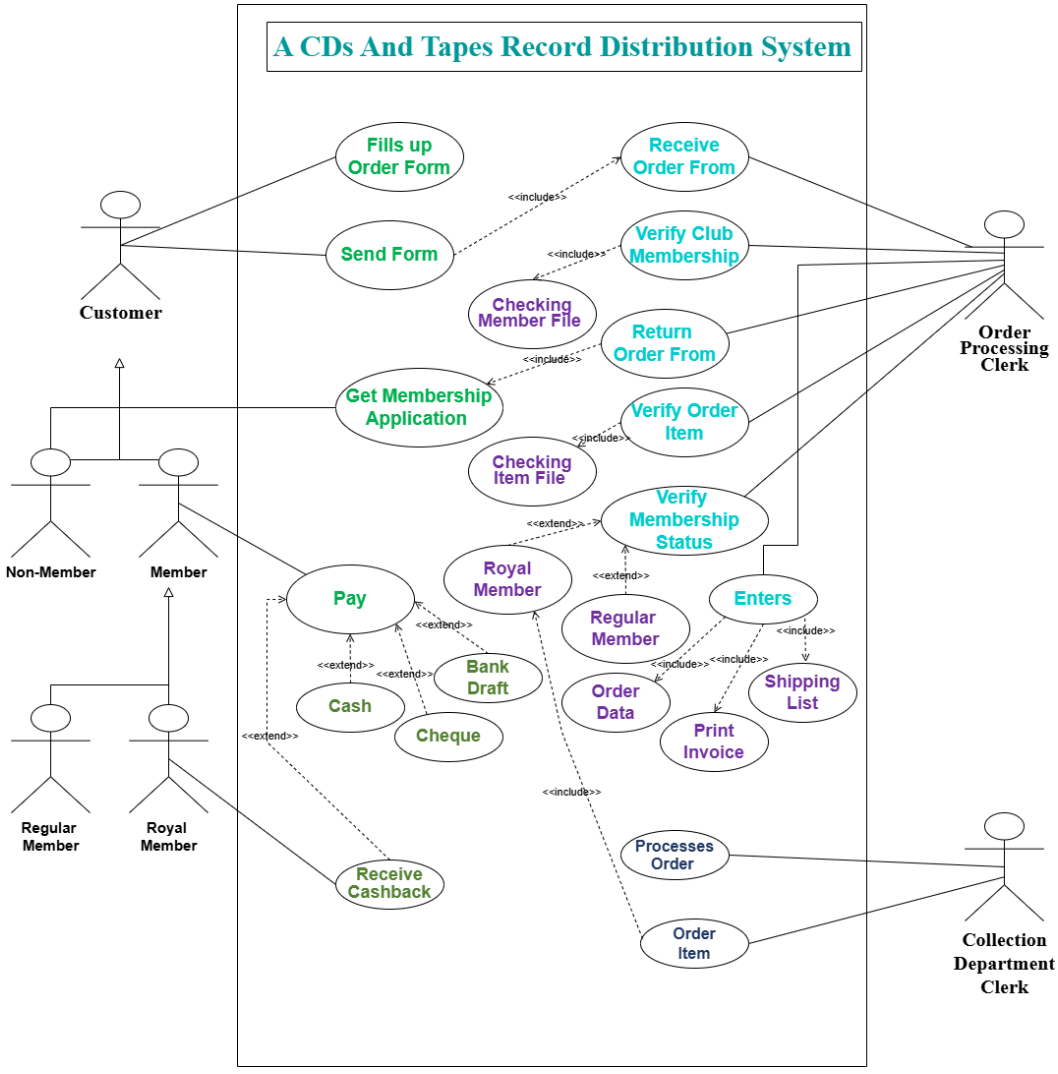
3. Case: 03

In a hospital management system, a patient's medical history is **created** by a doctor, A patient may be **referred** by a doctor to be admitted in the hospital. A patient can **rent hospital facilities** and an administrative officer **deal** with the renting. The system automatically **checks whether the patient is referred by a doctor** before he can rent any hospital facility. The types of facilities are **rooms, beds, or ICUs.** Admitted patients are regularly **visited** by doctors and a nurse **updates patient's medical history** after each visit. When the nurse updates the medical history, she **writes** either the **date-time** of doctor's visit or **prescribed medication or test**. A doctor also writes the **discharge note** of the patient when he leaves which is also updated in the patient's medical history. An account clerk **prepares the bill**. The bill is calculated from the elements written in the medical history, i.e. **number of doctor's visit, prescribed medication, tests**. The patient may **pay** by **cash or card** when the bill is prepared.



4. Case: 04

- ABCT Records is a company that distributes CDs and tapes at discount prices to club members. A customer **fills up an order form** and **sends** it to the order processing clerk. When the order processing clerk **receives** an order form, she **verifies whether the customer** is a club member by **checking the member file**. If the sender is not a member, the clerk **returns the order form along with a membership application form**. If the customer is a member, the clerk **verifies the order item** data by **checking the item file**. She also **checks the status** of the member whether he is a royal class member or regular class member. Then the clerk enters the order data, prints invoice and shipping list for each order. A collection department clerk **processes the order**. If the items are not available and the member is a royal class member, they are **ordered**. The members can **pay** by **cash, cheque or bank draft**. Royal class members sometimes **receive cashback** on payment.



Class Diagram

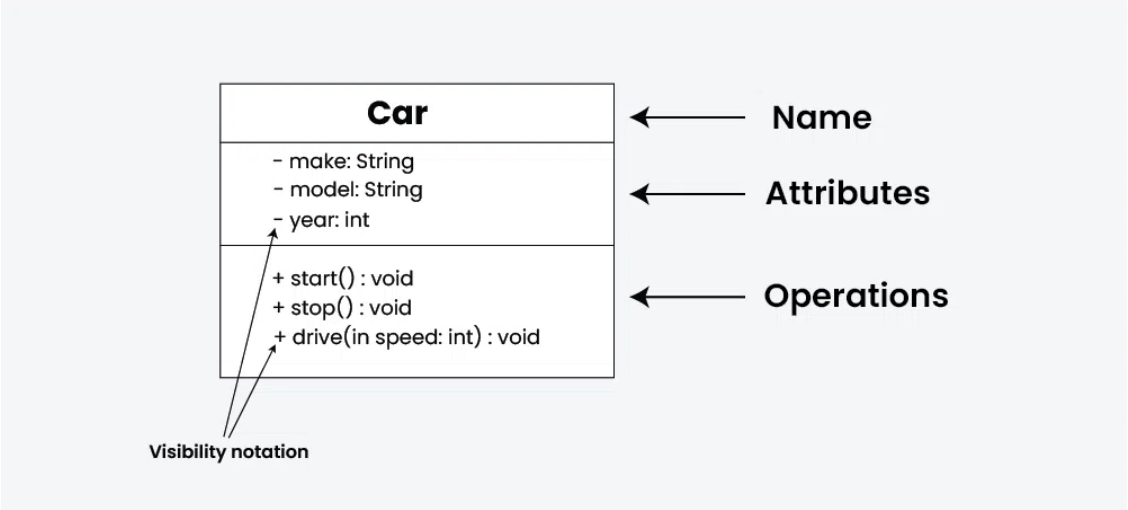
Ch: 03

What is class?

- A class is a blueprint or represents a collection of objects having same characteristic properties that exhibit common behavior.

- A class describes a group of objects with
  - similar properties (attributes),
  - common behavior (operations),
  - common relationships to other objects,
  - and common meaning ("semantics").

  - Examples:



CRC CARD

- **C**lass **R**esponsibility **C**ollaboration
- CRC goals: provide the simplest possible conceptual introduction to OO design

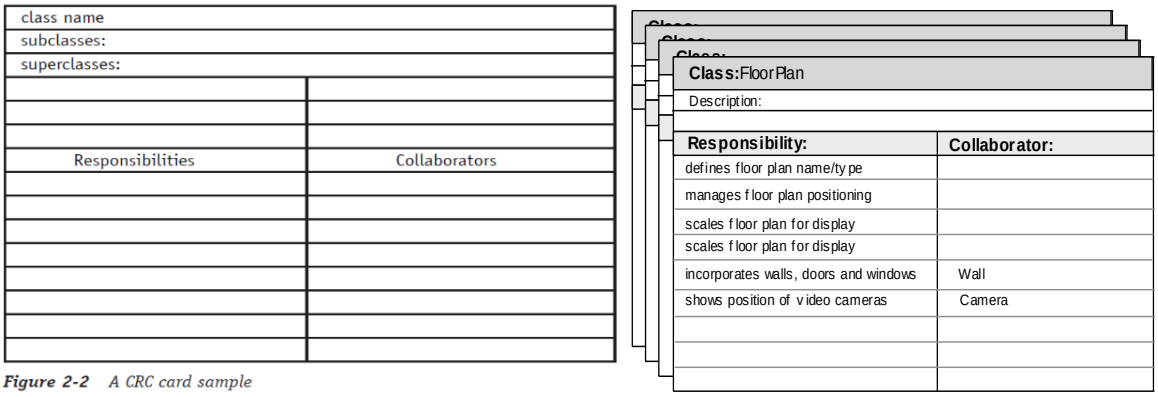
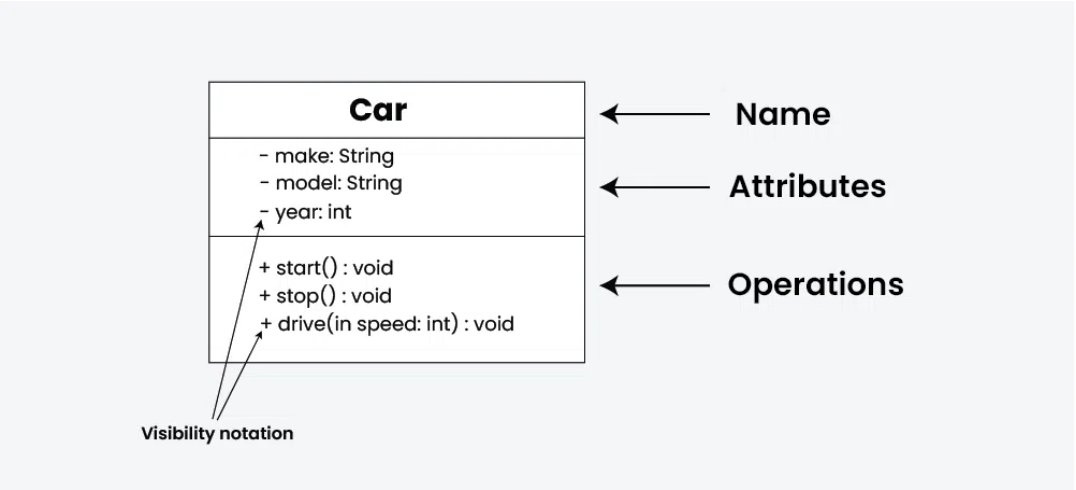


Figure 2-2 A CRC card sample

- A CRC card is a 3-x-5" or 4-x-6" lined index card.
- The physical nature of the cards emphasizes the division of responsibility across objects.
- The physical size of the cards also helps to establish limits for the size and complexity of the classes.
- The CRC card technique does not use the UML, instead it is used to discover information about classes that is then placed into a UML Class diagram.
- The body of the card is divided in half.
    - The left column/half lists the responsibilities of the class
    - The right column/half lists the other objects that it works with, the collaborators, to fulfill each responsibility.

Class Notation

- Class notation is a graphical representation used to depict classes and their relationships in object-oriented modeling.



1. **Class Name:**
   - The name of the class is typically written in the top compartment of the class box and is centered and bold.

2. **Attributes:**
   - Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.

3. **Methods:**
   - Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.

- NOTE: Classes are used to represent objects. Objects can be anything having properties and responsibility.

4. **Visibility Notation:**

- Visibility notations indicate the access level of attributes and methods. Common visibility notations include:
  - **"+"** for public (visible to all classes)
  - **"-"** for private (visible only within the class)
  - **"#"** for protected (visible to subclasses)
  - **"~"** for package or default visibility (visible to classes in the same package)
  - **Slash (/):** The derived attribute indicator is Optional

Drawing Class notation: Attribute

➢ **visibility / attribute name: data type = default value {constraints}**

- **Attribute name**: *Required.* Must be unique within the class. It is recommended that **attribute name should be a meaningful name** with the value it stores, i.e., **at least three character long.**
- **Data type**: *Required.* During analysis, the data type should reflect how the client sees the data.
- **Assignment operator and default value:** Default values (**usually zero**) serve two valuable purposes.
  - **First**, default values can provide significant ease-of-use improvements for the client.
  - **Second** and more importantly, they protect the integrity of the system from being corrupted by missing or invalid values.

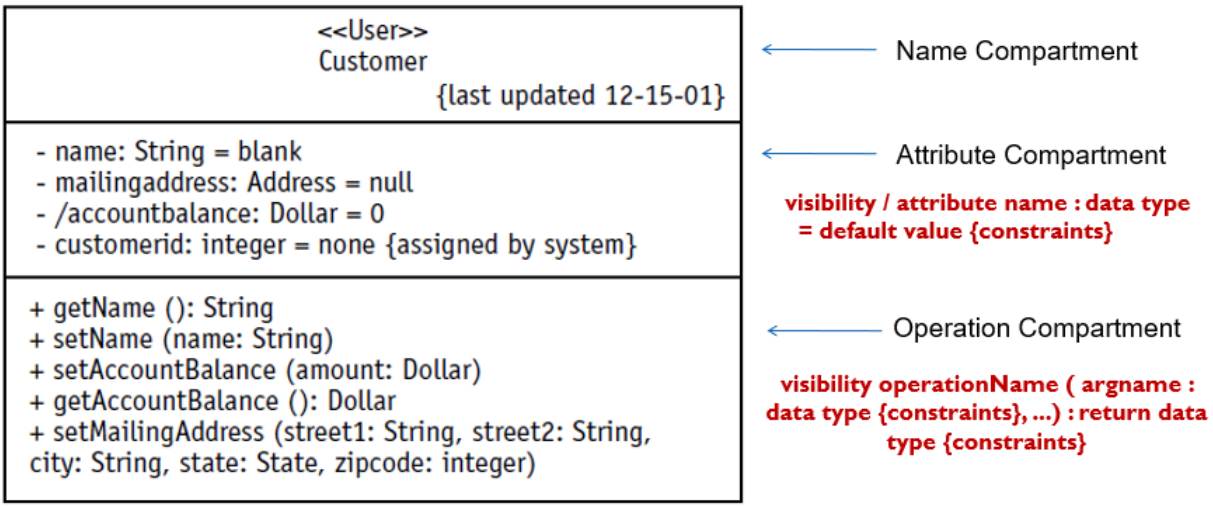
- **Constraints**: Constraints express all the rules required to guarantee the integrity of this piece of information. Any time another object tries to alter the attribute value, it must pass the rules established in the constraints. The constraints are typically implemented/enforced in any method that attempts to set the attribute value, e.g. {**ID is assign by the system**}

Drawing Class notation: Operation

➢ **visibility operationName (argname: data type {constraints}, ...): return data type {constraints}**

- **Operation name**: Required. Does not have to be unique, but the combination of name and parameters does need to be unique within a class.
- **Argument name:** Required for each parameter, but parameters are optional. Any number of arguments is allowed.
- **Argument data type:** Required for each parameter, but parameters are optional.
- **Constraints:** Optional. In general, constraints express rules that must be enforced in the execution of the operation.
- **Return data type:** Required for a return value, but return values are optional.

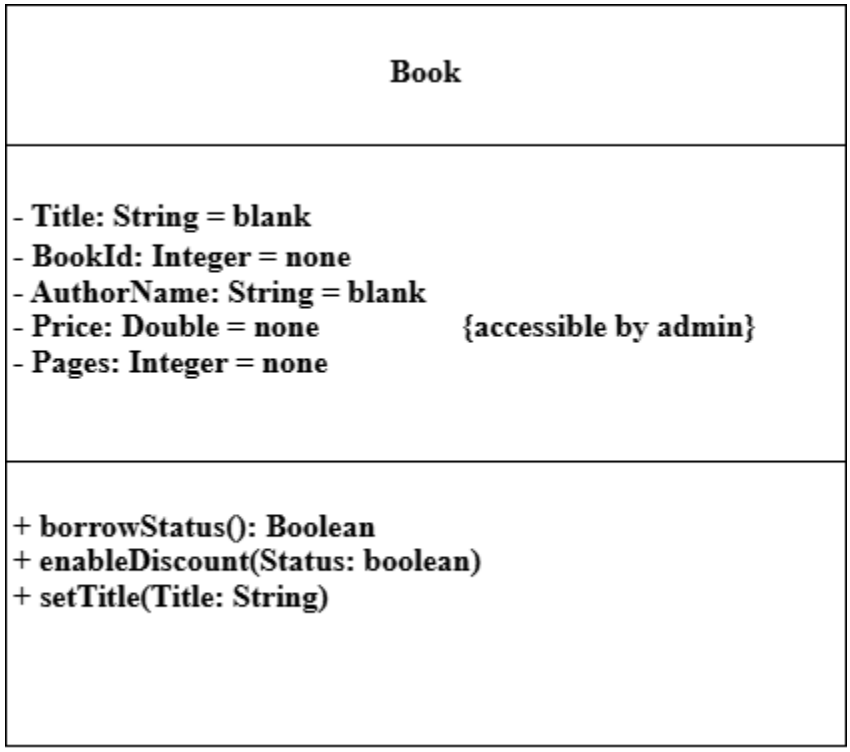
Example:



Practice Class Notation

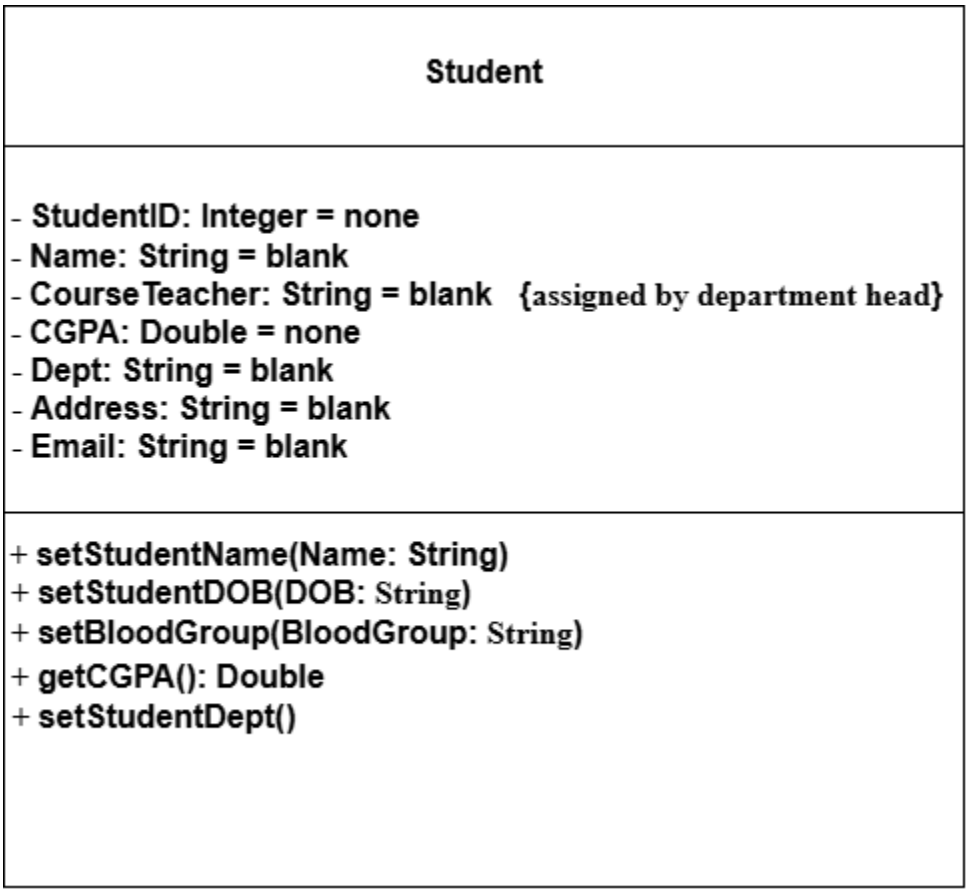
1. Case: 01

- In a system, there is a class named **Book**. It has five attributes: **Title**, **BookId**, **AuthorName**, **Price**, and **Pages**. All the attributes are **private** and **Price is only accessible by admin user**. **Title** and **AuthorName** are **String** in type, **BookId** and **Pages** are in integer and **Price is in double**. There are three methods in this class. They are**:** **borrowStatus()** which returns a boolean value, **EnableDiscount(boolean Status)** which returns nothing and **setTitle(String title)** which returns nothing. All the methods are public.



2. Case: 02

- Every year there are many students admitted to AIUB. AIUB authority stores different related data of each student in their database. Basic information of students is: **StudentID, Name, CourseTeacher, CGPA, Dept, Address and Email**. All information keeps **private** by the university authority. The **head** of the **department assigned** the **CourseTeacher** for the students every semester. All information data types in **String** except **StudentID** and **CGPA.** The data type of StudentID is an **integer** and CGPA is **double**. The methods of student's information are: **setStudentName(String Name)**, **setStudentDOB(String DOB)**, **setBloodGroup (String BloodGroup), getCGPA(),** and **setStudentDept().** Only **getCGPA()** method returns a **double value.** All methods are public.



3. Case: 03

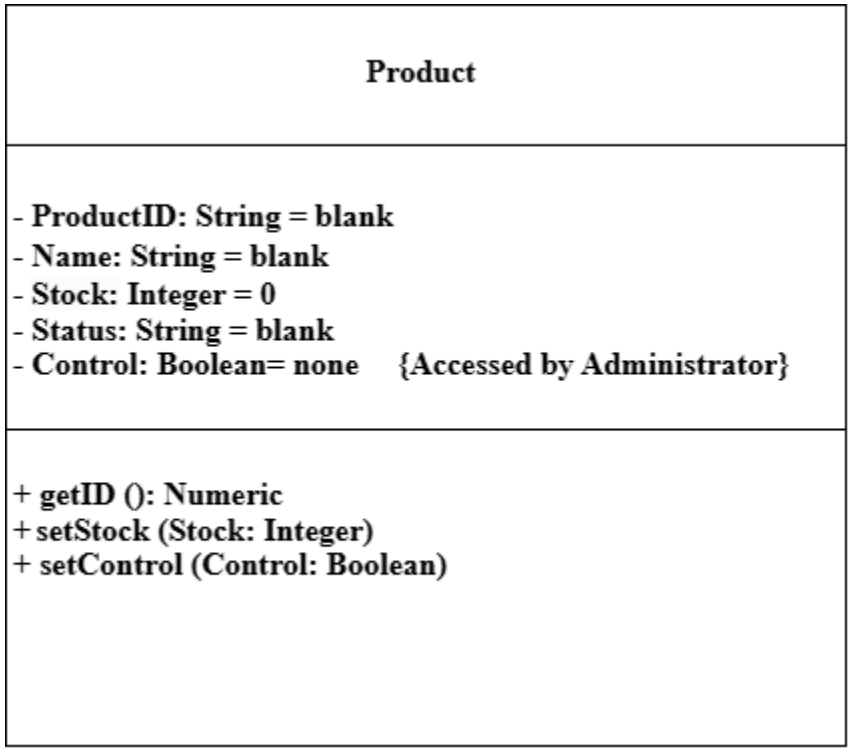
⇨ Draw the Class notation for Class Named ' **Product** '.

Private Attributes of the class are:

- ProductID (Datatype: String)
- Name (Datatype: String)
- Stock (Datatype: Integer, default value: zero)
- Status (Datatype: String)
- Control (Datatype: Boolean, type: static, constraint: Accessed by Administrator only)

Public Operations of this class are:

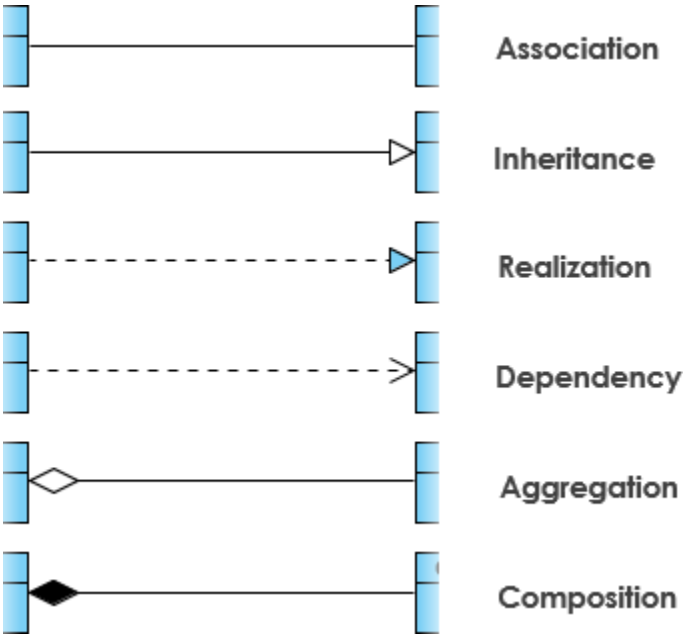
- getID (return datatype: Numeric)
- setStock (argument. Stock; datatype: Integer)
- setControl (argument: Control, datatype: Boolean).



Class Diagram Relationships

- In class diagrams, relationships between classes describe how classes are connected or interact with each other within a system.

There are:



1. Association:

- An association represents a bi-directional relationship between two classes. It indicates that instances of one class are connected to instances of another class.

- Example:

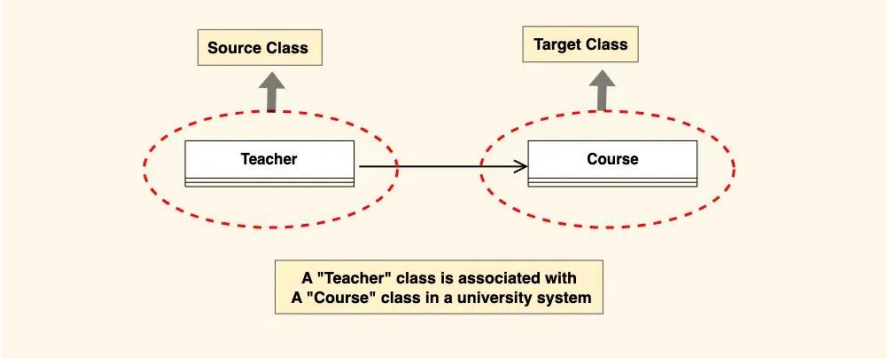
⇨ A department is associated with the college.

❑ Directed Association:

- In a directed association, an arrowhead is added to the association line to indicate the direction of the relationship. The arrow points from the class that initiates the association to the class that is being targeted or affected by the association.

❑ Example:

⇨ A "Teacher" class is associated with a "Course" class in a university system. The directed association arrow may point from the "Teacher" class to the "Course" class, indicating that a teacher is associated with or teaches a specific course.



2. Aggregation:

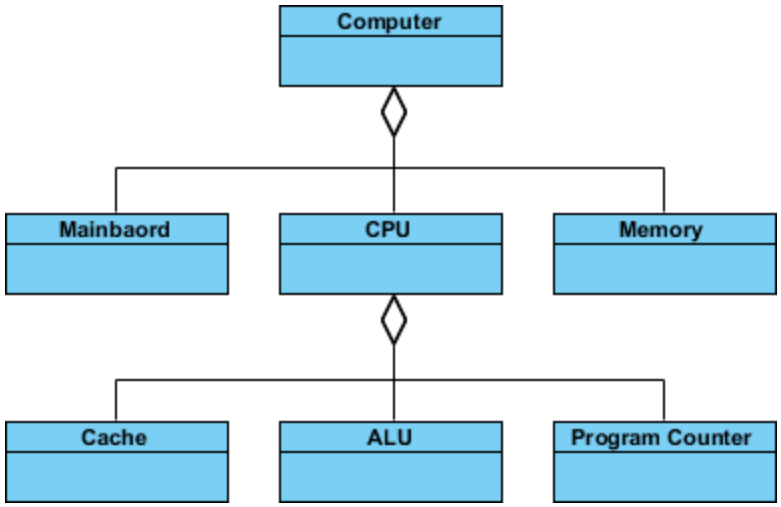
➢ **A special type of association. It represents a "part of" relationship.**

- 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

❑ Example 1:

⇨ The company encompasses a number of employees, and even if one employee resigns, the company still exists.

❑ Example 2:



3. Composition:

- The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

Example:

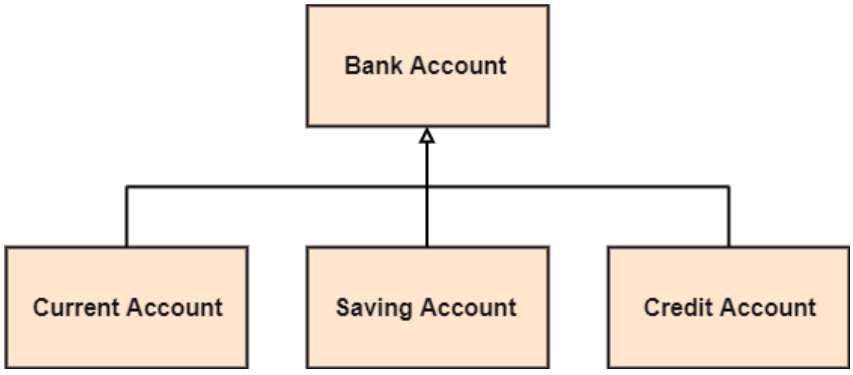
- A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

4. Generalization (Inheritance):

- A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class.

Example:

- The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.

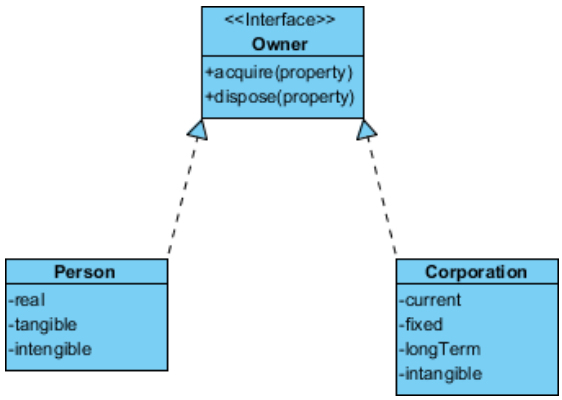


5. Realization (Interface Implementation):

- This as the relationship between the interface and the implementing class.

Example:

- The Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.

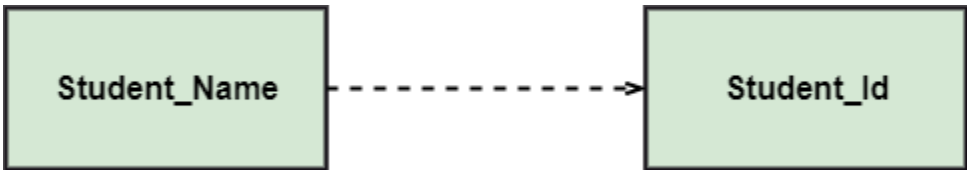


6. Dependency:

- A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.

Example:

- The In the following example, Student_Name is dependent on the Student_Id.

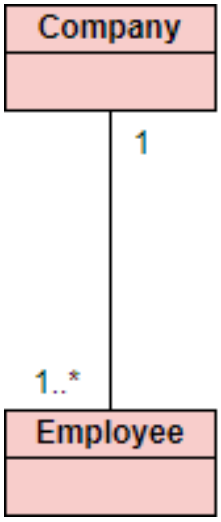


Multiplicity

➔ It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

❑ Example:

⇨ One company will have one or more employees, but each employee works for one company only.



Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Multiplicity Indicator

Exactly one	1
Many	*
Zero or more	0..*
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

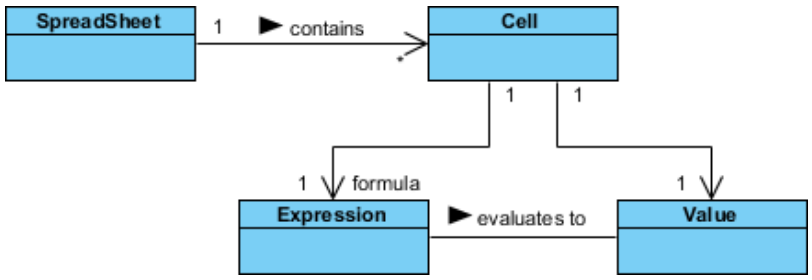
Relationship Names

⇨ Names of relationships are written in the middle of the association line.

➢ Good relation names make sense when you read them out loud

❑ Example:

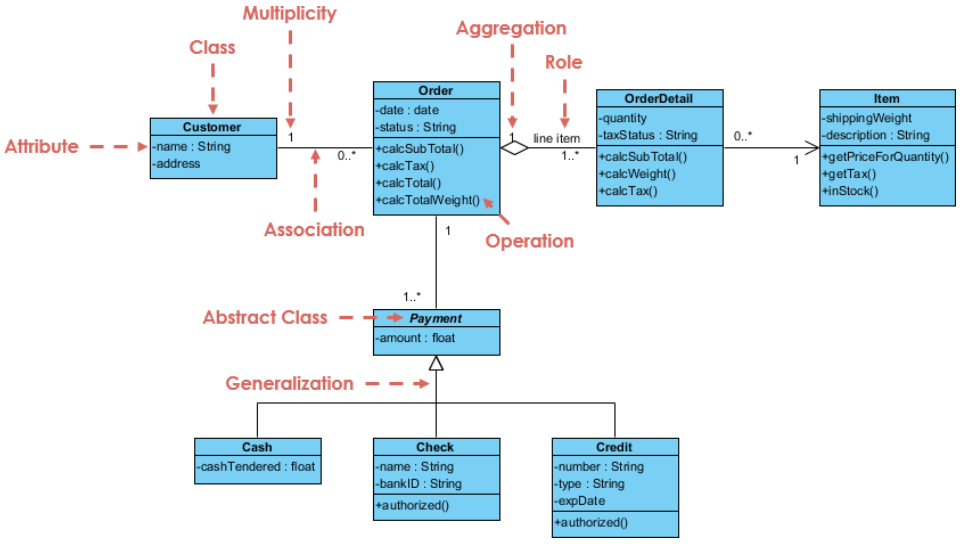
- "Every spreadsheet contains some number of cells",
- "an expression evaluates to a value"



- They often have a **small arrowhead to show the direction** in which direction to read the relationship, e.g., expressions evaluate to values, but values do not evaluate to expressions.

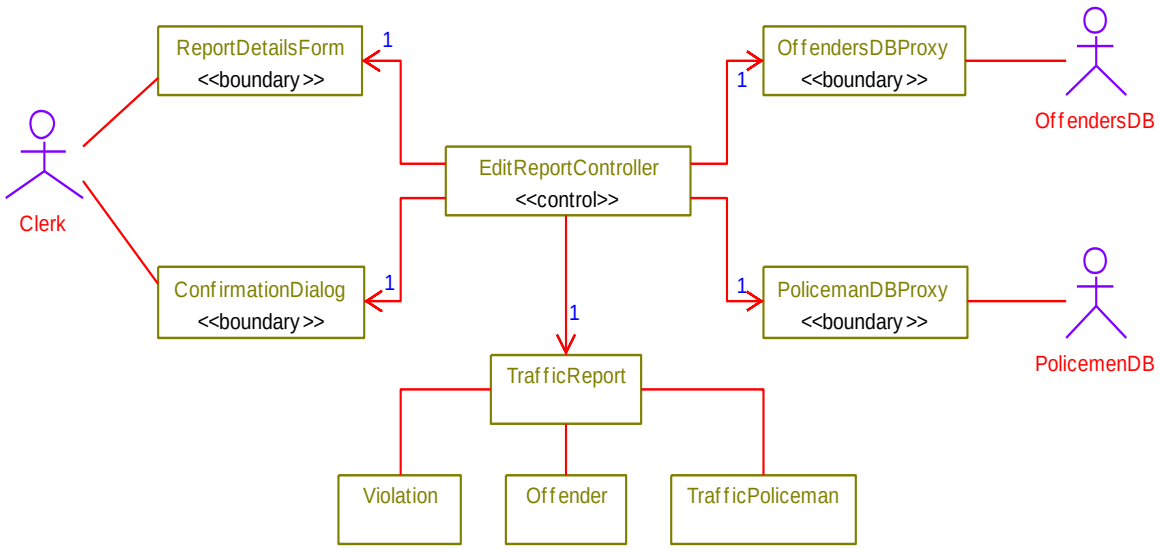
Class Diagram - Example

– Example: Order System



Classes Categorization

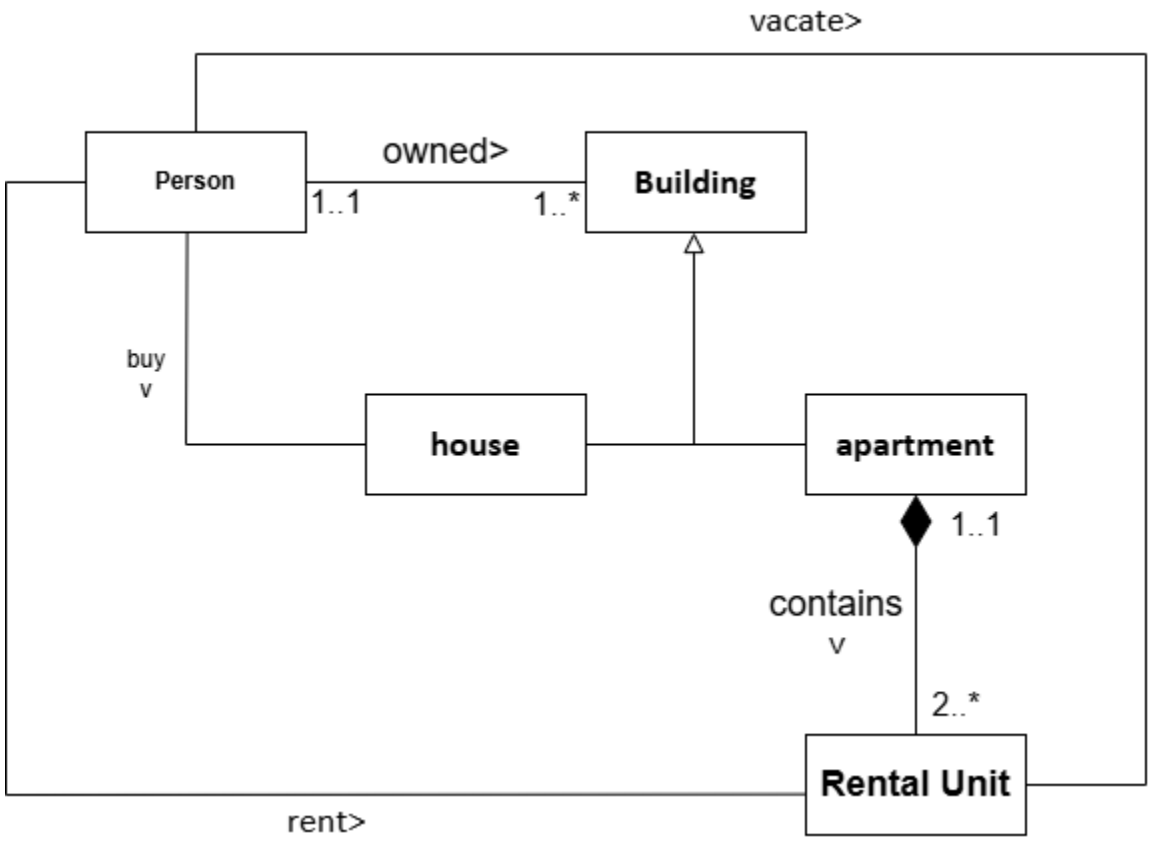
- **Boundary Classes**
  - Models the interaction between the system's surroundings and its inner workings
  - User interface classes, Concentrate on what information is presented to the user, don't concentrate on user interface details
  - System / Device interface classes, concentrate on what protocols must be defined. don't concentrate on how the protocols are implemented
- **Entity Classes**
  - Models the key concepts of the system
  - Usually models information that is persistent
  - Contains the logic that solves the system problem
  - Can be used in multiple behaviors
- **Control Classes**
  - Controls and coordinates the behavior of the system
  - A control class should tell other classes to do something and should never do anything except for delegating (directing) the work to other classes
  - Control classes separate boundary and entity classes



Class Diagram – Case Studies

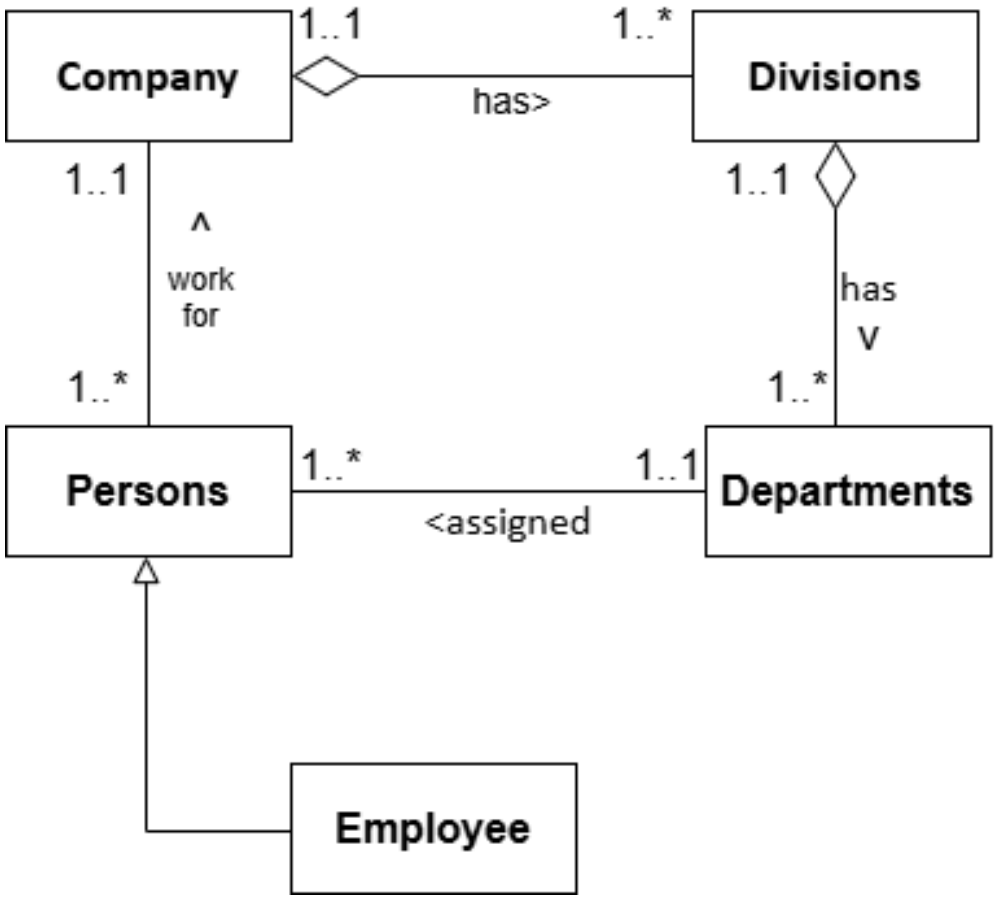
Case: 01

- A **building** is owned by one **person**. A person may own more than one building. Each building is either a **house** or an **apartment**. An apartment contains two or more **Rental Units**. It is possible to buy a house, and rent or vacate a Rental Unit



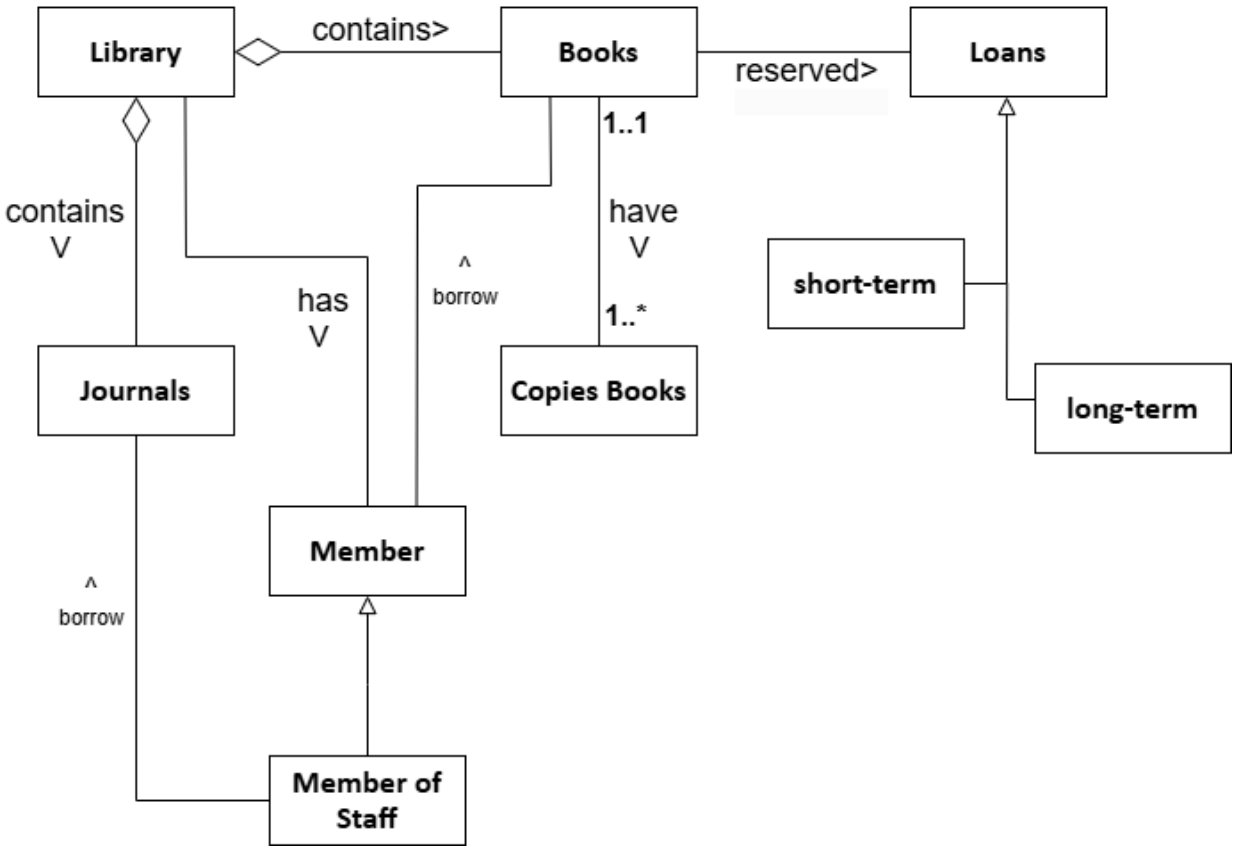
Case: 02

A **company** has one or more **divisions**. A division has one or more **departments**. A company can have one or more **persons** working as employees. A person is associated with the company as an **employee**. A department can have one or more persons assigned to it. A person is assigned to one department.



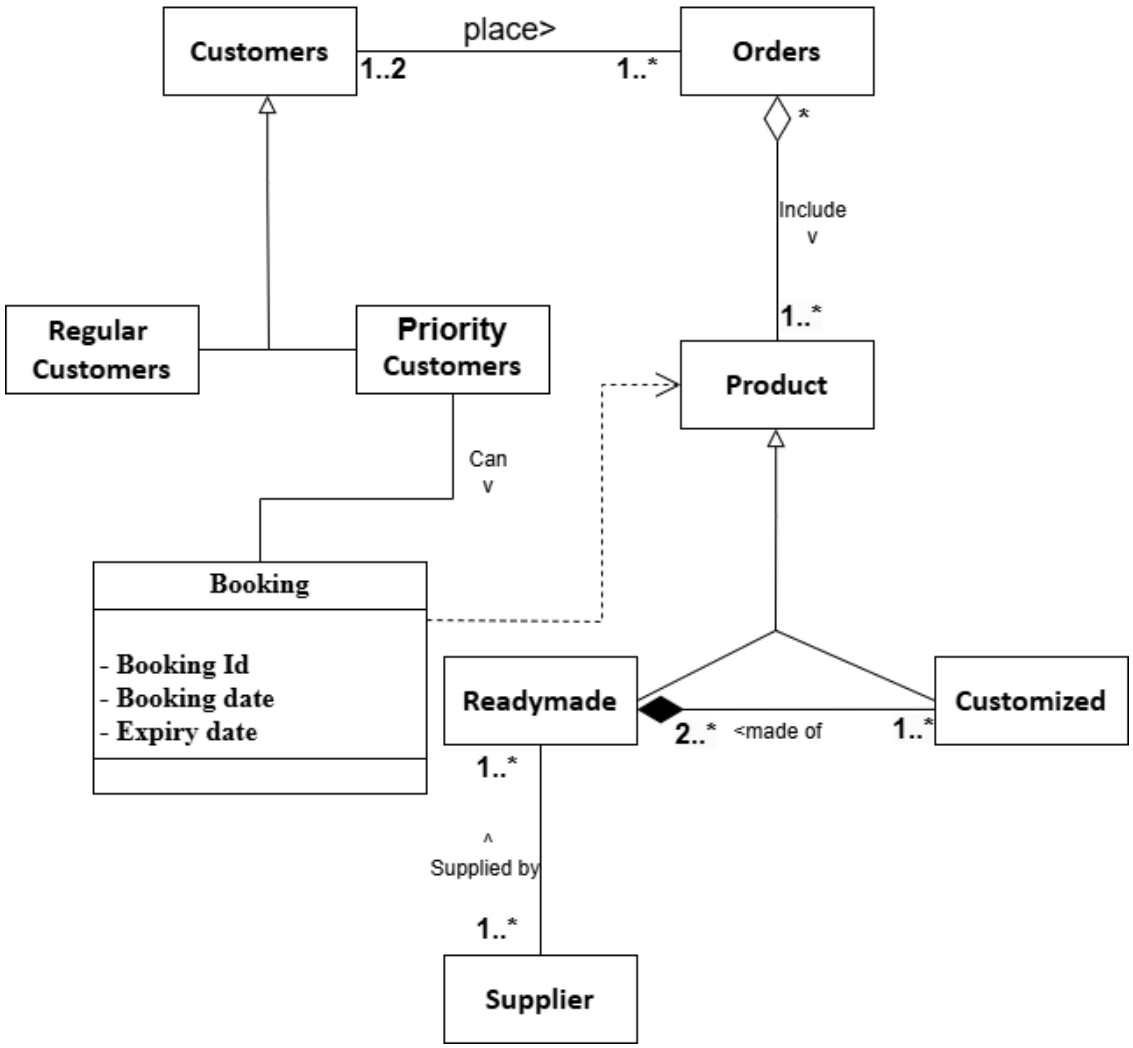
Case: 03

- The **library** contains **books** and **journals**. It may have several **copies** of a given book. Some of the books can be reserved for **short-term loans** and others for **long-term**. **Member** of library can borrow books. There is a special type of member- **Member of Staff**, who can borrow the journals. The system must keep **track** of when books and journals are borrowed.



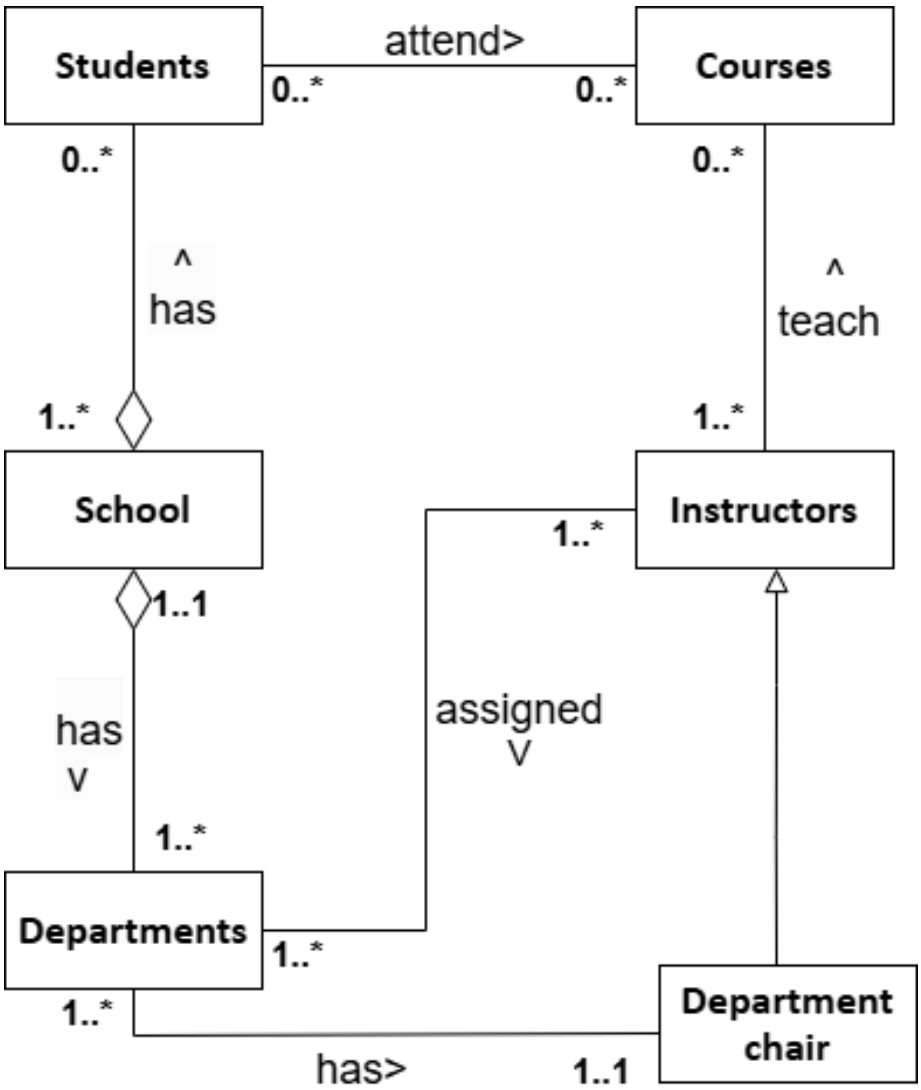
Case: 04

A system is required to manage customers, and their orders. **Customers** can place **orders** for products. One customer can place as many orders as they wish; one order can be placed by minimum 1 or maximum 2 customers. One order must include at least one **product**. One product may be included in many orders. The organization sells two types of products - **readymade** and **customized**. A readymade product is sold as it is bought from the **supplier**. A customized product is made of at least 2 readymade products. One readymade product can be bought from many suppliers. One supplier can sell many readymade products. The organization has two types of customers-**regular** and **priority**. Priority customers can also book products. Every booking is identified by a booking ID. There are also booking date and expiry date attributes for every booking.



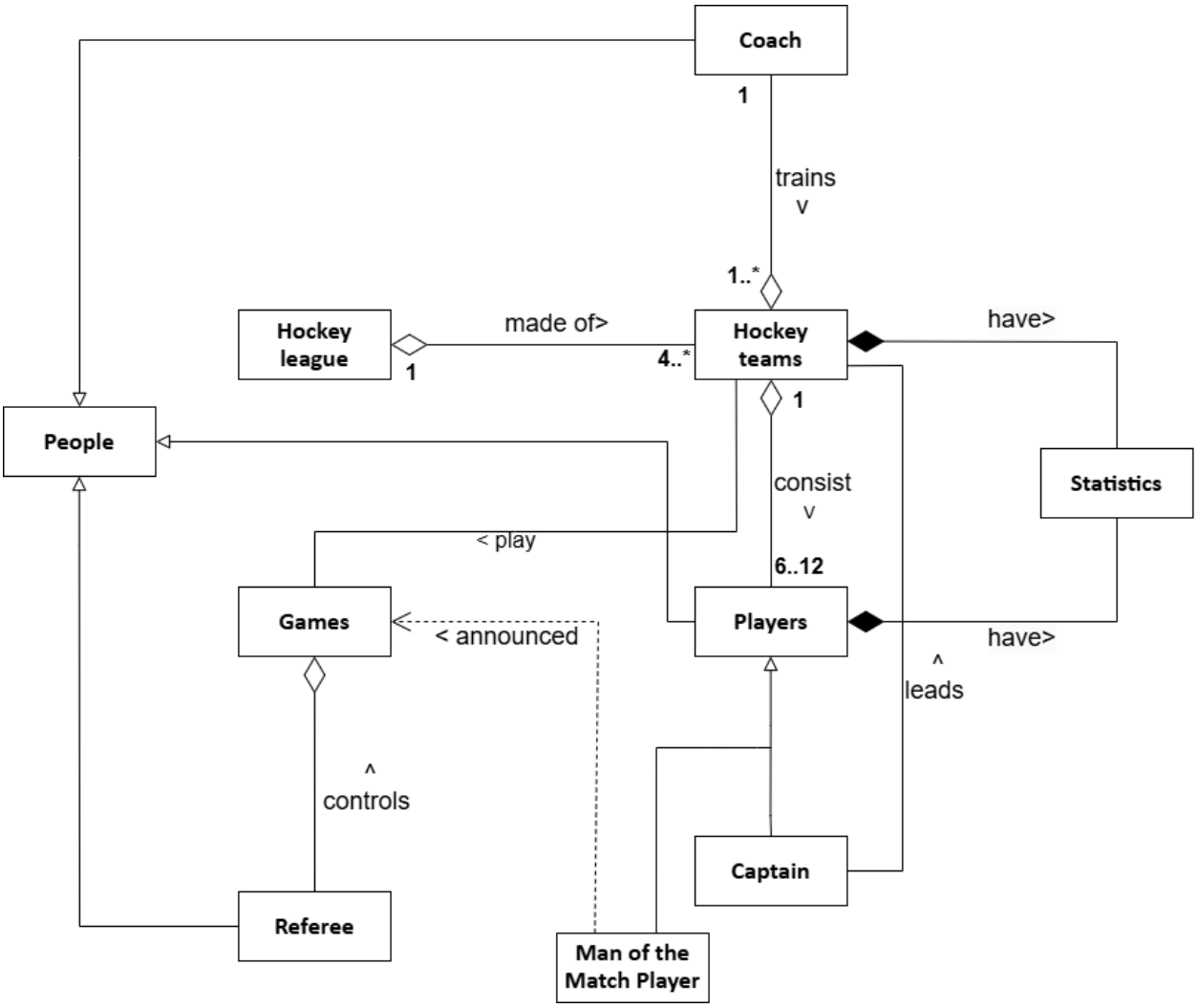
Case: 05

- A **Students** may attend any number of **courses**. Every course may have any number of students. **Instructors** teach courses. For every course there is at least one instructor. Every instructor may teach zero or more courses. A **school** has zero or more students. Each student may be a registered member of one or more schools. A school has one or more **departments**. Each department belongs to exactly one school. Every instructor is assigned to one or more departments. Each department has one or more instructors. For every department there is exactly one instructor acting as the **department chair**.



Case: 06

A **hockey league** made of at least four **hockey teams**. A hockey team can consist of 6 to 12 **players**. Players have a **captain** who leads the hockey team. Players and hockey teams have **statistics**. Hockey teams play games against each other. A **referee** controls the game. At the end of each match, a player is announced as **Man of the Match**. A **coach** trains the hockey team. A coach may train one or more hockey team. The Coach, players and referees are people.



MCQ Practice

⇨ A **hockey league** made of at least four **hockey teams**. A hockey team can consist of 6 to 12 **players**. Players have a **captain** who leads the hockey team. Players and hockey teams have **statistics**. Hockey teams play games against each other. A **referee** controls the game. At the end of each match, a player is announced as **Man of the Match**. A **coach** trains the hockey team. A coach may train one or more hockey team. The Coach, players and referees are **people**.

1. The Relationship between **hockey league** and **hockey teams** is:

   a) Association
   b) Composition
   c) Aggregation
   d) Dependency

2. The Relationship between **players** and **captain** is:

   a) Association
   b) Composition
   c) Aggregation
   d) Generalization

3. The Relationship between **players** and **statistics** is:

   a) Association
   b) Composition
   c) Aggregation
   d) Generalization

4. The Relationship between **people** and **coach** is:

   a) Association
   b) Composition
   c) Aggregation
   d) None

5. The Relationship between **hockey league** and **referee** is:

   a) Association
   b) Generalization
   c) Aggregation
   d) None

6. The Relationship between **hockey teams** and **captain** is:

a) Association
c) Aggregation
b) Generalization
d) None

7. Which of the following is not a part of the Class Diagram?

a) Attributes
c) Operations
b) Object
d) Stereotype

8. What does reflexive association mean?

a) Association among objects of two different classes
c) Association among objects of the same class
b) Association among objects of two different classes using index
d) Association among different subclasses of the same parent

9. Which of the following addresses the verbs of the domain?

a) Structural Things
c) Behavioral Things
b) Annotational Things
d) Grouping Things

10. Which of the following classes is created due to relationship?

a) Component
c) Active Class
b) Interface Class
d) Association Class

11. In Which of the following phases of **RUP** the least amount of **Testing** is Carried out?

a) Inception
c) Elaboration
b) Construction
d) Transition

12. Which one of the following is not part of a Static Diagram?

a) Association

c) Interaction

b) Generalization

d) Aggregation

13. Alpha testing is expected at the end of following phases of RUP?

a) Inception

c) Elaboration

b) Construction

d) Transition

14. In UML, Which of the following type of visibility allows access only inside the class?

a) Public

c) Protected

b) Private

d) Packaged

15. Super Class - Sub Class relationship known as?

a) Association

c) Aggregation

b) Composition

d) Generalization

16. What is the purpose of a use case diagram?

a) To represent a structural design view

c) To express how a system performs its tasks for actor

b) To address the dynamic interaction view of the system

d) To establish a common understanding with end user

17. Which of the following represents exchange of messages among objects?

a) State Machine
c) Activity
b) Interaction
d) Component

18. The number of objects involved in association relationship is identify by:

a) Association Role
c) Multiplicity
b) Association Constraint
d) Qualified Association

19. Which of the following represents whole-part relationship among objects?

a) Association
c) Aggregation
b) Composition
d) Generalization

20. Value of an attribute that remains same for all the object of a class is known as:

a) Default Value attribute
c) Derived attribute
b) Slash
d) Static attribute

21. Which of the following is not a part of the Class Diagram?

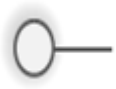
a) Attributes
c) Operations
b) Object
d) Stereotype

22. Which of the following is a **Specialized** element?

a) Parent Class
c) Super Class
b) Child Class
d) Interface

Match the notation of **next 5 question** with the item in following list:

a) Note b) Interface c) Interaction d) Active Class e) Node

23.  24.  25.  26.  27. 

28. Which of the following relationship is not found in a use case diagram?

a) Association c) Realization

b) Dependency d) Generalization

29. Class Diagram represents?

a) Static use case view c) Static design view

b) Static process view d) Dynamic process view

30. Interaction is a pre-requisite of:

a) Behavior c) Things

b) Relationship d) None

31. Which of the following is an abstract item?

a) Component c) Association Class

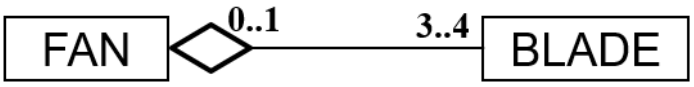
b) Active Class d) Interface Class

32. Value of an attribute that remains same for all the object of a class is known as:

a) Default Value attribute c) Derived attribute

b) Slash d) Class level Attribute

Next 2 question refer the following diagram:



33. Which of the following statement is correct based on the above diagram?

a) A fan contains maximum 4 blades and minimum 0 blade

c) A fan may have minimum 3 blade; maximum 4 blades.

b) A fan contains minimum 4 blades or more

d) A fan contains exactly 4 blades

34. Which of the following statement is correct based on the above diagram?

a) A blade is not a part of any fan

c) A blade must belong exactly one fan

b) A blade maybe a part of a maximum one fan or no fan at all.

d) A blade belongs to minimum 1 fan.

35. The property of inheritance is denoted by the relationship known as:

a) Association

c) Realization

b) Dependency

d) Generalization

36. A view that captures the vocabulary of the system is?

a) Process view

c) Design view

b) Deployment view

d) Use case view

37. Which of the following is a behavioral thing that focuses on the lifeline of the object?

a) State Machine

c) Activity

b) Interaction

d) Component

38. Which of the following a Class?

a) Component

c) use case

b) Interface

d) Association

39. Which of the following is general element?

a) Child Class

c) Sub Class

b) Interface

d) Parent Class

40. Qualified association in the class diagram indicates a qualifier to be used for:

a) Inheritance

c) Indexing

b) Interaction

d) Instantiation

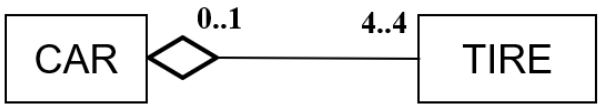
41. Which of the following is pre-requisite of Interaction?

a) Behavior

c) Things

b) Relationship

d) None



42. Which of the following statement is correct based on the above diagram?

a) A car contains maximum 4 tires and minimum 1 tire

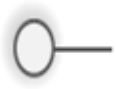
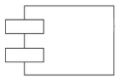
c) A tire must belong to exactly one car.

b) A car contains minimum 4 tires or more.

d) A tire may belong to maximum one car Or no car at all.

Match the notation of **next 4 question** with the item in following list:

a) Artifact b) Interface c) Component d) Active Class e) Node

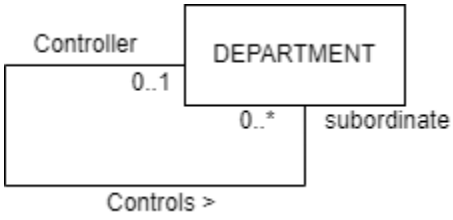
43.  **44.**  **45.**  **46.** 

Match the notation of **next 4 question** with the item in following list:

a) Dependency b) Generalization c) Interaction d) Aggregation e) Realization

47.  **48.**  **49.**  **50.** 

True/False Practice

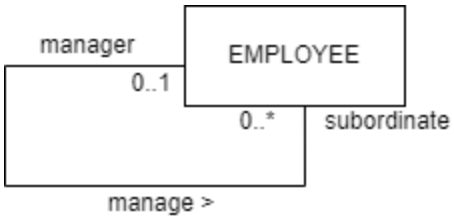


Question No. 1-3 are based on the above diagram

1. Every department must be controlled by exactly one other department.

2. There may be no subordinate under one department.

3. Every department must be controlled by another department,



Question No. 4-6 are based on the above diagram

4. Every employee must be managed by exactly one other employee

5. There may be no subordinate under one employee.

6. Every employee must be a manager.

7. The maximum multiplicity of the association at the whole side can be **more than one** in case of composition.

8. Generalization is a top-down process.

9. Objects are templates of classes.

10. Relationship is prerequisite of interaction.

11. Dependency relationships in use case diagram are of two types - include and exclude

12. Behavioral things address mostly the dynamic part of the system.

13. Association Class stores information about relationship.

14. Objects are blueprints of classes.

15. Structural things address mostly the runtime environment of the system.

16. Relationship is prerequisite of interaction.

17. Association Class encapsulates information about relationship.

18. Generalization is a bottom-up process.

19. Relationship among actors is identified using association in a use case diagram

20. In a use case diagram, a system can be an actor of another system.

21. Include dependency relationship in a use case diagram is denoted for optional condition.

22. Structural things address mostly the dynamic part of the system.

23. Qualified association provides approximately the same functionality as indexes in database.

24. The minimum multiplicity of the association at the whole side can be **zero** in case of composition.

25. Association Class encapsulates information about relationship.

26. Interaction is prerequisite of behavior.