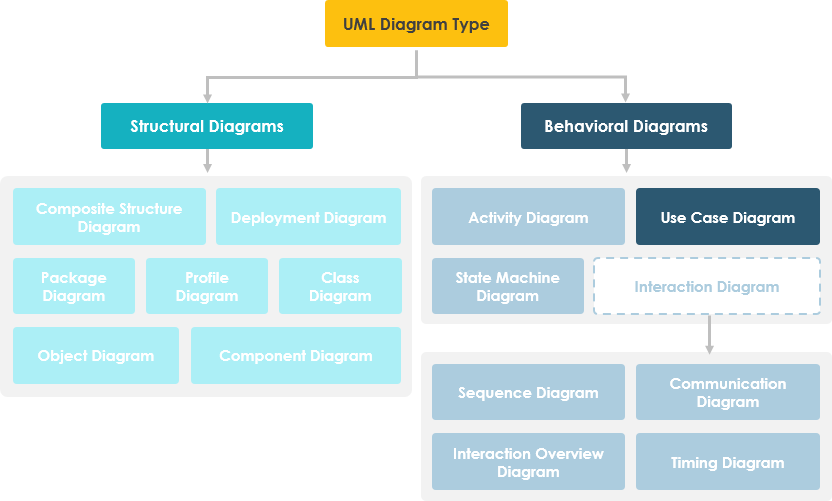
**UML Diagrams**

**Use Case Diagram**



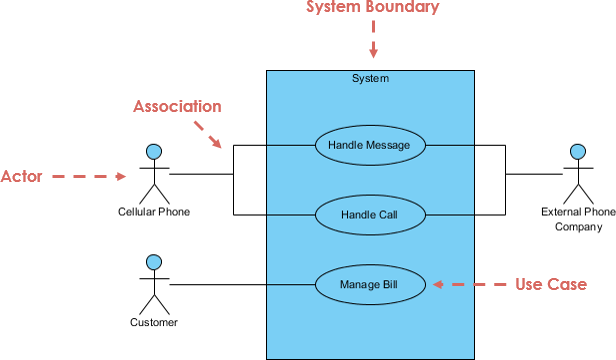
Purpose of Use Case Diagram

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

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

## Use Case Diagram at a Glance

A standard form of use case diagram is defined in the Unified Modeling Language as shown in the Use Case Diagram example below:



|  |  |
| --- | --- |
| **Notation Description** | **Visual Representation** |
| **Actor**   * Someone interacts with use case (system function). * Named by noun. * Actor plays a role in the business * Similar to the concept of user, but a user can play different roles * Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs). | Use Case Diagram Notation - Actor |
| **Use Case**   * System function (process - automated or manual) * Named by verb + Noun (or Noun Phrase). * i.e. Do something * Each Actor must be linked to a use case, while some use cases may not be linked to actors. | Use Case Diagram Notation - Use Case |
| **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. | Use Case Diagram Notation - Communication Link |
| **Boundary of system**   * The system boundary is potentially the entire system as defined in the requirements document. * For large and complex systems, each module may be the system boundary. * For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc. * can form a system boundary for use cases specific to each of these business functions. * The entire system can span all of these modules depicting the overall system boundary |  |

## Structuring Use Case Diagram with Relationships

Use cases share different kinds of relationships. Defining the relationship between two use cases is the decision of the software analysts of the use case diagram. A relationship between two use cases is basically modeling the dependency between the two use cases.

|  |  |
| --- | --- |
| **Use Case Relationship** | **Visual Representation** |
| **Extends**   * Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow. * The stereotype "<<extends>>" identifies as an extend relationship | Use Case Diagram Notation - Extend  Indicates that an **"Invalid Password"** use case may include (subject to specified in the extension) the behaviour specified by base use case **"Login Account"**. |
| **Include**   * When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship. * A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case. * The stereotype "<<include>>" identifies the relationship as an include relationship. | Use Case Diagram Notation - Include  An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow. |
| **Generalization**   * A generalization relationship is a parent-child relationship between use cases. * The child use case is an enhancement of the parent use case. * Generalization is shown as a directed arrow with a triangle arrowhead. | Use Case Diagram Notation - Generalization  The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case. |

## Use Case Example

## 

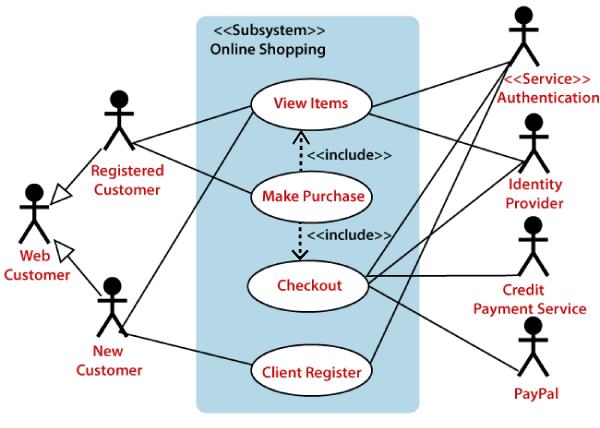
|  |  |
| --- | --- |
| **Use Case Example - Association Link** A Use Case diagram illustrates a set of use cases for a system, i.e. the actors and the relationships between the actors and use cases. | Use Case Diagram Example |
| **Use Case Example - Include Relationship** The include relationship adds 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. | Use Case Diagram Include Example |
| **Use Case Example - 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". | Use Case Diagram Extend Example |
| **Use Case Example - 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 Generalization Example |

## How to Identify Actor

* 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?

* 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?



## Class Diagram

The UML modeling tool lets you model the structure of system by modeling its classes, their attributes and operations in a UML class diagram. UML class diagram is a blueprint of the classes (code level) required to build a software system. Programmers implement a software system with the help of both the class diagram and the class specification.

## Purpose of Class Diagrams

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

## Components of a Class Diagram

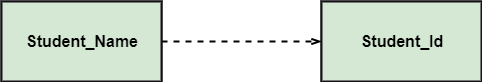
* **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
  1. Capitalize the initial letter of the class name.
  2. Place the class name in the center of the upper section.
  3. A class name must be written in bold format.
  4. The name of the abstract class should be written in italics format.
* **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:
  1. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
  2. The accessibility of an attribute class is illustrated by the visibility factors.
  3. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.
* **Lower Section:** The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



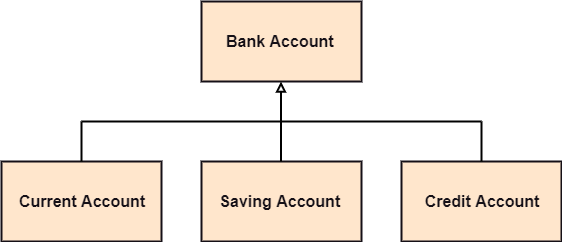
## Relationships

In UML, relationships are of three types:

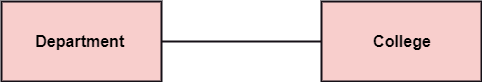
* **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.  
  In the following example, Student\_Name is dependent on the Student\_Id.



* **Generalization:** 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.  
  For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.

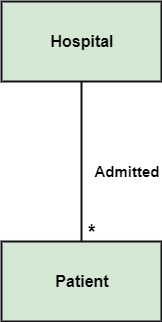


* **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.  
  For example, a department is associated with the college.



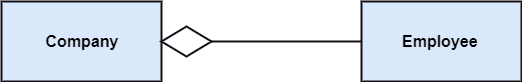
**Multiplicity:** It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

For example, multiple patients are admitted to one hospital.



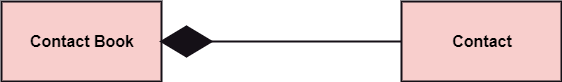
**Aggregation:** An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

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

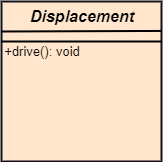


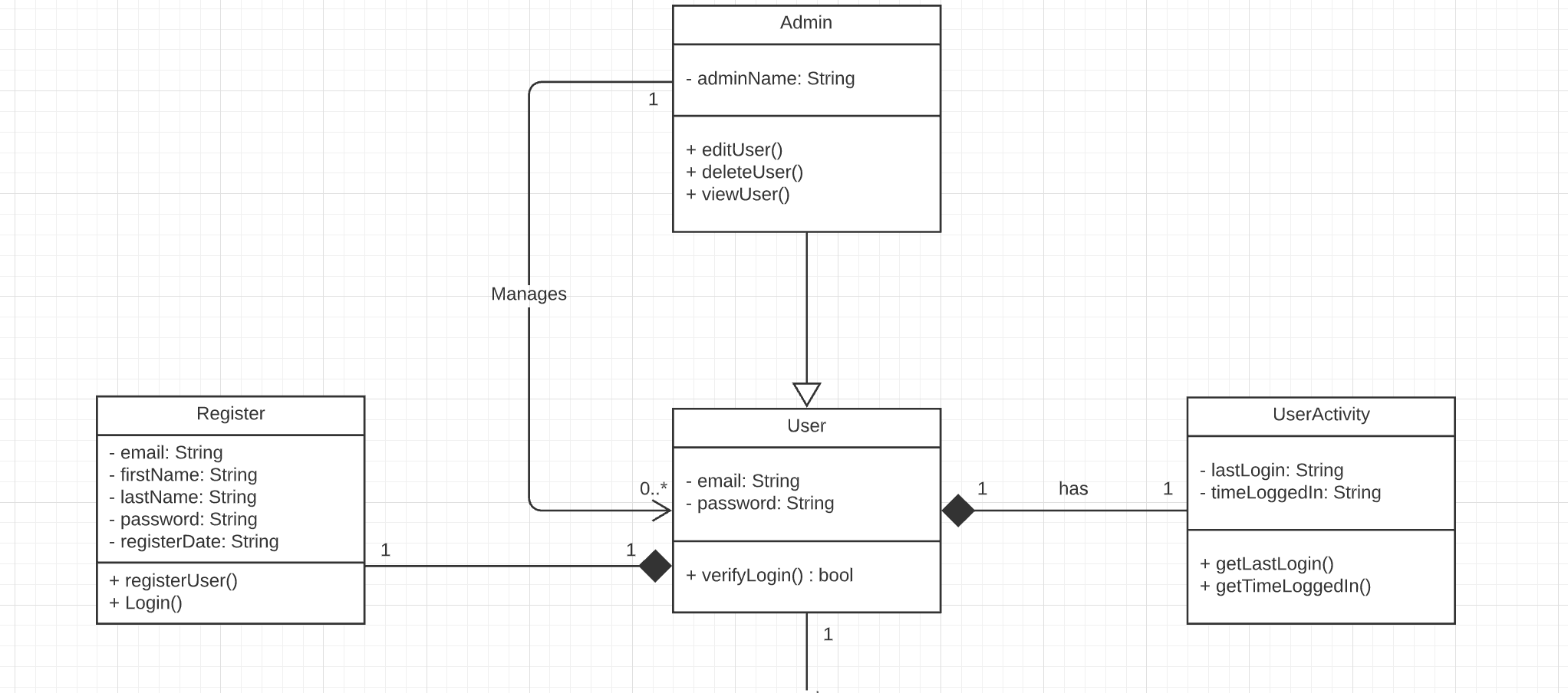
**Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

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



## Abstract Classes





User
Customer
-userld: string
-password: string
-loginStatus: string
1
-customerName: string
-address: string
-email: string

## Sequence Diagram

Visualize the **interactions** between users, systems and sub-systems over time through message passing between objects or roles. If class diagram represents the skeleton of classes by showing their attributes and methods, UML sequence diagram complete the classes by representing the programming logic to be filled in methods' body.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Purpose of a Sequence Diagram  1. To model high-level interaction among active objects within a system. 2. To model interaction among objects inside a collaboration realizing a use case. 3. It either models generic interactions or some certain instances of interaction.  Notations of a Sequence Diagram  |  |  | | --- | --- | | **Lifeline** An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram. | Sequence Diagram | | **Actor** A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa. | Sequence Diagram | | **Activation** It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively. | Sequence Diagram | | **Messages** The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.  Following are types of messages enlisted | * **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation. Sequence Diagram * **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message. Sequence Diagram * **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked. Sequence Diagram * **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls. Sequence Diagram * **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system. Sequence Diagram |   Sequence Diagram Benefits of a Sequence Diagram  1. It explores the real-time application. 2. It depicts the message flow between the different objects. 3. It has easy maintenance. 4. It is easy to generate. 5. Implement both forward and reverse engineering. 6. It can easily update as per the new change in the system.  The drawback of a Sequence Diagram  1. In the case of too many lifelines, the sequence diagram can get more complex. 2. The incorrect result may be produced, if the order of the flow of messages changes. 3. Since each sequence needs distinct notations for its representation, it may make the diagram more complex. 4. The type of sequence is decided by the type of message. |

## Activity Diagram

Use UML activity diagram, a flowchart-based diagram to model the **flow of control**. Partition actions according to the type of participant involved.

## Components of an Activity Diagram

Following are the component of an activity diagram:

|  |  |
| --- | --- |
| **Activities**  The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.  The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node. | UML Activity Diagram |
| **Activity partition /swimlane**  The swimlane is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal. It used to add modularity to the activity diagram. It is not necessary to incorporate swimlane in the activity diagram. But it is used to add more transparency to the activity diagram. | UML Activity Diagram |
| **Forks**  Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision parameters. Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges. It split a single inward flow into multiple parallel flows. | UML Activity Diagram |
| **Join Nodes**  Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge. | UML Activity Diagram |

## Notation of an Activity diagram

Activity diagram constitutes following notations:

**Initial State:** It depicts the initial stage or beginning of the set of actions.

**inal State:** It is the stage where all the control flows and object flows end.

**Decision Box:** It makes sure that the control flow or object flow will follow only one path.

**Action Box:** It represents the set of actions that are to be performed.

