What is Unified Modeling Language (UML)?

UML (http://www.omg.org/spec/UML/), short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article, we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.

The Origin of UML

The goal of UML is to provide a standard notation that can be used by all object-oriented methods and to select and integrate the best elements of precursor notations. UML has been designed for a broad range of applications. Hence, it provides constructs for a broad range of systems and activities (e.g., distributed systems, analysis, system design and deployment).

UML is a notation that resulted from the unification of OMT from

1. **Object Modeling Technique OMT (https://en.wikipedia.org/wiki/Object-modeling_technique)** [**James Rumbaugh (https://en.wikipedia.org/wiki/James_Rumbaugh)** 1991] - was best for analysis and data-intensive information systems.
2. Booch [**Grady Booch (https://en.wikipedia.org/wiki/Grady_Booch)** 1994] - was excellent for design and implementation. Grady Booch had worked extensively with the **Ada (https://en.wikipedia.org/wiki/Ada_(programming_language))** language, and had been a major player in the development of Object Oriented techniques for the language. Although the Booch method was strong, the notation was less well received (lots of cloud shapes dominated his models - not very tidy)
3. OOSE (Object-Oriented Software Engineering [**Ivar Jacobson (https://en.wikipedia.org/wiki/Ivar_Jacobson)** 1992]) - featured a model known as Use Cases. Use Cases are a powerful technique for understanding the behaviour of an entire system (an area where OO has traditionally been weak).

In 1994, Jim Rumbaugh, the creator of OMT, stunned the software world when he left General Electric and joined Grady Booch at Rational Corp. The aim of the partnership was to merge their ideas into a single, unified method (the working title for the method was indeed the "Unified Method").

By 1995, the creator of OOSE, Ivar Jacobson, had also joined Rational, and his ideas (particularly the concept of "Use Cases") were fed into the new Unified Method - now called the Unified Modelling Language1. The team of Rumbaugh, Booch and Jacobson are affectionately known as the "Three Amigos"

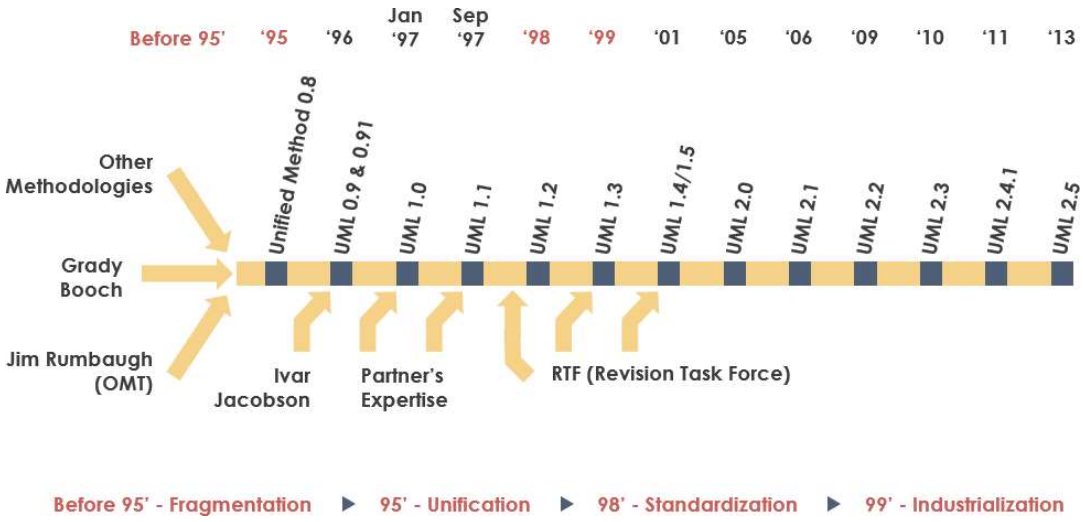
UML has also been influenced by other object-oriented notations:

- Mellor and Shlaer [1998]
- Coad and Yourdon [1995]
- Wirfs-Brock [1990]
- Martin and Odell [1992]

UML also includes new concepts that were not present in other major methods at the time, such as extension mechanisms and a constraint language.

History of UML

1. During 1996, the first Request for Proposal (RFP) issued by the **Object Management Group (OMG) (http://www.omg.org/)** provided the catalyst for these organizations to join forces around producing a joint RFP response.
2. Rational established the UML Partners consortium with several organizations willing to dedicate resources to work toward a strong UML 1.0 definition. Those contributing most to the UML 1.0 definition included:
    - Digital Equipment Corp
    - HP
    - i-Logix
    - IntelliCorp
    - IBM
    - ICON Computing
    - MCI Systemhouse
    - Microsoft
    - Oracle
    - Rational Software
    - TI
    - Unisys
3. This collaboration produced UML 1.0, a modeling language that was well-defined, expressive, powerful, and generally applicable. This was submitted to the OMG in January 1997 as an initial RFP response.1
4. In January 1997 IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam also submitted separate RFP responses to the OMG. These companies joined the UML partners to contribute their ideas, and together the partners produced the revised UML 1.1 response. The focus of the UML 1.1 release was to improve the clarity of the UML 1.0 semantics and to incorporate contributions from the new partners. It was submitted to the OMG for their consideration and adopted in the fall of 1997.1 and enhanced 1.1 to 1.5, and subsequently to UML 2.1 from 01 to 06 (now the UML current version is 2.5)



Why UML

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. The primary goals in the design of the UML summarize by Page-Jones in Fundamental Object-Oriented Design in UML as follows:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

UML - An Overview

Before we begin to look at the theory of the UML, we are going to take a very brief run through some of the major concepts of the UML.

The first thing to notice about the UML is that there are a lot of different diagrams (models) to get used to. The reason for this is that it is possible to look at a system from many different viewpoints. A software development will have many stakeholders playing a part.

For Example:

- Analysts
- Designers
- Coders
- Testers
- QA
- The Customer
- Technical Authors

All of these people are interested in different aspects of the system, and each of them require a different level of detail. For example, a coder needs to understand the design of the system and be able to convert the design to a low level code. By contrast, a technical writer is interested in the behavior of the system as a whole, and needs to understand how the product functions. The UML attempts to provide a language so expressive that all stakeholders can benefit from at least one UML diagram.

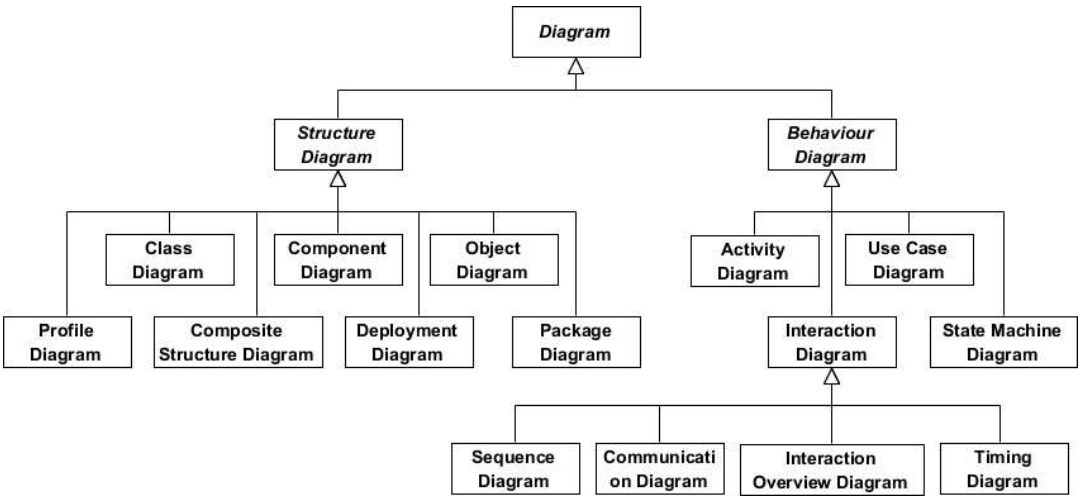
Here's a quick look at each one of these 13 diagrams in as shown in the UML 2 Diagram Structure below:

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:

- Class Diagram
- Component Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram
- Composite Structure Diagram
- Profile Diagram

Behavior diagrams show the **dynamic behavior** of the objects in a system, which can be described as a series of changes to the system over **time**, there are seven types of behavior diagrams as follows:

- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram



What is a Class Diagram?

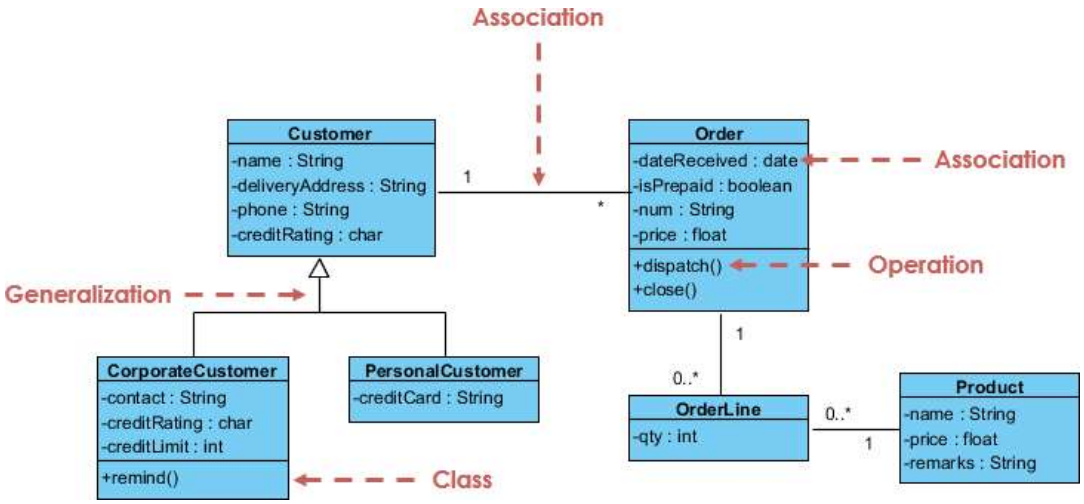
The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.

Relationships

There are three principal kinds of relationships which are important:

1. **Association** - represent relationships between instances of types (a person works for a company, a company has a number of offices.
2. **Inheritance** - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.
3. **Aggregation** - Aggregation, a form of object composition in object-oriented design.

Class Diagram Example

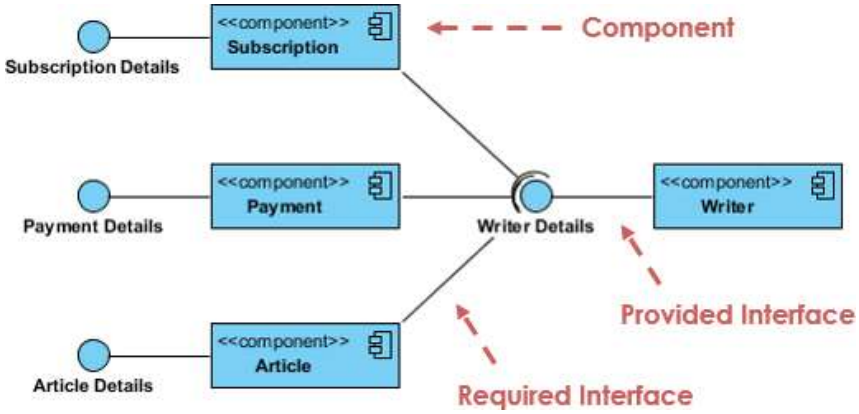


For more details about Class Diagram, please read the article **What is Class Diagram? (/guide/uml-unified-modeling-language/what-is-class-diagram/)**

What is Component Diagram?

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components or software systems. It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components also the source code components.

Component Diagram Example

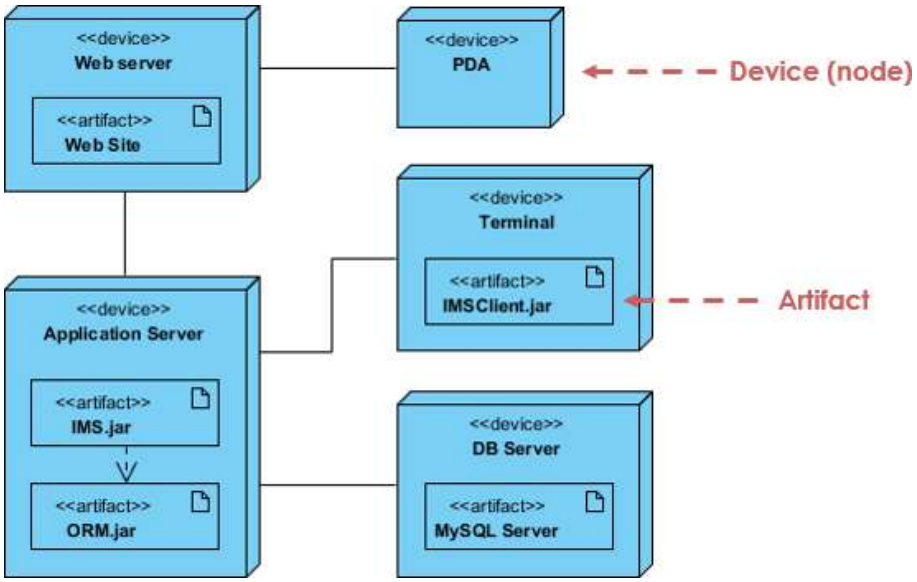


For more details about Component Diagram, please read the article **What is Component Diagram? (/guide/uml-unified-modeling-language/what-is-component-diagram/)**

What is a Deployment Diagram?

The Deployment Diagram helps to model the physical aspect of an Object-Oriented software system. It is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. Artifacts represent concrete elements in the physical world that are the result of a development process. It models the run-time configuration in a static view and visualizes the distribution of artifacts in an application. In most cases, it involves modeling the hardware configurations together with the software components that lived on.

Deployment Diagram Example



For more details about Deployment Diagram, please read the article **What is Deployment Diagram? (/guide/uml-unified-modeling-language/what-is-deployment-diagram/)**

What is an Object Diagram?

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature. The use of object diagrams is fairly limited, namely to show examples of data structure.

Class Diagram vs Object Diagram - An Example

Some people may find it difficult to understand the difference between a UML Class Diagram and a UML Object Diagram as they both comprise of named "rectangle blocks", with attributes in them, and with linkages in between, which make the two UML diagrams look similar. Some people may even think they are the same because in the UML tool they use both the notations for Class Diagram and Object Diagram are put inside the same diagram editor - Class Diagram.

But in fact, Class Diagram and Object Diagram represent two different aspects of a code base. In this article, we will provide you with some ideas about these two UML diagrams, what they are, what are their differences and when to use each of them.

Relationship between Class Diagram and Object Diagram

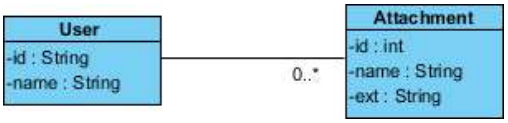
You create "classes" when you are programming. For example, in an online banking system you may create classes like 'User', 'Account', 'Transaction', etc. In a classroom management system you may create classes like 'Teacher', 'Student', 'Assignment', etc. In each class, there are attributes and operations that represent the characteristic and behavior of the class. Class Diagram is a UML diagram where you can visualize those classes, along with their attributes, operations and the inter-relationship.

UML Object Diagram shows how object instances in your system are interacting with each other at a particular state. It also represents the data values of those objects at that state. In other words, a UML Object Diagram can be seen as a representation of how classes (drawn in UML Class Diagram) are utilized at a particular state.

If you are not a fan of those definition stuff, take a look at the following UML diagram examples. I believe that you will understand their differences in seconds.

Class Diagram Example

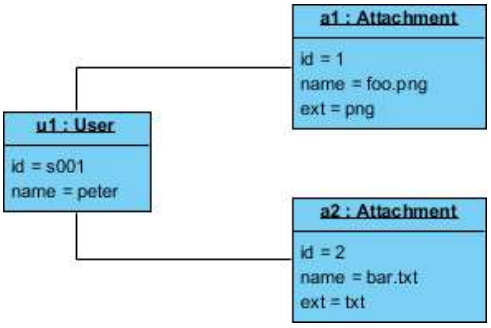
The following Class Diagram example represents two classes - User and Attachment. A user can upload multiple attachment so the two classes are connected with an association, with 0..* as multiplicity on the Attachment side.



Object Diagram Example

The following Object Diagram example shows you how the object instances of User and Attachment class "look like" at the moment Peter (i.e. the user) is trying to upload two attachments. So there are two Instance Specification for the two attachment

objects to be uploaded.

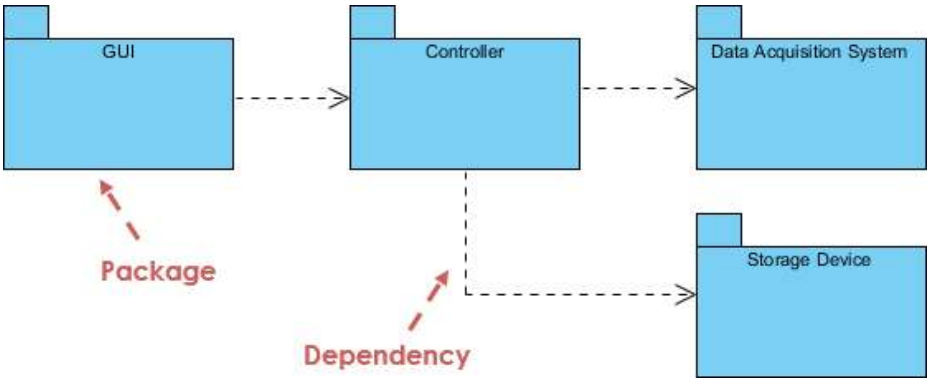


For more details about Object Diagram, please read the article **What is Object Diagram? (/guide/uml-unified-modeling-language/what-is-object-diagram/)**

What is a Package Diagram?

Package diagram is UML structure diagram which shows packages and dependencies between the packages. Model diagrams allow to show different views of a system, for example, as multi-layered (aka multi-tiered) application - multi-layered application model.

Package Diagram Example



For more details about Package Diagram, please read the article **What is Package Diagram? (/guide/uml-unified-modeling-language/what-is-package-diagram/)**

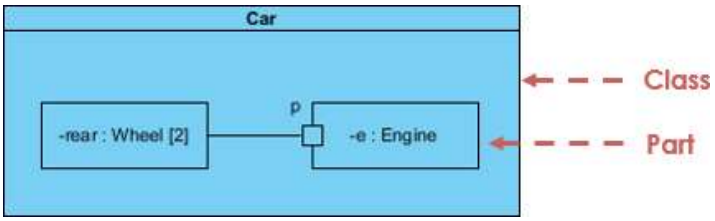
What is a Composite Structure Diagram?

Composite Structure Diagram is one of the new artifacts added to UML 2.0. A composite structure diagram is similar to a class diagram and is a kind of component diagram mainly used in modeling a system at micro point-of-view, but it depicts

individual parts instead of whole classes. It is a type of static structure diagram that shows the internal structure of a class and the collaborations that this structure makes possible.

This diagram can include internal parts, ports through which the parts interact with each other or through which instances of the class interact with the parts and with the outside world, and connectors between parts or ports. A composite structure is a set of interconnected elements that collaborate at runtime to achieve some purpose. Each element has some defined role in the collaboration.

Composite Structure Diagram Example

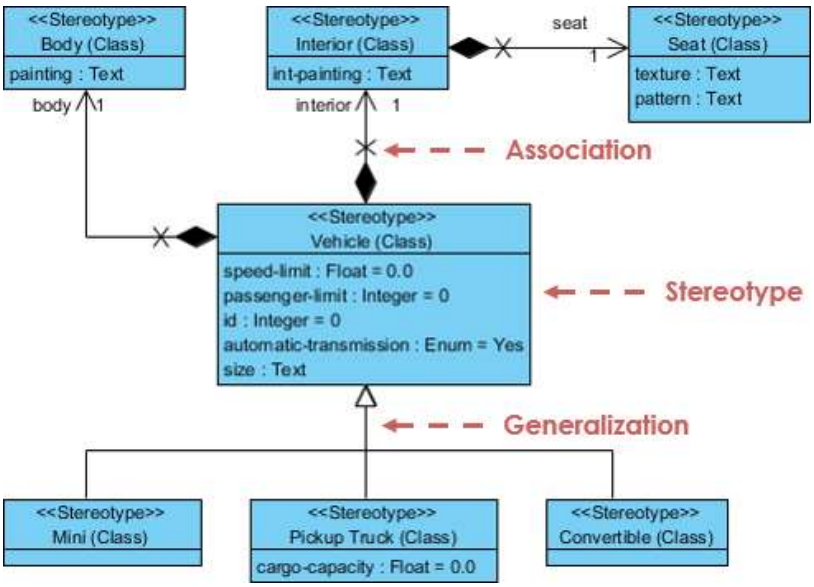


For more details about Composite Structure Diagram, please read the article **What is Composite Structure Diagram? (/guide/uml-unified-modeling-language/what-is-composite-structure-diagram/)**

What is a Profile Diagram?

A profile diagram enables you to create domain and platform specific stereotypes and define the relationships between them. You can create stereotypes by drawing stereotype shapes and relate them with composition or generalization through the resource-centric interface. You can also define and visualize tagged values of stereotypes.

Profile Diagram Example



For more details about Profile Diagram, please read the article **What is Profile Diagram in UML? (/guide/uml-unified-modeling-language/what-is-profile-diagram/)**

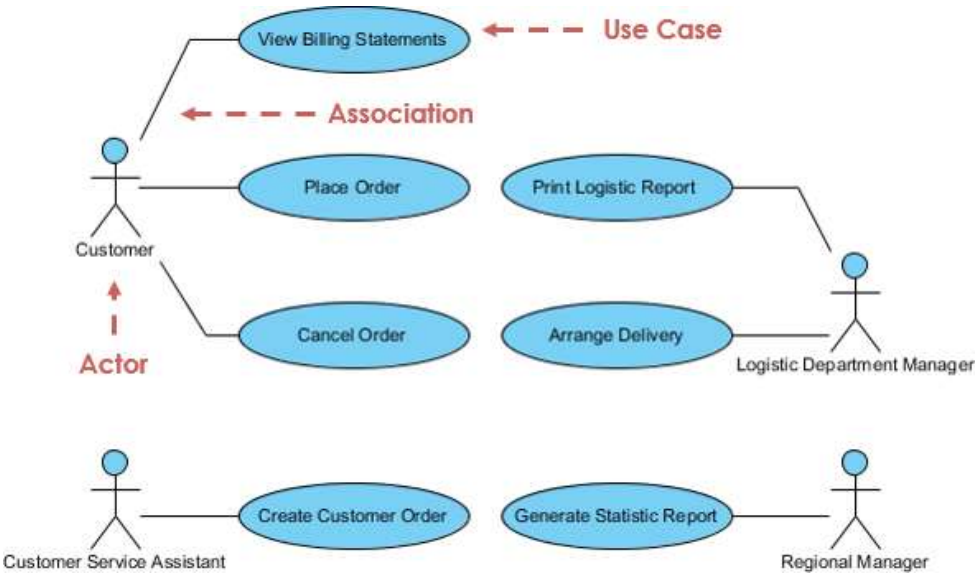
What is a Use Case Diagram?

A use-case model describes a system's functional requirements in terms of use cases. It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Think of a use-case model as a menu, much like the menu you'd find in a restaurant. By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on. By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu, in effect, "models" the restaurant's behavior.

Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members.

Use Case Diagram Example

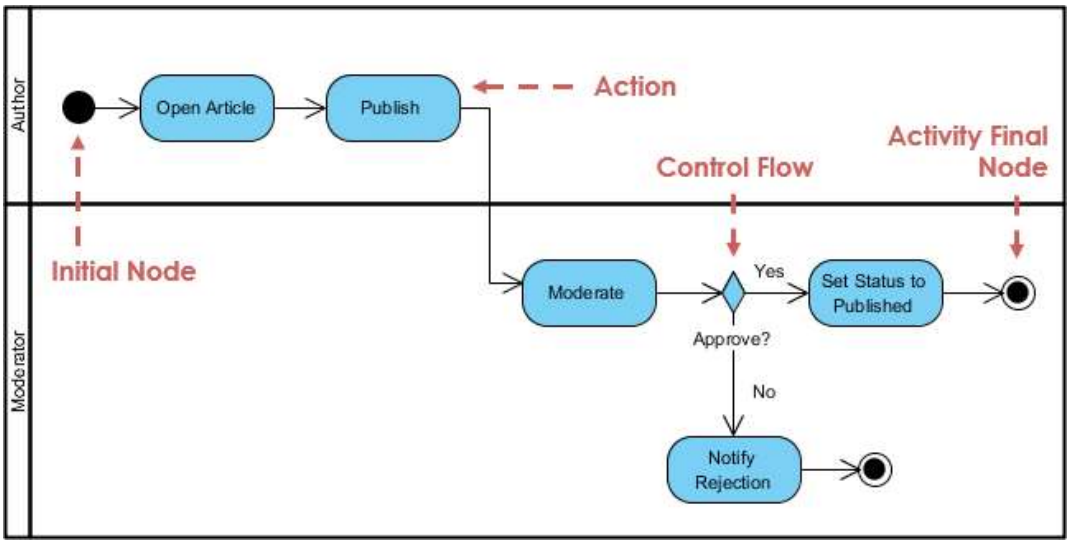


For more details about Use Case Diagram, please read the article **What is Use Case Diagram? (/guide/uml-unified-modeling-language/what-is-use-case-diagram/)**

What is an Activity Diagram?

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. It describes the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).

Activity Diagram Example

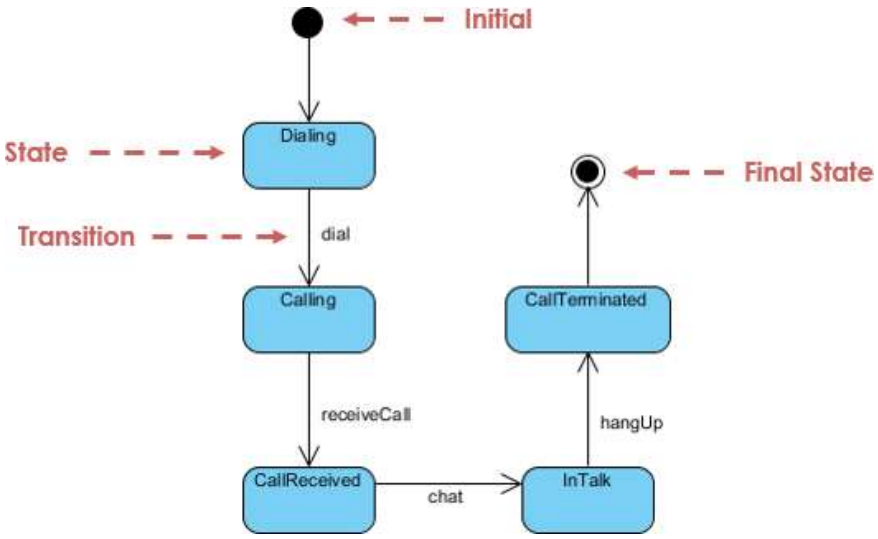


For more details about Activity Diagram, please read the article **What is Activity Diagram? (/guide/uml-unified-modeling-language/what-is-activity-diagram/)**

What is a State Machine Diagram?

A state diagram is a type of diagram used in UML to describe the behavior of systems which is based on the concept of state diagrams by David Harel. State diagrams depict the permitted states and transitions as well as the events that effect these transitions. It helps to visualize the entire lifecycle of objects and thus help to provide a better understanding of state-based systems.

State Machine Diagram Example

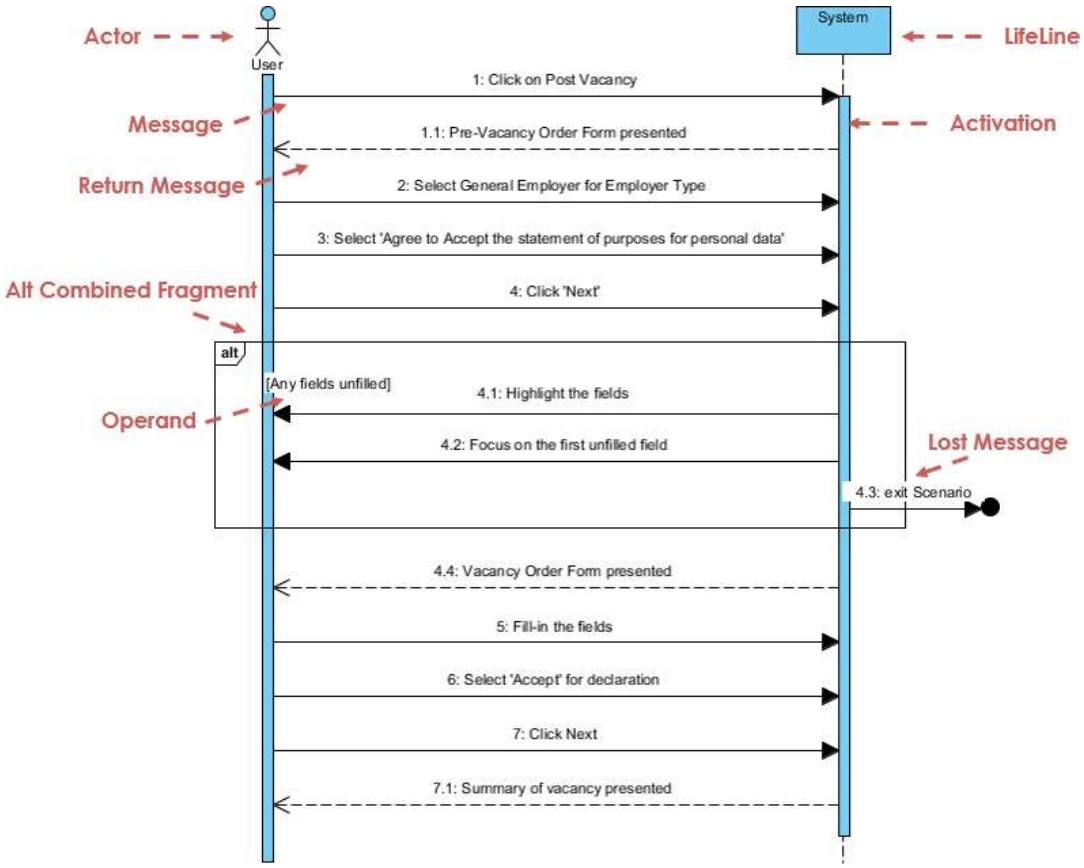


For more details about State Machine Diagram, please read the article **What is State Machine Diagram? (/guide/uml-unified-modeling-language/what-is-state-machine-diagram/)**

What is a Sequence Diagram?

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, some modeling tool such as Visual Paradigm can generate sequence diagram from the flow of events which you have defined in the use case description.

Sequence Diagram Example

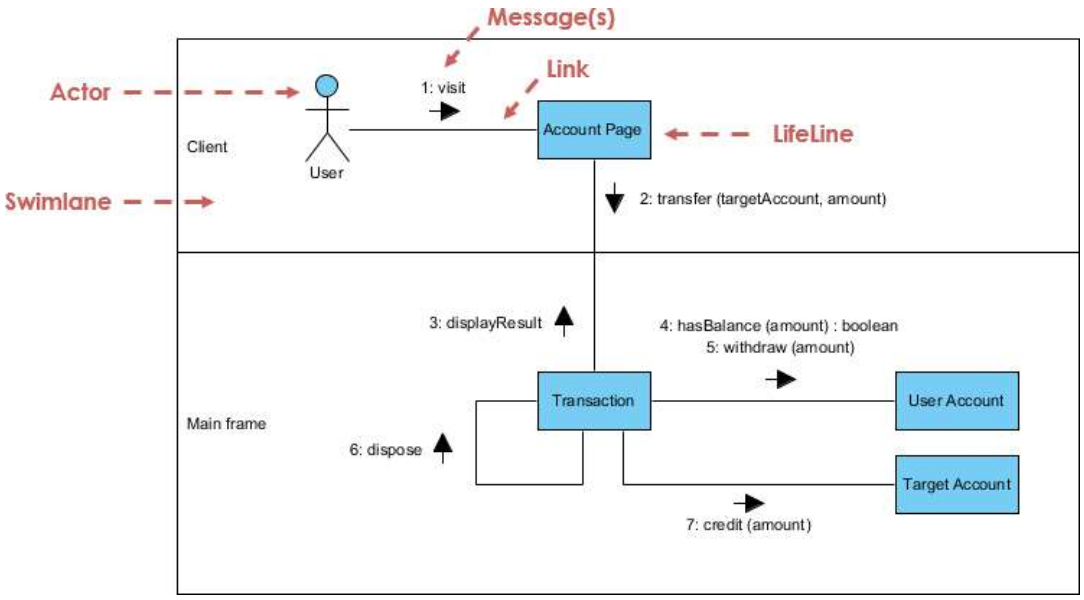


For more details about Sequence Diagram, please read the article **What is Sequence Diagram? (/guide/uml-unified-modeling-language/what-is-sequence-diagram/)**

What is a Communication Diagram?

Similar to Sequence Diagram, the Communication Diagram is also used to model the dynamic behavior of the use case. When compare to Sequence Diagram, the Communication Diagram is more focused on showing the collaboration of objects rather than the time sequence. They are actually semantically equivalent, so some of the modeling tool such as, Visual Paradigm allows you to generate it from one to the other.

Communication Diagram Example

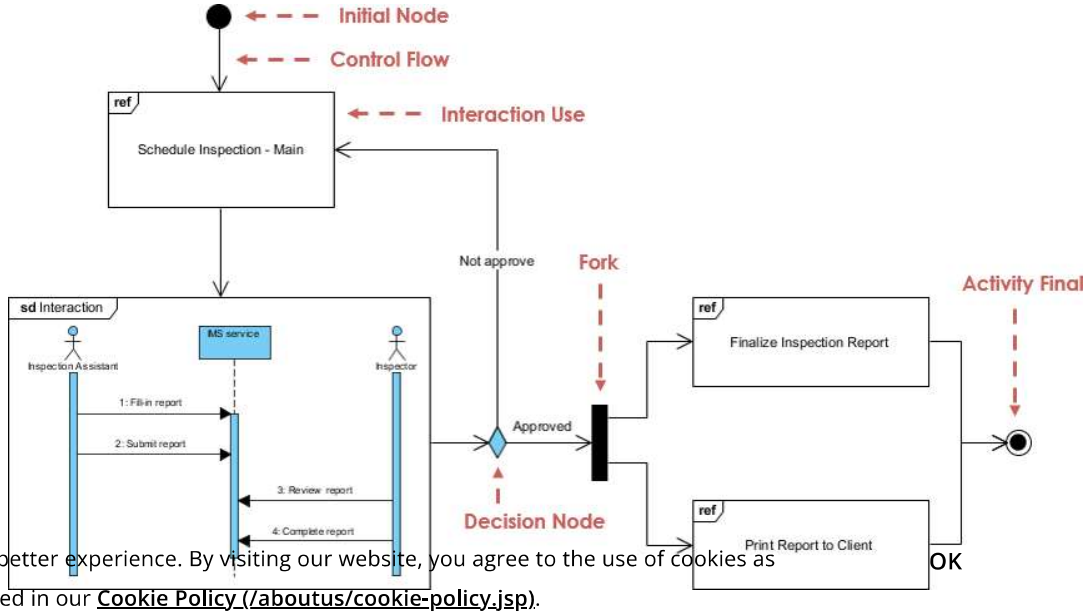


For more details about Communication Diagram, please read the article **What is Communication Diagram? (/guide/uml-unified-modeling-language/what-is-communication-diagram/)**

What is Interaction Overview Diagram?

The Interaction Overview Diagram focuses on the overview of the flow of control of the interactions. It is a variant of the Activity Diagram where the nodes are the interactions or interaction occurrences. The Interaction Overview Diagram describes the interactions where messages and lifelines are hidden. You can link up the "real" diagrams and achieve high degree navigability between diagrams inside the Interaction Overview Diagram.

Interaction Overview Diagram Example

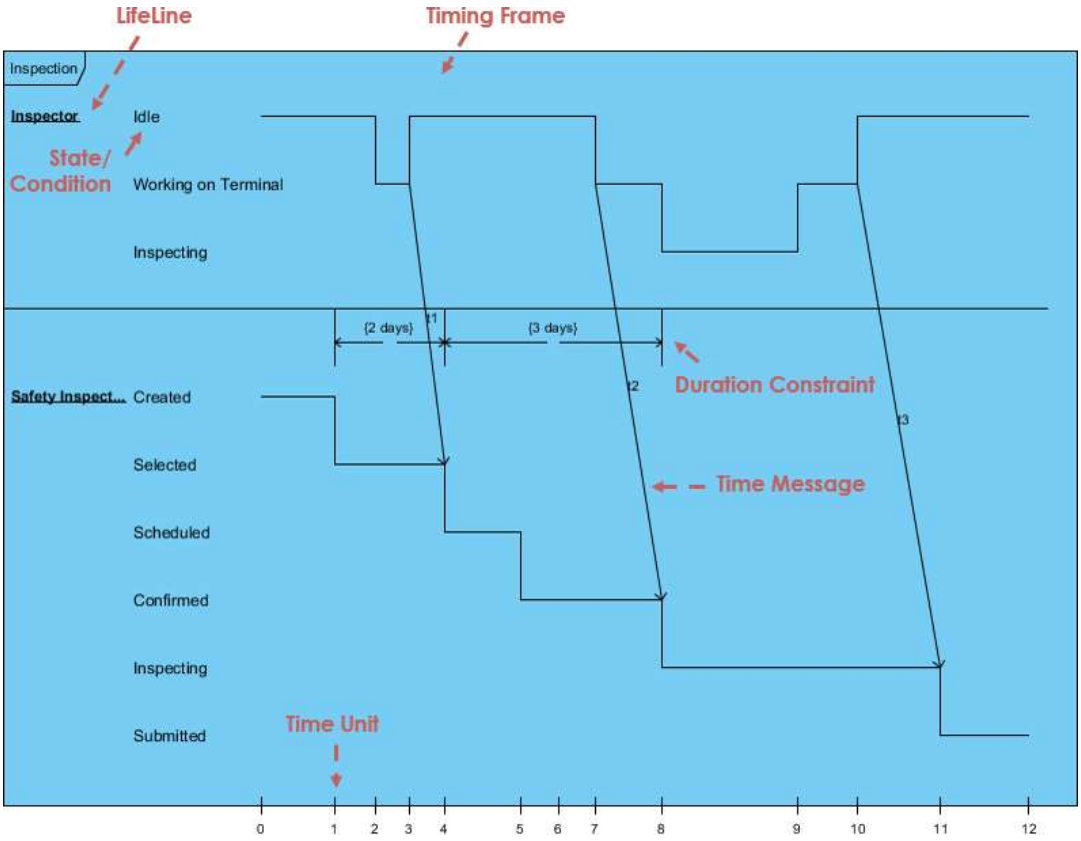


For more details about Interaction Overview Diagram, please read the article **What is Interaction Overview Diagram? (/guide/uml-unified-modeling-language/what-is-interaction-overview-diagram/)**

What is Timing Diagram?

Timing Diagram shows the behavior of the object(s) in a given period of time. Timing diagram is a special form of a sequence diagram. The differences between timing diagram and sequence diagram are the axes are reversed so that the time are increase from left to right and the lifelines are shown in separate compartments arranged vertically.

Timing Diagram Example



For more details about Timing Diagram, please read the article **What is Timing Diagram? (/guide/uml-unified-modeling-language/what-is-timing-diagram/)**

UML Glossary and Terms

- **Abstract Class** - A class that will never be instantiated. An instance of this class will never exist.
- **Actor** - An object or person that initiates events the system is involved with.
- **Activity**: A step or action within an Activity Diagram. Represents an action taken by the system or by an Actor.
- **Activity Diagram**: A glorified flowchart that shows the steps and decisions and parallel operations within a process, such as an algorithm or a business process.
- **Aggregation** - Is a part of another class. Shown with a hollow diamond next to the containing class in diagrams.
- **Artifacts** - Documents describing the output of a step in the design process. The description is graphic, textual, or some combination.
- **Association** - A connection between two elements of a Model. This might represent a member variable in code, or the association between a personnel record and the person it represents, or a relation between two categories of workers, or any similar relationship. By default, both elements in an Association are equal, and are aware of each other through the Association. An Association can also be a Navigable Association, meaning that the source end of the association is aware of the target end, but not vice versa.
- **Association Class**: A Class that represents and adds information to the Association between two other Classes.
- **Attributes** - Characteristics of an object which may be used to reference other objects or save object state information.
- **Base Class**: A Class which defines Attributes and Operations that are inherited by a Subclass via a Generalization relationship.
- **Branch**: A decision point in an Activity Diagram. Multiple Transitions emerge from the Branch, each with a Guard Condition. When control reaches the Branch, exactly one Guard Condition must be true; and control follows the corresponding Transition.

- **Class**: A category of similar Objects, all described by the same Attributes and Operations and all assignment-compatible.
- **Class Diagram** - Shows the system classes and relationships between them.
- **Classifier**: A UML element that has Attributes and Operations. Specifically, Actors, Classes, and Interfaces.
- **Collaboration**: A relation between two Objects in a Communication Diagram, indicating that Messages can pass back and forth between the Objects.
- **Communication Diagram** - A diagram that shows how operations are done while emphasizing the roles of objects.
- **Component**: A deployable unit of code within the system.
- **Component Diagram**: A diagram that shows relations between various Components and Interfaces.
- **Concept** - A noun or abstract idea to be included in a domain model.
- **Construction Phase** - The third phase of the Rational Unified Process during which several iterations of functionality are built into the system under construction. This is where the main work is done.
- **Dependence**: A relationship that indicates one Classifier knows the Attributes and Operations of another Classifier, but isn't directly connected to any instance of the second Classifier.
- **Deployment Diagram**: A diagram that shows relations between various Processors.
- **Domain** -The part of the universe that the system is involved with.
- **Elaboration Phase** - The second phase of the Rational Unified Process that allows for additional project planning including the iterations of the construction phase.
- **Element**: Any item that appears in a Model.
- **Encapsulation** - Data in objects is private.
- **Generalization** - Indicates that one class is a subclass on another class (superclass). A hollow arrow points to the superclass.
- **Event**: In a State Diagram, this represents a signal or event or input that causes the system to take an action or switch States.
- **Final State**: In a State Diagram or an Activity Diagram, this indicates a point at which the diagram completes.
- **Fork**: A point in an Activity Diagram where multiple parallel control threads begin.
- **Generalization**: An inheritance relationship, in which a Subclass inherits and adds to the Attributes and Operations of a Base Class.
- **GoF** - Gang of Four set of design patterns.
- **High Cohesion** - A GRASP evaluative pattern which makes sure the class is not too complex, doing unrelated functions.
- **Low Coupling** - A GRASP evaluative pattern which measures how much one class relies on another class or is connected to another class.

- **Inception Phase** - The first phase of the Rational Unified Process that deals with the original conceptualization and beginning of the project.
- **Inheritance** - Subclasses inherit the attributes or characterics of their parent (superclass) class. These attributes can be overridden in the subclass.
- **Initial State**: In a State Diagram or an Activity Diagram, this indicates the point at which the diagram begins.
- **Instance** - A class is used like a template to create an object. This object is called an instance of the class. Any number of instances of the class may be created.
- **Interface**: A Classifier that defines Attributes and Operations that form a contract for behavior. A provider Class or Component may elect to Realize an Interface (i.e., implement its Attributes and Operations). A client Class or Component may then Depend upon the Interface and thus use the provider without any details of the true Class of the provider.
- **Iteration** - A mini project section during which some small piece of functionality is added to the project. Includes the development loop of analysis, design and coding.
- **Join**: A point in an Activity Diagram where multiple parallel control threads synchronize and rejoin.
- **Member**: An Attribute or an Operation within a Classifier.
- **Merge**: A point in an Activity Diagram where different control paths come together.
- **Message** - A request from one object to another asking the object receiving the message to do something. This is basically a call to a method in the receiving object.
- **Method** - A function or procedure in an object.
- **Model** - The central UML artifact. Consists of various elements arranged in a hierarchy by Packages, with relations between elements as well.
- **Multiplicity** - Shown in a domain model and indicated outside concept boxes, it indicates object quantity relationship to quantiles of other objects.
- **Navigability**: Indicates which end of a relationship is aware of the other end. Relationships can have bidirectional Navigability (each end is aware of the other) or single directional Navigability (one end is aware of the other, but not vice versa).
- **Notation** - Graphical document with rules for creating analysis and design methods.
- **Note**: A text note added to a diagram to explain the diagram in more detail.
- **Object** - Object: In an Activity Diagram, an object that receives information from Activities or provides information to Activities. In a Collaboration Diagram or a Sequence Diagram, an object that participates in the scenario depicted in the diagram. In general: one instance or example of a given Classifier (Actor, Class, or Interface).
- **Package** - A group of UML elements that logically should be grouped together.

- **Package Diagram**: A Class Diagram in which all of the elements are Packages and Dependencies.
- **Pattern** - Solutions used to determine responsibility assignment for objects to interact. It is a name for a successful solution to a well-known common problem.
- **Parameter**: An argument to an Operation.
- **Polymorphism** - Same message, different method. Also used as a pattern.
- **Private**: A Visibility level applied to an Attribute or an Operation, indicating that only code for the Classifier that contains the member can access the member.
- **Processor**: In a Deployment Diagram, this represents a computer or other programmable device where code may be deployed.
- **Protected**: A Visibility level applied to an Attribute or an Operation, indicating that only code for the Classifier that contains the member or for its Subclasses can access the member.
- **Public**: A Visibility level applied to an Attribute or an Operation, indicating that any code can access the member.
- **Reading Direction Arrow** - Indicates the direction of a relationship in a domain model.
- **Realization**: Indicates that a Component or a Class provides a given Interface.
- **Role** - Used in a domain model, it is an optional description about the role of an actor.
- **Sequence Diagram**: A diagram that shows the existence of Objects over time, and the Messages that pass between those Objects over time to carry out some behavior. State chart diagram - A diagram that shows all possible object states.
- **State**: In a State Diagram, this represents one state of a system or subsystem: what it is doing at a point in time, as well as the values of its data.
- **State Diagram**: A diagram that shows States of a system or subsystem, Transitions between States, and the Events that cause the Transitions.
- **Static**: A modifier to an Attribute to indicate that there's only one copy of the Attribute shared among all instances of the Classifier. A modifier to an Operation to indicate that the Operation stands on its own and doesn't operate on one specific instance of the Classifier.
- **Stereotype**: A modifier applied to a Model element indicating something about it which can't normally be expressed in UML. In essence, Stereotypes allow you to define your own "dialect" of UML.
- **Subclass**: A Class which inherits Attributes and Operations that are defined by a Subclass via a Generalization relationship.
- **Swimlane**: An element of an Activity Diagram that indicates what parts of a system or a domain perform particular Activities. All Activities within a Swimlane are the responsibility of the Object, Component, or Actor represented by the Swimlane.
- **Time Boxing** - Each iteration will have a time limit with specific goals.

- **Transition**: In an Activity Diagram, represents a flow of control from one Activity or Branch or Merge or Fork or Join to another. In a State Diagram, represents a change from one State to another.
- **Transition Phase** - The last phase of the Rational Unified Process during which users are trained on using the new system and the system is made available to users.
- **UML** - Unified Modeling Language utilizes text and graphic documents to enhance the analysis and design of software projects by allowing more cohesive relationships between objects.
- **Use Case**: In a Use Case Diagram, represents an action that the system takes in response to some request from an Actor.
- **Use Case Diagram**: A diagram that shows relations between Actors and Use Cases.
- **Visibility**: A modifier to an Attribute or Operation that indicates what code has access to the member. Visibility levels include Public, Protected, and Private.
- **Workflow** - A set of activities that produces some specific result.

Popular UML Books

Listed below are some of the best selling UML books you can read to learn UML.

1. UML Distilled: A Brief Guide to the Standard Object Modeling Language (https://www.amazon.com/UML-Distilled-Standard-Modeling-Language/dp/0321193687)
2. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (https://www.amazon.com/UML-Unified-Process-Practical-Object-Oriented/dp/0321321278)
3. Learning UML 2.0 (https://www.amazon.com/Learning-UML-2-0-Pragmatic-Introduction/dp/0596009828)
4. Building Web Applications with UML (https://www.amazon.com/Building-Web-Applications-UML-2nd/dp/0201730383/)
5. The Unified Modeling Language Reference Manual (https://www.amazon.com/Unified-Modeling-Language-Reference-Manual/dp/020130998X/)
6. The Elements of UML 2.0 Style (https://www.amazon.com/Elements-UMLTM-2-0-Style-ebook/dp/B00E3UR01K/)
7. UML for Java Programmers (https://www.amazon.com/UML-Java%C2%BF-Programmers-Robert-Martin/dp/0131428489/)
8. Schaum's Outline of UML (https://www.amazon.com/Schaums-Outline-UML/dp/0077107411/)
9. The Unified Modeling Language User Guide (https://www.amazon.com/Unified-Modeling-Language-User-Guide/dp/0321267974/)

10. UML 2 Certification Guide: Fundamental and Intermediate Exams (https://www.amazon.com/UML-Certification-Guide-Fundamental-Intermediate/dp/0123735858/)
11. Fundamentals of Object-Oriented Design in UML (https://www.amazon.com/Fundamentals-Object-Oriented-Design-Meilir-Page-Jones/dp/020169946X/)
12. Applying Use Case Driven Object Modeling with UML: An Annotated E-Commerce Example (https://www.amazon.com/Applying-Driven-Object-Modeling-Commerce/dp/0201730391/)
13. Designing Flexible Object Oriented Systems With UML (https://www.amazon.com/Designing-Flexible-Object-Oriented-Systems-UML/dp/1578700981/)
14. Use Case Driven Object Modeling with UML (https://www.amazon.com/Use-Case-Driven-Object-Modeling/dp/1430243058/)
15. Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach (https://www.amazon.com/Systems-Analysis-Design-Version-Object-Oriented/dp/0471348066/)
16. UML 2.0 in a Nutshell (https://www.amazon.com/UML-2-0-Nutshell-Desktop-Reference/dp/0596007957/)
17. Object-Oriented Analysis and Design with Applications (https://www.amazon.com/Object-Oriented-Analysis-Design-Applications-3rd/dp/020189551X/)
18. UML Explained (https://www.amazon.com/UML-Explained-Kendall-Scott/dp/0201721821/)
19. Design Patterns: Elements of Reusable Object-Oriented Software (https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/)
20. The Object Primer: Agile Model-Driven Development with UML 2.0 (https://www.amazon.com/Object-Primer-Agile-Model-Driven-Development-ebook/dp/B00AHTN2U4/)

Related Links

1. Professional UML design tool for visual modeling (/features/uml-tool/)

Turn every software project into a successful one.

Try Visual Paradigm Free (/download/)

(/)

Product

- Features (/features/)
- Editions (/editions/)
- Try Now (/download/)
- Pricing (/shop/)
- Visual Paradigm Online (https://online.visual-paradigm.com/)

Support

- Forums (http://forums.visual-paradigm.com/)
- Request Help (/support/#support-form)
- Customer Service (https://cs.visual-paradigm.com/)

Learn

- Community Circle (https://circle.visual-paradigm.com/)
- Know-how (https://knowhow.visual-paradigm.com/)
- Demo Videos (/features/demo/)
- Tutorials (/tutorials/)
- Documents (/support/documents/)

Company

- About Us (/aboutus/)
- Newsroom (/aboutus/newsreleases/)
- YouTube Channel (https://www.youtube.com/us
- Academic Partnership (/partner/academic/)

Legal (/aboutus/legal.jsp) Privacy statement (/aboutus/privacy.jsp)

(https://twitter.com/visualparadign

(https://www.facebook.com/Visual-Paradigm-822068561487170/)

(https://www.linkedin.com/compar
p

(https://www.pinterest.com/visualp

(https://www.youtube.com/user/VisualParadigm)

(https://www.instagram.com/visualparadigm)