United Modeling Language (UML)

The Unified Modeling Language (UML) is a diagramming language or notation to specify, visualize and document models of Object Oriented software systems. UML is not a development method that means it does not tell you what to do first and what to do next or how to design your system, but it helps you to visualize your design and communicate with others. UML is controlled by the Object Management Group (OMG) and is the industry standard for graphically describing software.

UML is designed for Object Oriented software design and has limited use for other programming paradigms.

UML is composed of many model elements that represent the different parts of a software system. The UML elements are used to create diagrams, which represent a certain part, or a point of view of the system. The following types of diagrams are supported by Umbrello UML Modeler:

- Use Case Diagrams show actors (people or other users of the system), use cases (the scenarios when they use the system), and their relationships
- Class Diagrams show classes and the relationships between them
- Sequence Diagrams show objects and a sequence of method calls they make to other objects.
- Collaboration Diagrams show objects and their relationship, putting emphasis on the objects that participate in the message exchange
- State Diagrams show states, state changes and events in an object or a part of the system
- Activity Diagrams show activities and the changes from one activity to another with the events occurring in some part of the system
- Component Diagrams show the high level programming components (such as KParts or Java Beans).
- Deployment Diagrams show the instances of the components and their relationships.
- Entity Relationship Diagrams show data and the relationships and constraints between the data.

Use case diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how.*

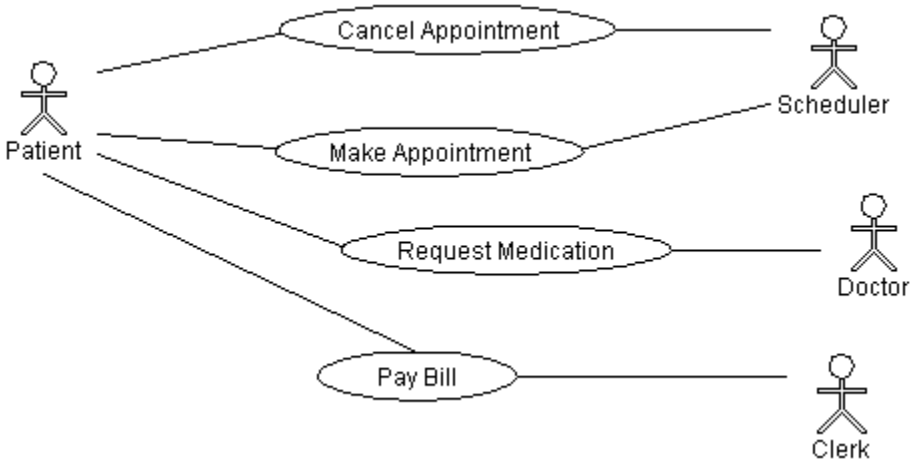
Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system. Here is a scenario for a medical clinic.

> "A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. "

A **use case** is a summary of scenarios for a single task or goal. An **actor** is who or what initiates the events involved in that task. Actors are simply roles that people or objects play. The picture below is a **Make Appointment** use case for the medical clinic. The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).

Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. We've put **Make Appointment** as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.



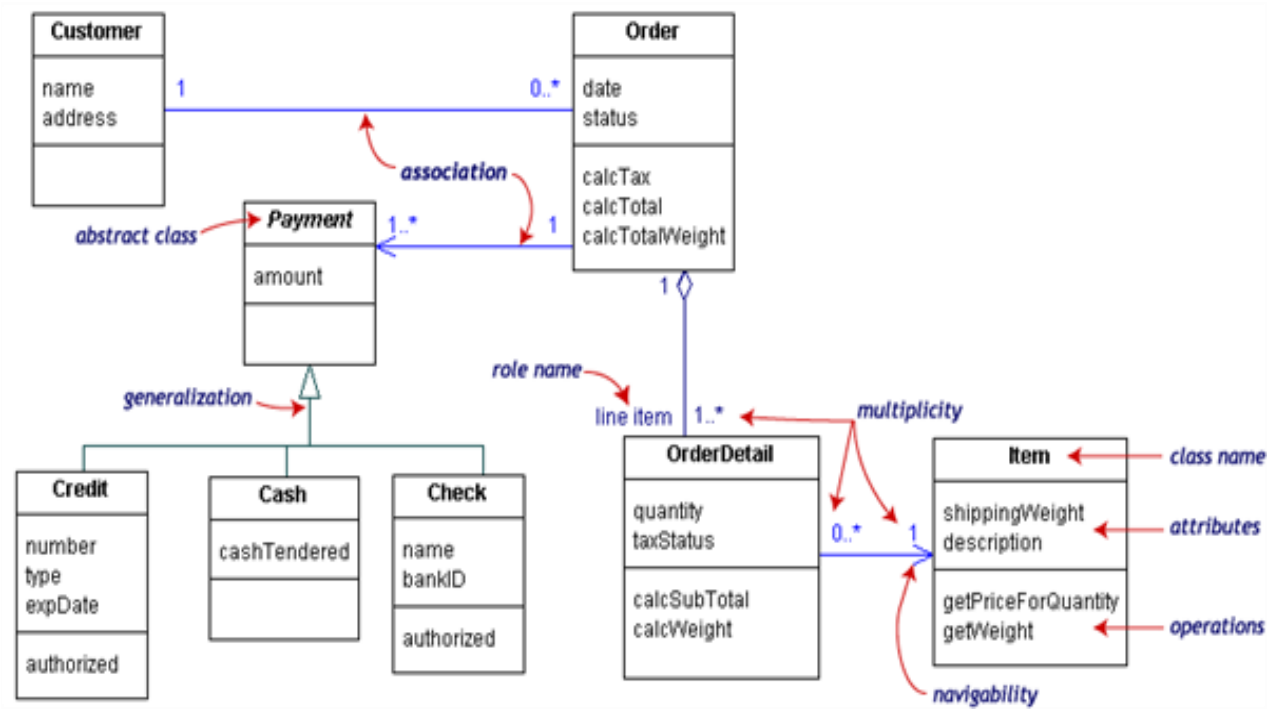
Use case diagrams are helpful in three areas.

- **determining features (requirements)**. New use cases often generate new requirements as the system is analyzed and the design takes shape.
- **communicating with clients**. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- **generating test cases**. The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

Class diagrams

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

The class diagram models a customer order from a retail catalog. The central class is the **Order**. Associated with it is the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.



UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes, such as ***Payment***, are in italics. Relationships between classes are the connecting links.

Our class diagram has three kinds of relationships.

- **Association** -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- **Aggregation** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, **Order** has a collection of **OrderDetails**.

- **Generalization** -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. ***Payment*** is a superclass of **Cash**, **Check**, and **Credit**.

An association has two ends. An end may have a **role name** to clarify the nature of the association. For example, an **OrderDetail** is a line item of each **Order**.

A **navigability** arrow on an association shows which direction the association can be traversed or queried. An **OrderDetail** can be queried about its **Item**, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, **OrderDetail** has an **Item**. Associations with no navigability arrows are bi-directional.

The **multiplicity** of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers. In our example, there can be only one **Customer** for each **Order**, but a **Customer** can have any number of **Orders**.

This table gives the most common multiplicities.

Multiplicities	Meaning
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* ****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

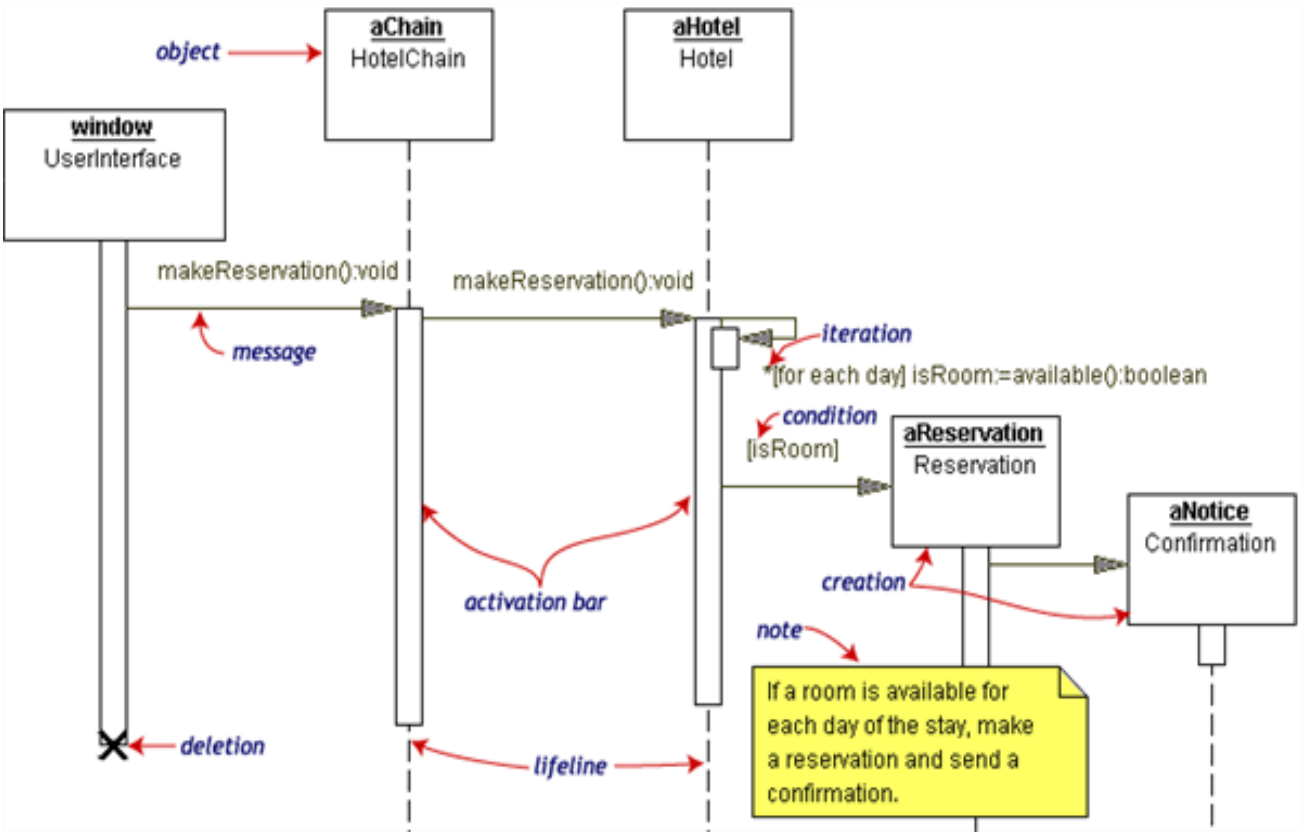
Every class diagram has classes, associations, and multiplicities. Navigability and roles are optional items placed in a diagram to provide clarity.

Sequence diagrams

Class and object diagrams are static model views. **Interaction diagrams** are dynamic. They describe how objects collaborate.

A **sequence diagram** is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is **a Reservation window**.



The **Reservation window** sends a makeReservation() message to a **HotelChain**. The **HotelChain** then sends a makeReservation() message to a **Hotel**. If the**Hotel** has available rooms, then it makes a **Reservation** and a **Confirmation**.

Each vertical dotted line is a **lifeline**, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the **activation bar** of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In our diagram, the **Hotel** issues a **self call** to determine if a room is available. If so, then the **Hotel** creates a **Reservation** and a **Confirmation**. The asterisk on the self call means **iteration** (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, [ ], is a **condition**.

The diagram has a clarifying **note**, which is text inside a dog-eared rectangle. Notes can be put into any kind of UML diagram.

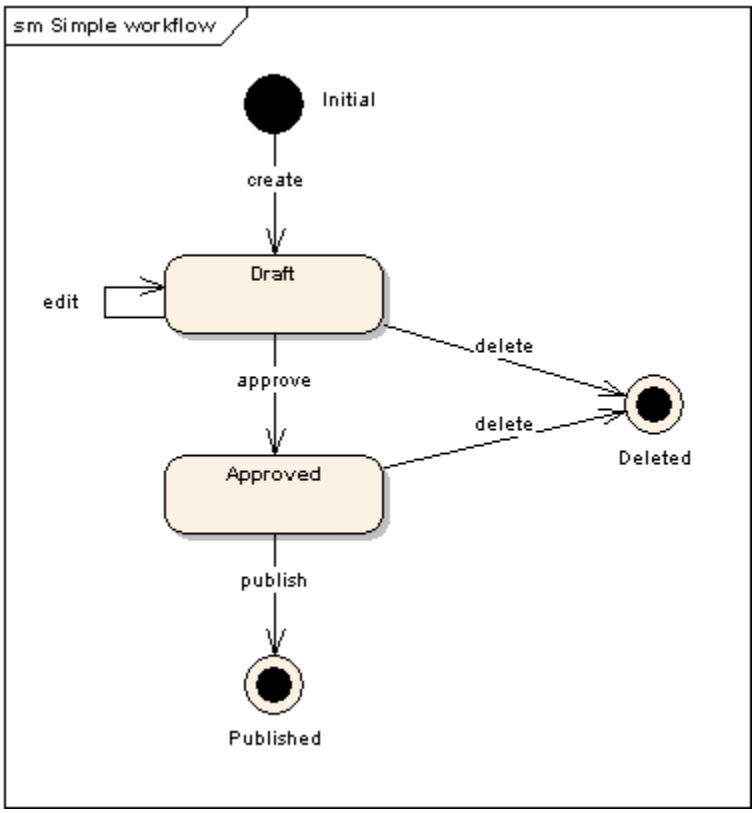
State Diagrams

State diagrams are created during the **analysis** and **design** phase to describe the behavior of nontrivial objects. State diagrams are good for describing the behavior of one object across several use cases and are used to identify object attributes and to refine the behavior description of an object.

A state is a condition in which an object can be at some point during its lifetime, for some finite period of time (Scott, 2001). State diagrams describe all the possible states a particular object can get into and how the objects state changes as a result of external events that reach the object. (Fowler, 2000) In this section, we'll present instead the notation for state diagrams that was first introduced by Harel (Harel, 1987), and then adopted by UML. In a state diagram:

• A state is represented by a rounded rectangle.
• A start state is represented by a solid circle.
• A final state is represented by a solid circle with another open circle around it.
• A transition is a change of an object from one state (the source state) to another (the target state) triggered by events, conditions, or time. Transitions are represented by an arrow connecting two states.

State Diagrams show the different states of an Object during its life and the stimuli that cause the Object to change its state.



Object Oriented Systems Development Life Cycle