A picture is worth a thousand words. That’s why Unified Modeling Language (UML) diagramming was created: to forge a common visual language in the complex world of software development that would also be understandable for business users and anyone who wants to understand a system. Learn the essentials of UML diagrams along with their origins, uses, concepts, types and guidelines on how to draw them using our UML diagram tool

**What is UML?**

The Unified Modeling Language (UML) was created to forge a common, semantically and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally. UML has applications beyond software development, such as process flow in manufacturing.

It is analogous to the blueprints used in other fields, and consists of different types of diagrams. In the aggregate, UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it.

UML is not a programming language but there are tools that can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design.

**UML and its role in object-oriented modeling and design**

There are many problem-solving paradigms or models in Computer Science, which is the study of algorithms and data. There are four problem-solving model categories: imperative, functional, declarative and object-oriented languages (OOP).  In object-oriented languages, algorithms are expressed by defining ‘objects’ and having the objects interact with each other. Those objects are things to be manipulated and they exist in the real world. They can be buildings, widgets on a desktop, or human beings.

Object-oriented languages dominate the programming world because they model real-world objects. UML is a combination of several object-oriented notations: Object-Oriented Design, Object Modeling Technique, and Object-Oriented Software Engineering.

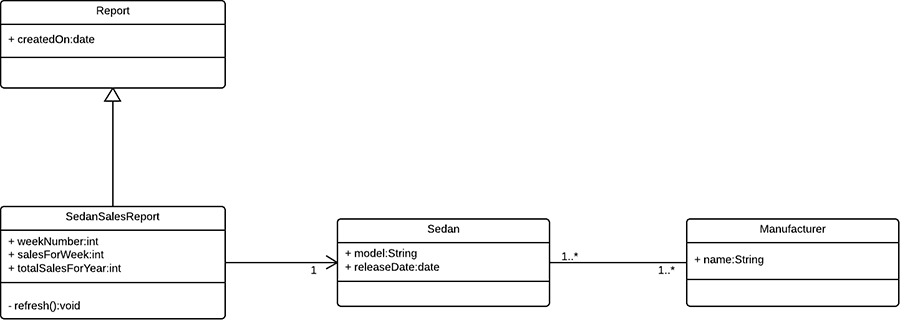
different types of UML diagrams, try one or all of these tutorials to guide you through the process of drawing both structural and behavioral diagrams.

Structural Diagram Tutorial Examples

CLASS DIAGRAMS

Class diagrams represent the static structures of a system, including its classes, attributes, operations, and objects. A class diagram can display computational data or organizational data in the form of implementation classes and logical classes, respectively. There may be overlap between these two groups.

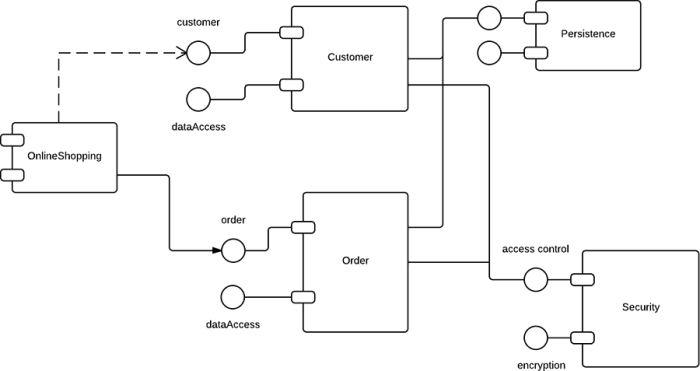
1. Classes are represented with a rectangular shape that is split into thirds. The top section displays the class name, while the middle section contains the class' attributes. The bottom section features the class operations (also known as methods).
2. Add class shapes to your class diagram to model the relationship between those objects. You may need to add subclasses, as well.
3. Use lines to represent association, inheritance, multiplicity, and other relationships between classes and subclasses. Your preferred notation style will inform the notation of these lines.



COMPONENT DIAGRAMS

Component diagrams show how components are combined to form larger components or software systems. These diagrams are meant to model the dependencies of each component in the system. A component is something required to execute a stereotype function. A component stereotype may consist of executables, documents, database tables, files, or library files.

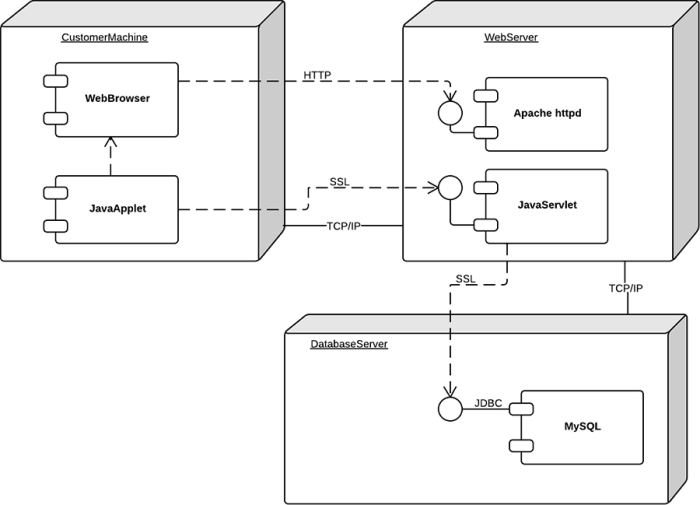
1. Represent a component with a rectangle shape. It should have two small rectangles on the side, or feature an icon with this shape.
2. Add lines between component shapes to represent the relevant relationships.



DEPLOYMENT DIAGRAMS

A deployment diagram models the physical deployment and structure of hardware components. Deployment diagrams demonstrate where and how the components of a system will operate in conjunction with each other.

1. When drawing a deployment diagram, use the same notation that you use for a component diagram.
2. Use a 3-D cube to model a node (which represents a physical machine or virtual machine).
3. Label the node in the same style that is used for sequence diagrams. Add other nodes as needed, then connect with lines.

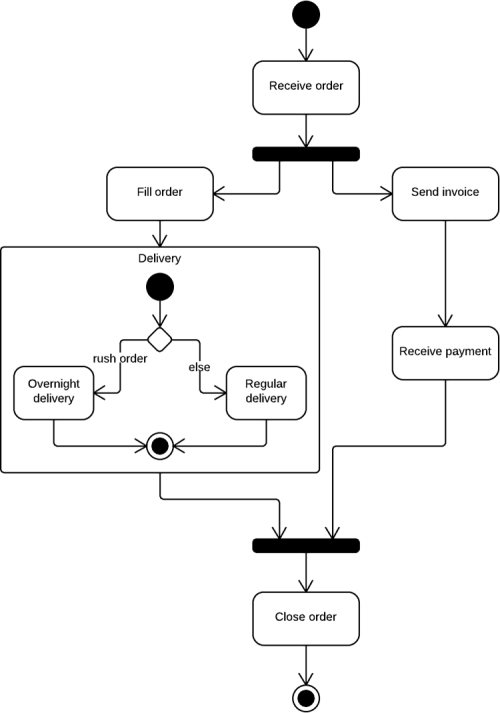


Behavioral Diagram Tutorial Examples

ACTIVITY DIAGRAM

Activity diagrams show the procedural flow of control between class objects, along with organizational processes like business workflows. These diagram are made of specialized shapes, then connected with arrows. The notation set for activity diagrams is similar to those for state diagrams.

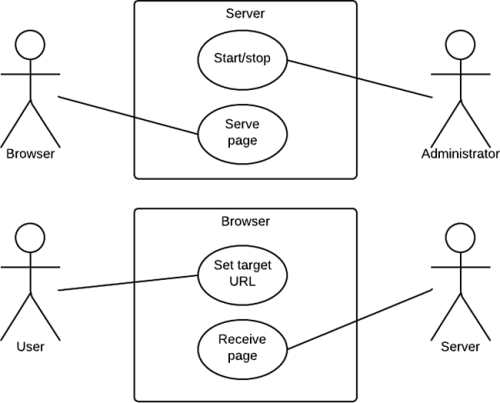
1. Begin your activity diagram with a solid circle.
2. Connect the circle to the first activity, which is modeled with a round-edged rectangle.
3. Now, connect each activity to other activities with lines that demonstrate the stepwise flow of the entire process.
4. You can also try using swimlanes to represent the objects that perform each activity.



USE CASE DIAGRAM

A use case is a list of steps that define interaction between an actor (a human who interacts with the system or an external system) and the system itself. Use case diagrams depict the specifications of a use case and model the functional units of a system. These diagrams help development teams understand the requirements of their system, including the role of human interaction therein and the differences between various use cases. A use case diagram might display all use cases of the system, or just one group of use cases with similar functionality.

1. To begin a use case diagram, add an oval shape to the center of the drawing.
2. Type the name of the use case inside the oval.
3. Represent actors with a stick figure near the oval, then use lines to model relationships between actors and use cases.



SEQUENCE DIAGRAM

Sequence diagrams, also known as event diagrams or event scenarios, illustrate how processes interact with each other by showing calls between different objects in a sequence. These diagrams have two dimensions: vertical and horizontal. The vertical lines show the sequence of messages and calls in chronological order, and the horizontal elements show object instances where the messages are relayed.

1. To create a sequence diagram, write the class instance name and class name in a rectangular box.
2. Draw lines between class instances to represent the sender and receiver of messages.
3. Use solid arrowheads to symbolize synchronous messages, open arrowheads for asynchronous messages, and dashed lines for reply messages.

