**UML**

**Unified Modelling Language** is a standardized modelling language used in the field of software engineering. It's a graphical language that enables software developers and system architects to visualize, specify, design, and document software systems. UML is a versatile tool that provides a common visual representation of a system's architecture, structure, and behaviour.

**Types of UML Diagrams**

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

The two most broad categories that encompass all other types are Behavioral UML diagram and Structural UML diagram. As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components. The different types are broken down as follows:

**Behavioural UML Diagram**

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

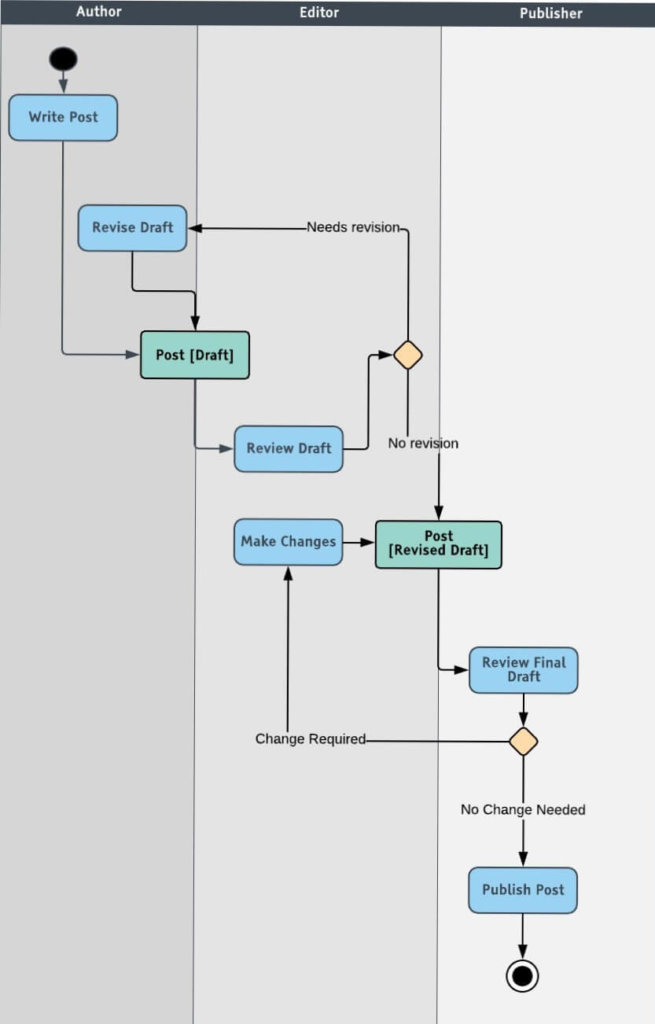
**Structural UML Diagram**

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

The most frequently used ones in software development are: Use Case diagrams, Class diagrams, and Sequence diagrams.

1. **Activity Diagram**

Activity diagrams are probably the most important UML diagrams for doing business process modelling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced by an activity and the relationship between the different activities. All the above are essential in business process modelling.



Diamond shape 🡺 Describe processes 🡺 branching or loops.

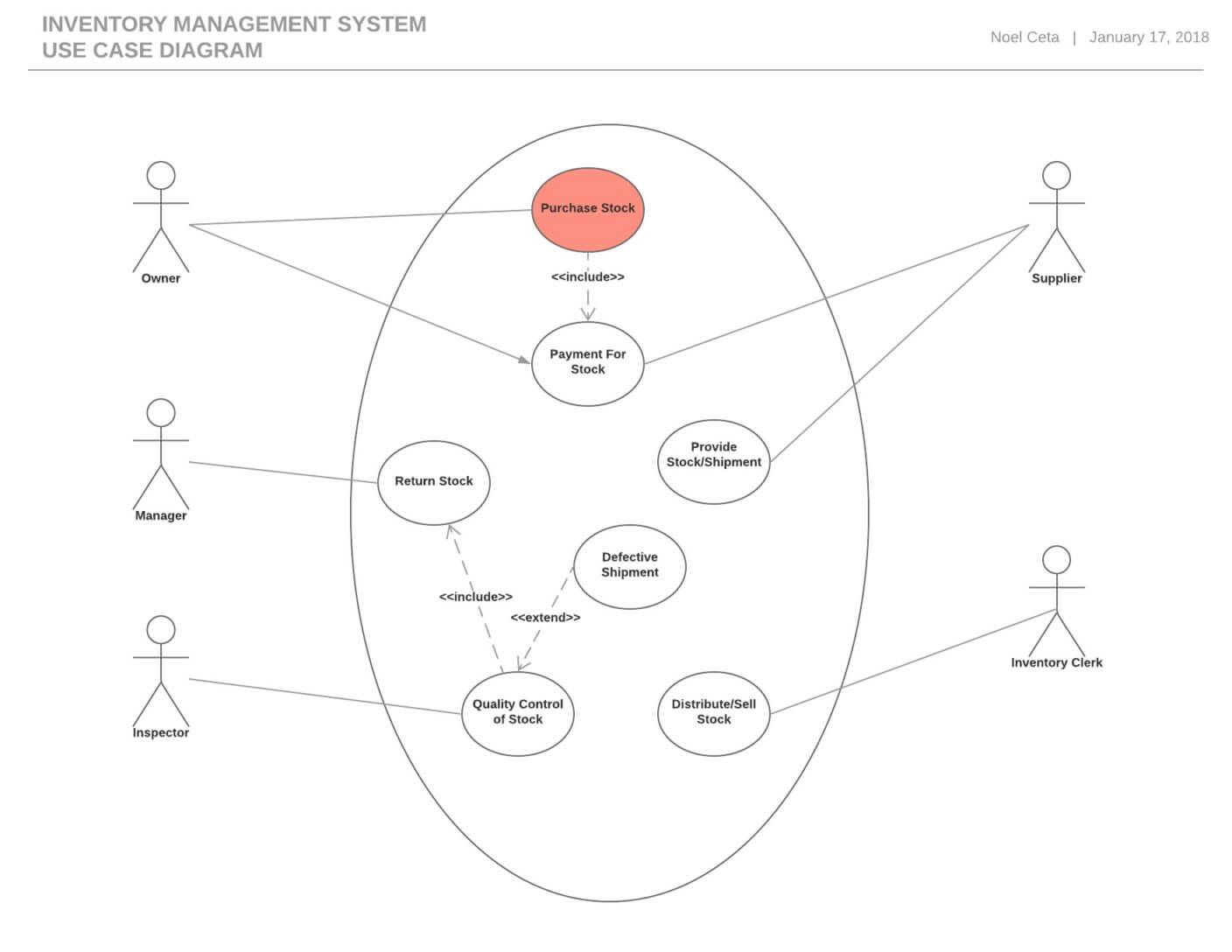
1. **Use Case Diagram**

Components of this UML diagram:

* Functional requirements – represented as use cases; a verb describing an action
* Actors – they interact with the system; an actor can be a human being, an organization or an internal or external application
* Relationships between actors and use cases – represented using straight arrows

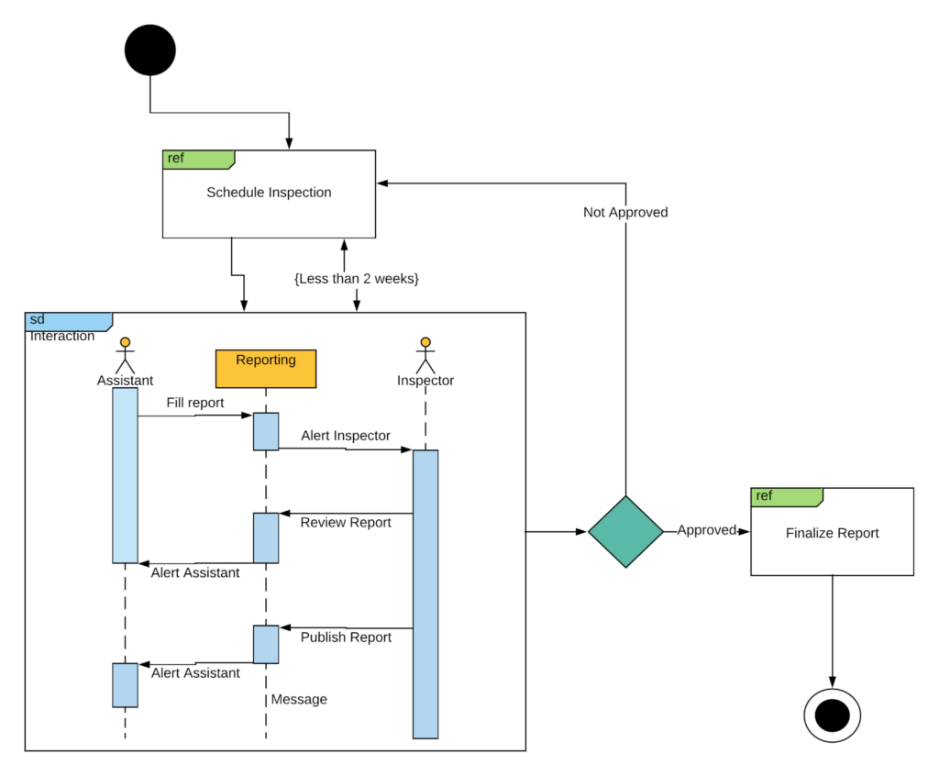
Within the circular containers, we express the actions that the actors perform. Such actions are: purchasing and paying for the stock, checking stock quality, returning the stock or distributing it. As you might have noticed, use case UML diagrams are good for showing dynamic behaviors between actors within a system, by simplifying the view of the system and not reflecting the details of implementation.

Inventory management system.



1. **Interaction Overview Diagram**

Interaction overview diagram is an activity diagram made of different interaction diagrams. It is a mix of activity diagrams with interaction diagrams

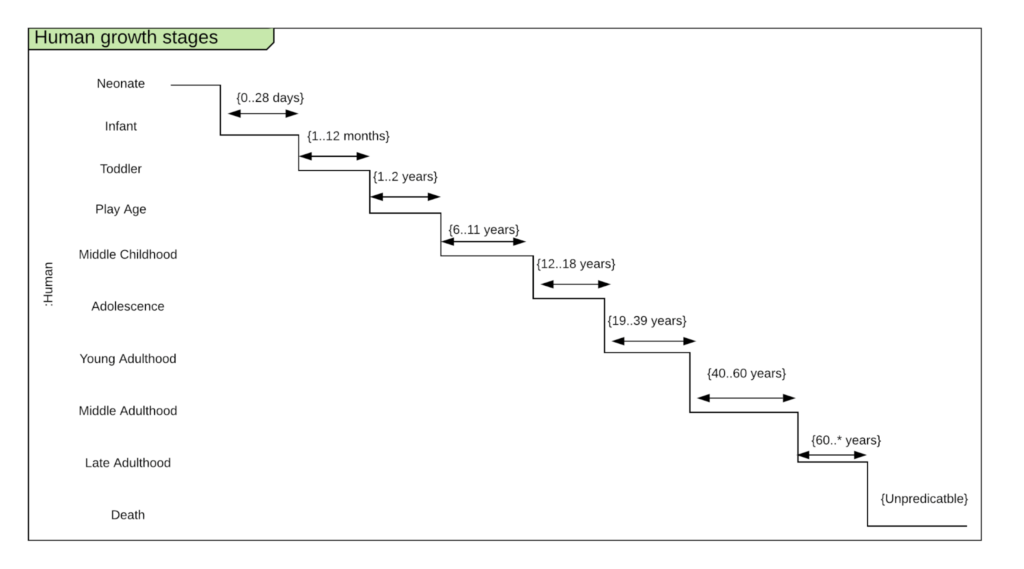


1. **Timing diagram**

Timing UML diagrams are used to represent the relations of objects when the center of attention rests on time.

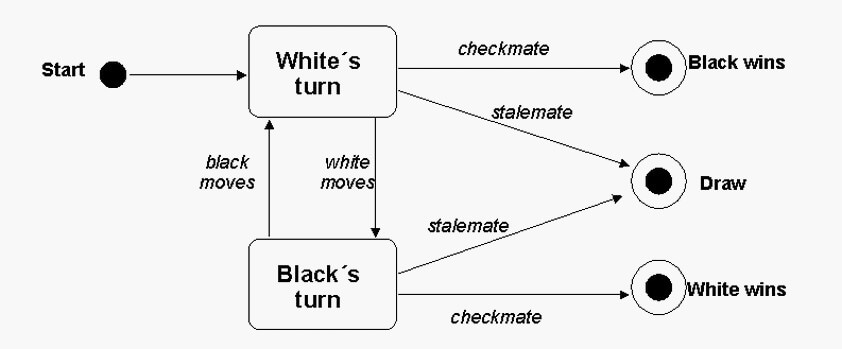
Components of a timing UML diagram

* Lifeline – individual participant
* State timeline – a single lifeline can go through different states within a pipeline
* Duration constraint – a time interval constraint that represents the duration of necessary for a constraint to be fulfilled
* Time constraint – a time interval constraint during which something needs to be fulfilled by the participant
* Destruction occurrence – a message occurrence that destroys the individual participant and depicts the end of that participant’s lifeline



1. **State Machine UML diagram**

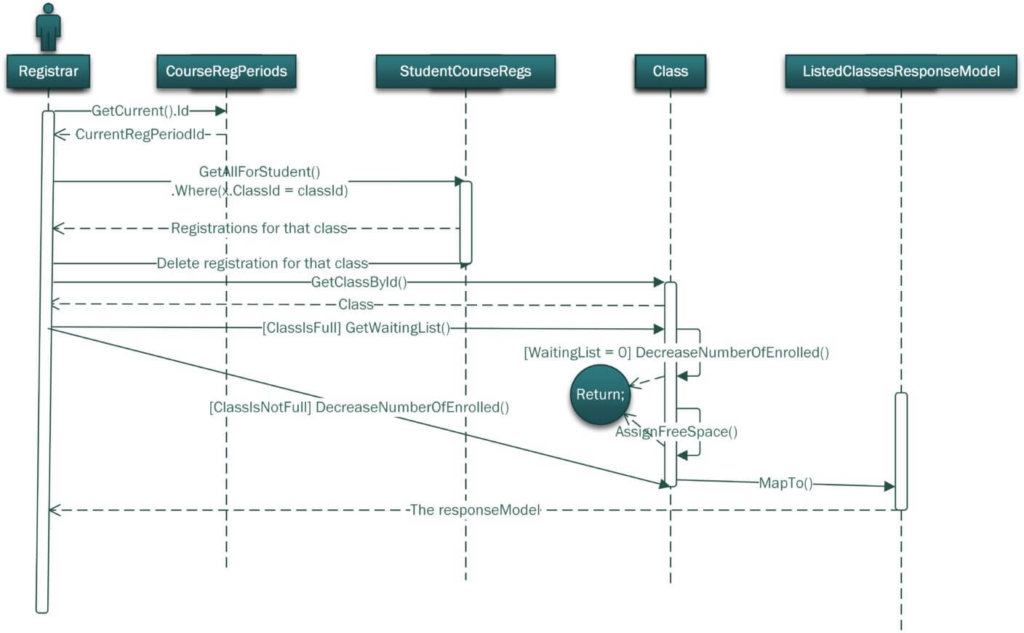
State machine UML diagrams, also referred to as State chart diagrams, are used to describe the different states of a component within a system. It takes the name state machine because the diagram is essentially a machine that describes the several states of an object and how it changes based on internal and external events.



1. **Sequence UML Diagram**

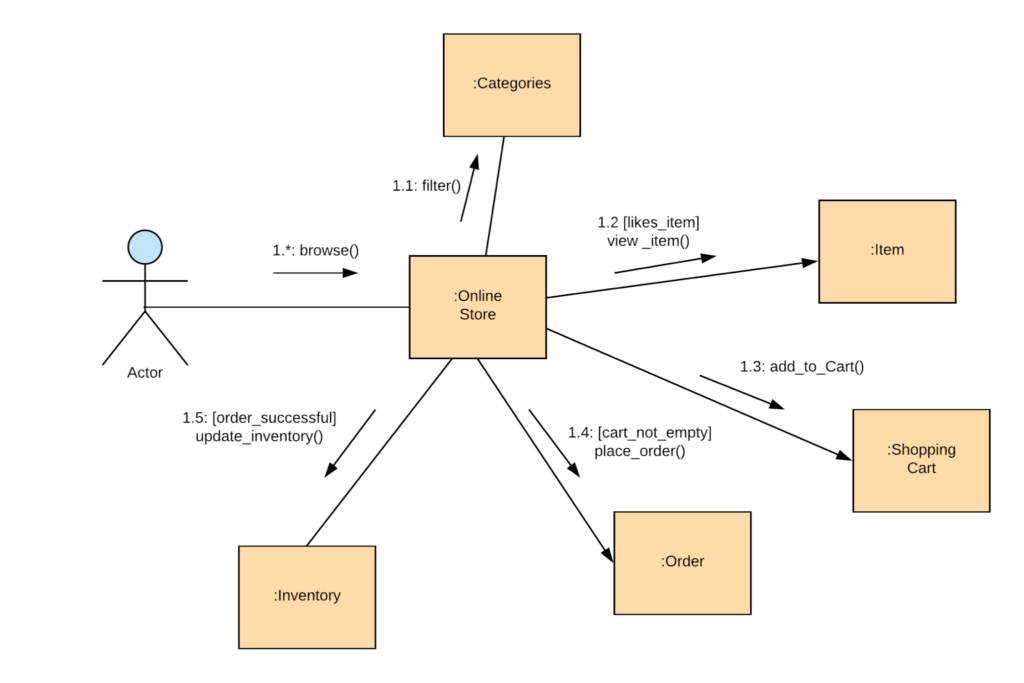
Visually self-explanatory nature. Describe the sequence of messages and interactions that happen between actors and objects.

Course registration system.



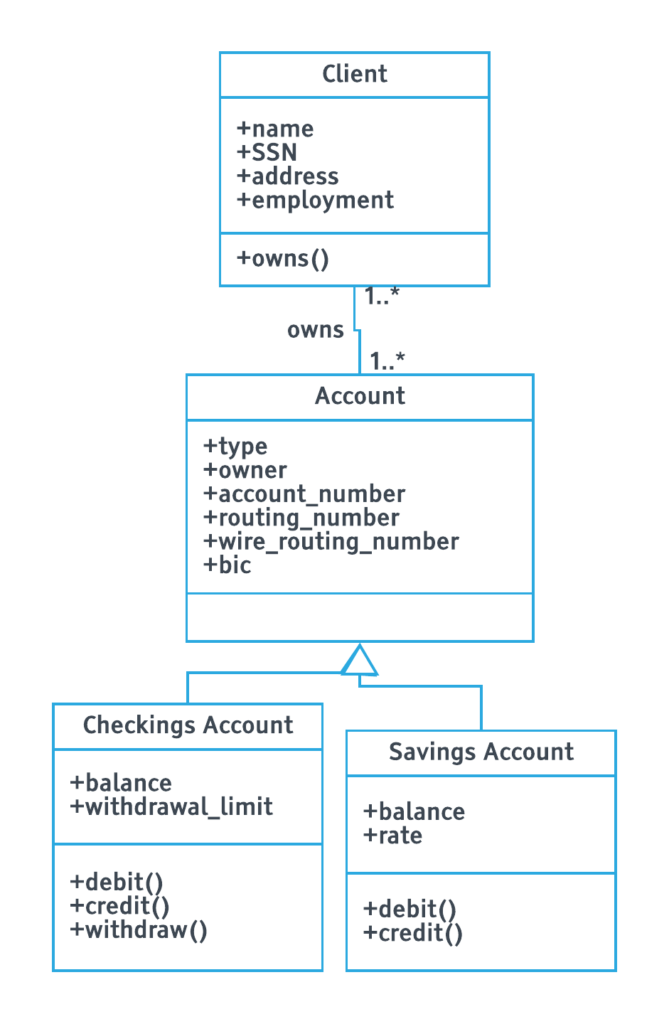
1. **Communication UML diagram**

Communication between objects.



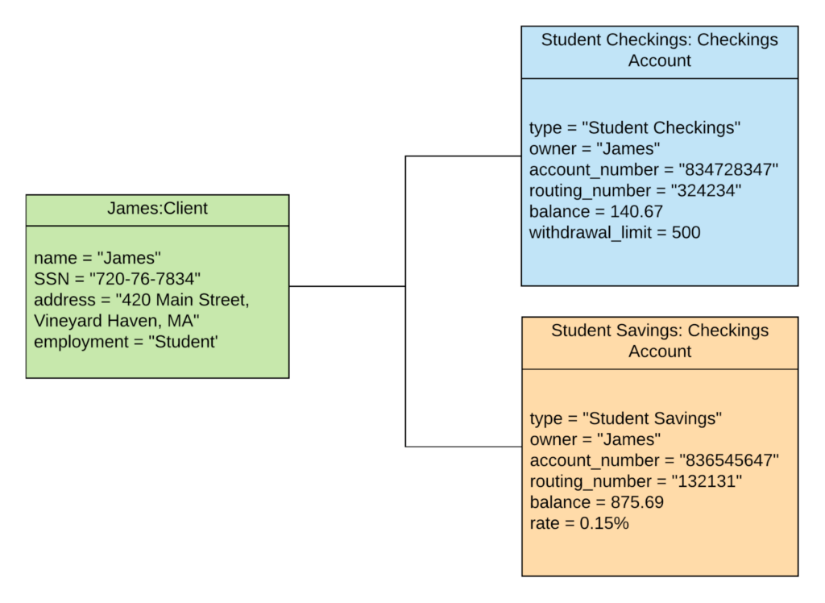
1. **Class Diagram**

Class UML diagram is the most common diagram type for software documentation. Contain classes, alongside with their attributes (also referred to as data fields) and their behaviors (also referred to as member functions). More specifically, each class has 3 fields: the class name at the top, the class attributes right below the name, the class operations/behaviors at the bottom. The relation between different classes (represented by a connecting line), makes up a class diagram.

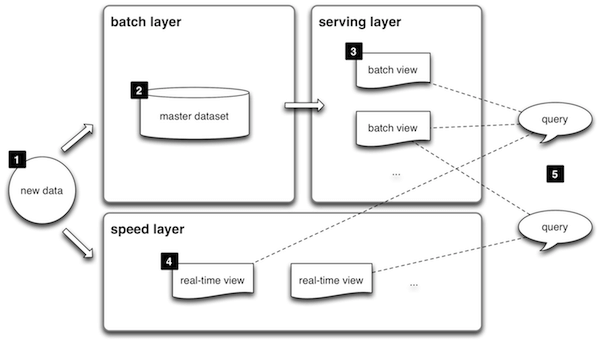


1. **Object Diagram**

Object UML diagrams help software developers check whether the generic abstract structure that they have created (class diagram), represents a viable structure when put into practice, i.e: when the objects of a class are instantiated. Some developers see it as a secondary level of accuracy checking.



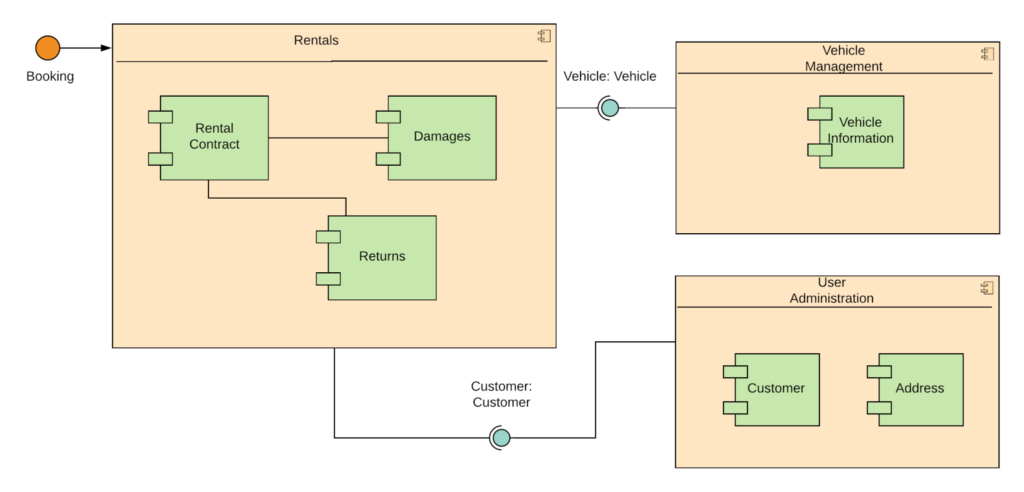
1. **Component Diagram**



The image above shows how a component diagram can help us get a simplified top-level view of a more complex system. The annotations used here are not tailored according to UML standards, however, they are very similar and provide a good visual example.

1. **Composite Structure Diagram**

This type of UML diagram is not commonly used because its function is very specific. It only represents the internal structure of a class and the relations between different class components.



1. **Deployment Diagram**

Deployment diagrams are used to visualize the relation between software and hardware. To be more specific, with deployment diagrams we can construct a physical model of how software components (artifacts) are deployed on hardware components, known as nodes.

A typical simplified deployment diagram for a web application would include:

* Nodes (application server and database server)
* Artifacts (application client and database schema

The nodes host the artifacts. The database schema runs on the database server and the application client runs on the application server.

As the name suggests, the deployment diagram shows exactly where each software component is deployed.

1. **Package Diagram**

The package diagram is like a macro container for deployment UML diagrams that we explained above. Different packages contain nodes and artifacts. They organize the model diagrams and components into groups, the same way a namespace encapsulates different names that are somewhat interrelated.

1. **Profile Diagram**

Profile diagram is not the typical UML diagram type. In fact, it can be regarded more as an extensibility mechanism rather than a diagram type like any other.

**Usage and Applications of UML**:

1. **System Design**: UML is primarily used for designing software systems. It allows developers to create visual models of the system's structure, components, and interactions.
2. **Communication**: UML diagrams serve as a common language that can be easily understood by all stakeholders, including developers, project managers, and clients. They facilitate effective communication and collaboration.
3. **Documentation**: UML diagrams act as documentation for a software system. They provide a clear and structured way to capture the system's design and functionality.
4. **Code Generation**: UML tools can automatically generate code from UML diagrams, making the development process more efficient and reducing the chances of errors.
5. **Testing and Validation**: UML can be used to model the behavior of a system, helping in the creation of test cases and validation of the system's functionality.
6. **Reverse Engineering**: UML can be used to reverse-engineer existing code to create visual representations of a system for better understanding and maintenance.

=========================================

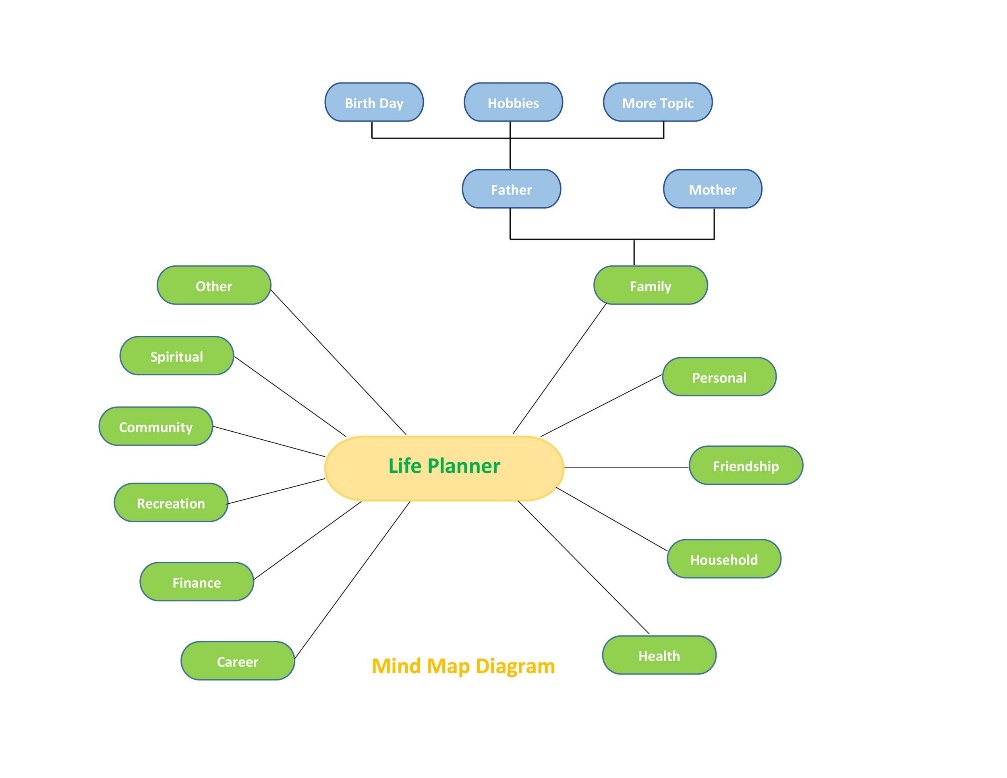
**Mental Maps**:

Mental maps, in the context of software development, refer to the cognitive models that developers and architects create in their minds to understand how a software system works. These mental maps are essentially a mental representation of the system's architecture, components, and functionality. They are crucial for effective problem-solving, decision-making, and communication within a development team.

**Mind maps** provide a structured way to capture and organize ideas and information. They help users to understand concepts by breaking them down into their component parts. The technique is used to develop new ideas, or to break down and better understand existing information.

**Using Mental Maps for Software Architecture and Components**:

1. **System Understanding**: Developers use mental maps to grasp the overall architecture of a system. This includes the main components, their relationships, and the flow of data and control.
2. **Troubleshooting**: When issues arise, developers rely on their mental maps to pinpoint potential problem areas. They use these maps to understand the flow of code and data and trace the issue to its source.
3. **Communication**: Mental maps help developers communicate with team members. When discussing software design or problems, they can use these maps to illustrate their understanding and ideas.
4. **Refactoring and Enhancement**: When making changes to an existing system, developers consult their mental maps to ensure that modifications don't disrupt the existing architecture and functionality.
5. **Planning and Decision-Making**: When planning new features or making architectural decisions, developers use their mental maps to assess the impact on the entire system.



While mental maps are valuable, they can also be limited by an individual's understanding. UML complements mental maps by providing a standardized way to create visual representations that can be shared and understood by the entire development team, improving communication and reducing misunderstandings. UML acts as a bridge between individual mental maps, ensuring that everyone is on the same page when it comes to the system's architecture and functionality.