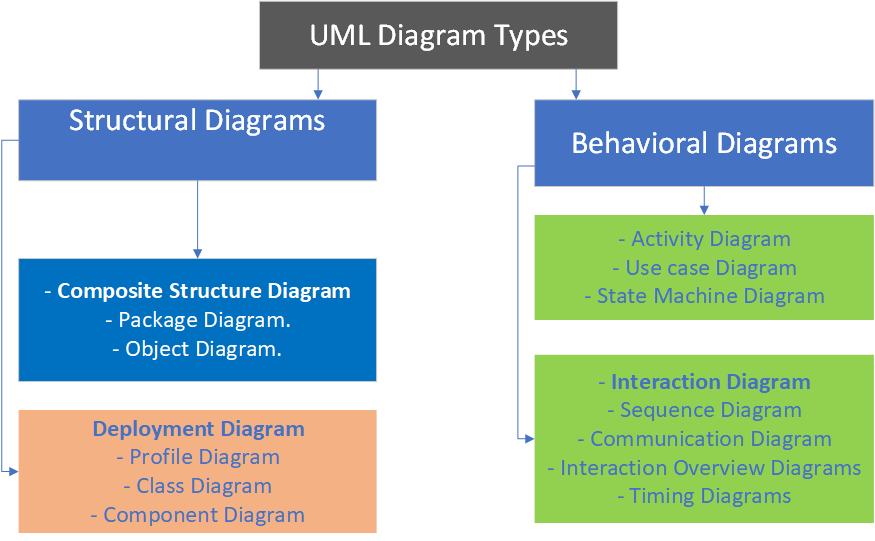
UML DIAGRAMS

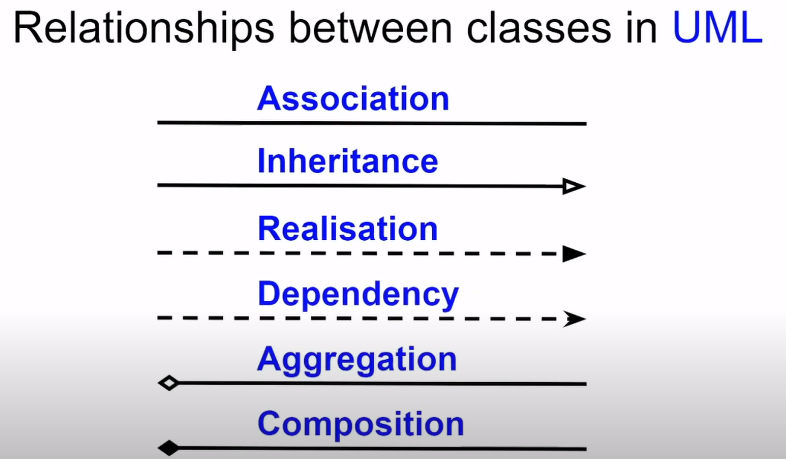
Unified Modelling Language helps in design, visualize and document software and business systems. It helps teams communicate, explore potential designs and validate software architecture designs.

Structure Diagrams are of 7 types: Composite Structure Diagrams, Package Diagrams, Object Diagrams, Deployment Diagrams, Class Diagrams and Component Diagrams.

Behavioral Diagrams include: Activity diagram, Use-case Diagrams, State Machine Diagrams, Interaction Diagrams, Sequence diagrams, Communication diagrams, Interaction overview diagrams, and Timing Diagrams.

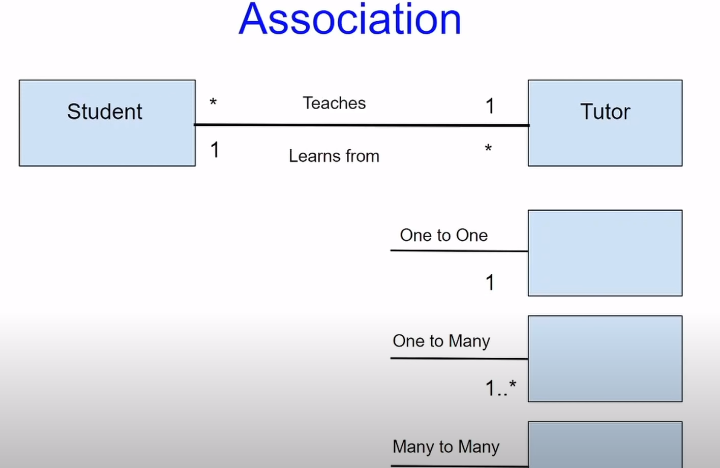


# RELATIONSHIPS BETWEEN UML DIAGRAMS



*Figure showing the 6 common relationship notation types between UML classes.*

1. Association this is the type of relationship that occurs between classes. Its denoted with a straight line connecting the classes which shows the relationship between instances, types and objects.

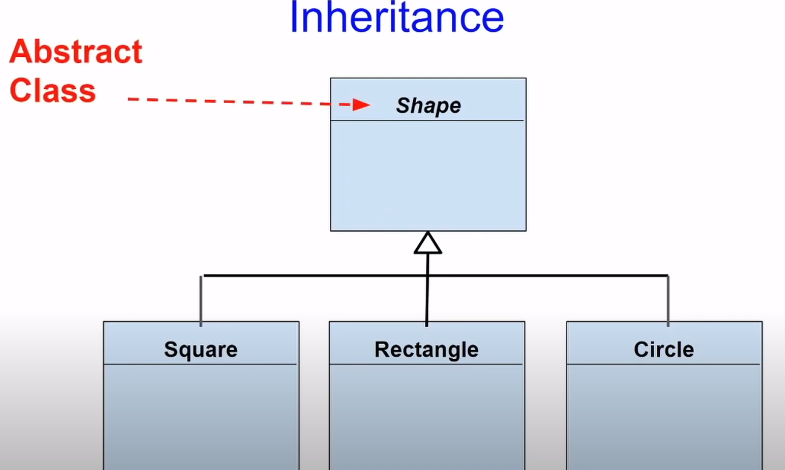


*Figure showing the relationship between two classes. The \* and 1 denotes the multiplicity of relationships*

The 1 and \* relationship shows the multiple association between the Student and Tutor classes. With the example it means that Many students can learn from 1 tutor and 1 student can learn from many tutors.

1. **Inheritance.**

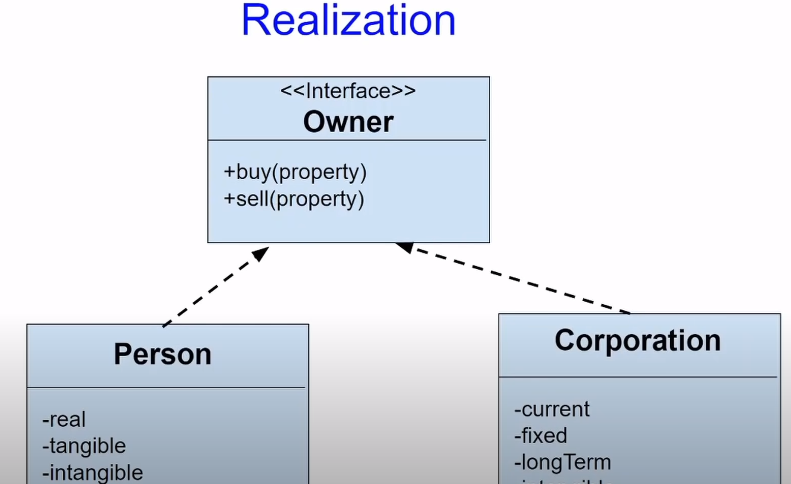
This is straight arrow that denotes the child-parent relationship between objects where the arrow points to the parent. By default, the parent class name should be in italic if it’s an abstract class.



*Figure showing an inheritance relationship.*

1. **Realization.**

This is the relationship between an interface and object implementing this interface.



*Figure showing a realization relationship example.*

1. **Dependency.**

This is special type of association in which one class depends on another class. Therefore, the value of one class will change if the dependent class changes its value. It’s a dotted line with an arrow which points to the dependent.

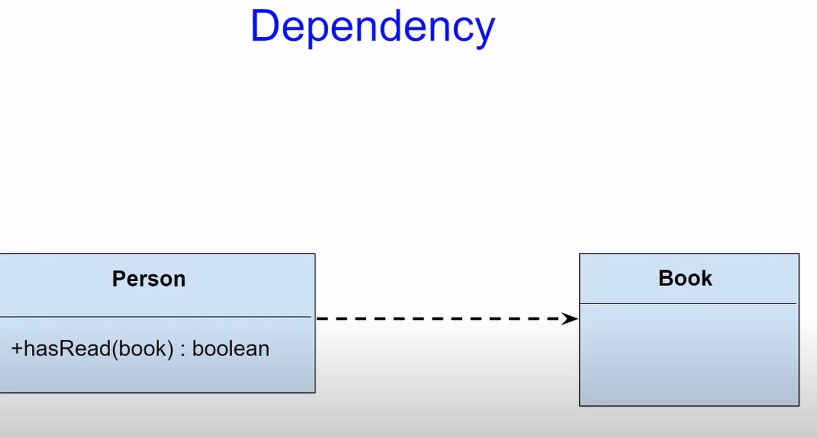
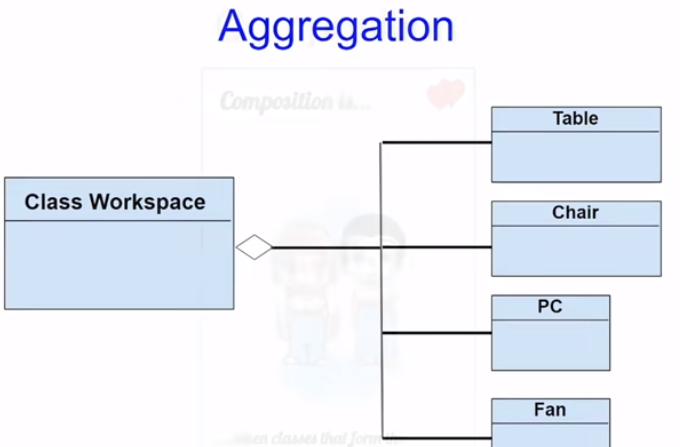


Figure showing a Dependency relationship between the Book and Person class.

1. **Aggregation.**

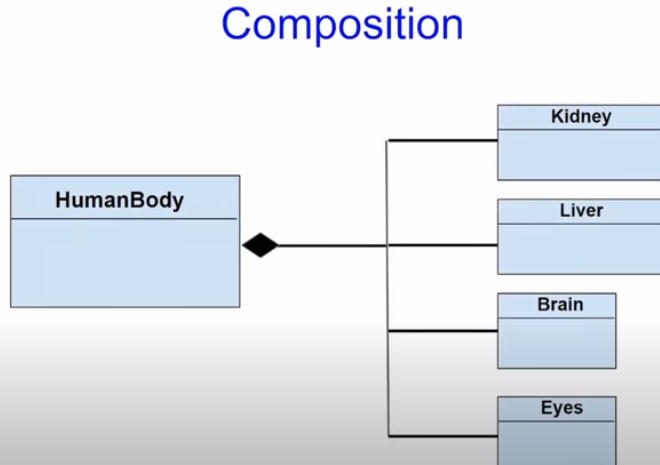
A straight line with a parallelogram or diamond at the end) that denotes object composition in which one class represents the whole while the other class represents a part of that whole. The diamond at the end shows the class that aggregates the other example car and wheel the end of the diamond points towards the “Wheel” class.

It’s a relationship where one independent class forms a part of another class. For example, a workstation with PC, Chair, Laptop. In this case, the entire unit forms the workstation class. The important distinguishing factor is that the class still performs even when the other aggregates are not present. Thus, the workstation will function without the fan.

****

1. **Composition.**

This is special type of aggregation relationship in which one class affects the functioning and features of another class. For example, altering the value of the Kidney class affects the entire body.

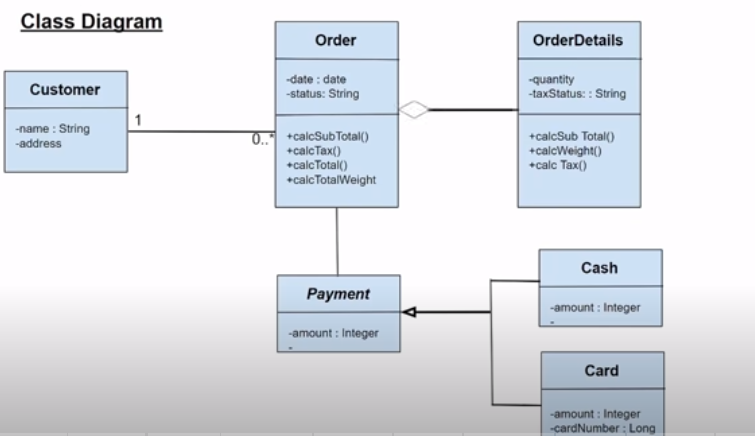


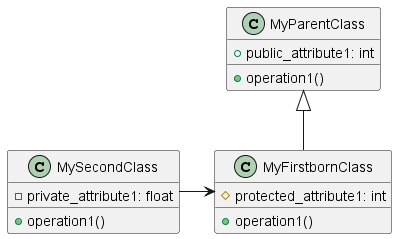
*Figure showing a composition relationship type.*

# Structure Diagrams

## Class Diagrams

Class Diagrams in the standard model for Object Oriented Programs. It describes the objects in the system and the relationships that exist between them.



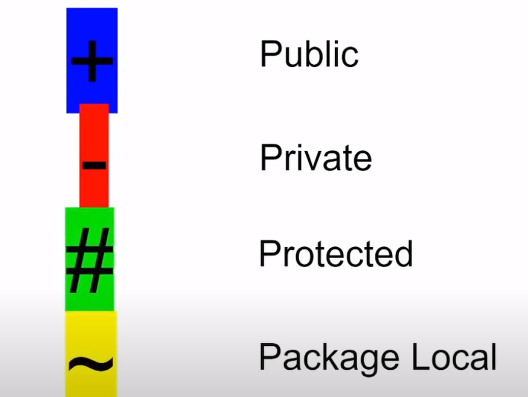


*Figure showing an illustration of a class diagram.*

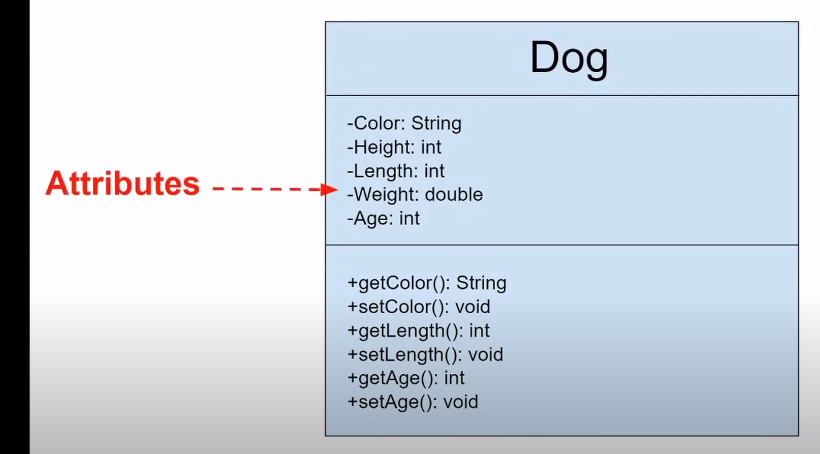
The three common relationships between the classes include:

1. **Association**
2. **Inheritance**.
3. **Aggregation**

The access modifiers before the class methods, and attributes specify the access control features associated with each them:

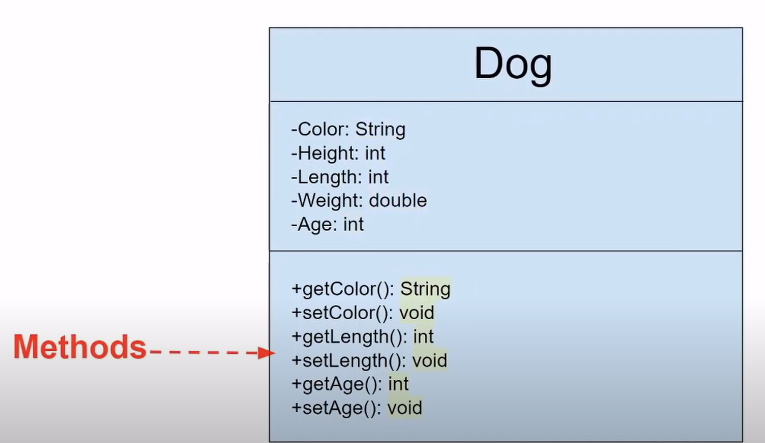


1. + Denotes Public. Methods and attributes defined with this syntax can be accessed anywhere within the class.
2. – Denotes Public
3. # Denotes Protected
4. ~ denotes Local.



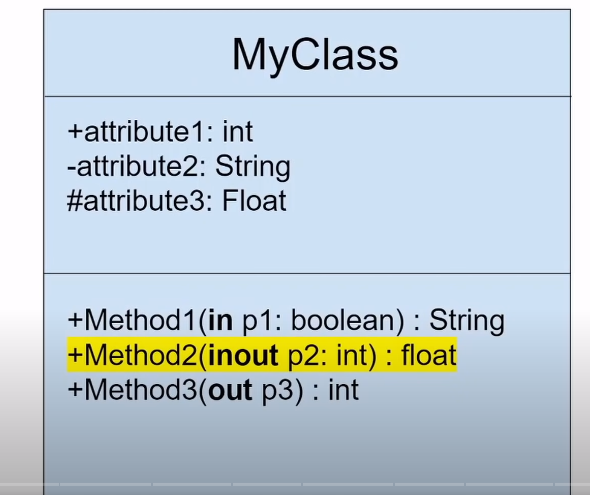
*Figure showing the Private access modifiers in the class diagram.*

The type of attribute/method is denoted after the name of the attribute/method as shown below



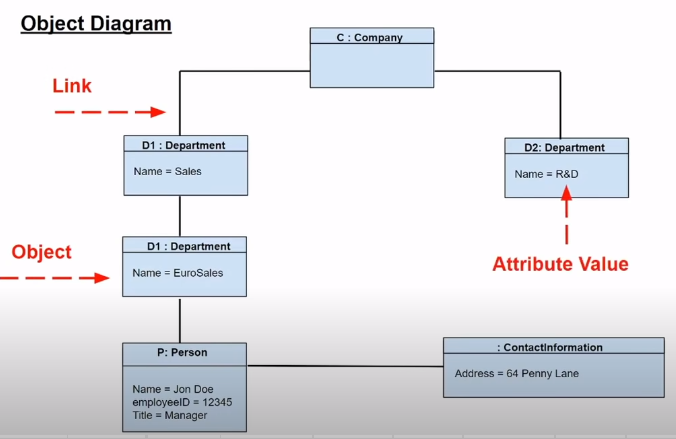
*Figure showing the types e.g., string, void, int after the Class methods and attributes in a class diagram.*

The class diagram offers the option to specify the method parameters. For example, in the figure below *myClass* accepts p1 as a boolean input parameter and p2 as a int input/output parameters as shown.

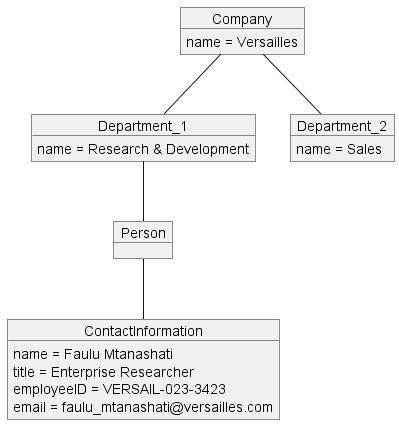


## Object Diagram

It’s an instance of a class diagram. Unlike the class diagram that shows the snapshot (abstract model) of the system, the diagram shows an instance of the system at a particular moment showing the examples of different data structures.



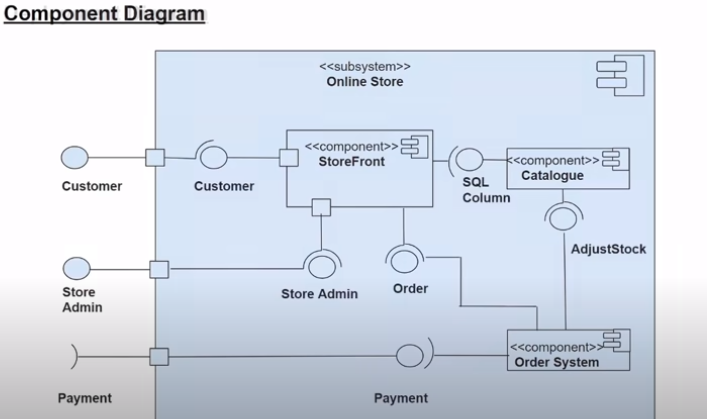
*Figure showing an object diagram.*

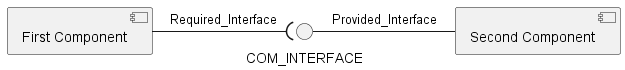


*Figure showing an object diagram example.*

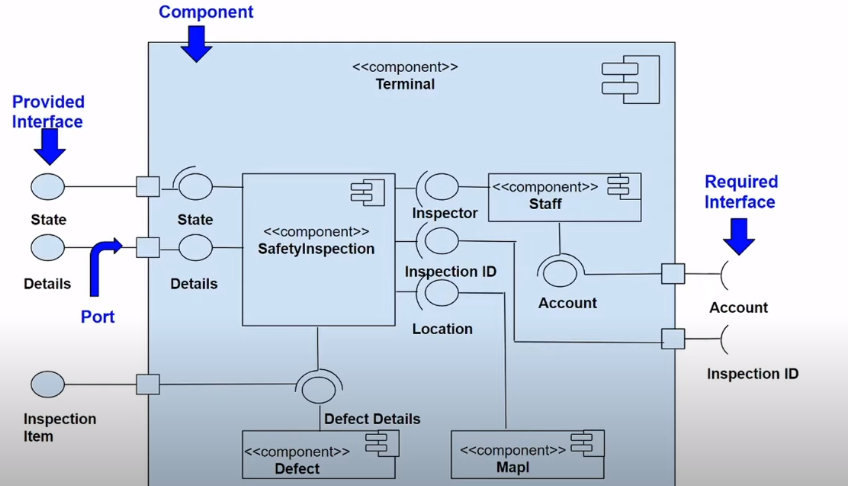
## Component Diagram

It shows the system architecture by illustrating how small components form a larger component. It includes runtime, executable components and source code components.



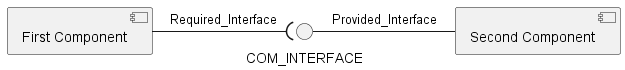


*Figure showing an example of Component Diagram.*

**

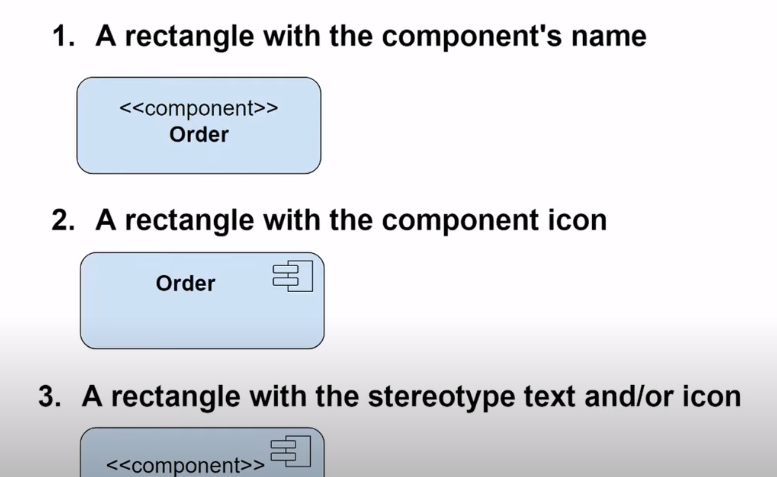
*Figure showing an example of Component Diagram with ports, Interfaces.*

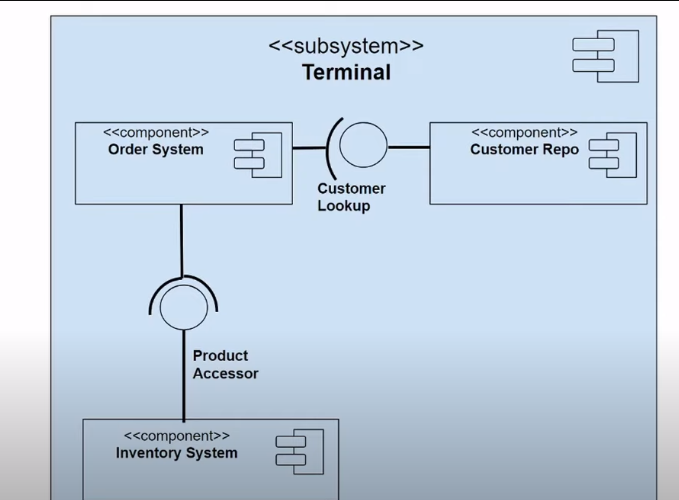
Interfaces are of two types required interfaces which are user-input based and provided interfaces which are system based.

**

*Figure with Provided interface and Required Interface.*

The components can be with just their names, their icon or a combination of both name and icon.

*Figure with the component name and component icon.*



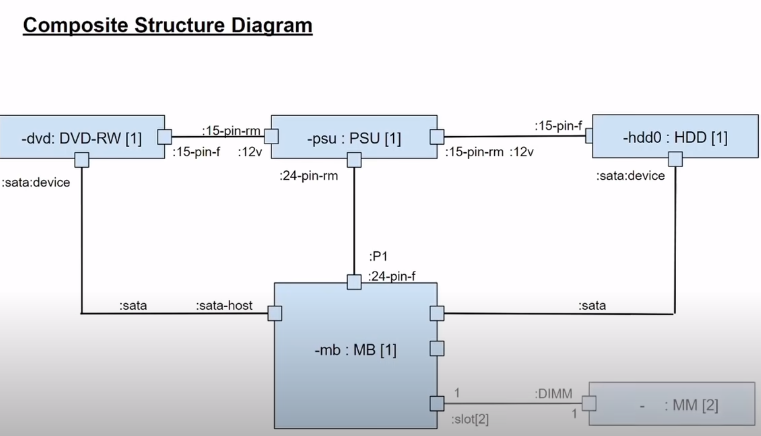
*Figure showing a subsystem using the component diagram. Subsystem has the keyword <<subsystem>> for distinguishing ability.*

The three common relationships between the objects in a component diagram include:

1. Association
2. Composition.
3. Aggregation.
4. Dependency.

## Composite Structure Diagram

It shows the internal structures of a class and the interactions within the class. It includes internal parts, connectors and ports of the external structure.



*Figure showing the Composite Structure diagram.*

## Profile Diagram

It helps to create domain stereotypes and platform stereotypes.



*Figure showing a profile diagram.*

## Deployment Diagram

It models the block diagram showing the physical components and their distribution of software artifacts in the system.



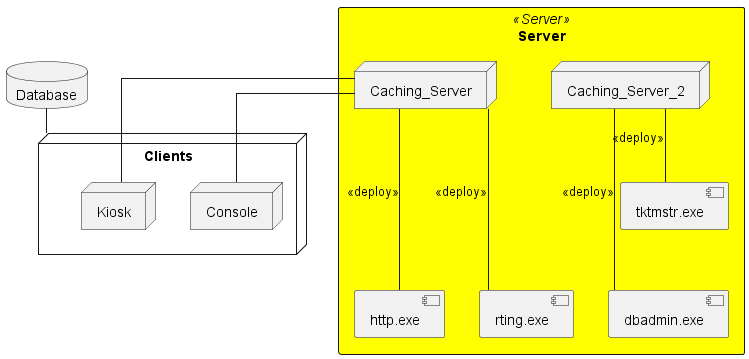
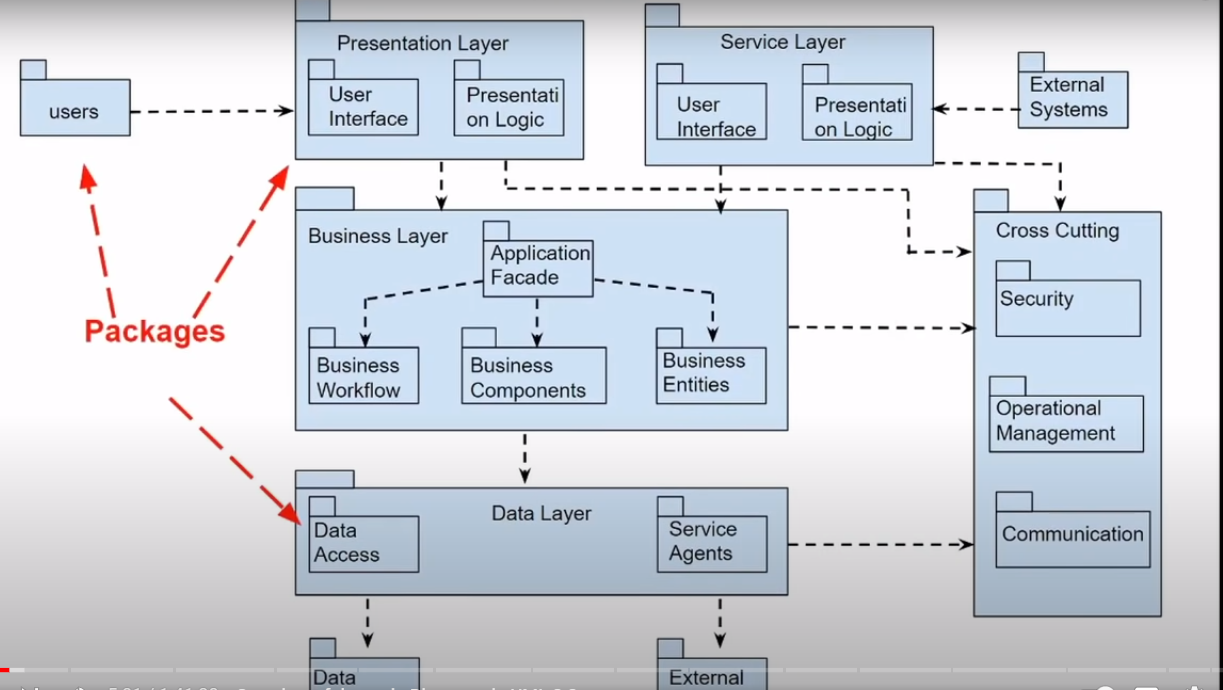
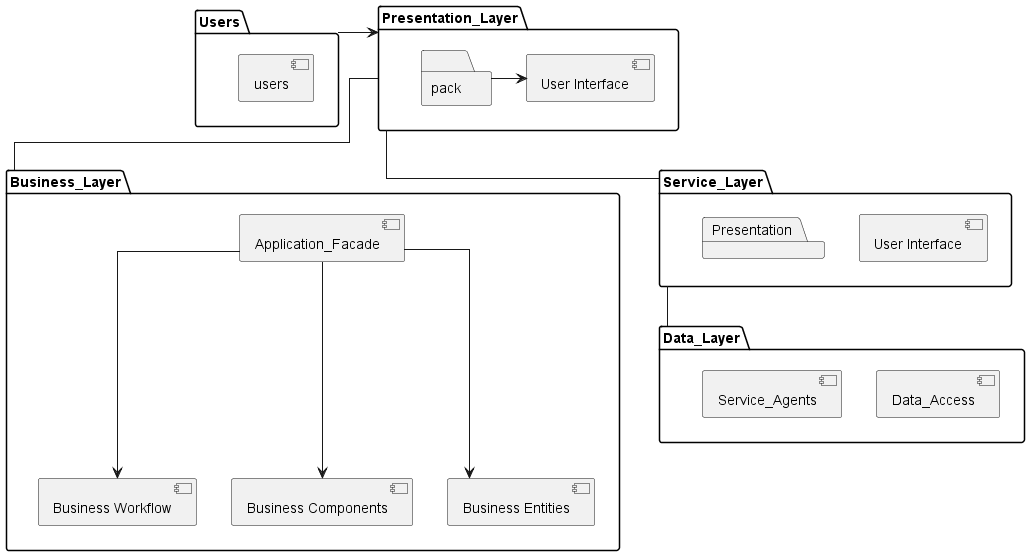


Figure showing the Deployment Diagram.

## Package diagram

This structural diagram shows the different views, packages and dependencies of the system.





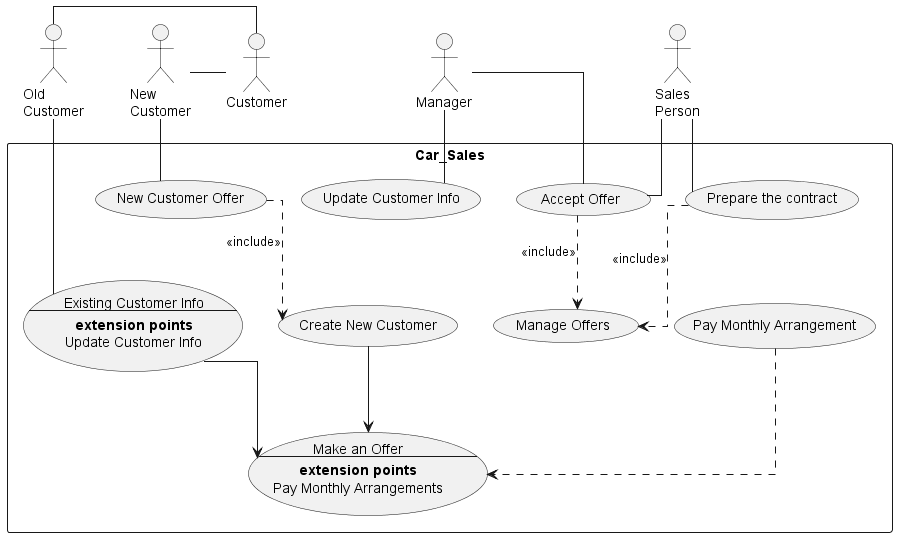
*Figure showing the package diagram.*

# Behavioral Diagrams

## Use-case Diagram

This a model of the intended features, functionality of the system and their interaction with environment (actors). It also defines the functional requirements of the system. It links what you need the system to do and how to achieve it with the system.

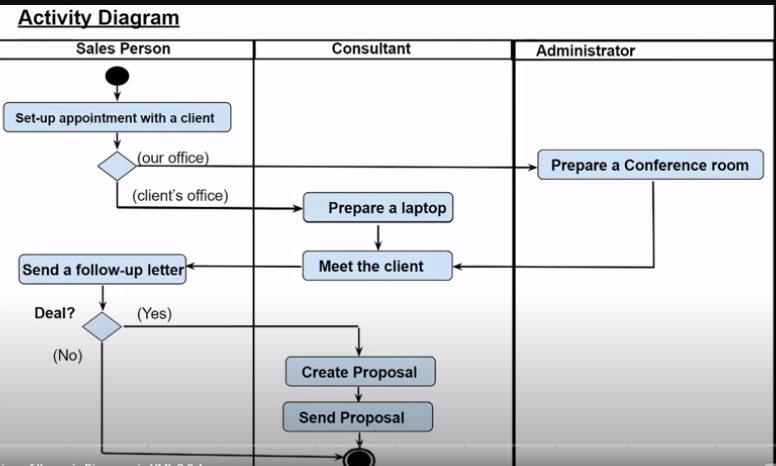


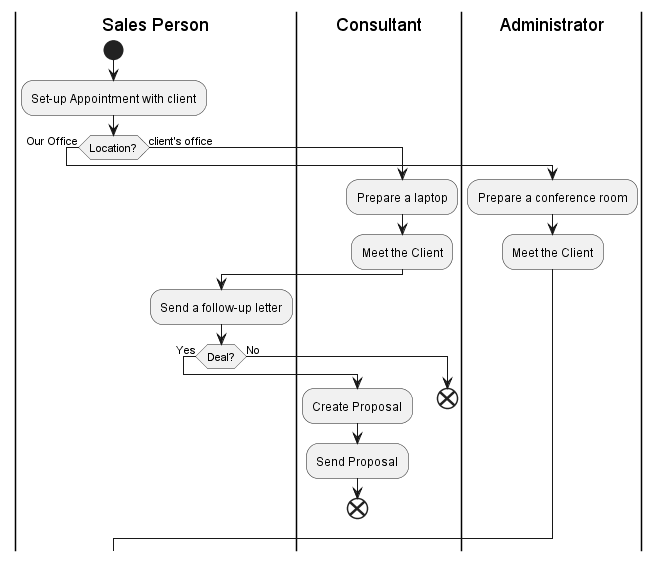


*Figure showing the Use-case diagram for a Car Sales company.*

## Activity Diagram

These are graphical workflow representations and actions in the system. They display complex roles and the interconnections of activities with different people. These diagrams model the computational and organizational roles associated with the system.

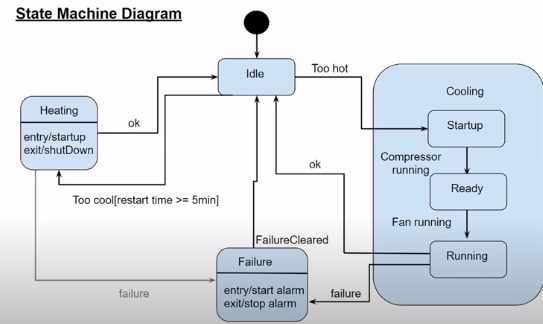




*Figure showing an activity diagram.*

## State Machine Diagram

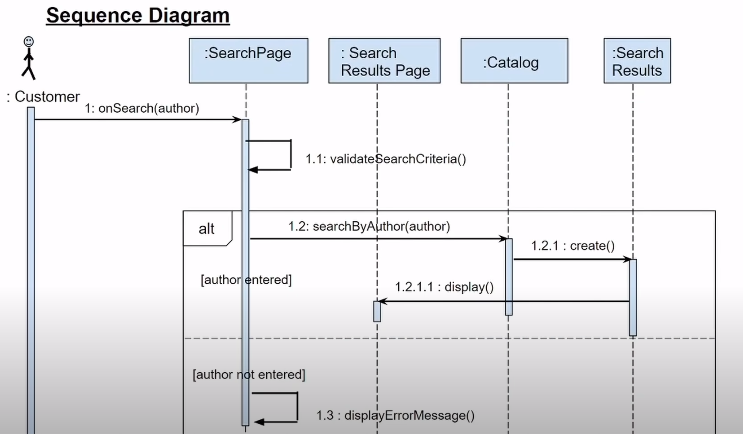
This diagram visualizes the states, transitions, and events of the life cycle of objects. It helps to better understand the system’s state of behavior.



*Figure showing State Machine diagram.*

## Sequence Diagram

It models the how objects interact with each other in a particular use-case scenario.

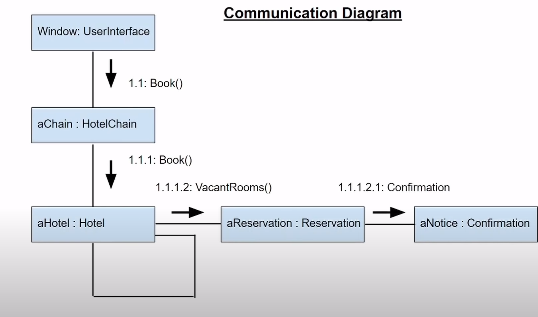




*Figure showing sequence diagram.*

## Communication Diagram

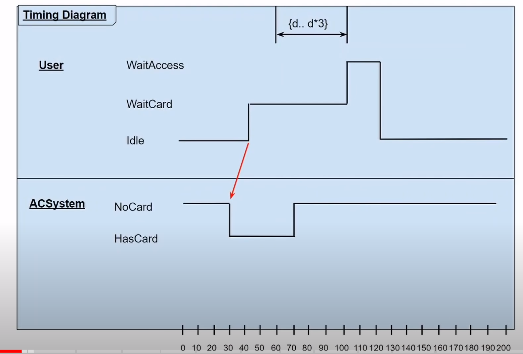
It models the dynamic behavior of a particular use-case by showing the object collaborations unlike time sequence in sequence diagrams.



*Figure showing the communication diagram.*

## Timing Diagram

This a special sequence diagram that shows the behaviour of objects at a given time.



*Figure showing a Timing Diagram.*

## Interaction Overview Diagram

It focuses on the overview of control similar to an activity diagram in which it shows the interaction occurrences.

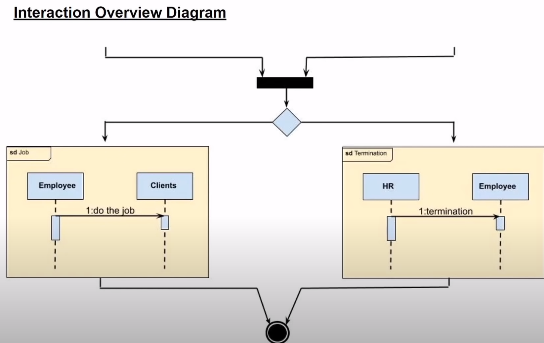


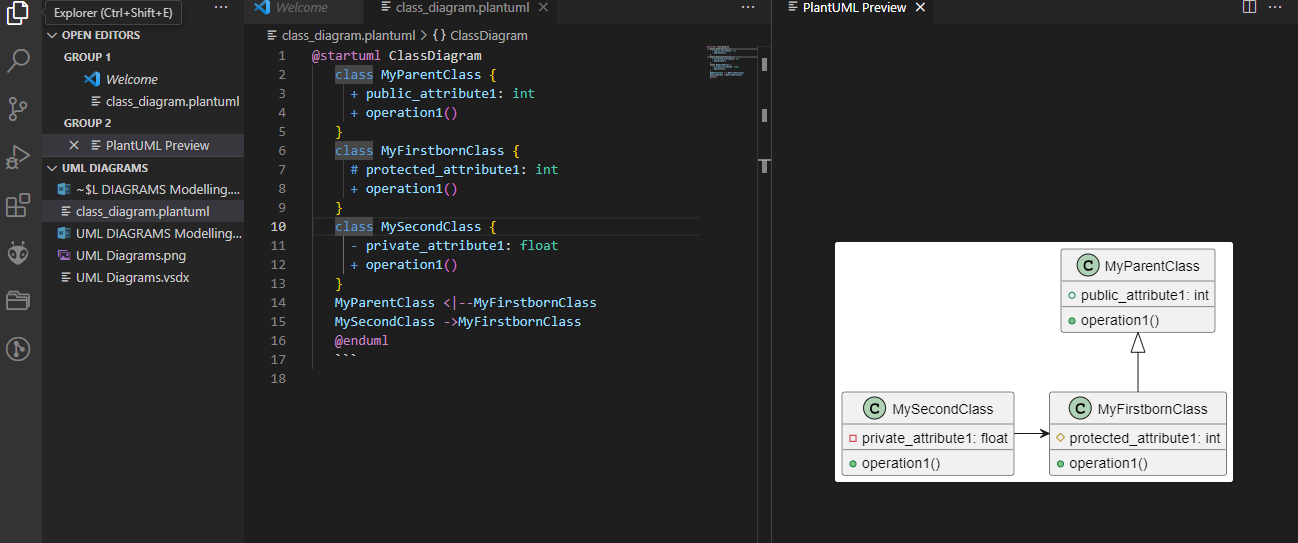
Figure showing the interaction overview.

# PLANT UML

PlantUML is a textual diagramming tool that can be used to create various UML diagrams, including both structural and behavioral diagrams. To create different UML diagrams using PlantUML code, you can follow these examples for each type of diagram:

## Structural Diagrams

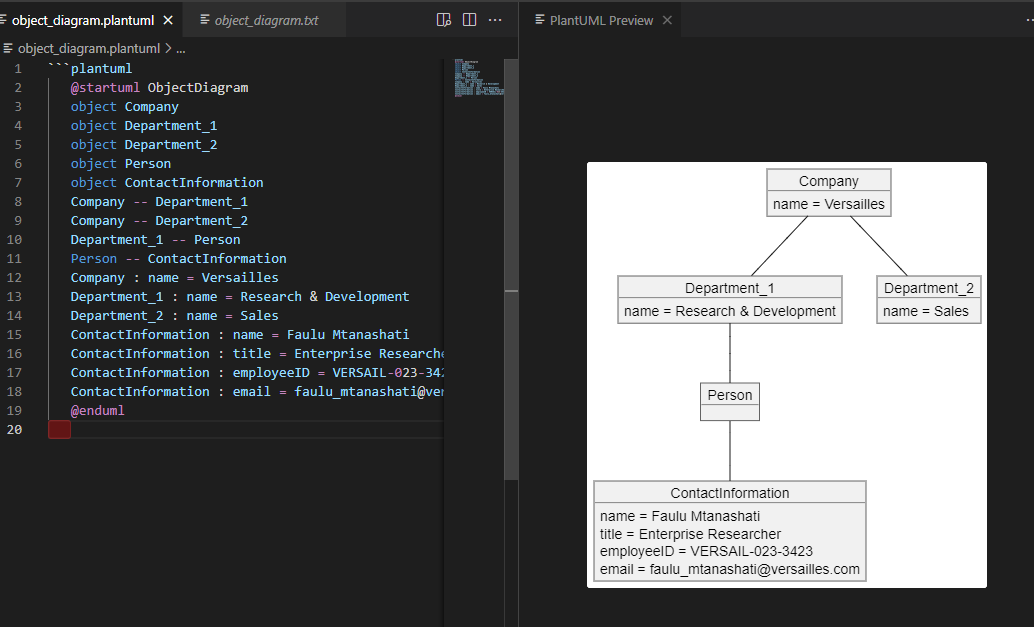
### 1. Class Diagram:



*Figure showing a Plant UML Class diagram.*

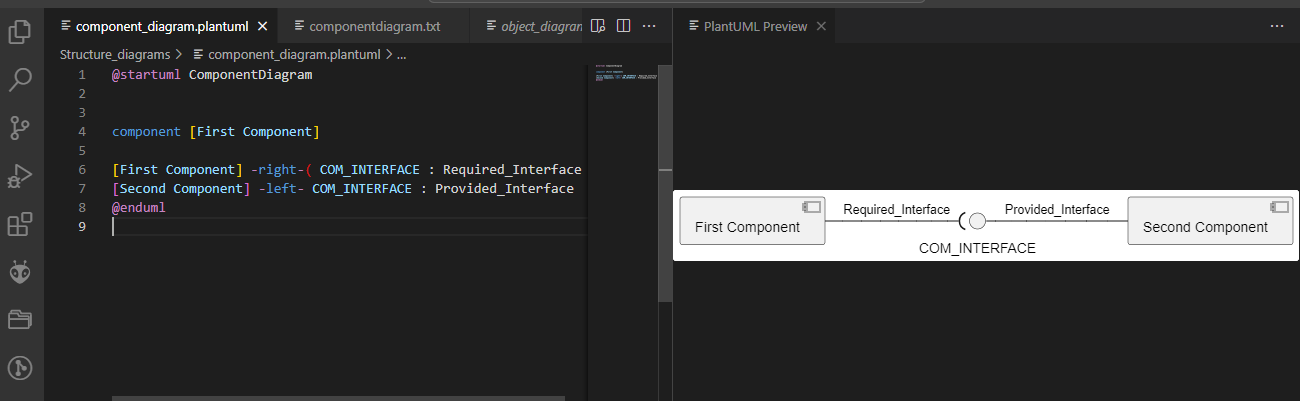
In the example above the Parent Class is the Parent, the arrow points to it as the parent from the firstbornclass. The second class has an association relationship that connects the Firstborn class with the second class. All methods are public denoted with the + access modifier, the attributes (variables) have different access modifiers i.e., either public, private or protected.

### 2. Object Diagram



*Figure showing an object diagram using Plant UML.*

### 3. Component Diagram



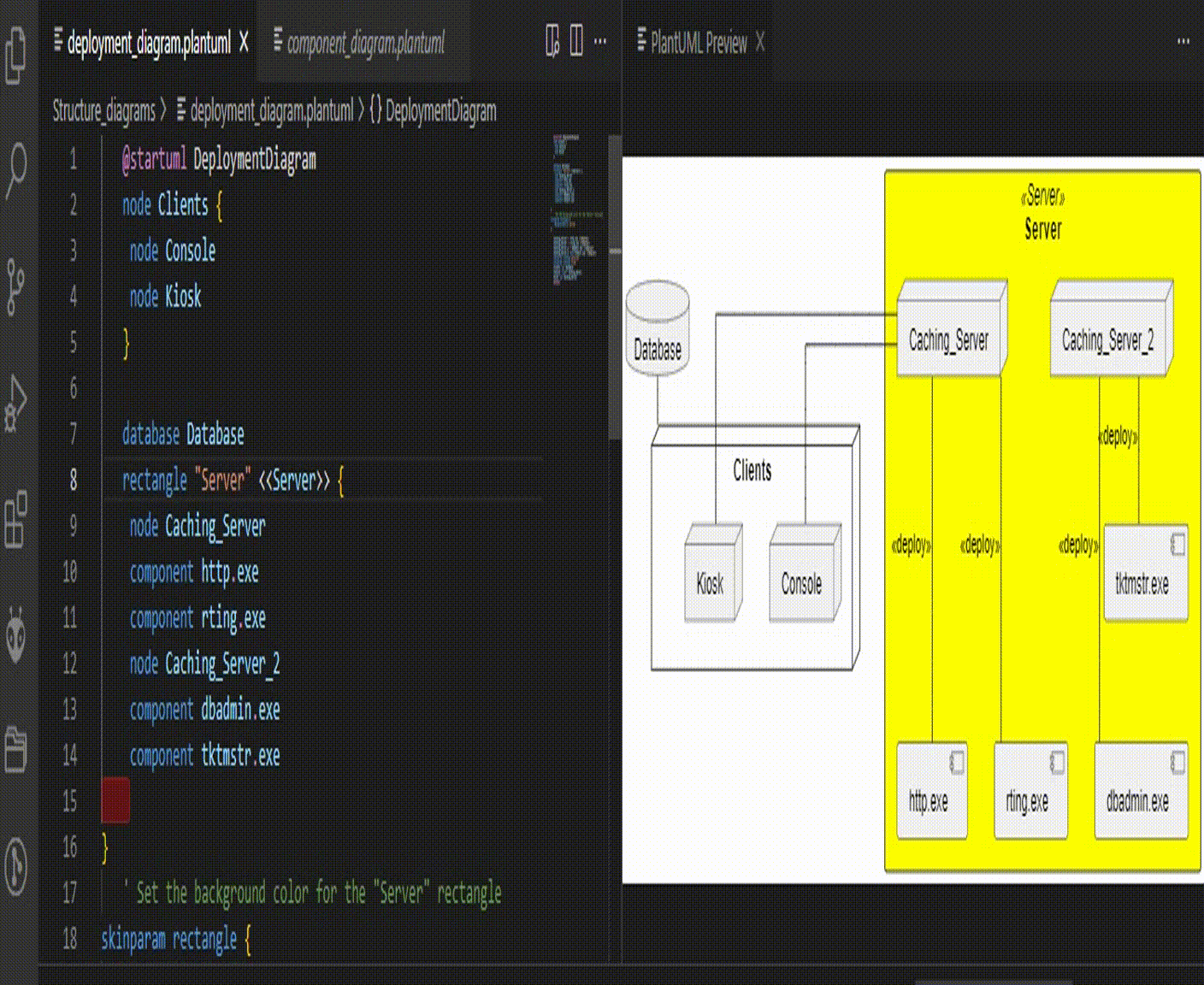
### 4. Composite Structure Diagram

You can use class diagrams to represent the internal structure of a composite structure.

### 5. Profile Diagram

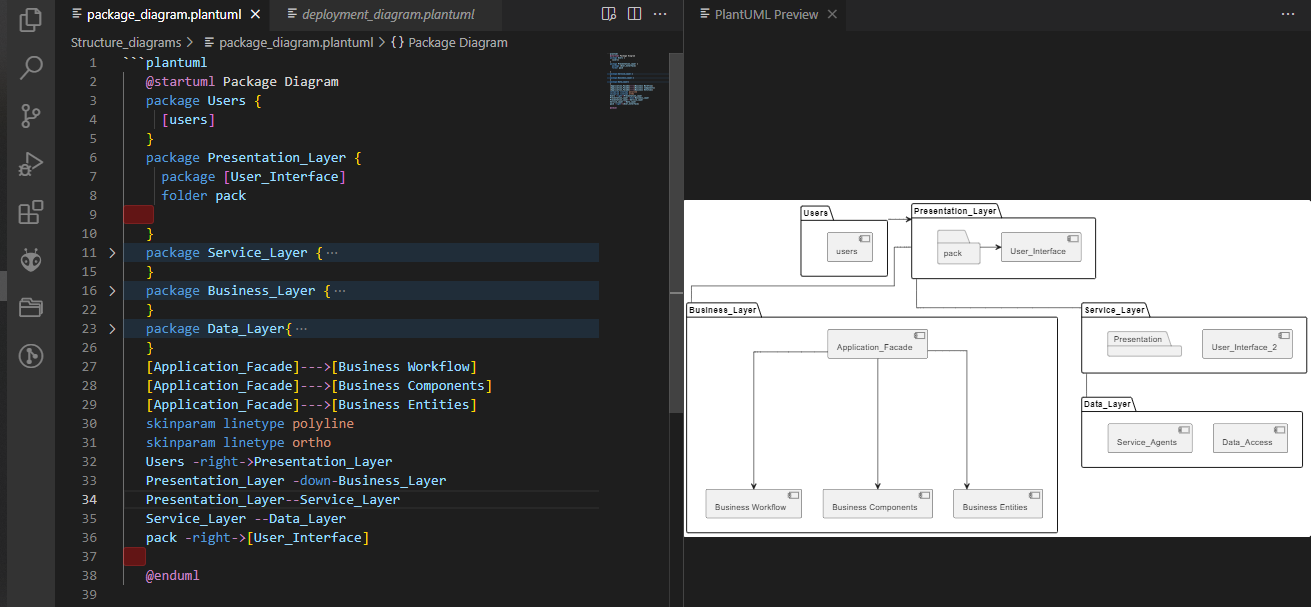
Profile diagrams in PlantUML are similar to class diagrams with specific notations for stereotypes and extensions.

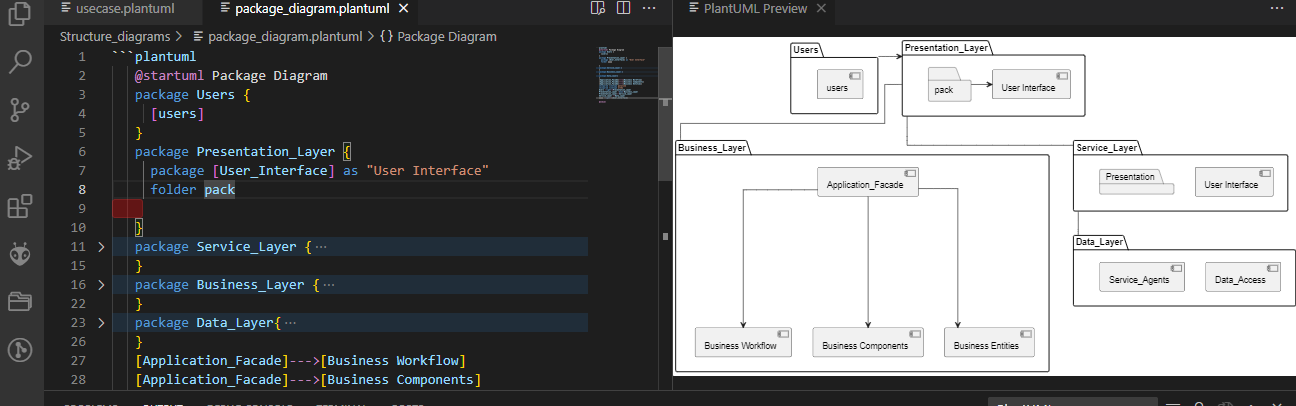
### 6. Deployment Diagram



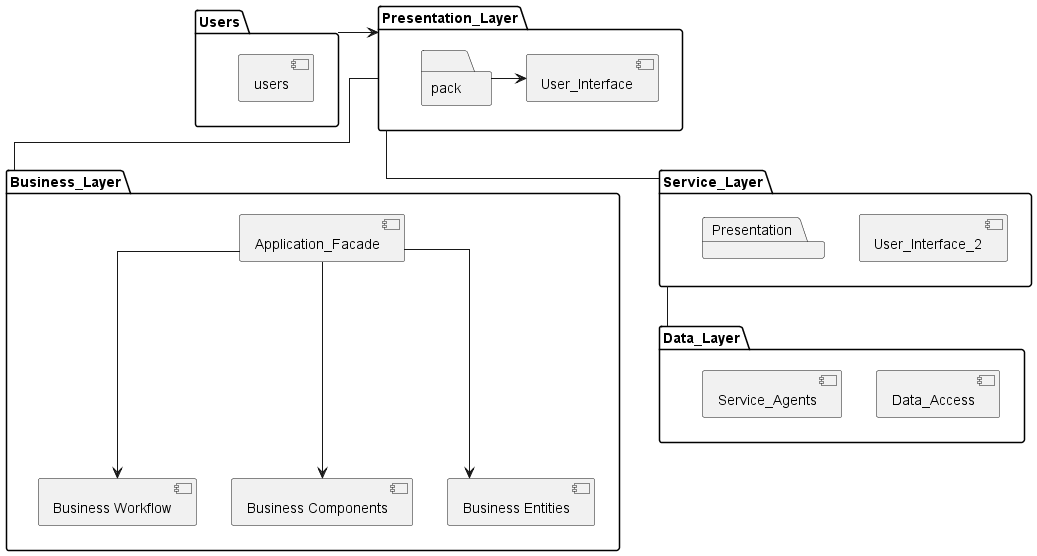
*Figure showing Deployment Diagram*

### 7. Package Diagram:



**

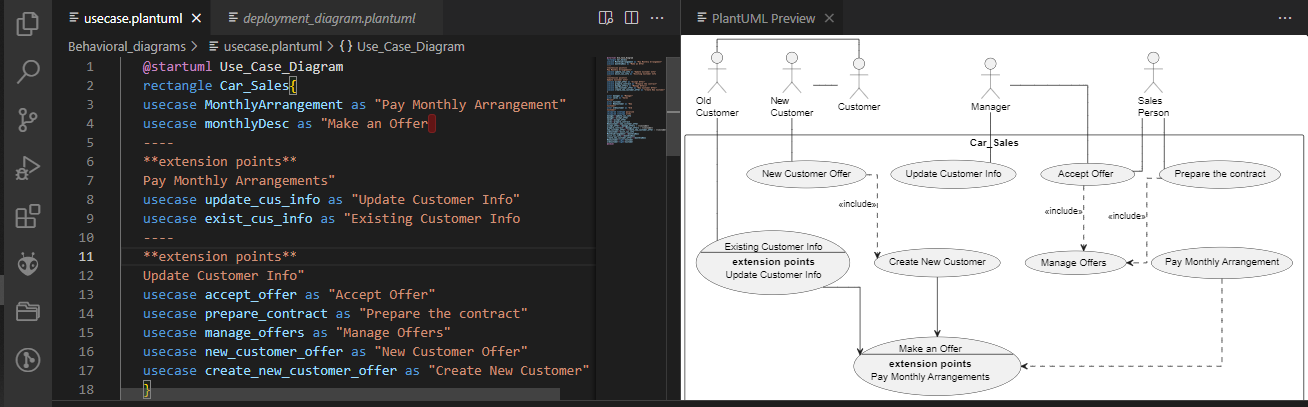
*Figure showing the Plant UML code for a package diagram.*

**

*Figure showing the final Package diagram.*

## Behavioral Diagrams

### 1. Use Case Diagram



*Figure showing use-case diagram using Plant UML.*

### 2. Activity Diagram

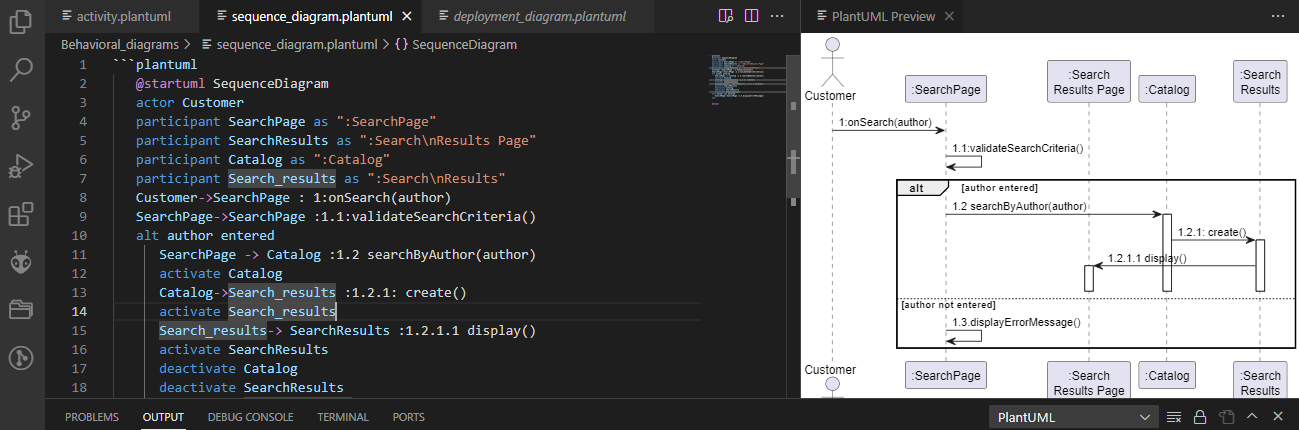


*Figure showing the Plant UML activity diagram.*

### 3. State Machine Diagram

You can represent state machines using class diagrams with specific notations.

### 4. Sequence Diagram



*Figure showing a Plant UML Sequence Diagram.*

5. \*\*Communication Diagram\*\*:

Similar to sequence diagrams but with some notation differences.

6. \*\*Timing Diagram\*\*:

Timing diagrams are not natively supported in PlantUML, but you can use textual representations for time sequences.

7. \*\*Interaction Overview Diagram\*\*:

Interaction overview diagrams are not natively supported in PlantUML, but you can represent them as combinations of sequence diagrams and activity diagrams.

Please note that PlantUML may not provide all the notational features for every type of UML diagram, but you can often adapt and represent the concepts of each diagram using class diagrams as a foundation. You can further customize and extend PlantUML's capabilities by defining your own shapes and symbols using the `!define` and `!definelong` commands.