Palash Bajpai

UML Class

Diagrams

Table of Contents

[**1. What is UML?** 2](#_Toc192045044)

[1. Types of UML 2](#_Toc192045045)

[**2. Understanding Class Diagrams** 4](#_Toc192045046)

[**3. Basics of UML Class Diagrams** 6](#_Toc192045047)

[1. Class Representation 6](#_Toc192045048)

[2. Attributes and methods 7](#_Toc192045049)

[**4. Relationships in Class Diagrams** 7](#_Toc192045050)

[1. Associations 8](#_Toc192045051)

# **1. What is UML?**

Unified Modeling Language (UML) is a standardized visual modeling language used to design and document software systems. It helps developers, architects, and stakeholders understand and communicate the structure and behavior of a system

✔️ provides a standard way to visualize system architecture.

✔️ Supports object-oriented design (OOD).

✔️ Used for designing, analyzing, and documenting software systems.

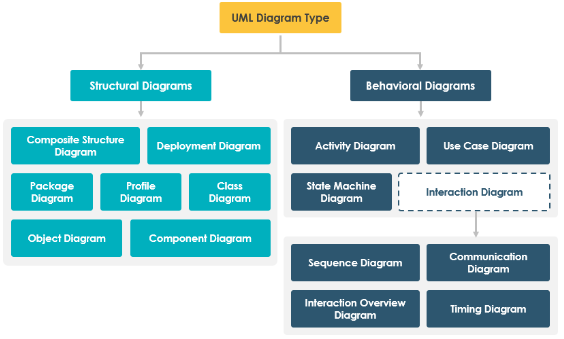
✔️ Independent of programming languages and development methodologies.

**Real World Example:**

Imagine you are developing an E-commerce Application (like Amazon). UML helps visualize:

* How users interact with the system.
* How products, orders, and payments are related.
* How different components communicate.

## 1. Types of UML



**Structural Diagrams**

These diagrams represent the static aspects of a system—how components are structured.

| **UML Diagram** | **Purpose** | **Example** |
| --- | --- | --- |
| **Class Diagram** | Defines object structure & relationships. | Customer has Orders in an e-commerce system. |
| **Object Diagram** | Represents instances of classes at a specific time. | A real-time snapshot of Order and its associated Product. |
| **Component Diagram** | Shows how components interact. | Payment module in a banking app. |
| **Deployment Diagram** | Represents hardware/software deployment. | AWS cloud infrastructure for a web app. |
| **Package Diagram** | Organizes related classes. | Organizing Customer, Order, and Product in different packages. |

**Behavioral Diagrams**

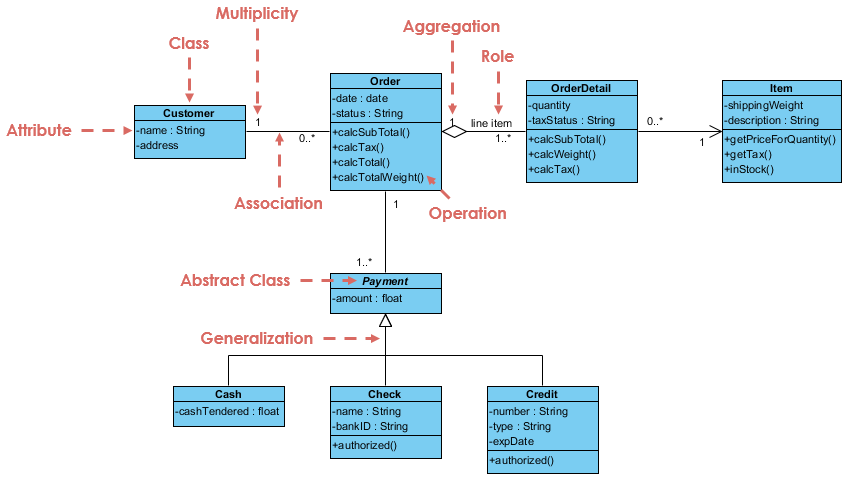
These diagrams represent the dynamic aspects of a system—how components behave over time.

| **UML Diagram** | **Purpose** | **Example** |
| --- | --- | --- |
| **Use Case Diagram** | Defines user interactions with the system. | A Customer placing an Order. |
| **Sequence Diagram** | Shows message flow between objects. | User authentication in a login system. |
| **Activity Diagram** | Represents workflows & processes. | Steps in an online payment transaction. |
| **State Diagram** | Represents object states & transitions. | Order transitioning from Placed → Shipped → Delivered. |

# **2. Understanding Class Diagrams**

A Class Diagram is a UML diagram that represents the static structure of a system. It shows:

* Classes (objects) and their properties/methods.
* Relationships between classes (Association, Inheritance, etc.).
* Visibility (public, private, protected).✔️ provides a standard way to visualize system architecture.



**Role of Class Diagrams**

* Blueprint of the System: Helps design object-oriented software before coding.
* Easy Communication: Developers, architects, and stakeholders understand relationships better.
* System Maintenance: Helps in debugging and extending functionalities.
* Reduces Complexity: Breaks large software into smaller, manageable parts..

**Similarity with OOPs**

Mapping of UML concepts to OOP

| **UML Concept** | **OOP Concept** | **Example** |
| --- | --- | --- |
| **Class** | Blueprint of objects | BankAccount, Customer, Order |
| **Attributes** | Class variables | balance in BankAccount |
| **Methods** | Functions in a class | withdraw(), deposit() |
| **Association** | Object relationship | Customer has multiple Orders |
| **Generalization** | Inheritance | AdminUser extends User |
| **Composition** | "Has-a" relationship | Car has an Engine |

# **3. Basics of UML Class Diagrams**

## Class Representation

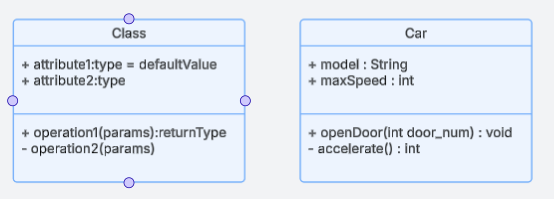
A Class in UML represents a blueprint for objects. It defines attributes (data) and methods (behavior).

A class in UML is depicted as a rectangle divided into three sections:

1️. Class Name (First Section)

2️. Attributes (Properties/Fields) (Second Section)

3️. Methods (Functions/Operations) (Third Section)



## Attributes and methods

**Attributes:** Attributes define the state of a class (fields/variables).

**Methods:** Methods define behavior (actions).

**Visibility:** Visibility notations indicate the access level of attributes and methods.

| **Visibility** | **Symbol** | **Meaning** |
| --- | --- | --- |
| Public | + | Accessible by any class |
| Private | - | Only accessible within the class |
| Protected | # | Accessible by the class and subclasses |

# **4. Relationships in Class Diagrams**

In UML, relationships between classes define how objects of different classes interact. Understanding these relationships is crucial for object-oriented design and software modelling.

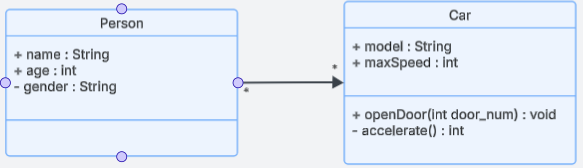
## 1. Associations

An Association represents a relationship between two or more classes where objects of one class are linked to objects of another class. It can be bidirectional or unidirectional.

**Notation**

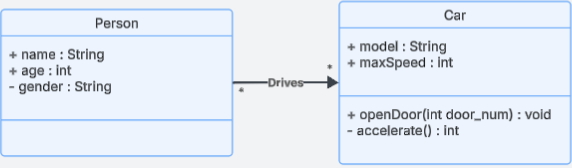
* Solid Line (—) connects related classes.
* Arrow (→) for unidirectional associations.

Eg1: User can use car, but car can’t use user



Multiple times we can even add uses on the connecting arrows

Eg2: Person drives the car



**Types of Associations:**

1️. One-to-One

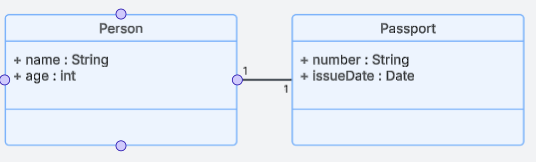
2️. One-to-Many

3️. Many-to-Many

1. **One-to-One (1:1)**

Each instance of Class A is related to exactly one instance of Class B.

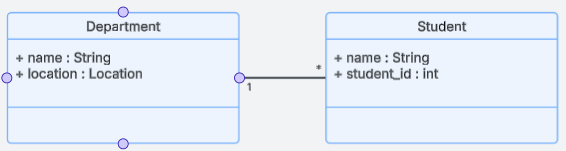
Eg: A person can have exactly one Passport



1. **One-to-Many (1:M)**

One instance of Class A is related to multiple instance of Class B.

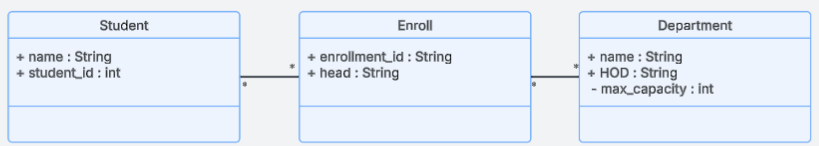
Eg: A department can have multiple students



1. **Many-to-Many (1:M)**

Multiple instance of Class A is related to multiple instance of Class B.

Eg: A Student can enrol in multiple Courses, and a Course can have multiple Students.



## 2. Aggregation

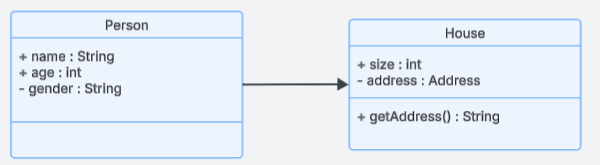
Both Aggregation and Composition represent “whole-part” relationships but differ in ownership strength.

Aggregation denotes a stronger relationship where one class (the whole) contains or is composed of another class (the part).

**Key points**

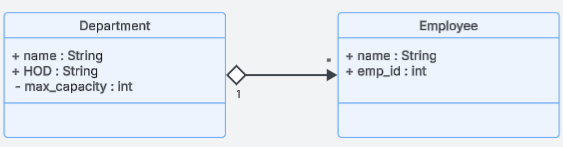
* Represents "Has-A" relationship
* Is a weaker relationship
* Child can exist independently of Parent
* **Notation**: Hollow Diamond (◊) at the whole

Eg: Person has a house, but house can exist independent from person



Aggregation is represented by a diamond shape on the side of the whole class. In this kind of relationship, the child class can exist independently of its parent class.

Eg 2: A Department (whole class) has multiple Employees, but Employees can exist independent of department too.



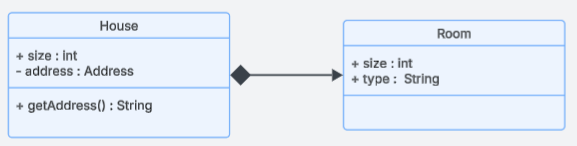
## 3. Composition

Composition is a stronger form of aggregation, indicating a more significant ownership or dependency relationship. In composition, the part class cannot exist independently of the whole class.

**Key Points:**

* Represents "Part-Of" relationship
* Child CANNOT exist without Parent
* Notation: Filled Diamond (◆) at the whole

Eg: A House has multiple Rooms, but a Room cannot exist without a House.



**Key Differences:**

| **Feature** | **Aggregation** | **Composition** |
| --- | --- | --- |
| **Dependency** | Child exists independently | Child cannot exist without Parent |
| **Lifespan** | Child’s life is independent | Child’s life depends on Parent |
| **Example** | Company & Employees | House & Rooms |

## 4. Generalization (Inheritance)

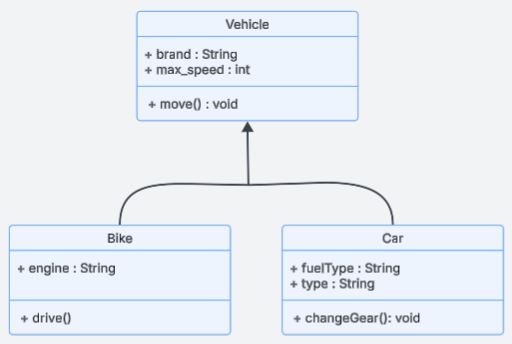
It represents an "is-a" relationship between a parent class (superclass) and its child classes (subclasses). It models inheritance in object-oriented design, allowing the child classes to inherit attributes and behaviors from the parent class while also enabling specialization.

**Key Points:**

* Represents "IS-A" relationship (Parent-Child relationship).
* Superclasses (Parent) define common attributes and methods.
* Subclasses (Child) inherit these features.
* Notation: Arrow (▲) pointing to Superclass

Eg: Vehicle -> Car, bike

* Car and Bike inherit Vehicle properties (brand, speed, move()).
* Car has its own method drive(), and Bike has ride().

****