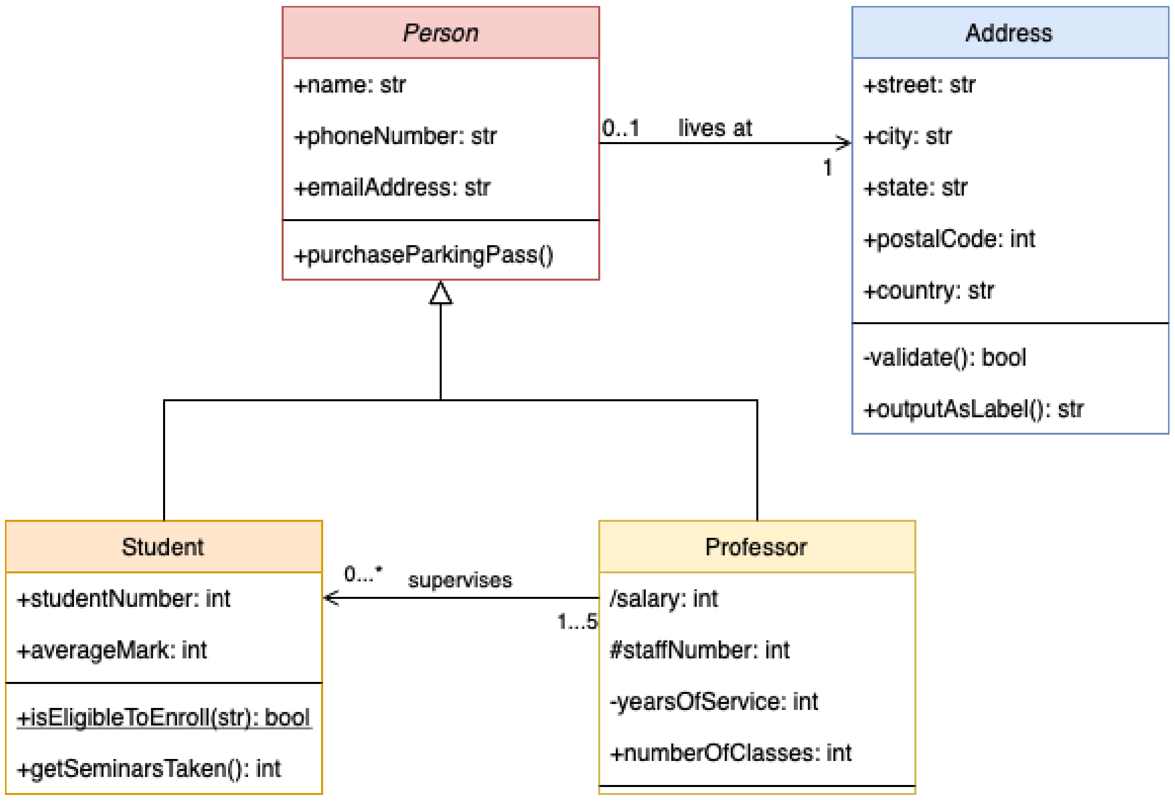
Unified Modeling Language (UML) is a visual language used to design and document software systems. It provides a set of instructions and notations that help developers visualize and communicate complex system structures and behaviours.

UML class diagrams are used to illustrate the types of data or attributes stored within each 'class', the methods (operations or functions) that each class provides, and the relationships between the classes.



Class diagrams are likely the most common and important type of UML diagram.

UML class shapes always have a **Class Name**.

Classes in detailed diagrams have two additional sections for attributes and methods, and usually follow the UML notation, detailed in the sections below.

Detailed UML classes are separated into three sections with the class name at the **top**.

- The **middle section** contains the attributes (data or variables) of the class: attributeName: type
- The **lower section** contains the methods (operations or functions) that the class can execute: method Name(parameter Name: type): type
- Parameter names, and data types are optional. They are typically used in a detailed class diagrams of an implementation or for formal specification documents.

The visibility of attributes or methods is indicated with a symbol before their names.

- + **Public** - elements are accessible from outside that class.
- - **Private** - elements are only accessible to methods inside that class.

- # **Protected** - elements are only accessible within its namespace (a group of classes and packages).

A forward slash / indicates an attribute is **derived** or computed from other attributes.

Scope

- **Instance scope** is assumed by default - attributes can have different values whenever that class is instanced, and the methods that are executed may alter that instance.
- **Class scope** or static scope is shown by underlining the name of an attribute - attribute values are the same across all instances of that class.

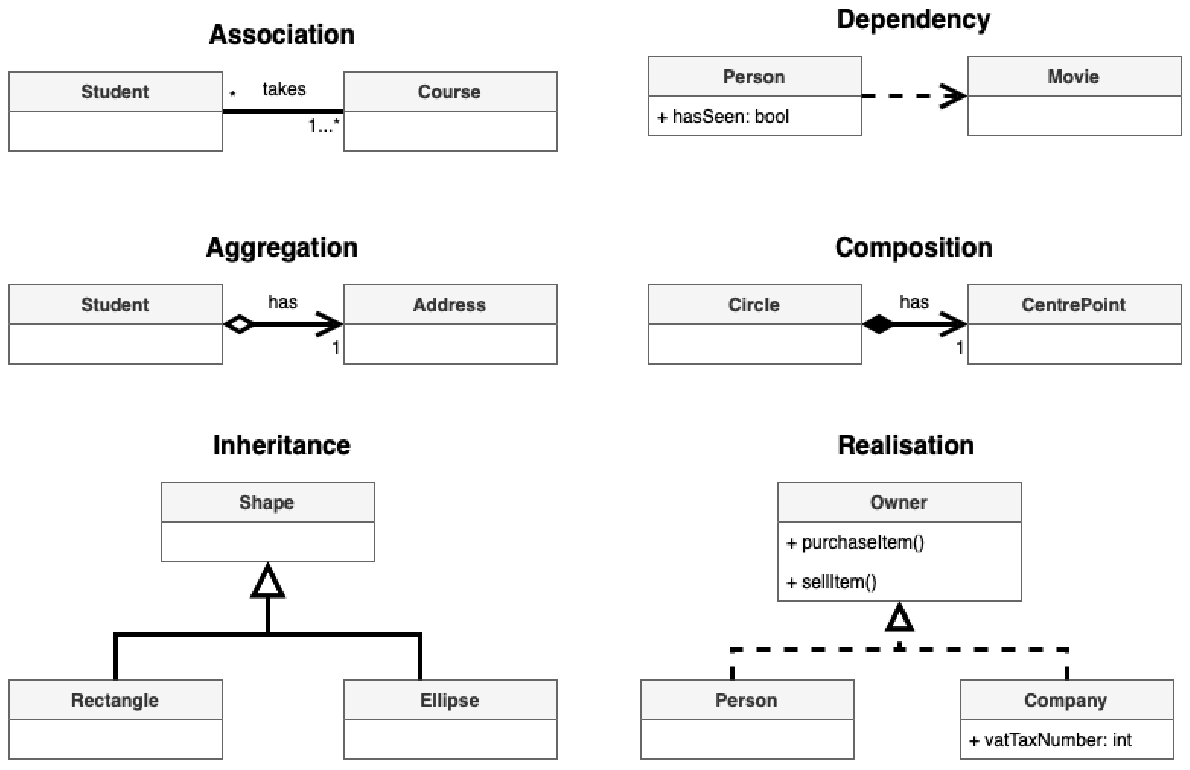
You can add the data types to attributes and methods, including both the data passed to that method, and the data returned by it. Use a colon followed by the data type.

For example: +name: string or +area (radius: float): double

Define the type of relationship between classes using a connector that has a specific symbol, labels at either end, and either a solid or dashed line.

- **Inheritance (generalisation):** An open triangle at the parent class. This shows that the subclasses are specialisations of the parent class - they extend the parent class.
- **Implementation (realisation or execution):** A dashed line with an open triangle at the blueprint class. This is used where a class implements the functionality of a 'blueprint' class, and may be implemented differently depending on each class that implements it.
- **Dependency:** A dashed line with an open arrow. If the definition of class 2 changes, it will change class 1, but not the other way around (depends on).
- **Association:** A solid line between two classes. Add arrows at either end or both ends to show that the classes are aware (or unaware) of each other.
- **Aggregation:** An open diamond at class 1. A special type of association that shows that class 2 is *a part of* class 1.

- **Composition:** A solid diamond at class 1. A special type of aggregation that shows that class 2 *can not exist without* class 1.



You can add labels to a connector to further define the relationship between the classes.

- Specify how many instances of each class are in the relationship: one to one, one to many, many to many, an exact number, etc.
- Explain why the relationship exists.
- Note ownership or roles.
- Clarify visibility.

Other UML Diagrams:

- Activity Diagram
- Use case Diagram
- Sequence Diagram
- State Machine Diagram
- Component Diagram

Note: Reference has been taken from below given book.

UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition by Martin Fowler

UML 2.5 User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, et al.

Head First UML: A Practical Guide to Using UML by Craig Larman