UML Class Diagrams

Table of contents

Class boxes 1
Relationship types 2
Layout 3
Multiplicity 4
Why use UML? 4

UML (Unified Modelling Language) diagrams are the standard, language-agnostic way of visually describing the classes in a system and the relationships between them — useful for planning a design *before* writing code, and for communicating that design to others.

This note is the notation reference. The lecture note is 2025-10-07-uml[1]; for how CSSE1001 draws diagrams generally, see csse1001-diagram-notation[2].

Class boxes

A class box is a rectangle in three compartments, separated by full-width horizontal rules:

```
ClassName
-----------------------
attributes
-----------------------
methods()
```

Each attribute/method is prefixed with a **visibility** marker:

[1] courses/csse1001/2025-10-07-uml.pdf
[2] courses/csse1001/csse1001-diagram-notation.pdf

Symbol	Visibility
`-`	private
`+`	public
`#`	protected

CSSE1001 conventions:

- **Attributes are typed Python-style**: `name: type`, e.g. `#age: int`, `-nationality: str`, `-pets: []`.
- **An empty compartment is kept, not collapsed.** A class with no attributes of its own is still drawn with a blank attributes band.
- **Names follow the course's Python naming rules** — `CamelCaps` for class names, `snake_case` for members (see python-pep8-style-guide[3]).
- **Single-compartment, name-only boxes are acceptable** when only the shape of a hierarchy is being shown.
- **Method return types are optional**: the same method can be written `+info(): str` or `+info()`. Do not read meaning into the difference.

Relationship types

Relationship	Line style	Meaning
Inheritance	solid line, **filled** triangular arrowhead pointing at the parent	"is-a" — subclass inherits from superclass
Association	plain solid line, no head, optional verb label at one end	one class uses/references another, without ownership
Aggregation	solid line, hollow (open) diamond at the "whole" end	weak "has-a" — the part *can* exist independently of the whole
Composition	solid line, filled (solid) diamond at the "whole" end	strong "has-a" — the part *cannot* exist independently of the whole
Dependency	dashed line, small **filled** arrowhead	one class depends on another

[3] courses/csse1001/python-pep8-style-guide.pdf

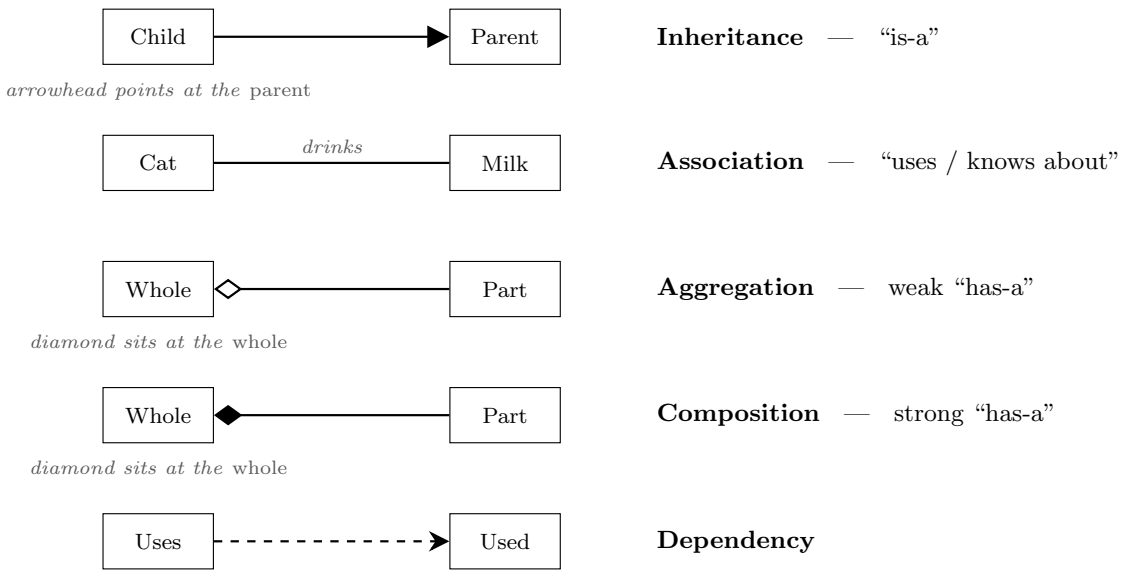

Figure 1: The relationship symbols as CSSE1001 draws them. The inheritance arrowhead is filled, which is the course's deviation from textbook UML, and the diamond always sits at the whole end. The dependency head is filled as well. Child, Parent, Whole and Part are generic stand-ins for concrete classes.

> **Filled, not hollow**
>
> Standard UML draws generalisation with a **hollow** triangle. CSSE1001 draws it **solid filled** on every inheritance arrow. Follow the course, not the textbook. See csse1001-diagram-notation[4].

> **The dependency head is solid as well**
>
> The dashed dependency arrow ends in a **filled** head in CSSE1001, not an *open* one. The head takes two forms: a **filled concave-backed** "classic" arrow, or a plain **straight-backed filled triangle**. Both are solid, and both differ in *shape* from the inheritance head.

Layout

- **Routing is orthogonal.** Relationship lines use right-angle elbows, never diagonals.

[4] courses/csse1001/csse1001-diagram-notation.pdf

- **A parent with several children gets one bus**: a single vertical stem from the parent down to a horizontal line, with a vertical drop to each child — not one arrow per child.
- **Parent above, children in a row beneath.** A grandchild hangs directly below its own parent with its own arrow.
- **Non-hierarchy classes usually sit to the side**, at the same height as the class they attach to (`Vet`, `Heart` and `Owner` in the examples in 2025-10-07-uml[5]). The exception is `Milk`, which hangs directly *below* `Cat` with the association line running down `Cat`'s left edge.

Multiplicity

Multiplicity annotations on association/aggregation/composition lines describe how many objects on each end participate in the relationship. They are placed **above the line, next to the end they qualify**:

Notation	Meaning
`0..1`	zero to one
`n`	an exact number
`0..*`	zero to many
`1..*`	one to many
`m..n`	a specific range

Why use UML?

- Plan a class hierarchy/object graph before writing any code.
- Communicate a design to teammates without needing to read source code.
- Document the intended structure and relationships for future maintenance.
- Class diagrams are the most commonly used UML diagram type in object-oriented design, but UML also includes other diagram types (e.g. sequence diagrams, use-case diagrams) not covered in this course.

[5] courses/csse1001/2025-10-07-uml.pdf