Slide 14: Unified Modeling Language

Made by Md. Mehedi Hasan Rafy

Class Diagram

A Class Diagram represents classes and their inter-relationship. It describes the static structure of a simple system.

Classes can be related to each other in four ways.

1. Generalization
2. Aggregation
3. Association
4. Different dependencies

Class

A class represents a set of entities with common features. It is drawn as a solid ***rectangle*** (can be split into compartments: name, attributes, operations). For example:

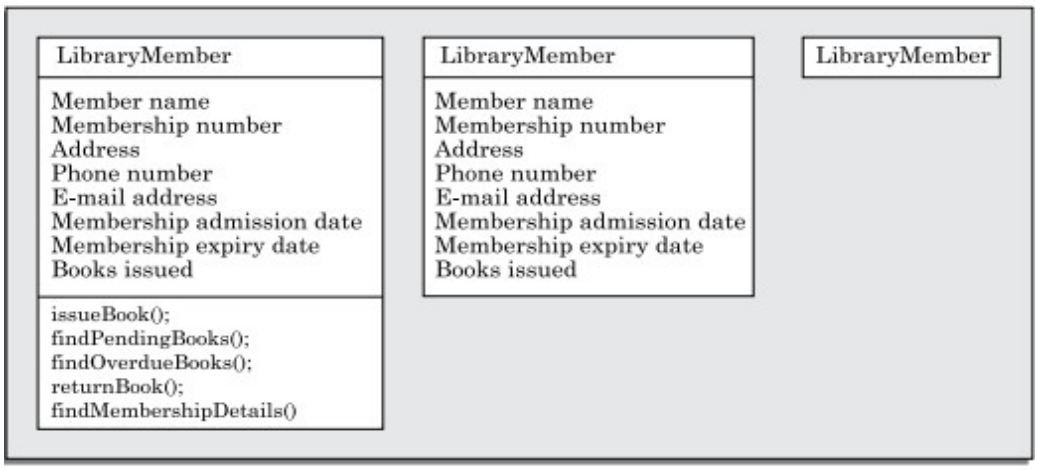


Fig: Different representation of the LibraryMember class

- `LibraryMember` (class name, starts with uppercase).
- A specific `studentMember` (object name, starts with lowercase).

Classes can have attributes and operations compartments. If a class appears in multiple diagrams its attributes and operations are suppressed into one diagram. At the start of the design only class names are identified.

Attributes

An attribute is a named property of a class. It represent the kind of data that an object might store. (e.g., `name:String`, `age:Int`). Attributes –

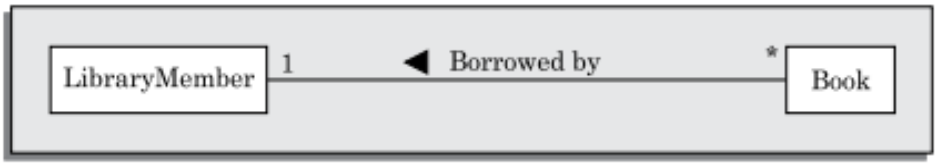
- Written in lowercase (e.g., `name`, `age`).
- Can include type after a colon (e.g., `age:Int`).
- Can include initial value or constraints (optional).
- Use [ ] for multiplicity (e.g., `grades[6]:Float` → each object has 6 grades).

Association

Association is shown as a ***straight line*** between two classes and ***name*** of the association, and an arrowhead indicating the reading direction of the annotated association name is written along the line.

Multiplicity which indicates how many instances of one class is associated with other classes is written at each end like this –

- `1` = one // single number
- `1..5` = between 1 and 5 // range – separated by two dots.
- `*` = many // zero or more

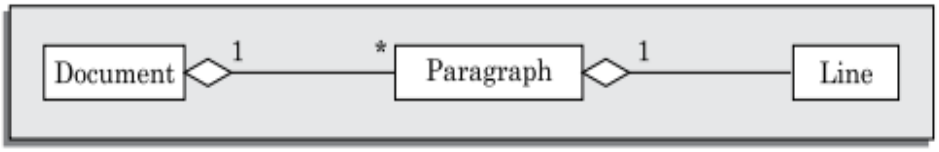


Example: *One LibraryMember can borrow many Books; each Book belongs to exactly one member.*

Aggregation

A special type of association that shows a whole–part relationship. The whole object can create and destroy its parts, but the parts can still exist independently.

- Represented with an ***empty (hollow) diamond*** at the aggregate end of the relationship.



Example: A `Document` is an aggregation of `Paragraph`s → one Document (1) contains many Paragraphs (*).

Composition

Composition is stronger form of aggregation where parts cannot exist without the whole. The whole and parts share the same lifetime → created and destroyed together.

- Represented with a ***filled (black) diamond*** at the aggregate end of the relationship.

Example: An `Order` and its `Items` → when the Order is deleted, all its Items are deleted too.

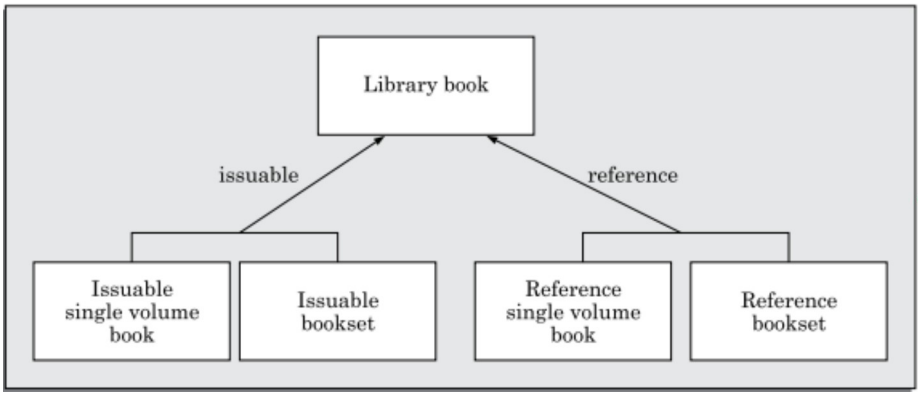
Aggregation vs. Composition

Feature	***Aggregation***	***Composition***
Type	Weak "whole–part" relationship	Strong "whole–part" relationship
Symbol	Hollow diamond ◇	Filled diamond ◆
Independence of parts	Parts can exist independently of the whole	Parts cannot exist without the whole
Lifetime	Whole and parts may have different lifetimes	Whole and parts have the same lifetime
Creation/ Destruction	Whole may know parts but doesn't strictly control them	Whole creates and destroys parts automatically

Inheritance

Inheritance is represented by an ***empty triangle arrow*** pointing from the ***subclass → superclass***.

If multiple subclasses share the same base class, their arrows can be merged into one line, labeled with the aspect they specialize in called a discriminator.



Example: A `Book` superclass may have subclasses `IssuableBook` and `ReferenceBook`.

Discriminator: *issuable vs reference*.