UML Diagrams

Fall 2025 - OOP

UML

- Unified Modeling Language
  - Visual representation of a system
  - Shows how different components interact
- Help organize complex ideas into clear and structured visuals.

Two types

- **Structural diagrams**
  - Focus on system architecture
  - Details classes, objects, relationships
- **Behavioral diagrams**
  - How different components interact
  - Captures workflows, use-cases, system responses

Structural diagrams

- Represents the static aspects of the system.
- Allows you to visualize architecture and relationships.
- Maps out system components and how they connect and depend on each other.
    - *Class diagram*
    - *Object diagram*
    - *Component diagram*
    - *Deployment diagram*
    - *Package diagram*

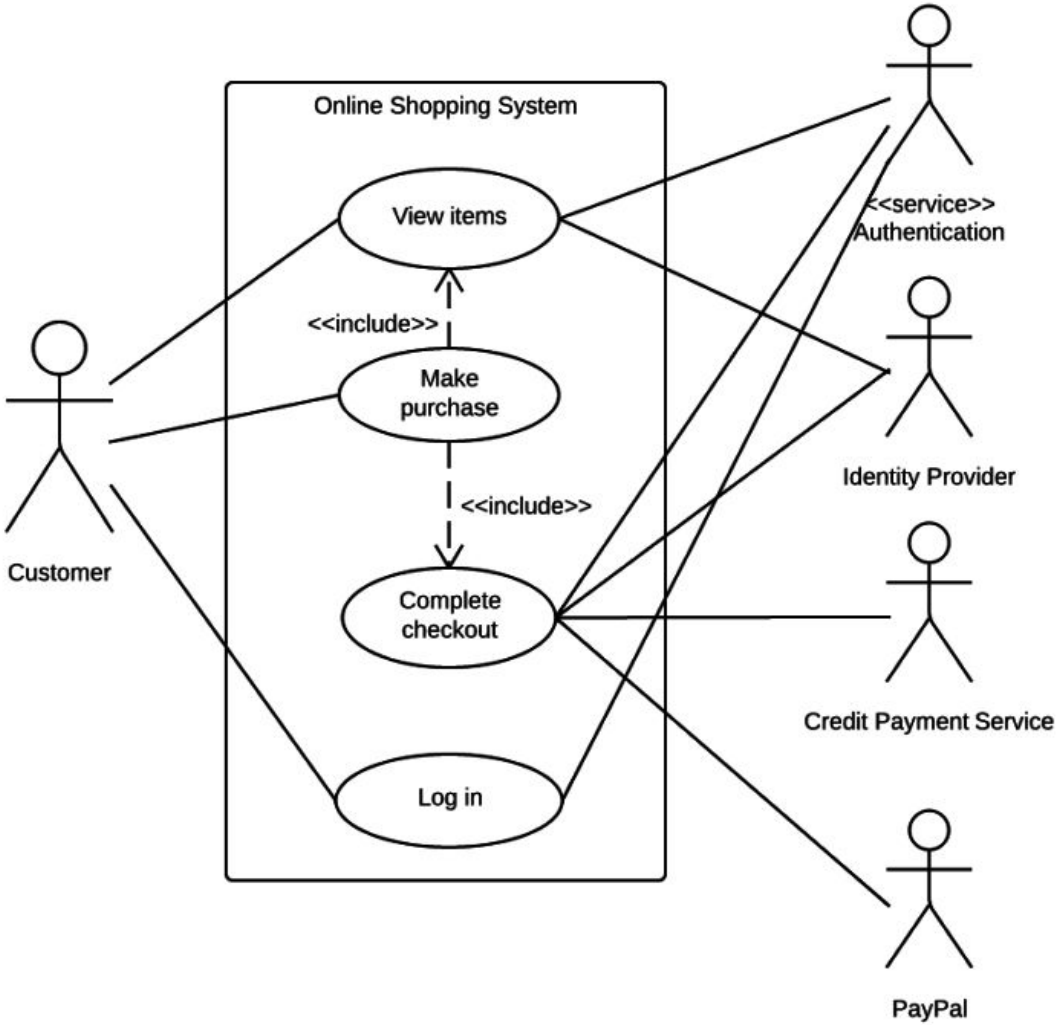
Behavioral diagrams

- Model system interactions
- Show how different elements work together over time.
- Capture
  - workflows
  - communication
  - state changes
- Key types:
  - *use case diagram*
  - *sequence diagram*
  - *activity diagram*
  - *state diagram*
  - *communication diagram*
  - ...

Why?

- Makes the system clearer and more organized.
- **Improve documentation**
  - Visual representation makes it easier to reference system components and workflows.
- **Enhance communication**
  - Developers, designers, stakeholders can quickly align on system functionality.
- Streamline design
  - Identify gaps and inefficiencies early.

Use case


Online Shopping System
View items
<<include>>
Make purchase
<<include>>
Complete checkout
Log in
Customer
<<service>> Authentication
Identity Provider
Credit Payment Service
PayPal

Use case

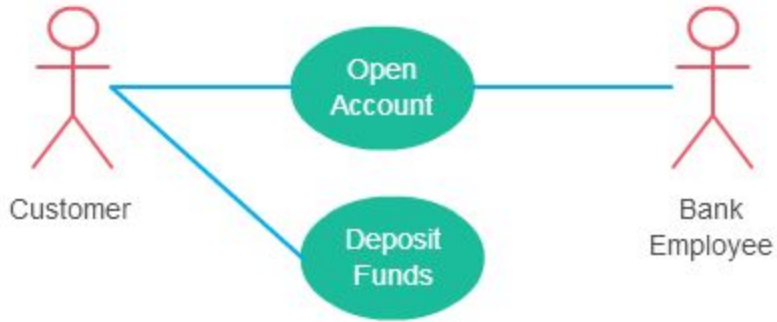
- They answer the following questions:
  - **What** should the system do?
  - **Who** interacts with the system?
  - **Why** does each user interaction occur?
- They do not describe sequence or implementation details.
- They focus on **functional requirements**

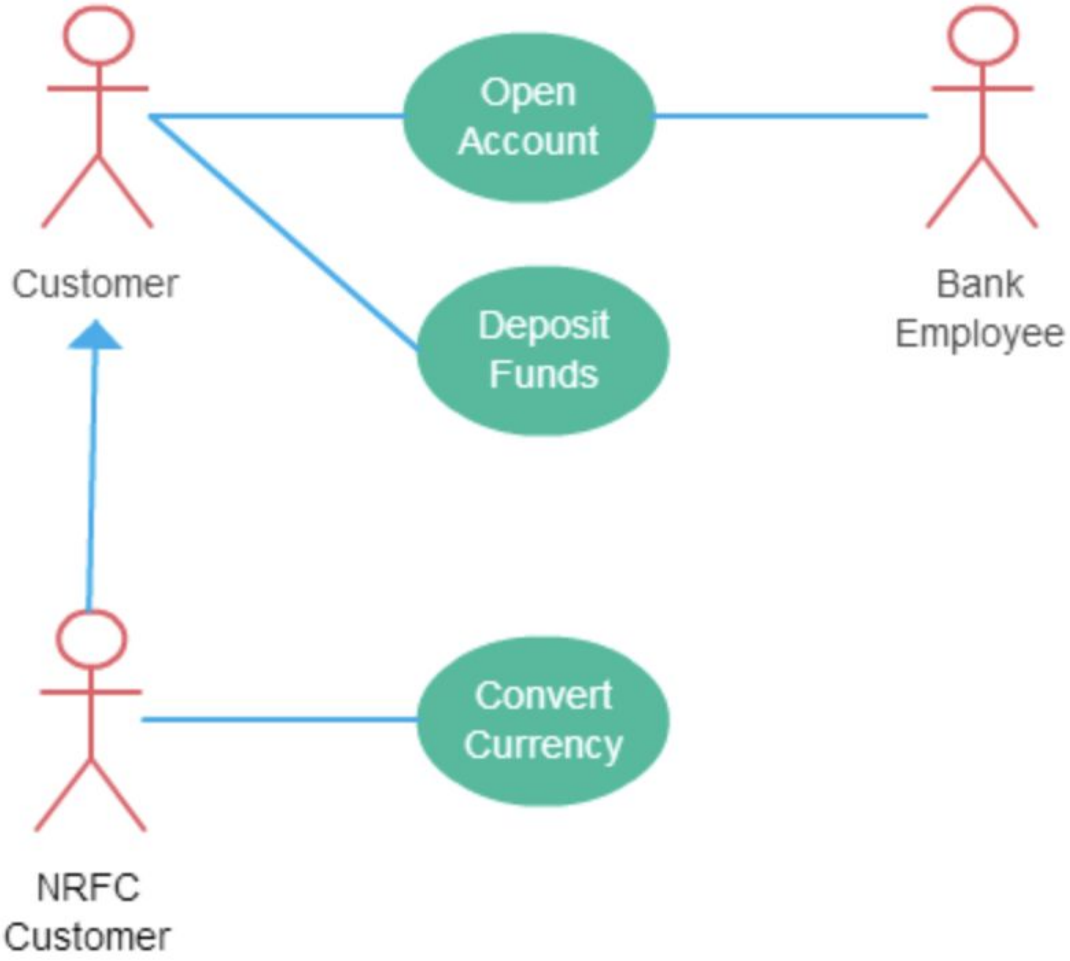
Key elements

- Actor
  - An external entity interacting with the system.
    - Primary actor → initiates use case
    - Secondary actor → supports & provides services
  - e.g. Customer, Admin, Sensor
- Use Case
  - A **goal** the actor wants to achieve through the system.
  - e.g. Login, Place Order, Generate Report
- System Boundary
  - A box that defines the system’s scope

Use case relationships

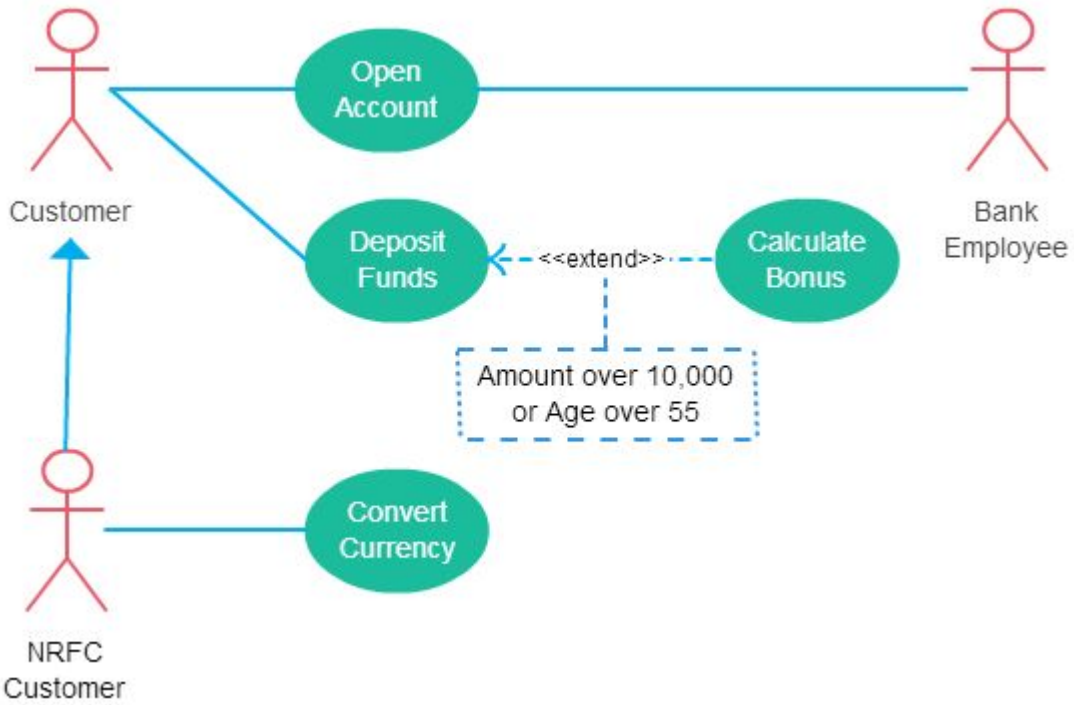
- Include (<<include>>)
  - Used when a use case always uses another use case.
  - e.g. Checkout **includes** Validate Payment
- Extend (<<extend>>)
  - The extension use case is *optional* or triggered by a condition.
  - e.g. Place order **extended by** Apply Discount.
- Generalization
  - Actor or use case inherits behavior from another.
  - e.g. *Registered User* inherits from *User*


Open
Account
Customer
Deposit
Funds
Bank
Employee

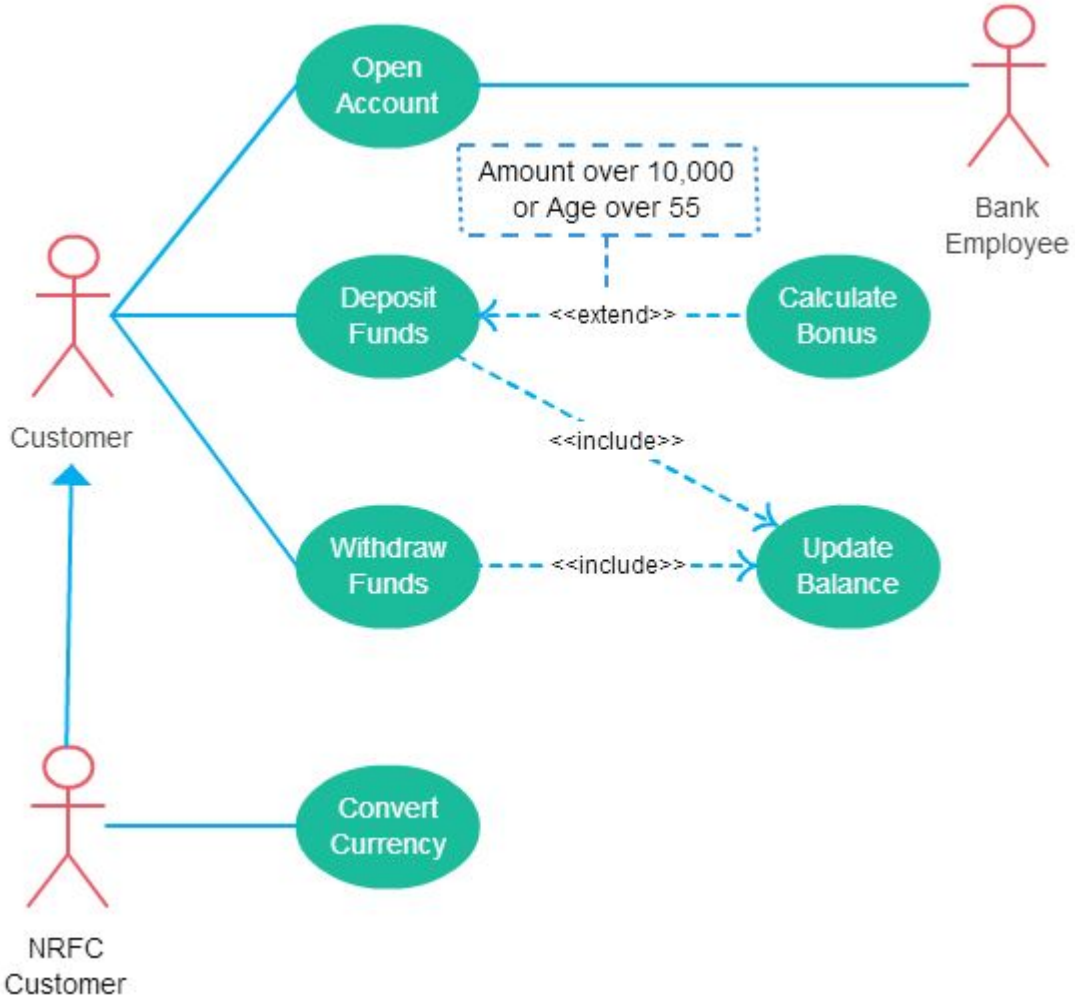


A generalized actor in an use case diagram

- Extending use case is dependent on base case.
- Extending use case is usually optional and can be triggered *conditionally*.
    - e.g. triggered only over 10k or age > 55
- extended (base) case must be meaningful on its own
    - must be independent
    - should not rely on extending.



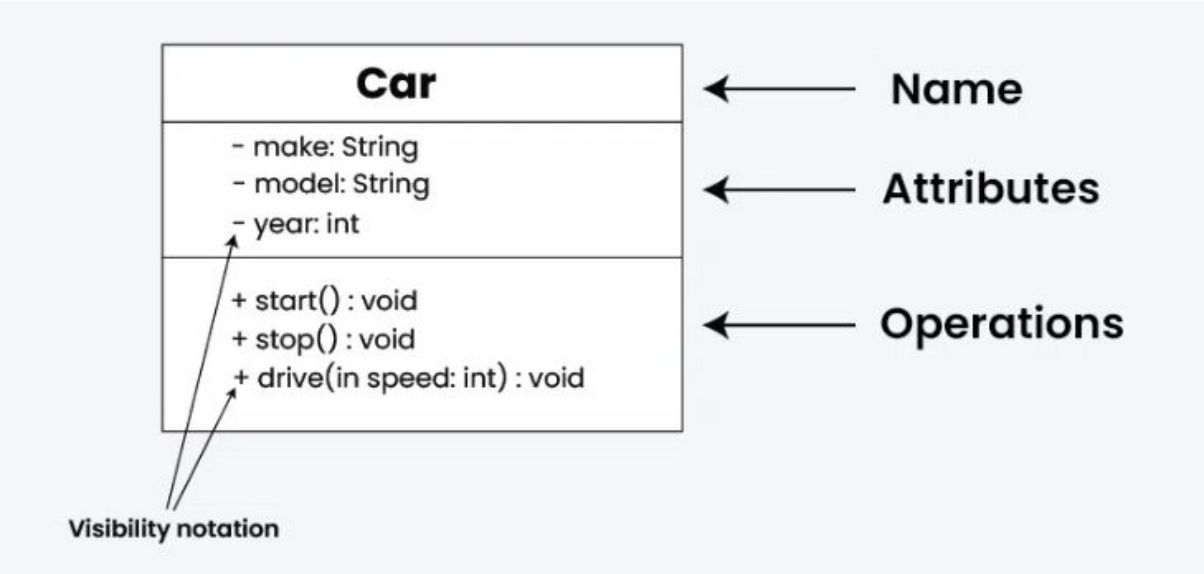
- Include relationship show that the behavior of the included use case is part of the including (base) use case.
- The main reason for this is to reuse common actions across multiple use cases.
- In some situations, this is done to simplify complex behaviors.
- Few things to consider when using the <<include>> relationship.
    - The base use case is incomplete without the included use case.
    - The included use case is mandatory and not optional.



Class Diagrams

- Class diagrams capture the **static architecture** of the system:
  - Classes
  - Attributes (fields)
  - Operations (methods)
  - Relationships (inheritance, association, dependency, etc)

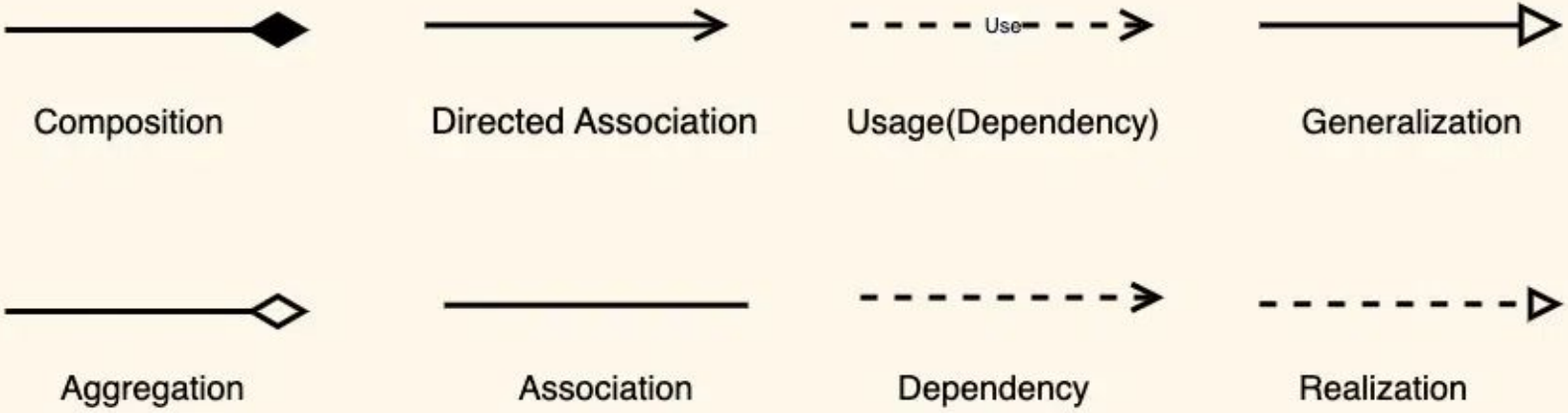
- A UML box has 3 sections.
- - → private
- + → public
- # → protected
- Underline: **static**



Class Relationships

- Association
  - Generic connection between classes
- Aggregation
  - “Weak whole-part” relationship.
  - The part can exist independently.
- Composition
  - Strong whole-part relationship
  - Part cannot meaningfully exist without the whole
- Generalization / Inheritance
  - Superclass-subclass
- Realization
  - Used between interfaces and classes that implement them
- Dependency
  - Class temporarily depends on another

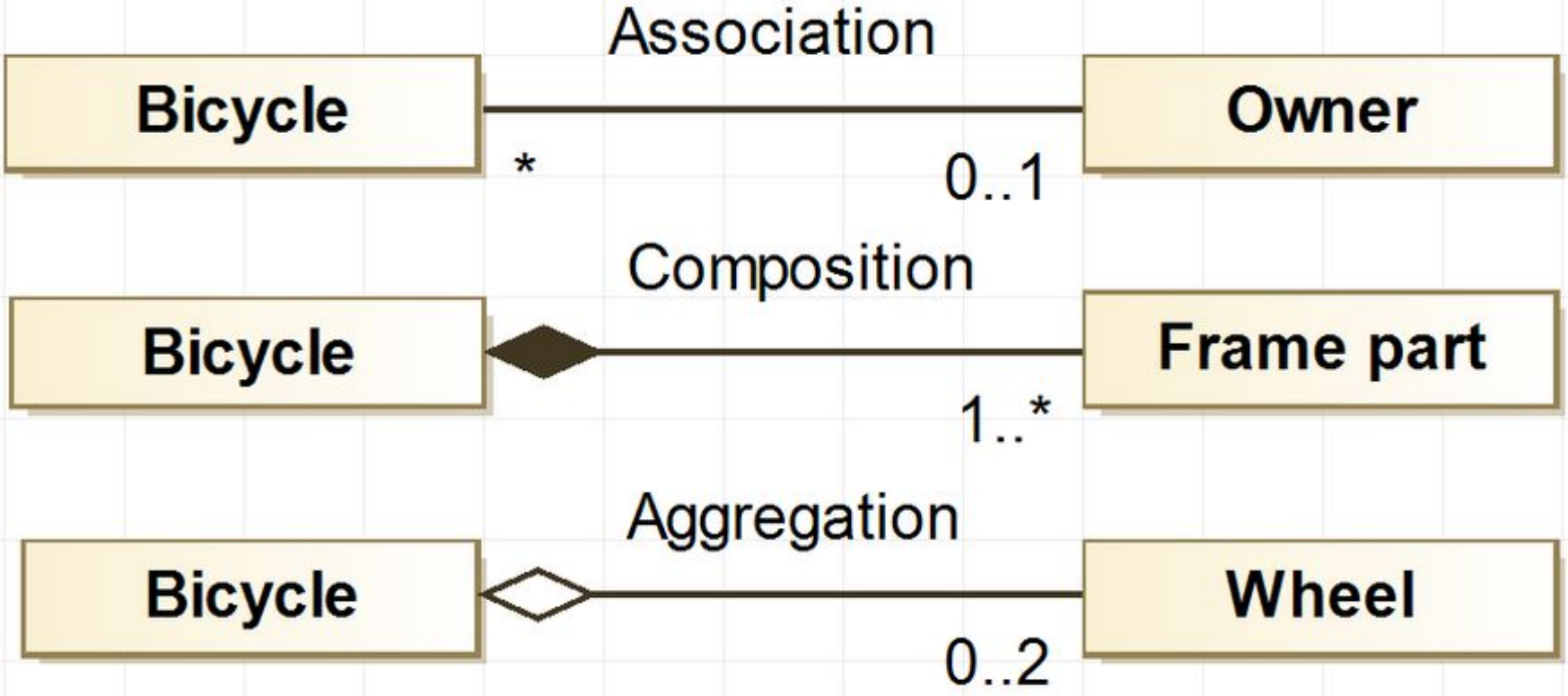
Class Diagram Relationships

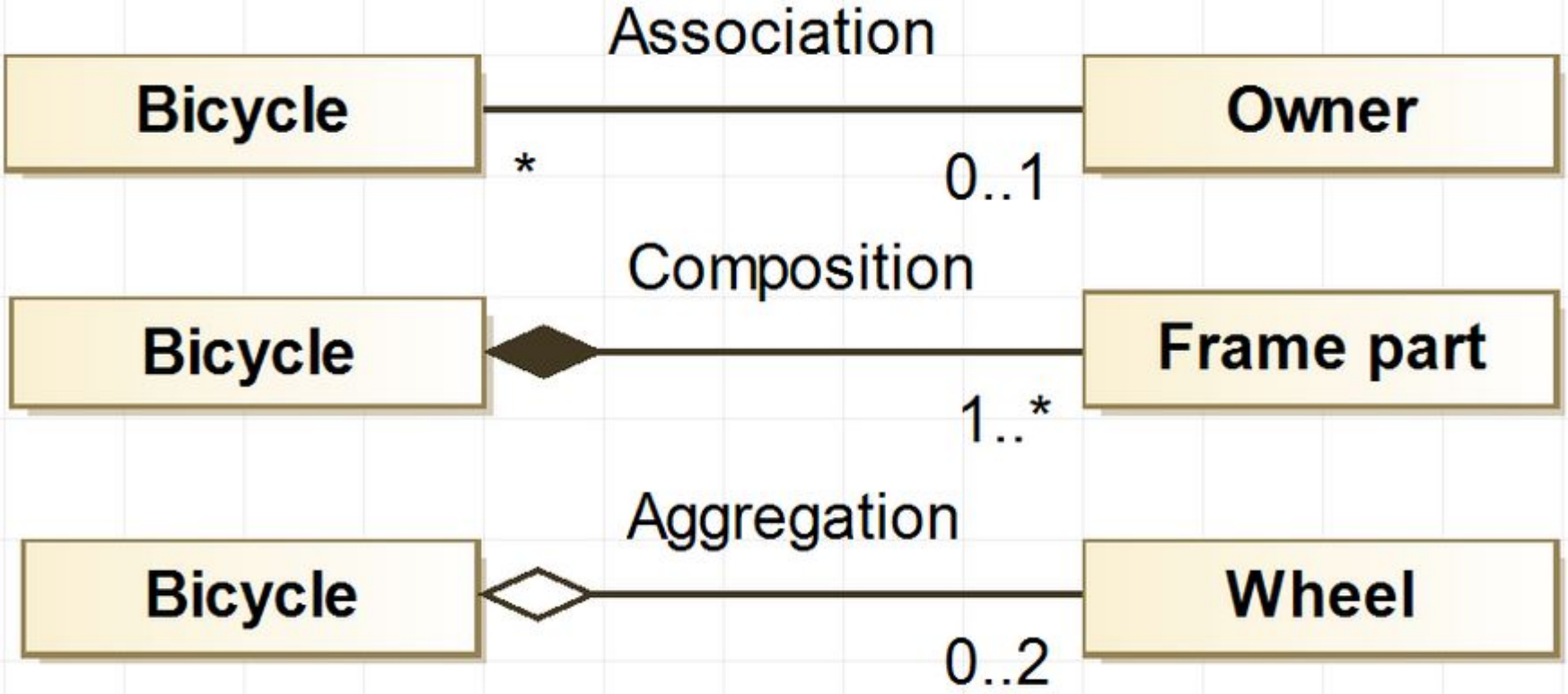


Association (→)

- Generic connection between classes.
- **1** :exactly one
- **0..1** : optional
- ***** : many
- **1..*** : one or more
  - Example
    - Customer 1..* → Order

- **Aggregation (○—)**
  - Team ○— Player
    - Deleting the team does **not** delete the players.
- **Composition (◆—)**
  - House (◆—) Room
  - Delete the house → rooms go with it.
- **Generalization / Inheritance (▷)**
- **Realization ⇢**
  - Used between interfaces and classes that implement them.
  - IReadable ⇢ Book
- **Dependency →**
  - A class temporarily depends on another
  - ReportGenerator → uses → DatabaseConnection

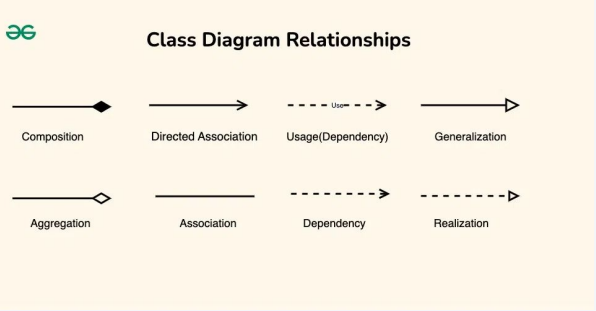

Association
Bicycle
Owner
*
0..1
Composition
Bicycle
Frame part
1..*
Aggregation
Bicycle
Wheel
0..2



if a bicycle is destroyed you can still have wheels

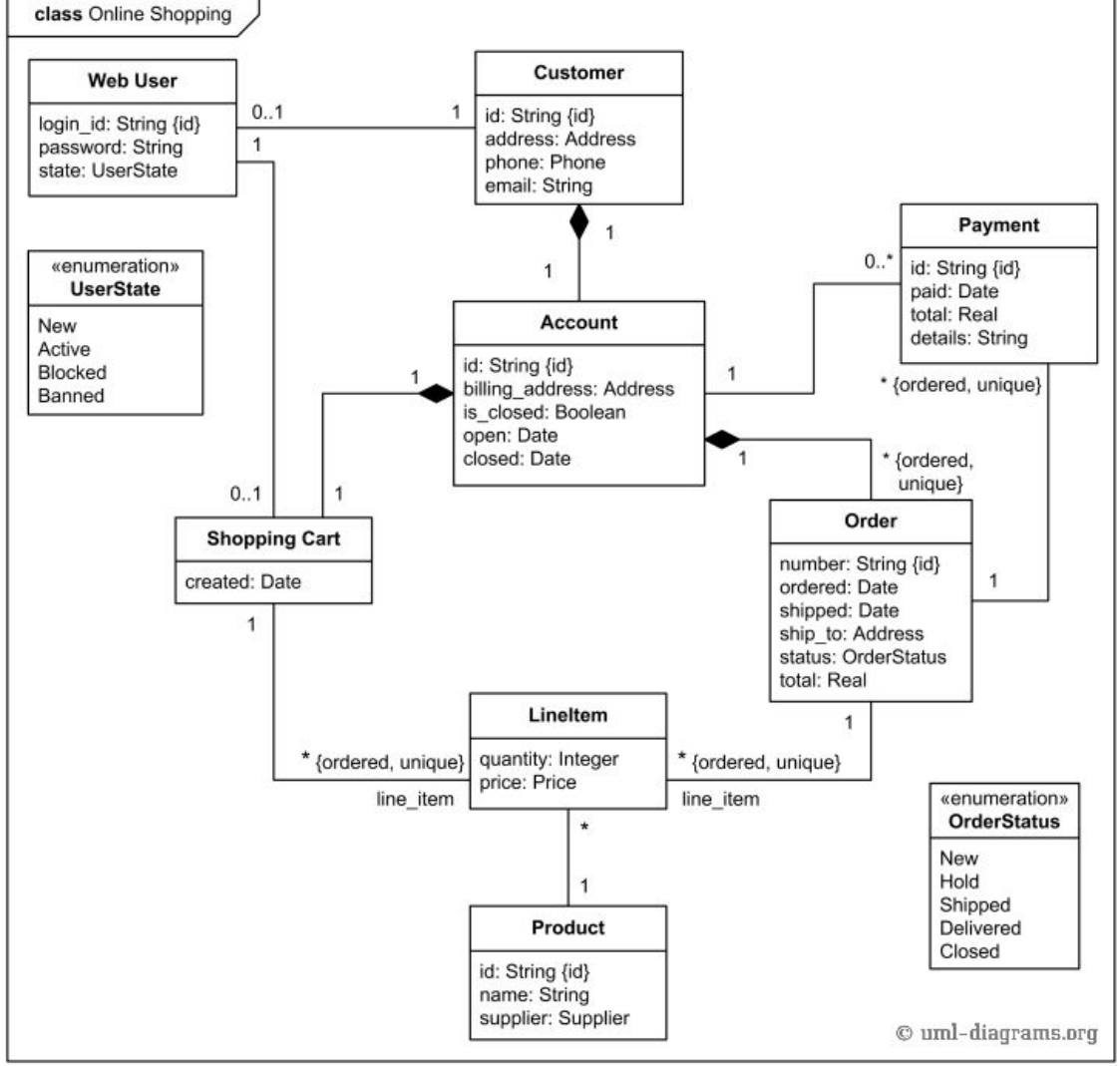
Composition vs Aggregation

- Aggregation
  - A team **has** players
  - Even something happens to the team, players are still there.
- Composition
  - House **has** rooms
  - Something happens to the house, there are no rooms.



- - → private
- + → public
- # → protected
- Underline: **static**

- **Aggregation (○—)**
  - Team ○— Player
    - Deleting the team does **not** delete the players.
- **Composition (◆—)**
  - House (◆—) Room
  - Delete the house → rooms go with it.
- **Generalization / Inheritance (▷)**
- **Realization ⇢**
  - Used between interfaces and classes that implement them.
  - IReadable ⇢ Book
- **Dependency →**
  - A class temporarily depends on another
  - ReportGenerator → uses → DatabaseConnection



- **Web User, Customer**
  - A customer may have 0 or 1 web accounts
  - A web user *must* be linked to 1 customer.
- **Web User, ShoppingCart**
  - A webuser may have a shopping cart.
  - A shopping card must be linked to a webuser.
- **Customer, account**
  - Every customer have exactly one account.
  - Every account belongs to one customer
- **Account, Shopping cart**
  - Every account **has** a shopping cart.
  - If account is gone, cart is gone.
  - Every cart belongs to one account.
- **Shopping cart → LineItem**
  - A cart contains many line items
  - Line items have specific order and unique.
- **LineItem, Product**
  - Each lineitem refers to a single product
  - A product can appear in many line items
- **Account, order**
  - Account **has** order, many (ordered and unique)
  - Each order belong to **one** account
  - If account is gone, order is gone.
- **Order, LineItem**
  - Order can contains multiple lineitems
- **Account, Payment**
  - An account may have a payment, or multiple payments