# UML-I

Manas Jyoti Das, PhD
Computer Science

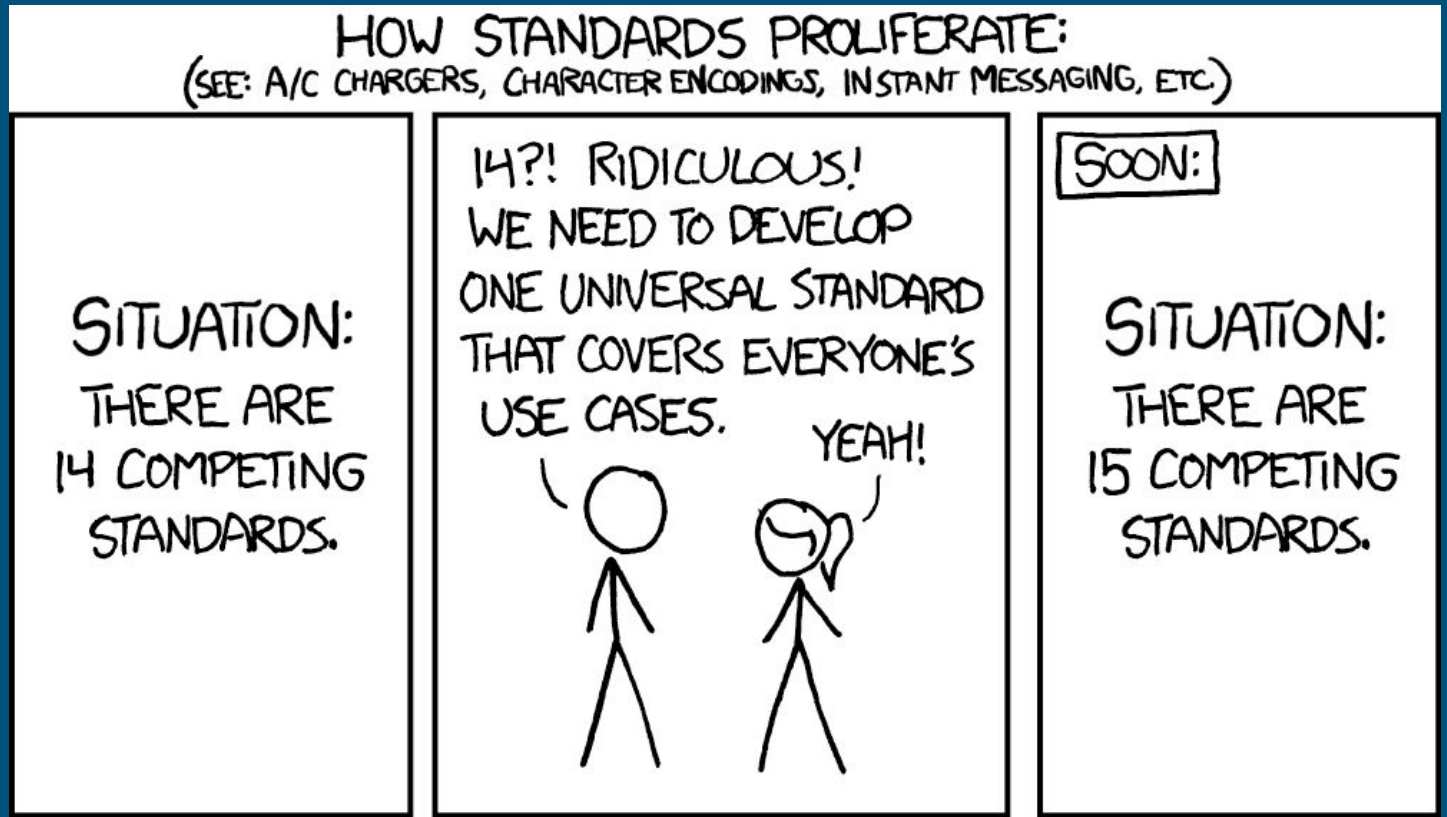
# UML

- Unified Modelling Language
- It is used to visualize, construct, and document the artifacts of software systems
- Artifacts means:
  - Code
  - Documentation
  - Configuration file
  - Requirement document
  - etc.
- Graphical symbols with some rules will be used to construct the diagrammatic representation of the software

# Unified?

- OMT [Rumbough 1991]
- Booch's methodology [Booch, 1991]
- OOSE [Jacobson, 1992]
- Odell's methodology [Odell, 1992]
- Shlaer and Mellor [Shlaer, 1992]

- UML used symbols and rules from all these techniques and also proposed some new symbols and rules
- Large portion of UML is similar to OMT


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

# OMG!!

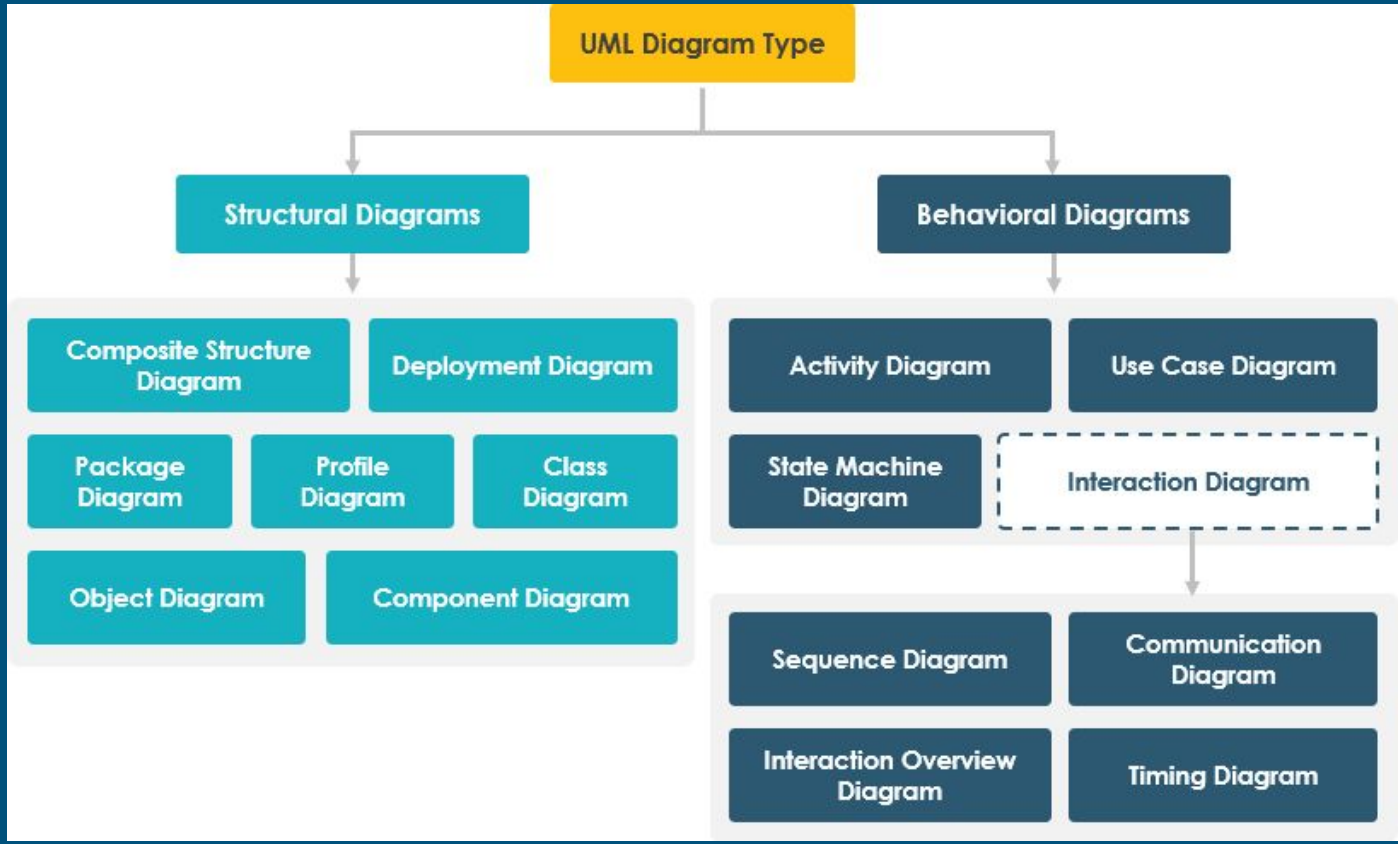
- As a standard was adopted in 1997 by OMG (Object Management Group)
- OMG is a consortium of industries
- Current version is UML 2.5.x

# Why modelling?

- It is a technique by which we can abstract the software
- To have a meaningful understanding of what the system will do
- Instead of reading many pages of requirement and how the software will interact, easy to understand a graphical representation
- Difference between modelling and design:

| | Characteristic | Model | Design |
|---|---|---|---|
| 2 | Level of abstraction | High | Low |
| 3 | Level of detail | Low | High |
| 4 | Purpose | To understand and communicate the system's requirements and functionality | To provide a detailed plan for how to implement the system |
| 5 | Audience | Stakeholders, such as users, business analysts, and developers | Developers |

# Various UML Diagrams



# Structural Diagrams

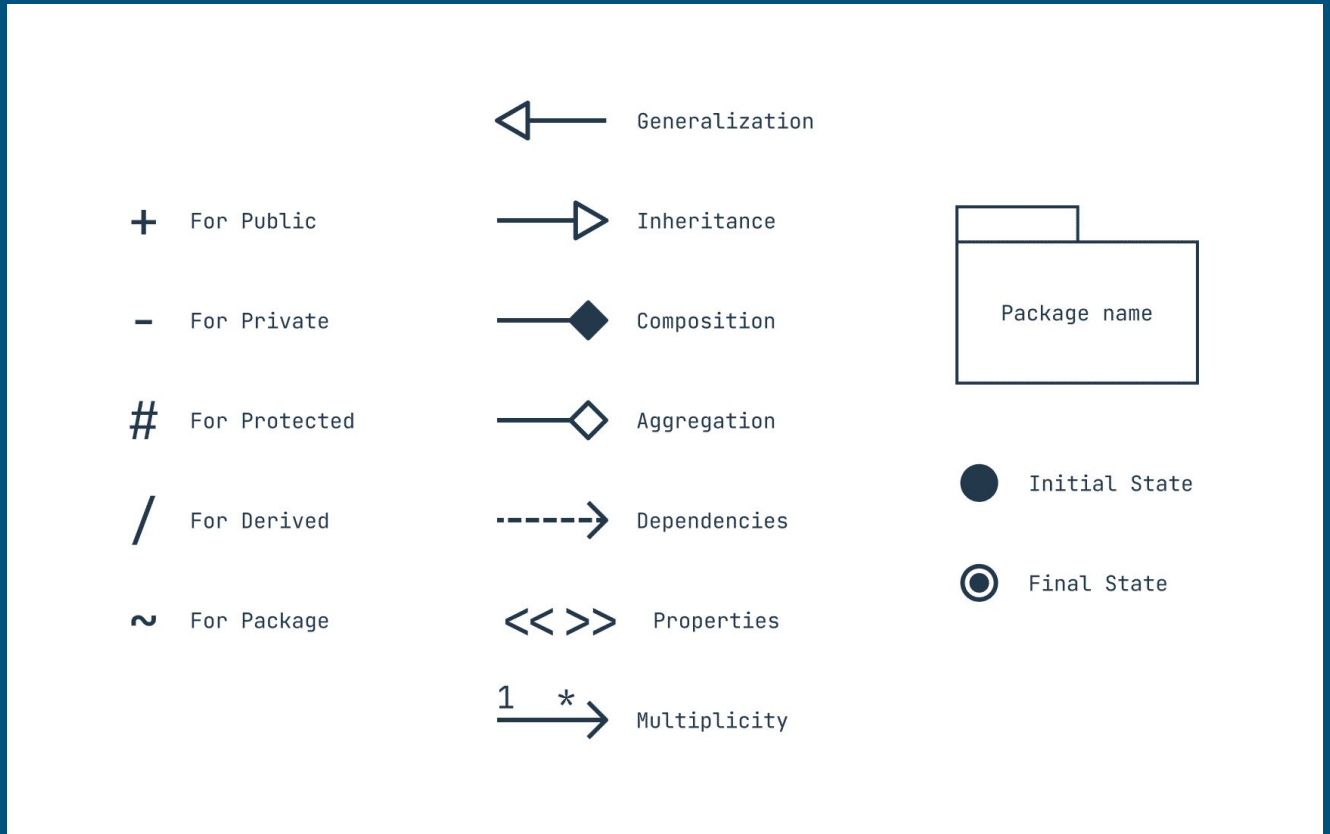
- These diagrams depict the static structure of a system, focusing on its components and their relationships. They answer questions like "What are the building blocks of the system?" and "How are they connected?". Here are some key types of structural diagrams:
    - Class Diagram: Shows classes, their attributes, methods, and relationships (inheritance, association, aggregation, composition).
    - Component Diagram: Represents physical components of a system and their dependencies.
    - Composite Structure Diagram: Depicts the internal structure of complex classes or components.
    - Deployment Diagram: Illustrates the physical deployment of software components on hardware nodes.

# Behavioral Diagrams

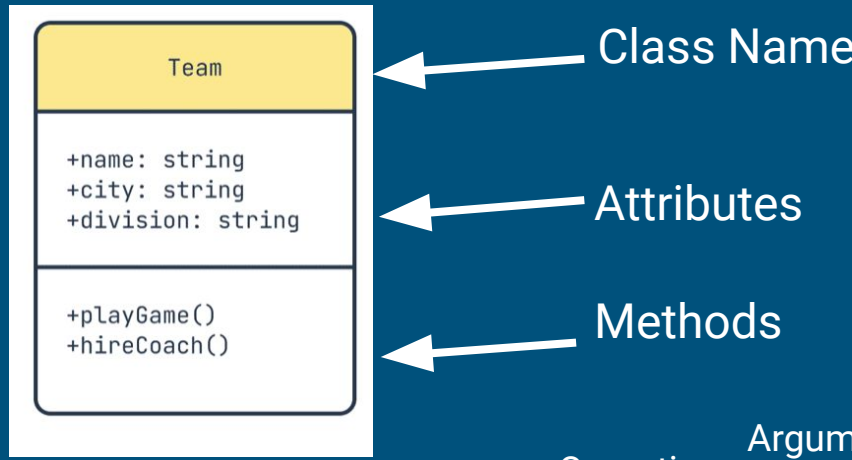
- These diagrams illustrate the dynamic behavior of a system, showing how objects interact and change over time. They answer questions like "How does the system react to events?" and "What sequences of actions take place?". Here are some key types of behavioral diagrams:

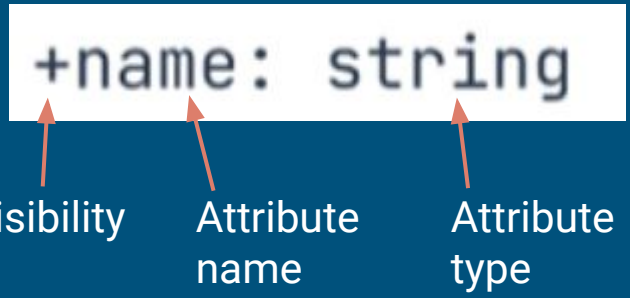
    - Sequence Diagram: Shows the chronological order of messages exchanged between objects in a specific scenario.
    - Collaboration Diagram: Similar to a sequence diagram, but emphasizes object roles and collaborations.
    - State Machine Diagram: Illustrates the different states an object can be in and the transitions between them.
    - Activity Diagram: Models the flow of activities within a system or subprocess.
    - Interaction Overview Diagram: Provides a high-level overview of how interaction diagrams relate to each other.

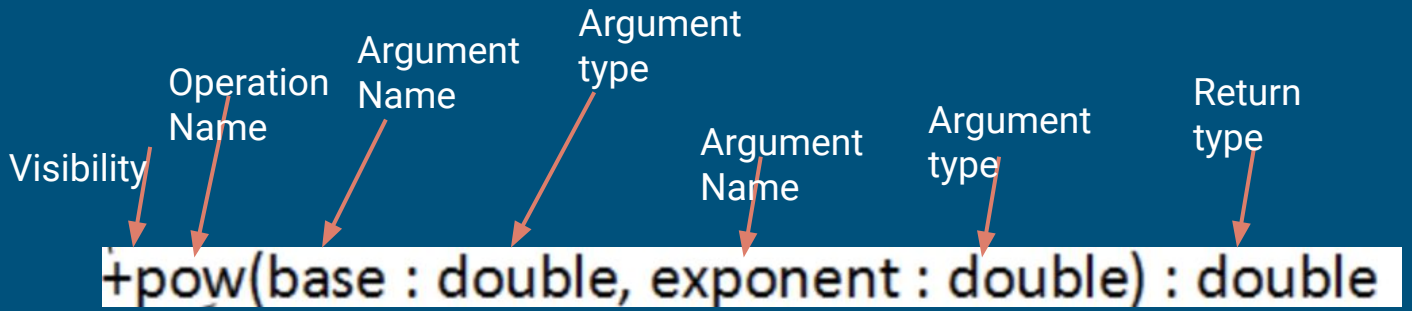
# Symbols


Generalization
+ For Public
Inheritance
- For Private
Composition
# For Protected
Aggregation
/ For Derived
Dependencies
~ For Package
<< >> Properties
1 *
Multiplicity
Package name
Initial State
Final State

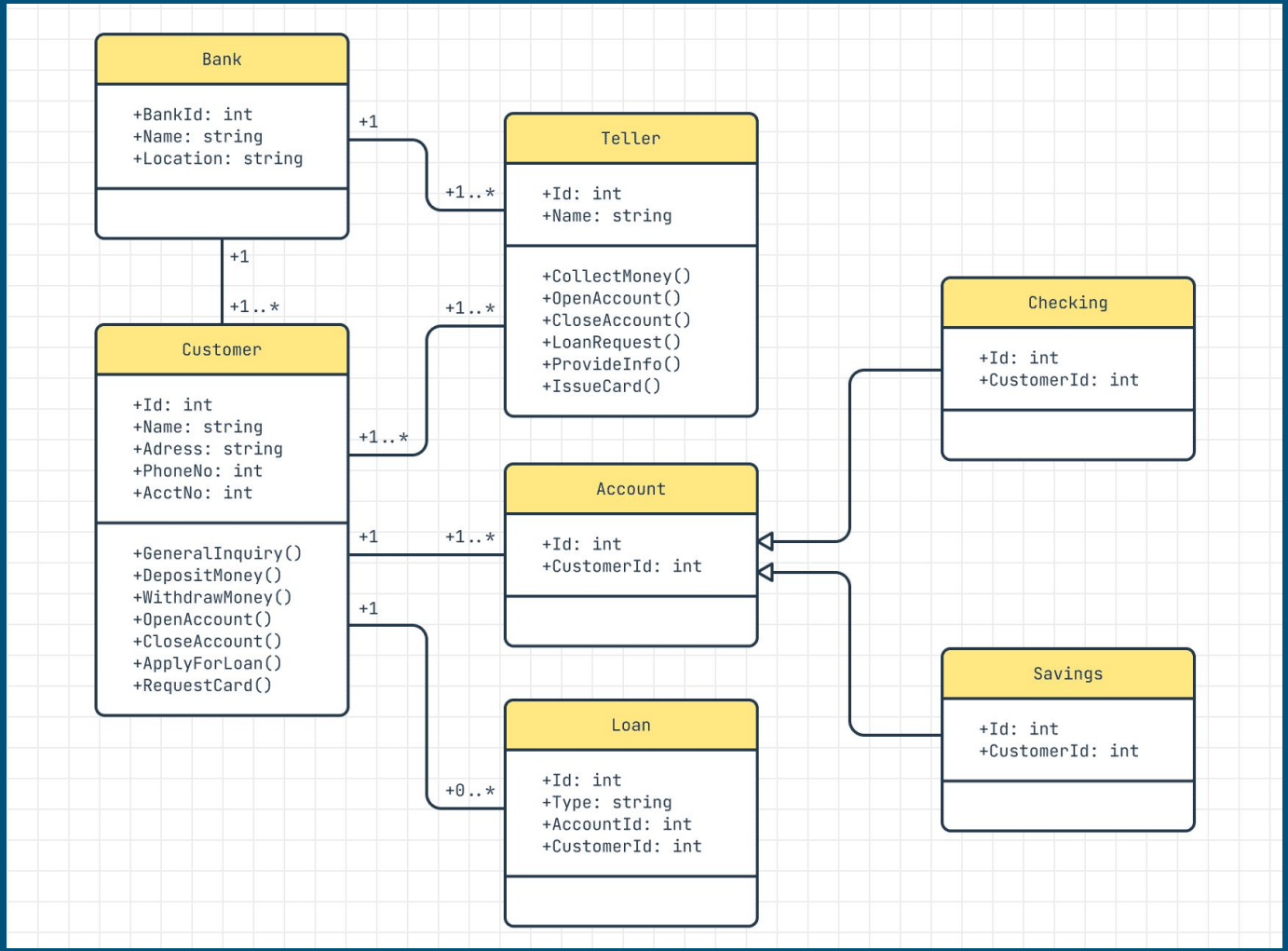
# UML Class Diagram



# UML Class Diagram




0..1 = zero to one

1 = exactly one

0..* = zero to many

1..* = one to many

n..m = specific number range

# UML Class Diagram

- Inheritance: a.k.a generalization, this is a relationship where a subclass inherits attributes, methods and other relationships from a superclass.
  - A solid line with an arrow head pointing to the parent class. It represents an "is-a" relationship between two classes
- Association: this is a relationship that indicates one class is related to another but not necessarily dependent
- Aggregation: A special type of association, aggregation is where one class contains, or is composed of, other classes. A whole and its parts
- Composition: On the flipside is composition, where a part cannot exist without the whole, or superclass
- Dependency: This is a relationship where one class relies on another in some way, often through method parameters, return types or temporary associations

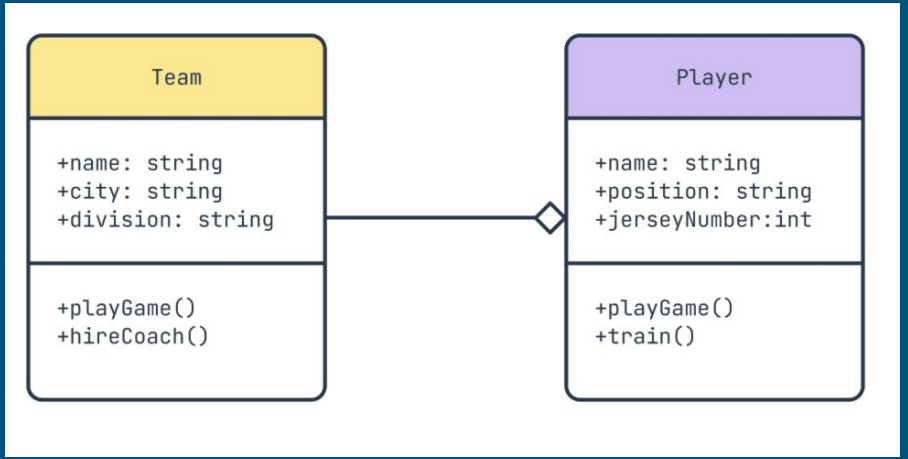
# Aggregation and Composition

- Part-of relationship (Aggregation)
  - A school has students
  - A company has employees
  - A website has pages
  - Note: This type of aggregation relationship is useful for modeling parts of an object that are important for the object to function, but are not essential for the object to exist. For example, a library can exist without any books, but it is less useful without them
- Part-of exclusive (Composition)
  - A house has rooms
  - A computer has a motherboard, CPU, and RAM
  - A human has a heart, lungs, and brain
  - Note: This type of aggregation relationship is useful for modeling parts of an object that are essential for the object to exist. For example, a car cannot exist without wheels

# Aggregation

Composition: Expanding the example further and adding a couple more classes; stadiums and snack bars. If the team's stadium is torn down, the snack bar can't exist

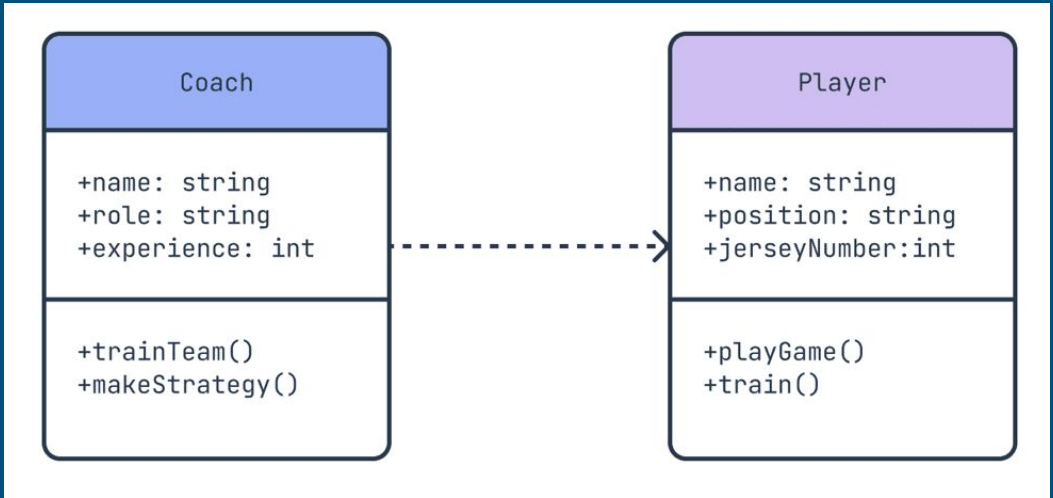
This will be represented by solid diamond shape, as shown below:

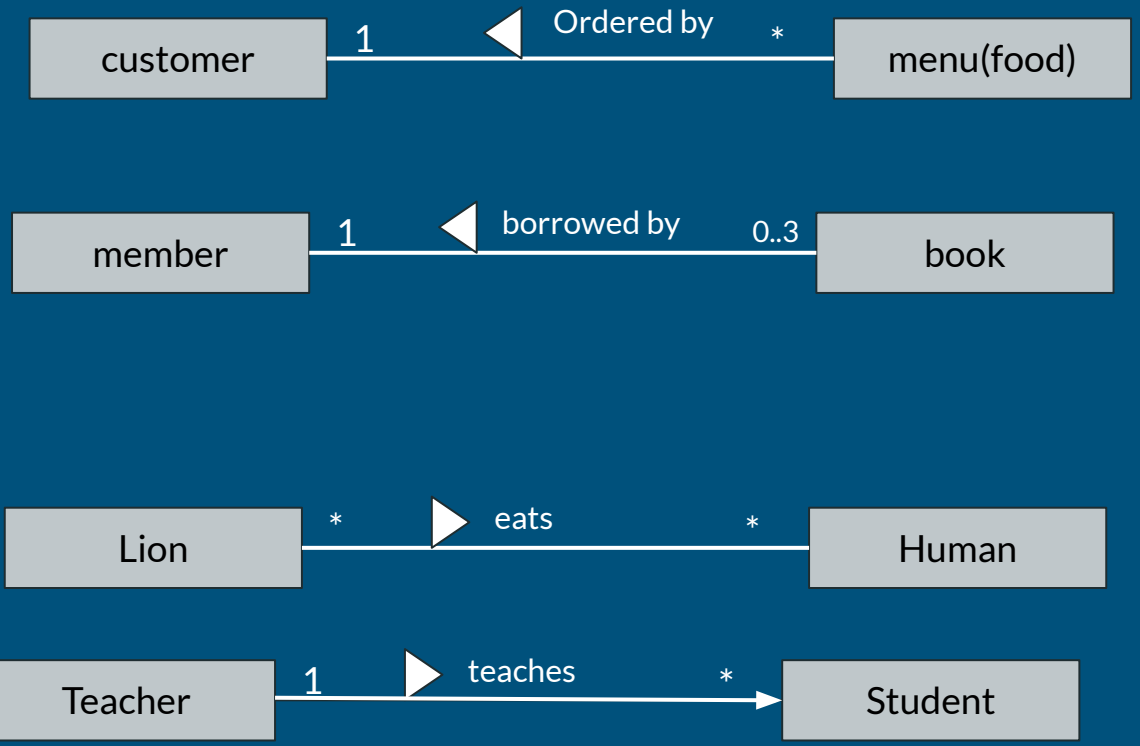

Aggregation

# Dependency

Example: Coaches and players are related in a way, where the coach depends on the performance data or behavior of players to make strategic decisions



# Multiplicity




Multiplicity: The number of objects from one class that relate with a single object in an associated class.

0..1 = zero to one

1 = exactly one

0..* = zero to many

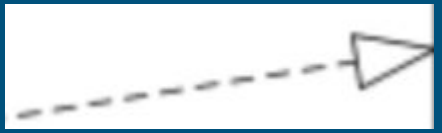
1..* = one to many

n..m = specific number range

# << >> : stereotype

- In UML, the << >> notation refers to a stereotype, which is a way to provide additional meaning to existing UML elements
- It indicates that the corresponding attribute should be implemented using getter and setter methods in your programming language. This promotes encapsulation by controlling how the data is accessed and modified
- In some cases, the stereotype might imply additional custom behavior beyond simple getters and setters

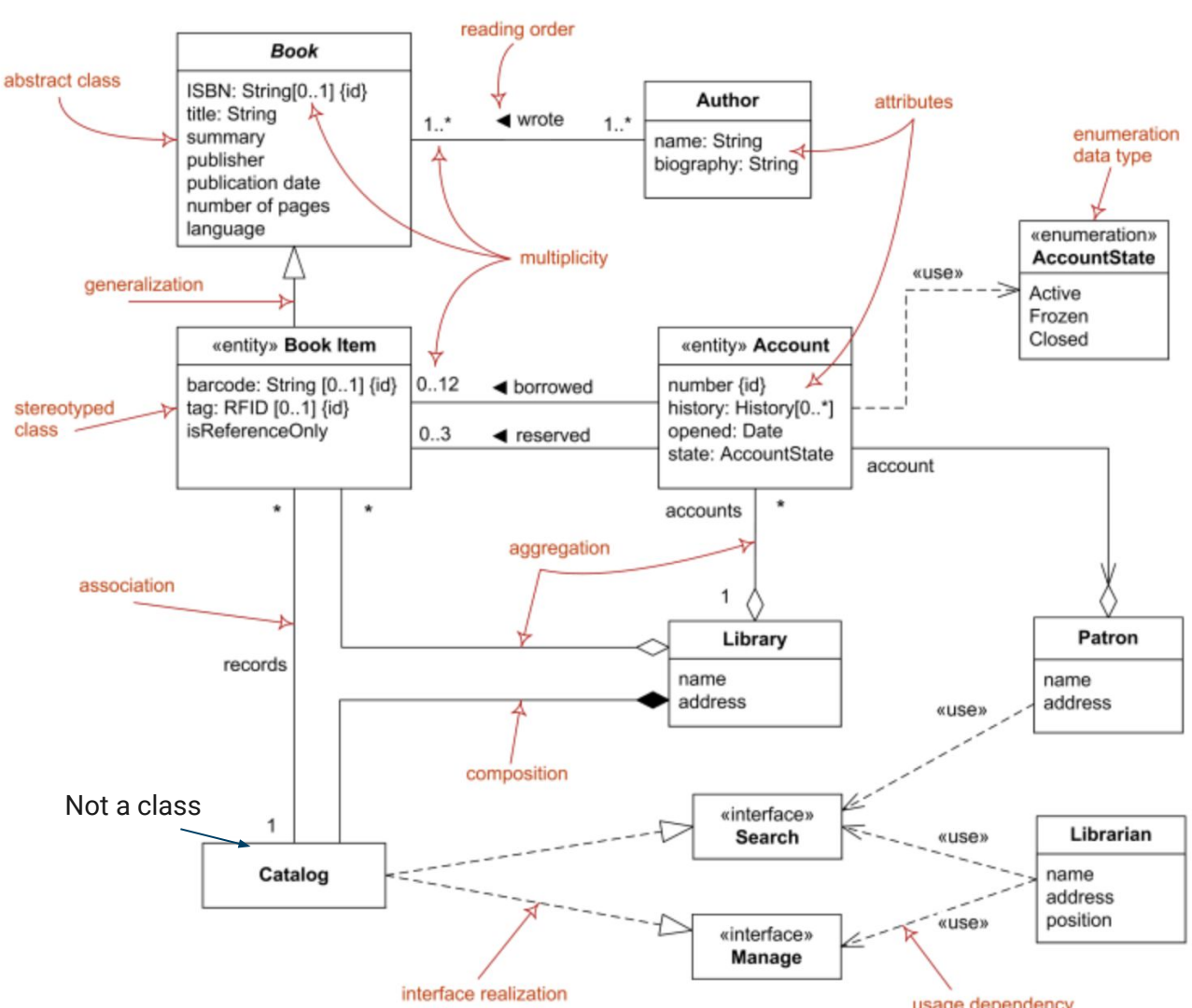
# Realization



- Realization in UML refers to a specific relationship between two model elements, where one element provides the specification (what something should do) and the other element implements the functionality (how it actually does it)
- Implementation focus: The client element is responsible for carrying out the functionality outlined in the supplier element
- No inheritance: While inheritance involves inheriting both attributes and behaviors, realization only focuses on implementing the specified behavior

# Big Picture

Example: Just look at it and try to understand it.



# Lets solve...

cat/dog tax

# Thank you