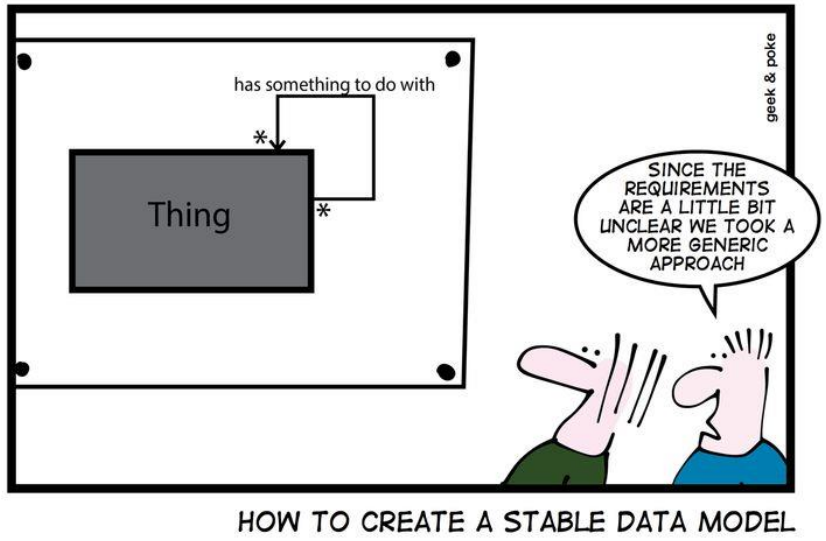
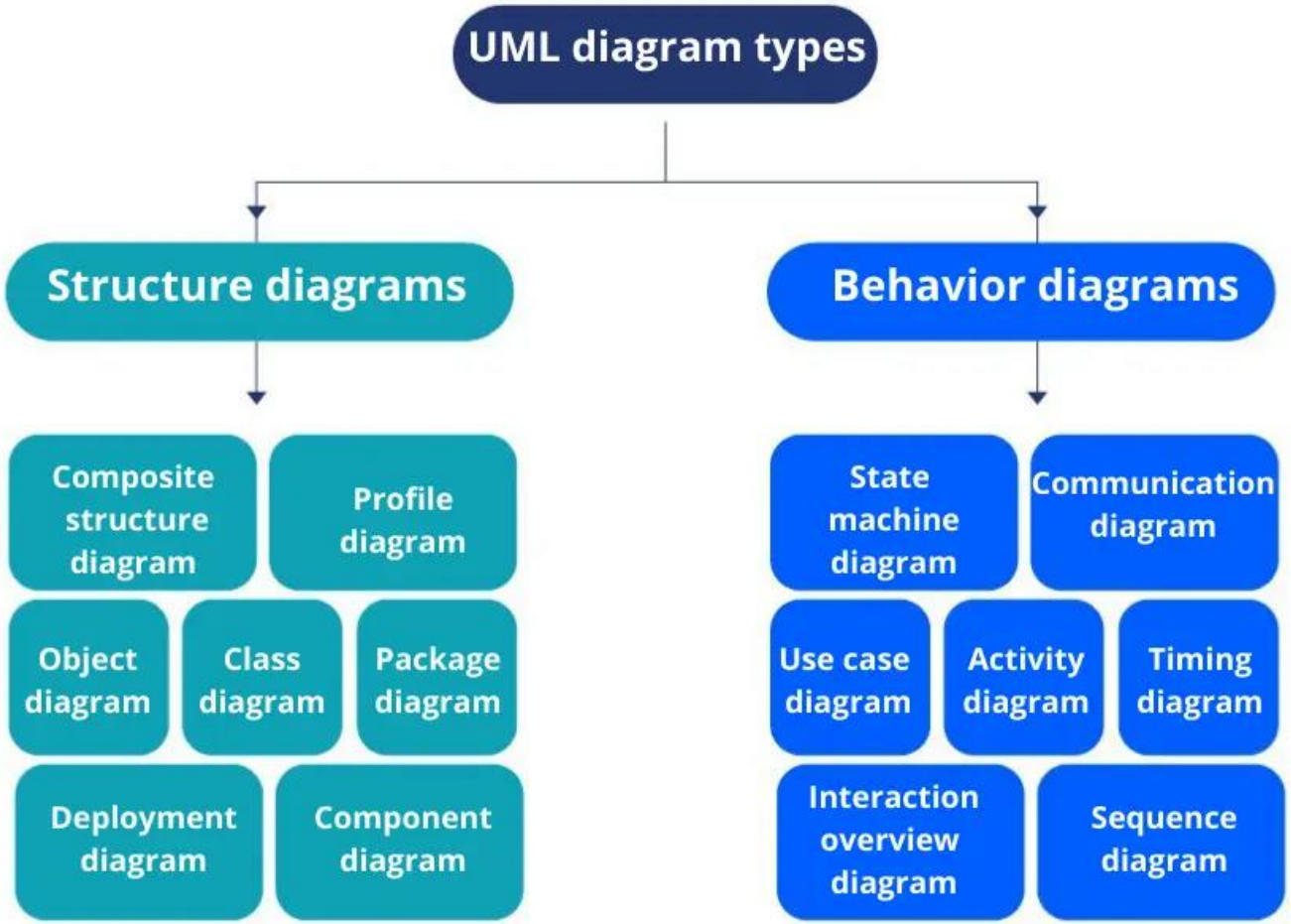
University
Mohammed VI
Polytechnic



Class Diagram

Pr. Imane Fouad

Types of UML Diagrams



Class Diagram

A class diagram describes the structure of a system.

It shows classes, their attributes, operations, and relationships.

Considered the most important UML diagram for modeling.

Foundation for design and implementation.

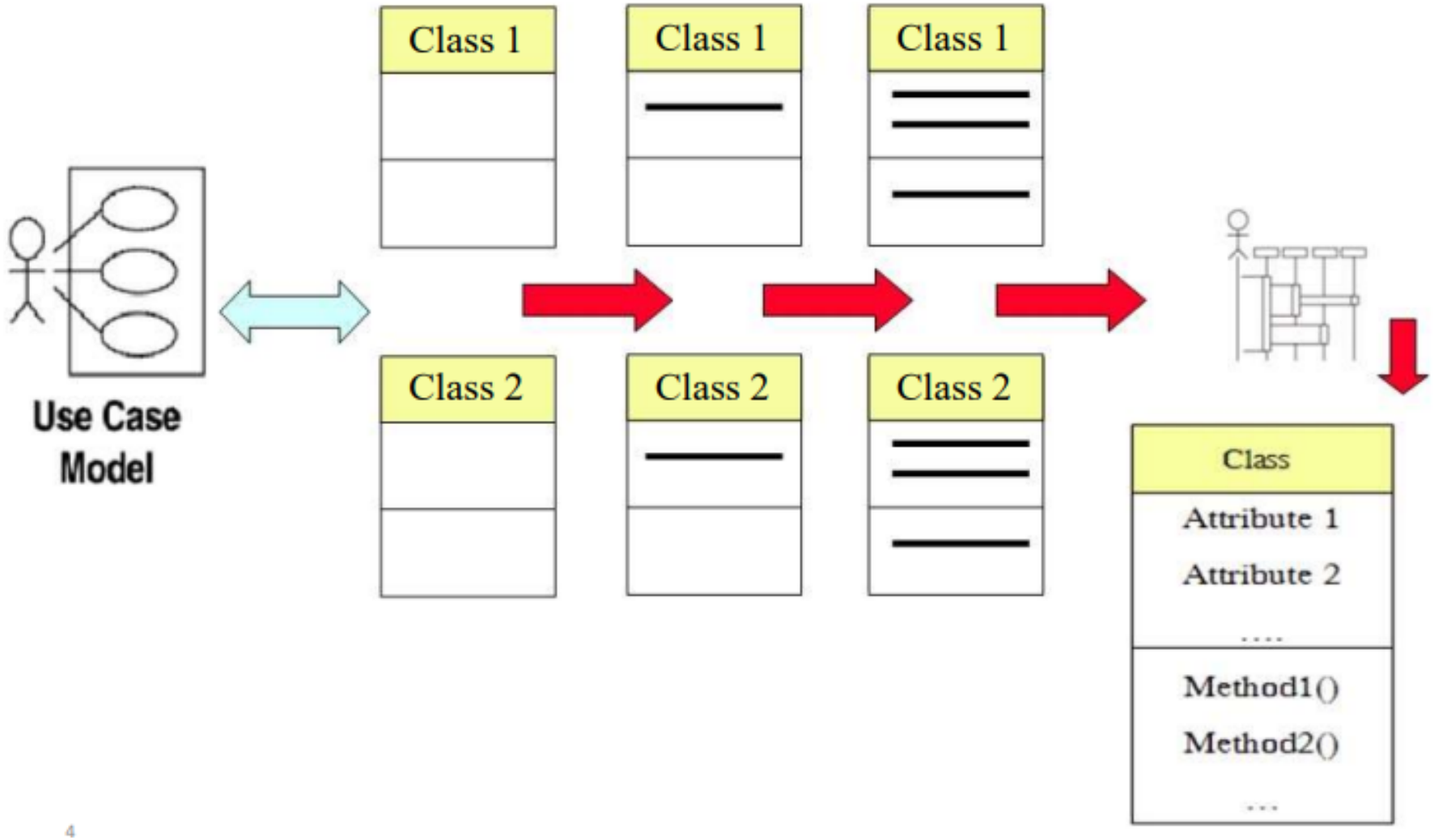
Goal of Class Diagrams

Use case diagrams show *what* the system does.

The system is made of objects that interact with each other and with actors.

Class diagrams specify the structure and relationships of these objects.

From Use Cases to Objects, Attributes & Operations



From Use Cases to Objects, Attributes & Operations

1. Start from **Use Case Descriptions**
2. Look for **Nouns** → **Candidate Classes or Attributes**
3. Look for **Verbs** → **Candidate Operations (Methods)**
4. Analyze **relationships** mentioned between nouns
5. Refine the list → Keep **relevant classes only**
6. Build your **Class Diagram** from the results

From Use Cases to Objects, Attributes & Operations

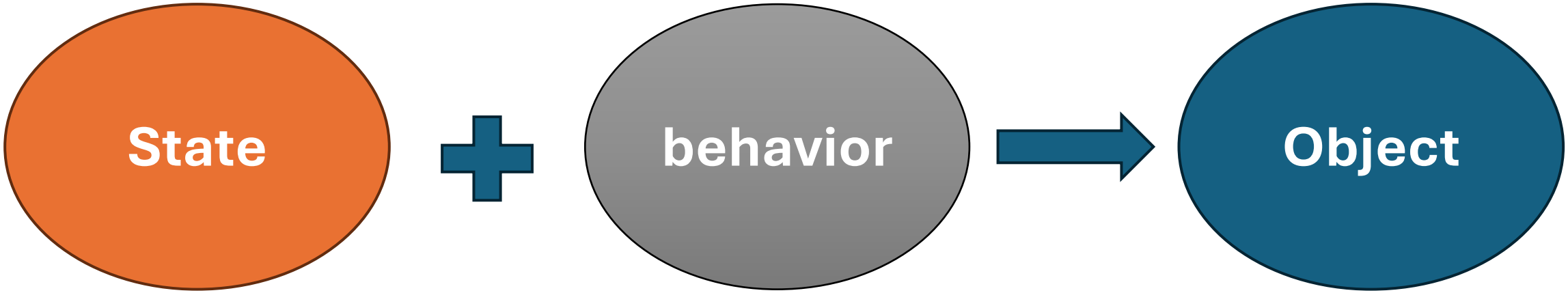
Use Case: **Student enrolls in a Class**

- **Nouns** → Student, Class, Enrollment *(candidate classes)*
- **Verbs** → enrolls *(candidate operation)*
- **Relationships** → Student ↔ Class

Definition of an Object

- According to **Rumbaugh:** “An object is a discrete entity with a well-defined boundary, having both state and behavior.”
- An object represents a real-world entity.
- The **state** of an object = the values of its attributes.
- The **behavior** of an object = the operations (methods) it can perform.
  - These operations often change the object’s state.

Object



What is a Class?

- A class is a **module** from which we create **instances (objects)**.
- A general template that we use to create specific instances or objects in the application domain
- Only the **relevant characteristics** for the studied problem are included.
- Abstractions that specify the attributes and behaviors of a set of objects

A class is an abstraction of a real-world thing with common properties and behaviors.

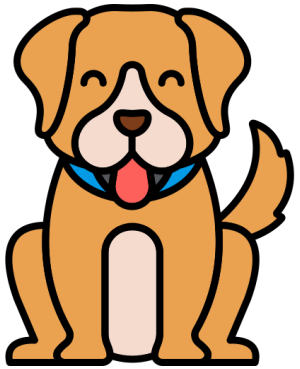
Classes and Objects

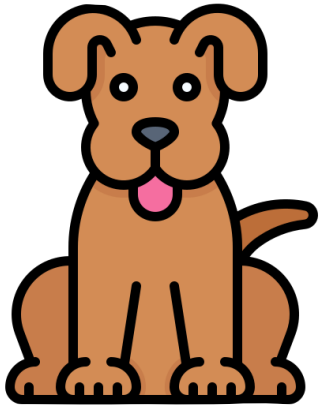
- An instance is the realization of an abstract concept.

- Example:
  - Concept: Pen
  - Instance: the pen you are using now (with its own color, shape, wear).

- An object is an instance of a class.
  - A class describes the common structure of its objects (title, author, label, etc.).

Classes and Objects





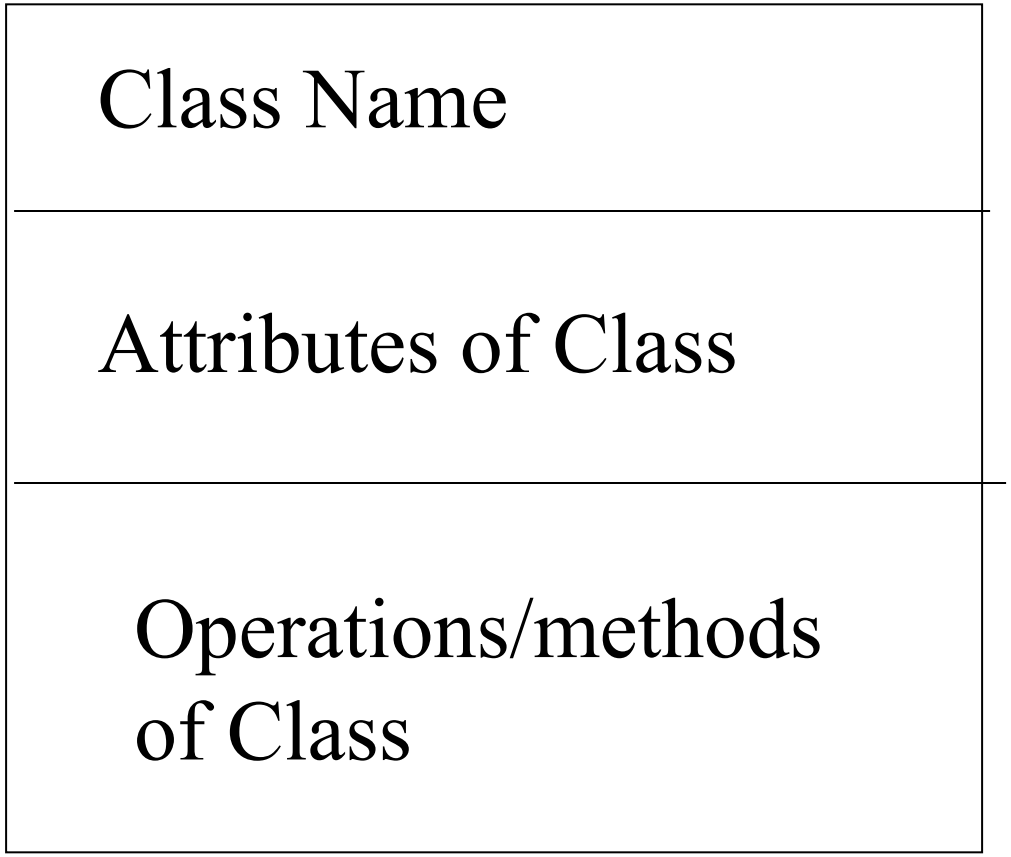
Abstraction

Dog
age
run()

UML Representation of Class

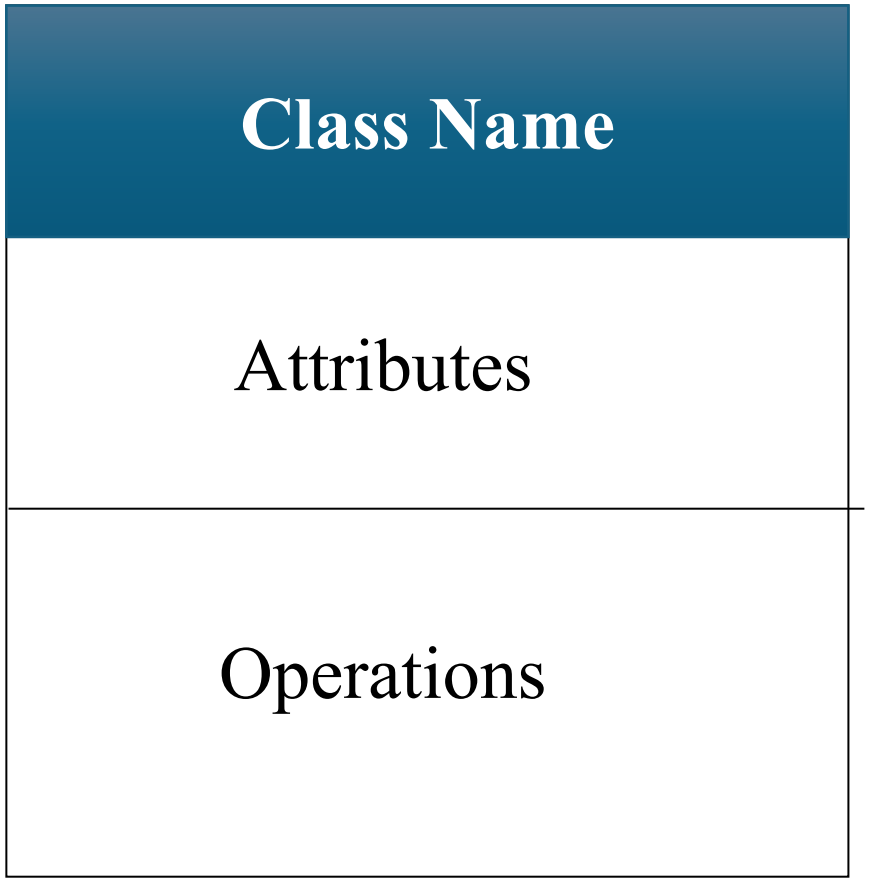
A class is shown as a **rectangle divided into 3 parts**:

- **Class name** (top section)
- **Attributes** (middle section)
- **Operations / methods** (bottom section)



Class Names

The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.



UML Class Notation – Class Names

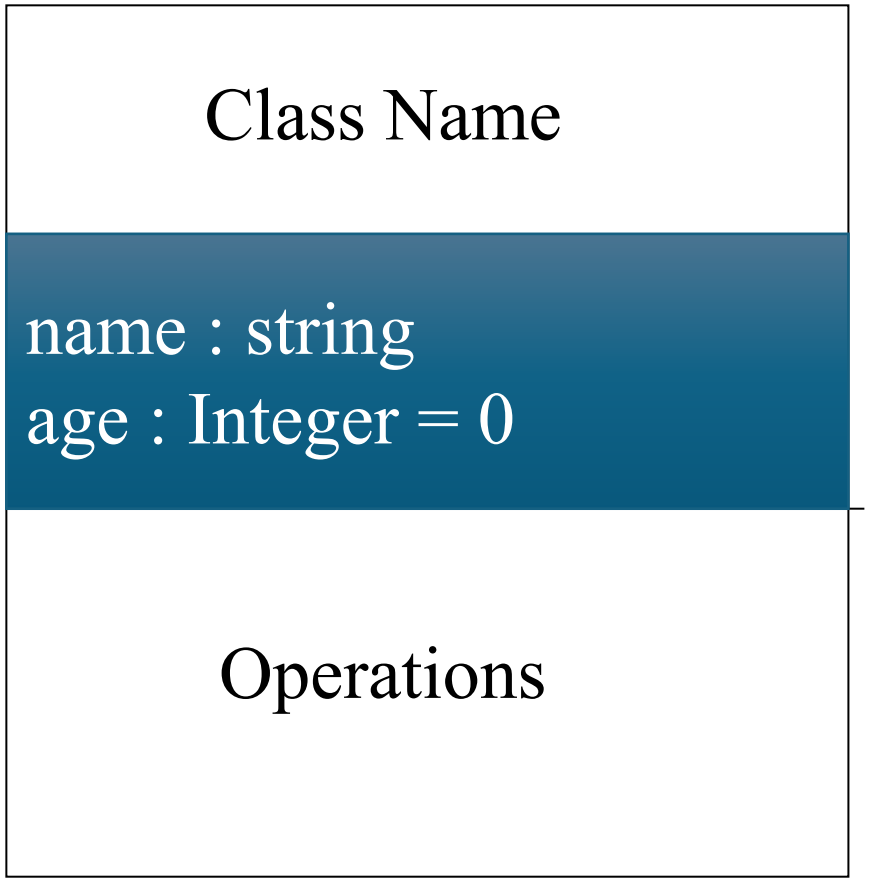
✅ Use **UpperCamelCase** convention.
The first letter of each word is uppercase.
Example: Agent, Account, InvoiceLine, ValidPostalOrder.

🚫 **Avoid abbreviations.**
Use full, clear names.
Example:
- ✅ DetailedValidInvoice
- ❌ InvoiceVD

🚫 **Do not use verbs** for class names.
Classes represent **things** (objects or concepts), not **actions**.
Example:
- ✅ Customer, Payment
- ❌ Calculate, ManageOrder

Class Attributes

- Attributes = **properties or data** of a class.
- Each attribute can have a **type** that defines the kind of values it can hold.
- Optionally, an attribute can have a **default/initial value**.
- Placed in the **middle section** of the UML class rectangle.



UML Attributes

- The type of an attribute can be a simple type:
  boolean, byte, char, double, float, int, long, or short.

- The type can also be complex, meaning it can be another class.

- The initial value indicates the value of the attribute when an instance of the class is created.

Syntax: *visibility name: type[multiplicity] = initialValue*

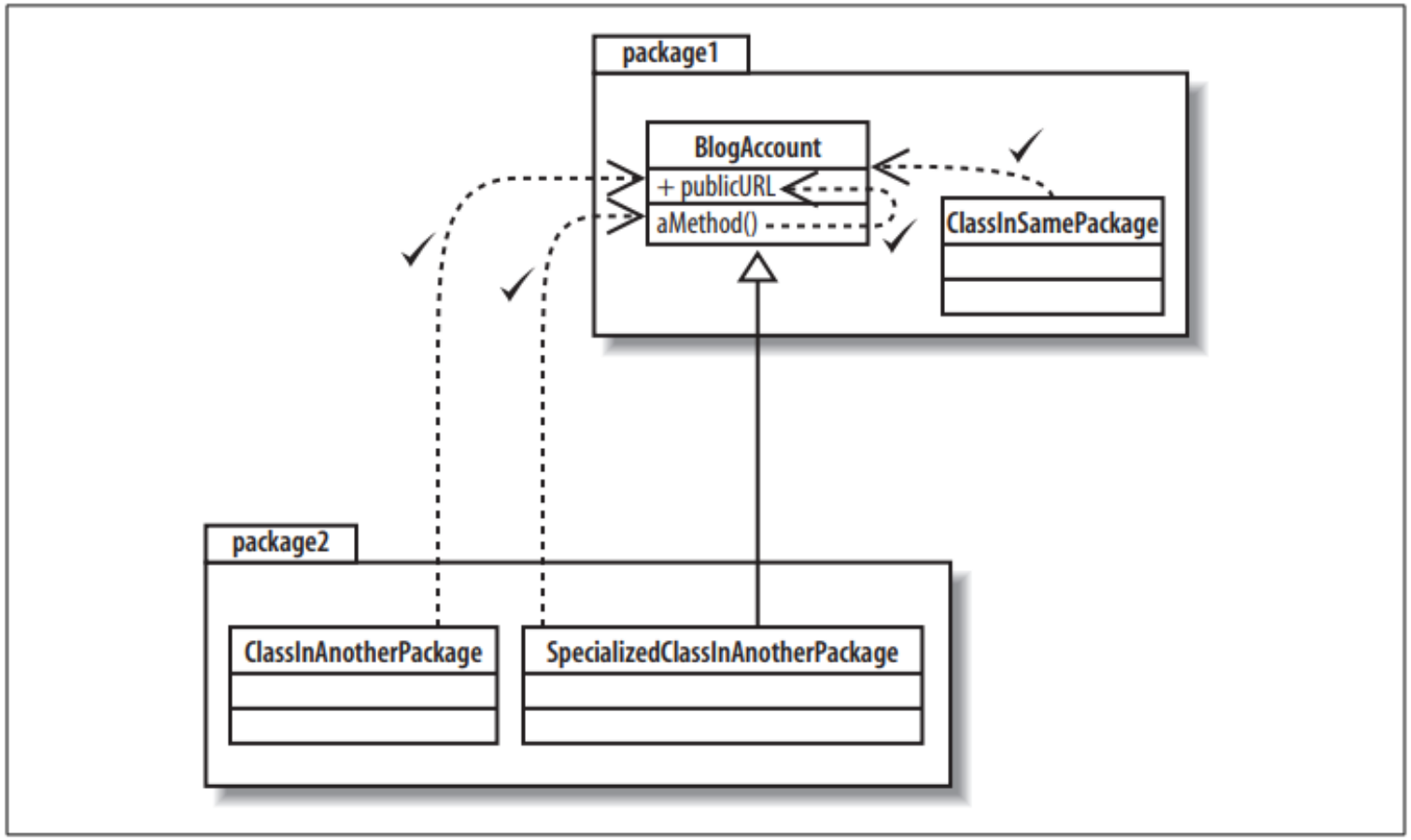
Visibility of Attributes and Operations

- Relates to the level of information hiding to be enforced
- Visibility applies to both **attributes** and **operations**.
- During the **analysis phase**, visibility is **not important**.

Visibility of Attributes and Operations

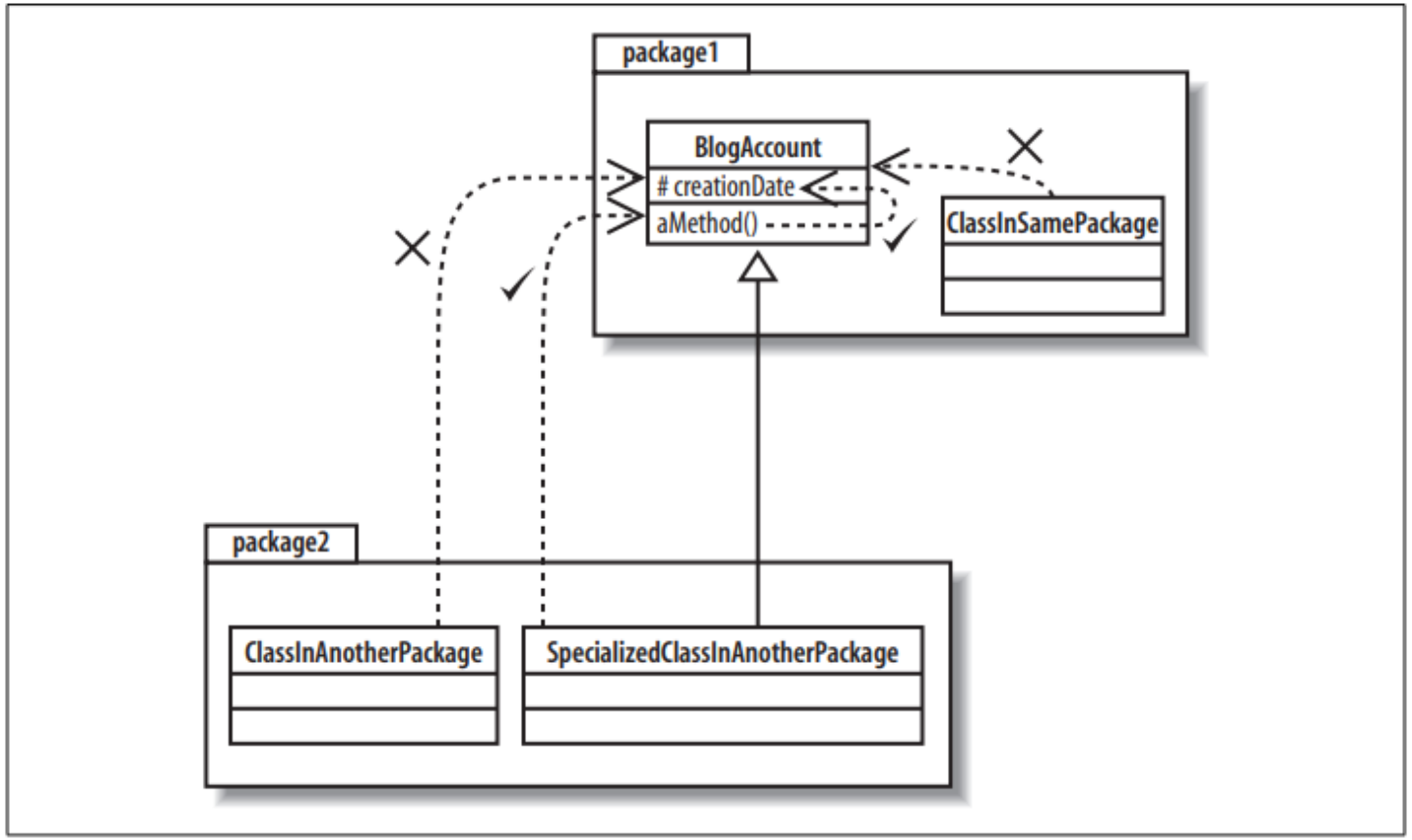
Visibility	Symbol	Accessible To
Public	+	All objects within your system.
Protected	#	Instances of the implementing class and its subclasses.
Private	-	Instances of the implementing class.
Package	~	accessible within the same package

Public Visibility



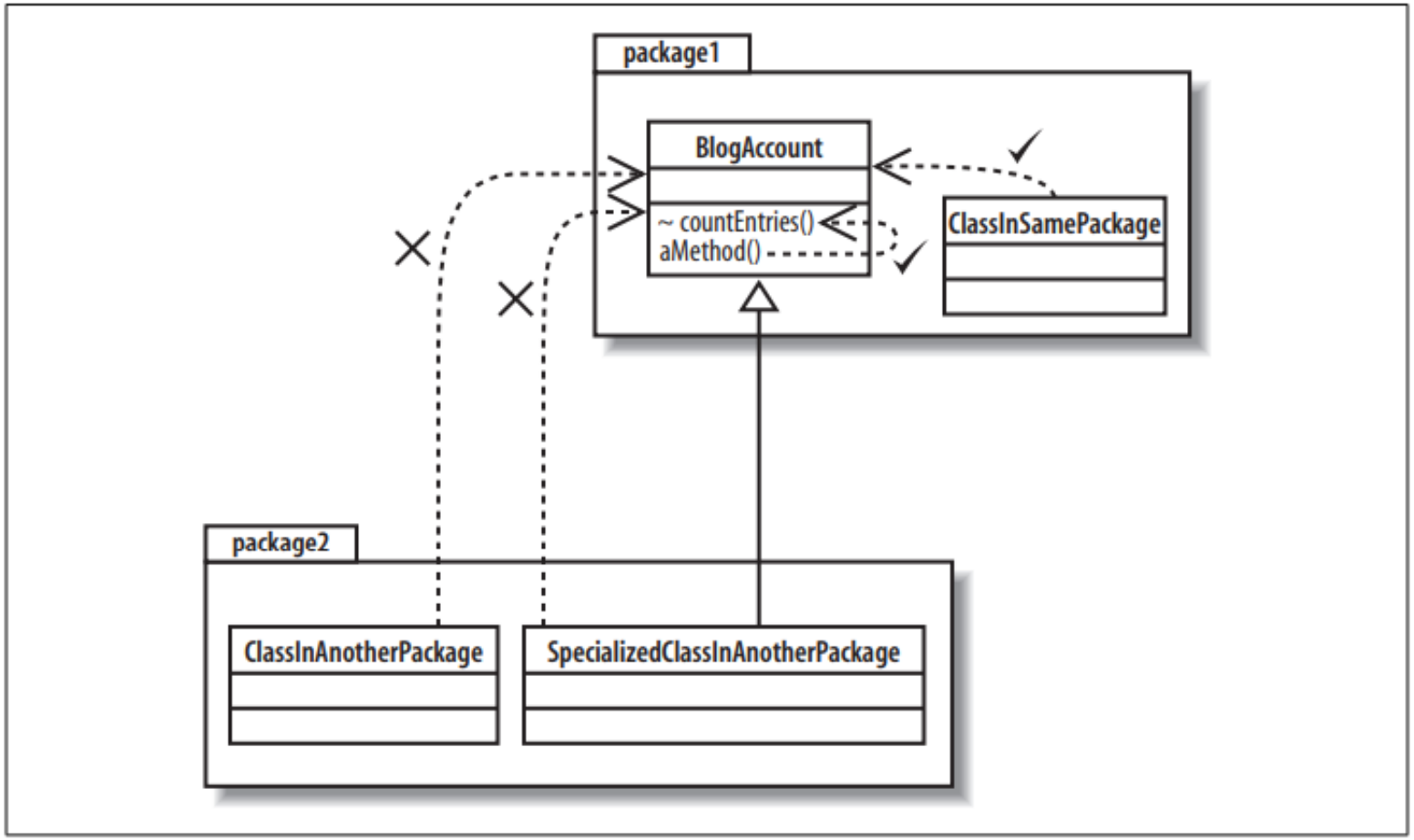
Using public visibility, any class within the model can access the publicURL attribute

Protected Visibility



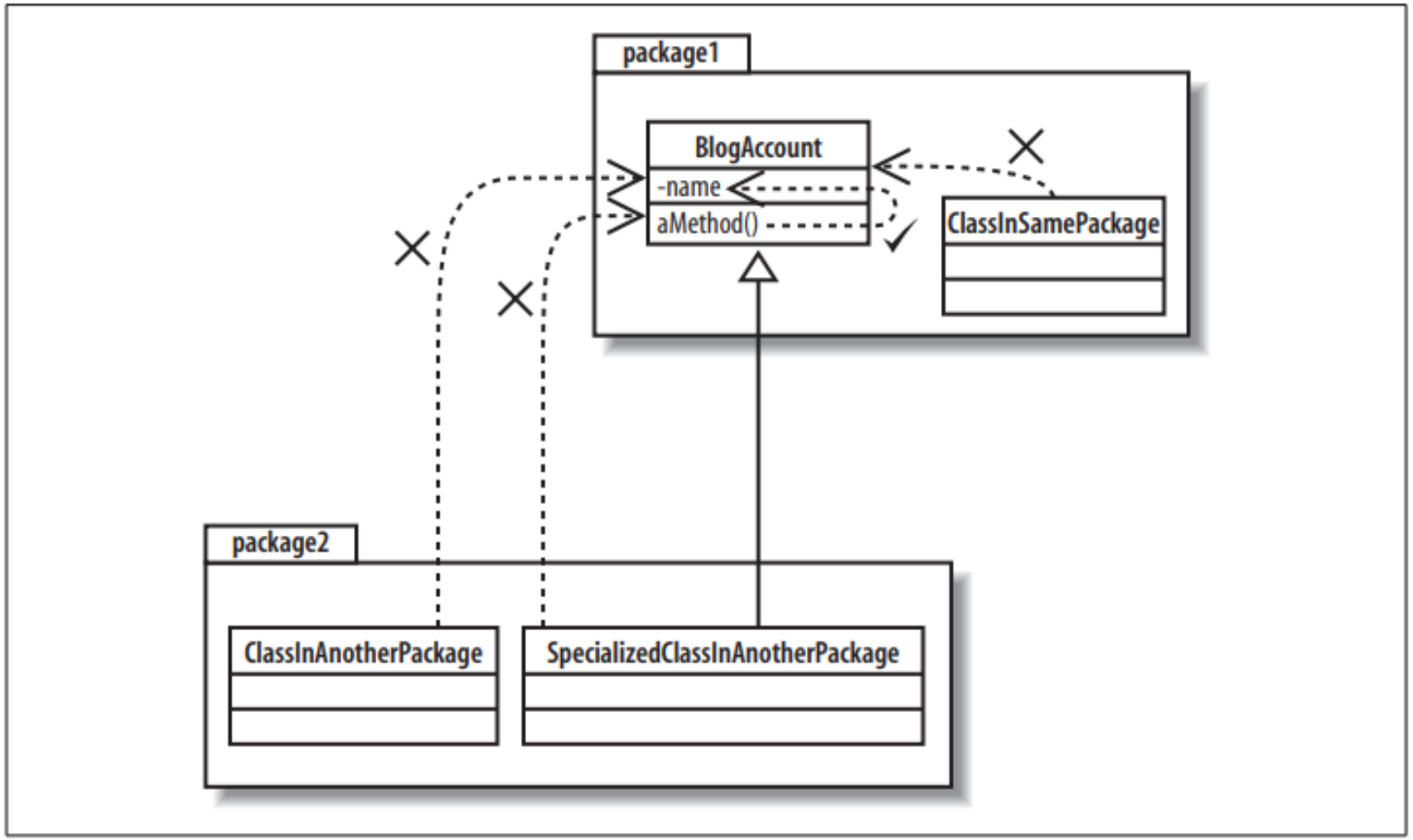
Any methods in the BlogAccount class or classes that inherit from the BlogAccount class can access the protected creationDate attribute

Package Visibility



The countEntries operation can be called by any class in the same package as the BlogAccount class or by methods within the BlogAccount class itself

Package Visibility

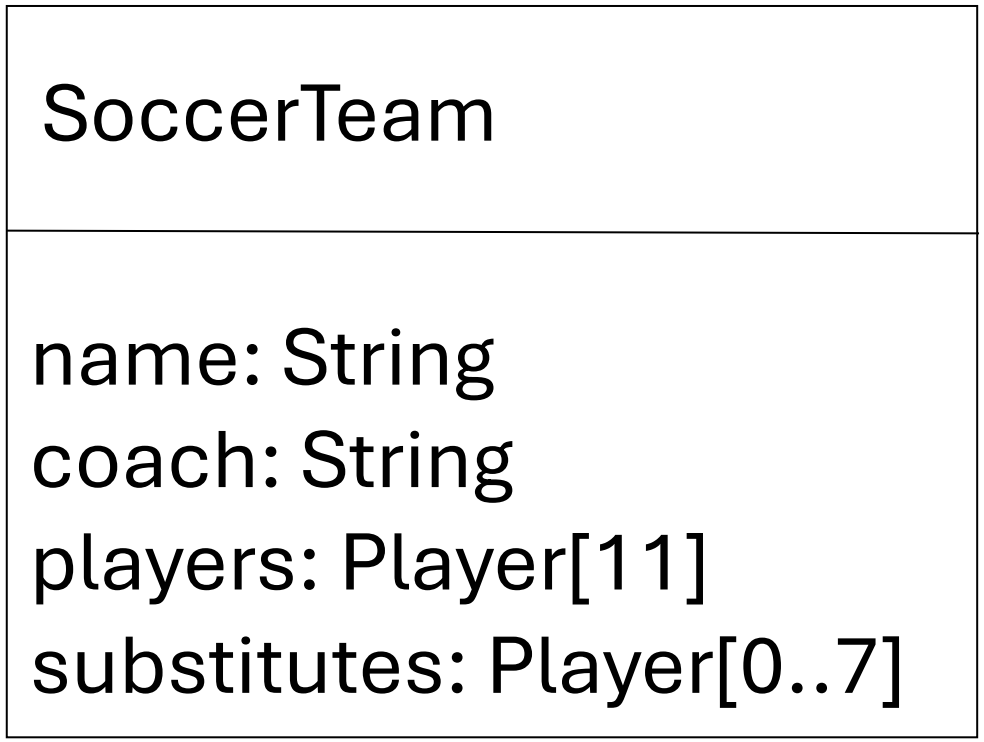


aMethod is part of the BlogAccount class, so it can access the private name attribute; no other class's methods can see the name attribute

Attribute Multiplicity

- Multiplicity indicates that an attribute can have multiple values.
- Multiplicity is similar to arrays
- Examples:
  - int values[7] → The class contains exactly 7 values.
  - int values[2..*] → The class contains at least 2 values.
  - int values[2..7] → The class contains at least 2 and at most 7 values.

Attributes – Exp



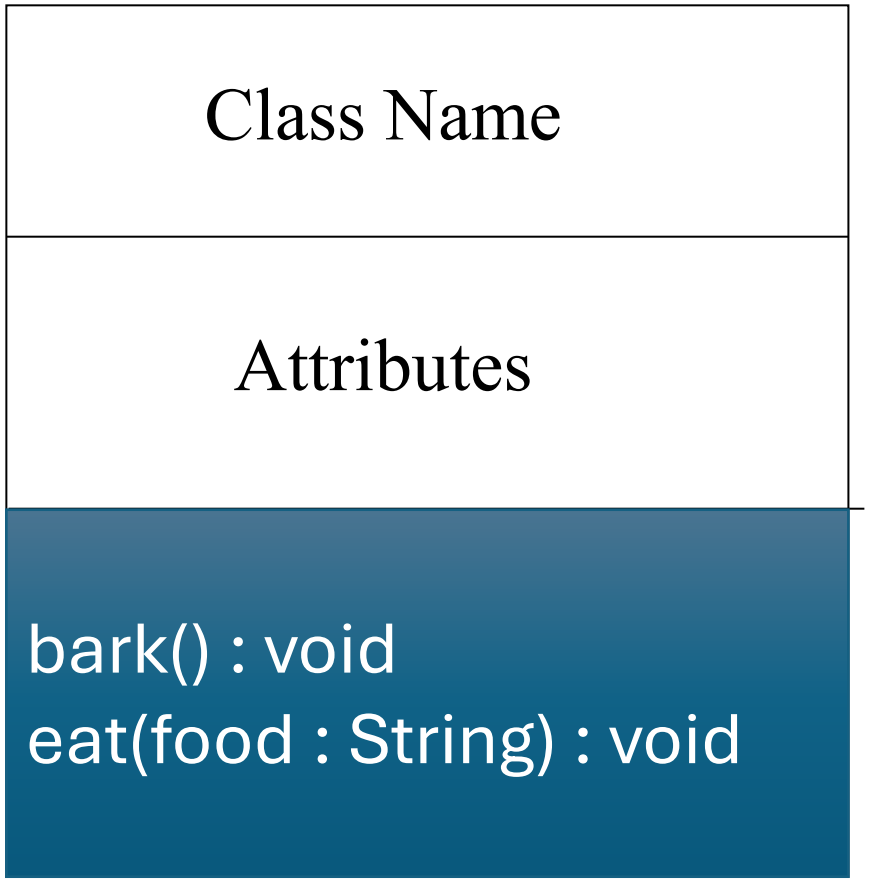
← attribute with multiplicity

Recommendations for Identifying Attributes

- Identify properties that describe objects
- Focus on information that must be stored for the system.
- Some attributes may hide complexity and should be modeled as separate classes.
- Example: address in a Person class.
- Instead of a single attribute, create an Address class with multiple properties:
  - street
  - city
  - zipCode

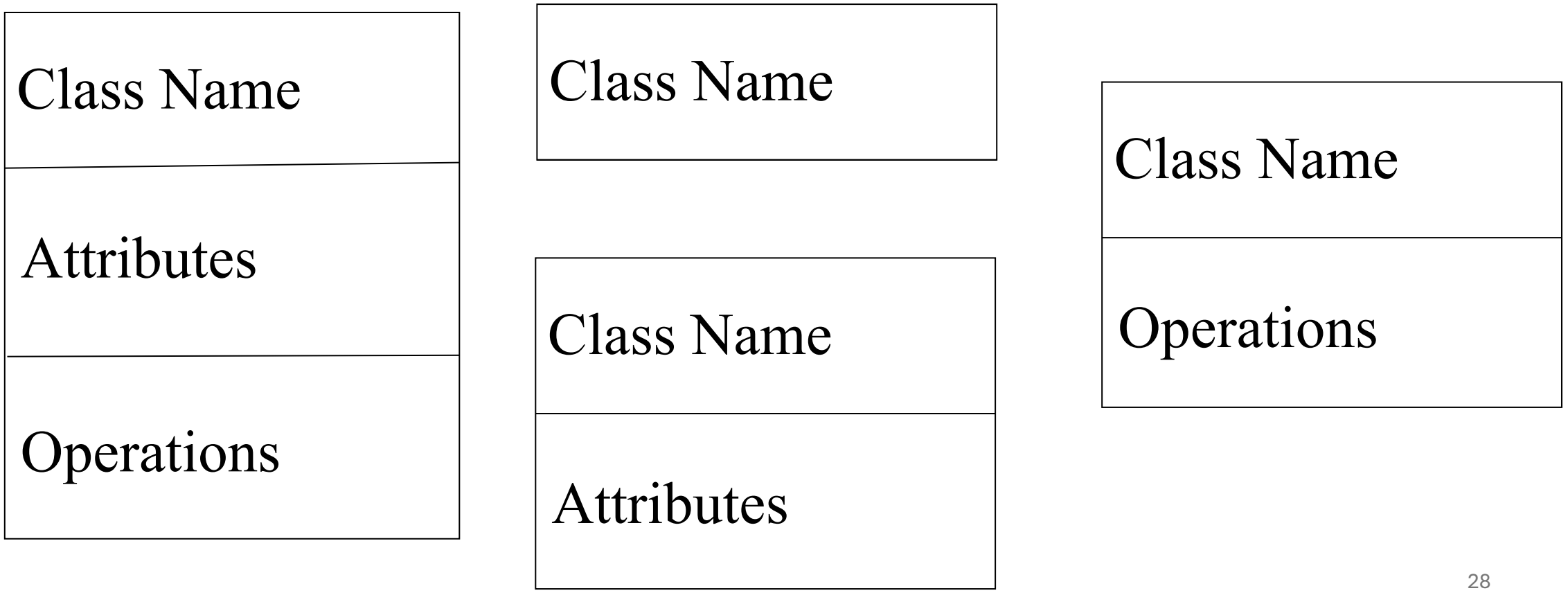
Class Operations

- Operations = functions or behaviors of a class.
- Placed in the bottom section of the UML class rectangle.
- **Can have arguments** (inputs) and a **return type** (output).
- Syntax: visibility name(parameters) : returnType



UML Representation of Class

A class **can be shown empty**, no attributes or operations.

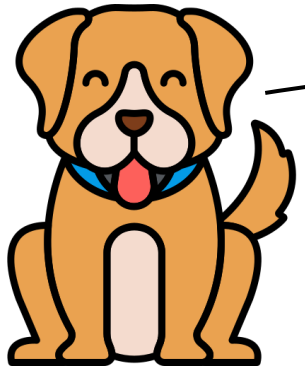


Relationships among Classes

Represents a connection between multiple classes or a class and itself

- **Association**: A basic link between two classes without special semantics.
- **Aggregation** : A “whole–part” relationship where parts can exist independently of the whole.
- **Composition** → A strong “whole–part” relationship where parts cannot exist without the whole.
- **Inheritance (Generalization)** → A relationship where a subclass specializes and inherits from a superclass.

Associations



Abstraction

Dog
age
run()

Person
age
run()

Associations

- In the real world, objects are **linked physically or functionally**.
- In UML, these links are represented at the **class level** as **associations**.
- An **association** = a **static structural relationship** between two or more classes.

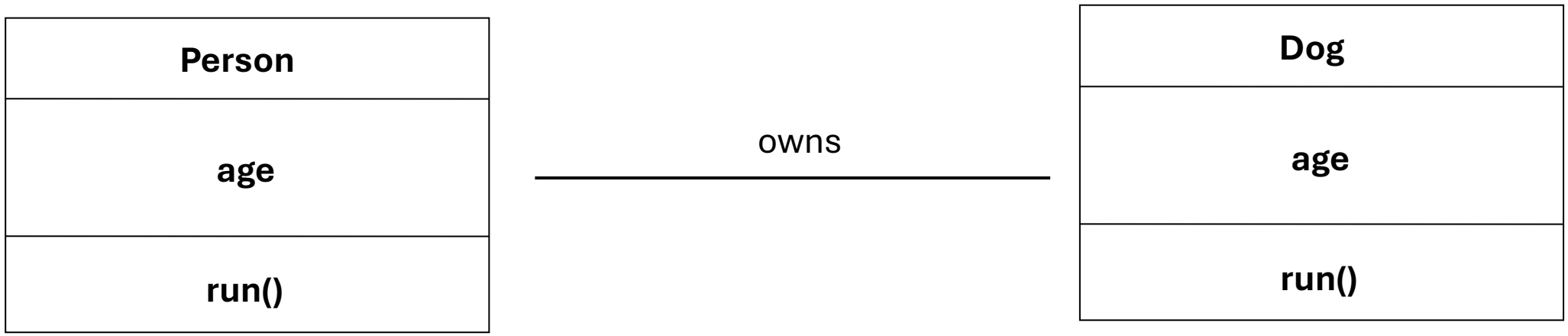
An association is an abstraction of links that may exist between instances of those classes.

Association (Notation)

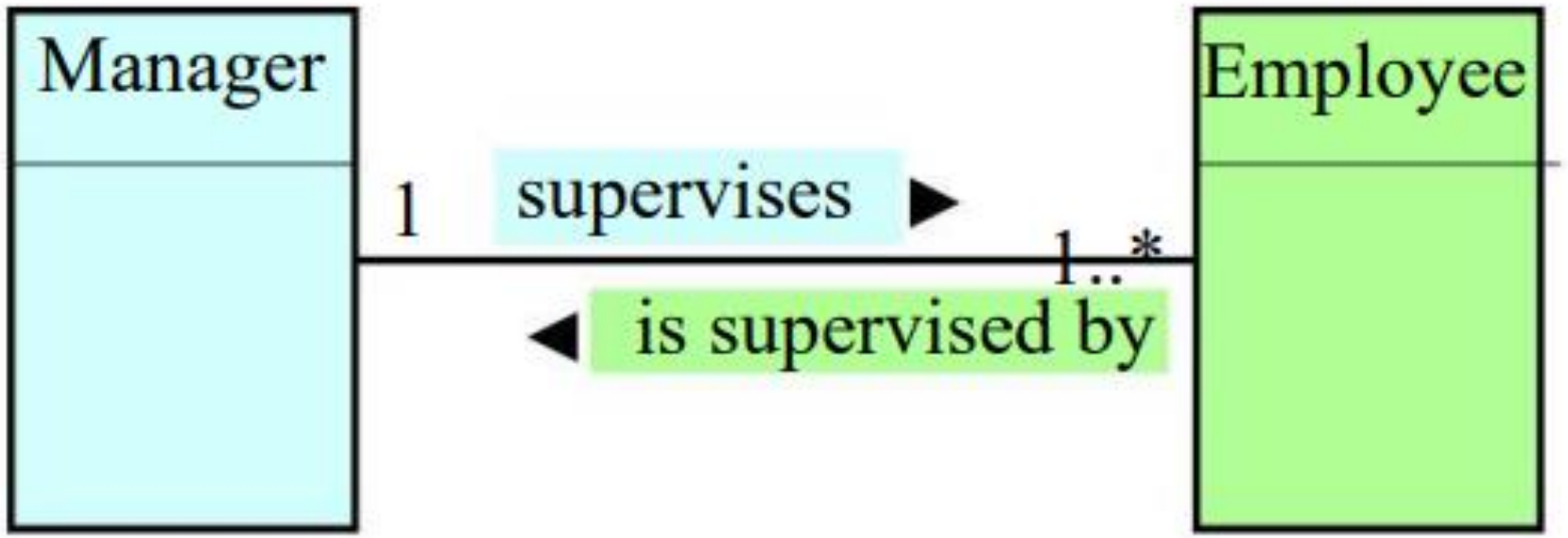
- Represented by a solid line between classes.
- May have a **name/label** describing the relationship.
- **Roles** can be indicated at each end (e.g., owner, pet).
- **Multiplicity** shows how many objects can be linked

Association Name

- Usually written in **verbal form** (active or passive).
- Describes the **meaning of the link**.
- The name is **optional**.
- Must appear **on the association line**, not attached to the ends.



Association Name



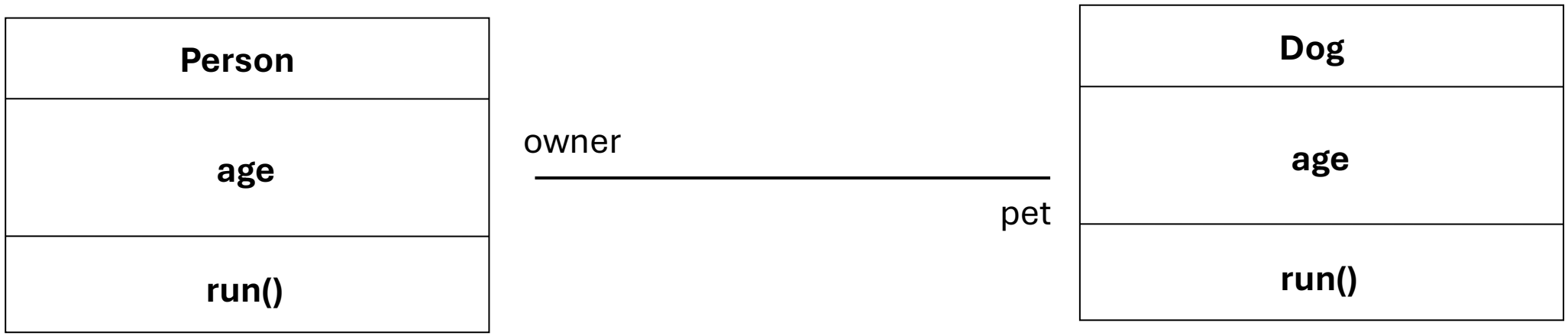
Roles in Associations

- **Role names** describe the purpose of each class in an association. It **Describe how a class perceives another class** through the association.

- Placed **near the ends of the association line**. Usually **singular nouns**, lowercase.

- Clarify how **instances participate** in the relationship.

- **Optional**, not mandatory if the association is clear.



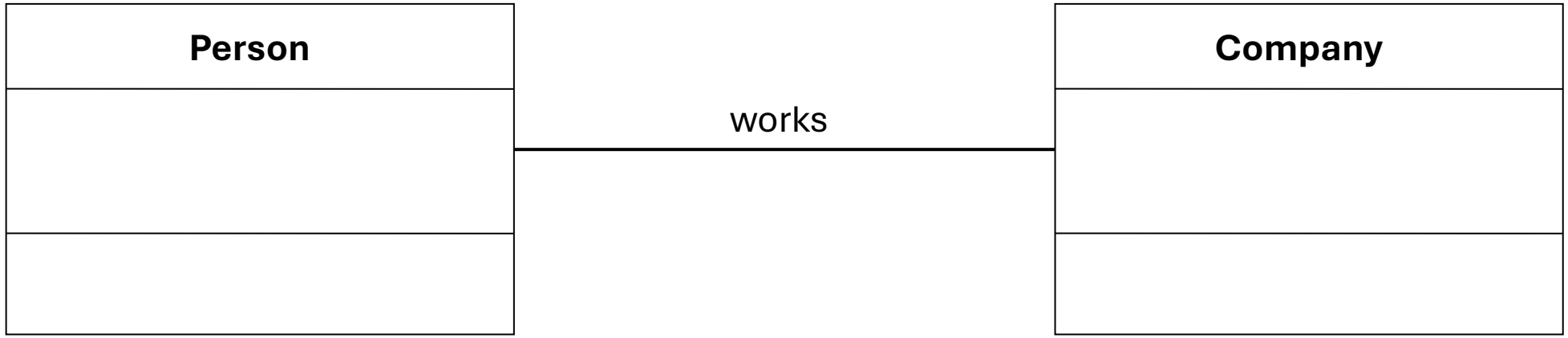
Multiplicity in Associations

- **Multiplicity** is attached to an end of an association.
- Indicates **how many instances of a class** can be linked to **one instance of the other class**.

Multiplicity in Associations

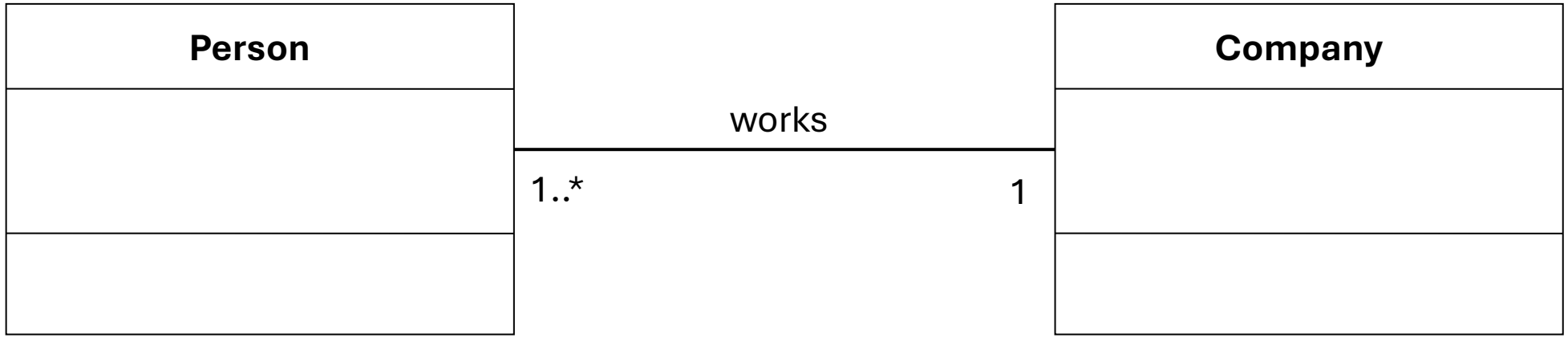
Notation	Meaning	Example
`1`	Exactly one	Each Student has exactly one ID card
`0..1`	Zero or one	A Car may or may not have a GPS device
`*`	Many (0 or more)	A Person can have many Dogs
`0..*`	Zero or many	//
`N`	Exactly N	A soccer team must have 11 Players
`M..N`	Between M and N	A classroom has between 10 and 30 Students
`1..*`	One or many	Each Order must have at least one Product
`N..*`	N or more	A conference must have at least 2 speakers

Multiplicity in Associations



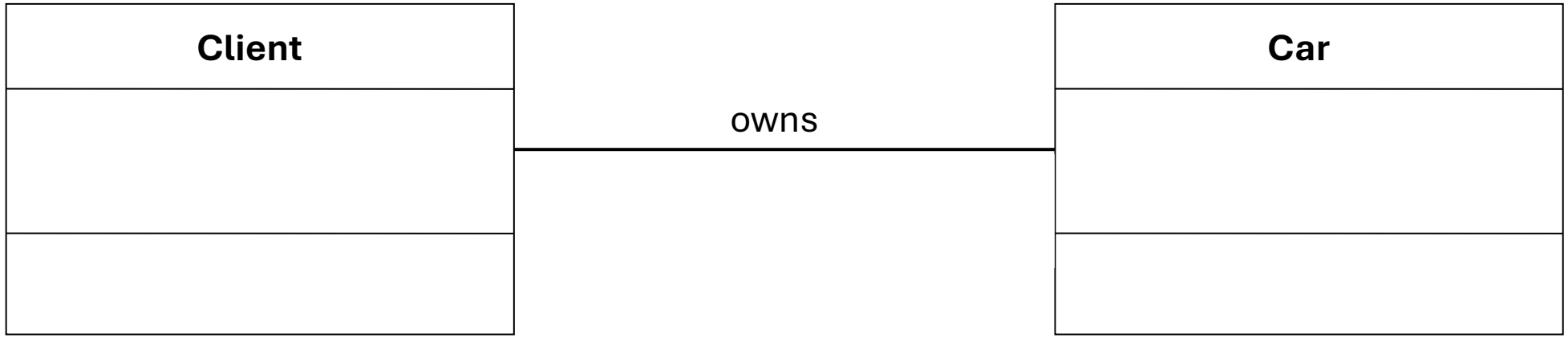
- Each Person works in **exactly one company**
- Each Company employs **one or more people**

Multiplicity in Associations



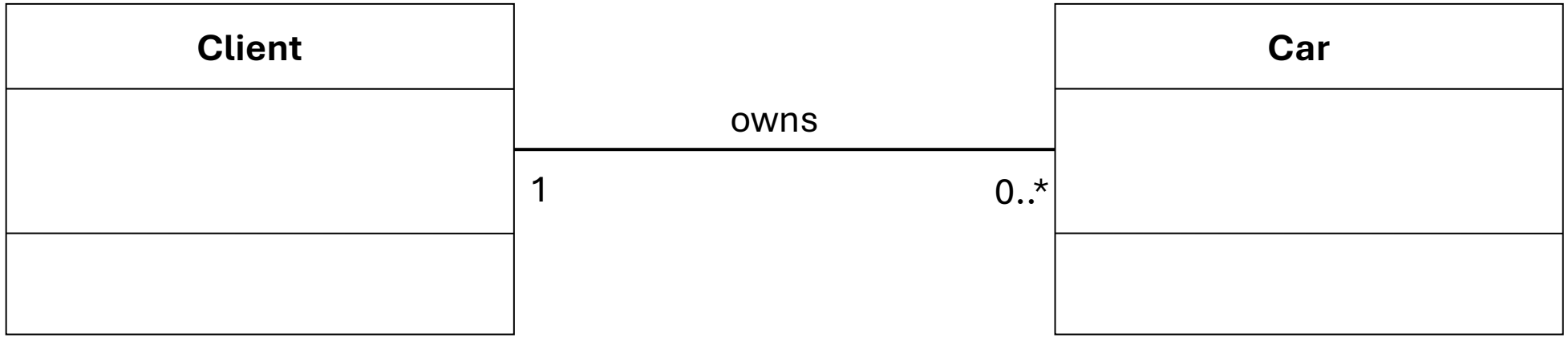
- Each Person works in **exactly one company**
- Each Company employs **one or more people**

Multiplicity in Associations



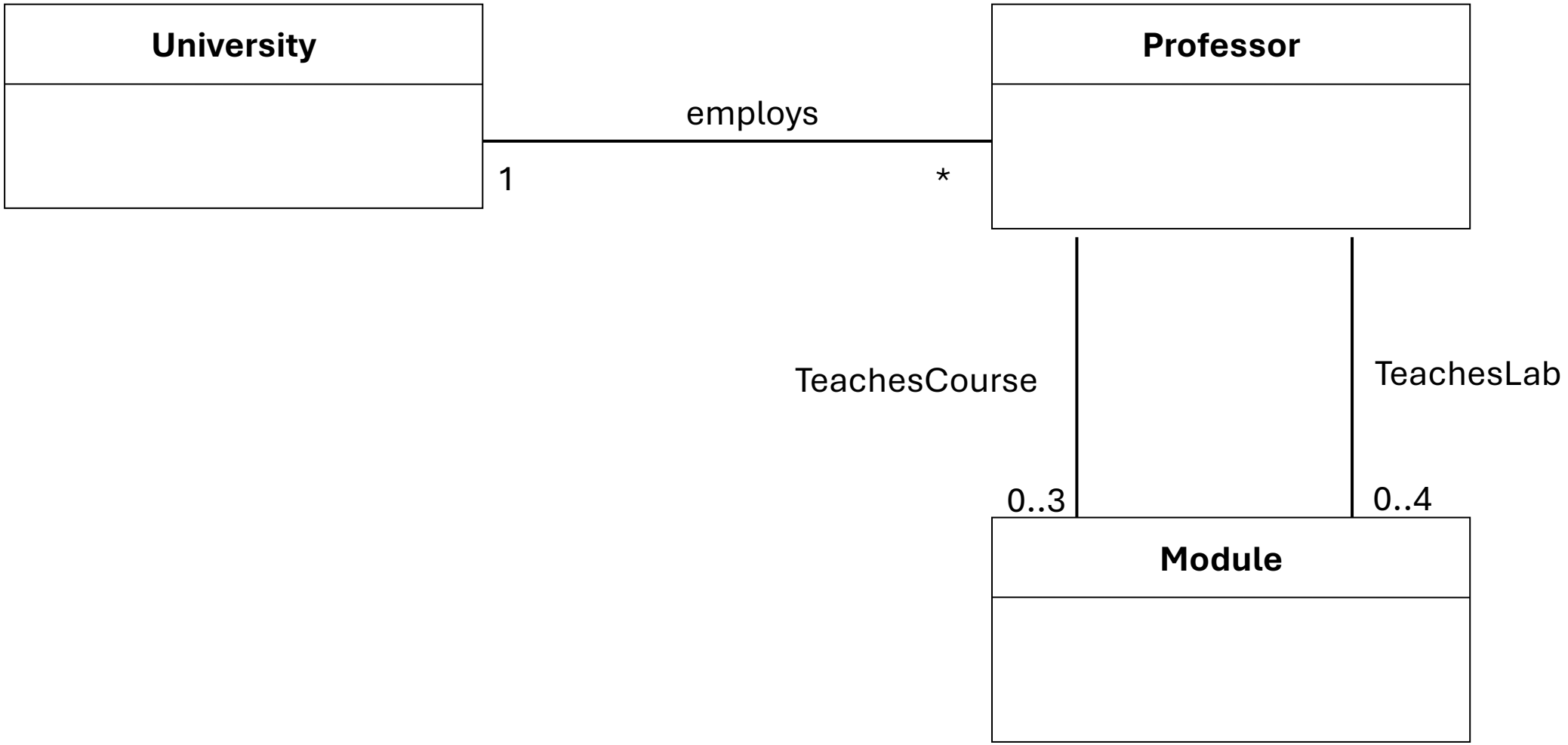
- Each Car is owned by **exactly one client**
- A Client may own **zero or many cars**

Multiplicity in Associations



- Each Car is owned by **exactly one client**
- A Client may own **zero or many cars**

Multiplicity in Associations



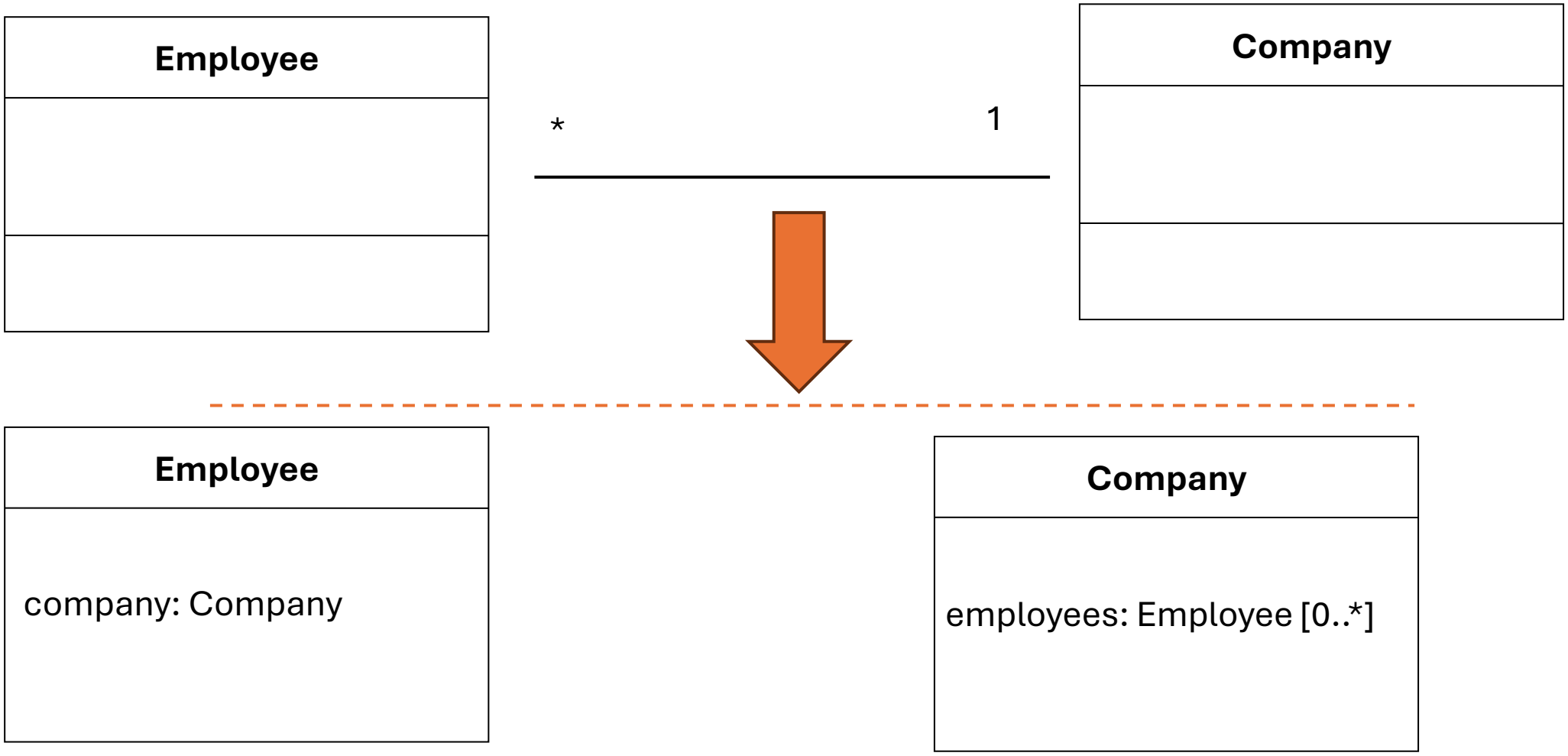
Exercise

1. A **Teacher** teaches **one or more Courses**.
2. A **Course** can have **many Students** enrolled.
3. A **Student** can enroll in **many Courses**.
4. Each **Course** takes place in **one Classroom**.

Associations and Attributes

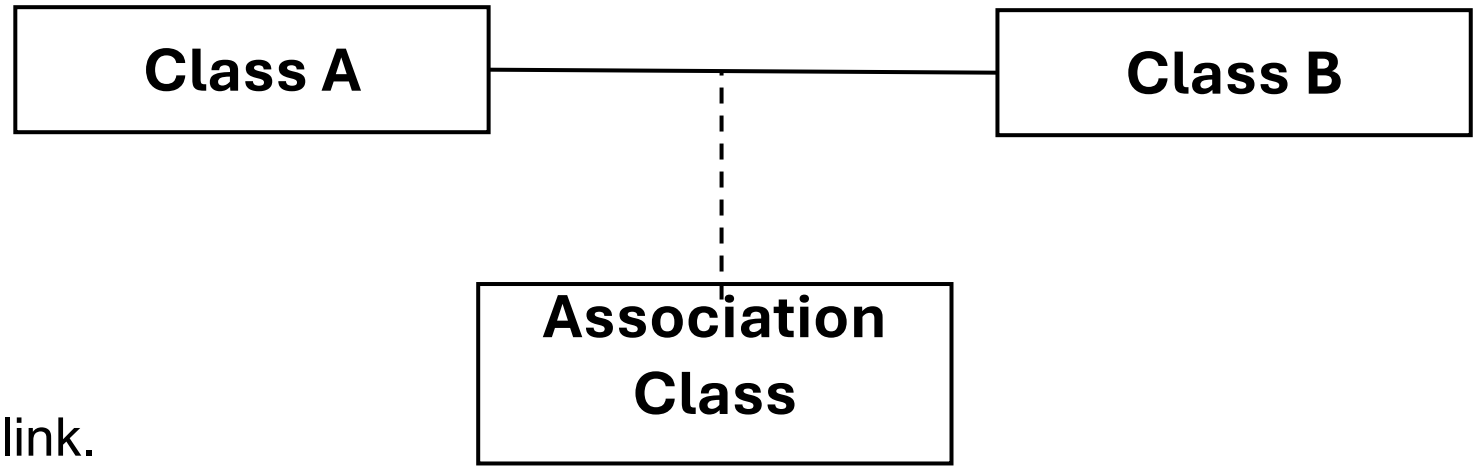
- Attributes can be **another way to represent an association**.
- When generating code, **associations are implemented as attributes**.

Associations and Attributes

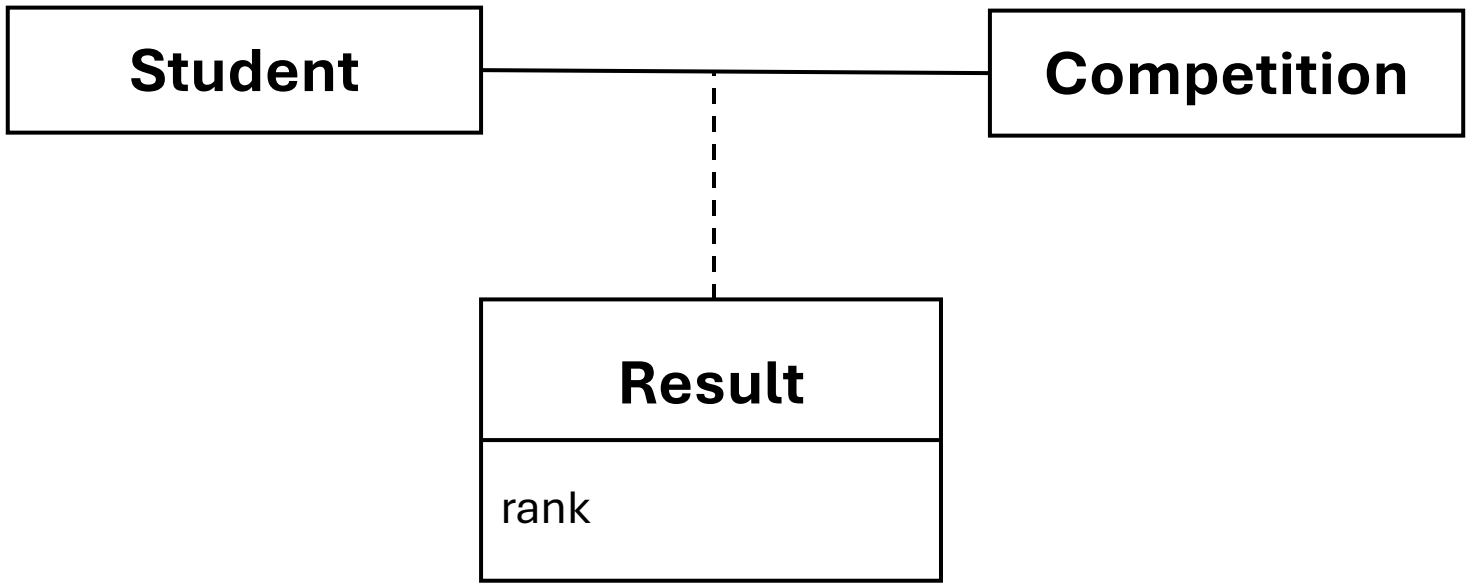


Association Class

- An **association** can sometimes need its own attributes.
- Since **only classes** can have attributes, the association becomes a **class-association**.
- A class-association combines:
  - The **link between classes**.
  - The **attributes** describing that link.

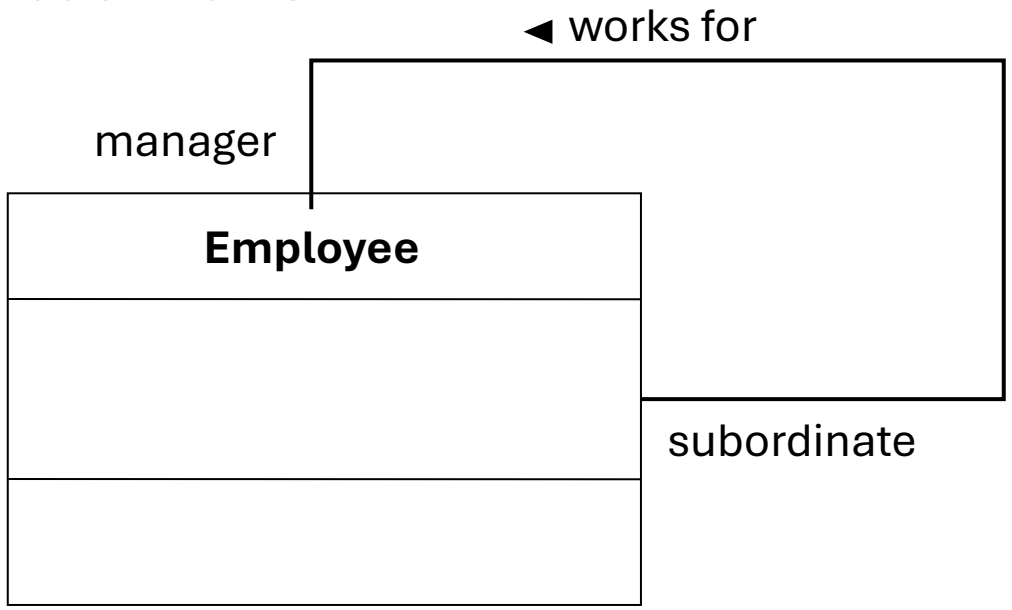


Association Class – Exp 1



Reflexive Associations

- Connects a **class to itself**.
- Used to represent **hierarchical or self-referencing relationships**.
- Reading direction is indicated with:
  - **Roles** (e.g., manager vs subordinate)
  - Or a **small solid triangle beside the association name**



Binary Associations

- Most common type of association.
- Connects **exactly two classes**.

Aggregation

- **Definition:** Aggregation is a **special kind of association**.
- Represents a relationship of **whole–part (inclusion)**.
- Whole & parts objects **can exist independently**

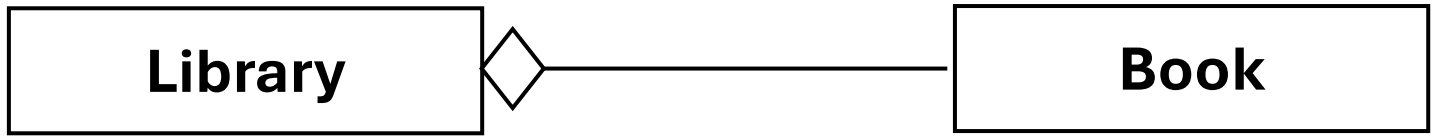
Example: a bank (whole) has customers (as parts)

- Deleting a bank does not cascade deleting customers
- Customers can move to another bank

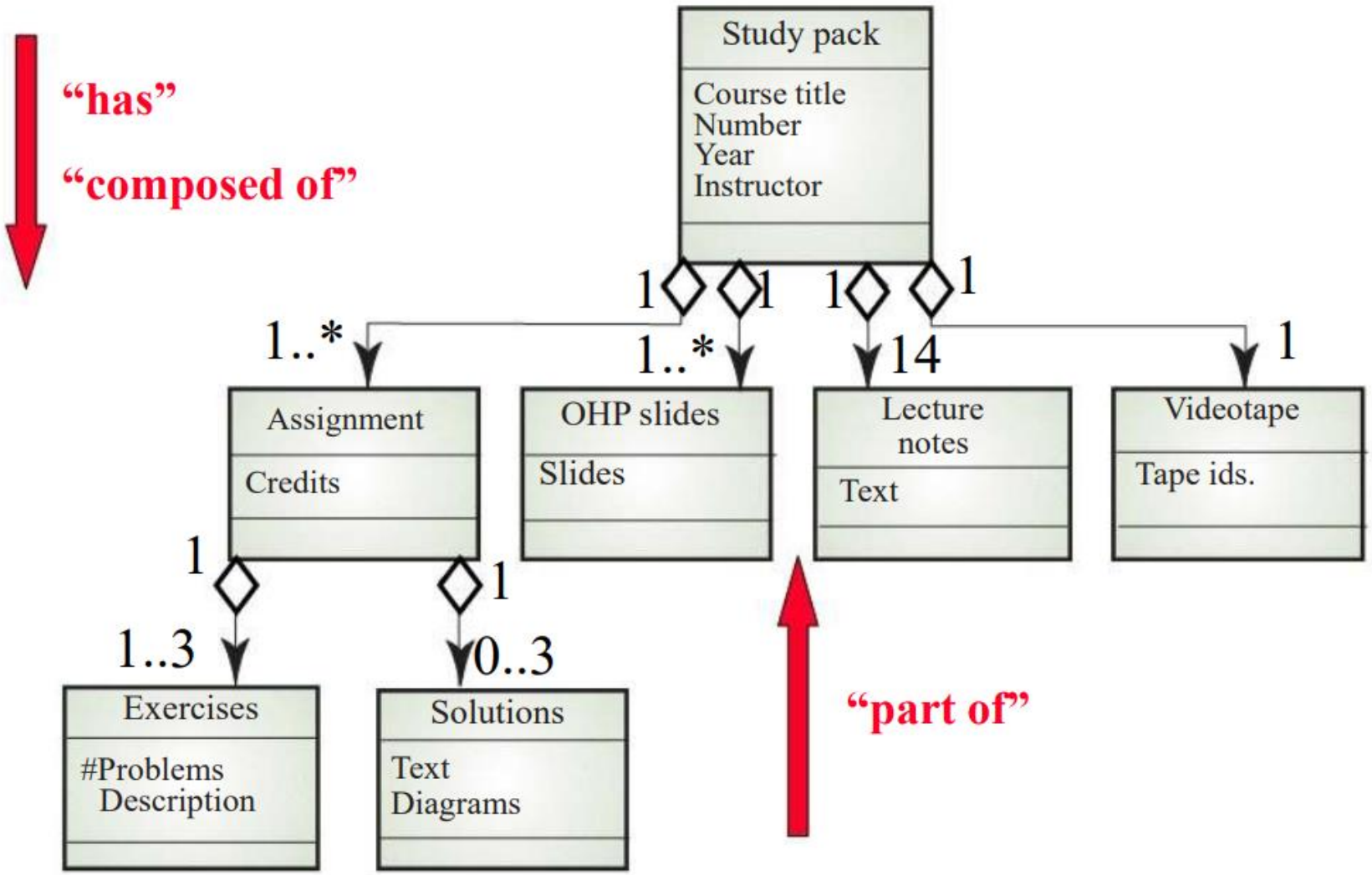
Aggregation

- Aggregation model shows how classes (which are collections) are composed of other classes.

- A line joins a whole to a part (component) with an open diamond on the line near the whole.



Aggregation



Composition

- A composite is a **strong** type of aggregation.
- Each component in a composite can belong to **just one whole.**
- The symbol for a composite is the same as the symbol for an aggregation except the diamond is filled.

Composition - Example 1

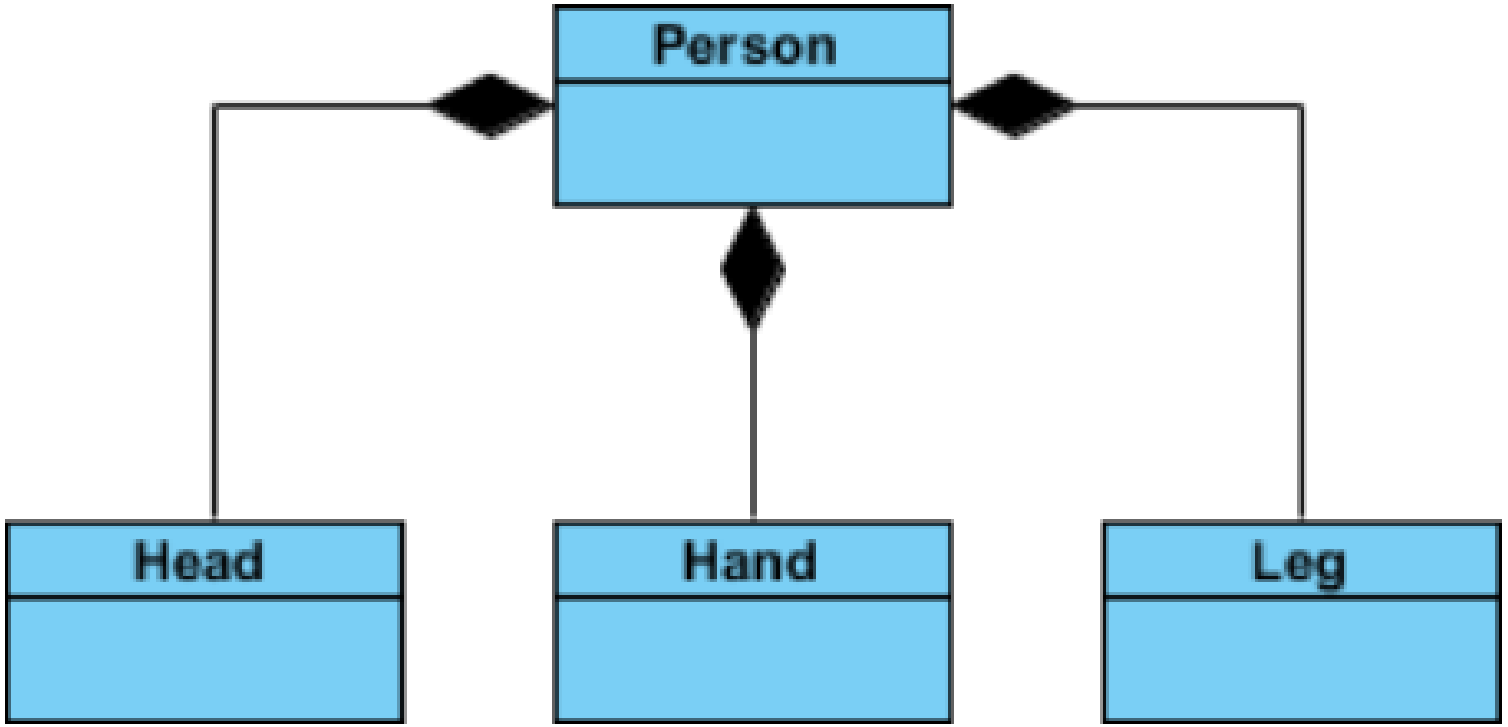
- Human's outside:

  Every person has: head, body, arms and legs.

- **A composite association**. In this association each component belongs to exactly **one** whole.

- Whole & parts objects can NOT exist independently

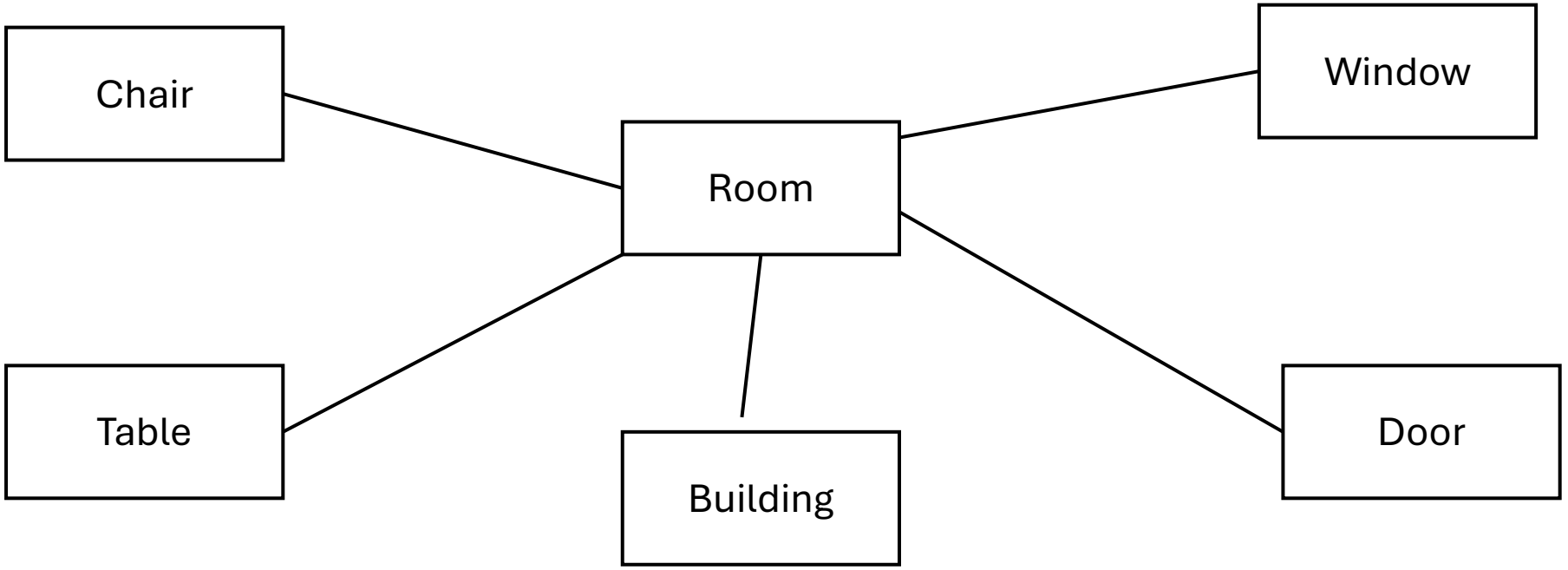
Composition - Example 1



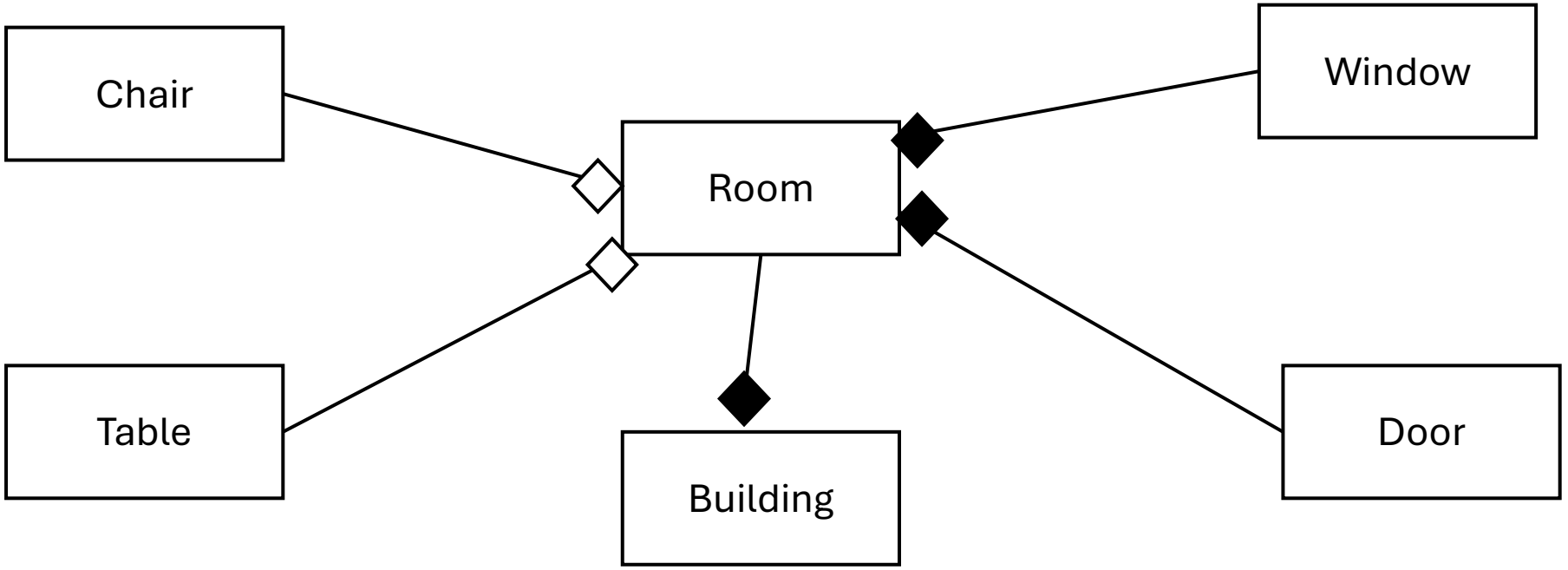
Composition - Example 2

- A bank (whole) has many branches (parts)
- Branches can not exist independently of the whole (parts objects can NOT exist independently)
- Deleting a bank (whole) cascades deleting branches (parts)
- But, if a branch (part) is deleted, the bank (whole) may remain

Exercise

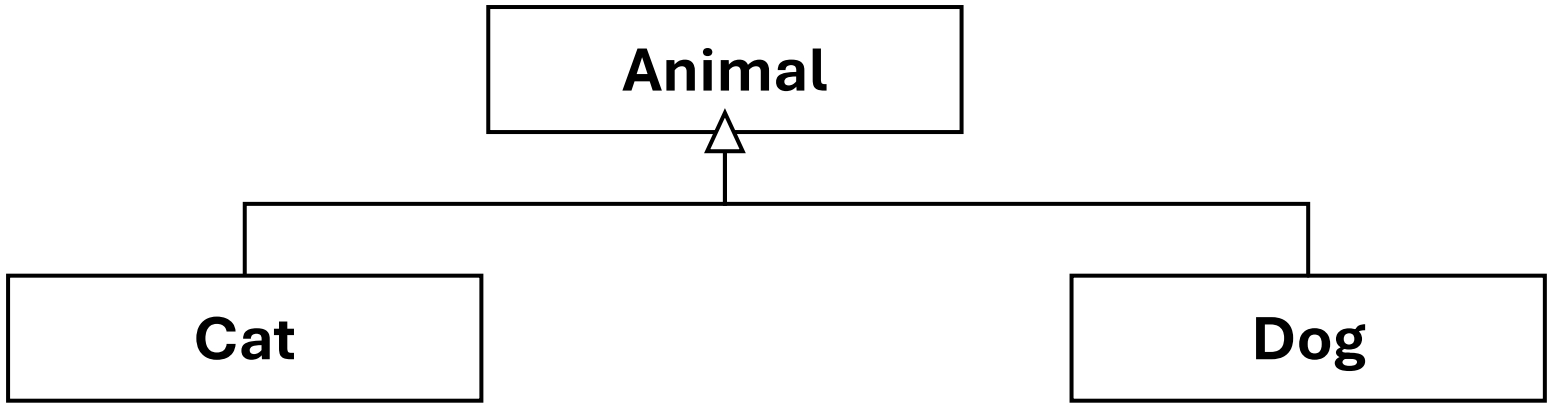


Exercise

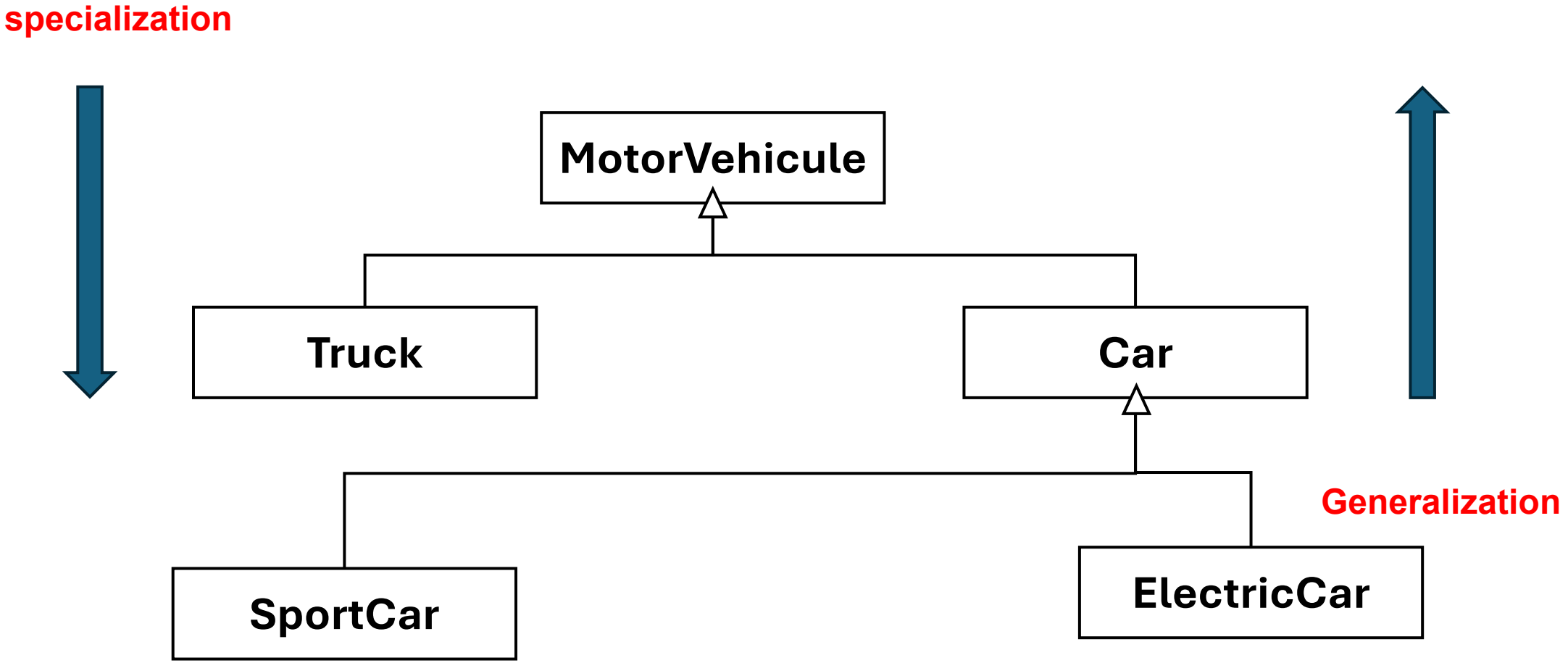


Inheritance

- Inheritance expresses a **generalization–specialization** relationship.
- A **superclass (parent)** defines common attributes and operations.
- **Subclasses (children)** inherit them and may add or redefine features.
- Notation: a **solid line with a hollow triangle** pointing to the superclass.



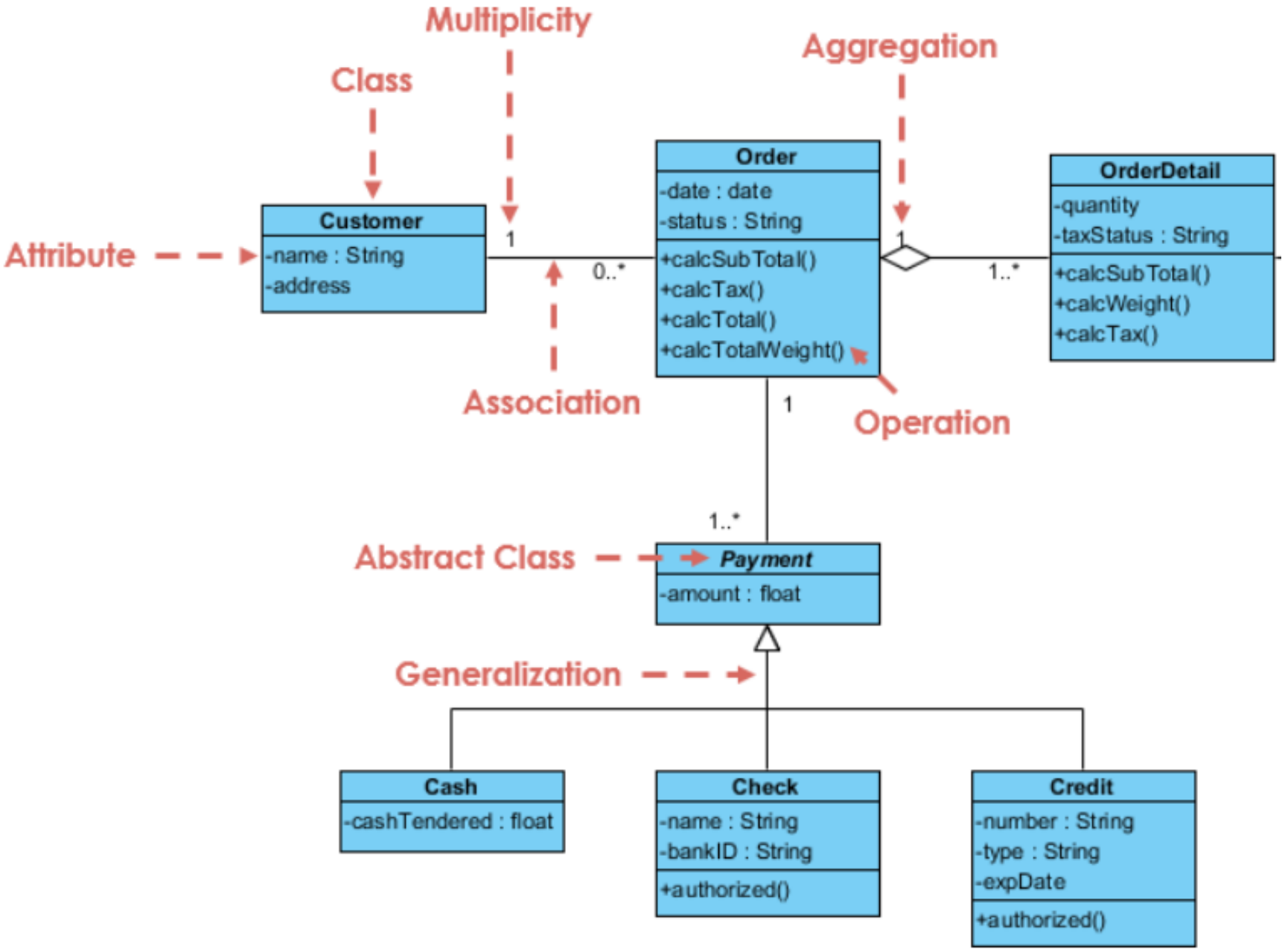
Inheritance



Abstract Classes

- Sometimes a class can only exist **through its subclasses**
- A class may **delegate** the implementation of certain operations to its subclasses
- An operation **without implementation** in the current class is called an **abstract operation**
- A class that contains **one or more abstract operations** is called an **abstract class**
- Class name written in *italics*

Example

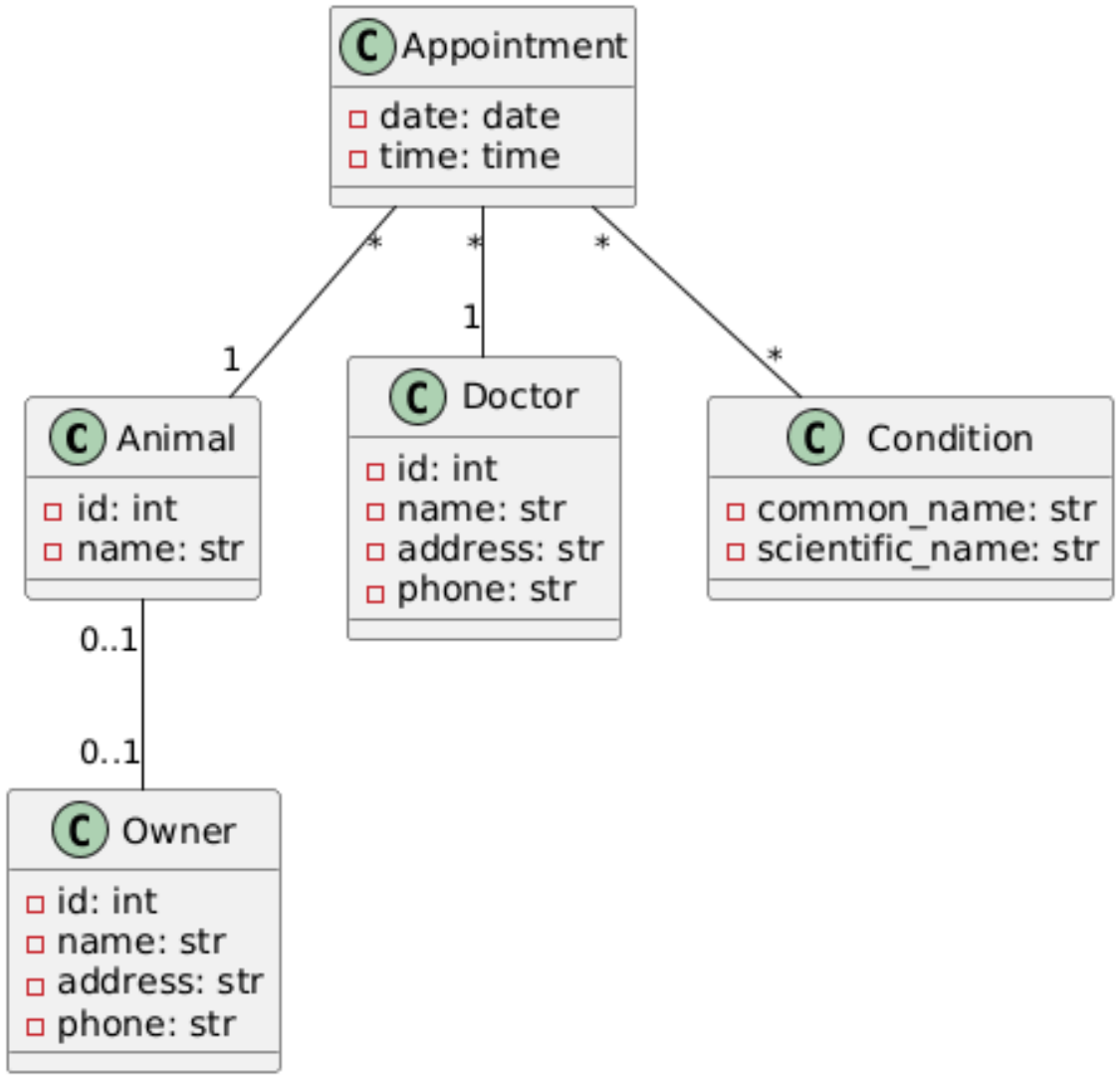

Multiplicity
Class
Aggregation
Attribute
Customer
-name : String
-address
1
0..*
Association
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
Operation
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()

Exercise

The owner of a veterinary clinic wants to create a database to store information about all veterinary services performed.

1. For each animal admitted, its name and owner must be recorded. Each animal must have a unique numeric ID.
2. For each owner, the name, address, and phone number must be recorded. Each owner has a unique numeric ID.
3. An animal can have no owner, because the clinic often rescues stray animals.
4. Each appointment always has a responsible doctor. All appointments start at a specific date and time and involve an animal.
5. For each doctor, record name, address, phone number, and a unique numeric ID.
6. During an appointment, multiple medical conditions can be detected. Each condition has a common name and a scientific name.

Exercise


Appointment
date: date
time: time
Animal
id: int
name: str
Doctor
id: int
name: str
address: str
phone: str
Condition
common_name: str
scientific_name: str
Owner
id: int
name: str
address: str
phone: str
*
*
*
1
1
*
0..1
0..1