Software Design & Architecture

UML & Database

Pengyu Nie

Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

Agenda

- UML Introduction
- Class diagram
- Data model

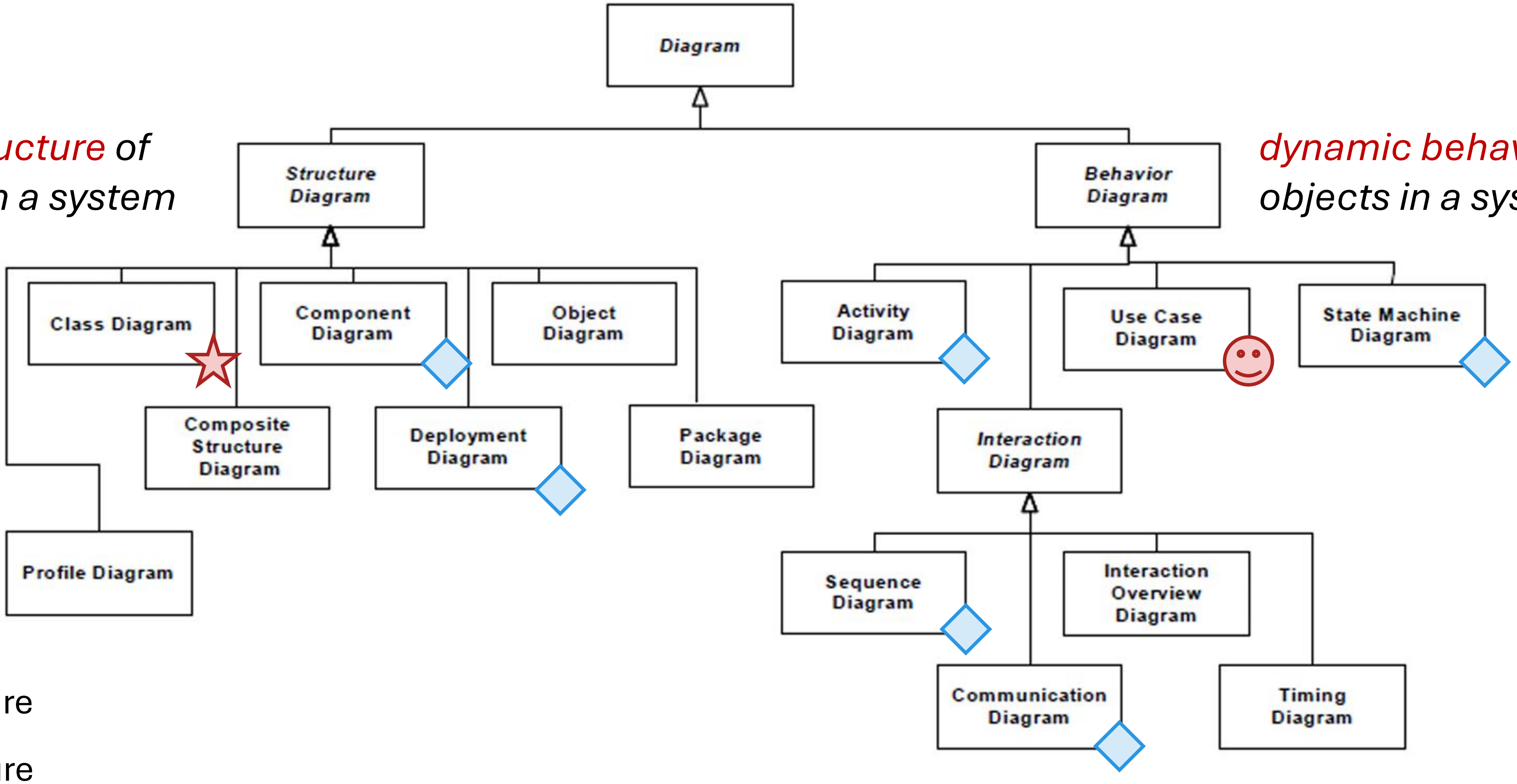
Unified Modeling Language (UML)

- UML is a set of notations, not a methodology or process
  - Official standard backed by OMG, version 2.5.1
  - Rational Software (now owned by IBM) is the big mover behind UML, but they don't "own" it
  - Lots of history and politics behind it
- Many expensive tools, seminars, books, hype, etc. ... but
  - "UML is just a bunch of notations"
  - UML doesn't solve your problems for you, it gives you a way of writing them down

Taxonomy of UML Diagrams

static structure of objects in a system

dynamic behavior of objects in a system

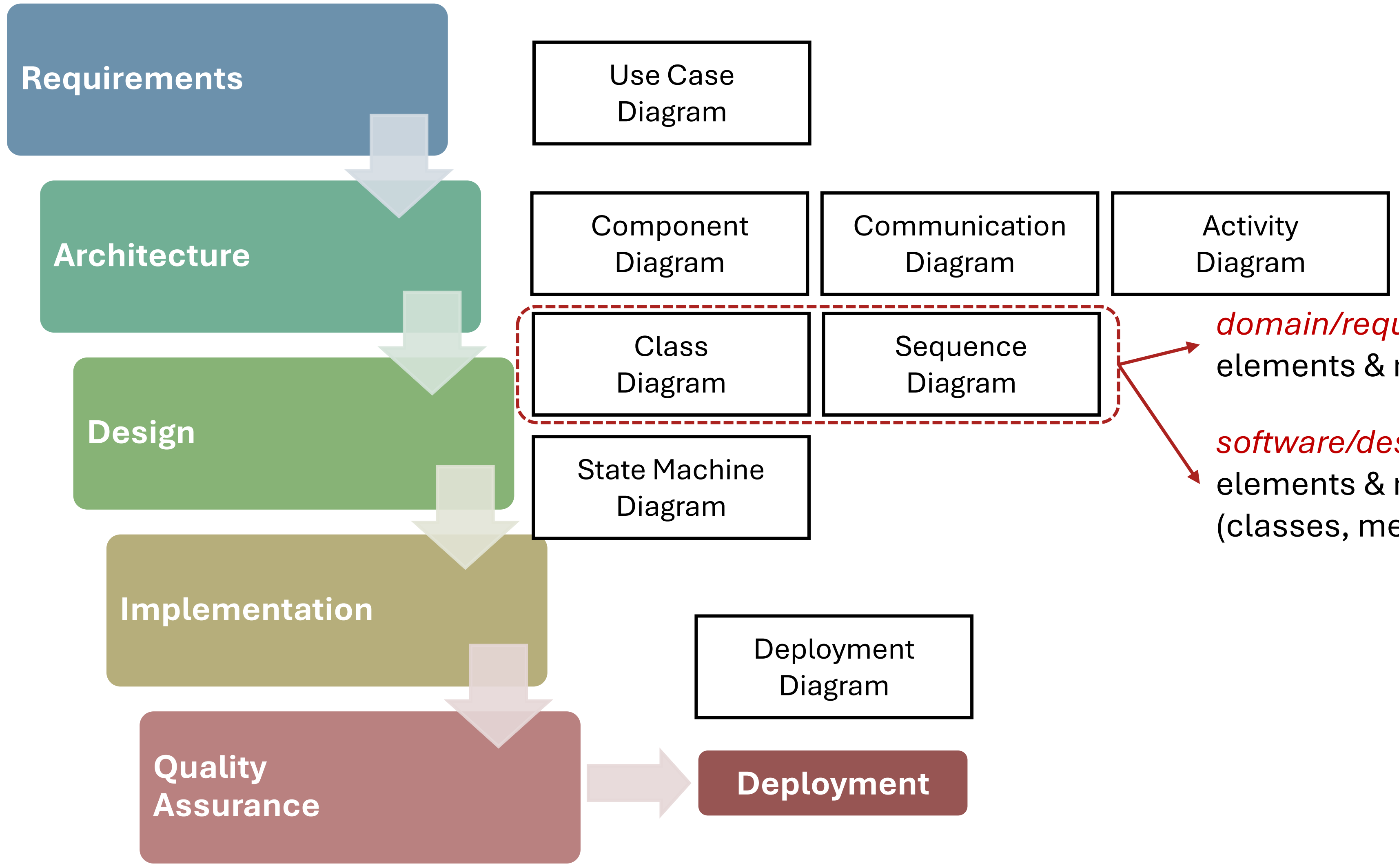


☺ last lecture

☆ this lecture

◇ next lecture

UML Diagrams by Phases / Perspectives

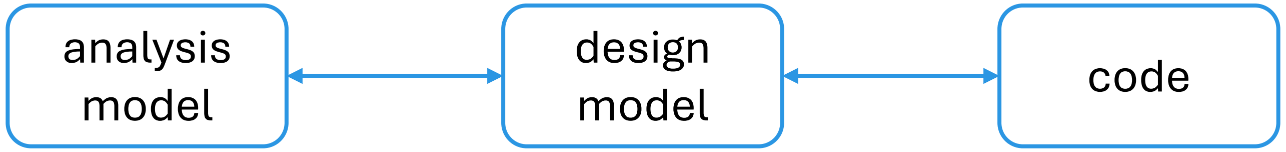


domain/requirements perspective:
elements & relationships maps to abstract concepts

software/design perspective:
elements & relationships maps to code elements (classes, method calls)

Usages of UML – Blueprint

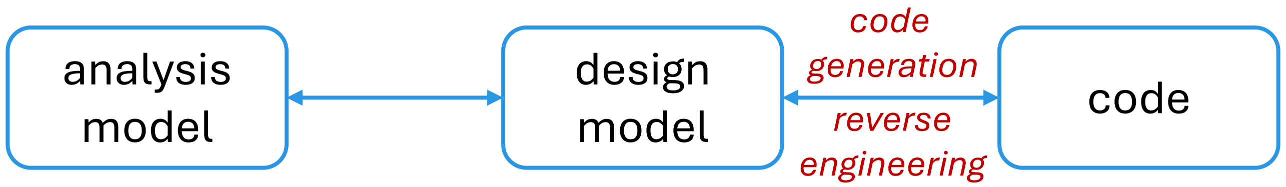
- Called as “religions” by Martin Fowler
- UML as blueprint
  - Goal is rigorous, complete specification of analysis and/or design of a software system



  - UML diagrams express partial semantics of system
    - e.g., structure, communication paths, control/data/other dependencies
  - UML diagrams do not completely specify low-level semantics
    - e.g., full details of what happens inside a method body

Usages of UML – Blueprint (cont.)

- UML as blueprint
  - Tool support is the key: “round trip engineering”
    - Code generation: UML models -> interfaces / class skeletons
    - Reverse engineering:
      source code -> class diagrams; execution trace -> sequence diagrams



  - Choose a desired level of detail (e.g., class + instantiate, call, inherit relationships);
    the UML models are complete with respect to that level of detail

Usages of UML – Programming Language

- UML as programming language
  - The UML diagrams are the system (as maintenance artifacts, not code)
  - Tool support is even more important!
    - generate code from detailed UML diagrams (e.g., state machine diagram)
  - Unfortunately, we are not quite there yet…
    The grand goal of the [MDA (model-driven architecture) movement]
  - Very hard to do the dynamic behavior aspects of the systems

UML Tools

- Drawing
  - Microsoft whiteboard https://whiteboard.office.com
  - draw.io https://app.diagrams.net/
- UML-specific drawing
  - ArgoUML, Microsoft Visio, OmniGraffle, etc.
- UML in plain text (as programming language)
  - Mermaid https://mermaid.live/edit
  - PlantUML https://www.plantuml.com/
- Different tools produce slightly different diagrams
  - don’t get stuck in the details
  - make sure the notations in the diagrams are consistent

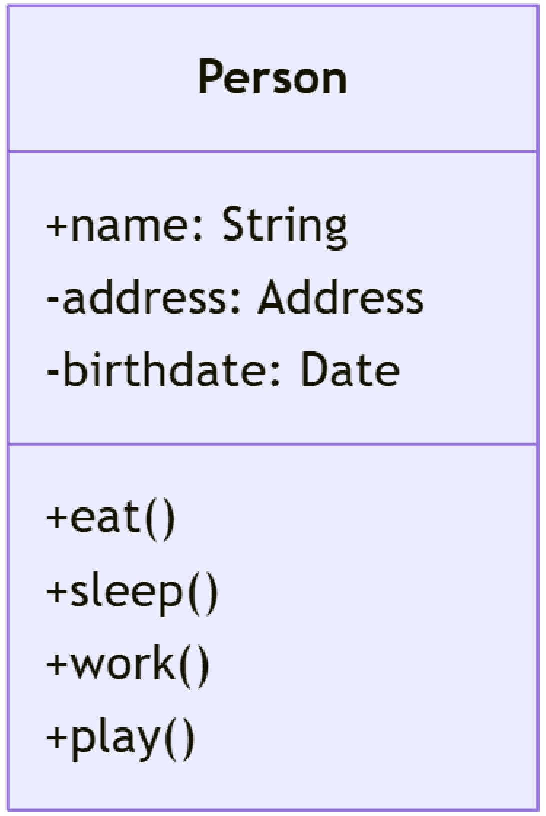
Class Diagram

- Definition: describe the types of objects in the system and the various kinds of static relationships that exist among them;
  show the properties and operations (features) of a class and
  the constraints that apply to the way objects are connected

- Element
  - class (with annotations: interface, enumeration, exception)
  - package

- Relationships
  - association, aggregation, composition
  - generalization
  - dependency

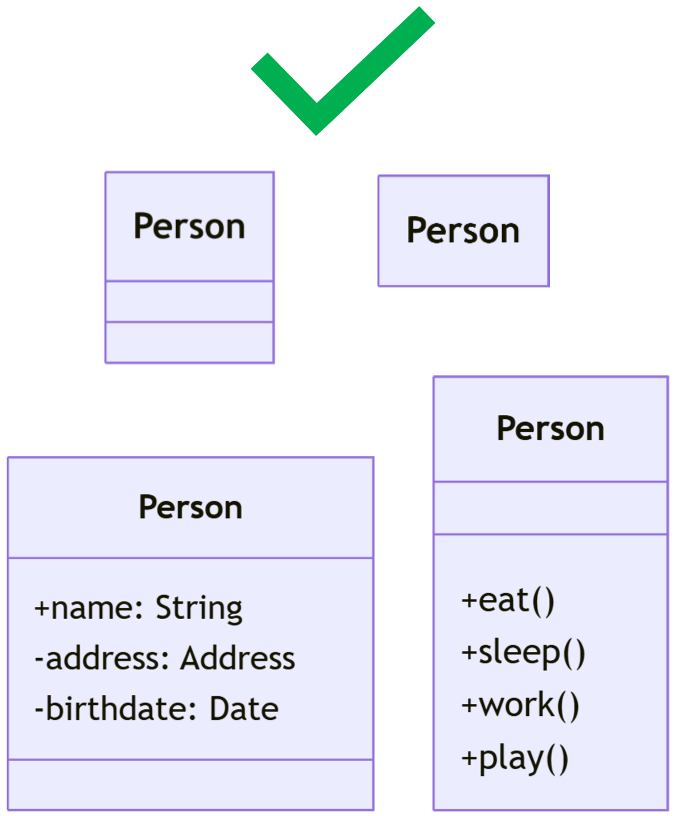
Class Diagram – Class



class name (required)

attributes (optional)
~= fields
structural features of a class

operations (optional)
~= methods/functions
actions that a class knows to carry out



Class Diagram – Class – Attributes & Operations

- Attributes
  - `visibility name: type [multiplicity] = default {property-string}`
  - example: `+name: String [1] = “Untitled” {readOnly}`
  - example: `-address: Address`
- Operations
  - `visibility name (parameter-list): return-type {property-string}`
  - example: `+getPhone(n: Name, a: Address): PhoneNumber`
  - example: `+eat()`

visibility

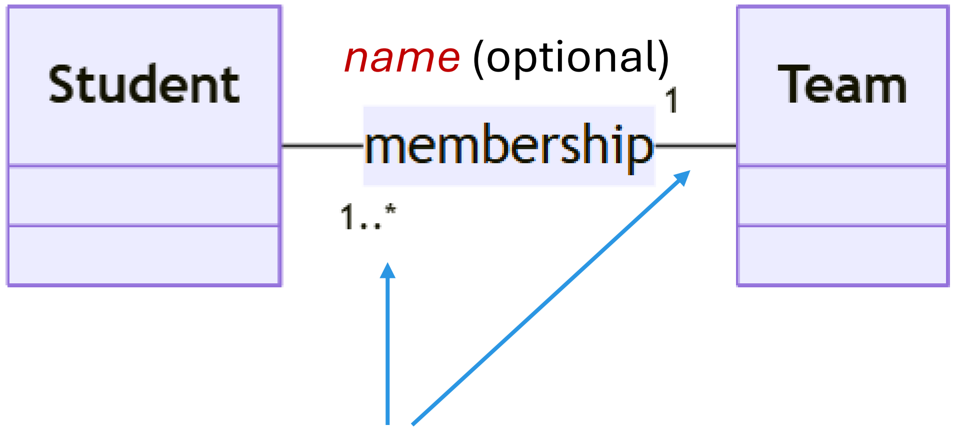
+: public	-: private
~: package	#: protected

multiplicity

how many objects may fill the property

1:	single-valued, exactly one
0..1:	optional, zero or one
*:	any number, zero or more
1..*:	one or more

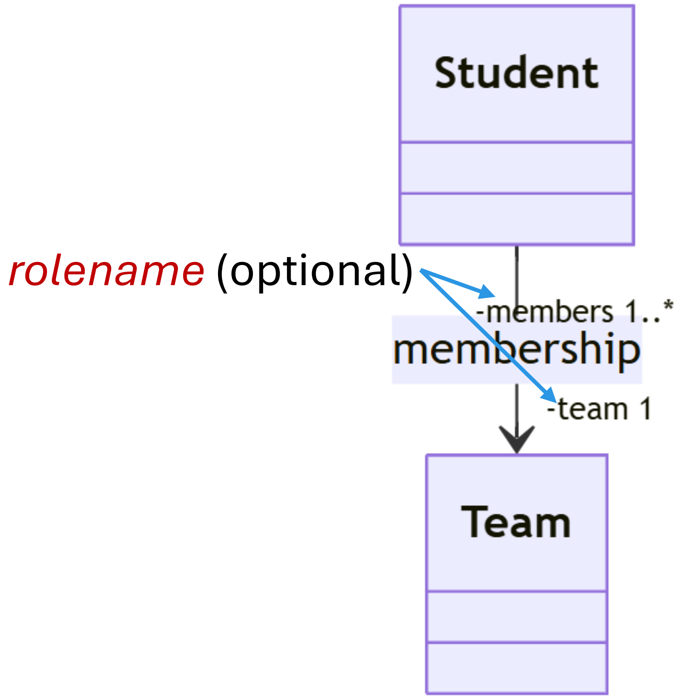
Class Diagram – Association



multiplicity (optional)

1:	single-valued, exactly one
0..1:	optional, zero or one
*:	any number, zero or more
1..*:	one or more

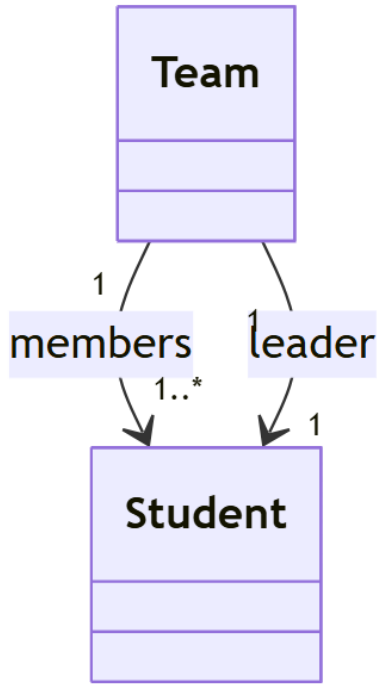
association
bidirectional / unidirectional
two classes that communicate with each other
another way to notate a property (other than attributes)



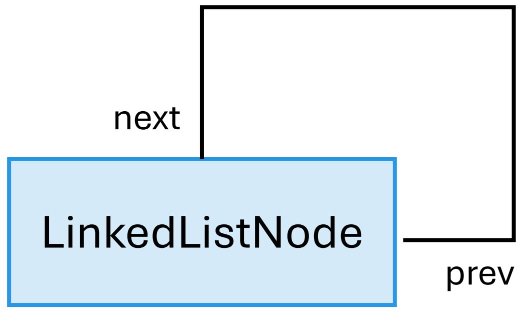
Class Diagram – Association (cont.)

✓

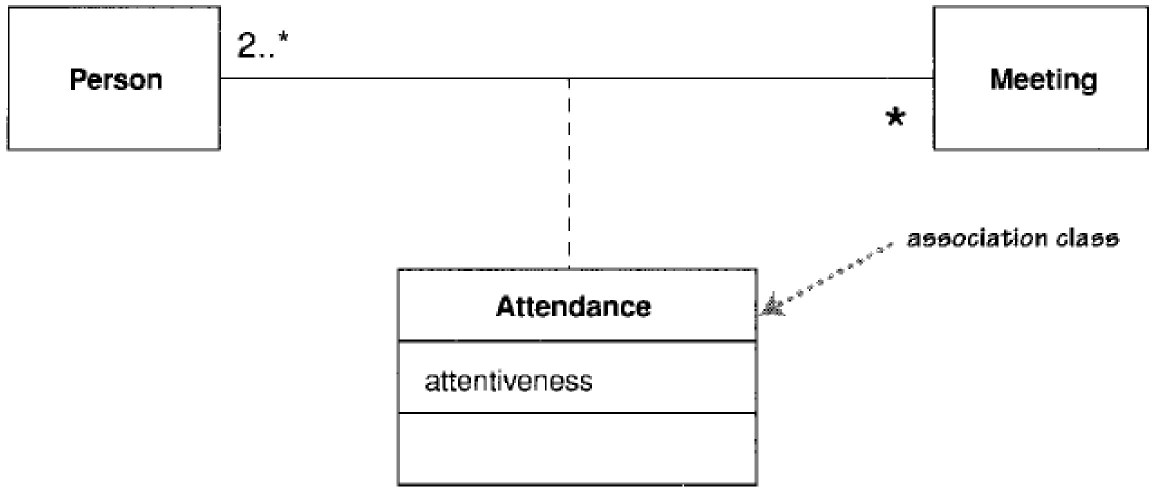
dual association between two classes



self association



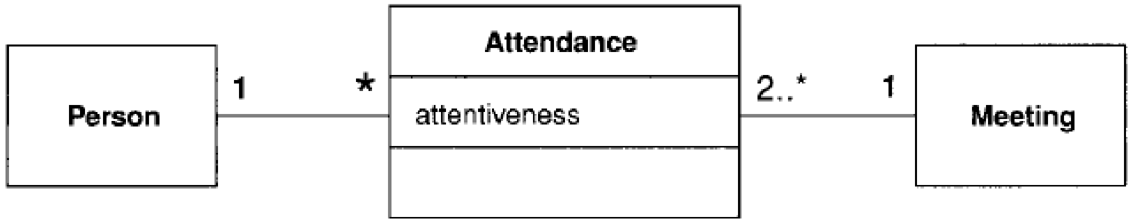
Class Diagram – Association (cont.)



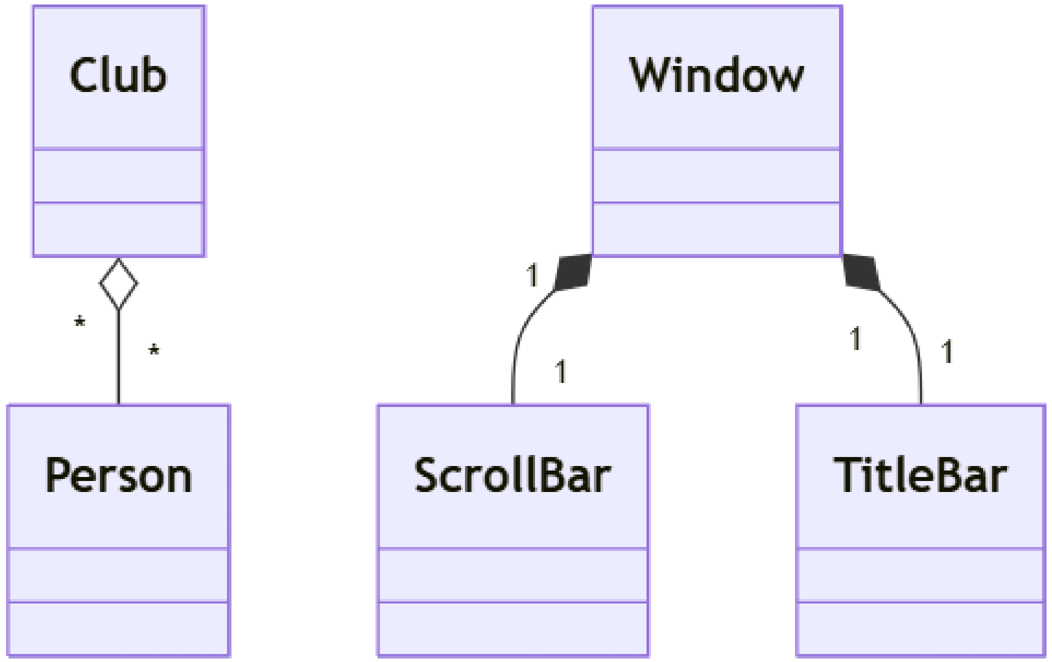
association class

allow adding attributes & operations to associations

can be prompted to a full class



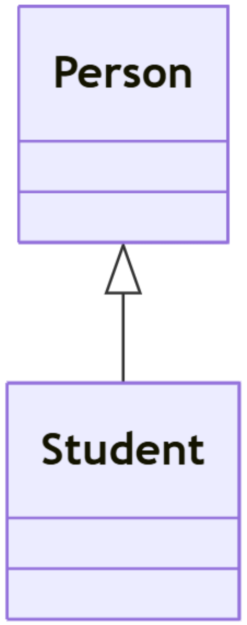
Class Diagram – Aggregation & Composition



aggregation
a whole-part relationship between
an aggregate (whole) and a constituent part,
where the part can exist independently from the aggregate

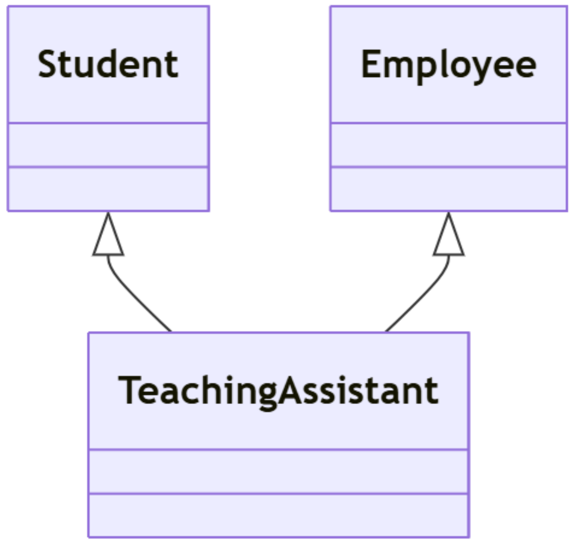
composition
a strong ownership and coinficient lifetime of
parts by the whole

Class Diagram – Generalization

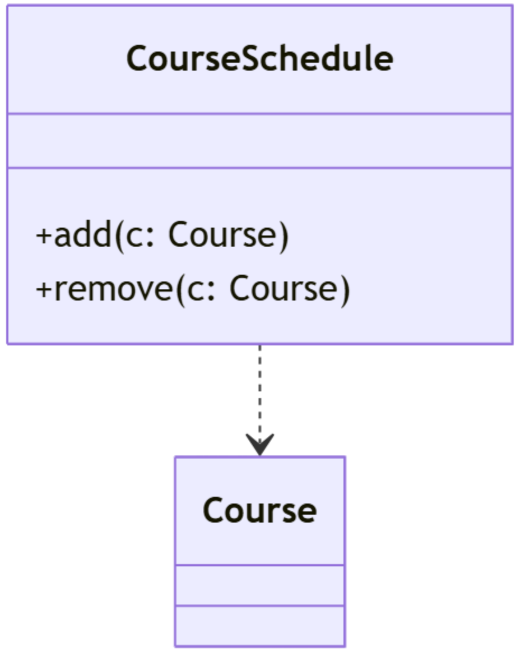


generalization
connects a subclass to its superclass
inheritance of attributes and operations
from the superclass to the subclass

✓ multiple inheritance

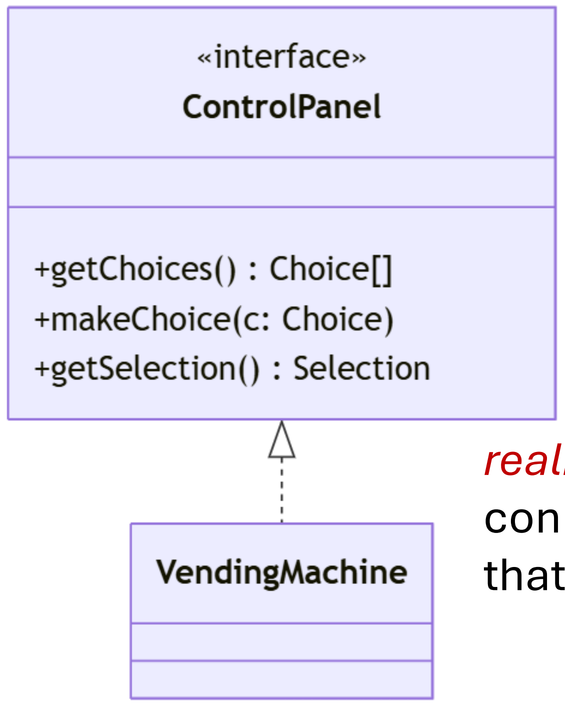


Class Diagram – Dependency



dependency
a semantic relationship between two elements

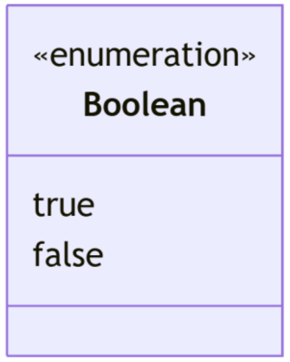
Class Diagram – Interface



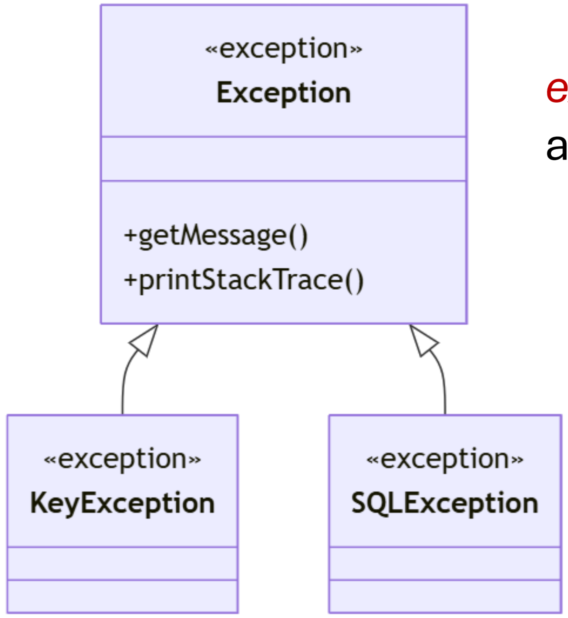
interface
a class that has no implementation
most likely no attributes, only operations

realization
connects a class with an interface
that supplies its behavioral specification

Class Diagram – Enumeration, Exception

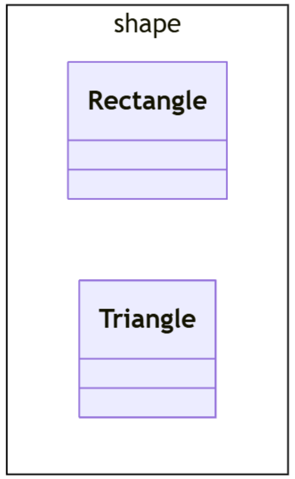


enumeration
a user-defined data type that consists of an ordered list of enumeration literals



exception
a class representing exceptional state of certain type

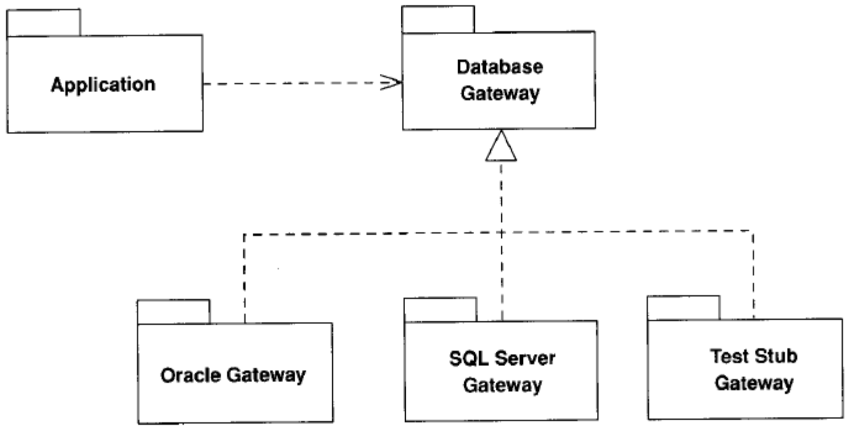
Class Diagram – Package



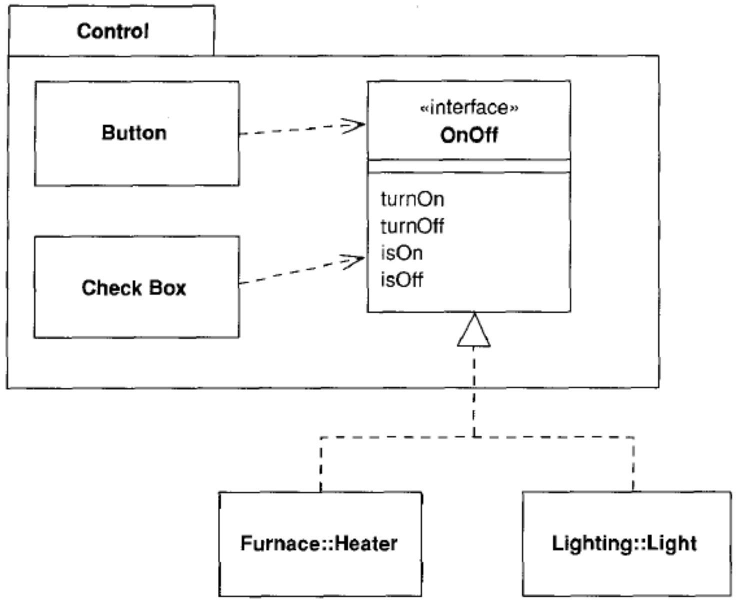
package
container-like element for organizing
other elements (classes, packages) into groups

Class diagram with packages can also be called as Package Diagram

dependencies between packages

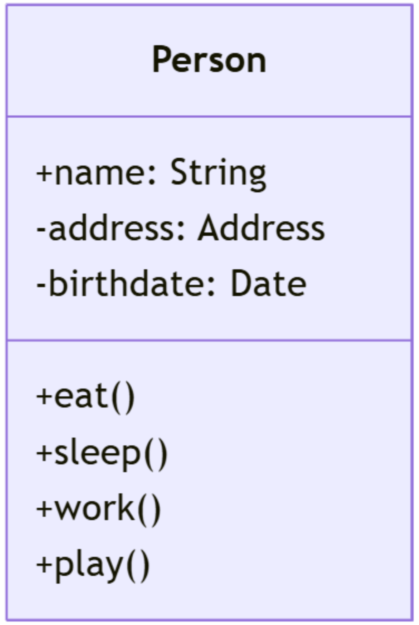


relationships between classes across packages



Database Design

- Class diagram can be a handy tool for designing your data model
  - data model: describing how real-world data is conceptually represented as computerized information, and the types of operations available to access and update this information

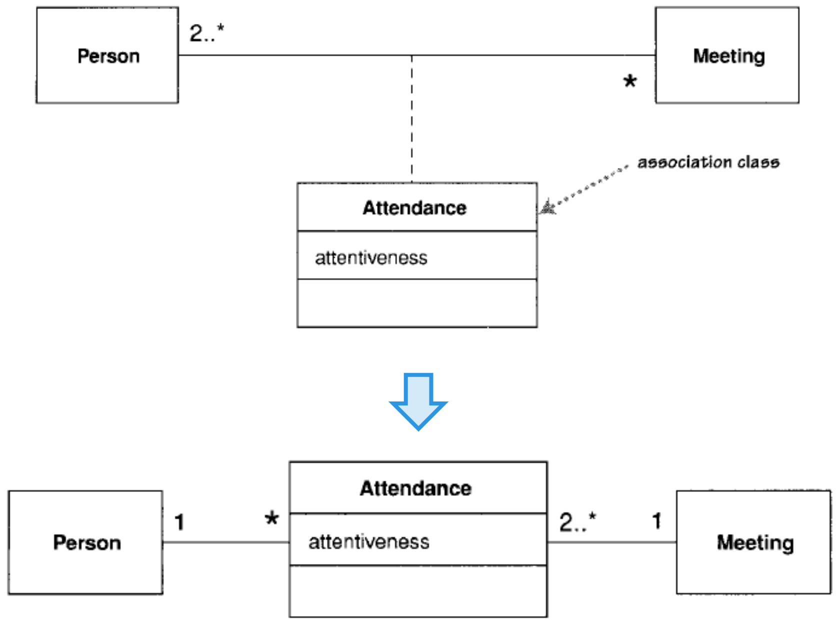


class name -> *table name*

attributes -> *columns* (name and type)

select/add an attribute as *primary key*

association -> *relationship*



Relational Database Normal Forms

- 1NF: each column contains atomic values of a single type
  - avoid collections/arrays attribute; use associations instead
  - e.g., phoneNumbers: String[] -> class PhoneNumber + 1..* association
- 2NF: all non-key attributes are fully functionally dependent on the primary key
  - decompose classes to eliminate partial dependencies
  - e.g., class Order { orderId, productId }
    class Product { productId, productName }
- 3NF: all attributes are functionally dependent only on the primary key
  - create new classes to eliminate transitive dependencies
  - e.g., class Employee { employeeId, departmentId }
    class Department { departmentId, departmentName }
- …

Relational Database Normal Forms

- Tradeoffs of using higher normal forms
  - robustness: 👍 less redundancy in database, better data integrity
  - scalability:
    👍 easier to scale up vertically (add more data into a table);
    👎 harder to scale up horizontally (add more classes/tables)
  - efficiency:
    👍 important queries can be more efficient with less data redundancy;
    👎 some queries can become complex and require multiple joins
  - complexity: 👎 too many components (classes) and connections
- Consider: What classes will have a lot of data to store?
  Which classes do you plan to put or not put into database?

Object/Relational Mapping Framework

- ORM: converting data between a relational database and memory of an object-oriented programming language
- Operate on a virtual object database using APIs (no need for SQL)
- Examples
  - Firebase – Kotlin / Android
  - Hibernate – Java
  - SQLAlchemy – Python

Agenda (recap)

- UML Introduction
- Class diagram
- Data model
- Additional UML References
    - *UML Distilled – Applying the Standard Object Modeling Language* by Martin Fowler and Kendall Scott
    - *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (3rd Edition)* by Craig Larman
- Review P2: Project Proposal requirements
- P1: Project Setup due this Friday!