Software Design and Architecture

Software Modeling with UML

Agenda

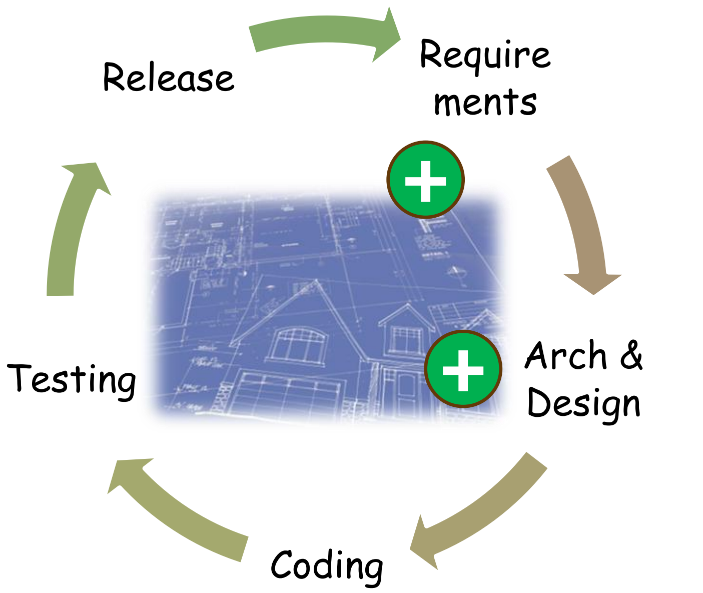
- why model software
- notation: UML
- component diagram
- class diagram

Winter 2026. Instructor: 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.

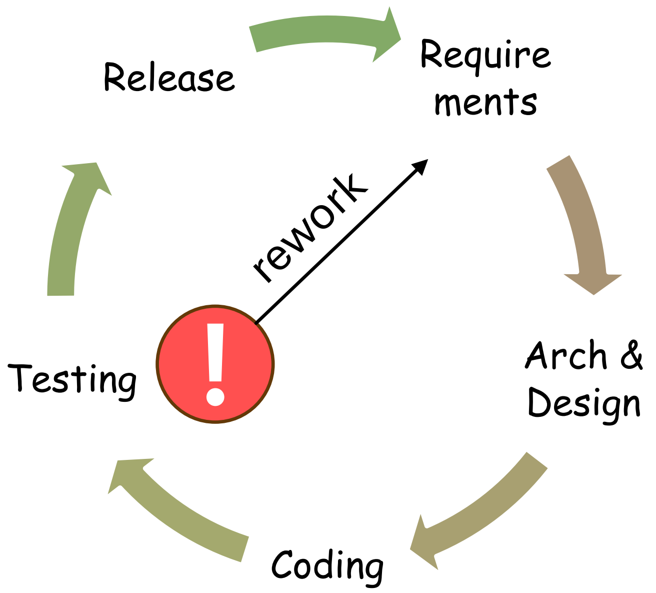
Why Model Software

- Document architecture and design decisions
- Reduce ambiguity; reason about missing requirements, risk, change, etc.
- Align teammates on responsibilities



document & iterate decisions early (when changes are cheap)

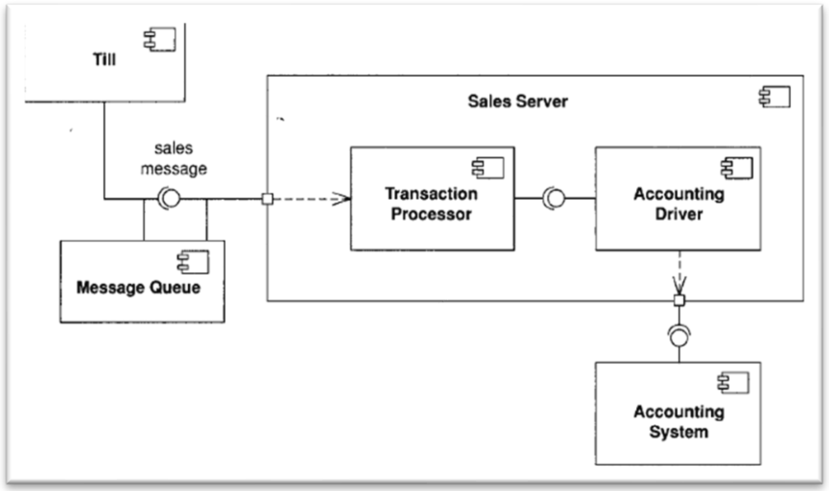
vs



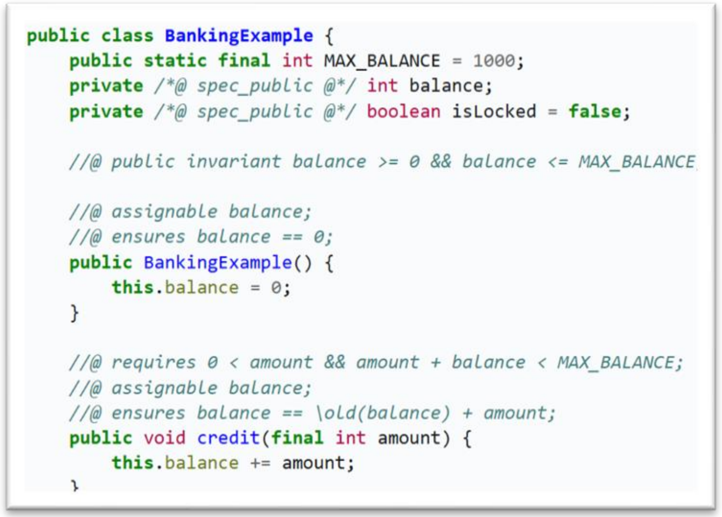
risk bigger rework later

What is a Software Model?

- An abstract representation of a software for a purpose
  - focus on one aspect / component / process
  - can be graphical or textual
- General principles of software modeling
  - model the essentials
  - provide perspective
  - enable effective communication



UML
(unified
modeling
language)



```
public class BankingExample {
    public static final int MAX_BALANCE = 1000;
    private /*@ spec_public @*/ int balance;
    private /*@ spec_public @*/ boolean isLocked = false;

    //@ public invariant balance >= 0 && balance <= MAX_BALANCE

    //@ assignable balance;
    //@ ensures balance == 0;
    public BankingExample() {
        this.balance = 0;
    }

    //@ requires 0 < amount && amount + balance < MAX_BALANCE;
    //@ assignable balance;
    //@ ensures balance == \old(balance) + amount;
    public void credit(final int amount) {
        this.balance += amount;
    }
```

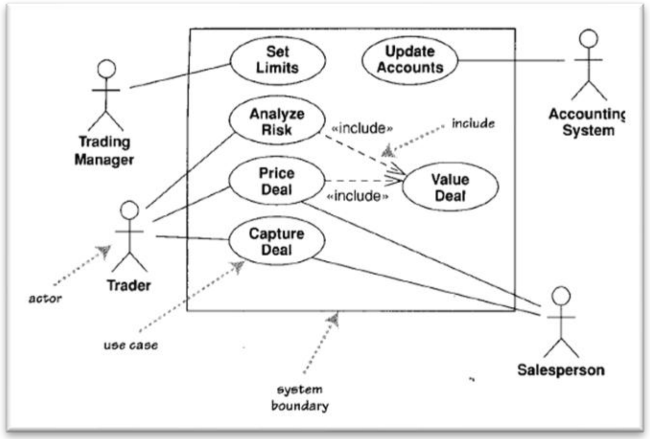
JML
(java modeling
language)

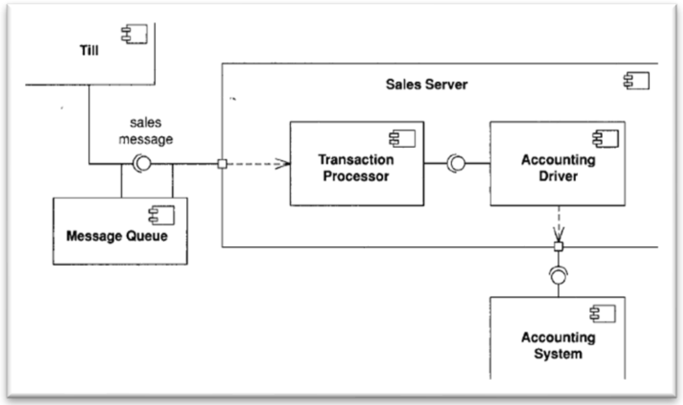
Reference: https://www.computer.org/resources/software-engineering-models#modeling

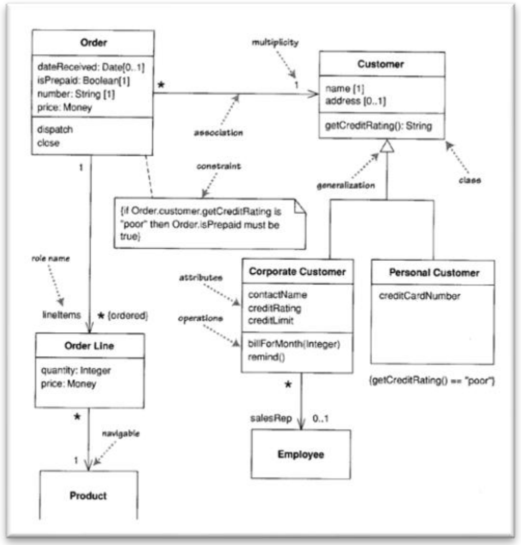
Software Modeling x Lifecycle

Requirements → Architecture → Design → Coding

Focus	goals & user scenarios	components & communication	classes & APIs
Example UML diagrams	• use case diagram	• component diagram • communication diagram • activity diagram	• class diagram • sequence diagram • state machine diagram

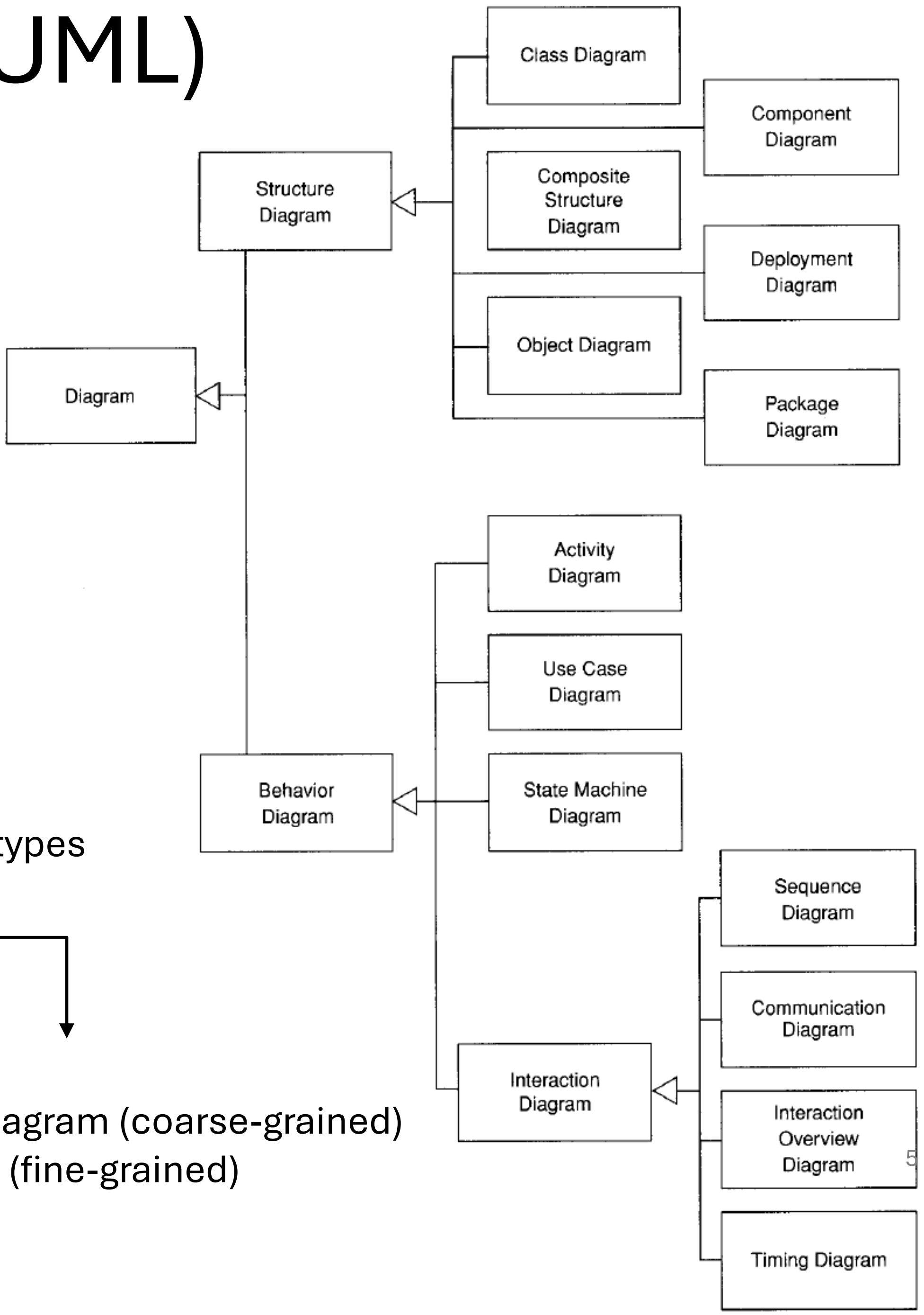






Unified Modeling Language (UML)

- UML is a set of notations, not a methodology or process
  - official standard backed by OMG, version 2.5.1
- UML doesn't solve your problems for you, it gives you a way of writing them down
- Focus on the parts that are useful to you



classification of UML diagram types
overall two classes:
- for static structure
- for dynamic behavior

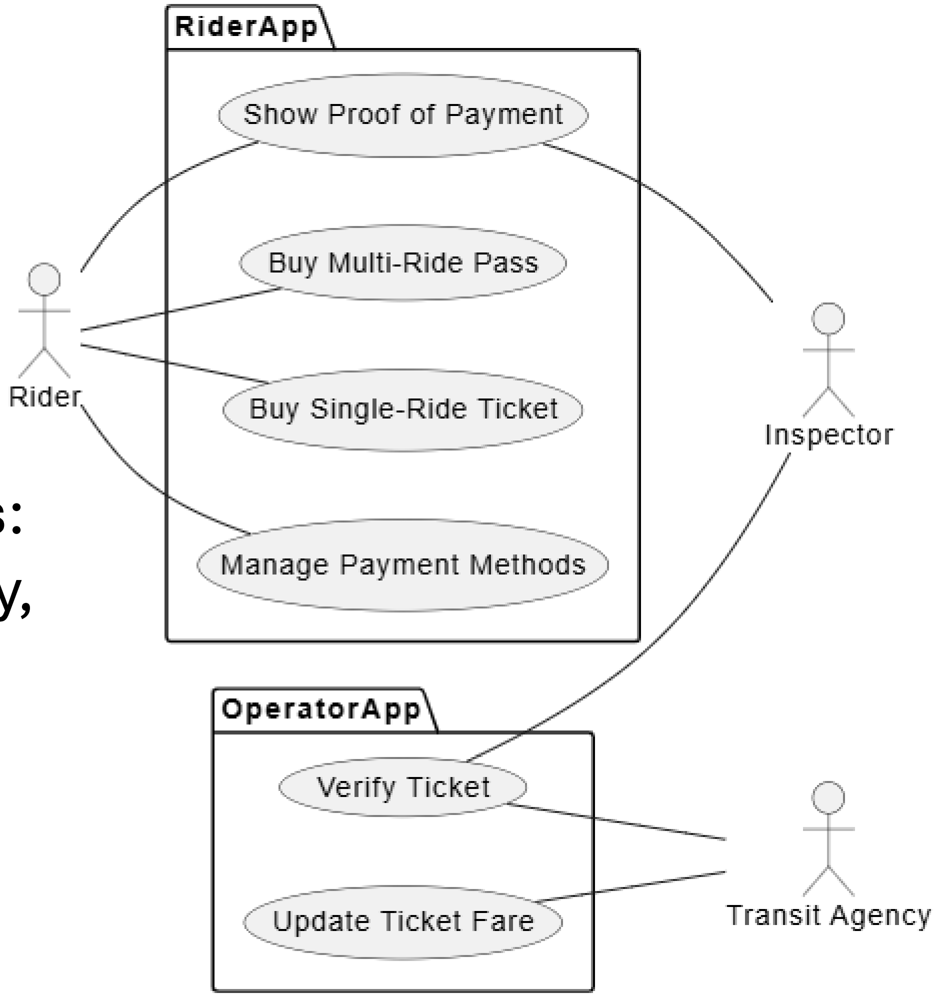
today's focus
- component diagram (coarse-grained)
- class diagram (fine-grained)

UML Tools

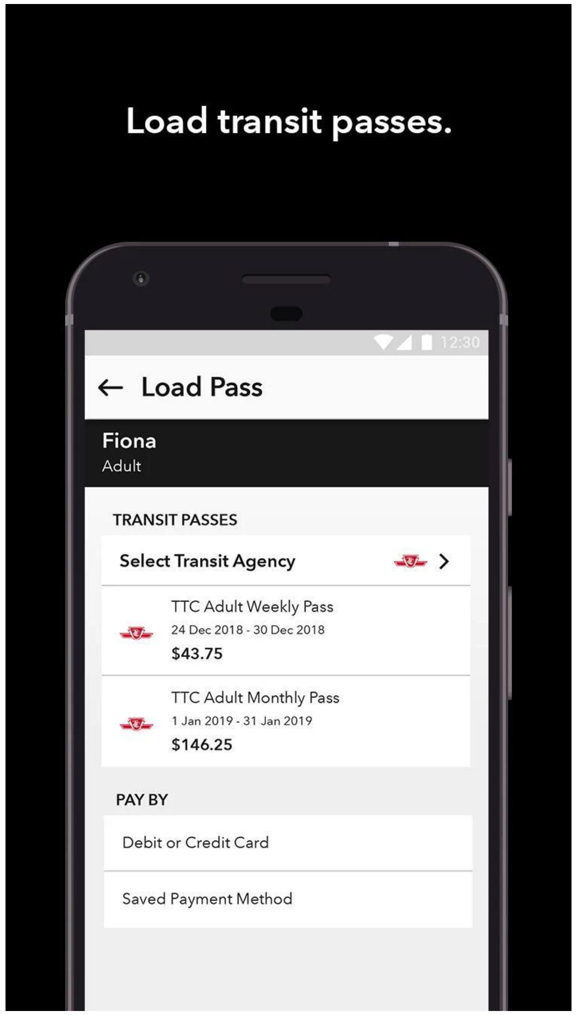
- Drawing
  - Microsoft whiteboard https://whiteboard.office.com
  - draw.io https://app.diagrams.net/
- UML-specific drawing
  - ArgoUML, Microsoft Visio, OmniGraffle, etc.
- UML in plain text (as programming language)
  - Mermaid https://mermaid.live/edit
  - PlantUML https://www.plantuml.com/
- Different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

Running Example

- Transit ticketing app
- Features:
  single-ride ticket,
  day/week/month pass,
  ticket wallet, ...
- Non functional requirements:
  usability, efficiency, reliability,
  security, ...



Use Case Diagram


Load transit passes.

← Load Pass

Fiona
Adult

TRANSIT PASSES

Select Transit Agency

TTC Adult Weekly Pass
24 Dec 2018 - 30 Dec 2018
$43.75

TTC Adult Monthly Pass
1 Jan 2019 - 31 Jan 2019
$146.25

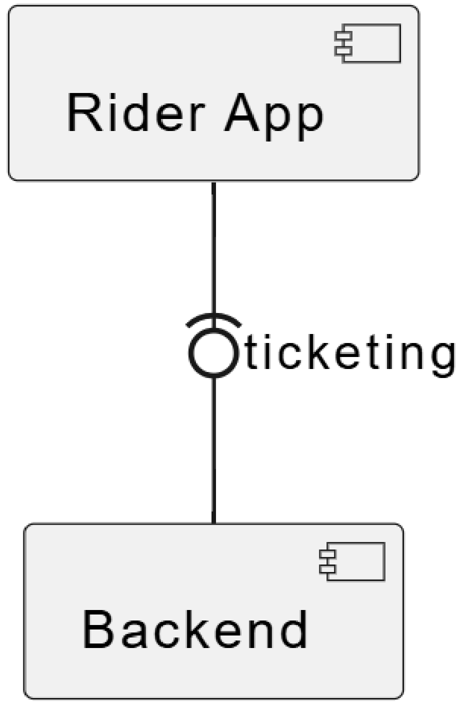
PAY BY

Debit or Credit Card

Saved Payment Method

Component Diagram

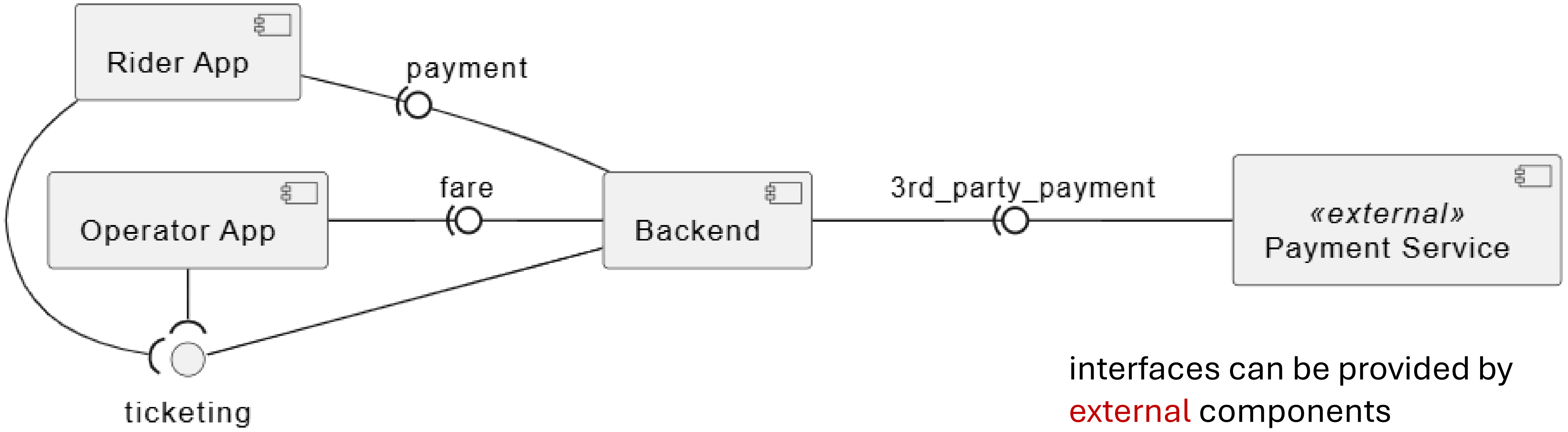
- Shows the organization and dependencies between components/subsystems



component

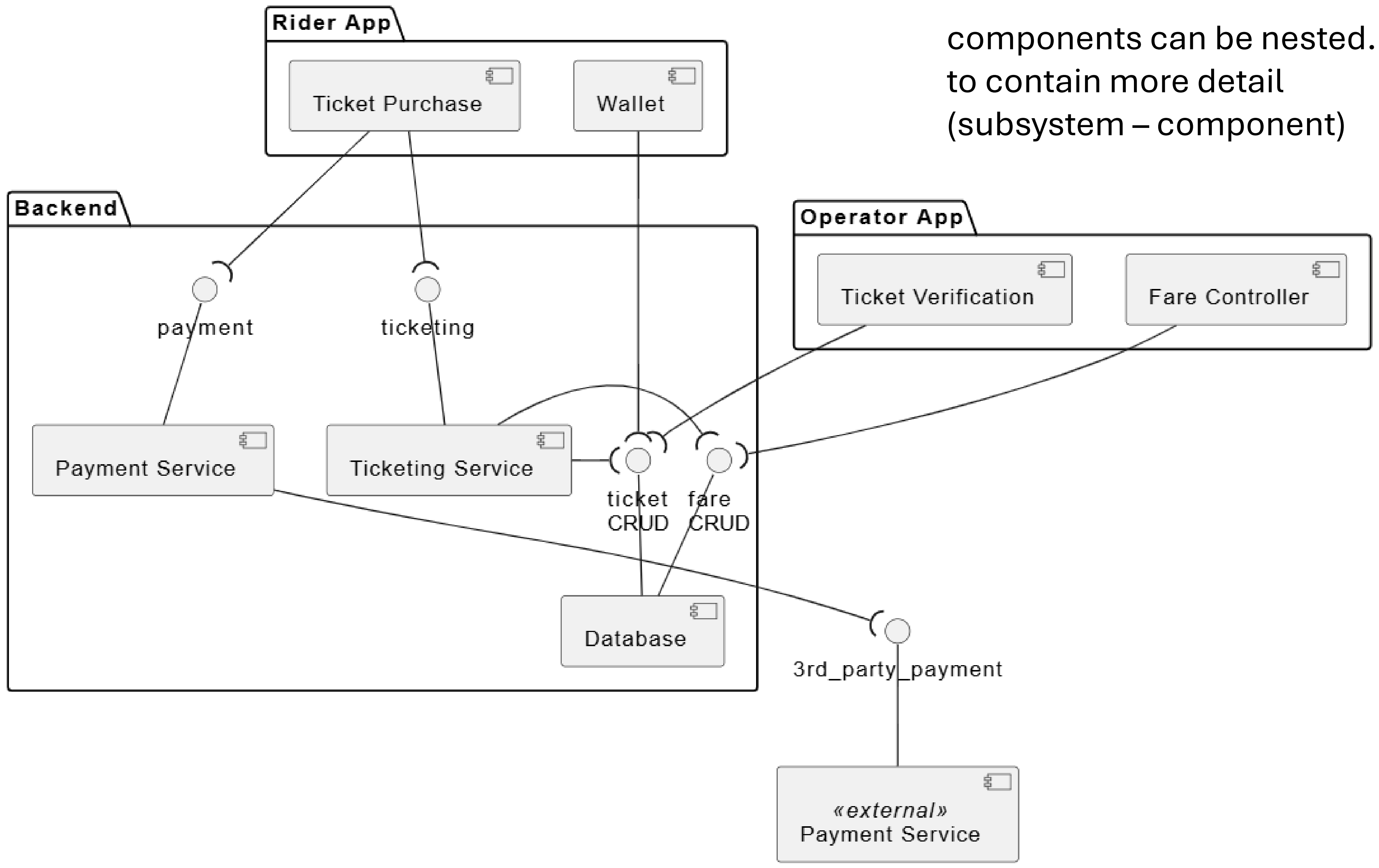
interface
connectors between two components
lollipop: provide an interface
socket: use an interface

Component Diagram



interfaces can be provided by external components

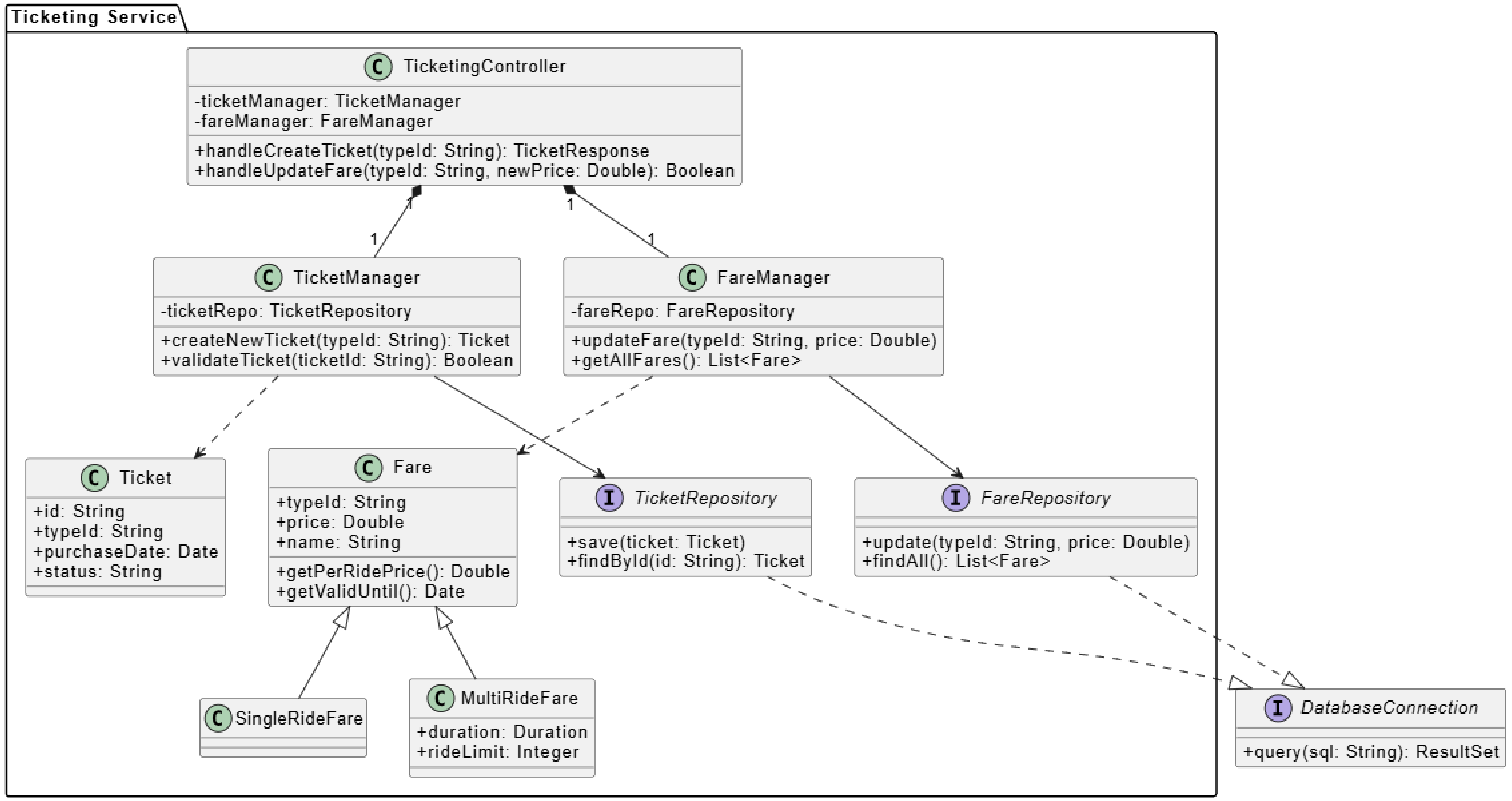
Component Diagram



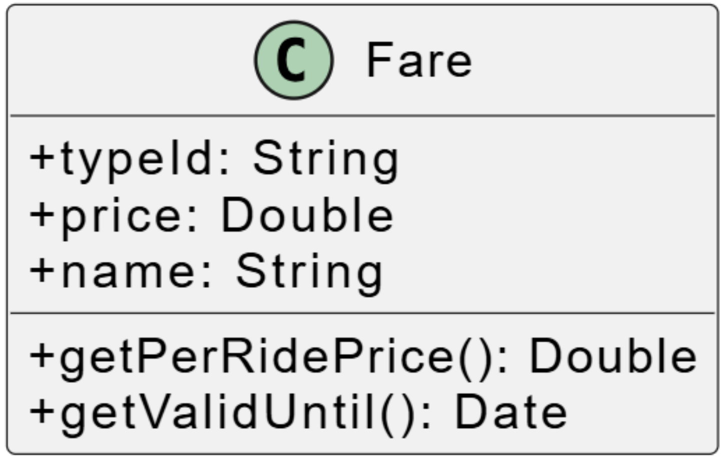
components can be nested…
to contain more detail
(subsystem – component)

Class Diagram

- Describe the types of objects in a component/system and their relationships


Ticketing Service
TicketingController
-ticketManager: TicketManager
-fareManager: FareManager
+handleCreateTicket(typeId: String): TicketResponse
+handleUpdateFare(typeId: String, newPrice: Double): Boolean
1
1
1
1
TicketManager
-ticketRepo: TicketRepository
+createNewTicket(typeId: String): Ticket
+validateTicket(ticketId: String): Boolean
FareManager
-fareRepo: FareRepository
+updateFare(typeId: String, price: Double)
+getAllFares(): List<Fare>
Ticket
+id: String
+typeId: String
+purchaseDate: Date
+status: String
Fare
+typeId: String
+price: Double
+name: String
+getPerRidePrice(): Double
+getValidUntil(): Date
TicketRepository
+save(ticket: Ticket)
+findById(id: String): Ticket
FareRepository
+update(typeId: String, price: Double)
+findAll(): List<Fare>
SingleRideFare
MultiRideFare
+duration: Duration
+rideLimit: Integer
DatabaseConnection
+query(sql: String): ResultSet

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

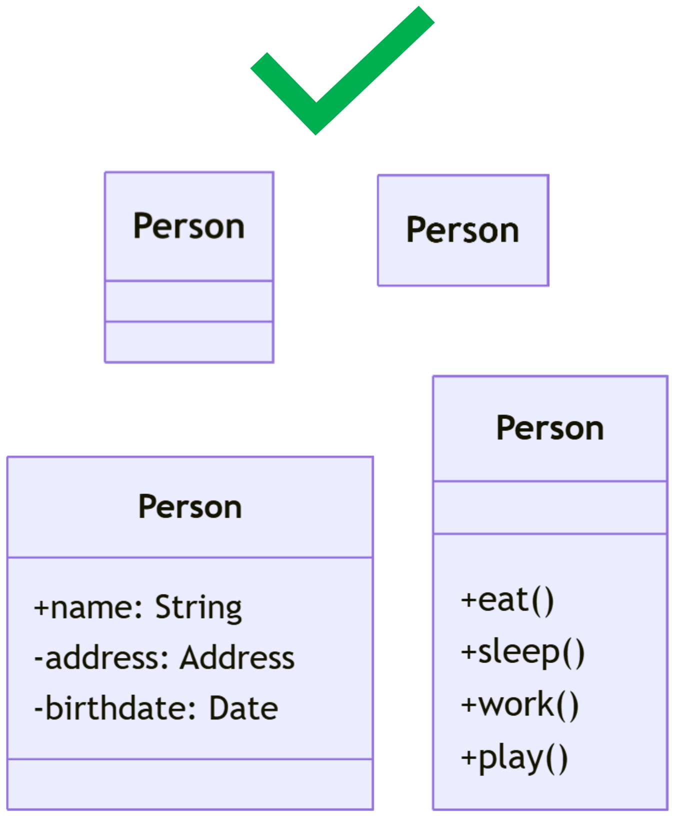
attribute format

```
visibility name: type [multiplicity] = default {property-string}
```

operation format

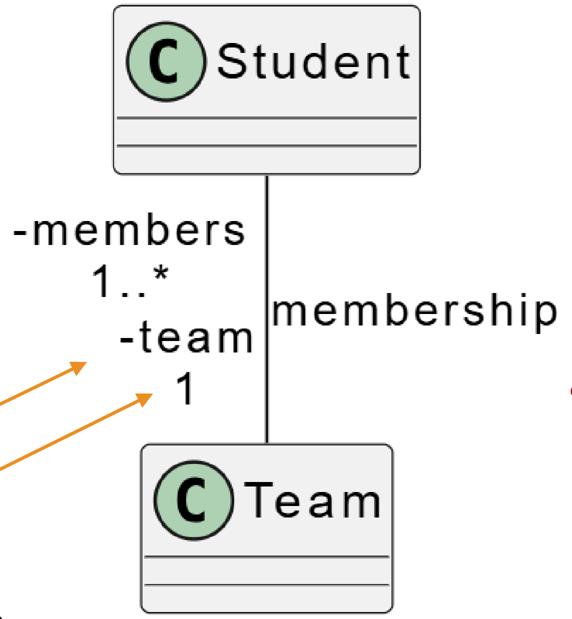
```
visibility name (parameter-list): return-type {property-string}
```

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



Class Diagram – Association

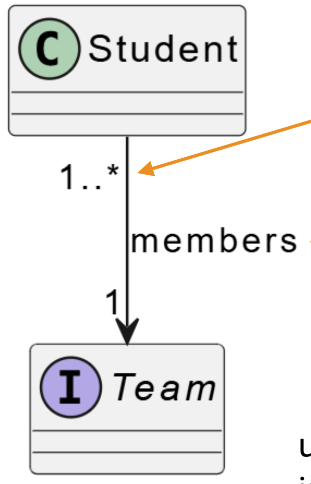
association
two classes that communicate with each other
another way to notate a property (other than attributes)



association name
(optional)

role name (optional)
multiplicity (optional)

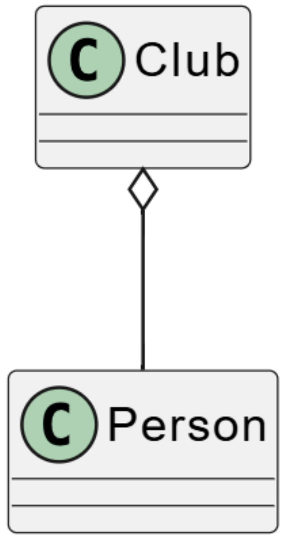
1:	single-valued, exactly one
0..1:	optional, zero or one
*:	any number, zero or more
1..*:	one or more

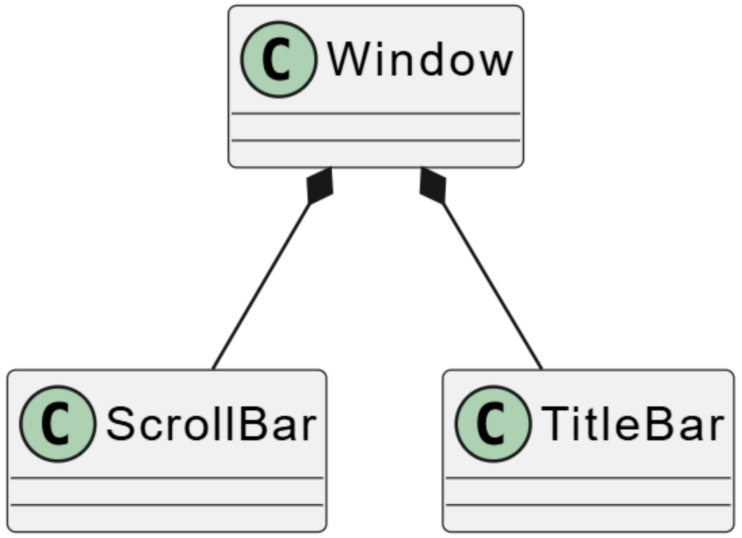


bidirectional ver.

unidirectional ver.
implies some kind of flow

Class Diagram – Aggregation & Composition

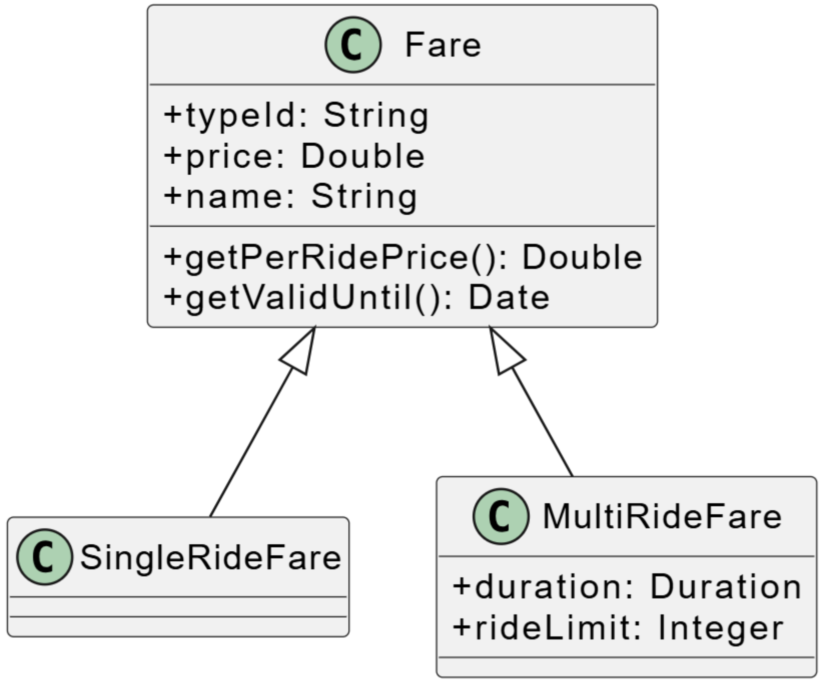


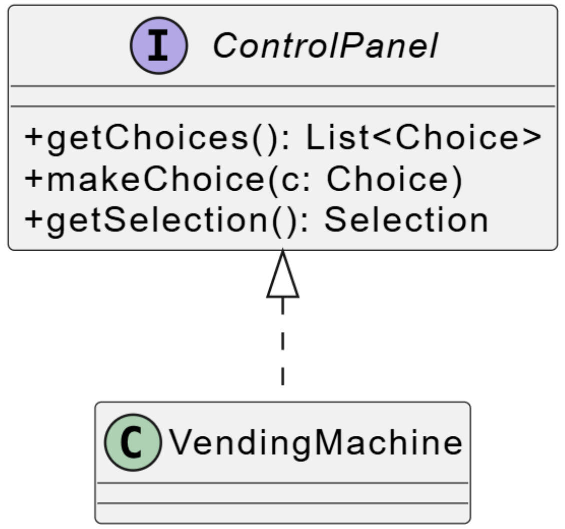


aggregation
a whole-part relationship between
an aggregate (whole) and a constituent part,
where the part can exist independently from the aggregate

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

Class Diagram – Generalization & Realization

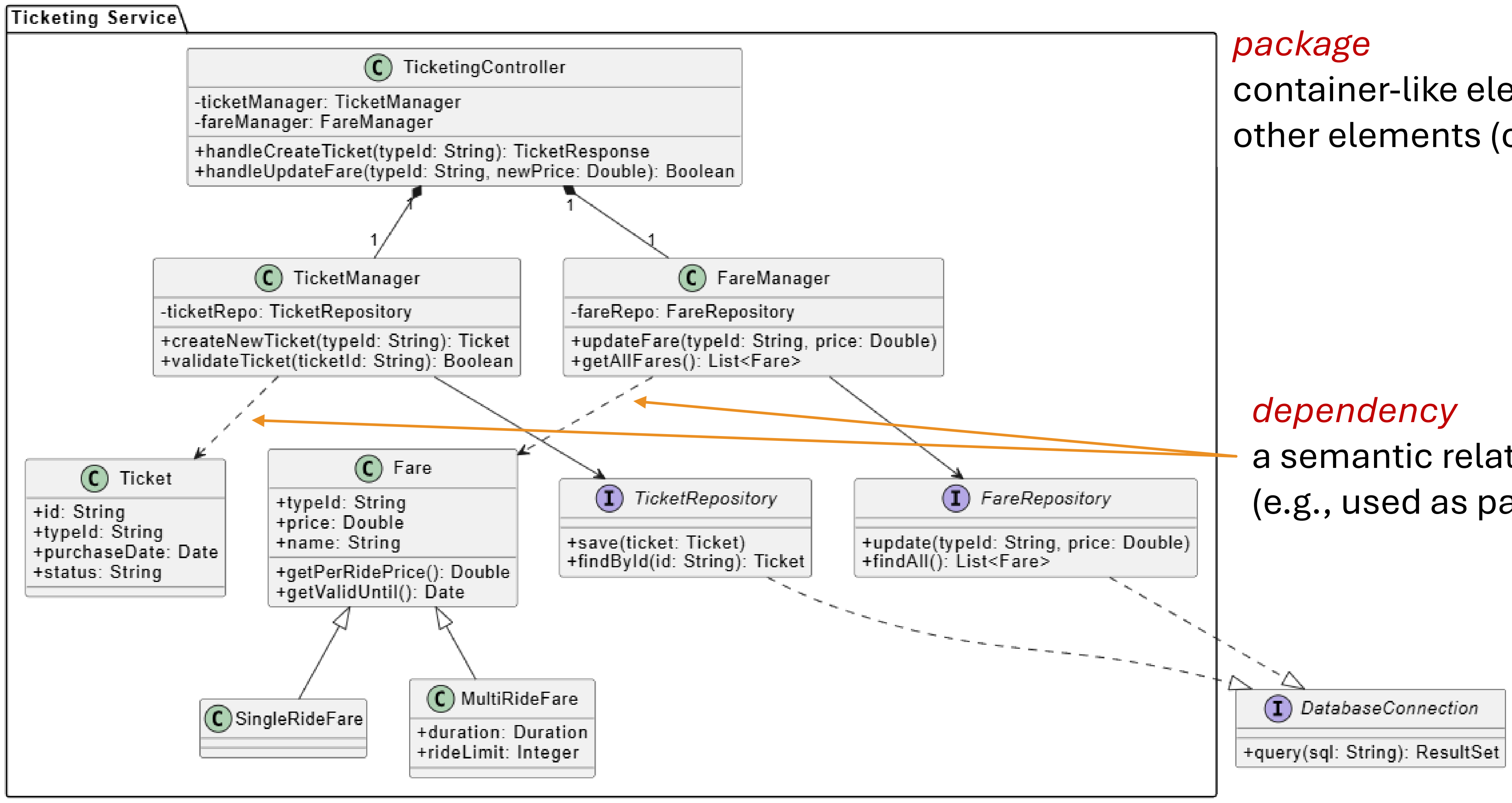




generalization `*extends*`
connects a subclass to its superclass
inheritance of attributes and operations
from the superclass to the subclass

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

Class Diagram – Dependency & Package



Ticketing Service
TicketingController
-ticketManager: TicketManager
-fareManager: FareManager
+handleCreateTicket(typeId: String): TicketResponse
+handleUpdateFare(typeId: String, newPrice: Double): Boolean
1
1
1
1
TicketManager
-ticketRepo: TicketRepository
+createNewTicket(typeId: String): Ticket
+validateTicket(ticketId: String): Boolean
FareManager
-fareRepo: FareRepository
+updateFare(typeId: String, price: Double)
+getAllFares(): List<Fare>
Ticket
+id: String
+typeId: String
+purchaseDate: Date
+status: String
Fare
+typeId: String
+price: Double
+name: String
+getPerRidePrice(): Double
+getValidUntil(): Date
TicketRepository
+save(ticket: Ticket)
+findById(id: String): Ticket
FareRepository
+update(typeId: String, price: Double)
+findAll(): List<Fare>
SingleRideFare
MultiRideFare
+duration: Duration
+rideLimit: Integer
DatabaseConnection
+query(sql: String): ResultSet

package

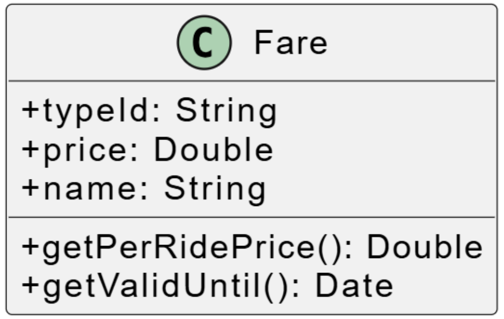
container-like element for organizing other elements (classes, packages) into groups

dependency

a semantic relationship between two elements (e.g., used as parameter or return type)

Class Diagram -> Data Model

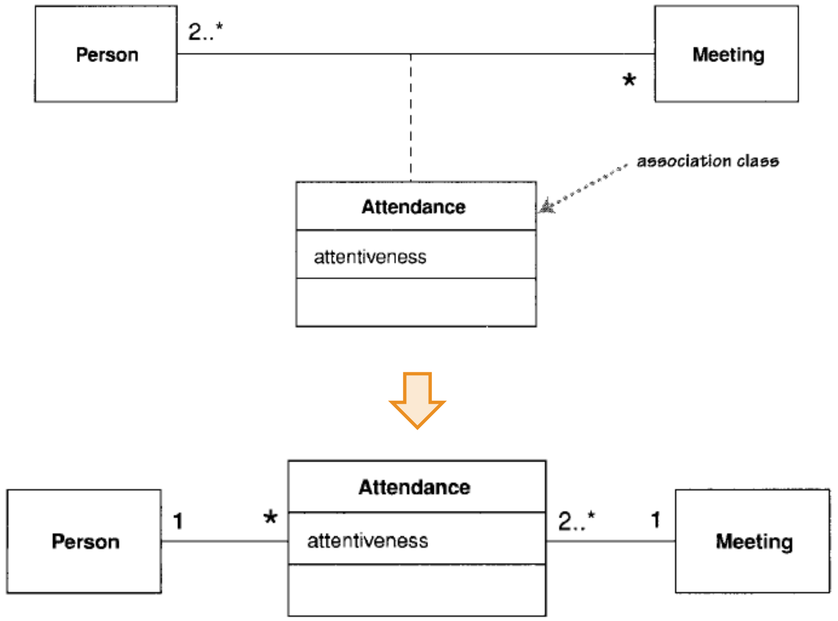
- Class diagram can be a handy tool for designing your data model
  - data model: describing how real-world data is conceptually represented as computerized information, and the types of operations available to access and update this information



class name -> *table name*

attributes -> *columns* (name and type)

select/add an attribute as *primary key*



association -> *relationship*

Recap

- why model software
- notation: UML
    - (use case diagram)
    - component diagram
    - class diagram

- Complete P1 Project Setup by this Friday (Jan 23)
- A1 UML Practice is out; due on next Friday (Jan 30)