

# Design a parking lot management system

Tanmay Kacker

# Agenda

KISS

Keep it simple stupid

① Context / Overview

② Requirements

③ Clarifications

④ Design

→ Entities

→ Use case diagram

5-10

→ Class diagram

⑤

Implementation

→ Structure

→ POJOs / Models

→ Use cases

---

LLD → Low Level details

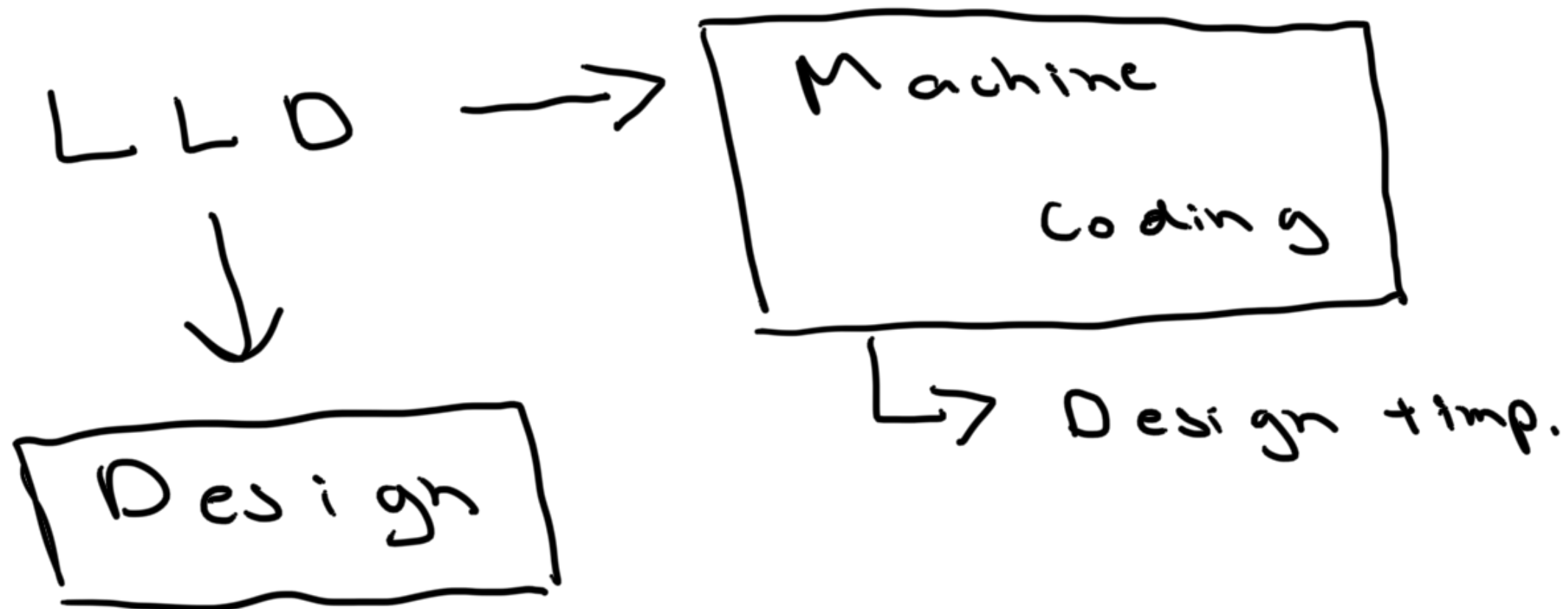
→ Classes

→ Functionalities

→ Relationship

→ Structure

— Design Patterns



→ Class diagram \* → classes

→ use case



functionalities

---

① Problem statement

→ Design tic tac toe  
Snakes & ladders

① Real world entity

→ Design a notebook

② Games

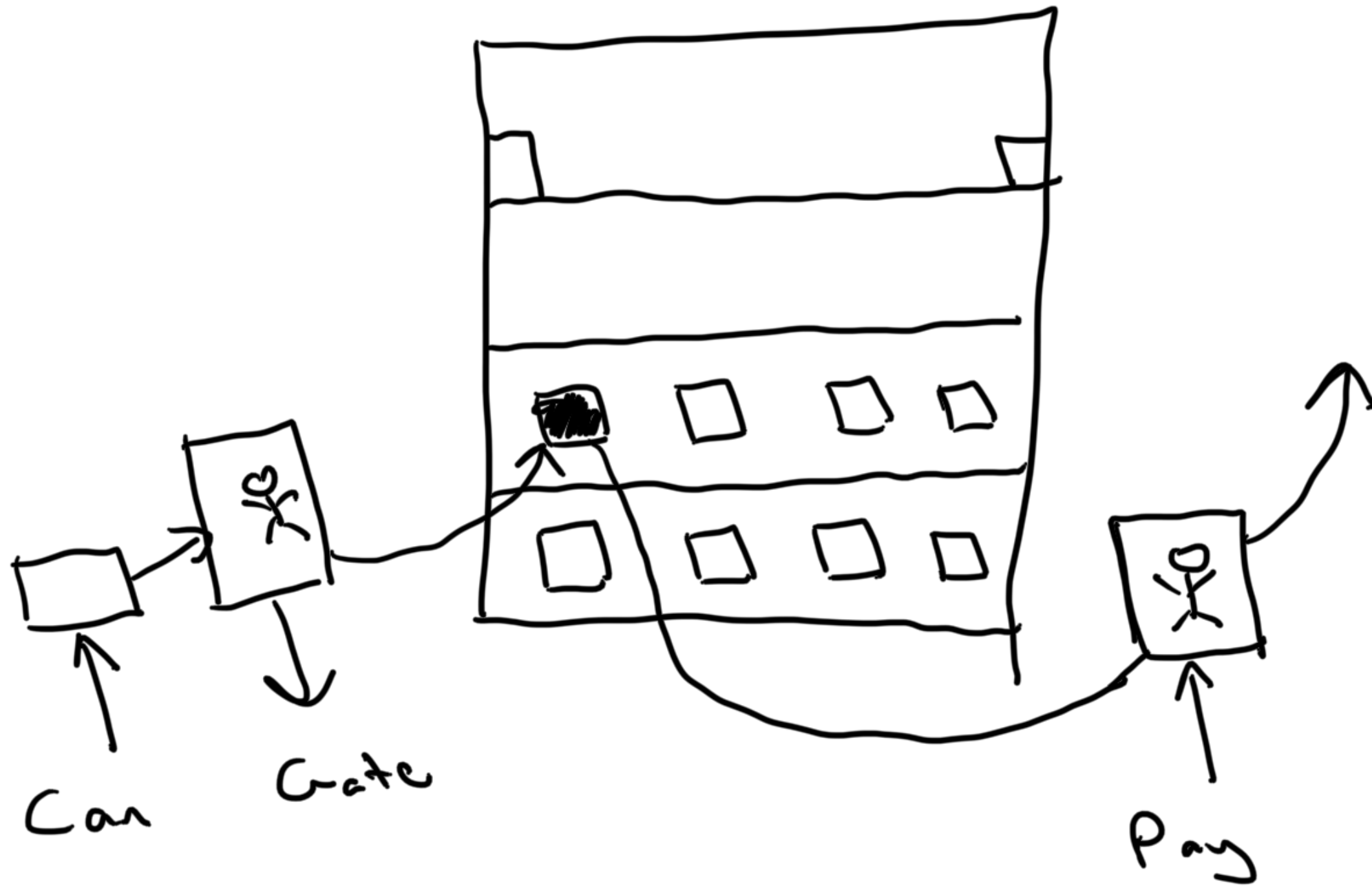
③ Management Systems

④\* Technical

→ URL Shortener

→ Cache

# ① Overview



## Questions

① Current scope -

① Can we only park cars?

② Will it have multiple floors?

③ How many entry/exit?

→ Multiple entry gates

→ Multiple exit gates



#### ④ Types<sup>of</sup> vehicles

① SMALL

② MEDIUM

③ LARGE



② Future scope

⑤ Behaviour

→ (How can we pay?)

→ How can we ...

→ How do we calculate fees?

→ How do we know if a slot is available?



Display board



$$\text{Cost} = \underline{\text{1st hour}} + \underline{\text{remaining hrs}}$$

SMALL - 50 , 90

MED - 80 , 100

LARGE - 100 , 120

How can we make pricing  
extensible?

Problem → questions



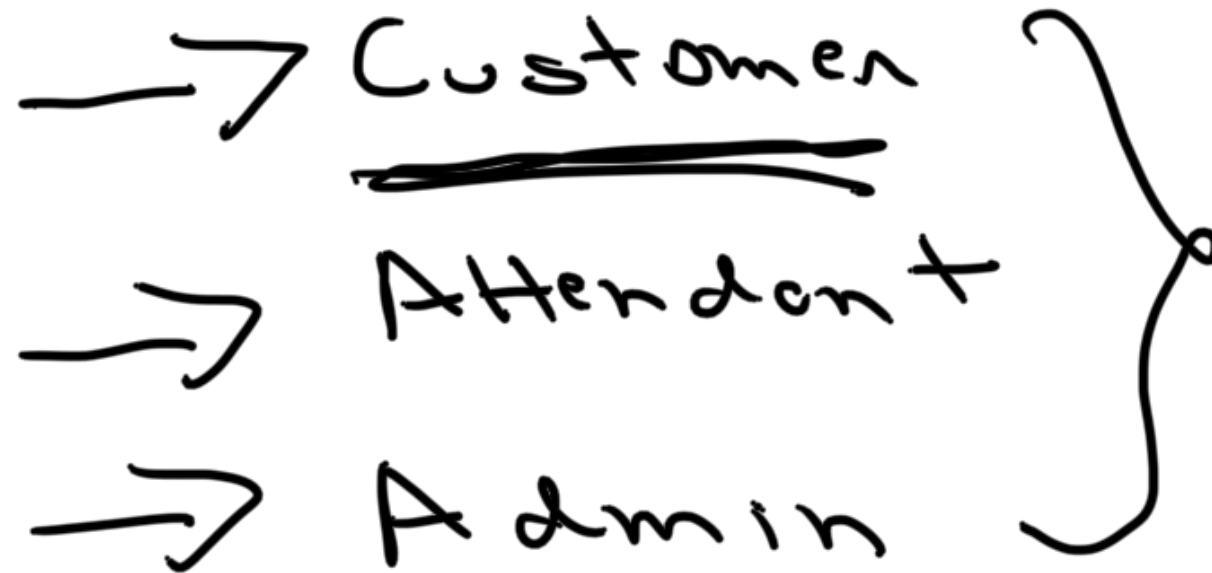
S - 10

---

Use case diagram

→ Actors

→ Customer  
→ Attendant  
→ Admin

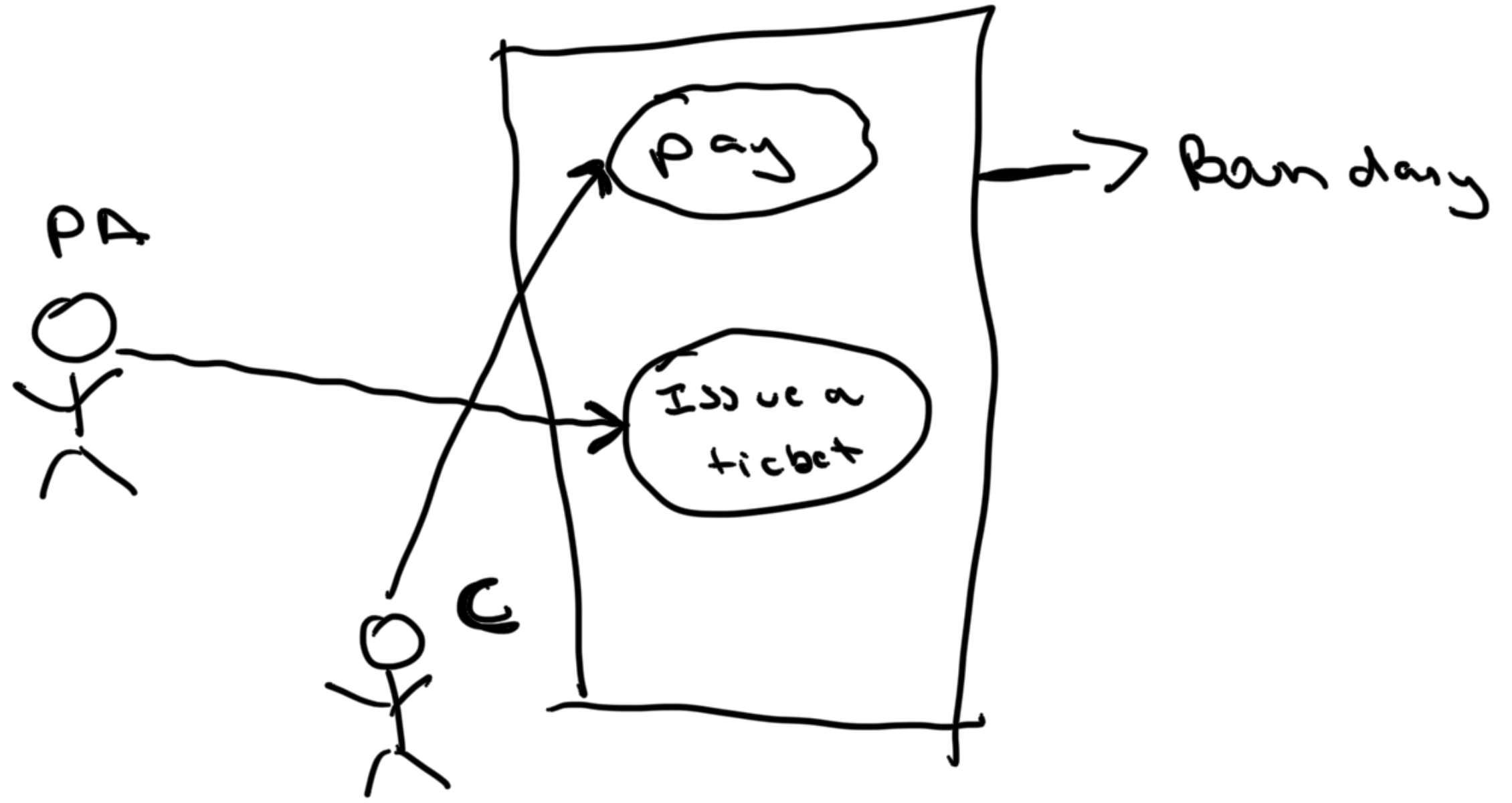


---

Use cases - functionalities



ADIS } features  
Commands }



UML

# Classes

- Parking lot
  - name ✓
  - address
  - Floors -
  - Entry gates -
  - Exit gates -
  - Display board
- Floor
  - = Floor #

- Parking Spot
- Display Board
- Payment Counter

→ Parking Spot

→ Spot #

→ Type - Small, Large, Medium  
- Enum

→ Status - Boolean - True  
- Enum

→ 0, 1, 2  
→ Chan

Ticket

→ Vehicle → #  
→ reg...

→ Spot |

→ Entry time

→ Customer ]

→ Parking Attendant

Invoice

→ Ticket

→ exit

→ amount

→ Payment

→ Exit etc



# → Entry Gate

## Class Diagram

### → Class

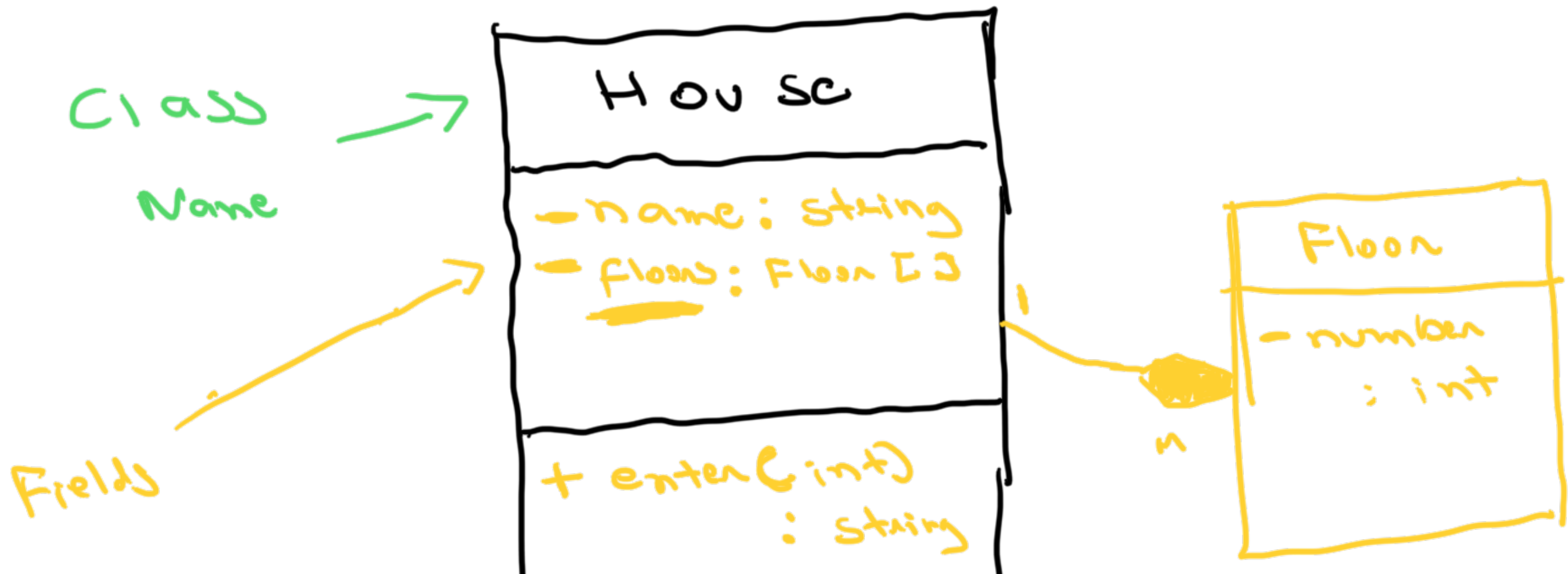
→ attributes

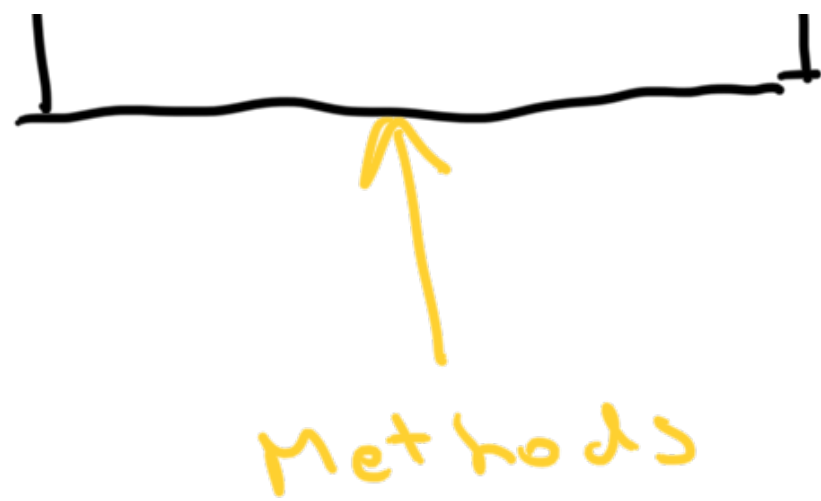
→ behaviour

### Relationships

→ Type of reln

→ Cardinality





① Inheritance - is A



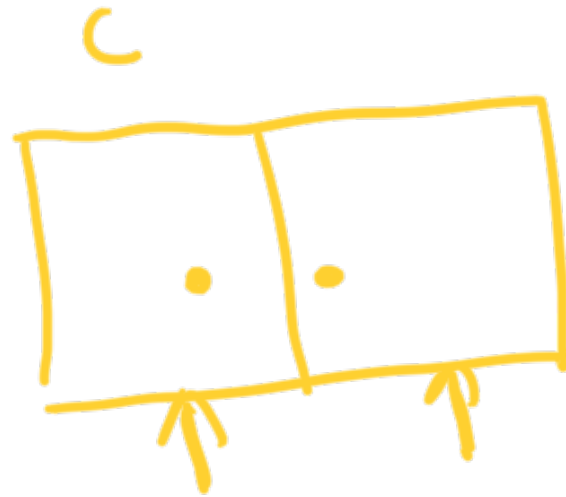
Man → Human



② Association - has A



① Composition



→ If parent is destroyed, child can also be destroyed.

---

② Aggregation

Husband ↗

wipe



Parent class → destroyed

Child → not destroyed

aggregation



Composition

Cardinality



- ① 1: m
- ② m: 1
- ③ 1: 1
- ④ m: m

Husband

Wife



Father



Master Class

M

Student

M

:

M : M

House

Room

1:M



Model



