**Lecture 2 - 管理类OOD: Parking Lot & Restaurant - Singleton Pattern**

* 管理类 - 题目后面都可以接上三个字: 管理员
  + e.g.: Gym, Parking lot, Restaurant, Library, Super market, Hotel
  + 设计一个模拟/代替管理员日常工作的系统
* 解题方法
  + Clarify: 除了题目中问的名词外，还需要从管理的名词来考
  + Core object: 有进有出
  + Cases: 从管理员角度考虑
    - Reserve: 预定
    - Serve: 服务
    - Checkout: 买单
  + Class: 经常可以使用**收据**的形式，来保管信息
  + Correctness

**Clarify**

* What
  + 关键词: Parking lot, Vehicle, Parking Spot
  + Parking lot: 考虑多层的Parking lot, 没有错层Graphical user interface

    Description automatically generated
  + Vehicle: 考虑三种大小的车
  + 不考虑残疾人停车位/充电车位
* How
  + 规则1: 如何停车
    - 一定要从**停车场**的角度来考虑,而不是车的角度
      * 停车场: 开进停车场 -> 返回一个能停的地方 ->停进一个位 置
      * 车: 开进停车场 -> 经过每一个位置看看能不能停 -> 停进一个位置
  + 规则2: 付费
    - 免费还是付费
* Who: Optional

**Core Object**

* Parking Lot
* Parking spot
* Car, Bus, Motorcycle
* 映射关系
  + Parking has a list of Spot
  + 错误方法: spot里面有car,或者parking lot有list of car

**Cases - 站在管理员的角度想**

* Bus / Car / Motorcycle
  + N/A
* Parking Lot
  + Get available count (reserve)
  + Park vehicle (serve)
  + Clear spot (checkout)
  + Calculate price (checkout)
* Parking Spot
  + N/A

**Class & Correctness**

* Draw UML per Use Case
* 从以下几方面检查:
  + Validate use cases (检查是否支持所有的use case)
  + Follow good practice (面试当中的加分项，展现一个程序员的经验)
  + S.O.L.I.D
  + Design pattern

**Design Pattern - Singleton**

public class ParkingLot {

private static ParkingLot \_instance = null;

private List<Level> levels;

private ParkingLost() {

levels = new ArrayList<List>();

}

public static synchronized ParkingLot getInstance() {

if (\_instance == null) {

\_instance = new ParkingLot();

}

return \_instance;

}

}

* synchronized getInstance()使得Singleton线程安全
* static \_instance 使得只有一个instance
* private ParkingLot()使得constructor只被调用一次