管理类面向对象设计

讲师：文泰来

加班主任，进班级答疑群
快速获取面试资料/课程福利

课程大纲

- Elevator System Follow-up
- 管理类OOD题型
- 管理类OOD解题思路
- Parking Lot

Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

- Solution 1: if - else

```
public void handleRequest(ExternalRequest r)
{
    if(time == TIME.PEAK)
    {
        // use peak hour handler
    }

    else if(time == TIME.NORMAL)
    {
        // use normal hour handler
    }
}
```

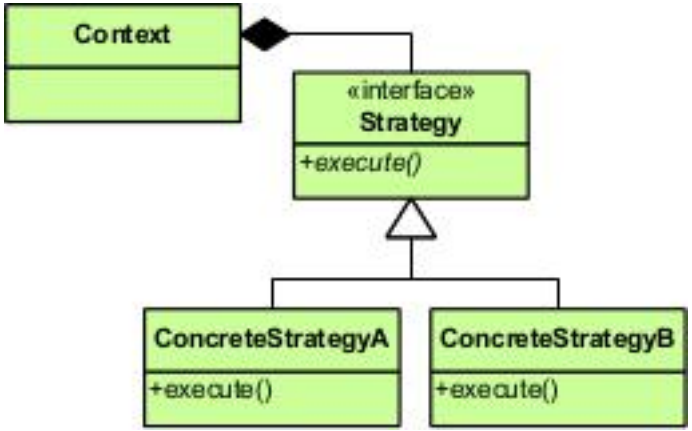
Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

Solution 2: Strategy design pattern

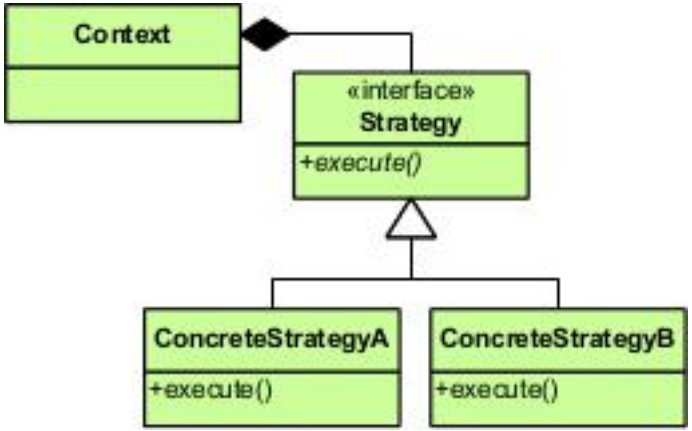
Challenge

- Strategy Pattern



Challenge

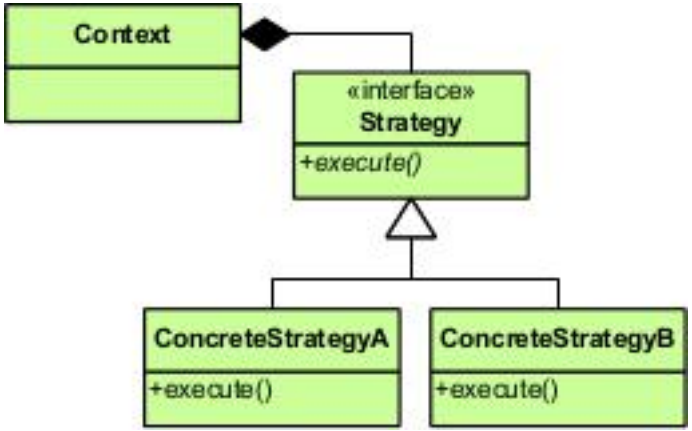
- Strategy Pattern



- **封装**了多种 算法/策略

Challenge

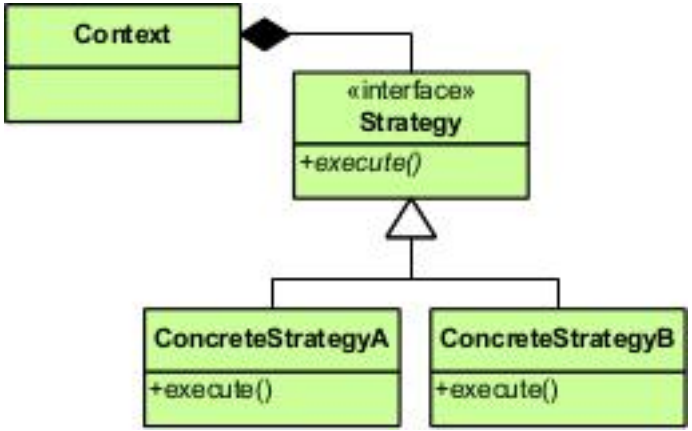
- Strategy Pattern



- **封装**了多种 算法/策略
- 使得算法/策略之间能够**互相替换**

Challenge

- Strategy Pattern



ElevatorSystem
- List<Elevator> elevators - HandleRequestStrategy strategy
+ void handleRequest(ExternalRequest r) + void setStrategy(HandleRequestStrategy s)

《interface》 HandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

PeakHourHandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

NormalHourHandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

Challenge

- Strategy design pattern

```
interface HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators);
}

class RandomHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {
        Random rand = new Random();

        int  n = rand.nextInt(elevators.size());

        elevators.get(n).handleExternalRequest(request);
    }
}

class AlwaysOneElevatorHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {

        elevators.get(0).handleExternalRequest(request);
    }
}
```

Challenge

- Strategy design pattern

```
class MyJavaApplication
{
    ElevatorSystem system = new ElevatorSystem();

    system.setStrategy(new RandomHandleRequestStrategy());

    ExternalRequest request = new ExternalRequest(Direction.UP, 3);

    system.handleRequest(request);
}

class ElevatorSystem
{
    private HandleRequestStrategy strategy = new HandleRequestStrategy();
    private List<Elevator> elevators = new ArrayList<>();

    public void setStrategy(HandleRequestStrategy strategy)
    {
        this.strategy = strategy;
    }

    public void handleRequest(ExternalRequest request)
    {
        strategy.handleRequest(request, elevators);
    }
}
```

```
interface HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators);
}

class RandomHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {
        Random rand = new Random();

        int  n = rand.nextInt(elevators.size());

        elevators.get(n).handleExternalRequest(request);
    }
}

class AlwaysOneElevatorHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {

        elevators.get(0).handleExternalRequest(request);
    }
}
```

管理类 – Management

管理类

什么是管理类面向对象设计？

管理类

- Gym
- Parking lot
- Restaurant
- Library
- Super market
- Hotel
- …

管理类

- Gym
- Parking lot
- Restaurant
- Library
- Super market
- Hotel
- …

题目后面都可以接上三个字：“管理员”

管理类

体育馆 管理员

停车场 管理员

餐厅 管理员

图书馆 管理员

超市 管理员

宾馆 管理员

...

管理类

设计一个模拟/代替管理员日常工作的系统

管理类题目特点

- 频率：高

管理类题目特点

- 频率：高
- 难度：中

管理类题目特点

- 频率：高
- 难度：中
- 题目：日常生活中常见的场景

- Clarify – What

除了题目中问的名词外，还需要从管理的名词来考虑

- Clarify – What

除了题目中问的名词外，还需要从管理的名词来考虑

例子：Design parking lot.

- Clarify – What

除了题目中问的名词外，还需要从管理的名词来考虑

例子：Design parking lot.

关键字1： Parking lot
关键字2： Vehicle

管理类解题方法

- Core object -> 有进有出

考虑这个管理系统中，Input和Output是什么

管理类解题方法

- Core object -> 有进有出

考虑这个管理系统中，Input和Output是什么

例子：Elevator System

管理类解题方法

- Core object -> 有进有出

考虑这个管理系统中，Input和Output是什么

例子：Elevator System

Input: Request
Output: Elevator

管理类解题方法

- Use case -> 从管理员角度考虑

管理类解题方法

- Use case -> 从管理员角度考虑

- Reserve

管理类解题方法

- Use case -> 从管理员角度考虑

- Reserve
- Serve

管理类解题方法

- Use case -> 从管理员角度考虑

- Reserve
- Serve
- Checkout

- Class

在设计类图的时候，经常可以使用收据的形式，来保管信息

- Class

在设计类图的时候，经常可以使用收据的形式，来保管信息

例子：图书馆

User

Book

- Class

在设计类图的时候，经常可以使用收据的形式，来保管信息

例子：图书馆

User

Receipt

Book

Parking lot

- Can you design a parking lot ?

Clarify

- What
- How

Clarify

- What

关键字：Parking lot

Clarify

- What

关键字：Parking lot

Parking lot 管理什么？

Clarify

- What

关键字：Parking lot

Parking lot 管理什么？

Clarify

- What

关键字：Parking lot

Parking lot 管理什么？

Clarify

- What

关键字：Parking lot

Parking lot 管理什么？
Vehicle/ Parking spots

Clarify

- What

关键字：Parking lot, Vehicle, Parking Spot

What

关键字：Parking lot

属性：？

What

关键字：Parking lot

www.alamy.com - G2GRB7

Challenge

- What are the differences between these parking lots?

Challenge

Parking lot -> Parking spaces

Parking lot -> Parking level -> Parking spaces

Parking lot -> Parking level(optional) -> Parking space -> Upper/Lower space

What

关键字： Vehicle

属性：？

What

关键字：Vehicle

What

关键字：Vehicle

- 设计的停车场是否需要纳不同大小的车辆？

What

关键字：Parking Spot

属性？

What

关键字：Parking Spot

- 设计的停车场是否要考虑残疾人停车位？

What

关键字：Parking Spot

- 设计的停车场是否要考虑充电车位？

What

针对本题：

- Parking lot: 考虑多层的Parking lot, 没有错层
- Vehicle: 考虑三种大小的车
- 不考虑残疾人停车位/充电车位

Challenge

如何设计停车场来支持停不同大小的车？

Challenge

如何设计停车场来支持停不同大小的车？

Challenge

- 当寻找合适的车位的时候，需要看边上的位置是否是空位

Challenge

- 当寻找合适的车位的时候，需要看边上的位置是否是空位

- 当有新的车形需要支持的时候，需要大量修改
- 利用率更低

- How

停车场有哪些规则？

- 规则1：如何停车?

VS.

How

- 针对本题：根据车的大小，横向停车

- 规则1：如何停车?

How

- 针对本题：停车场能够显示空闲位置的个数

How

- 规则2：收费

免费还是付费？

How

- 针对本题：根据时间收费

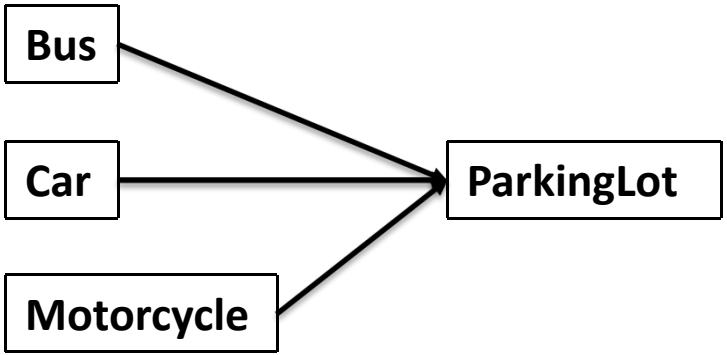
Core Object

- 什么是Core object？
- 为什么要定义Core Object？
- 如何定义Core Object？

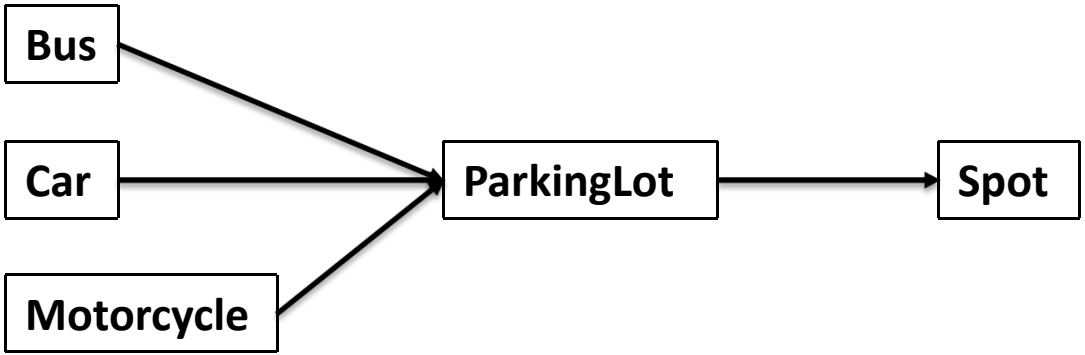
Core Object

ParkingLot

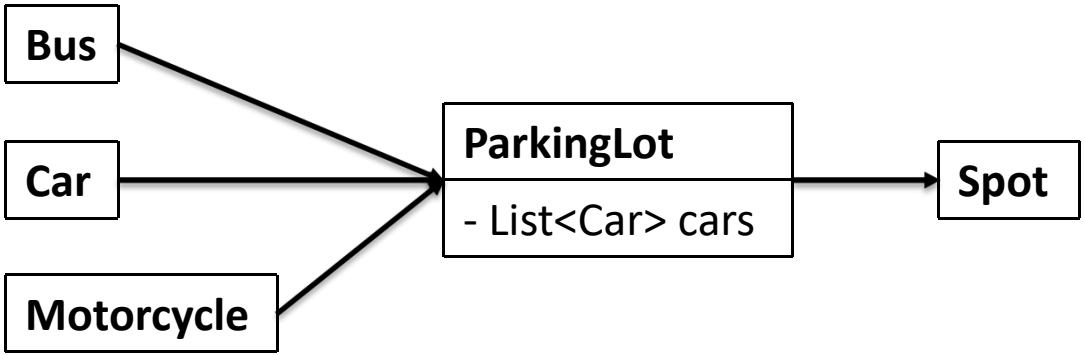
Core Object



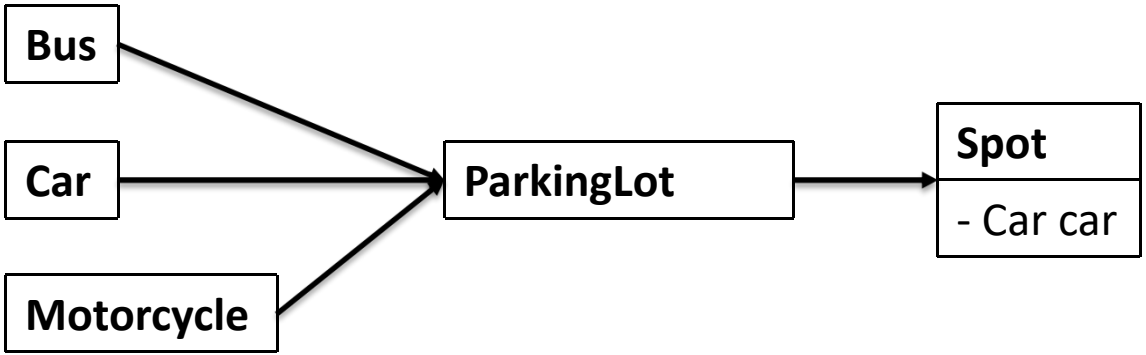
Core Object



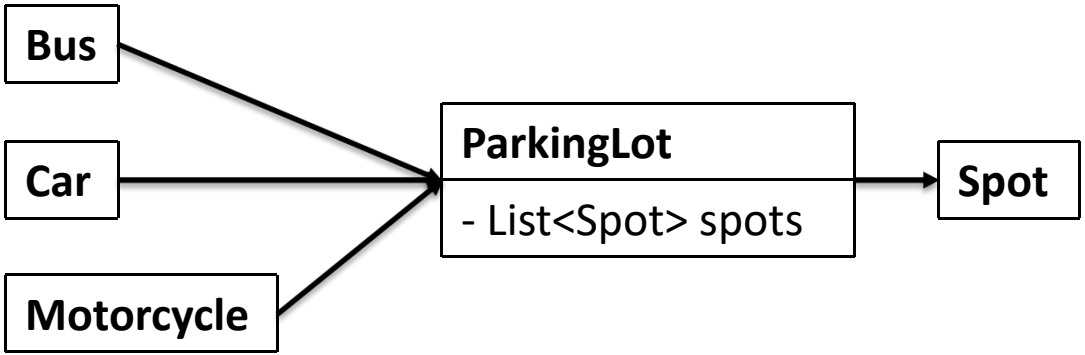
Core Object



Core Object



Core Object



Cases

- 什么是Use case？
- 为什么要写Use cases？
- 如何写Use cases？

Cases

- Bus / Car / Motorcycle

Cases

- Bus / Car / Motorcycle

站在管理员的角度想

Cases

- Bus / Car / Motorcycle

N/A

Cases

- ParkingLot

站在管理员角度考虑

Cases

- ParkingLot

- Get available count

Cases

- ParkingLot

- Get available count
- Park vehicle

Cases

- ParkingLot

- Get available count
- Park vehicle
- Clear spot

Cases

- ParkingLot

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Cases

- ParkingLot

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Management类常见Use case:

- Reservation : X

Cases

- ParkingLot

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Management类常见Use case:

- Reservation : X
- Serve: Park vehicle

Cases

- ParkingLot

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Management类常见Use case:

- Reservation : X
- Serve: Park vehicle
- Check out: Clear spot + Calculate price

Cases

- Spot

Cases

- Spot

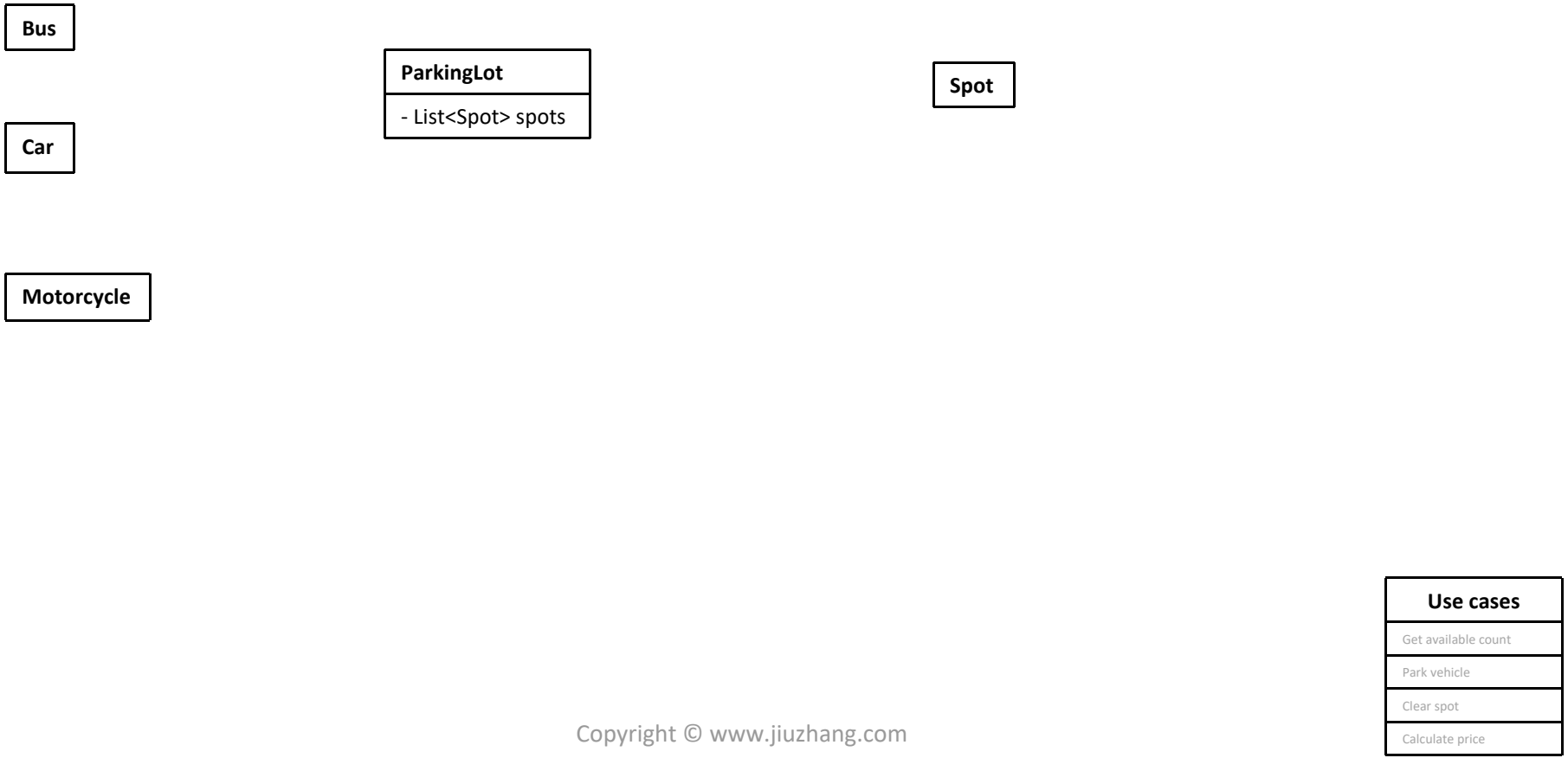
- N/A

Class

- 什么是类图？
- 为什么要画类图？
- 怎么画类图？

Class

Class



Class

- Use case: Get available counts

Parking lot shows how many **available spots** in total

Class

Bus

Car

Motorcycle

ParkingLot

- List<Spot> spots
- int availableCount

Spot

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

Bus

Car

Motorcycle

ParkingLot
- List<Spot> spots - int availableCount
+ int getAvailableCount()

Spot

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Challenge

- 如何分别显示出每一层的空位个数？

Challenge

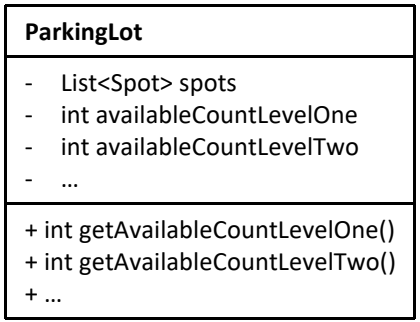
- 如何分别显示出每一层的空位个数？

Solution 1: 有几层就保存几个变量

Challenge

- 如何分别显示出每一层的空位个数？

Solution 1: 有几层就保存几个变量



Challenge

- 如何分别显示出每一层的空位个数？

Solution 2: 新建一个Level类

Class

Bus

Car

Motorcycle

ParkingLot

- List<Spot> spots
- int availableCount

+ int getAvailableCount()

Level

Spot

Use cases

Get available count

Park vehicle

Clear spot

Calculate price

Class

Bus

Car

Motorcycle

ParkingLot
- List<Spot> spots - int availableCount
+ int getAvailableCount()

Level
- List<Spot> spots

Spot

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

Bus

Car

Motorcycle

ParkingLot
- List<Level> levels - int availableCount
+ int getAvailableCount()

Level
- List<Spot> spots

Spot

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

Bus

Car

Motorcycle

ParkingLot

- List<Level> levels
- int availableCount

+ int getAvailableCount()

Level

- List<Spot> spots
- int availableCount

Spot

Use cases

Get available count

Park vehicle

Clear spot

Calculate price

Class

Bus

Car

Motorcycle

ParkingLot

- List<Level> levels
- int availableCount

+ int getAvailableCount()

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()

Spot

Use cases

Get available count

Park vehicle

Clear spot

Calculate price

Class

Bus

Car

Motorcycle

ParkingLot
- List<Level> levels
+ int getAvailableCount()

Level
- List<Spot> spots - int availableCount
+ int getAvailableCount()

Spot

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

- Use case: Park vehicle

Class

- Use case: Park vehicle

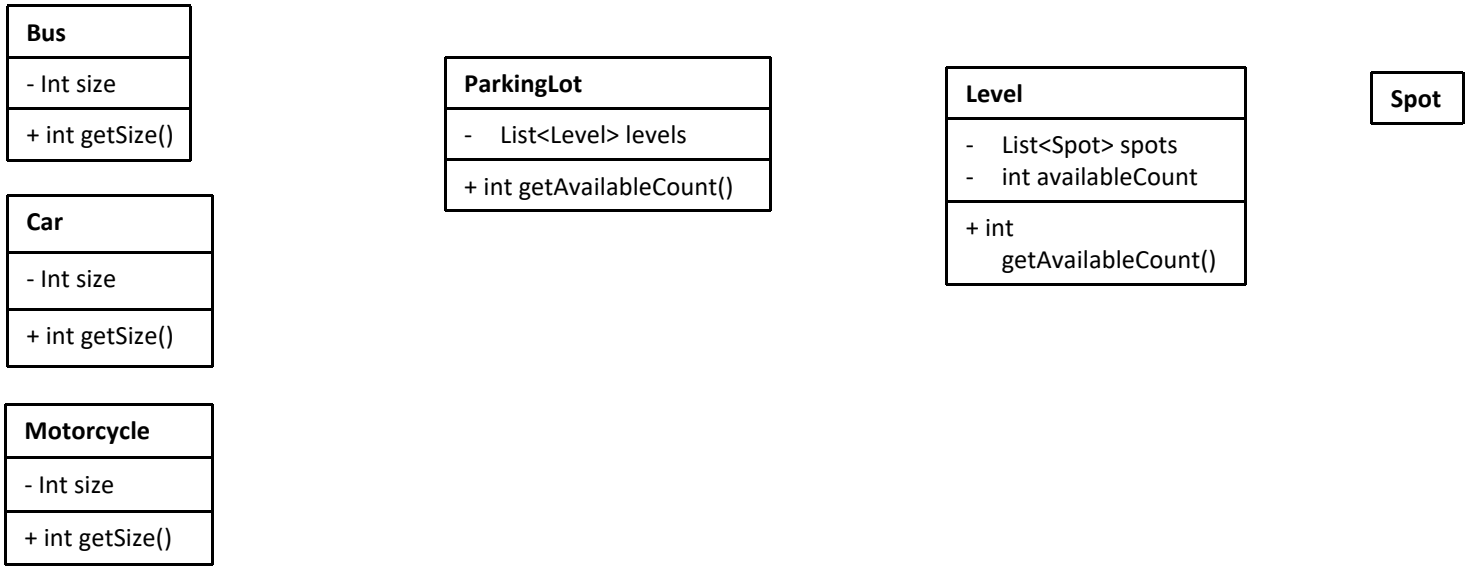
- Parking lot checks the size of vehicle
- Parking lot find an available spot for this vehicle
- Vehicle takes the spot(s)

Class

- Use case: Park vehicle

- Parking lot checks the size of vehicle

Class



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

- Use case: Park vehicle

- Parking lot checks the size of vehicle
- Parking lot find an available spot for this vehicle

Class

Bus
- Int size
+ int getSize()

Car
- Int size
+ int getSize()

Motorcycle
- Int size
+ int getSize()

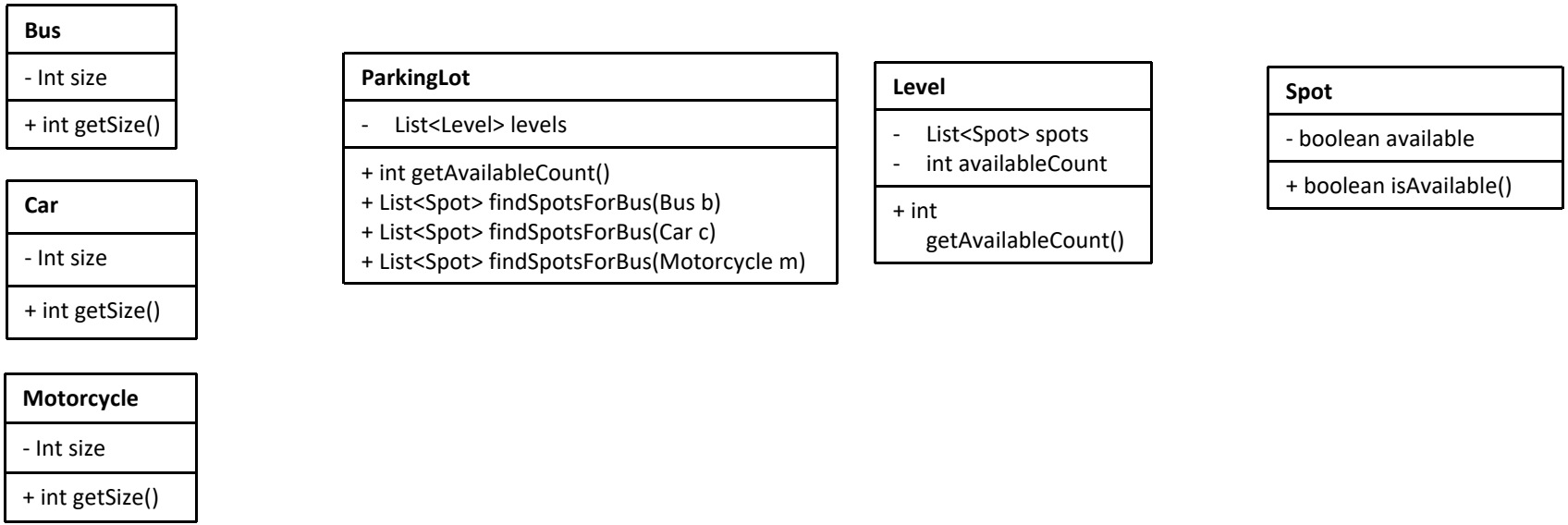
ParkingLot
- List<Level> levels
+ int getAvailableCount()

Level
- List<Spot> spots - int availableCount
+ int getAvailableCount()

Spot
- boolean available
+ boolean isAvailable()

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

- Use case: Park vehicle

- Parking lot checks the size of vehicle
- Parking lot find an available spot for this vehicle
- Vehicle takes the spot(s)

Class

Bus
- Int size
+ int getSize()

Car
- Int size
+ int getSize()

Motorcycle
- Int size
+ int getSize()

ParkingLot
- List<Level> levels
+ int getAvailableCount() + List<Spot> findSpotsForBus(Bus b) + List<Spot> findSpotsForBus(Car c) + List<Spot> findSpotsForBus(Motorcycle m)

Level
- List<Spot> spots - int availableCount
+ int getAvailableCount()

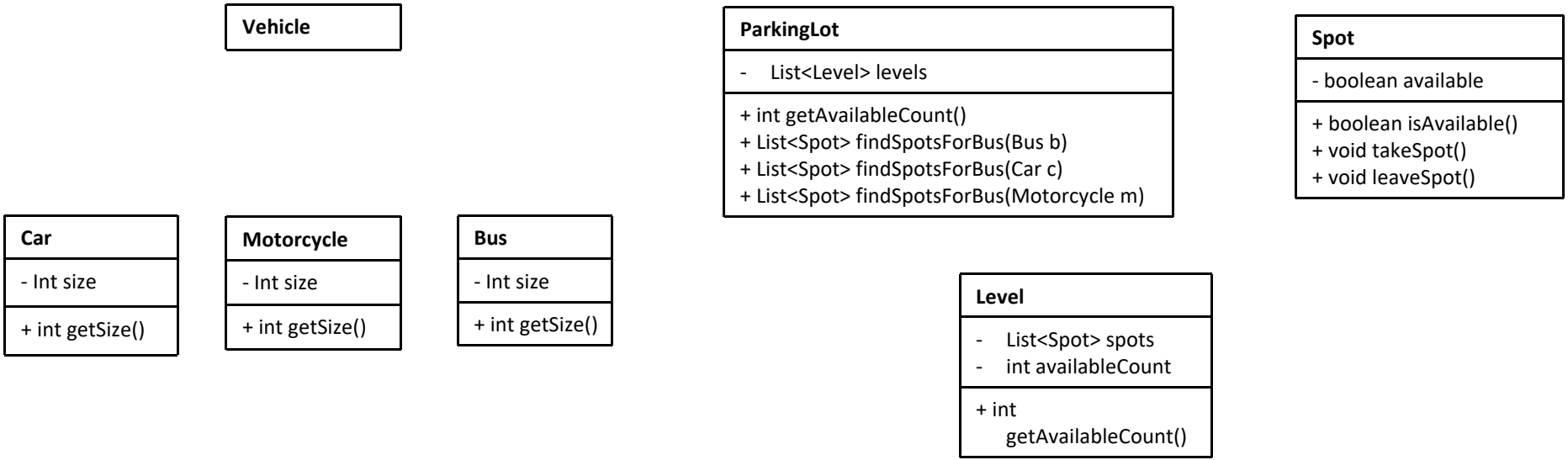
Spot
- boolean available
+ boolean isAvailable() + void takeSpot() + void leaveSpot()

Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Best practice

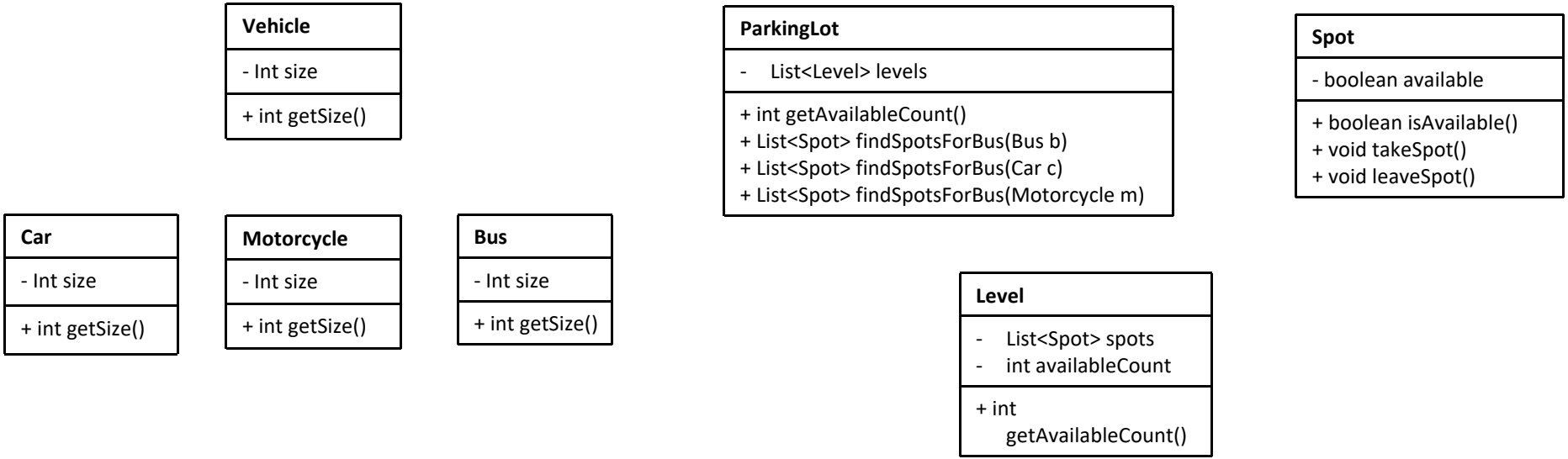
- Open for extension, close for modification

Best practice



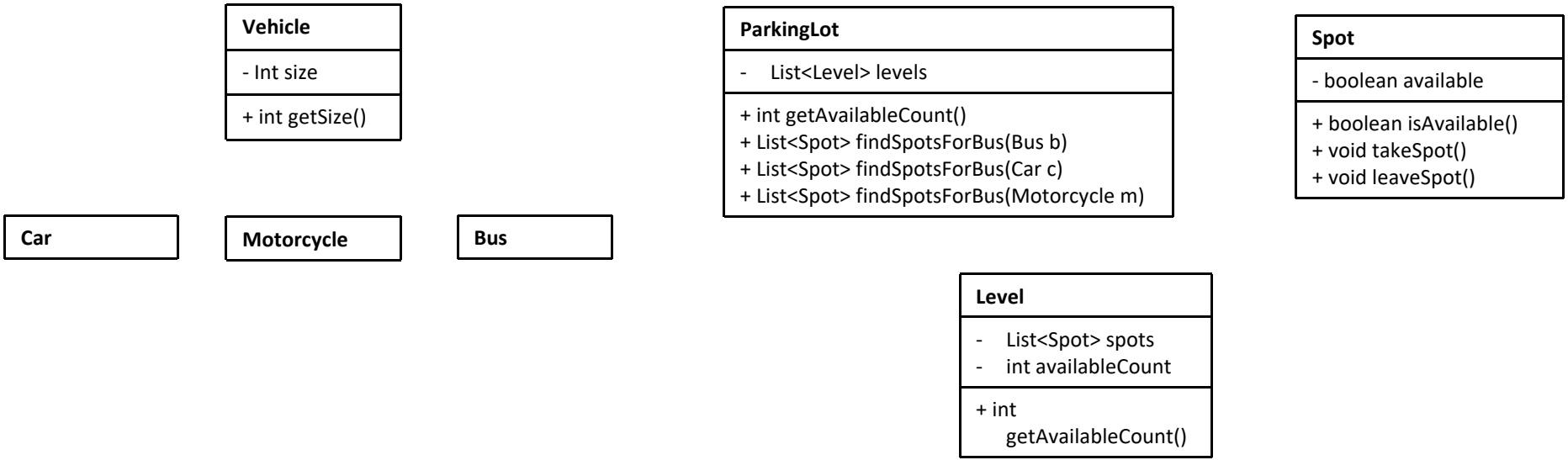
Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Best practice



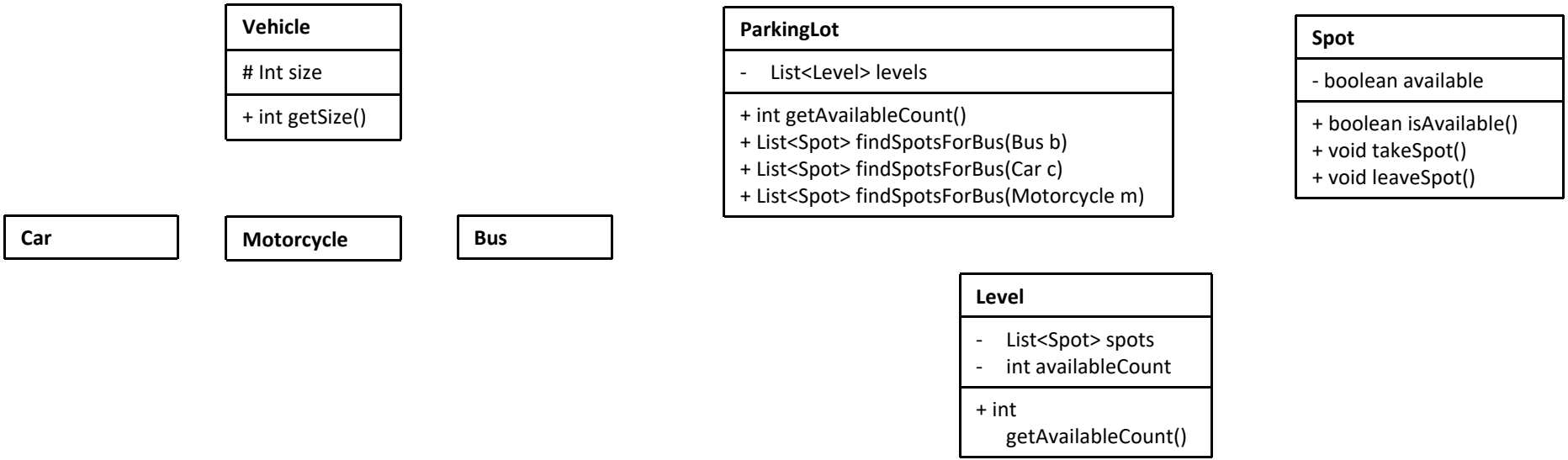
Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Best practice



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Best practice

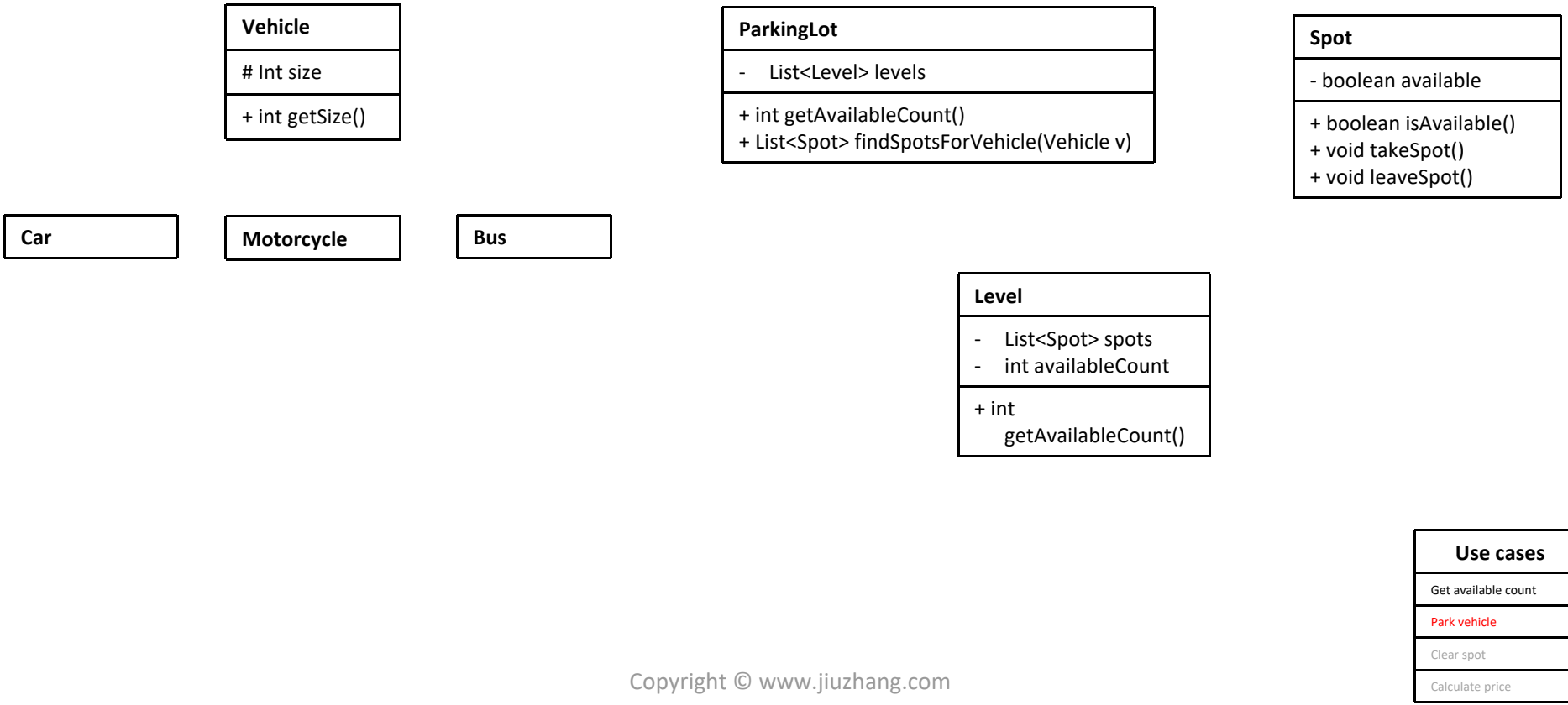


Best practice

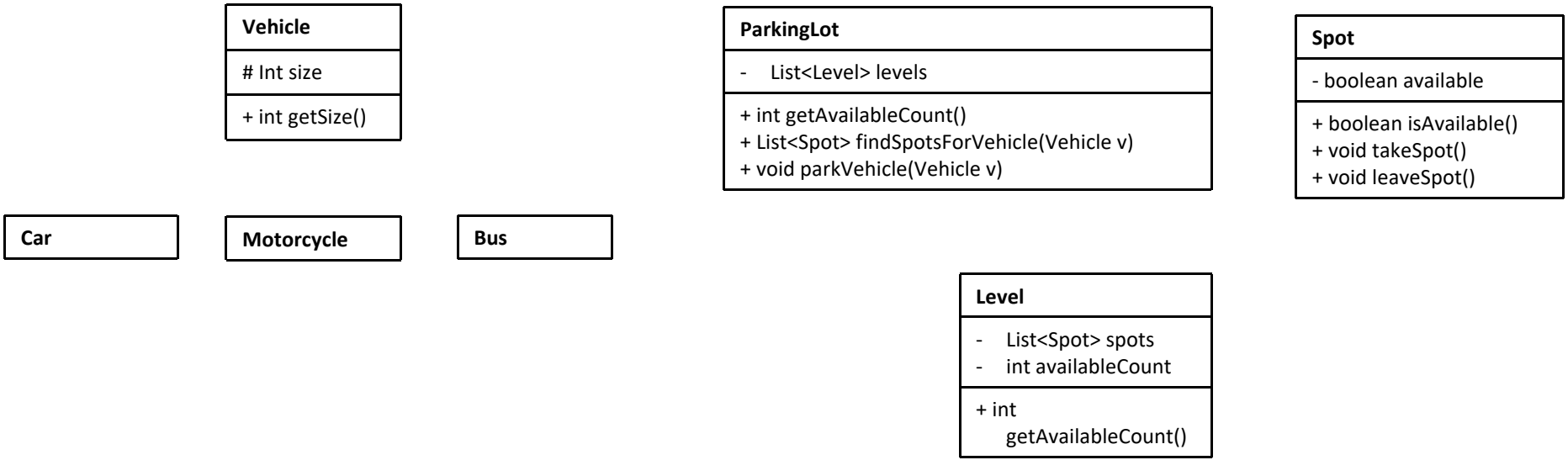
```
public class Vehicle
{
    private int size;

    public int getSize()
    {
        return size;
    }
}

public class Bus extends Vehicle
{
    private int size;

    public int getSize()
    {
        return size;
    }
}
```

```
public class Vehicle
{
    protected int size;

    public int getSize()
    {
        return size;
    }
}

public class Bus extends Vehicle
{
    public Bus()
    {
        size = 3;
    }
}
```

Best practice

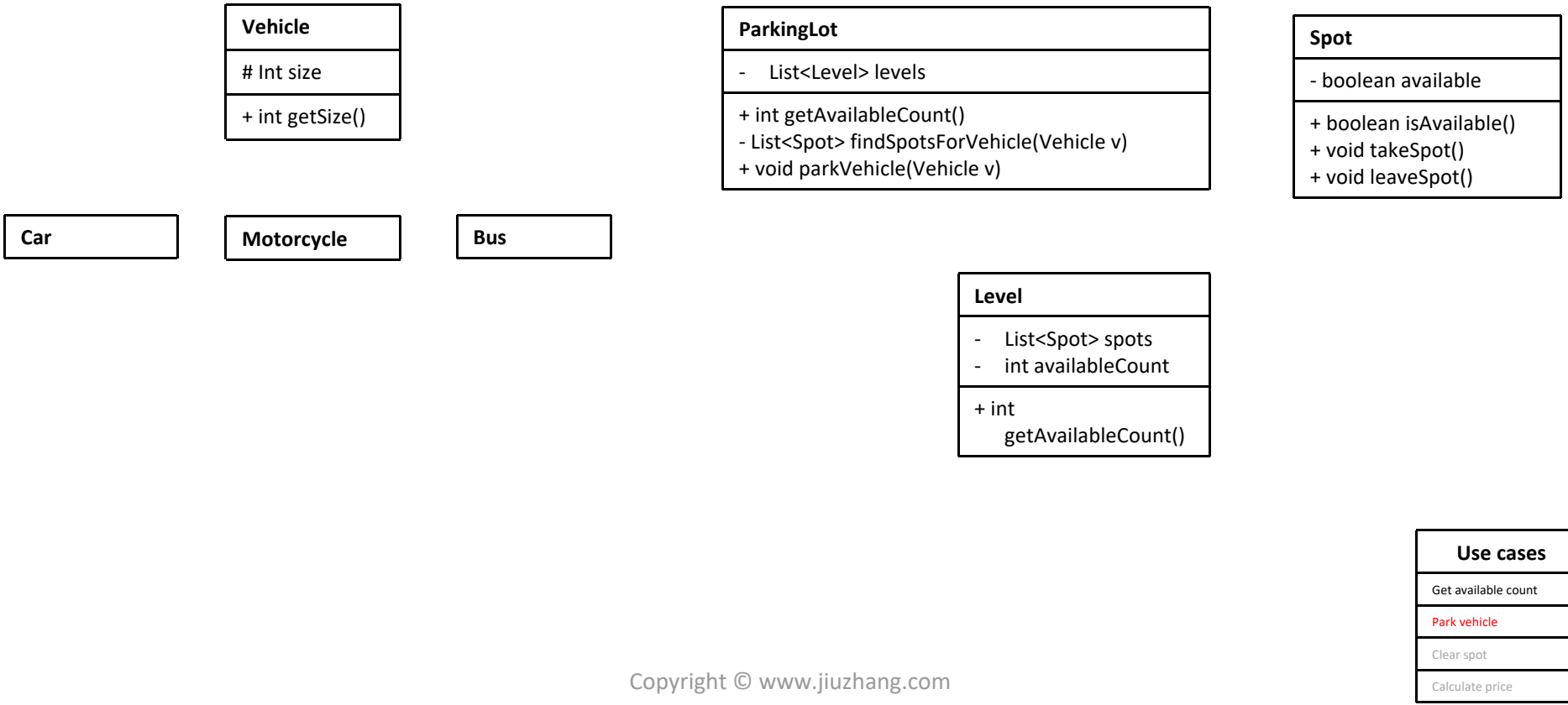


Class



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class



Class

- Use case: Clear spot

- Parking lot find the spot to clear
- Update spot to be available

Class

- Use case: Clear spot

- Parking lot find the spot to clear
- Update spot to be available
- Update available count for each level

Class

- Use case: Clear spot

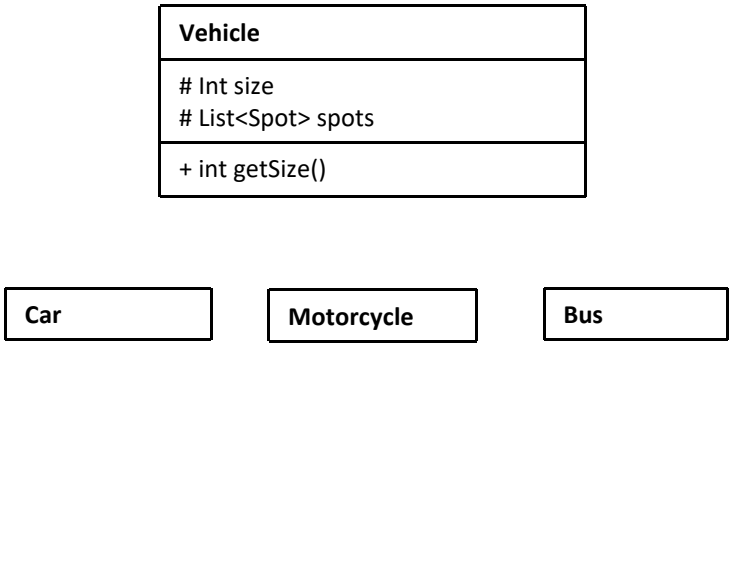
- Parking lot find the spot to clear

Challenge

- 如何找到需要被free的spot？

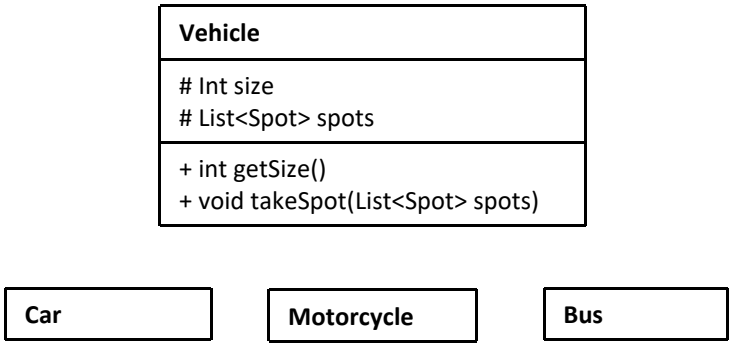
Solution 1: Vehicle保存停的车位

Class



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class



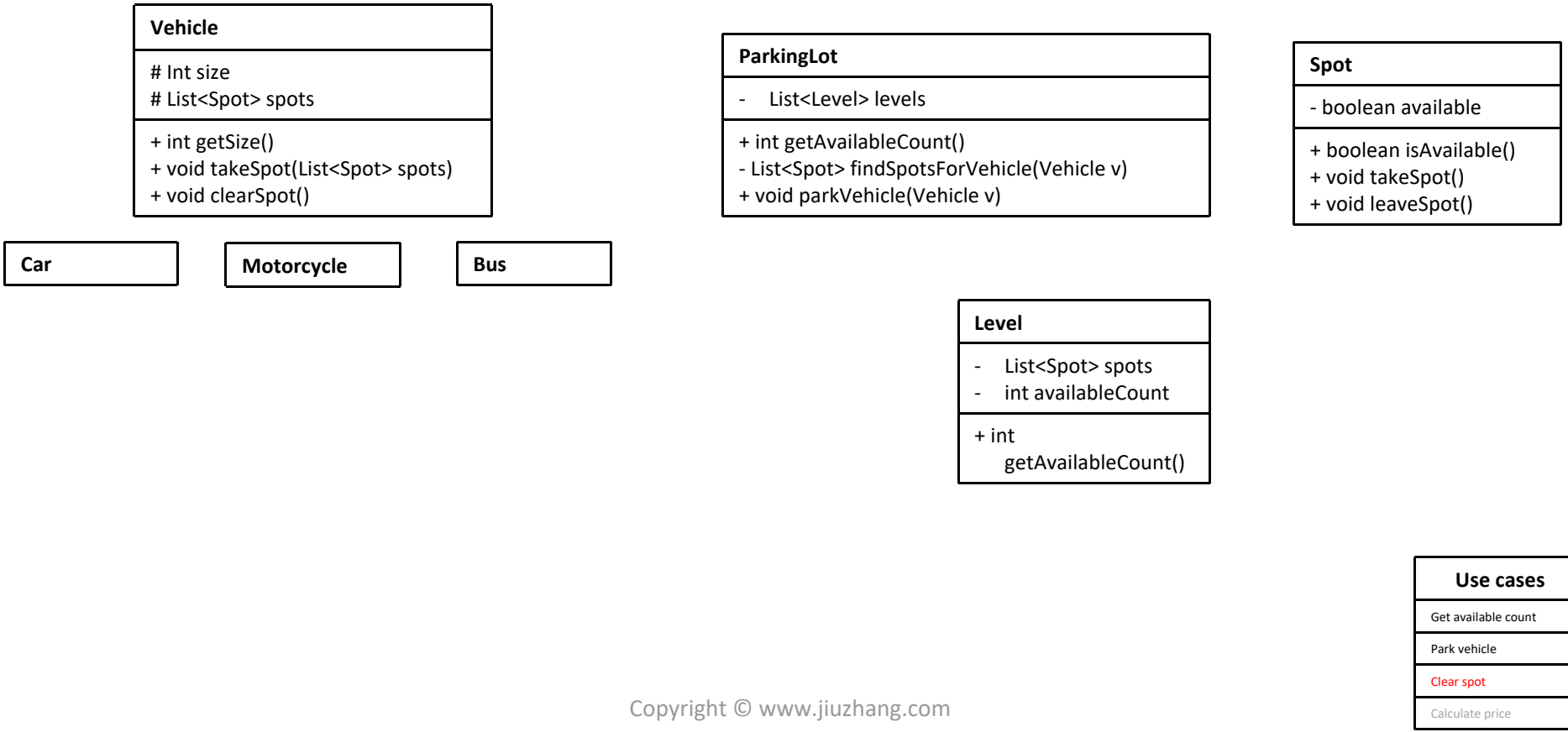
Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

- Use case: Clear spot

- Parking lot find the spot to clear
- Update spot to be available

Class

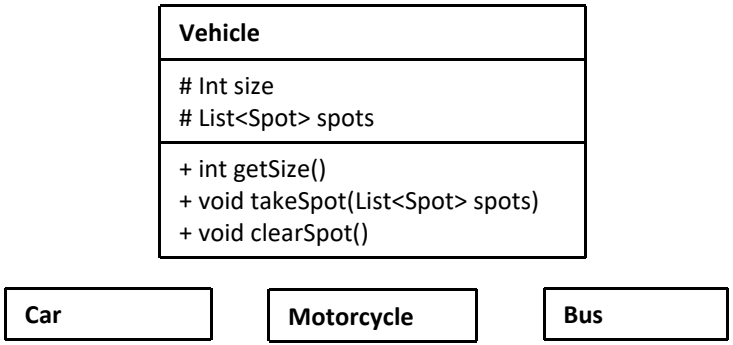


Class

- Use case: Clear spot

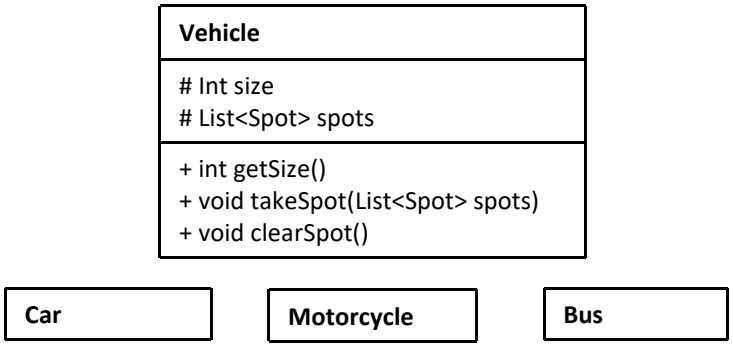
- Parking lot find the spot to clear
- Update spot to be available
- Update available count for each level

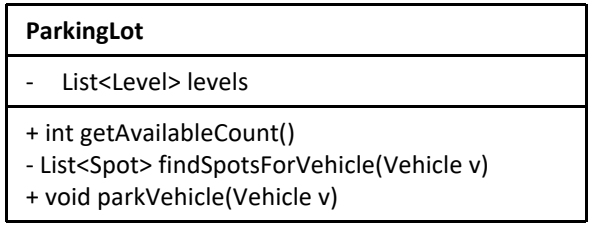
Class

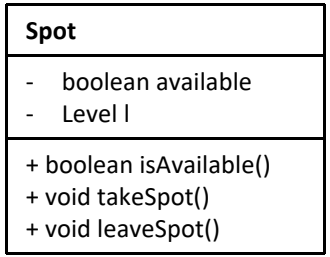


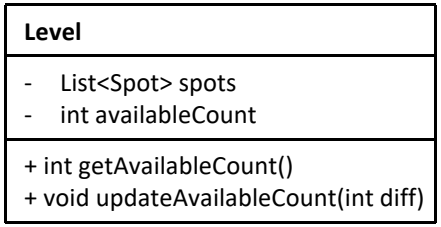
Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class









Best practice

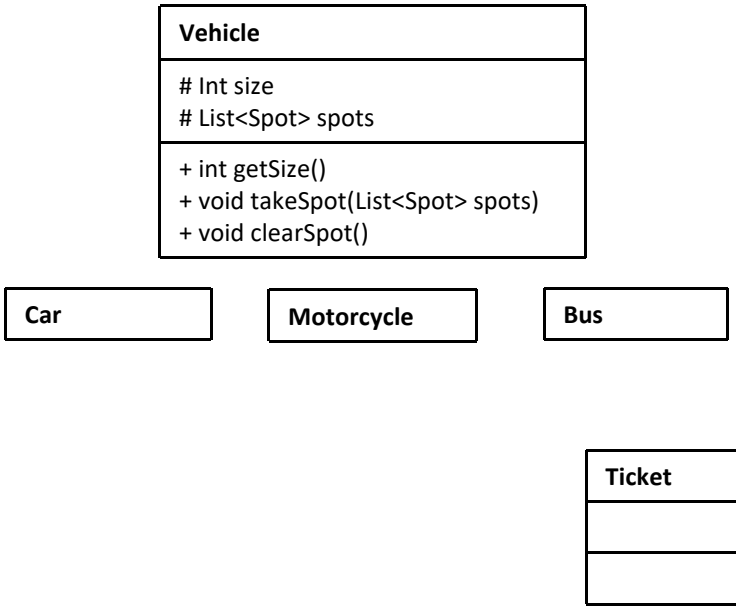
- 不是说好从Parking lot出发吗？

Vehicle
Int size # List<Spot> spots
+ int getSize() + void takeSpot(List<Spot> spots) + void clearSpot()

Best practice

- Solution: **Receipt**

Class



Use cases
Get available count
Park vehicle
Clear spot
Calculate price

Class

Vehicle

Int size
List<Spot> spots

+ int getSize()
+ void takeSpot(List<Spot> spots)
+ void clearSpot()

ParkingLot

- List<Level> levels

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ void parkVehicle(Vehicle v)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Car

Motorcycle

Bus

Ticket

- Vehicle v

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Use cases

Get available count

Park vehicle

Clear spot

Calculate price

Class

Vehicle

Int size
List<Spot> spots

+ int getSize()
+ void takeSpot(List<Spot> spots)
+ void clearSpot()

ParkingLot

- List<Level> levels

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ void parkVehicle(Vehicle v)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Car

Motorcycle

Bus

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Ticket

- Vehicle v
- List<Spot> spots

Use cases

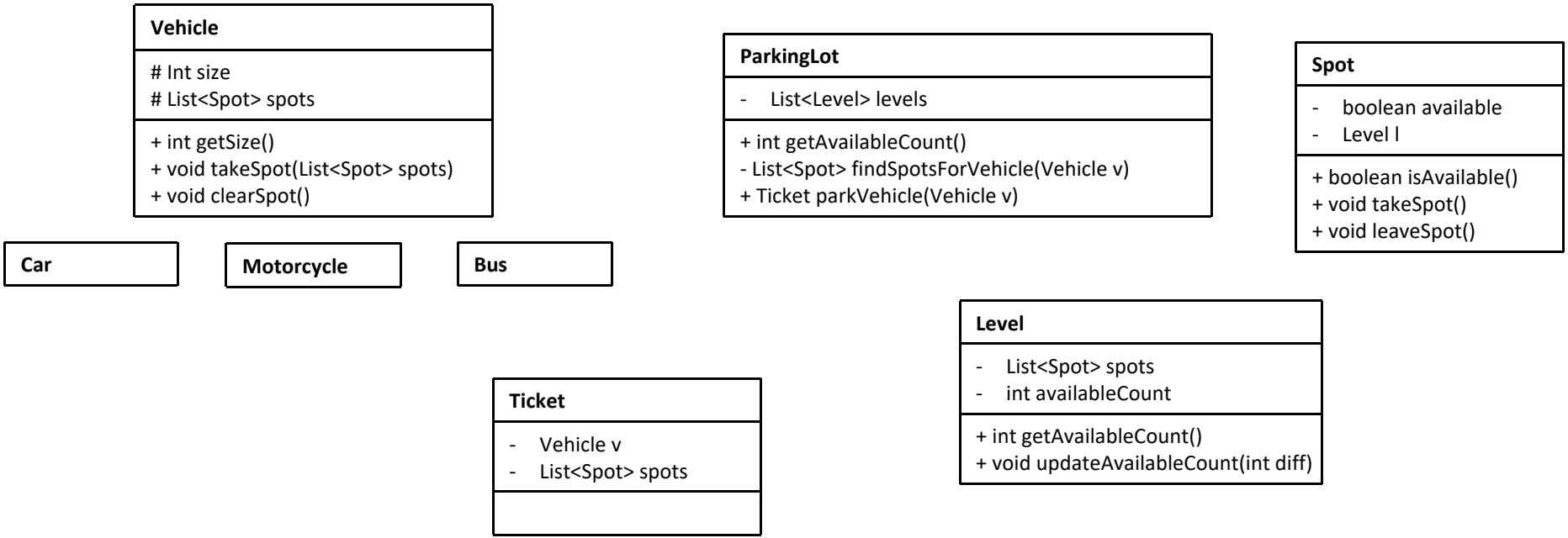
Get available count

Park vehicle

Clear spot

Calculate price

Class



Class

Vehicle

Int size
List<Spot> spots

+ int getSize()
+ void takeSpot(List<Spot> spots)
+ void clearSpot()

Car

Motorcycle

Bus

ParkingLot

- List<Level> levels

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Ticket

- Vehicle v
- List<Spot> spots

Use cases

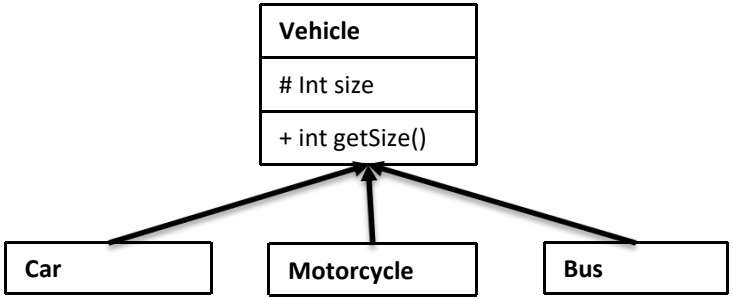
Get available count

Park vehicle

Clear spot

Calculate price

Class



ParkingLot

- List<Level> levels

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Ticket

- Vehicle v
- List<Spot> spots

Use cases

Get available count

Park vehicle

Clear spot

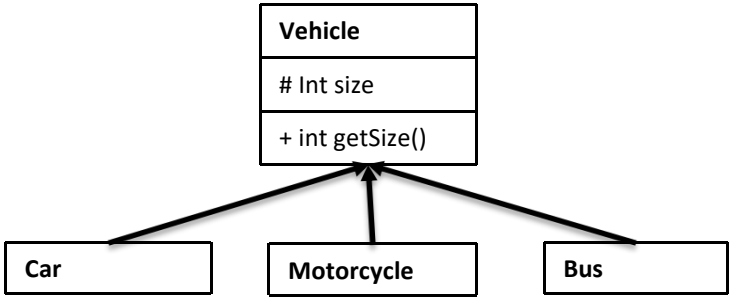
Calculate price

Class

- Use case: Calculate price

- When clear spot, parking lot calculates the expected price to pay

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

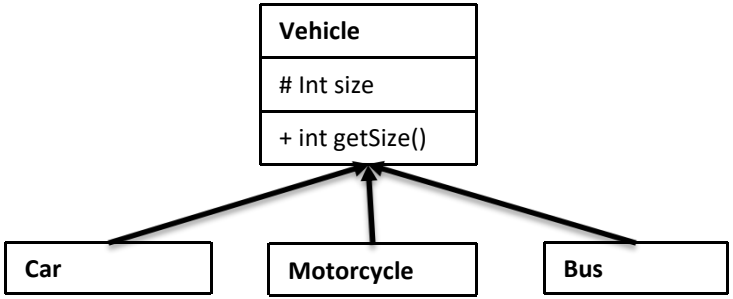
Ticket

- Vehicle v
- List<Spot> spots

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

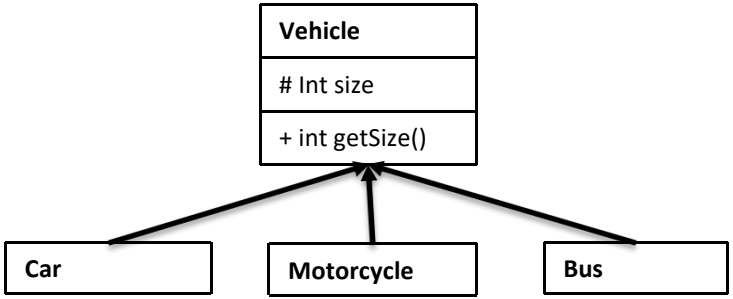
Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Use cases

Get available count

Park vehicle

Clear spot

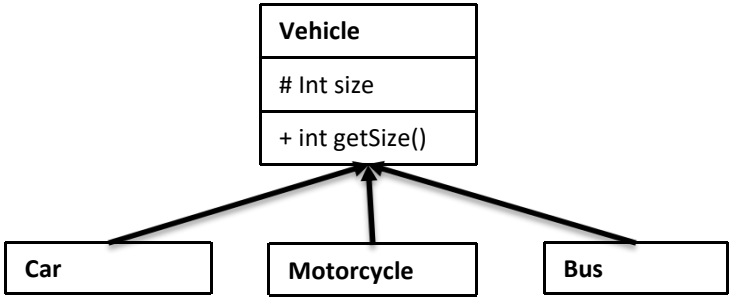
Calculate price

Correctness

- 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

Exceptions



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

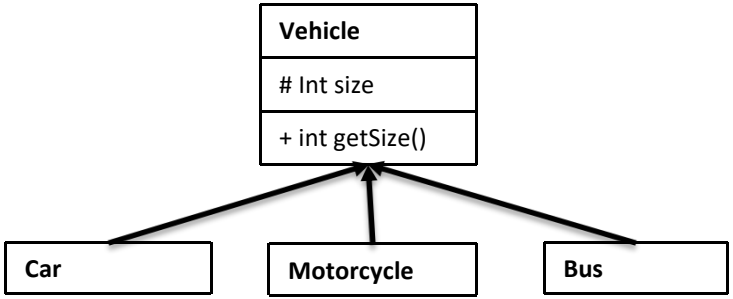
ParkingLotFullException

InvaidTicketException

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Class – Final view



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

ParkingLotFullException

InvaidTicketException

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Challenge

- Parking lot里每层的spots，是怎么排列的？当停Bus时，是否有问题？

Challenge

VS.

Challenge

- Solution 1:

- 在Level里加一个变量，作为每行固定的停车位个数
- 在Spot里加一个变量，作为Spot Id
- 这样能够知道哪些Spot在一行 / 一行有没有足够的Spots

Challenge

- Solution 2:

- 像添加Level一样，添加一个Row作为新的Class

Challenge

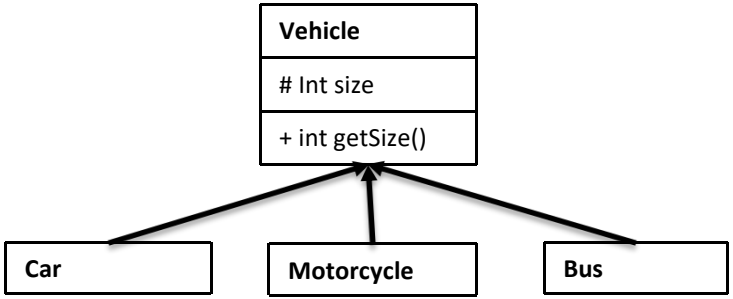
- Solution 1 VS. Solution 2 ?

Challenge

- Solution 1 VS. Solution 2 ?

如果用Solution 1, 每行的个数必须要一样

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Row

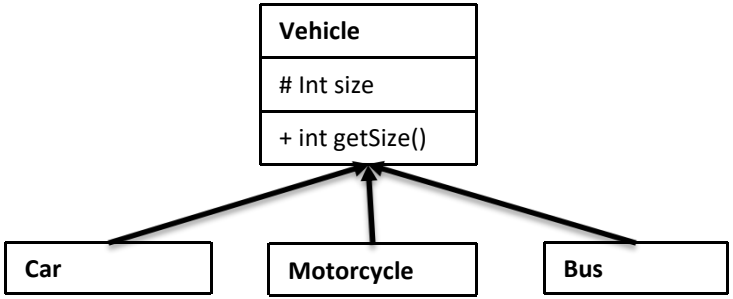
ParkingLotFullException

InvaidTicketException

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Spot> spots
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Row

- List<Spot> spots

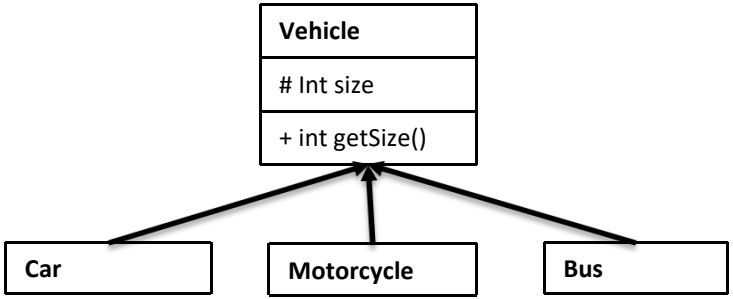
ParkingLotFullException

InvaidTicketException

Use cases

Get available count
Park vehicle
Clear spot
Calculate price

Class



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Row> rows
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Row

- List<Spot> spots

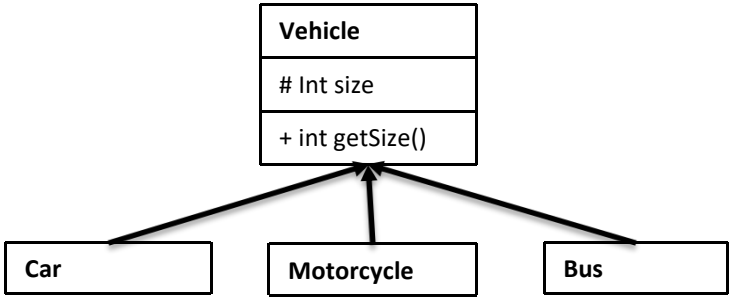
ParkingLotFullException

InvaidTicketException

Use cases

Get available count
Park vehicle
Clear spot
Calculate price

Class – Final view



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Row> rows
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

Row

- List<Spot> spots

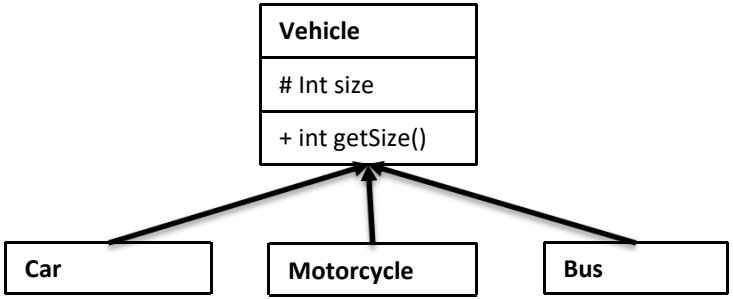
ParkingLotFullException

InvaidTicketException

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Exception



ParkingLot

- List<Level> levels
- float hourlyRate

+ int getAvailableCount()
- List<Spot> findSpotsForVehicle(Vehicle v)
+ Ticket parkVehicle(Vehicle v)
+ void clearSpot(Ticket t)
- float calculatePrice(Ticket t)

Spot

- boolean available
- Level l

+ boolean isAvailable()
+ void takeSpot()
+ void leaveSpot()

Row

- List<Spot> spots

Ticket

- Vehicle v
- List<Spot> spots
- Time startTime

Level

- List<Row> rows
- int availableCount

+ int getAvailableCount()
+ void updateAvailableCount(int diff)

ParkingLotFullException

InvaidTicketException

Use cases

- Get available count
- Park vehicle
- Clear spot
- Calculate price

Design pattern

- Clean and elegant

Design pattern

- Clean and elegant
- Keep code extendable

Design pattern

- Clean and elegant
- Keep code extendable

Design pattern

- Clean and elegant
- Keep code extendable
- Safe

Design pattern

- Clean and elegant
- Keep code extendable
- Safe
- Show off your skills ！

Design pattern

- Singleton

ensure a class has only one instance, and provide a global point of access to it

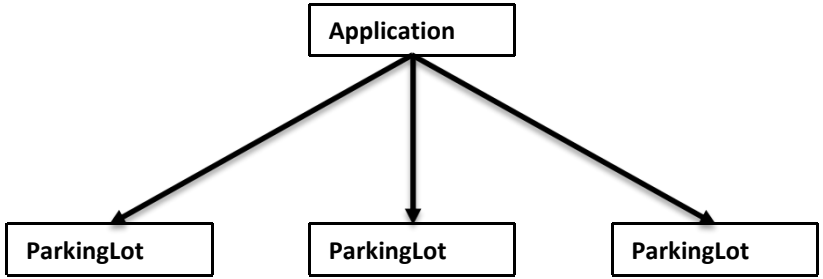
Design pattern

- Singleton

```
public class ParkingLot
{
    private List<Level> levels;

    public ParkingLot()
    {
        levels = new ArrayList<Level>();
    }
}
```

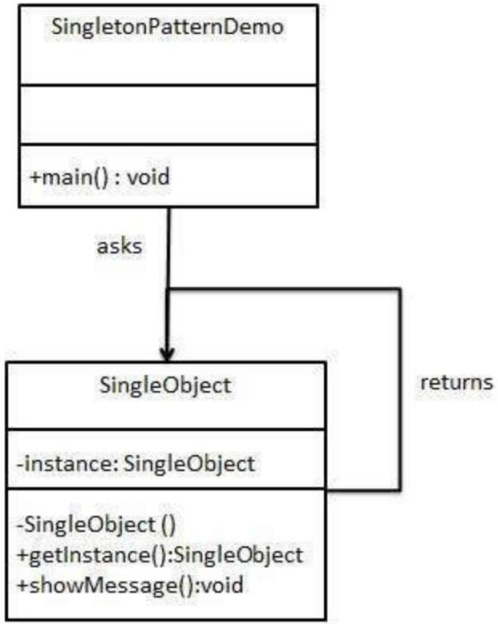
Design pattern

- Singleton



Design pattern

- Singleton



- Singleton – 基本式

```
public class ParkingLot
{
    private static ParkingLot _instance = null;

    private List<Level> levels;

    private ParkingLot()
    {
        levels = new ArrayList<Level>();
    }

    public static ParkingLot getInstance()
    {
        if(_instance == null)
        {
            _instance = new ParkingLot();
        }
        return _instance;
    }
}
```

Design pattern

- Singleton – 线程安全式

```
public class ParkingLot
{
    private static ParkingLot _instance = null;

    private List<Level> levels;

    private ParkingLot()
    {
        levels = new ArrayList<Level>();
    }

    public static synchronized ParkingLot getInstance()
    {
        if(_instance == null)
        {
            _instance = new ParkingLot();
        }
        return _instance;
    }
}
```

- Singleton – 静态内部类式

```
public class ParkingLot
{
    private ParkingLot(){}

    private static class LazyParkingLot
    {
        static final ParkingLot _instance = new ParkingLot();
    }

    public static ParkingLot getInstance()
    {
        return LazyParkingLot._instance;
    }
}
```

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
知乎专栏： http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com

Class

ExternalRequest

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons

InvalidExternalRequestException

ElevatorButton

Use cases

Handle request

Take external request

Take internal request

Open gate

Close gate

Check weight

Press button

Class

- Use case: Take external request

An **elevator** takes an external **request**, inserts in its stop list.

Class

ExternalRequest
- Direction d - int level

ElevatorSystem
- List<Elevator> elevators
+ void handleRequest(ExternalRequest r)

Elevator
- List<ElevatorButton> buttons

InvalidExternalRequestException

ElevatorButton

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

ElevatorButton

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

九章算法

ExternalRequest
- Direction d - int level

ElevatorSystem
- List<Elevator> elevators
+ void handleRequest(ExternalRequest r)

Elevator
- List<ElevatorButton> buttons
+ void handleExternalRequest(ExternalRequest r)

InvalidExternalRequestException

<<enumeration>> Direction
Up Down

ElevatorButton

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> stops

+ void handleExternalRequest(ExternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

ElevatorButton

Use cases

Handle request

Take external request

Take internal request

Open gate

Close gate

Check weight

Press button

Challenge

- 如果电梯目前在1L，有人按下了5L向上，之后又有人按下了3L向上，电梯会怎样行动？

stops will be {5,3}

Expected is: {3,5}

Challenge

- 如果电梯目前在1L，有人按下了5L向上，之后又有人按下了3L向上，电梯会怎样行动？

stops will be {5,3}

Expected is: {3,5}

Solution1: sort stops every time we add to it.

Challenge

- 如果电梯目前在1L，有人按下了5L向上，之后又有人按下了3L向上，电梯会怎样行动？

stops will be {5,3}

Expected is: {3,5}

Solution2: use priority queue instead of list

Challenge

- 如果电梯目前在1L，有人按下了5L向上，之后又有人按下了3L向上，紧接着这台电梯又被分配了一个2L向下的request。这台电梯会如何行动？

stops will be {2，3，5}

Expected is: {3，5，2}

Challenge

- 如果电梯目前在1L，有人按下了5L向上，之后又有人按下了3L向上，紧接着这台电梯又被分配了一个2L向下的request。这台电梯会如何行动？

stops will be {2，3，5}

Expected is: {3，5，2}

Solution: keep 2 lists for different direction

Class

ExternalRequest

- Direction d
- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops

+ void handleExternalRequest(ExternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

ElevatorButton

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

- Use case: Take internal request

An **elevator** takes an internal **request**, determine if it's valid, inserts in its stop list.

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops

+ void handleExternalRequest(ExternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

ElevatorButton

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

Use cases

- Handle request
- Take external request
- Take internal request
- Open gate
- Close gate
- Check weight
- Press button

Challenge

- 如何判断一个Internal request 是否为Valid?

Challenge

- 如何判断一个Internal request 是否为Valid?

Solution:

If elevator going up

    requested level lower than current level

        invalid

If elevator going down

    requested level higher than current level

        invalid

Challenge

- 如何判断一个Internal request 是否为Valid?

Solution:

If elevator **going up**

requested level lower than **current level**

invalid

If elevator **going down**

requested level higher than **current level**

invalid

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>> Direction

Up
Down

<<enumeration>> Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

- Use case: Open gate

Class

- Use case: Open gate

并行 VS 串行
单线程 VS 多线程

- Use case: Open gate

单线程：

{3，5，2} -> {5, 2} -> {2} -> {}

(1, Up) -> Open gate -> (3, Up) -> Close gate -> (3, Up) -> Open Gate -> (5, Up) -> Close gate -> (5, Down) -> Open gate -> (2, Down) -> Close Gate -> (2, Idle)

Class

- Use case: Open gate

多线程：

{3，5，2} -> {5, 2} -> {2} -> {} Critical Data

```
public class Elevator implements Runnable
{
    @Override
    public void run()
    {
        while(true)
        {
            if(thereIsSomethingLeftInStop())
            {
                operating();
            }
            else
            {
                Thread.sleep();
            }
        }
    }
}
```

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>> Direction

Up
Down

<<enumeration>> Status

Up
Down
Idle

Use cases

- Handle request
- Take external request
- Take internal request
- Open gate
- Close gate
- Check weight
- Press button

- Use case: Close gate

An **elevator**
checks if overweight;
close the door;
then check stops corresponds to current status;
if no stops left, check the reserve direction stops;
change status to reserve direction or idle.

Class

- Use case: check weight

An **elevator** checks its **current weight** and compare with **limit** to see if overweight

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases

- Handle request
- Take external request
- Take internal request
- Open gate
- Close gate
- Check weight
- Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

<<enumeration>> Direction

Up
Down

<<enumeration>> Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

- Use case: press button

A **button** inside elevator is pressed, will generate an **internal request** and send to the **elevator**.

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorButton

- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

ElevatorButton

- int level

+ boolean pressButton()

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

ElevatorButton

- int level
- Elevator elevator

+ InternalRequest pressButton()

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Class – Final view

ExternalRequest

- Direction d
- int level

InternalRequest

- int level

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

ElevatorButton

- int level
- Elevator elevator

+ InternalRequest pressButton()

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>>
Direction

Up
Down

<<enumeration>>
Status

Up
Down
Idle

Use cases
Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Correctness

- 从以下几方面检查：

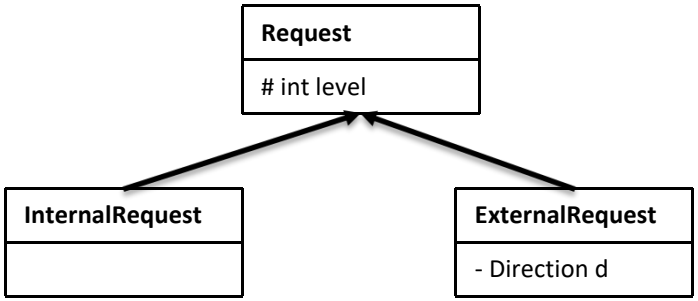
- Validate use cases (检查是否支持所有的use case)
- Follow good practice (面试当中的加分项，展现一个程序员的经验)
- S.O.L.I.D
- Design pattern

Good Practice

- 继承

检查你的设计中，是否有重复的类，可以采用继承的方式来表现

Good Practice



ElevatorButton

- int level
- Elevator elevator

+ InternalRequest pressButton()

Elevator

- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

+ void handleExternalRequest(ExternalRequest r)
+ void handleInternalRequest(InternalRequest r)
+ void openGate()
+ void closeGate()
- float getCurrentWeight()
- boolean isRequestValid(InternalRequest r)

ElevatorSystem

- List<Elevator> elevators

+ void handleRequest(ExternalRequest r)

InvalidExternalRequestException

OverWeightException

<<enumeration>> Direction

Up
Down

<<enumeration>> Status

Up
Down
Idle

Use cases

Handle request
Take external request
Take internal request
Open gate
Close gate
Check weight
Press button

Challenge

- How do you handle an external request?
- What if I want to apply different ways to handle external requests during different time of a day?
- Can you implement it in code?

Challenge

- How do you handle an external request?

如我们最早和面试官讨论的结果：

同方向 > 静止 > 反向

Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

- Solution 1: if - else

```
public void handleRequest(ExternalRequest r)
{
    if(time == TIME.PEAK)
    {
        // use peak hour handler
    }

    else if(time == TIME.NORMAL)
    {
        // use normal hour handler
    }
}
```

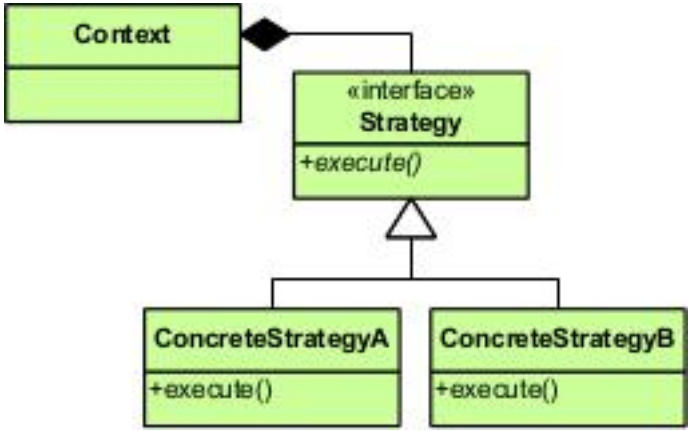
Challenge

- What if I want to apply different ways to handle external requests during different time of a day?

Solution 2: Strategy design pattern

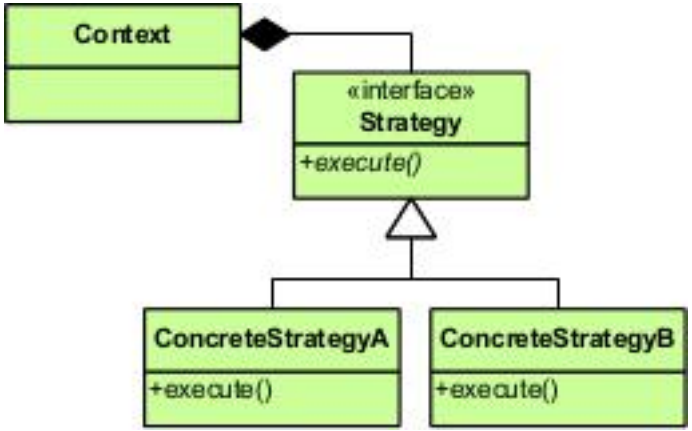
Challenge

- Strategy Pattern



Challenge

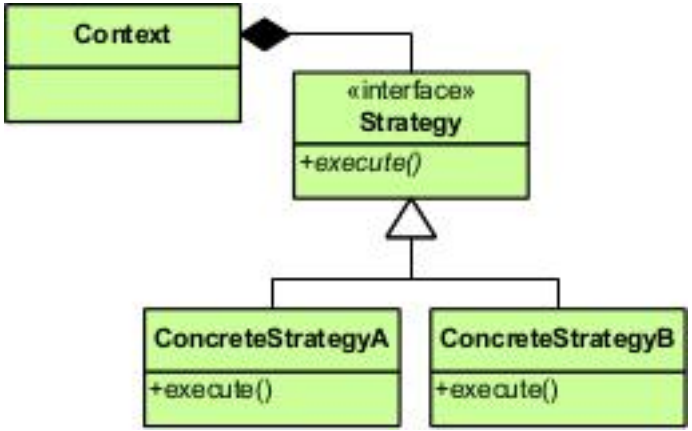
- Strategy Pattern



- **封装**了多种 算法/策略

Challenge

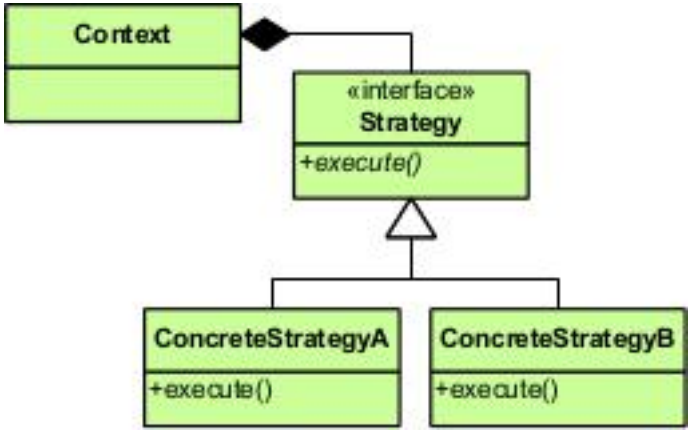
- Strategy Pattern



- **封装**了多种 算法/策略
- 使得算法/策略之间能够**互相替换**

Challenge

- Strategy Pattern



ElevatorSystem
- List<Elevator> elevators - HandleRequestStrategy strategy
+ void handleRequest(ExternalRequest r) + void setStrategy(HandleRequestStrategy s)

《interface》 HandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

PeakHourHandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

NormalHourHandleRequestStaregy
+ void handleRequest(Request r, List<Elevator> elevators)

Challenge

- Strategy design pattern

```
interface HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators);
}

class RandomHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {
        Random rand = new Random();

        int  n = rand.nextInt(elevators.size());

        elevators.get(n).handleExternalRequest(request);
    }
}

class AlwaysOneElevatorHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {

        elevators.get(0).handleExternalRequest(request);
    }
}
```

Challenge

- Strategy design pattern

```
class MyJavaApplication
{
    ElevatorSystem system = new ElevatorSystem();

    system.setStrategy(new RandomHandleRequestStrategy());

    ExternalRequest request = new ExternalRequest(Direction.UP, 3);

    system.handleRequest(request);
}

class ElevatorSystem
{
    private HandleRequestStrategy strategy = new HandleRequestStrategy();
    private List<Elevator> elevators = new ArrayList<>();

    public void setStrategy(HandleRequestStrategy strategy)
    {
        this.strategy = strategy;
    }

    public void handleRequest(ExternalRequest request)
    {
        strategy.handleRequest(request, elevators);
    }
}
```

```
interface HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators);
}

class RandomHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {
        Random rand = new Random();

        int  n = rand.nextInt(elevators.size());

        elevators.get(n).handleExternalRequest(request);
    }
}

class AlwaysOneElevatorHandleRequestStrategy implements HandleRequestStrategy
{
    public void handleRequest(ExternalRequest request, List<Elevator> elevators)
    {

        elevators.get(0).handleExternalRequest(request);
    }
}
```

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
知乎专栏： http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com