

课程大纲

- 什么是OOD
- 这门课是否适合我
- OOD面试当中的陷阱
- 如何评判OOD面试的好坏
- 5C解题法
- 实战演练: Design elevator system
- 后续课程介绍



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

Copyright © www.jiuzhang.com

Copyright © www.jiuzhang.com

OOD介绍

- OOD vs. System Design
- OOA vs. OOD vs. OOP
- 面试频率
- 面试当中的重要性
- OOD介绍小结

OOD vs. System Design

	Object Oriented Design	System Design
面试者	应届毕业生, SDE I -	有经验的面试者, SDE I +
出题目的	OOD常被当做考察面试者综合素质的标准	需要处理大量数据, 提供Service的部门
常见公司	Amazon, Bloomberg,...	Facebook, Google,...
关键字	Viability	Scalability
例题	Design Elevator System	Design Twitter

Copyright © www.jiuzhang.com

Copyright © www.jiuzhang.com

OOD和面试



These questions are not so much about regurgitating design patterns as they are about demonstrating that you understand how to create elegant, maintainable object-oriented code. Poor performance on this type of question may raise serious red flags.

- 面试频率：
 - Phone interview 低
 - Onsite interview 中高频
- 高频公司：
 - Amazon, Bloomberg, TripAdvisor, EMC, Uber...

Copyright © www.jiuzhang.com

OOD和面试



- 出题问法
 - Can you design an elevator system? (设计一个完整的系统)
 - Little amount of coding
 - Can you design scheduler for elevator system? (设计系统中的一个部分)
 - Need implementation

Copyright © www.jiuzhang.com

OOD介绍小结



OOD在面试当中很重要

- 尤其对于Intern和Entry level

题目没有标准答案

- Everything is an object

题目类型明显，可用固定的模板解题

Copyright © www.jiuzhang.com

课程要求



- Coding skill
 - Java entry level, 有基本的Java知识, 了解基本的数据结构如Array, List, HashMap等
- Design pattern
 - 不需要design pattern的基础, 我们将会在课程中讲解如何运用常见的design pattern来为面试加分
- Time commitment
 - 每节课时2小时, 一周两节课, 一共五节课10小时
 - Lintcode 做题, 每周一小时

Copyright © www.jiuzhang.com

OOD面试中的陷阱



- 起手式的陷阱
- 解题中的陷阱
- 收手式的陷阱

Copyright © www.jiuzhang.com

起手式的陷阱



- 先下手为强

例子:

Okay, for a parking lot, we need multiple levels. Also let's make it able to park bicycles.

忌: 自说自话, 倏忽而谈

宜: 先通过和面试官的交流获得有效信息

Copyright © www.jiuzhang.com

起手式的陷阱



- 指鹿为马

例子:

Let's say you need to support 5000 people to park in this parking lot, morning and night will be your peak hour...

I can try to use this algorithm/data structure to find the closest spot.

忌: Algorithm, OOD, System design 傻傻分不清楚

宜: 询问清楚题目要求

Copyright © www.jiuzhang.com

解题中的陷阱



- 朝出夕改

例子:

Oh, wait a second, I think I can optimize the process by modifying/removing this class

忌: 思路混乱, 反复修改

宜: 先给面试官呈现一个Viable的成果

Copyright © www.jiuzhang.com

收手式的陷阱

- 虎头蛇尾

忌：虎头蛇尾，功亏一篑

宜：检查你的设计，是否能够支持所需的功能



Copyright © www.jiuzhang.com

S.O.L.I.D原则

- S – Single responsibility principle
- O – Open close principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency inversion principle

Copyright © www.jiuzhang.com

S.O.L.I.D原则

- Single responsibility principle 单一责任原则

一个类应该有且只有一个去改变他的理由，这意味着一个类应该只有一项工作。

```
public class AreaCalculator
{
    private float result;

    public float calculateArea(Shape s)
    {
        //calculates the area for a given shape, store in result
    }

    public String printAreaAccurateToOneDecimalPlace()
    {
        //prints result accurate to one decimal place
    }
}
```

减小一个类的 dependency

Copyright © www.jiuzhang.com



S.O.L.I.D原则

- Open close principle 开放封闭原则

对象或实体应该对扩展开放，对修改封闭 (Open to extension, close to modification)。

```
public class AreaCalculator
{
    public float calculateArea(Triangle t)
    {
        //calculates the area for triangle
    }

    public float calculateArea(Rectangle r)
    {
        //calculates the area for rectangle
    }
}
```

扩展开放：随着需求的增加，
设计也要很好的 handle

可以用一个 interface
有更多需求的时候可以 implement 这个 interface

interface 是一份合同
要履行的义务就是把里面定义的所有的函数要
implement 出来

可以用很多 interface，只能 extend 一个父类

Copyright © www.jiuzhang.com

S.O.L.I.D原则

- Liskov substitution principle 里氏替换原则

任何一个子类或派生类应该可以替换它们的基类或父类

```
public class Shape
{
    abstract public float calculateVolume();
    abstract public float calculateArea();
}

public class Rectangle extends Shape
{
    //...
}

public class Cube extends Shape
{
    //...
}
```

Copyright © www.jiuzhang.com

S.O.L.I.D原则

- Interface segregation principle 接口分离原则

不应该强迫一个类实现它用不上的接口

```
public interface Shape
{
    public float calculateVolume();
    public float calculateArea();
}

public class Rectangle implements Shape
{
    //...
}

public class Cube implements Shape
{
    //...
}
```

Copyright © www.jiuzhang.com

S.O.L.I.D原则

- Dependency inversion principle 依赖反转原则

抽象不应该依赖于具体实现，具体实现应该依赖于抽象

```
public class AreaCalculator
{
    public Shape shape;
    public String shape_string;

    public AreaCalculator()
    {
        if(shape_string.equals("Triangle"))
        {
            shape = new Triangle();
        }
        else if(shape_string.equals("Rectangle"))
        {
            shape = new Rectangle();
        }
    }

    public float area()
    {
        return shape.area();
    }
}
```

high level 的实体
不应该依赖于 low level 的实体

Copyright © www.jiuzhang.com

```
public class AreaCalculator
{
    public float calculateArea(ShapeInterface s)
    {
        //calculates the area for triangle
    }
}

interface ShapeInterface
{
    public float area();
}

public class Rectangle implements ShapeInterface
{
    private float length;
    private float width;

    public float area()
    {
        return length * width;
    }
}

public class Triangle implements ShapeInterface
{
    private float height;
    private float width;

    public float area()
    {
        return (height * width) / 2;
    }
}
```



Copyright © www.jiuzhang.com

Elevator

- Can you design an elevator system for this building?

奔雷手 - 5C解题法

- Clarify
- Core objects
- Cases
- Classes
- Correctness



Copyright © www.jiuzhang.com

九章算法

奔雷手 - 5C解题法

Clarify

说人话: 通过和面试官交流, 去除题目中的歧义, 确定答题范围
争取问出隐藏信息, 自己掌握大概的框架

Core objects

说人话: 确定题目所涉及的类, 以及类之间的映射关系

Cases

说人话: 确定题目中所需要实现的场景和功能

Classes

说人话: 通过类图的方式, 具体填充题目中涉及的类

Correctness

说人话: 检查自己的设计, 是否满足关键点

Copyright © www.jiuzhang.com

Clarify

Example: Glass of water ?



Copyright © www.jiuzhang.com



Clarify

- What
- How
- Who

Copyright © www.jiuzhang.com

九章算法

Clarify



- What

针对题目中的关键字来提问，帮助自己更好的确定答题范围。

*大多数的关键字为名词，通过名词的属性来考虑

Copyright © www.jiuzhang.com

Clarify



关键字1: Elevator

属性？

Copyright © www.jiuzhang.com

Clarify



关键字1: Elevator



问重量，而不是问能装多少人

Copyright © www.jiuzhang.com

Clarify



关键字1: Elevator



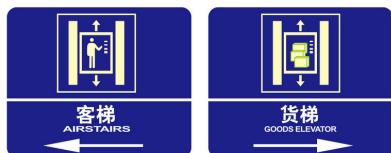
- 没有电梯厢
- 每层楼之间不同

Copyright © www.jiuzhang.com

Clarify



关键字1: Elevator



- 是否需要设计两种类，如果需要它们之间是什么关系？
- 客梯和货梯有什么区别？

Copyright © www.jiuzhang.com

Clarify



关键字1: Elevator

针对本题：所有电梯厢均为相同规格

Copyright © www.jiuzhang.com

Clarify



关键字2: Building

属性？

Copyright © www.jiuzhang.com

Clarify



关键字2: Building



楼有多大/楼有多高/楼内能容纳多少人？

- 通用属性，对于题目帮助不大

Copyright © www.jiuzhang.com

Clarify



关键字2: Building



是否有多处能搭乘的电梯口?

- 当收到一个搭乘电梯的请求时，有多少电梯能够响应？

Copyright © www.jiuzhang.com

Clarify



关键字2: Building

针对本题：每层仅一处能搭乘，所有电梯均可响应

Copyright © www.jiuzhang.com

Clarify



- How

针对问题主题的规则来提问，帮助自己明确解题方向。

*此类问题没有标准答案，你可以提出一些解决方法，通过面试官的反应，选择一个你比较有信心（简单）的方案

Copyright © www.jiuzhang.com

Clarify



电梯有哪些规则？

Copyright © www.jiuzhang.com

Clarify



当按下按钮时，哪一台电梯会相应？

- 同方向 > 静止 > 反向
- 一半负责奇数楼层，一半负责偶数楼层
- ...

Copyright © www.jiuzhang.com

Clarify



当电梯在运行时，哪些按键可以响应？

- 是否能按下反向的楼层

Copyright © www.jiuzhang.com

Clarify



规则：

对于本题：同向 > 静止 > 反向，当运行时不能按下反向的楼层

信息：电梯至少需要三种状态，并且要知道当前在哪一层

Copyright © www.jiuzhang.com

Clarify



- Who

设计由人主导 VS. 设计由系统主导？

Copyright © www.jiuzhang.com

Clarify



- Who

设计由人主导 VS. 设计由系统主导?

Input → BLACK BOX → Output

Copyright © www.jiuzhang.com

Clarify



- Who

Optional, 通过思考题目当中是否有人的出现, 来帮助确定解题范围

*一般可以考虑人的角色以及人的属性, 看是否题目需要

Copyright © www.jiuzhang.com

Clarify



人物属性?

Copyright © www.jiuzhang.com

Clarify



Copyright © www.jiuzhang.com

电梯系统如何获取每位乘客的重量?

- Passenger class包含重量
- 电梯能够自动感应当前重量

Clarify

对于本题：

电梯能够获取当前重量



Copyright © www.jiuzhang.com



Core Object

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

Copyright © www.jiuzhang.com

Core Object



- 什么是Core Object

为了完成设计，需要哪些类？

Copyright © www.jiuzhang.com



Core Object

- 为什么要定义Core Object ?
 - 这是和面试官初步的纸面contract
 - 承上启下，来自于Clarify的结果，成为Use case的依据
 - 为画类图打下基础

Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?
- 以一个Object作为基础，线性思考
- 确定Objects之间的映射关系

Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ? 写在白板上

ElevatorSystem

Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?



Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?

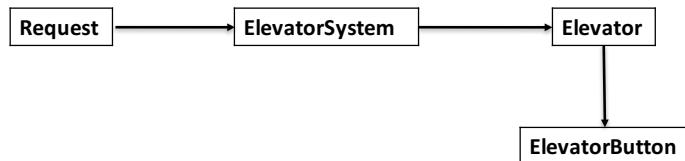


Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?

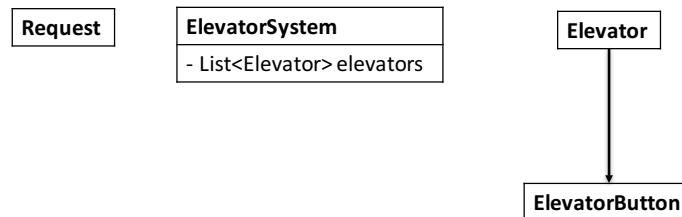


Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?

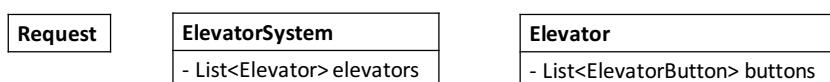


Copyright © www.jiuzhang.com

Core Object



- 如何定义Core Object ?



“-”这个减号的意思是 private



Copyright © www.jiuzhang.com

Good Practice



- Access modifier 不同级别的 access level
 - package
 - public
 - private
 - protected

Copyright © www.jiuzhang.com

Good Practice



- package

如果什么都不声明，变量和函数都是package level visible的，在同一个package内的其他类都可以访问

Example:

```
String input = "test";
void printInput()
{
    System.out.println(input);
}
```

一般不要使用默认，什么都不画
package level 的用途是方便用于 test

在类图中，避免使用default的package level access

Copyright © www.jiuzhang.com

Good Practice



- public

如果声明为public，变量和函数都是public level visible的，任何其他的类都可以访问

Example:

```
public static void main(String[] arguments)
{
    //...
}
```

无论所有 package，都可以访问到
java 的 main 函数必须要 public
画类图用“+”代表 public

在类图中，用“+”表示一个变量或者函数为public

Copyright © www.jiuzhang.com

Good Practice



- private

如果声明为private，变量和函数都是class level visible的，这是所有access modifier中限制最多的一个。仅有定义这些变量和函数的类自己可以访问。

private也是OOD当中实现封装的重要手段。

Example:

```
public class AreaCalculator()
{
    private Logger log;
}

# protected: class level + child of this class
           实现继承的重要手段
```

- private: class level, 用于封装,
可以只 expose 你想 expose 的部分

在类图中，用“-”表示一个变量或者函数为private

Copyright © www.jiuzhang.com

Good Practice



- protected

如果声明为protected，变量和函数在能被定义他们的类访问的基础上，还能够被该类的子类所访问。

protected也是OOD当中实现继承的重要手段。

```
class AudioPlayer
{
    protected Speaker speaker;
}

class StreamingAudioPlayer extends AudioPlayer
{
    public void openSpeaker()
    {
        speaker.open();
    }
}
```

在类图中，用“#”表示一个变量或者函数为protected

Copyright © www.jiuzhang.com

Cases



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

Copyright © www.jiuzhang.com

Cases



- 什么是Use case ?

在你设计的系统中，需要支持哪些功能？

Copyright © www.jiuzhang.com

Cases



- 为什么要写Use cases ?
- 这是你和面试官白纸黑字达成的第二份共识，把你将要实现的功能列在白板上
- 帮助你在解题过程中，理清条例，一个一个Case实现
- 作为检查的标准

Copyright © www.jiuzhang.com

Cases



- 怎么写Use cases ?
 - 利用定义的Core Object，列举出每个Object对应产生的use case.
 - 每个use case只需要先用一句简单的话来描述即可

Copyright © www.jiuzhang.com

Cases



- ElevatorSystem
 - Handle request

Copyright © www.jiuzhang.com

Cases



- Elevator
 - Take external request
 - Take internal request
 - Open gate
 - Close gate
 - Check weight

每个 class 可以拥有多个函数，
都是跟电梯相关的功能。

所以除了 check weight 可以使用一个外面的 sensor，
其他都是电梯的功能。

What about single responsibility principle?

Copyright © www.jiuzhang.com

Cases



- ElevatorButton
 - Press button

Copyright © www.jiuzhang.com

Class



- 什么是类图？
- 为什么要画类图？
- 怎么画类图？

In software engineering,
a class diagram in the Unified Modeling Language (UML)
is a type of static structure diagram that describes the structure of a system

by showing the system's classes, their attributes, operations (or methods),
and the relationships among objects.

Copyright © www.jiuzhang.com

Class



- Class diagram (类图)

Class Name
Attributes
Functions

Copyright © www.jiuzhang.com

Class



- 为什么要画类图?
- 可交付, Minimal Viable Product
- 节省时间, 不容易在Coding上挣扎
- 建立在Use case上, 和之前的步骤层层递进, 条例清晰, 便于交流和修改
- 如果时间允许/面试官要求, 便于转化成Code

Copyright © www.jiuzhang.com

Class



- 怎么画类图?
- 遍历你所列出的use cases
- 对于每一个use case, 更加详细的描述这个use case在做什么事情
(例如: take external request -> ElevatorSystem takes an external request, and decide to push this request to an appropriate elevator)
- 针对这个描述, 在已有的Core objects里填充进所需要的信息

Copyright © www.jiuzhang.com

Class



Request	ElevatorSystem	Elevator
	- List<Elevator> elevators	- List<ElevatorButton> buttons



Copyright © www.jiuzhang.com

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

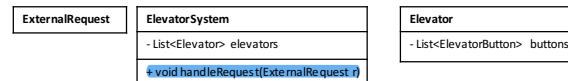
Class



- Use case: Handle request

ElevatorSystem takes an **external request**, and decide to push this request to an appropriate **elevator**

Class



Copyright © www.jiuzhang.com



Copyright © www.jiuzhang.com

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

Good Practice



- 如何知道一个函数，是否成功完成任务？

地下一层电梯关闭，这时有人在地下一层按了向上的按钮，会发生什么？

Good Practice



- 如何知道一个函数，是否成功完成任务？

- Use boolean instead of void

成功的话返回true, 否则返回false

缺点：一元返回值，只知道对错，并不知道错的原因

- 如何知道是什么地方出错？

Copyright © www.jiuzhang.com

Copyright © www.jiuzhang.com

Good Practice

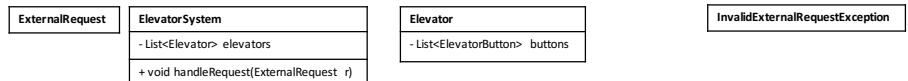


- 如何知道一个函数，是否成功完成任务？

- Use exceptions throw 异常，为了记录，知道哪里有问题

Copyright © www.jiuzhang.com

Class



ElevatorButton

Copyright © www.jiuzhang.com



Class

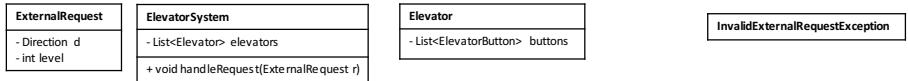


- Use case: Take external request

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

Copyright © www.jiuzhang.com

Class



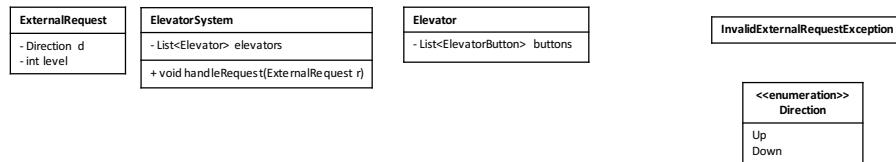
ElevatorButton

Copyright © www.jiuzhang.com



Class

九章算法



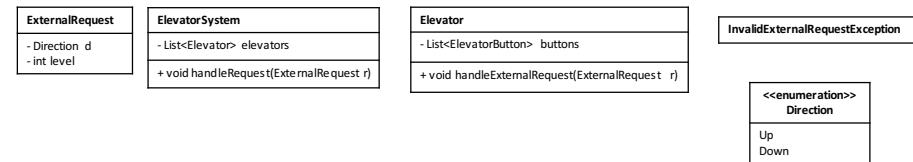
ElevatorButton

Copyright © www.jiuzhang.com



Class

九章算法



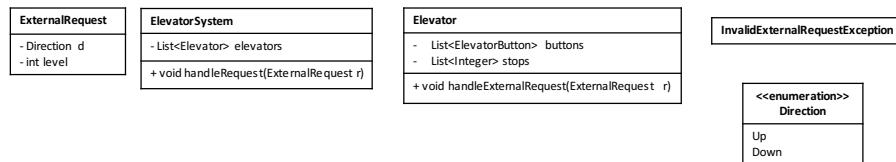
ElevatorButton

Copyright © www.jiuzhang.com



Class

九章算法



ElevatorButton

Copyright © www.jiuzhang.com



Challenge

九章算法

- 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}

Copyright © www.jiuzhang.com

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

Copyright © www.jiuzhang.com

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}

Copyright © www.jiuzhang.com

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？

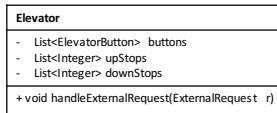
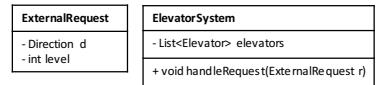
stops will be {2, 3, 5}

Expected is: {3, 5, 2}

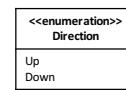
Solution: keep 2 lists for different direction

Copyright © www.jiuzhang.com

Class



InvalidExternalRequestException



ElevatorButton

Copyright © www.jiuzhang.com



Class

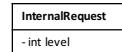
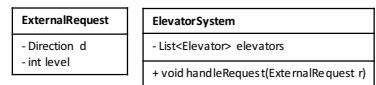
- Use case: Take internal request

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

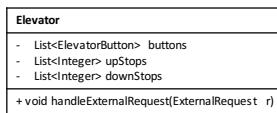


Copyright © www.jiuzhang.com

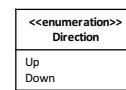
Class



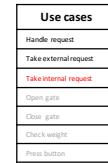
ElevatorButton



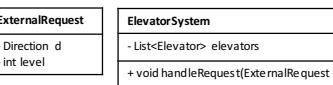
InvalidExternalRequestException



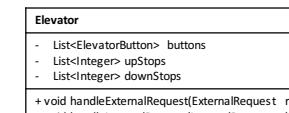
Copyright © www.jiuzhang.com



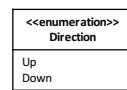
Class



ElevatorButton



InvalidExternalRequestException



Copyright © www.jiuzhang.com



Class

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

InternalRequest
- int level

九章算法

Class

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

InternalRequest	Elevator
- int level	- List<ElevatorButton> buttons
+ void handleInternalRequest(InternalRequest r)	- List<Integer> upStops
- boolean isRequestValid(InternalRequest r)	- List<Integer> downStops

InvalidExternalRequestException
<>enumeration>>

Direction
Up
Down

九章算法

ElevatorButton

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

Copyright © www.jiuzhang.com

ElevatorButton

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

Copyright © www.jiuzhang.com

Challenge

九章算法

- 如何判断一个Internal request 是否为Valid?

Copyright © www.jiuzhang.com

Challenge

九章算法

- 如何判断一个Internal request 是否为Valid?

Solution:

If elevator **going up**

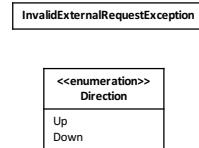
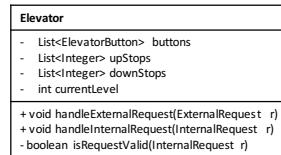
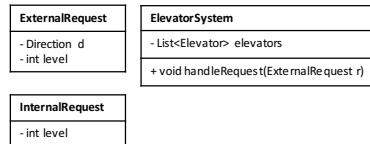
requested level lower than **current level**
invalid

If elevator **going down**

requested level higher than **current level**
invalid

Copyright © www.jiuzhang.com

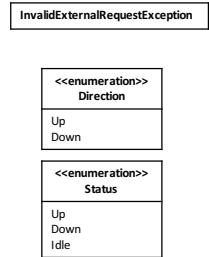
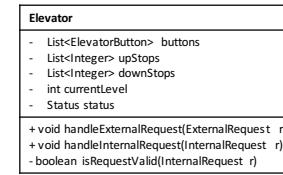
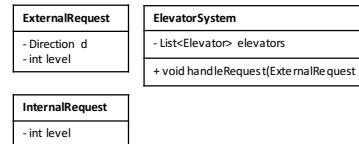
Class



Copyright © www.jiuzhang.com



Class



Copyright © www.jiuzhang.com

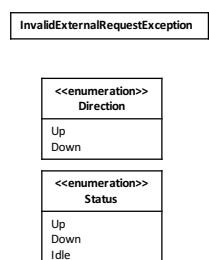
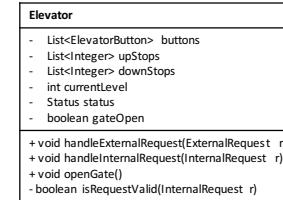
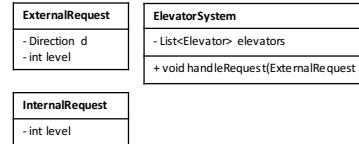
Class

- Use case: Open gate

An **elevator** reaches the destination level, open gate



Class



Copyright © www.jiuzhang.com

Class



- 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.

Copyright © www.jiuzhang.com

Class

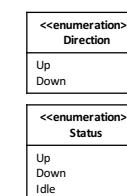
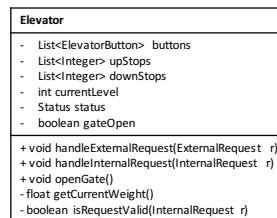
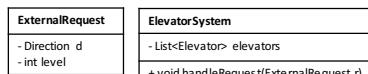


- Use case: check weight

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

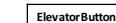
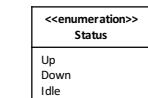
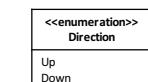
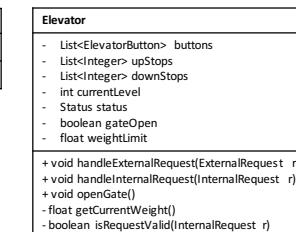
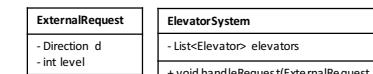
Copyright © www.jiuzhang.com

Class



Copyright © www.jiuzhang.com

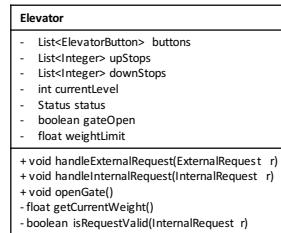
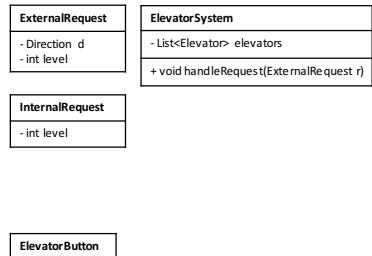
Class



Copyright © www.jiuzhang.com

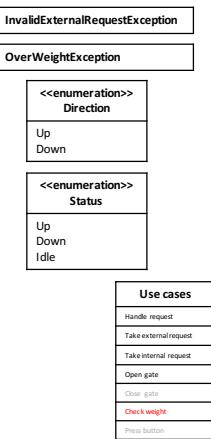


Class

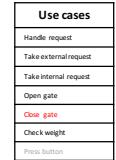
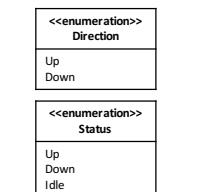
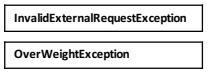
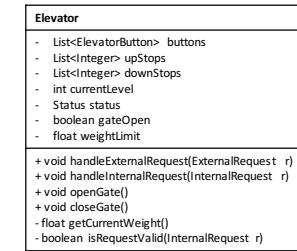
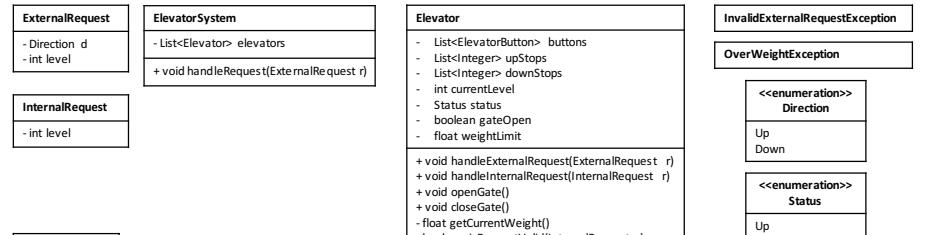


Copyright © www.jiuzhang.com

九章算法



Class



Copyright © www.jiuzhang.com

Class

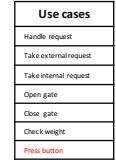
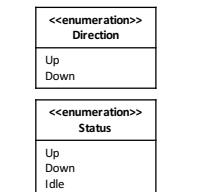
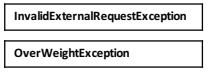
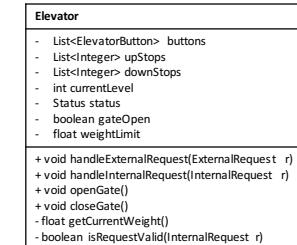
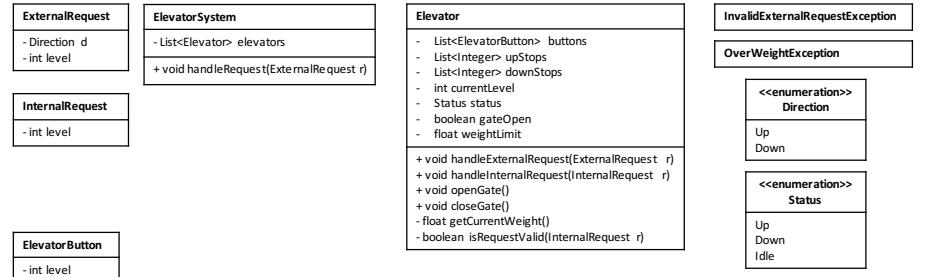
- Use case: press button

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

Copyright © www.jiuzhang.com

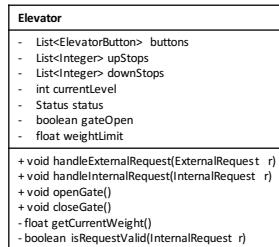
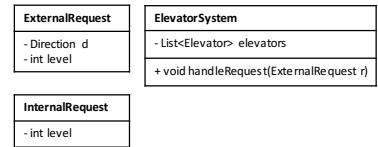
九章算法

Class



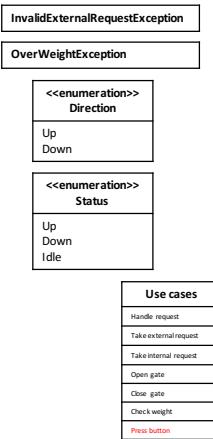
Copyright © www.jiuzhang.com

Class



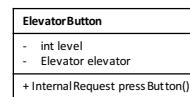
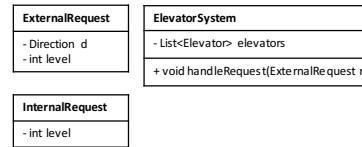
Copyright © www.jiuzhang.com

九章算法



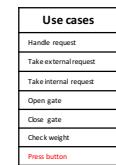
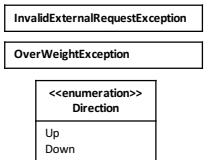
Copyright © www.jiuzhang.com

Class



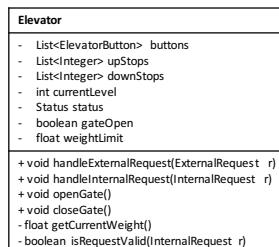
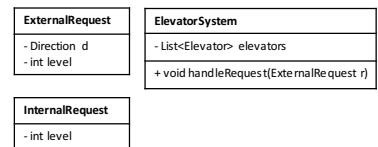
```
Elevator
- List<ElevatorButton> buttons
- List<Integer> upStops
- List<Integer> downStops
- int currentLevel
- Status status
- boolean gateOpen
- float weightLimit

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

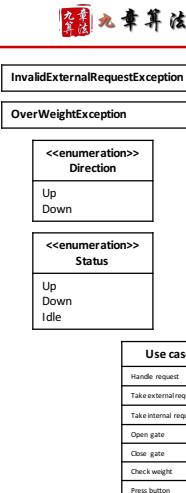


Copyright © www.jiuzhang.com

Class – Final view



Copyright © www.jiuzhang.com



Copyright © www.jiuzhang.com

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

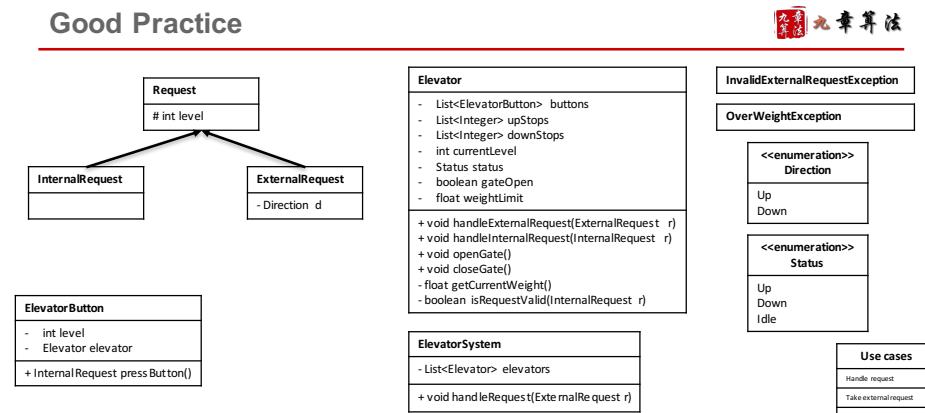
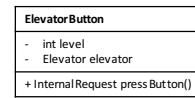
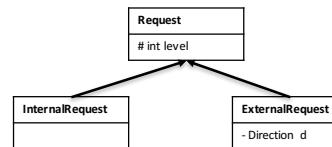
- 继承

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

Copyright © www.jiuzhang.com



Good Practice



Copyright © www.jiuzhang.com

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?

Copyright © www.jiuzhang.com

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

Copyright © www.jiuzhang.com



Challenge



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

Copyright © www.jiuzhang.com

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
    }
}
```

Copyright © www.jiuzhang.com

Challenge



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

Solution 2: Strategy design pattern

Context

interface
Strategy

1. 封装了多种算法/策略
※ 方便其他系统使用
※ 方便找到问题，修改
2. 使得算法/策略之间能够互相替换

strategyA

strategyB

Copyright © www.jiuzhang.com

Recap



- 什么是OOD
- SOLID原则
- 5C 解题法
- Good practice: Access modifier
- Good practice: Exception
- Design pattern: Strategy

Copyright © www.jiuzhang.com

Next...



第2章

本节大纲

管理类面向对象设计 OOD for Management System

- 管理类 OOD 面试题型特点分析
- 实战OOD面试真题：
 - 停车场问题 Parking lot
 - 餐厅管理问题 Restaurant
- 设计模式讲解 Design Pattern: Singleton

Next...



第3章

本节大纲

预定类面向对象设计 OOD for Reservation System

- 预定类面试题型特点分析
- 实战面试真题：
 - 酒店预订系统设计 Hotel Reservation
 - 航空机票预订系统设计 Airline Ticket Reservation

Copyright © www.jiuzhang.com

Copyright © www.jiuzhang.com

Next...



第4章

本节大纲

实物类面向对象设计 OOD for Real Life Object

- 实物类面试题型特点分析
- 实战面试真题：
 - Vending machine
 - Juke box
- 设计模式讲解 Design Pattern: Factory
- 设计模式讲解 Design Pattern: Adaptor

Next...



第5章

本节大纲

游戏棋牌类面向对象设计 OOD for Games

- 棋牌游戏类面试题型特点分析
- 棋牌游戏类面试题特殊技巧讲解
- 实战面试真题：
 - Black Jack
 - Chinese chess
- 课程总结及面试技巧点拨

Copyright © www.jiuzhang.com

Copyright © www.jiuzhang.com