

# Java 面向对象基础

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

- 我不太能够区分OOD和系统设计；想要能够系统的学习OOD的知识点

- 我不太能够区分OOD和系统设计；想要能够系统的学习OOD的知识点
- 我是在读的学生，还没有面试经验；想要学习如何准备OOD的面试

- 我不太能够区分OOD和系统设计；想要能够系统的学习OOD的知识点
- 我是在读的学生，还没有面试经验；想要学习如何准备OOD的面试
- 我经常被OOD题型的面试题难住，不知道应该从何下手；想要学习解题方法和技巧

# What will you be learning from lesson 1



九章算法

- 什么是OOD，他和系统设计有什么区别？
- OOD经常在面试中出现吗？它重要吗？
- 怎么样的设计才算是好的设计？
- 如何解答OOD的题目 – 5C解题法
- 这门课有什么要求吗？

# What will you be learning after lesson 1



九章算法

- 什么是OOD，他和系统设计有什么区别？
- OOD经常在面试中出现吗？它重要吗？
- 怎么样的设计才算是好的设计？
- 如何解答OOD的题目 – 5C解题法
- 这门课有什么要求吗？
  
- OOD的题目类型划分：
  - 管理类 / 预定类 / 实物类 / 游戏类



# OOD vs. System Design

	Object Oriented Design	System Design
面试者		
出题目的		
常见公司		
关键字		
例题		



# 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		
常见公司		
关键字		
例题		

# 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		
关键字		
例题		

# 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, Uber,...	Facebook, Twitter,...
关键字		
例题		

# 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, Uber,...	Facebook, Twitter,...
关键字	Viability	Scalability
例题		

# 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, Uber,...	Facebook, Twitter,...
关键字	Viability	Scalability
例题	Design Elevator System	Design Twitter

- 面试频率：
  - Phone interview 低
  - Onsite interview 中高频

- 面试频率：
  - Phone interview 低
  - Onsite interview 中高频
- 面试重要性：
  - 考察作为程序员的基础和全局观
  - 在一些公司拥有一票否决权

- 面试频率：
  - Phone interview 低
  - Onsite interview 中高频
- 面试重要性：
  - 考察作为程序员的基础和大局观
  - 在一些公司拥有一票否决权
- 高频公司：
  - Amazon, Bloomberg, TripAdvisor, EMC, Uber...



- Coding skill
  - Java entry level, 有基本的Java知识, 了解基本的data structure如Array, List, HashMap等

- Coding skill
  - Java entry level, 有基本的Java知识, 了解基本的data structure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

- Coding skill
  - Java entry level, 有基本的Java知识, 了解基本的data structure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

# 如何评判一轮OOD面试

---

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

- 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。

- 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 getResult()
    {
        return this.result;
    }

    public float calculateArea(Triangle t)
    {
        this.result = h * b / 2;
    }
}
```

- Single responsibility principle 单一责任原则

```
public class AreaCalculator
{
    private float result;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea(Triangle t)
    {
        this.result = h * b / 2;
    }
}
```

```
public class AreaCalculator
{
    private float result;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea(Triangle t)
    {
        this.result = h * b / 2;
    }

    public void printResultInJson()
    {
        jsonPrinter.initialize();
        jsonPrinter.print(this.result);
        jsonPrinter.close();
    }
}
```



- Single responsibility principle 单一责任原则

```
public class AreaCalculator
{
    private float result;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea(Triangle t)
    {
        this.result = h * b / 2;
    }
}

public class Printer
{
    public printInJson(float number)
    {
        jsonPrinter.initialize();
        jsonPrinter.print(this.result);
        jsonPrinter.close();
    }
}
```

- 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)。

- 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
    }
}
```

- 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 interface Shape
{
    public float getArea();
}

public class Triangle implements Shape
{
    public float getArea()
    {
        return b * h / 2;
    }
}
```

```
public class AreaCalculator
{
    private float result;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea(Shape s)
    {
        this.result = s.getArea();
    }
}
```

- 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

- 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

- 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 calculateVolumn();
    abstract public float calculateArea();
}

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

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

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

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

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



- 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

High-level的实体不应该依赖于low-level的实体

- 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

High-level的实体不应该依赖于low-level的实体

```
public class AreaCalculator
{
    private float result;
    private Triangle t;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea()
    {
        this.result = t.h * t.b / 2;
    }
}
```

- 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

High-level的实体不应该依赖于low-level的实体

```
public interface Shape
{
    public float getArea();
}

public class Triangle implements Shape
{
    public float getArea()
    {
        return b * h / 2;
    }
}
```

```
public class AreaCalculator
{
    private float result;

    public float getResult()
    {
        return this.result;
    }

    public float calculateArea(Shape s)
    {
        this.result = s.getArea();
    }
}
```

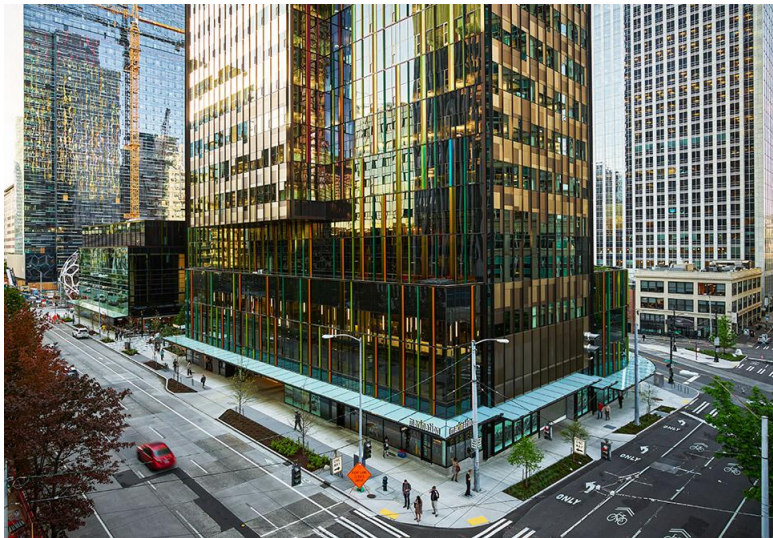
# 面试中应该怎么做？

---

- 实战演练

# Elevator

- Can you design an elevator system for this building?



- Clarify
- Core objects
- Cases
- Classes
- Correctness



## 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

## 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

Example: Glass of water ?



Example: Glass of water ?



# Clarify

Example: Glass of water ?



# Clarify

Example: Glass of water ?



# Clarify

Example: Glass of water ?



# Clarify

- What
- How

## - 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

关键字1: Elevator

属性?

## 关键字1: Elevator



- 可能需要考虑获取每辆电梯的目前重量

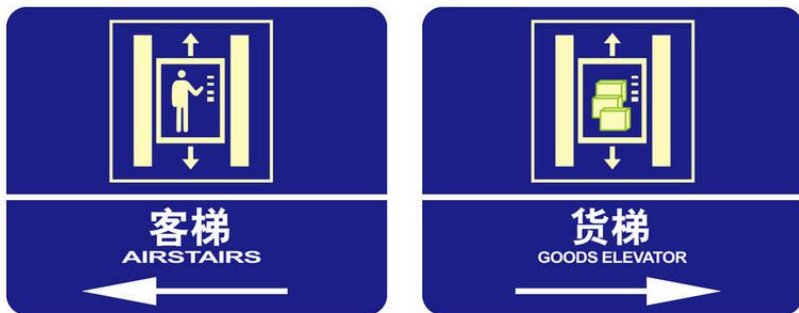


## 关键字1: Elevator



- 可能需要考虑获取每辆电梯的目前重量
- What's the weight limit of the elevator ?
- Do we need to consider overweight for our elevator system ?

## 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？

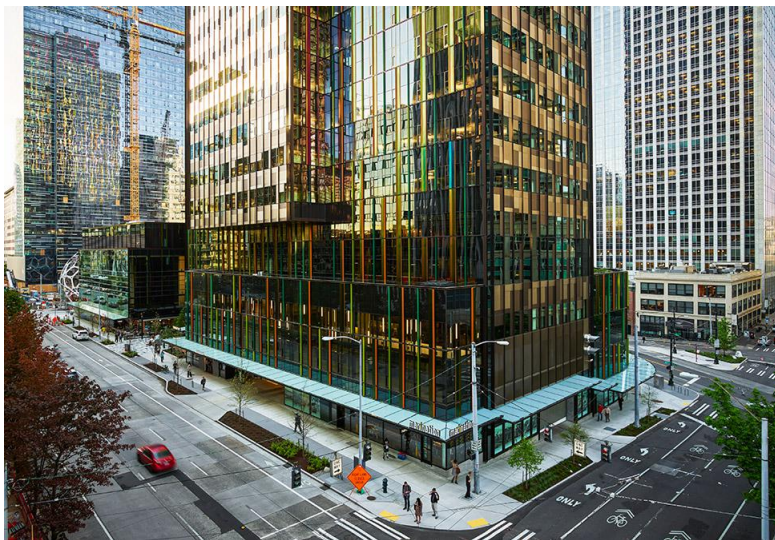
关键字1: Elevator

针对本题: 所有电梯厢均为相同规格

关键字2: Building

属性?

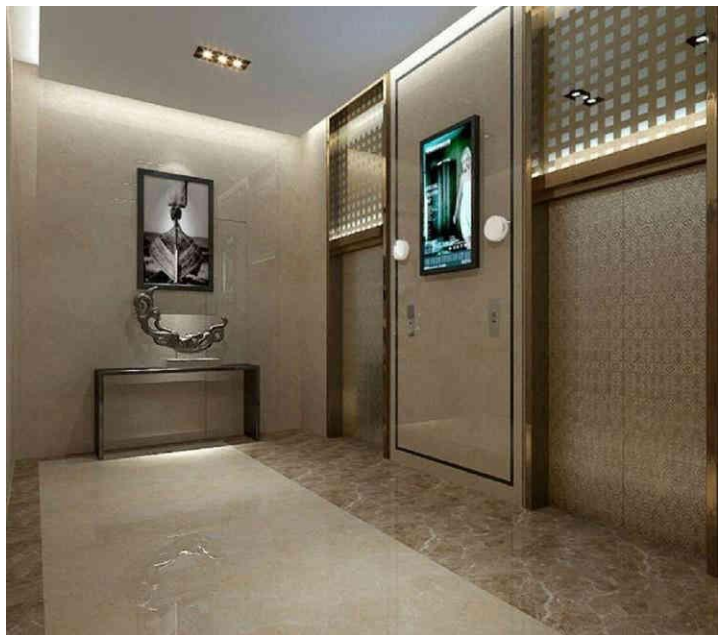
## 关键字2: Building



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

## 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？

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

## - 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



电梯有哪些规则？



如何判断电梯是否超重？



如何判断电梯是否超重？

- Passenger class包含重量
- 电梯能够自动感应当前重量



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
- ...



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

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

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

- 什么是Core Object

为了完成设计，需要哪些类？



- 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

- 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

- 如何定义Core Object ?

**ElevatorSystem**

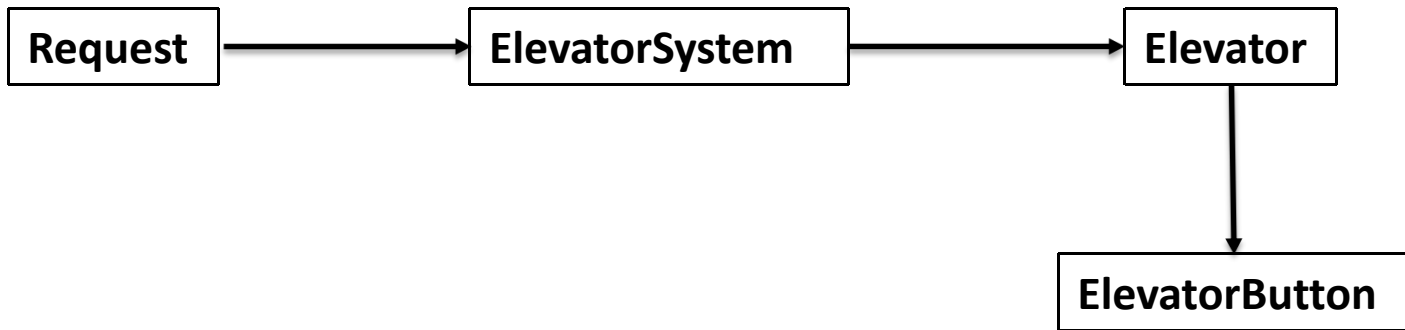
- 如何定义Core Object ?



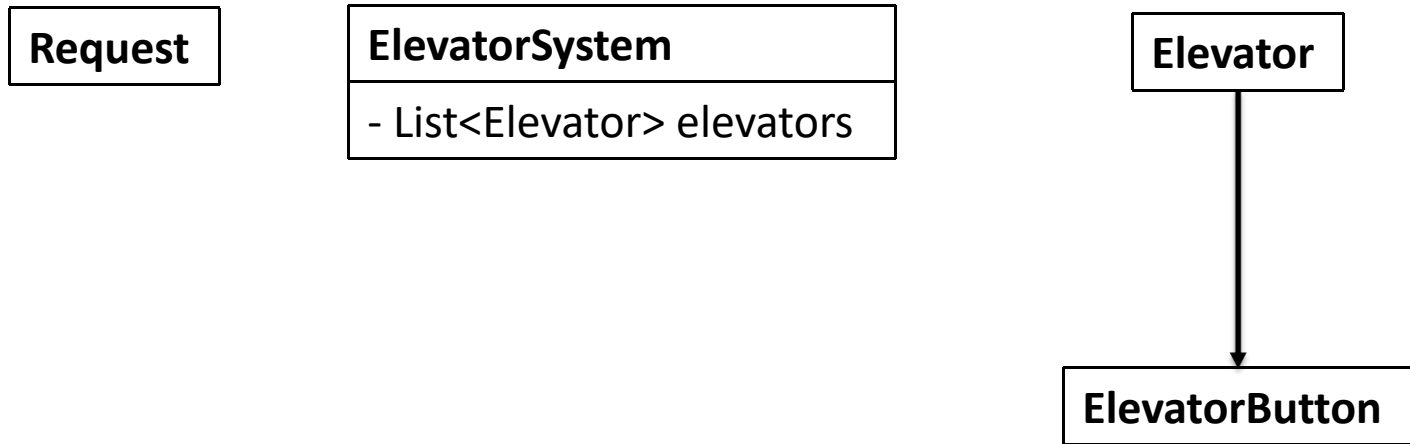
- 如何定义Core Object ?



- 如何定义Core Object？



- 如何定义Core Object ?



- 如何定义Core Object ?

**Request**

**ElevatorSystem**

- List<Elevator> elevators

**Elevator**

- List<ElevatorButton> buttons

**ElevatorButton**

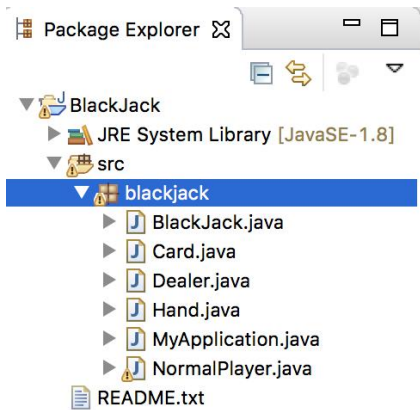


- Access modifier
  - package
  - public
  - private
  - protected

- 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:



```
public class Blackjack
{
    String name = "test";
}

public class Card
{
    public void printName()
    {
        System.out.println(name);
    }
}
```

在类图中，避免使用default的package level access

- 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)
{
    //...
}
```

在类图中，用“+”表示一个变量或者函数为public

- 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;  
}
```

在类图中，用“-”表示一个变量或者函数为private

- 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。

Example:

```
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

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

- 什么是Use case ?

在你设计的系统中，需要支持哪些功能？

- 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



- 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

- ElevatorSystem

- ElevatorSystem
  - Handle request

- Request

N/A

- Elevator

- Elevator
  - Take external request

- Elevator
  - Take external request
  - Take internal request

- Elevator
  - Take external request
  - Take internal request
  - Open gate



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

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

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

What about single responsibility principle?

- ElevatorButton

- ElevatorButton
- Press button

- 什么是类图？
- 为什么要画类图？
- 怎么画类图？

- Class diagram (类图)

Class Name
Attributes
Functions

- 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



- 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

# Class

**Request**

**ElevatorSystem**

- List<Elevator> elevators

**Elevator**

- List<ElevatorButton> buttons

**ElevatorButton**

## Use cases

Handle request

Take external request

Take internal request

Open gate

Close gate

Check weight

Press button

- Use case: Handle request

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

# Class

**ExternalRequest**

**ElevatorSystem**

- List<Elevator> elevators

+ void handleRequest(ExternalRequest  
r)

**Elevator**

- List<ElevatorButton> buttons

**ElevatorButton**

**Use cases**

**Handle request**

Take external request

Take internal request

Open gate

Close gate

Check weight

Press button

- 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？

- 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

- 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
- 如何知道是什么地方出错？

- 如何知道一个函数，是否成功完成任务？
  - Use exceptions



# 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

- 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

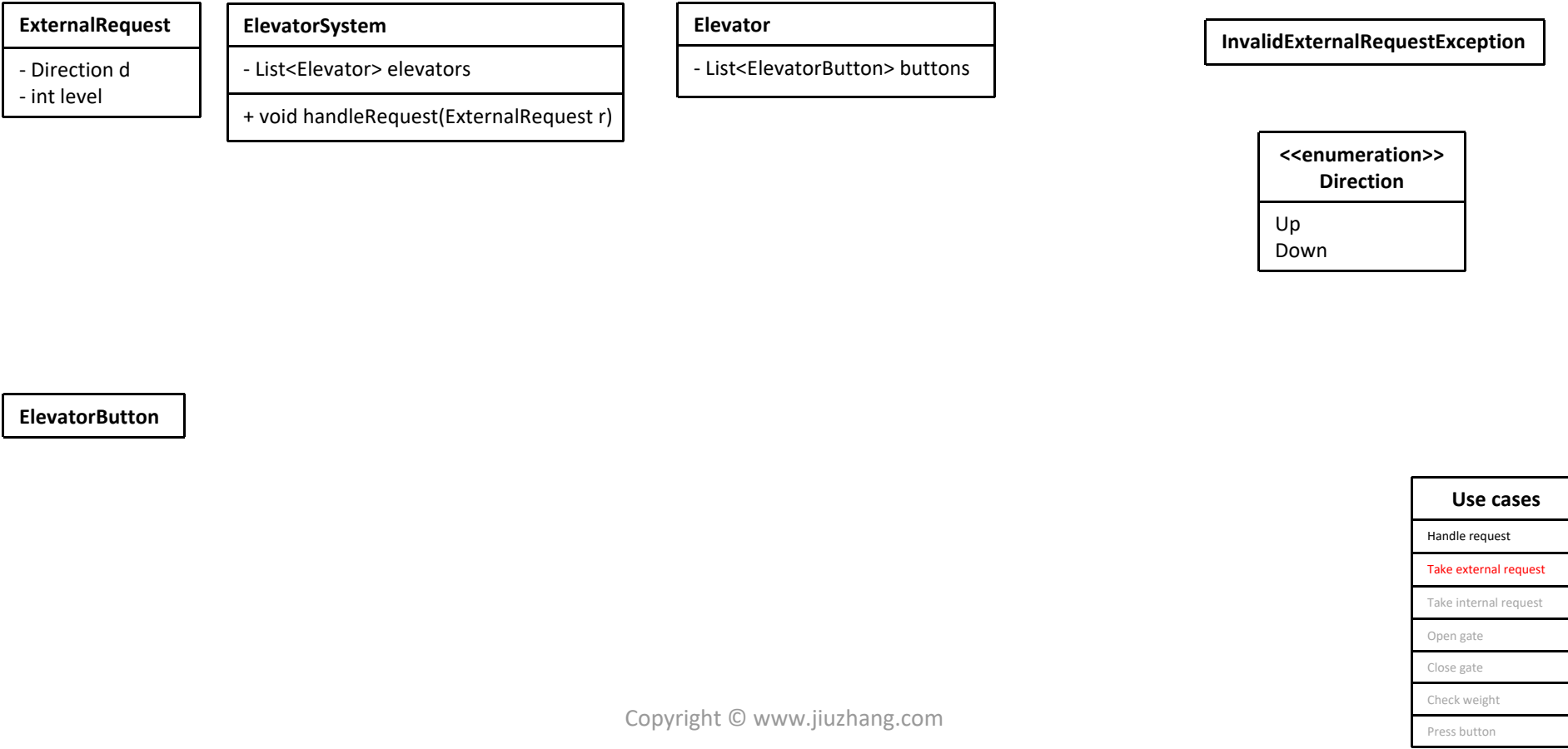
Take internal request

Open gate

Close gate

Check weight

Press button



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

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}

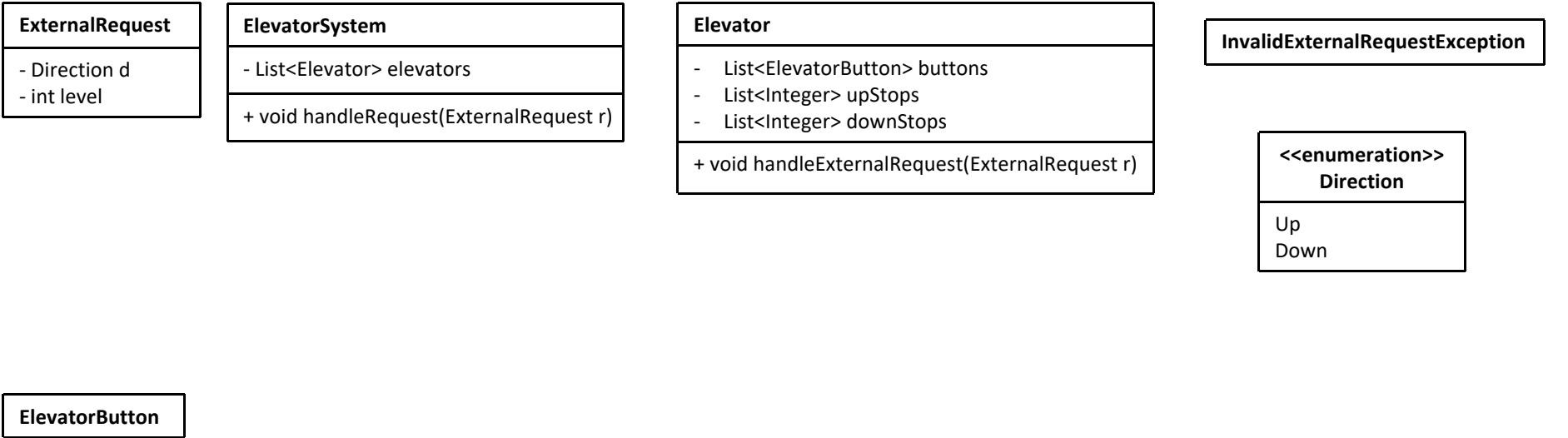
# Challenge

- 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



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?

- 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
    }
}
```



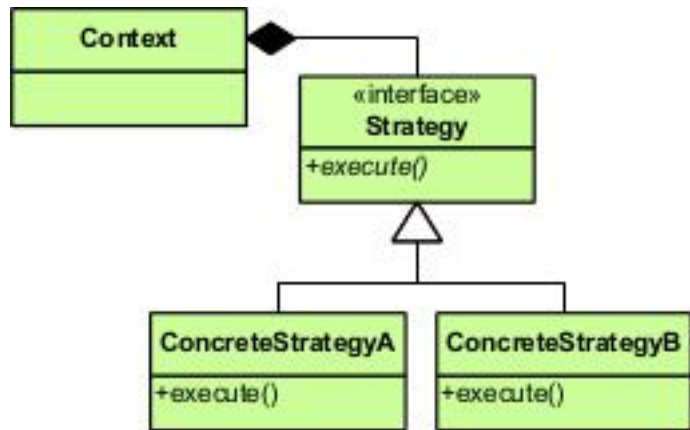
# Challenge

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

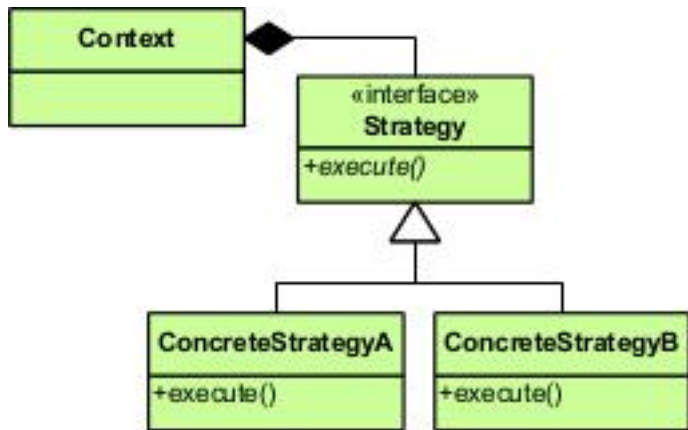
Solution 2: Strategy design pattern

# Challenge

- Strategy Pattern

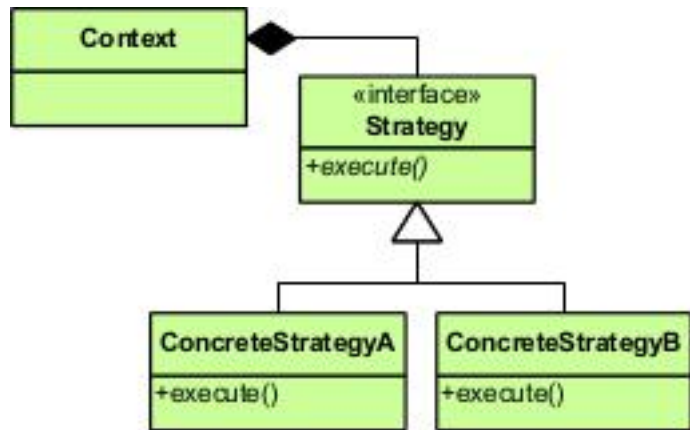


- Strategy Pattern



- 封装了多种 算法/策略

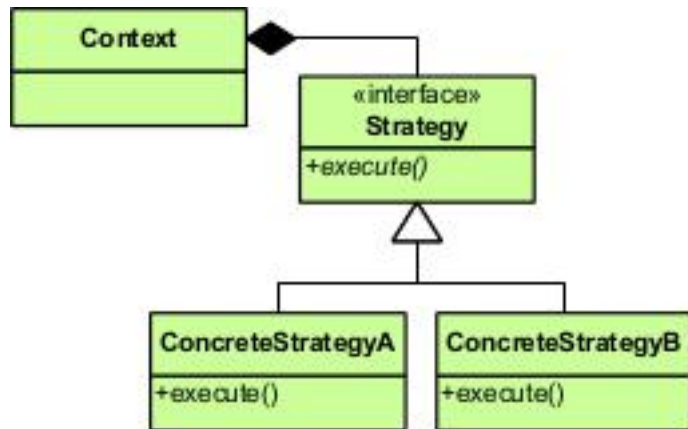
- Strategy Pattern



- 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)

- 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);
    }
}
```

- 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);
    }
}
```

- Use case: Take internal request

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



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

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)

InvalidExternalRequestException
---------------------------------

<<enumeration>> Direction
Up
Down

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

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

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

## 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)

## ElevatorButton

## 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: Open gate

- Use case: Open gate

并行 VS 串行

单线程 VS 多线程

- Use case: Open gate

单线程:

$\{3, 5, 2\} \rightarrow \{5, 2\} \rightarrow \{2\} \rightarrow \{\}$

(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)

- 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();
            }
        }
    }
}
```

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;

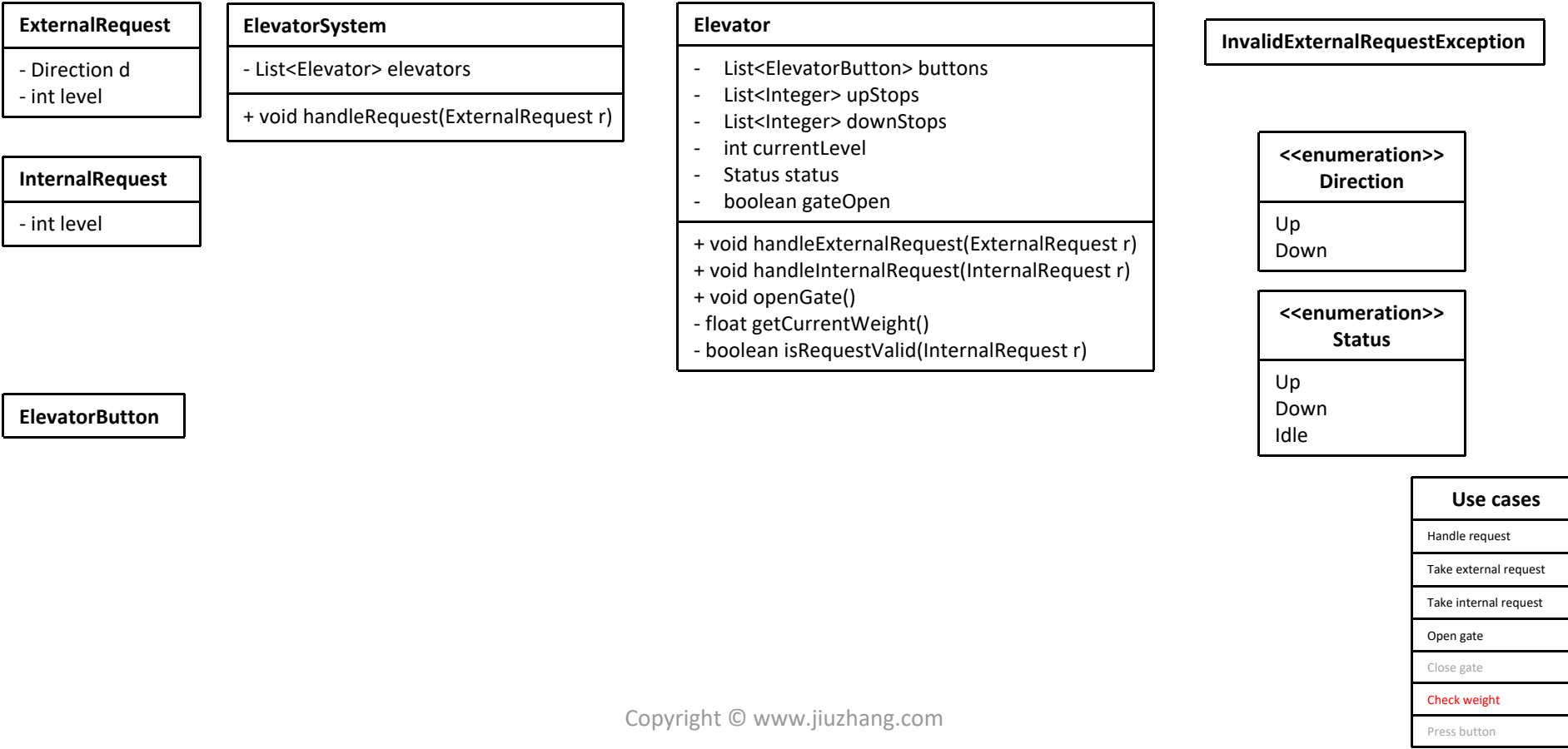
then check stops corresponds to current status;

if no stops left, check the reserve direction stops;

change status to reserve direction or idle.

- 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
- 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

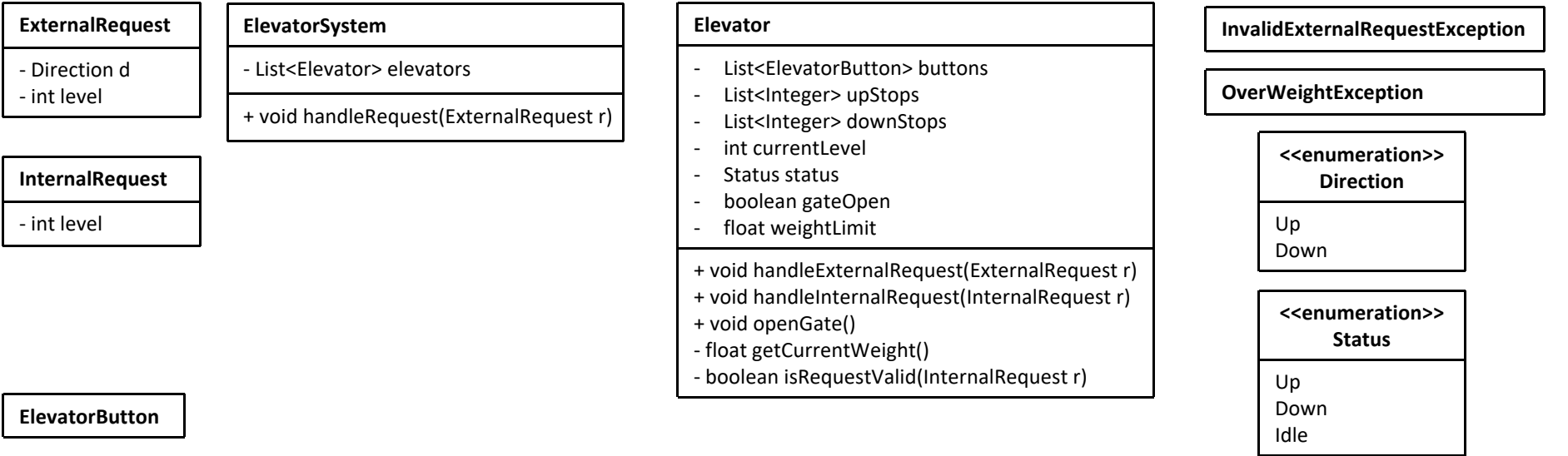
Take internal request

Open gate

Close gate

Check weight

Press button



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

- Use case: press button

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

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

ExternalRequest
- Direction d
- int level

InternalRequest
- int level

ElevatorButton
- int level
+ boolean pressButton()

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

## ElevatorSystem

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

## InternalRequest

- int level

## 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

## ElevatorSystem

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

## InternalRequest

- int level

## 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

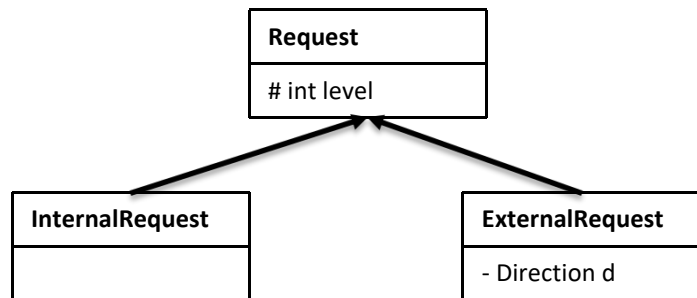
Press button



- 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

- 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



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

- 什么是OOD

# Recap

- 什么是OOD
- SOLID原则

# Recap

---

- 什么是OOD
- SOLID原则
- 5C 解题法

# Recap

---



- 什么是OOD
- SOLID原则
- 5C 解题法
- Good practice: Access modifier

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



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

## 第2章

## 管理类面向对象设计 OOD for Management System

### 本节大纲

- 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

## 第3章

## 预定类面向对象设计 OOD for Reservation System

### 本节大纲

- 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

## 第4章

## 实物类面向对象设计 OOD for Real Life Object

### 本节大纲

- 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

## 第5章

## 游戏棋牌类面向对象设计 OOD for Games

### 本节大纲

- 棋牌游戏类面试题特点分析
- 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



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://zhuankan.zhihu.com/jiuzhang>

微博: <http://www.weibo.com/ninechapter>

官网: [www.jiuzhang.com](http://www.jiuzhang.com)

**已参加**此次分享活动的同学请私聊橙子发送福利口令：**面向对象你和我**

【口令有效期：美西时间6月1日-6月8日】

**未参加**此次分享活动的同学请私聊人工九妹发送：福利 二字，即可参与此次活动。

福利内容：

1. 【价值 \$300】《面向对象设计专题班》抵价券
2. 【价值 ¥249】LintCode VIP 14天
3. 【价值 ¥249】硅谷求职精品讲座VIP 7天
4. ood 推荐书籍系列
5. ood 面试题汇总及参考答案
6. Google Facebook Amazon 秋招求职大礼包



新增了一个班主任

- 督学
- 第二节课开课前2天开班仪式



橙子微信号