OOD chapter 1

- 评判 OOD 面试
  - S – Single responsibility 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。

```java
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
    }
}
```

```java
public class AreaCalculator
{
    private float result;

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

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

public class Printer
{
    public String print(AreaCalculator ac)
    {
        System.out.print(ac.getResult());
    }
}
```

  - O – Open close principle

- 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)。

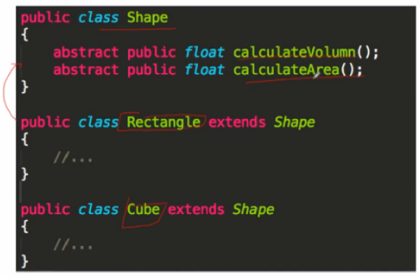
```java
public class AreaCalculator
{

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

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

```java
public class AreaCalculator
{
    public float calculateArea(Shape s)
    {
        //...
    }
}
```

  - L – Liskov substitution principle

- 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

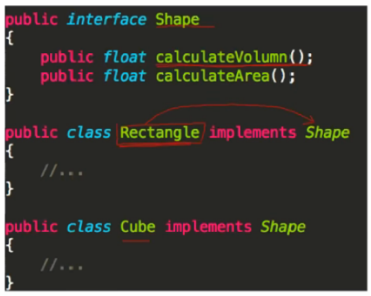


```
public class Shape
{
    abstract public float calculateVolumn();
    abstract public float calculateArea();
}

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

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

- I – Interface segregation principle

- 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

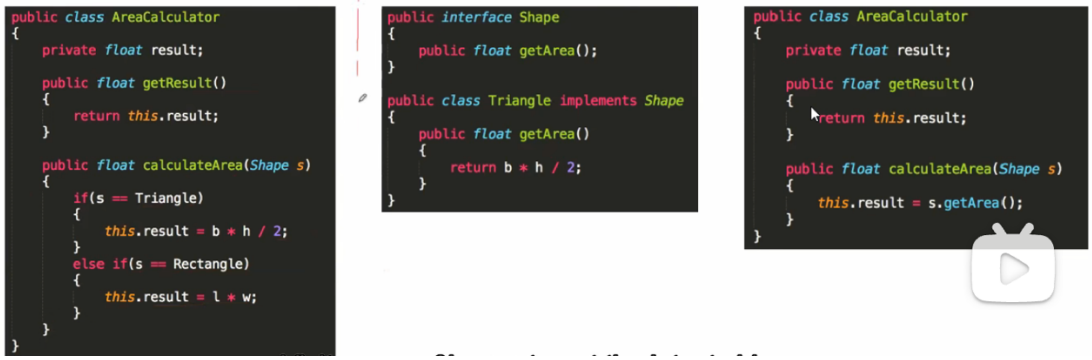


```
public interface Shape
{
    public float calculateVolumn();
    public float calculateArea();
}

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

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

- D – Dependency inversion principle

- 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
{
    private float result;

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

    public float calculateArea(Shape s)
    {
        if(s == Triangle)
        {
            this.result = b * h / 2;
        }
        else if(s == Rectangle)
        {
            this.result = l * w;
        }
    }
}
```

```
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();
    }
}
```

5C 解题法

- Clarify: 通过和面试官交流，去除题目中的歧义，确定答题范围
- Core objects: 确定题目所涉及的类，以及类之间的映射关系
- Cases: 确定题目中所需要实现的场景和功能
- Classes: 通过类图的方式，具体填充题目中涉及的类
- Correctness: 检查自己的设计，是否满足关键点

Design Elevator System

Clarify

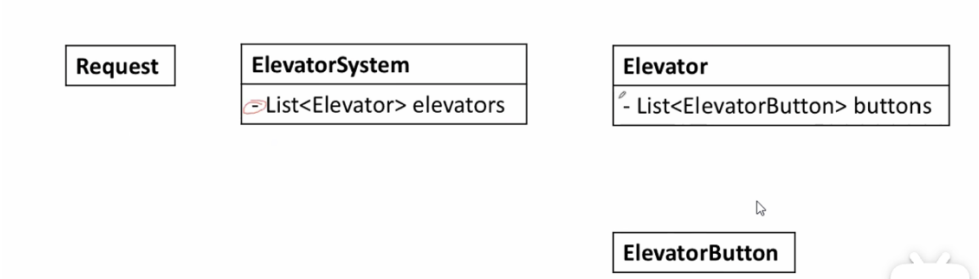
- Overview
    - What: 针对题目中的**关键字**来提问, 通过名词的属性来考虑
    - How: 针对问题主题的**规则**来提问
    - Who: 以**系统**为主导
- What - 关键词
    - Elevator:
        - 电梯能够获取当前重量?
        - 客梯和货梯?
    - Building
        - 是否有多处能搭乘的电梯口?
- How - 规则
    - 当按下按钮时，哪一台电梯会相应
    - 当电梯在运行时，哪些按键可以响应
- Who
    - 电梯系统如何获取每位乘客的重量

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

- 如何定义 Core Object ?
  - 以一个 Object 作为基础，**线性**思考
    - 只考虑 input/output
  - 确定 Objects 之间的映射关系



- Good Practice
  - Use Access modifier
    - package - 尽量避免

- 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

    - public
    - private - 封装

- 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 - 继承

- 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

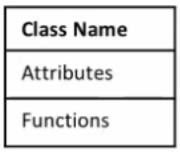
  - package
    - 如果什么都不声明，变量和函数都是 package level visible
    - 在同一个 package 内的其他类都可以访问
    - 在类图中，避免使用 default 的 package level access
    - 常用于写 Unit Test

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
- 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□述即可
- ElevatorSystem
    - Handle request
- Request
    - N/A
- Elevator （USE CASE）
    - Take external request
    - Take internal request
    - Open gate
    - Close gate
    - Check weight
- ElevatorButton
    - Press button

Class

- Class diagram (类图)



- 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
- Use case: Handle request
    - ElevatorSystem takes an external request, and decide to push this request to an appropriate elevator

Correctness

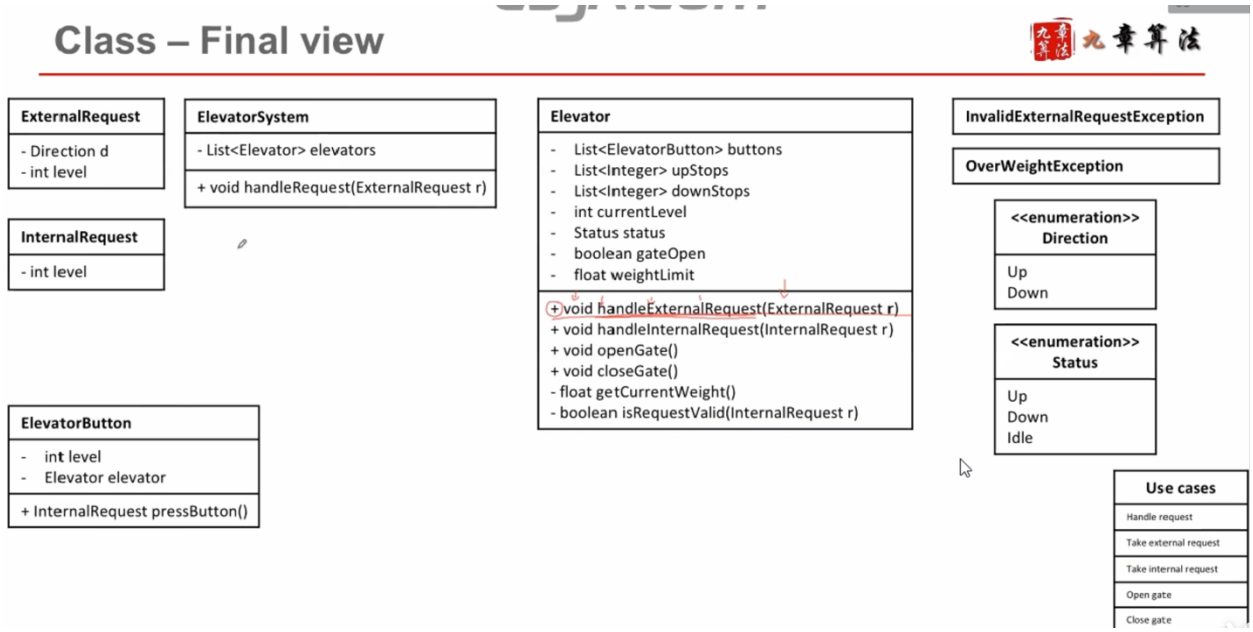
- 从以下几方面检查:
    - Validate use cases (检查是否支持所有的 use case)
    - Follow good practice (面试当中的加分项，展现一个程序员的经验)
    - S.O.L.I.D
    - Design pattern

Challenge

- Q: What if I want to apply different ways to handle external requests during different time of a day?
- A: Use Strategy design pattern

Design Pattern: 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



https://github.com/aliciabellbryner/leetcode-top-interview-questions-master/tree/main/leetcode-top-interview-questions-master/src/OOD