软件工程

装饰模式

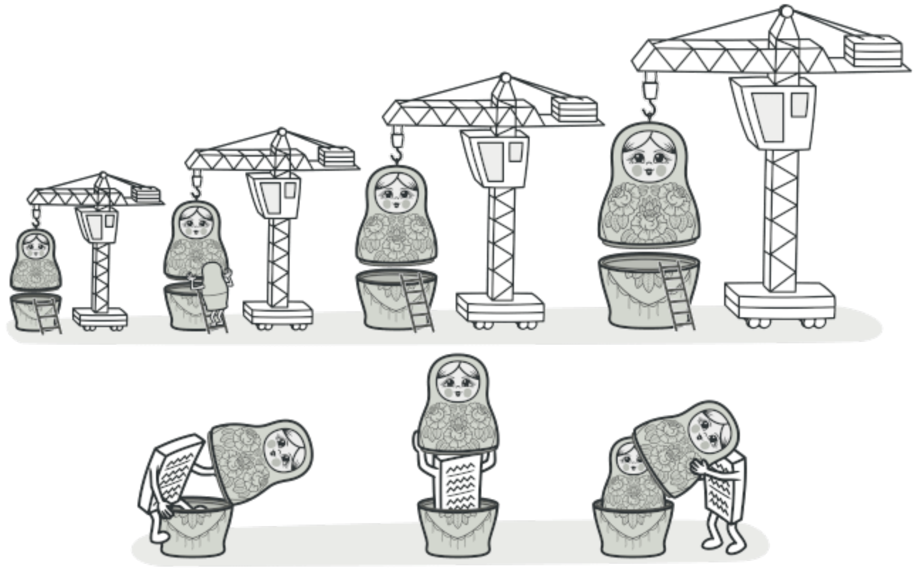
Spring 2022, SWUFE

复习

- 策略模式
- 单例模式
- *静态工厂方法*（不是设计模式，但很实用）
- 工厂方法模式

1. 装饰者模式

Decorator Pattern



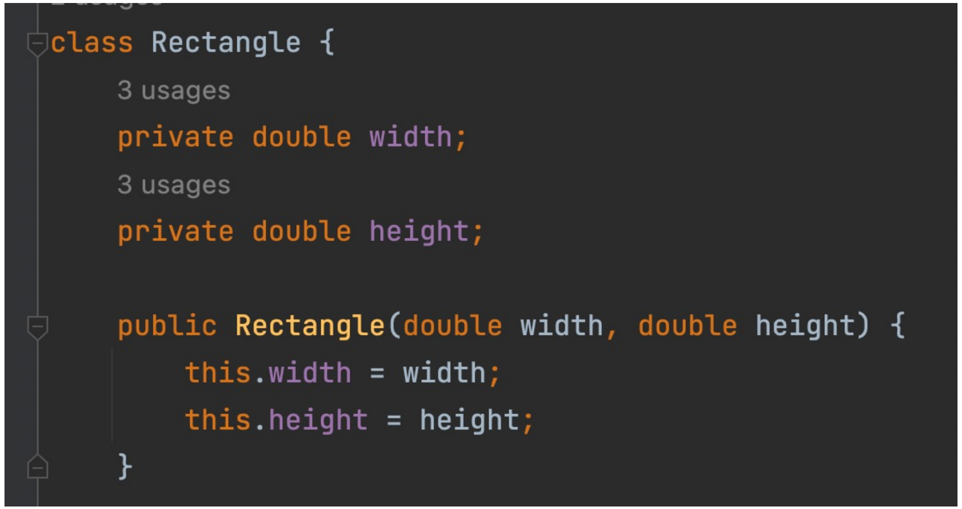
装饰模式允许你动态地添加类的行为。

开闭原则（Open-Closed Principle）

Classes should be open for extension, but closed for modification.

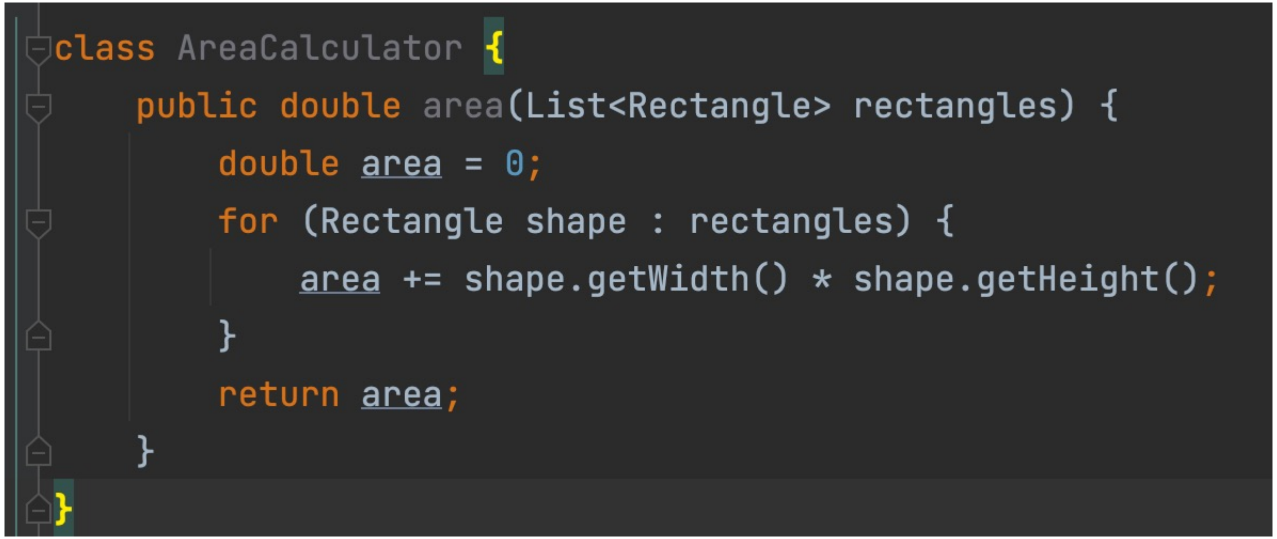
```
if (type.equals("cheese")) {
    pizza = new CheesePizza();
} else if (type.equals("greek")) {
    pizza = new GreekPizza();
} else if (type.equals("pepperoni")) {
    pizza = new PepperoniPizza();
}
```

问自己一个问题：<u>当需求发生改变时，代码是否需要修改</u>？



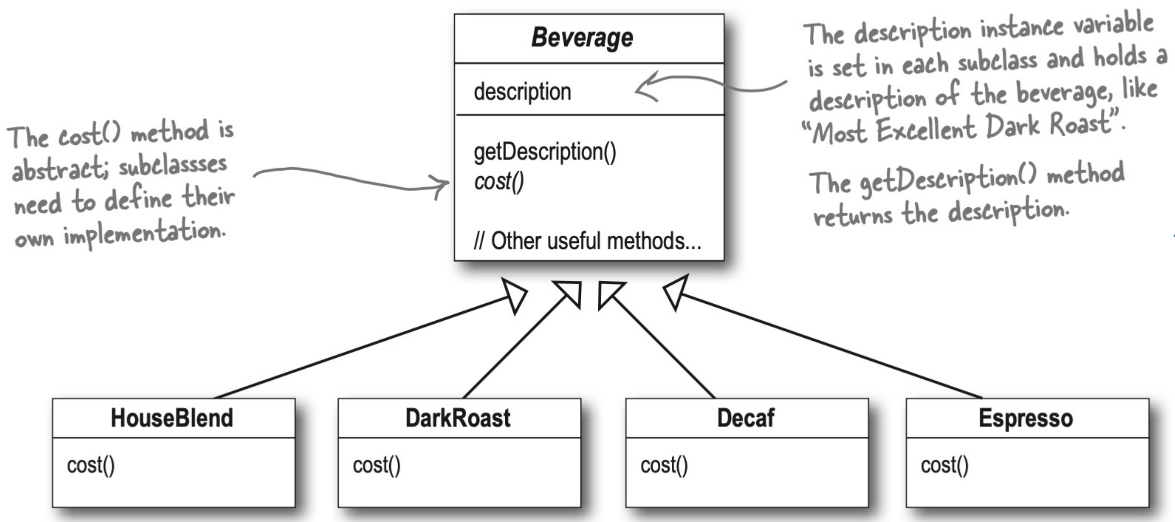
```
class Rectangle {
    3 usages
    private double width;
    3 usages
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
```

背景：你要实现一个图形绘制软件，其中一个需求是计算当前视窗内所有图形的面积之和。



```
class AreaCalculator {
    public double area(List<Rectangle> rectangles) {
        double area = 0;
        for (Rectangle shape : rectangles) {
            area += shape.getWidth() * shape.getHeight();
        }
        return area;
    }
}
```

1.1 案例：咖啡系统

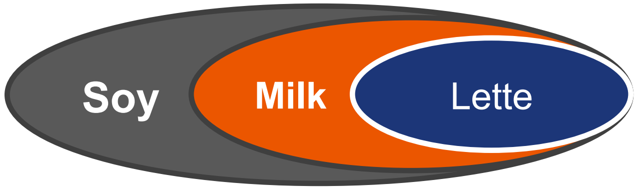

Beverage
description
getDescription()
cost()
// Other useful methods...
The cost() method is abstract; subclassses need to define their own implementation.
The description instance variable is set in each subclass and holds a description of the beverage, like "Most Excellent Dark Roast".
The getDescription() method returns the description.
HouseBlend
cost()
DarkRoast
cost()
Decaf
cost()
Espresso
cost()

如何在不修改现有代码的前提下，实现动态添加配料（功能）？

如果不考虑“开闭原则”

```
Beverage beverage = new Latte();
double price = beverage.cost();
~~beverage.add(new Milk());~~
```

```
abstract class Beverage {
   List<Condiment> condiments;
}
```

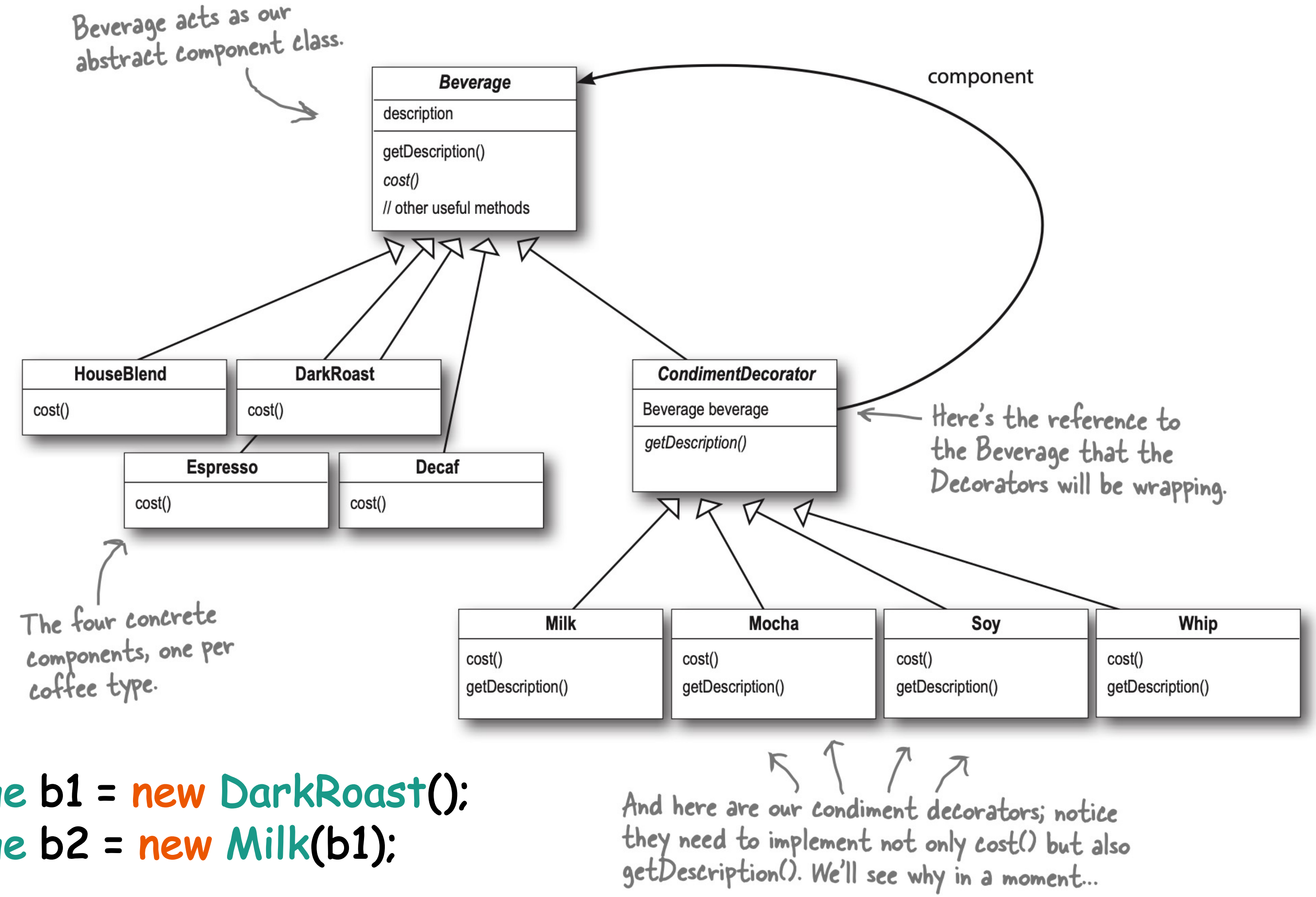


1.2 解决方案

让配料（Condiment）实现/继承（implement/extends）Beverage，同时 Condiment ***has a*** Beverage。

此时，把 Condiment 称为**装饰器**（decorator）。

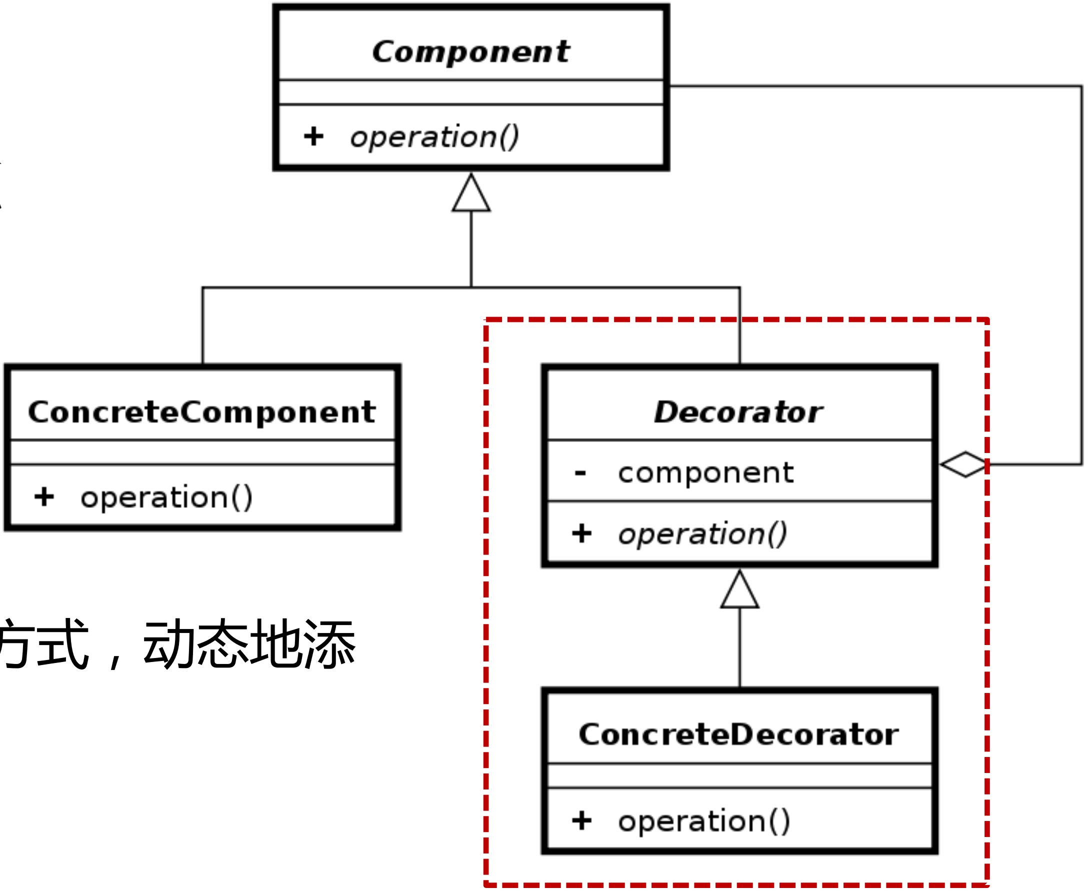
```
absract class CondimentDecorator extends Beverage{
    Beverage beverage;
}
```



Beverage acts as our abstract component class.
Beverage
description
getDescription()
cost()
// other useful methods
component
HouseBlend
cost()
DarkRoast
cost()
Espresso
cost()
Decaf
cost()
CondimentDecorator
Beverage beverage
getDescription()
Here's the reference to the Beverage that the Decorators will be wrapping.
The four concrete components, one per coffee type.
Milk
cost()
getDescription()
Mocha
cost()
getDescription()
Soy
cost()
getDescription()
Whip
cost()
getDescription()
And here are our condiment decorators; notice they need to implement not only cost() but also getDescription(). We'll see why in a moment...

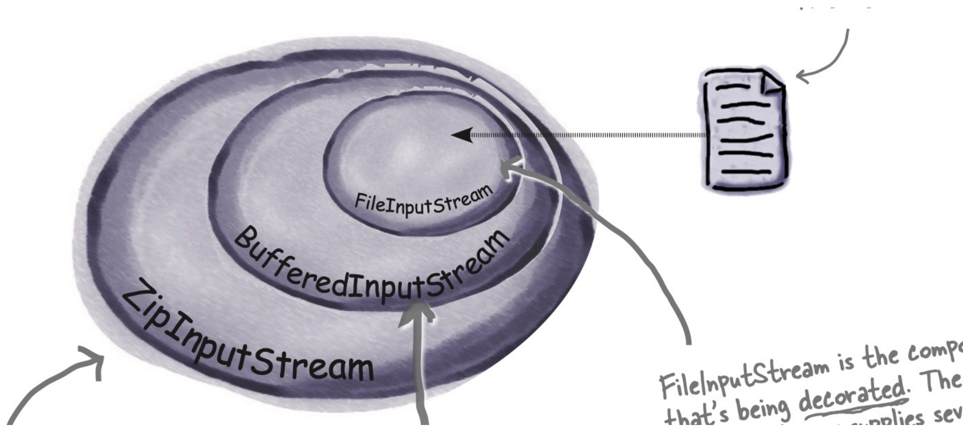
```
Beverage b1 = new DarkRoast();
Beverage b2 = new Milk(b1);
```

1.3 装饰者模式

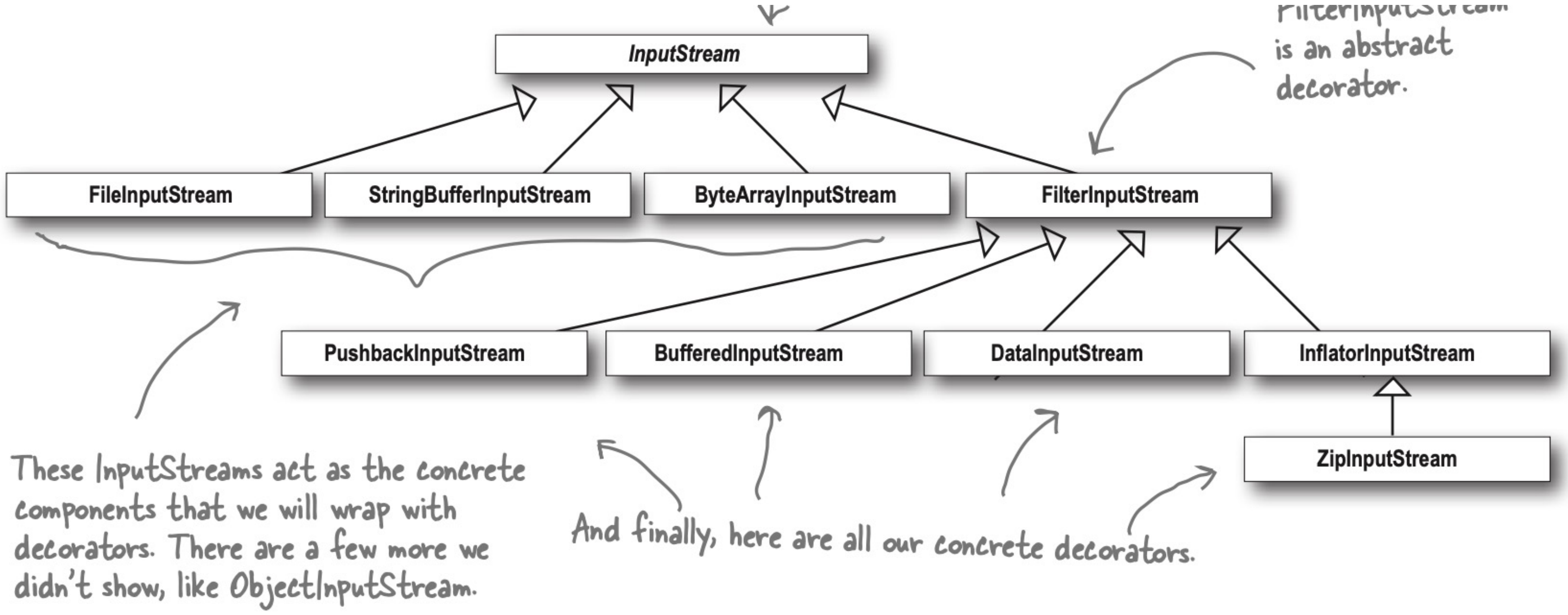


不采用子类继承的方式，动态地添加新的行为

1.4 Java IO: 修饰者模式



```
FileInputStream fin = new FileInputStream("file1.txt");
BufferedInputStream bin = new BufferedInputStream(fin);
```


FilterInputStream is an abstract decorator.
InputStream
FileInputStream
StringBufferInputStream
ByteArrayInputStream
FilterInputStream
PushbackInputStream
BufferedInputStream
DataInputStream
InflatorInputStream
ZipInputStream
These InputStreams act as the concrete components that we will wrap with decorators. There are a few more we didn't show, like ObjectInputStream.
And finally, here are all our concrete decorators.

Head First: Welcome, Decorator Pattern. We've heard that you've been a bit down on yourself lately?

/ˈglæm.ə.əs/ 有魅力的；令人嚮往的

Decorator: Yes, I know the world sees me as the glamorous design pattern, but you know, I've got my share of problems just like everyone.

HeadFirst: Can you perhaps share some of your troubles with us?

Decorator: Sure. Well, you know I've got the power to add flexibility to designs, that much is for sure, but I also have a *dark side*. You see, I can sometimes add a lot of small classes to a design, and this occasionally results in a design that's less than straightforward for others to understand.

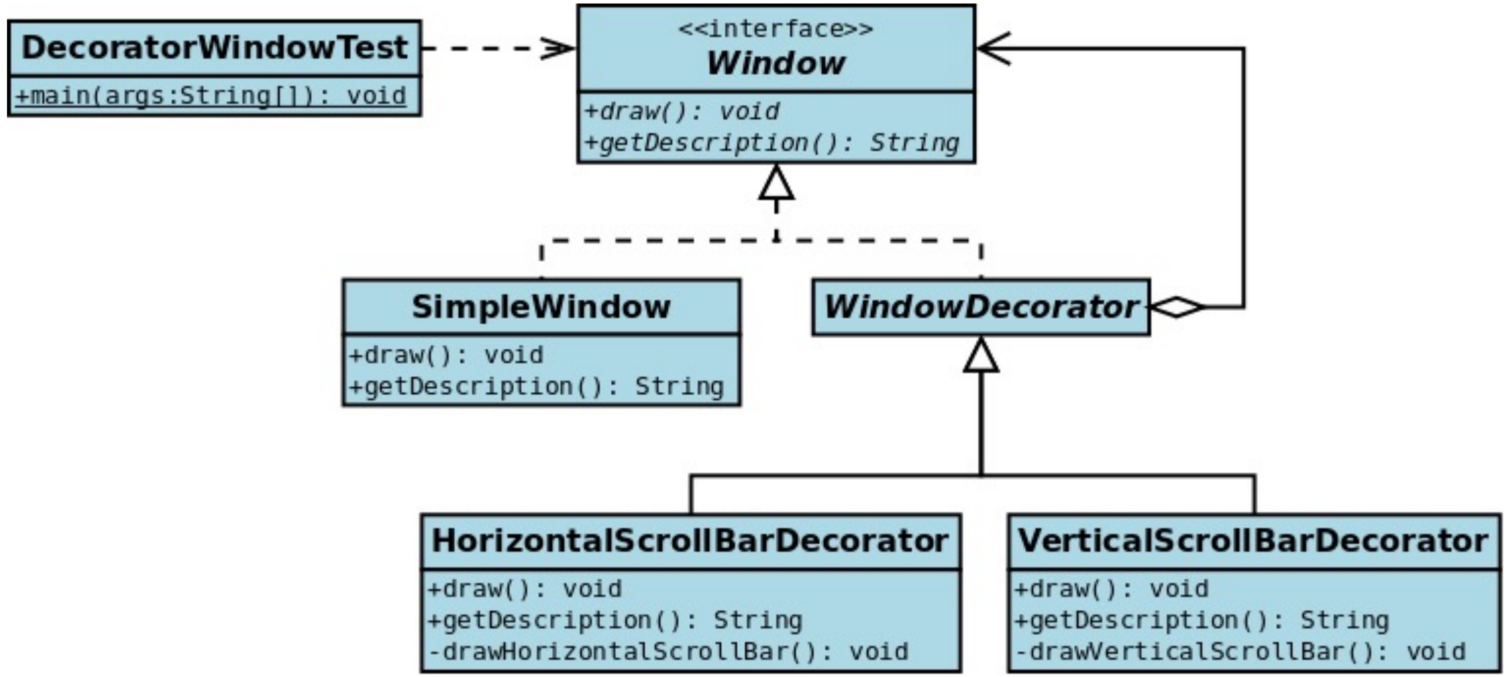
HeadFirst: Can you give us an example?

/noʊˈtɔːr.i.əs/ 臭名昭著的，聲名狼藉的

Decorator: Take the Java I/O libraries. These are notoriously difficult for people to understand at first. But if they just saw the classes as a set of wrappers around an InputStream, life would be much easier.

https://refactoringguru.cn/design-patterns/decorator

1.5 更多例子


DecoratorWindowTest
+main(args:String[]): void
<<interface>>
Window
+draw(): void
+getDescription(): String
SimpleWindow
+draw(): void
+getDescription(): String
WindowDecorator
HorizontalScrollBarDecorator
+draw(): void
+getDescription(): String
-drawHorizontalScrollBar(): void
VerticalScrollBarDecorator
+draw(): void
+getDescription(): String
-drawVerticalScrollBar(): void

1.6 课堂练习

用熟悉的面向对象语言实现装饰者模式。

https://github.com/bethrobson/Head-First-Design-Patterns/tree/master/src/headfirst/designpatterns/decorator/starbuzz

2. 外观模式

Facade Pattern

建築學術語，一般指建築物的外牆（尤其是正面）。

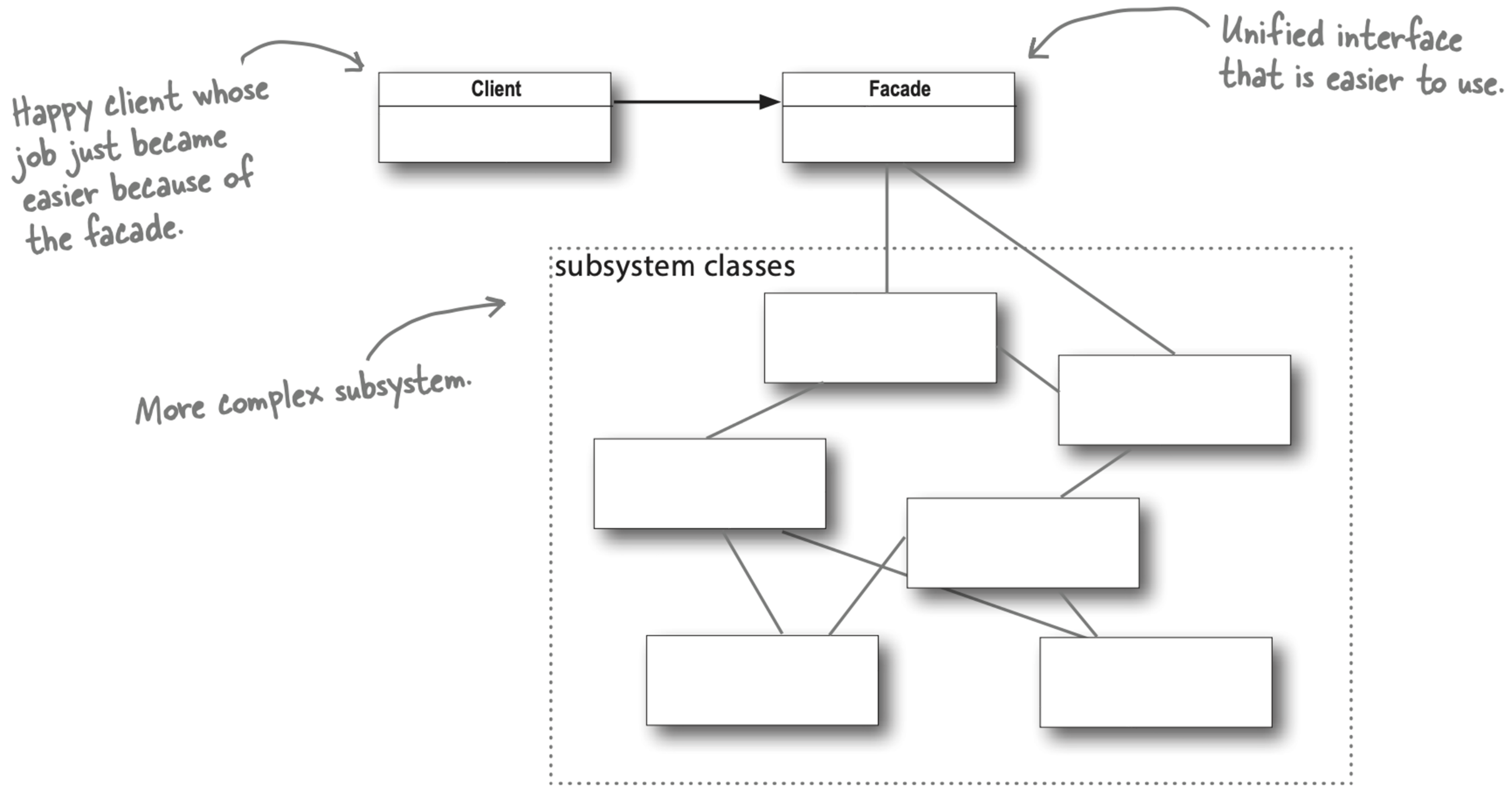
案例：上课与下课

- 上课：开门、开灯、开显示器、开电脑、开投影仪、开麦克风
- 下课：关门、关灯、光显示器、关电脑、关投影仪、关麦克风

上课

下课

为复杂系统提供一个简单的接口

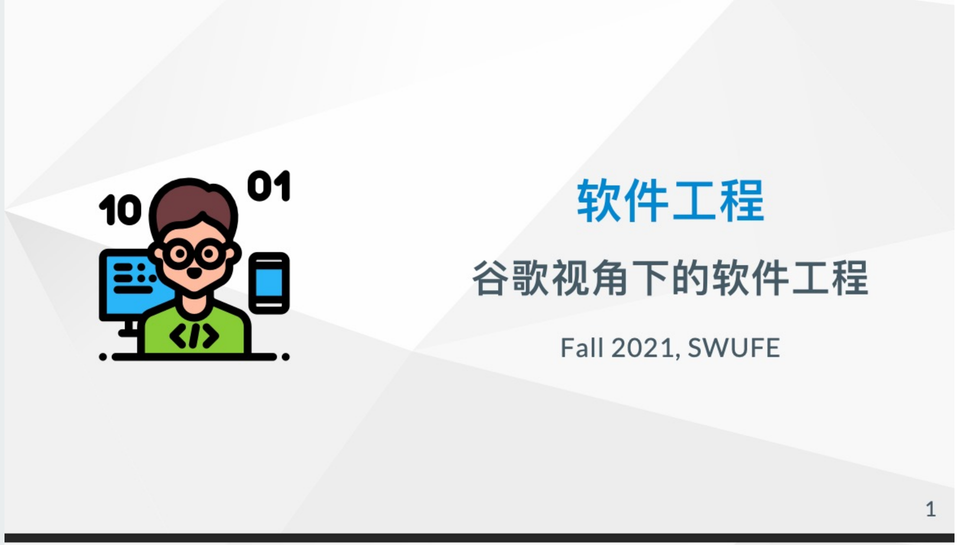
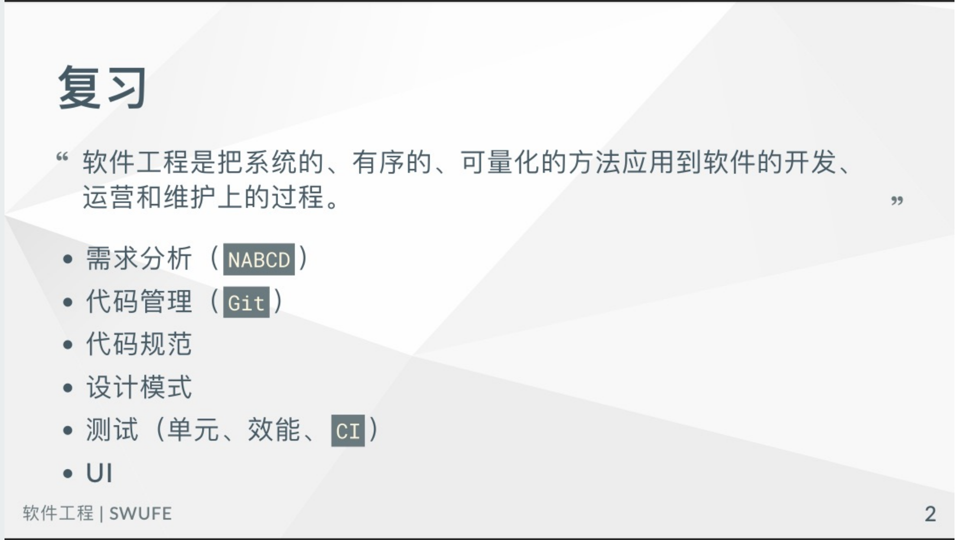

Happy client whose job just became easier because of the facade.
Client
Facade
Unified interface that is easier to use.
subsystem classes
More complex subsystem.

小结

- 装饰者模式
- 理解 Java IO 的设计
- 门面模式

3. Marp

使用Markdown写PPT

软件工程

谷歌视角下的软件工程

Fall 2021, SWUFE

1

复习

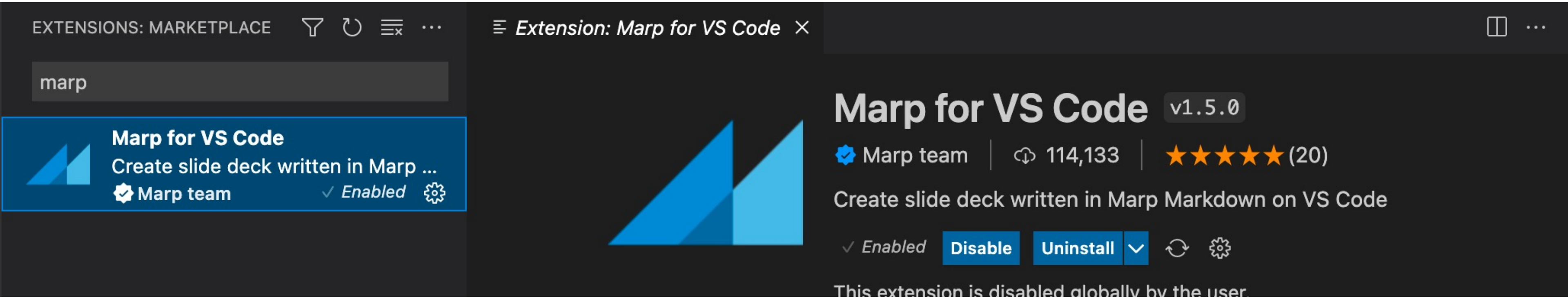
> 软件工程是把系统的、有序的、可量化的方法应用到软件的开发、运营和维护上的过程。

- 需求分析（`NABCD`）
- 代码管理（`Git`）
- 代码规范
- 设计模式
- 测试（单元、效能、`CI`）
- UI

软件工程 | SWUFE

2

2.1 Marp



EXTENSIONS: MARKETPLACE

marp

Marp for VS Code
Create slide deck written in Marp ...
Marp team ✓ Enabled

Extension: Marp for VS Code

Marp for VS Code v1.5.0

Marp team | 114,133 | ★★★★★ (20)

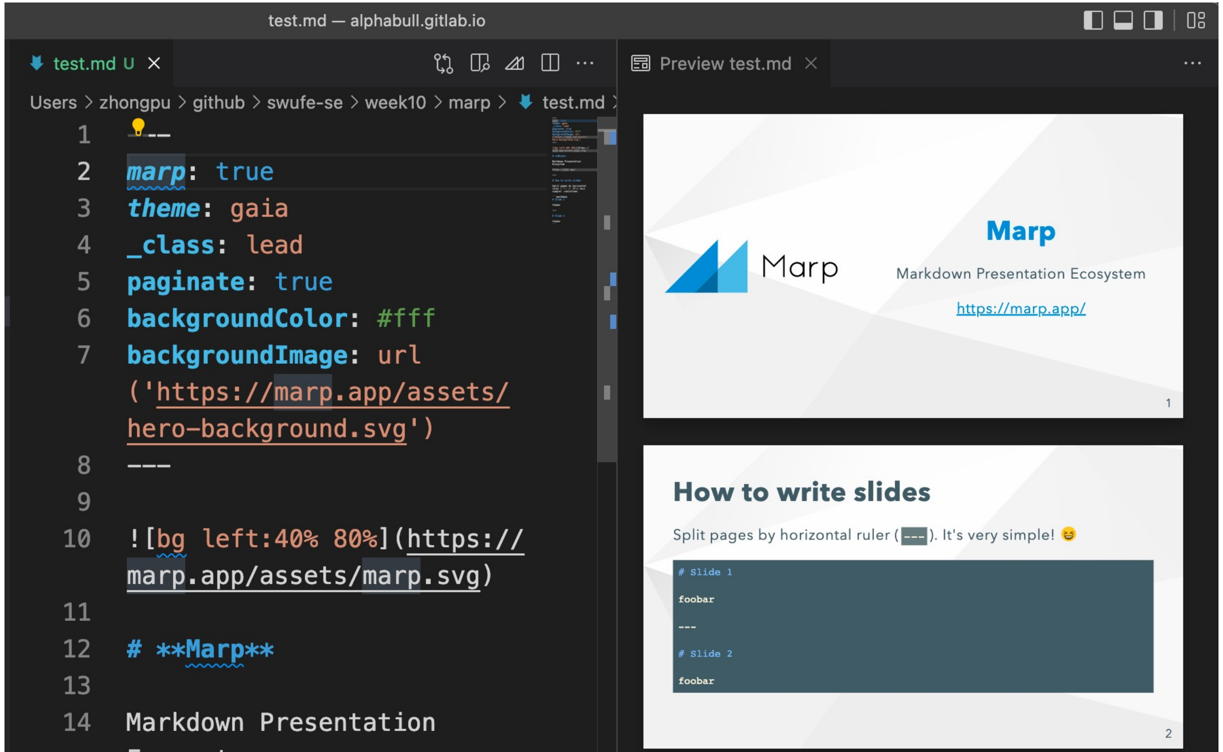
Create slide deck written in Marp Markdown on VS Code

✓ Enabled Disable Uninstall

This extension is disabled globally by the user.

2.2 练习

- 打开 https://marp.app/
- 复制Example代码


test.md — alphabull.gitlab.io
test.md U
Preview test.md
Users > zhongpu > github > swufe-se > week10 > marp > test.md

```
---
marp: true
theme: gaia
_class: lead
paginate: true
backgroundColor: #fff
backgroundImage: url('https://marp.app/assets/hero-background.svg')
---

![bg left:40% 80%](https://marp.app/assets/marp.svg)

# **Marp**

Markdown Presentation
```

Marp

Markdown Presentation Ecosystem

https://marp.app/

1

How to write slides

Split pages by horizontal ruler (---). It's very simple! 😆

```
# Slide 1

foobar

---

# Slide 2

foobar
```

2

个人作业

图文并茂地介绍仿 airdrop 软件，主要包括：

- 关键功能
- 关键代码
- 单元测试
- 使用教程

格式是重要评分依据。通过飞书文档提交。

Final Project

介绍小组的 final project，主要包括：

- 背景（重点解释为什么它是一个有需求的软件）
- 软件原型
- 亮点

（从16周开始，每组约10分钟。）