OCP – 开闭原则

开闭原则:

- 对扩展是开放的
- 对更改是封闭的

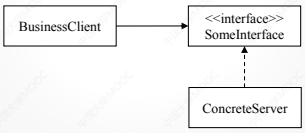
这意味着需求发生变化时，以对原有设计进行扩展方式应对，而不是以修改原有代码的形式应对。

OCP – 违反 OCP原则的例子

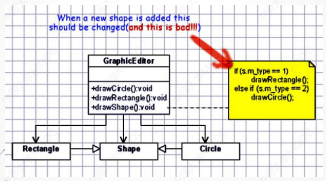
如果BusinessClient类具有对ConcreteServer类的引用，则替换ConcreteServer会导致修改BusinessClient。

OCP – 与 OCP原则一致

如果BusinessClient具有对接口的引用，则替换ConcreteServer不会导致BusinessClient的修改。



例子2



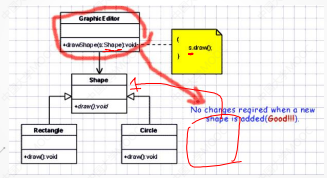
例子2

```
// Open-Close Principle - Bad example
class GraphicEditor {
        public void drawShape(Shape s) {
                if (s.m_type==1)
                drawRectangle(s);
                else if (s.m_type==2)
                        drawCircle(s);
        }
        public void drawCircle(Circle r) {....}
        public void drawRectangle(Rectangle r) {....}
}
```

例子2

```
class Shape {
        int m_type;
}
class Rectangle extends Shape {
        Rectangle() {
                super.m_type=1;
        }
}
class Circle extends Shape {
        Circle() {
                super.m_type=2;
        }
}
```

例子2



例子2

```
// Open-Close Principle - Good example
class GraphicEditor {
        public void drawShape(Shape s) {
        s.draw();
        }
}
class Shape {
   abstract void draw();
}
class Rectangle extends Shape {
   public void draw() {  // draw the rectangle  }
}
```

OCP

- **符合 OCP原则**
- **增加一个新的shape:**
  - 对扩展开放 – 为新的shape增加一个新的子类
  - 对更改封闭 – 无需修改 drawShape()
- **What about the smells?**
  - 僵硬性（Rigidity） – 不存在
  - 脆弱性（Fragility） – 不存在
  - 不透明性，不可重用性，不可移植性，.....没问题!
- **该代码针对这一特定需求变化封闭**

OCP – Shape 可能会有更多的问题

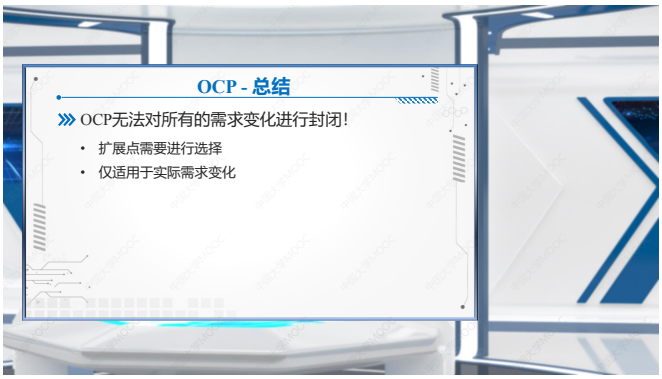
- 如果新要求声明shape必须按照某种排序顺序绘制，例如，所有圆形必须在所有方形之前画出：
- 无论一个模块如何 “封闭”，总是会有某种需求改变，它无法封闭。
- 设计师必须选择对哪个需求变化来封闭设计。 哪些变化更可能?
- 需求变化真实发生时，再做扩展性设计!

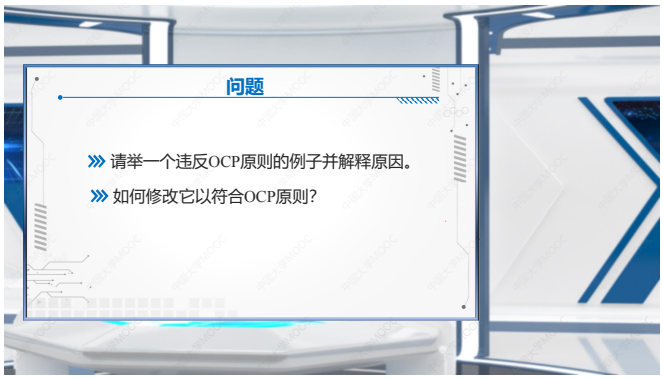
刺激变化

- 刺激变化
  - 先写测试。
  - 使用较短的开发周期
  - 先开发核心功能并经常向涉众展示这些功能
  - 首先开发最重要的功能。
  - 尽早且经常发布软件。

OCP - 总结

- **原则:**
  - 对扩展开放 - 添加新功能
  - 对修改封闭 - 现有代码未更改
- **实现:**
  - 对可能发生变化的部分进行抽象
  - 使用多态来添加不同的行为
  - 抽象也可以应用于非OO语言!


OCP - 总结
OCP无法对所有的需求变化进行封闭!
扩展点需要进行选择
仅适用于实际需求变化


问题
请举一个违反OCP原则的例子并解释原因。
如何修改它以符合OCP原则?