



# 软件工程

## 设计模式（2）

Spring 2022, SWUFE



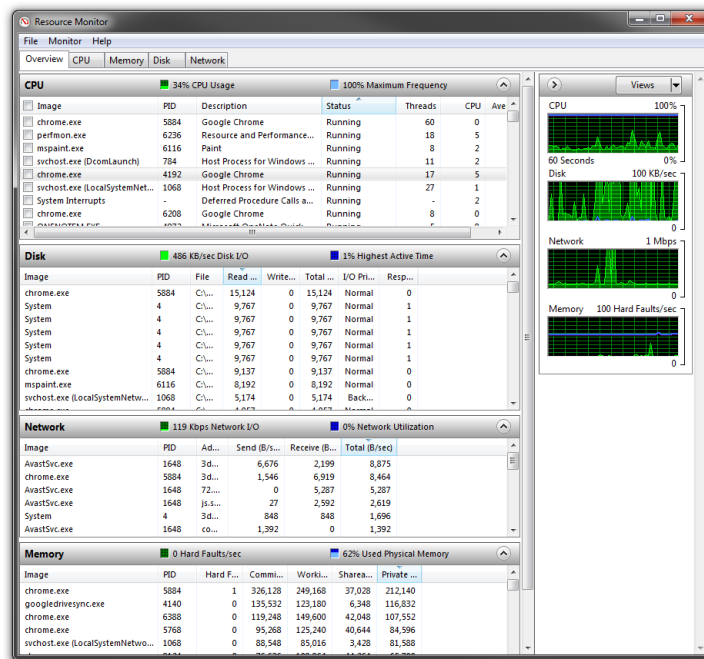
## 复习

**策略模式能让你定义一系列算法，并将每种算法分别放入独立的类中，以使算法的对象能够相互替换。**

- 策略模式
- Encapsulate what varies
- Favor composition over inheritance
- Program to interfaces, not implementations

# 1. 单例模式

Singleton Pattern



任务管理器只有一个！



## 1.1 简易版单例

问题：你是如何创建对象的？




```
MyObject object = new MyObject();
```



## 1.2 简易版单例

问题：如果其他人想再创建一个MyObject，是否可以再调用new？

问题：我们有个类，但不想被别人随便使用，该怎么办？




```
class MyObject {  
    private MyObject() { ... }  
}
```




## 1.2 简易版单例

问题：构造函数是private的，这就无法创建对象，有什么解决方案？



```
class MyClass {  
    private MyClass() { ... }  
  
    public static getInstance() { return new MyClass(); }  
}
```



```
class MyClass {  
    private static MyClass uniqueInstance;  
  
    private MyClass() { ... }  
  
    public static getInstance() {  
        if (uniqueInstance == null) {  
            uniqueInstance = new MyClass();  
        }  
        return uniqueInstance;  
    }  
}
```




## 1.3 多线程问题



```
public static getInstance() {  
  
    if (uniqueInstance == null) {  
        uniqueInstance = new MyClass();  
    }  
  
    return uniqueInstance;  
}
```





```
public class Singleton {  
    private volatile static Singleton uniqueInstance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (uniqueInstance == null) {  
            synchronized (Singleton.class) {  
                if (uniqueInstance == null) {  
                    uniqueInstance = new Singleton();  
                }  
            }  
        }  
        return uniqueInstance;  
    }  
}
```

一个安全、高效的单例模式



## Eager vs. lazy initialization



```
public static getInstance() {  
  
    if (uniqueInstance == null) {  
        uniqueInstance = new MyClass();  
    }  
  
    return uniqueInstance;  
}
```

还有种办法: Eager  
initialization



你觉得单例模式  
怎么实现更好？



## 1.4 最好的单例实现: enum



```
public enum Singleton {  
    UNIQUE_INSTANCE;  
  
    // other useful fields here  
  
    // other useful methods here  
    public String getDescription() {  
        return "I'm a thread safe Singleton!";  
    }  
}
```

---

与C++等语言中的枚举类型不同，  
Java中的enum具备“类”的功能！

<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>



## 1.5 更多资料

阅读 <https://refactoringguru.cn/design-patterns/singleton>

## 1.6 课堂练习

假设系统中需要一个 **ConfigManager**，记录着服务的host, port, password等关键信息。为了方便，上述信息可以记录在一个Map中。  
( 参考 <https://www.codiwan.com/singleton-design-pattern-real-world-example-java/> )



## 2. 静态工厂方法

它实际上不属于“设计模式”，但是应用广泛。

```
Boolean b = Boolean.valueOf(false);  
Calendar calendar = Calendar.getInstance();
```

Static Factory Method



## 2.1 静态工厂方法的好处

不像构造函数，它们有名字。

```
List<String> list = Arrays.asList("1", "2", "3");
```

不像构造函数，它们不是每次调用都创建对象。

- `Boolean b = Boolean.valueOf(false);`
- 单例模式



## 案例

```
class RandomIntGenerator {  
    private int min;  
    private int max;  
  
    public RandomIntGenerator(int min, int max) {  
  
    }  
  
    public int next() {  
        return 42;  
    }  
}
```

## 思考

```
public class User {  
  
    private static final Logger LOGGER = Logger.getLogger(User.class.getName());  
    private final String name;  
    private final String email;  
    private final String country;  
  
    // standard constructors / getters  
  
    public static User createWithLoggedInstantiationTime(  
        String name, String email, String country) {  
        LOGGER.log(Level.INFO, "Creating User instance at : {0}", LocalTime.now());  
        return new User(name, email, country);  
    }  
}
```

上面静态工厂方法的好处是什么？

<https://www.baeldung.com/java-constructors-vs-static-factory-methods>



## 2.2 静态工厂方法的缺点

用户可能不知道具体的名字。

- of
- valueOf
- from
- instance/getInstance
- getXXX
- newXXX
- XXX

### 3. 工厂方法模式

Factory Method Pattern



```
MyObject object = new MyObject();
```

“new”意味着只能返回一个**具体**的对象。



## 3.1 第0个版本

```
Pizza orderPizza() {  
    Pizza pizza = new Pizza();  
  
    pizza.prepare();  
    pizza.bake();  
    pizza.cut();  
    pizza.box();  
    return pizza;  
}
```

```
Pizza orderPizza(String type) {  
    Pizza pizza;
```

What  
the  
order

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

```
    pizza.prepare();  
    pizza.bake();  
    pizza.cut();  
    pizza.box();  
    return pizza;  
}
```

Once we have  
(you know, rolled  
sauce, and add  
bake it, cut it

Each Pizza subclass  
GreekPizza, etc.  
itself.

## 3.1 第0个版本

```
Pizza orderPizza(String type) {
```

```
    Pizza pizza;
```

```
    pizza.prepare();
```

```
    pizza.bake();
```

```
    pizza.cut();
```

```
    pizza.box();
```

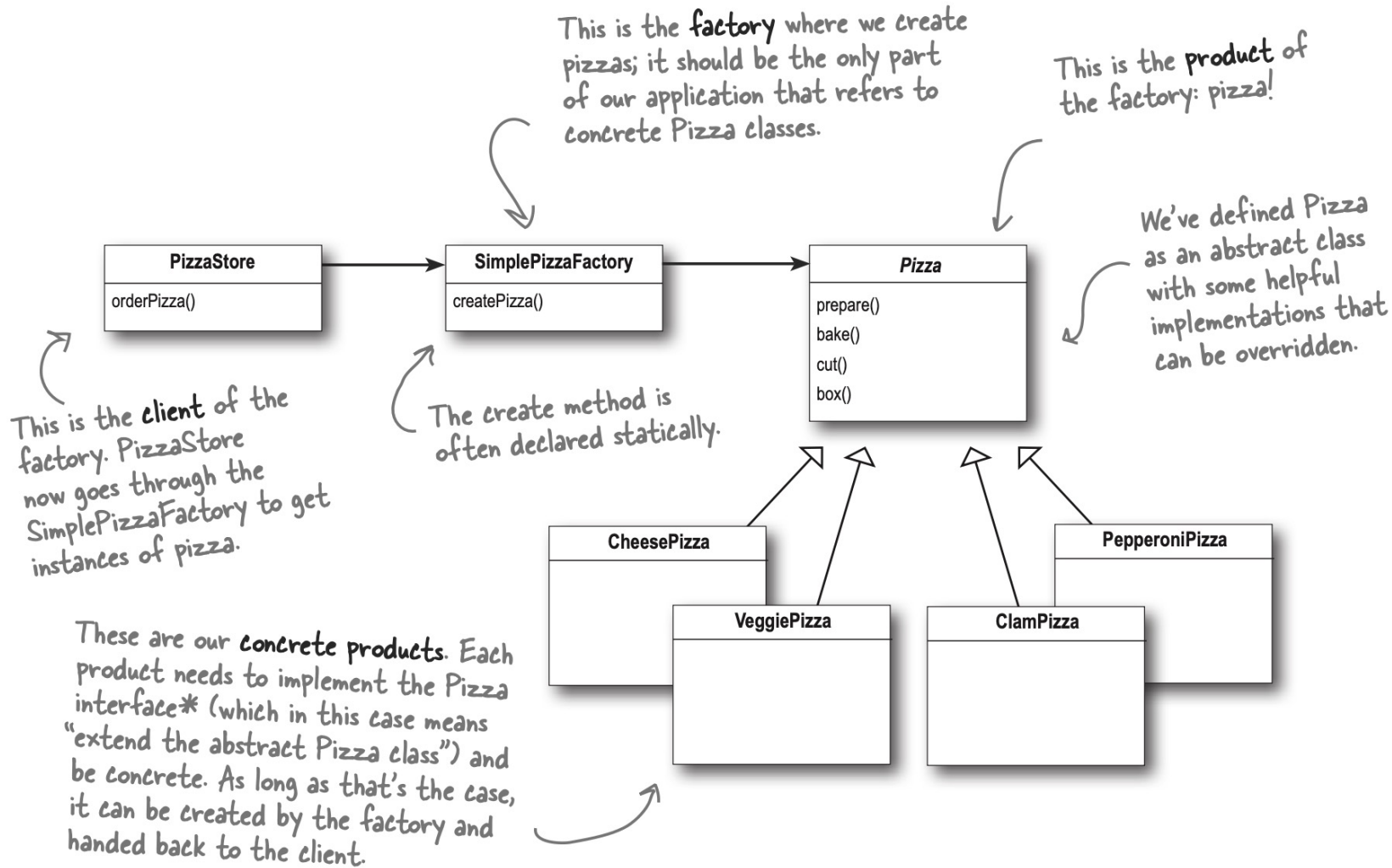
```
    return pizza;
```

```
}
```

*What's going on?*

Encapsulate what varies.





## 3.2 改进

目前只是 product 有多个种类，但 factory 本身可能也有多个种类。

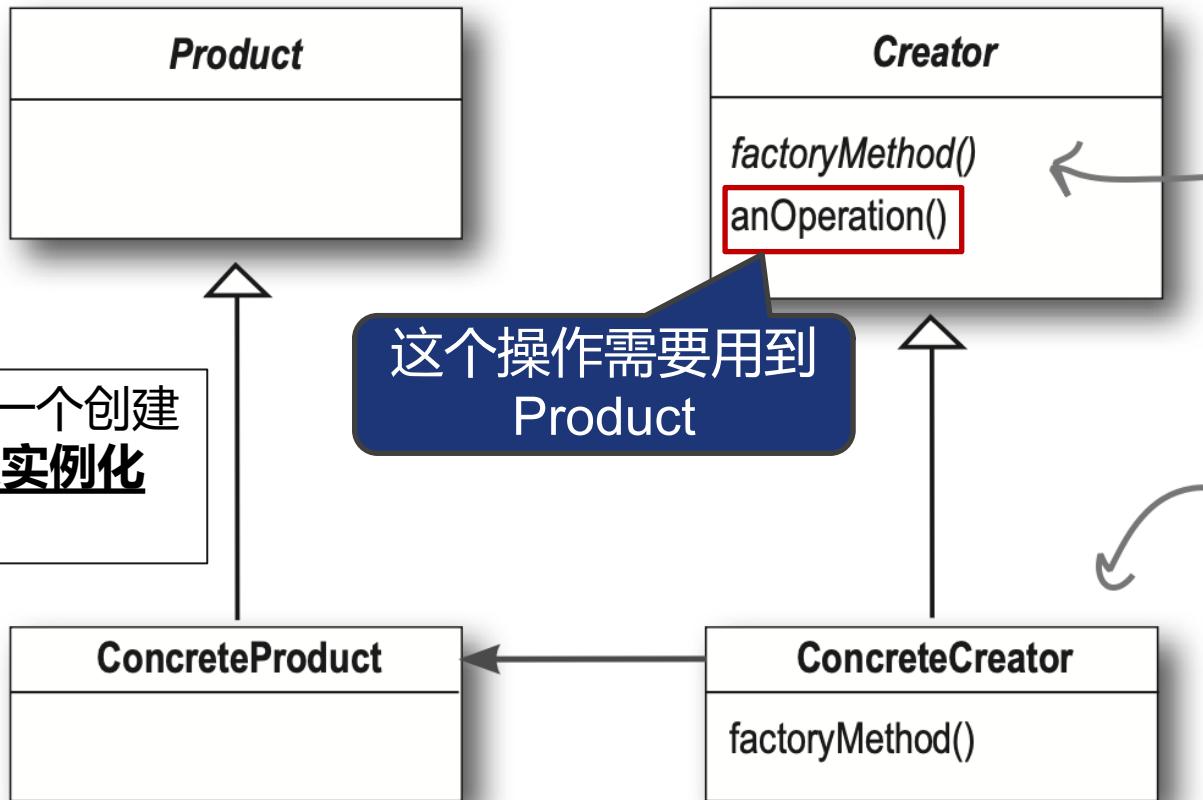


重庆	成都
鸳鸯火锅	鸳鸯火锅
毛肚火锅	毛肚火锅
羊肉火锅	羊肉火锅



## 工厂方法模式

工厂方法模式在父类中提供一个创建对象的方法，**允许子类决定实例化对象的类型。**



## 如果没有Factory Method Pattern?

```
public HotSpot createHotSpot(String style, String type) {  
    HotSpot spot = null;  
  
    if (style.equals("CQ")) {  
        if (type.equals("A")) {  
            spot = new CQAHotSpot();  
        } else if (type.equals("B")) {  
            spot = new CQBHotSpot();  
        }  
    } else if (style.equals("CD")) {  
        if (type.equals("A")) {  
            spot = new CDAHotSpot();  
        } else if (type.equals("B")) {  
            spot = new CDBHotSpot();  
        }  
    }  
    return spot;  
}
```

一个类依赖了过多的类；并且它依赖的是具体的类。

这是不好的设计！

Depend upon abstractions. Do not depend upon concrete classes.



## 工厂方法模式

```
HotSpotFactory factory;  
public HotSpot createHotSpot(String type) {  
    HotSpot spot = factory.createHotSpot(type);  
    return spot;  
}
```

Dependency Inversion Principle: Depend upon abstractions. Do not depend upon concrete classes.

### 3.3 思考

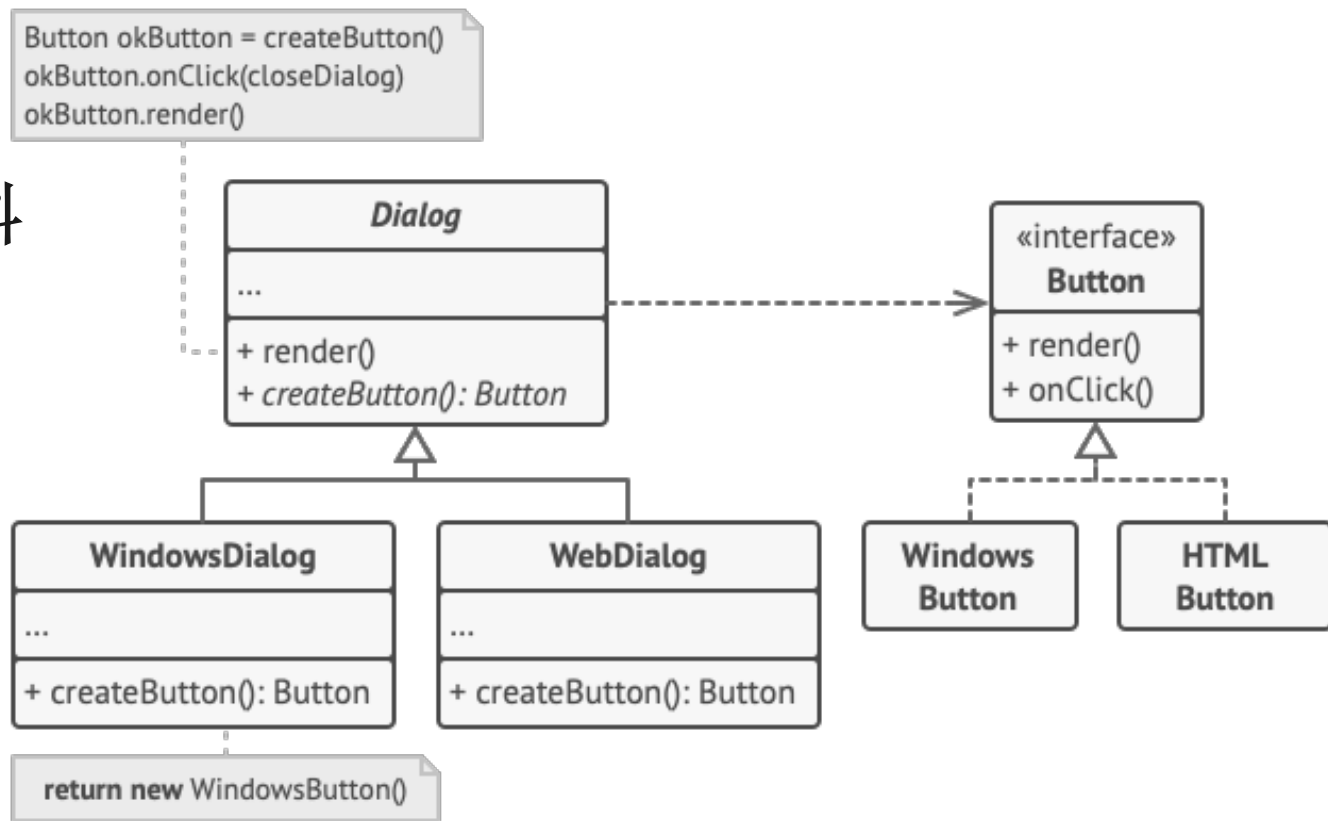


重庆	成都
重庆火锅	成都火锅

```
public HotSpot createHotSpot(String style) {  
    if (style.equals("CQ")) {  
        return new CQHotSpot();  
    } else if (style.equals("CD")) {  
        return new CDHotSpot();  
    }  
    return null;  
}
```

上面的代码是否属于 Factory Method Pattern ?

## 3.4 更多资料



阅读 <https://refactoringguru.cn/design-patterns/factory-method>



## Homework 7

- 使用工厂方法模式实现一个火锅订单系统。（ 10分 ）



## 小结

- 单例模式
- 静态工厂方法
- 工厂方法模式
- **依赖反转原则**