装饰(Decorator)模式

波波老师~研发总监/资深架构师

波波微课

spring2go.com

问题~蛋糕店建模问题

主营

搭配

- 继承方式
  - CakeWithCreamAndCherry
  - CakeWithCreamAndCherryAndScent
  - CakeWithCreamAndCherryAndScentAndNameCard
  - CakeWithCherryOnly
  - PastryOnly
  - PastryWithCreamAndCherry
  - PastryWithCreamAndCherryAndScent
  - PastryWithCreamAndCherryAndScentAndNameCard
  - PastryWithCherryOnly

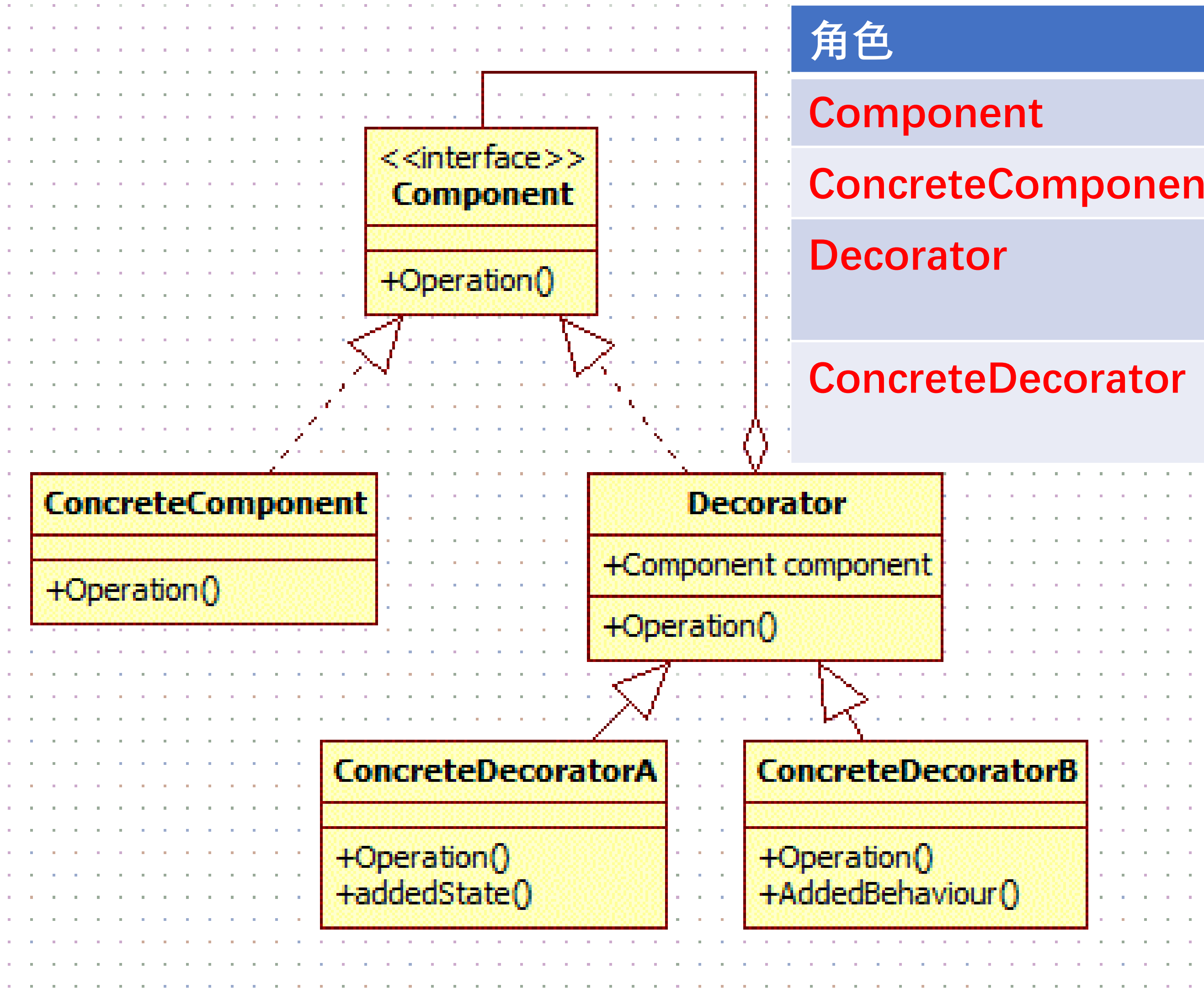
设计原理

- **组合**over**继承**（composite over inheritance）
  - 继承~编译时扩展
  - 组合~运行时扩展
- 开放封闭原理（open closed principle）

定义

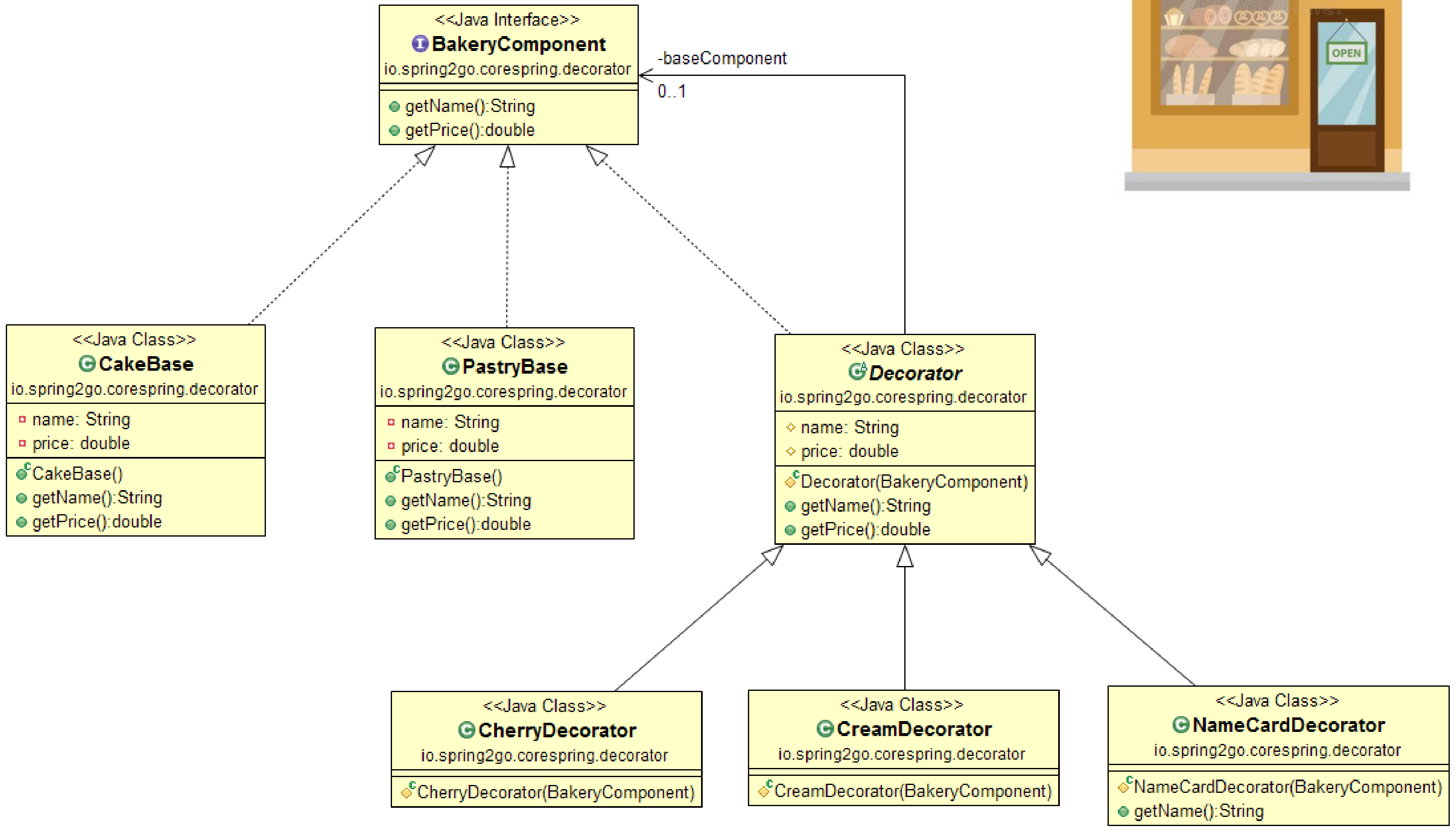
- 为对象动态添加额外的功能
- 在子类继承之外，提供一种新的扩展功能的方式
- 也称Wrapper
- 适用场合：跨横切面功能(AOP)
    - 安全认证授权
    - 日志
    - 缓存Caching
    - 校验
    - 异常处理

关系图



角色	职责
Component	定义待装饰对象的公共接口
ConcreteComponent	待装饰的实际对象
Decorator	定义所有可动态添加功能的公共接口，引用Component对象
ConcreteDecorator	所有可以添加额外功能到ConcreteComponent上的功能类。

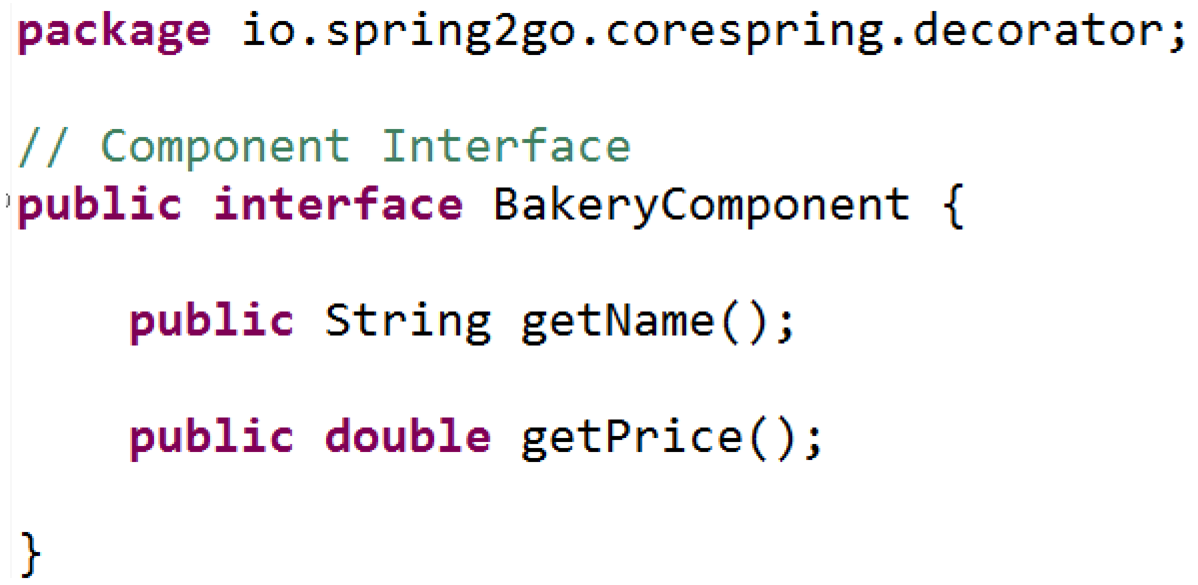
案例~蛋糕店



<<Java Interface>>
BakeryComponent
io.spring2go.corespring.decorator
getName():String
getPrice():double
-baseComponent
0..1
<<Java Class>>
CakeBase
io.spring2go.corespring.decorator
name: String
price: double
CakeBase()
getName():String
getPrice():double
<<Java Class>>
PastryBase
io.spring2go.corespring.decorator
name: String
price: double
PastryBase()
getName():String
getPrice():double
<<Java Class>>
Decorator
io.spring2go.corespring.decorator
name: String
price: double
Decorator(BakeryComponent)
getName():String
getPrice():double
<<Java Class>>
CherryDecorator
io.spring2go.corespring.decorator
CherryDecorator(BakeryComponent)
<<Java Class>>
CreamDecorator
io.spring2go.corespring.decorator
CreamDecorator(BakeryComponent)
<<Java Class>>
NameCardDecorator
io.spring2go.corespring.decorator
NameCardDecorator(BakeryComponent)
getName():String

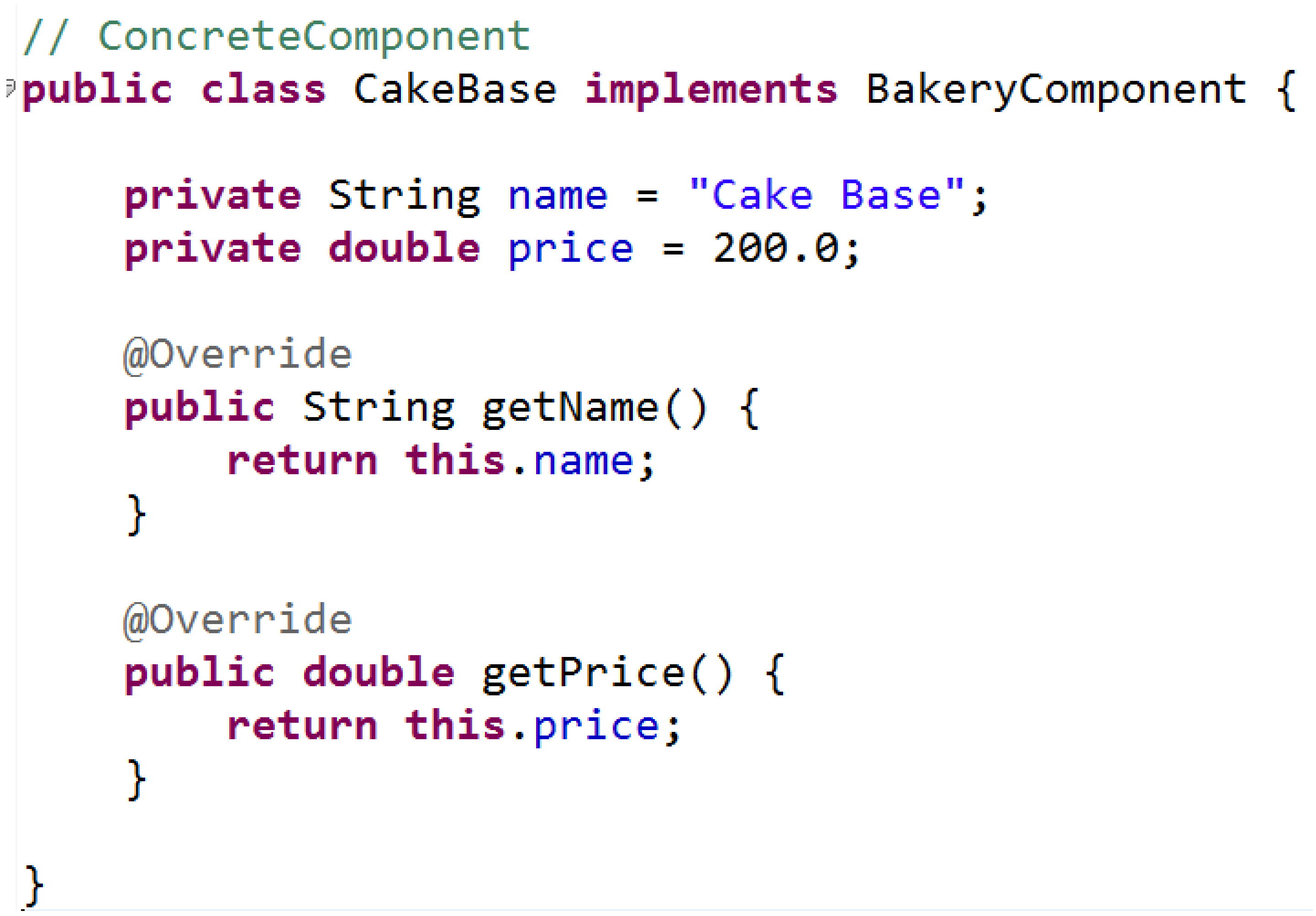
BAKERY SHOP
OPEN

实现~Component接口



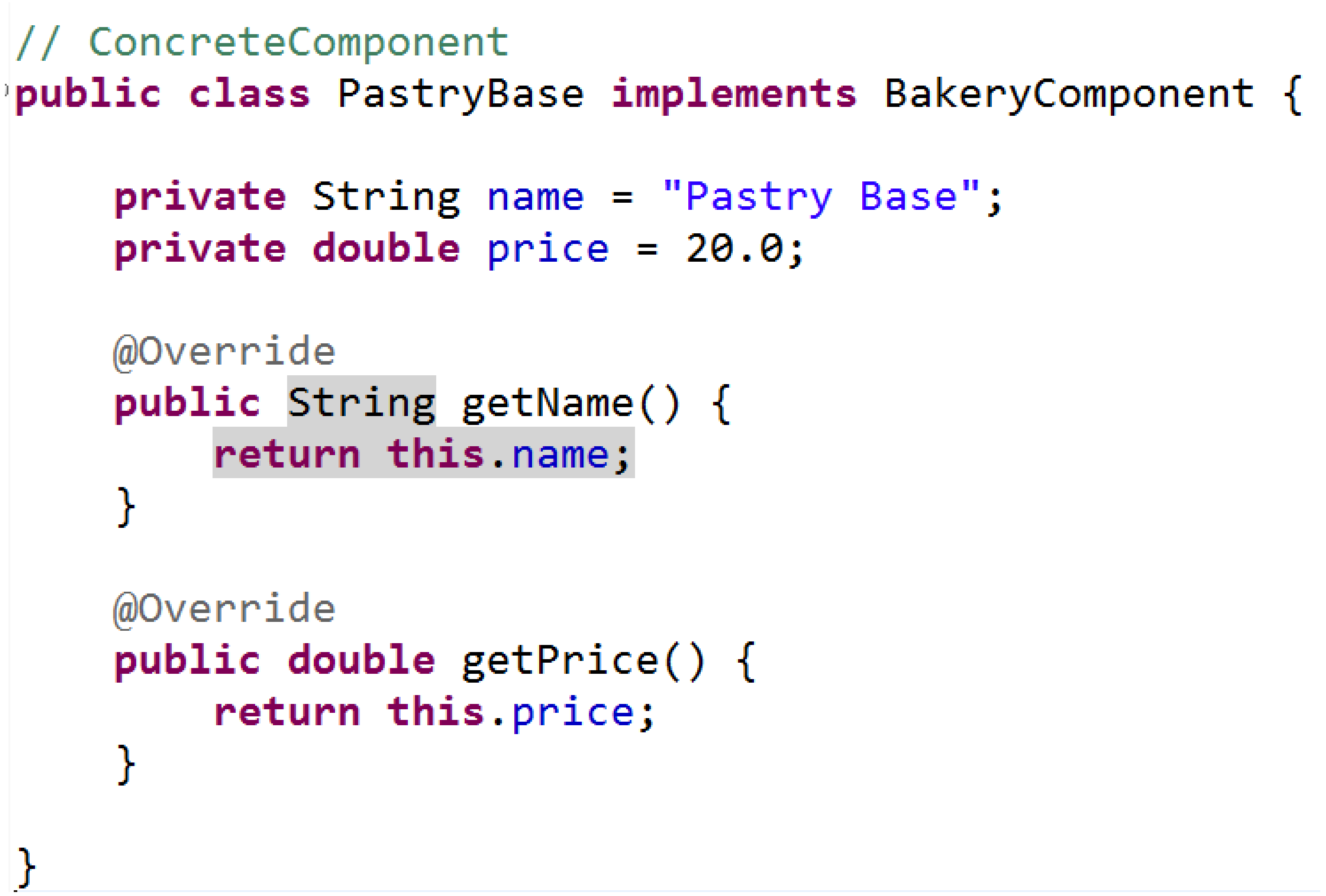
```java
package io.spring2go.corespring.decorator;

// Component Interface
public interface BakeryComponent {

    public String getName();

    public double getPrice();

}
```

实现~CakeBase



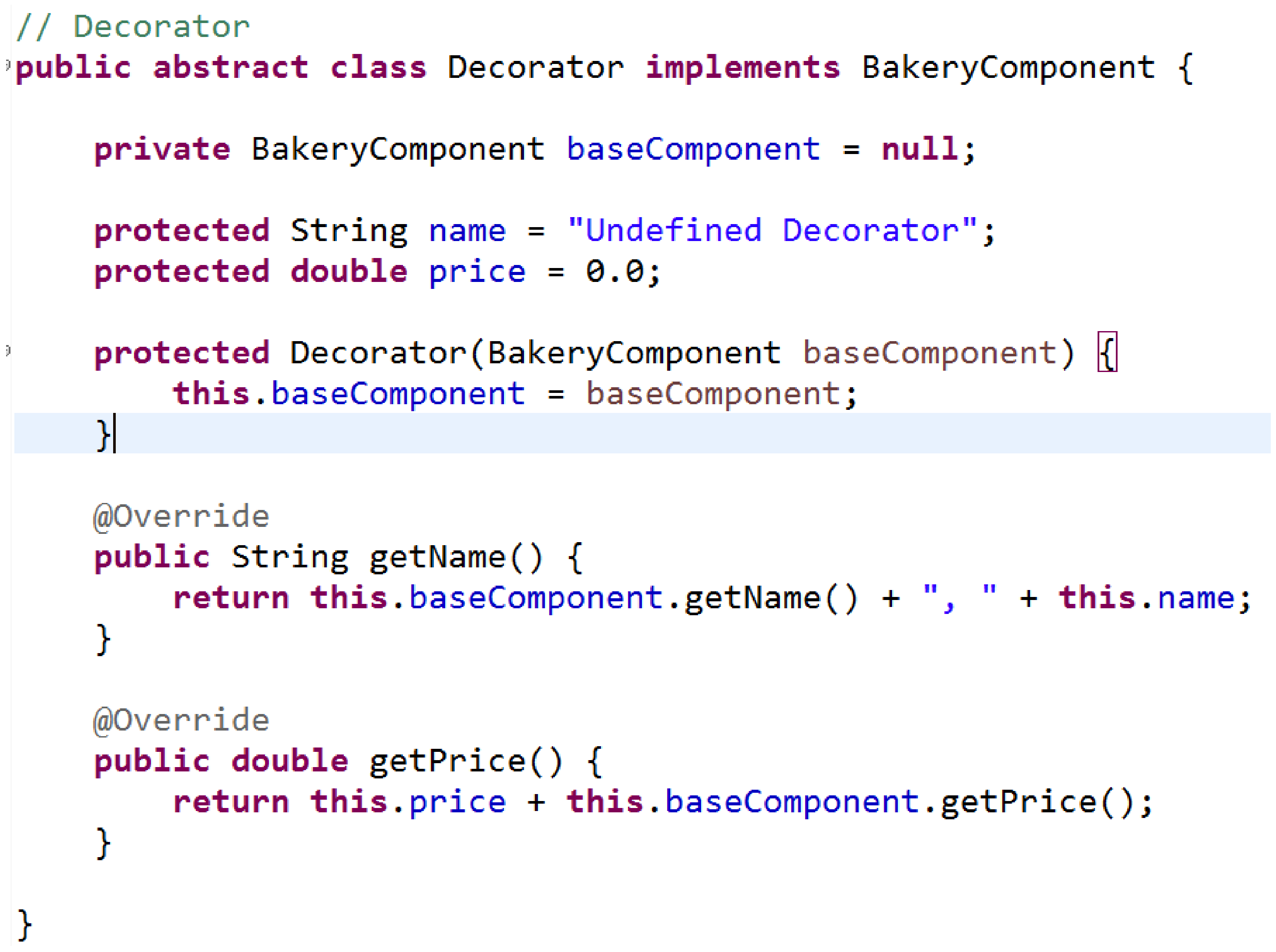
```java
// ConcreteComponent
public class CakeBase implements BakeryComponent {

    private String name = "Cake Base";
    private double price = 200.0;

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public double getPrice() {
        return this.price;
    }

}
```

实现~PastryBase



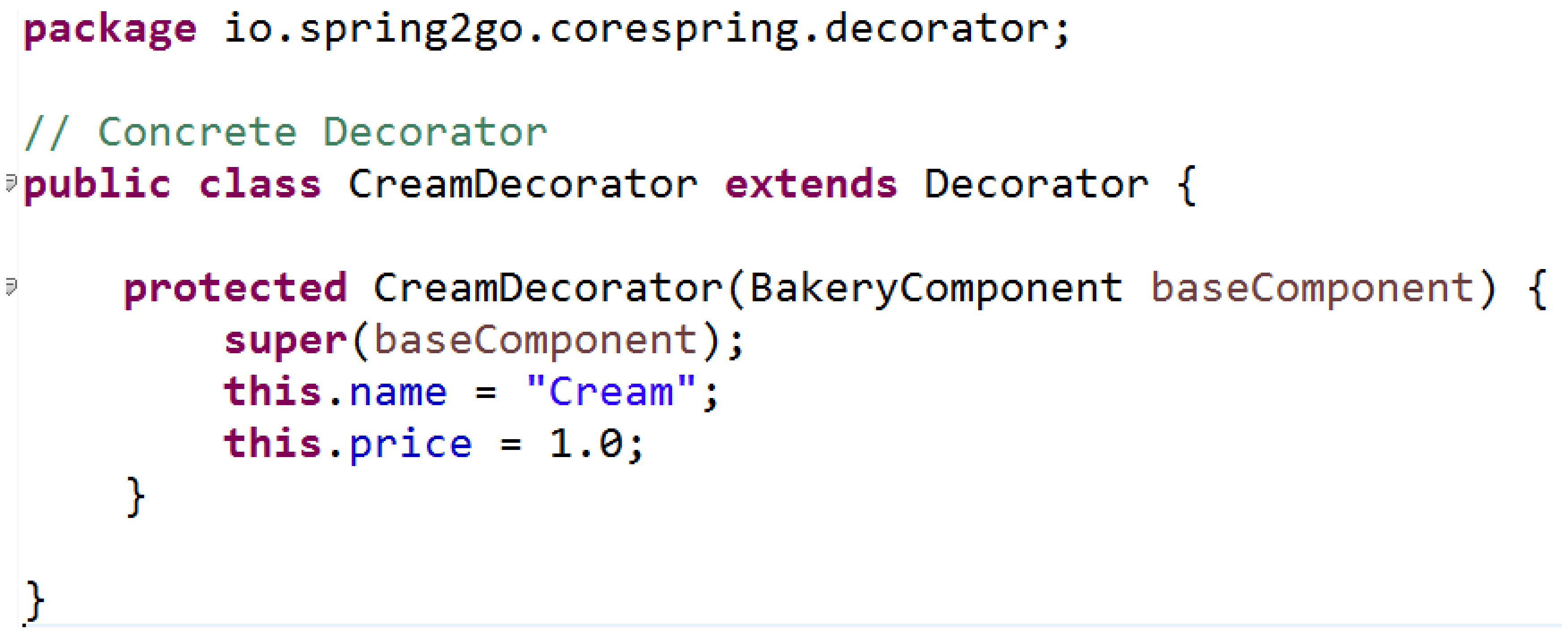
```java
// ConcreteComponent
public class PastryBase implements BakeryComponent {

    private String name = "Pastry Base";
    private double price = 20.0;

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public double getPrice() {
        return this.price;
    }

}
```

实现~Decorator



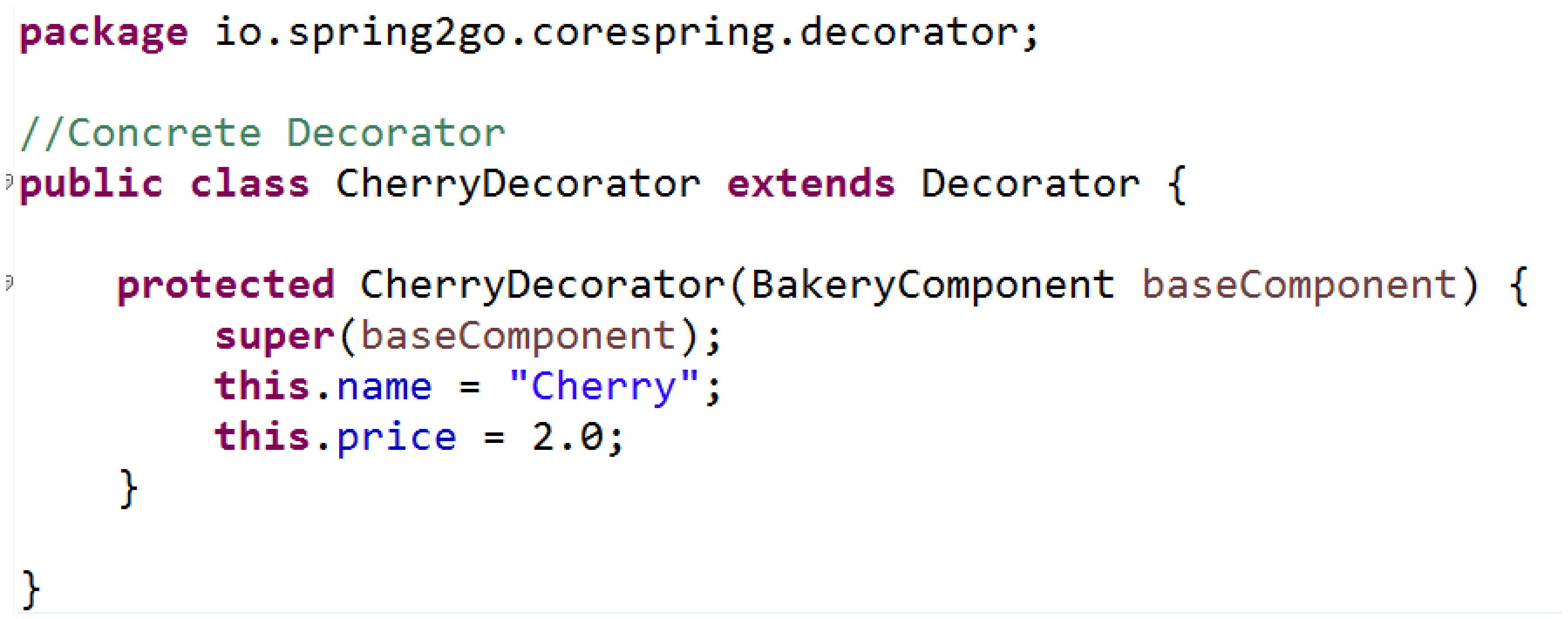
```java
// Decorator
public abstract class Decorator implements BakeryComponent {

    private BakeryComponent baseComponent = null;

    protected String name = "Undefined Decorator";
    protected double price = 0.0;

    protected Decorator(BakeryComponent baseComponent) {
        this.baseComponent = baseComponent;
    }

    @Override
    public String getName() {
        return this.baseComponent.getName() + ", " + this.name;
    }

    @Override
    public double getPrice() {
        return this.price + this.baseComponent.getPrice();
    }

}
```

实现~CreamDecorator



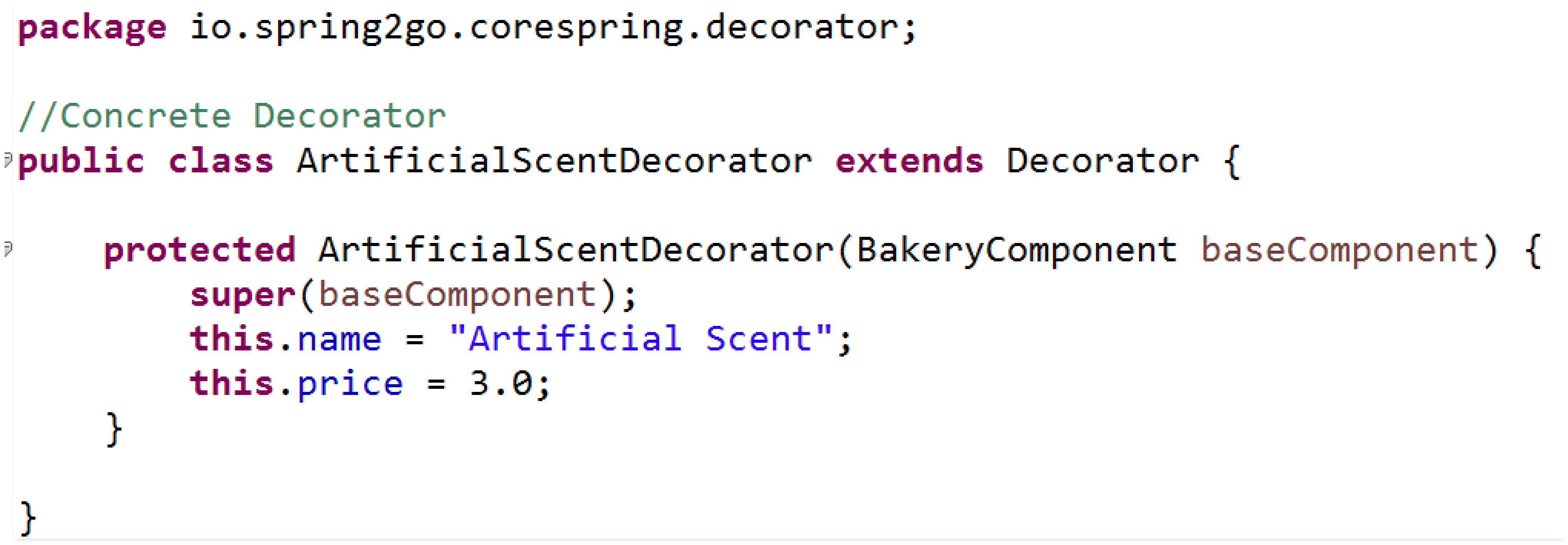
```java
package io.spring2go.corespring.decorator;

// Concrete Decorator
public class CreamDecorator extends Decorator {

    protected CreamDecorator(BakeryComponent baseComponent) {
        super(baseComponent);
        this.name = "Cream";
        this.price = 1.0;
    }

}
```

实现~CherryDecorator



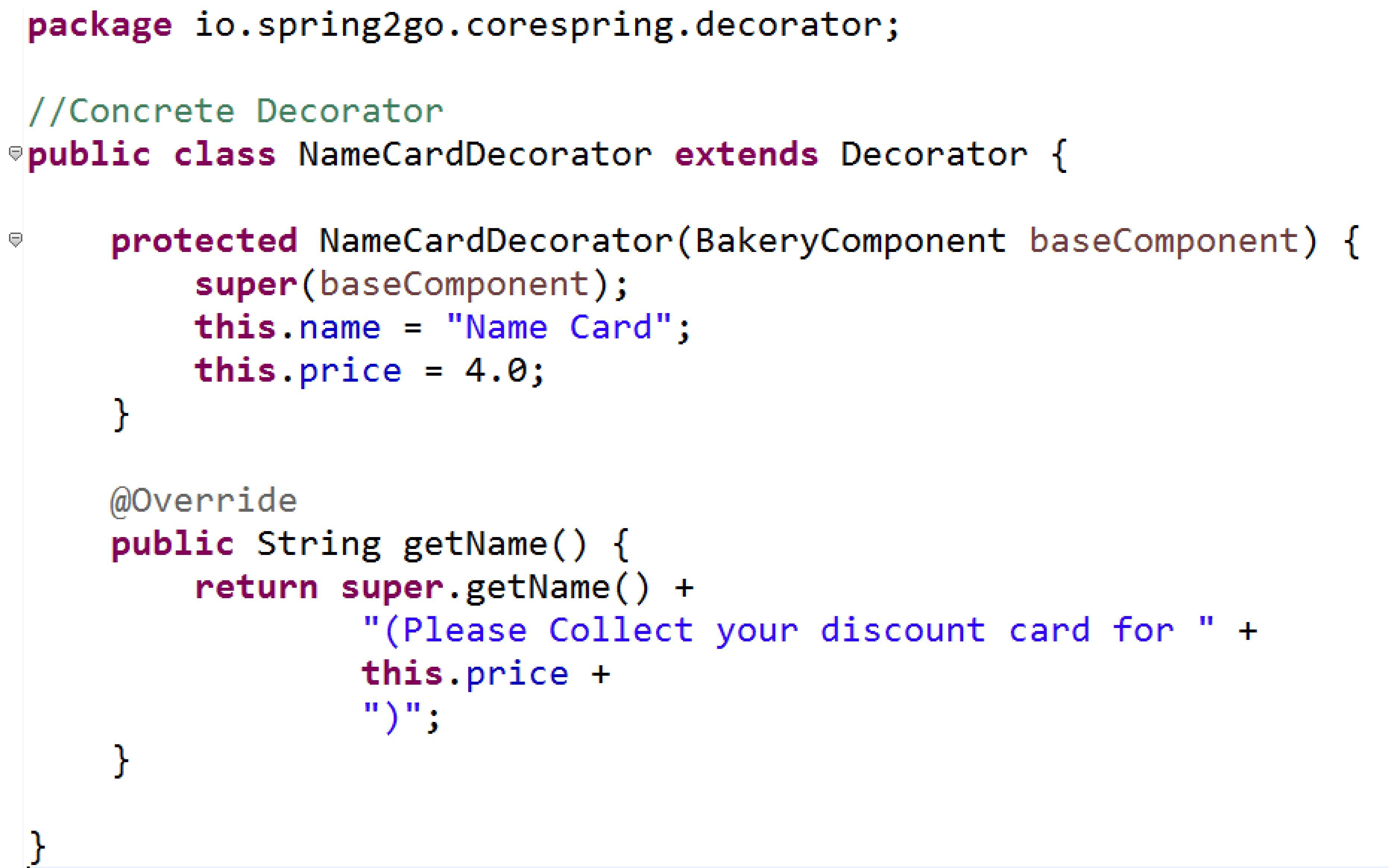
```java
package io.spring2go.corespring.decorator;

//Concrete Decorator
public class CherryDecorator extends Decorator {

    protected CherryDecorator(BakeryComponent baseComponent) {
        super(baseComponent);
        this.name = "Cherry";
        this.price = 2.0;
    }

}
```

实现~ArtificialScentDecorator



```java
package io.spring2go.corespring.decorator;

//Concrete Decorator
public class ArtificialScentDecorator extends Decorator {

    protected ArtificialScentDecorator(BakeryComponent baseComponent) {
        super(baseComponent);
        this.name = "Artificial Scent";
        this.price = 3.0;
    }

}
```

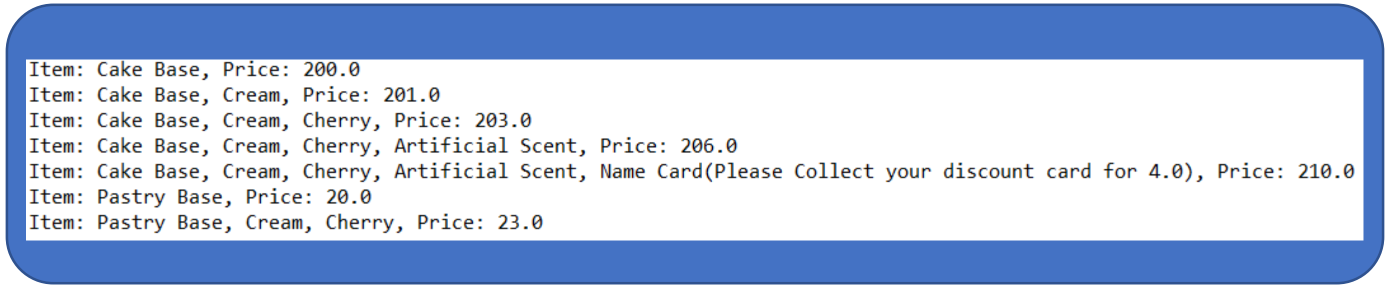
实现~NameCardDecorator



```java
package io.spring2go.corespring.decorator;

//Concrete Decorator
public class NameCardDecorator extends Decorator {

    protected NameCardDecorator(BakeryComponent baseComponent) {
        super(baseComponent);
        this.name = "Name Card";
        this.price = 4.0;
    }

    @Override
    public String getName() {
        return super.getName() +
                "(Please Collect your discount card for " +
                this.price +
                ")";
    }

}
```

实现~客户端

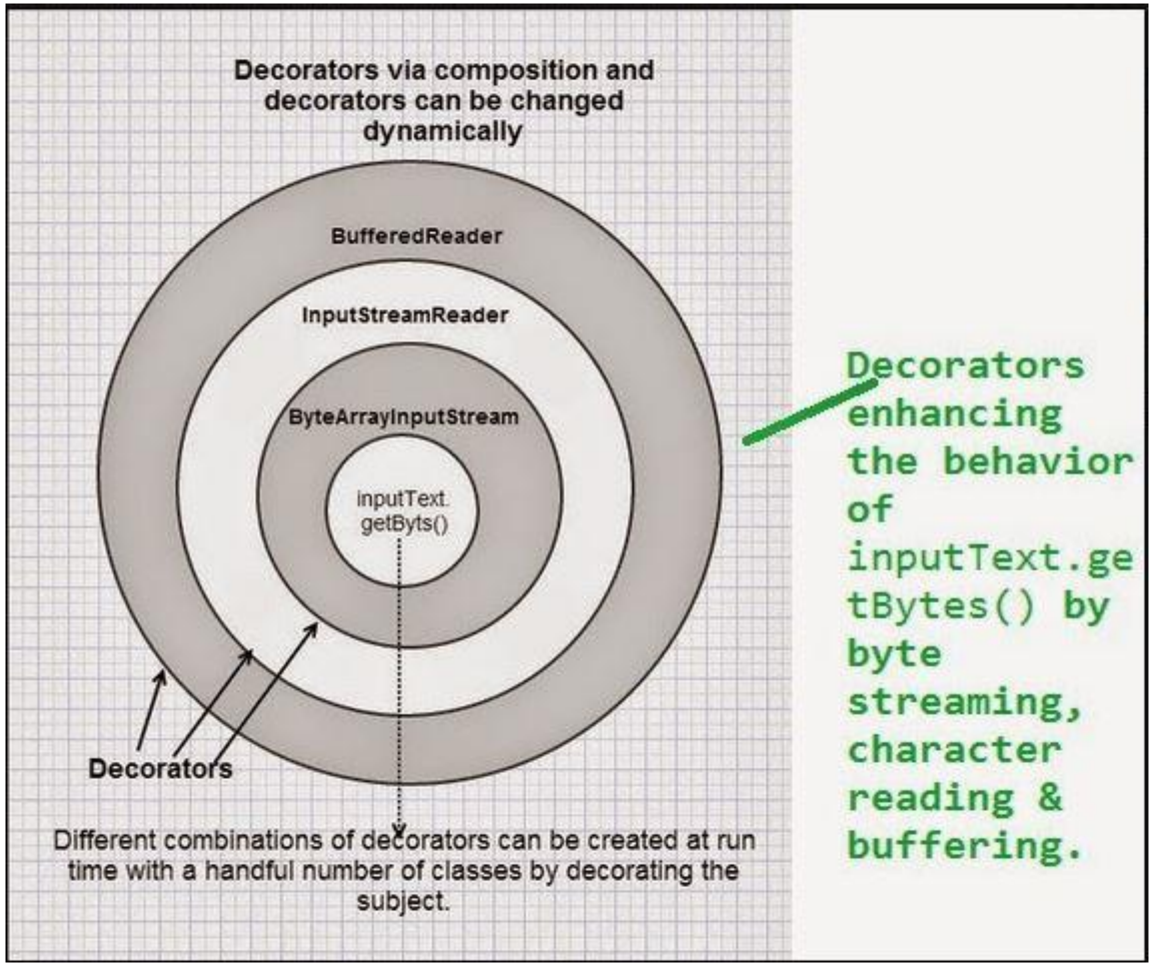
```java
package io.spring2go.corespring.decorator;

public class DecoratorMain {

    public static void main(String[] args) {
        // 先创建一个简单的Cake Base
        CakeBase cBase = new CakeBase();
        Util.printProductDetails(cBase);

        // 在蛋糕上添加奶油
        CreamDecorator creamCake = new CreamDecorator(cBase);
        Util.printProductDetails(creamCake);

        // 在蛋糕上添加樱桃
        CherryDecorator cherryCake = new CherryDecorator(creamCake);
        Util.printProductDetails(cherryCake);

        // 再添加香味
        ArtificialScentDecorator scentedCake =
                new ArtificialScentDecorator(cherryCake);
        Util.printProductDetails(scentedCake);

        // 最后在蛋糕上添加名片
        NameCardDecorator nameCardOnCake = new NameCardDecorator(scentedCake);
        Util.printProductDetails(nameCardOnCake);

        // 现在创建一个简单的糕点
        PastryBase pastry = new PastryBase();
        Util.printProductDetails(pastry);

        // 在糕点上只添加奶油和樱桃
        CreamDecorator creamPastry = new CreamDecorator(pastry);
        CherryDecorator cherryPastry = new CherryDecorator(creamPastry);
        Util.printProductDetails(cherryPastry);
    }

}
```



```
Item: Cake Base, Price: 200.0
Item: Cake Base, Cream, Price: 201.0
Item: Cake Base, Cream, Cherry, Price: 203.0
Item: Cake Base, Cream, Cherry, Artificial Scent, Price: 206.0
Item: Cake Base, Cream, Cherry, Artificial Scent, Name Card(Please Collect your discount card for 4.0), Price: 210.0
Item: Pastry Base, Price: 20.0
Item: Pastry Base, Cream, Cherry, Price: 23.0
```

应用

- Java IO library
- sitemesh


Decorators via composition and decorators can be changed dynamically
BufferedReader
InputStreamReader
ByteArrayInputStream
inputText.getByts()
Decorators
Different combinations of decorators can be created at run time with a handful number of classes by decorating the subject.
Decorators enhancing the behavior of inputText.getBytes() by byte streaming, character reading & buffering.

优劣

- 优点
  - 运行时扩展行为更灵活
  - 任意扩展decorator
  - 扩展不影响现有对象
- 不足
  - 产生大量类似decorator对象

问题

- 装饰模式和适配器模式的差异？
- 装饰模式和子类继承的差异？

参考

- Decorator Design Pattern
  - https://java2blog.com/decorator-design-pattern/
- Understanding and Implementing Decorator Pattern in C#
  - https://www.codeproject.com/Articles/479635/UnderstandingplusandplusImplementingplusDecoratorp

代码

- https://github.com/spring2go/core-spring-patterns

波波微课
spring2go.com