河北师范大学软件学院
Software College of Hebei Normal University

第六章 装饰模式

任课教师：武永亮
wuyongliang@edu2act.org

上节回顾

- 表示对象的部分-整体层次结构
- 用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

课程内容

- 环境及问题
- 装饰模式详解
- 装饰模式实现
- 扩展练习

课程内容

- 环境及问题
- 装饰模式详解
- 装饰模式实现
- 扩展练习

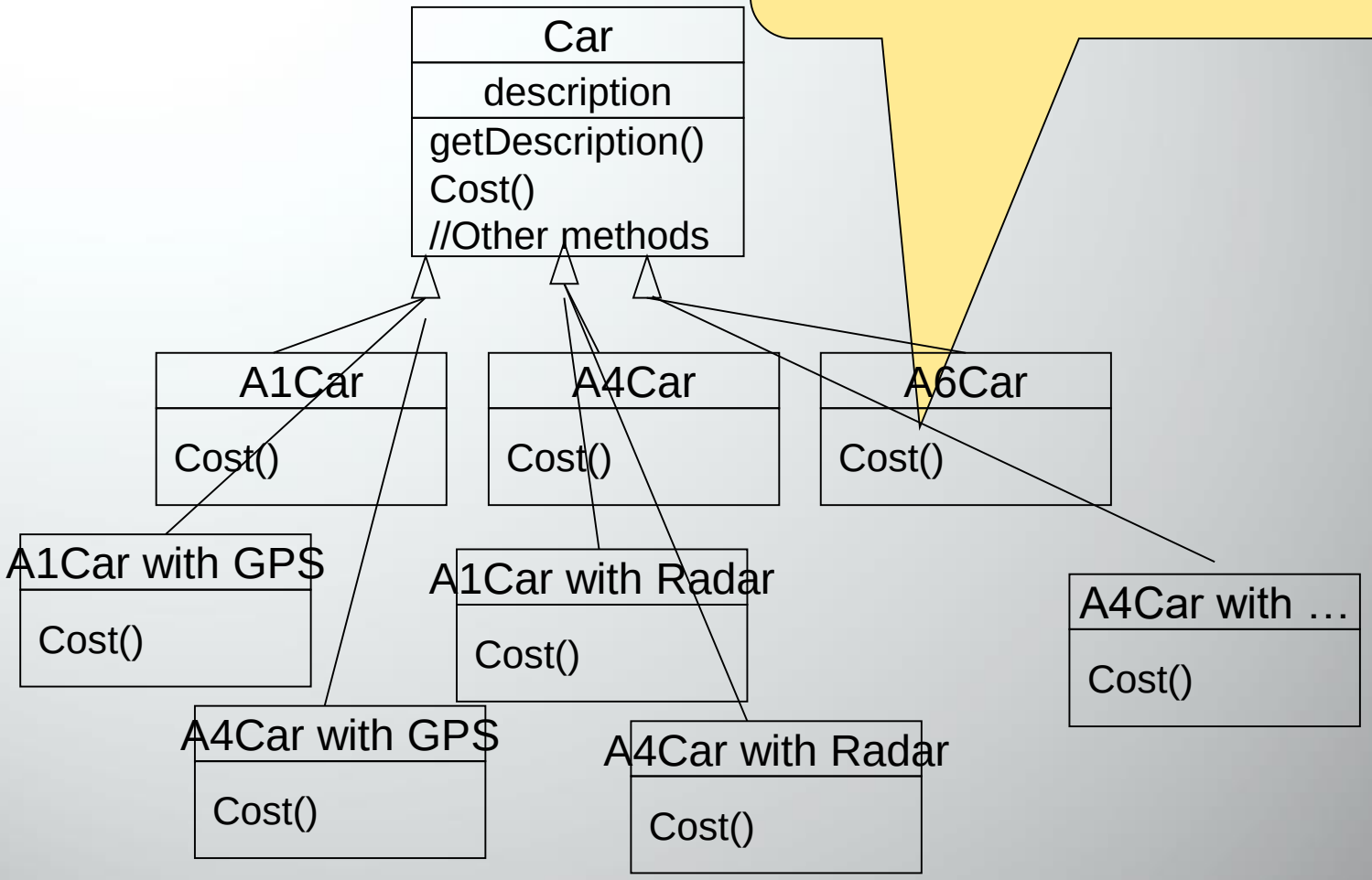
环境

- 汽车4s店汽车销售
- 出售奥迪A1，A4，A6；
- 可装饰的组件倒车雷达，真皮座椅，gps定位
- 用户可以挑选汽车的型号，加任意的组件

请利用15分钟时间对该系统进行设计

环境

每个cost()方法计算出每种汽车的价格。



问题

- 不必改变原类文件和使用继承的情况下，动态的扩展一个对象的的功能。
- 应用程序的可维护性，可扩展性差。

装饰模式（ Decorator ）

课程内容

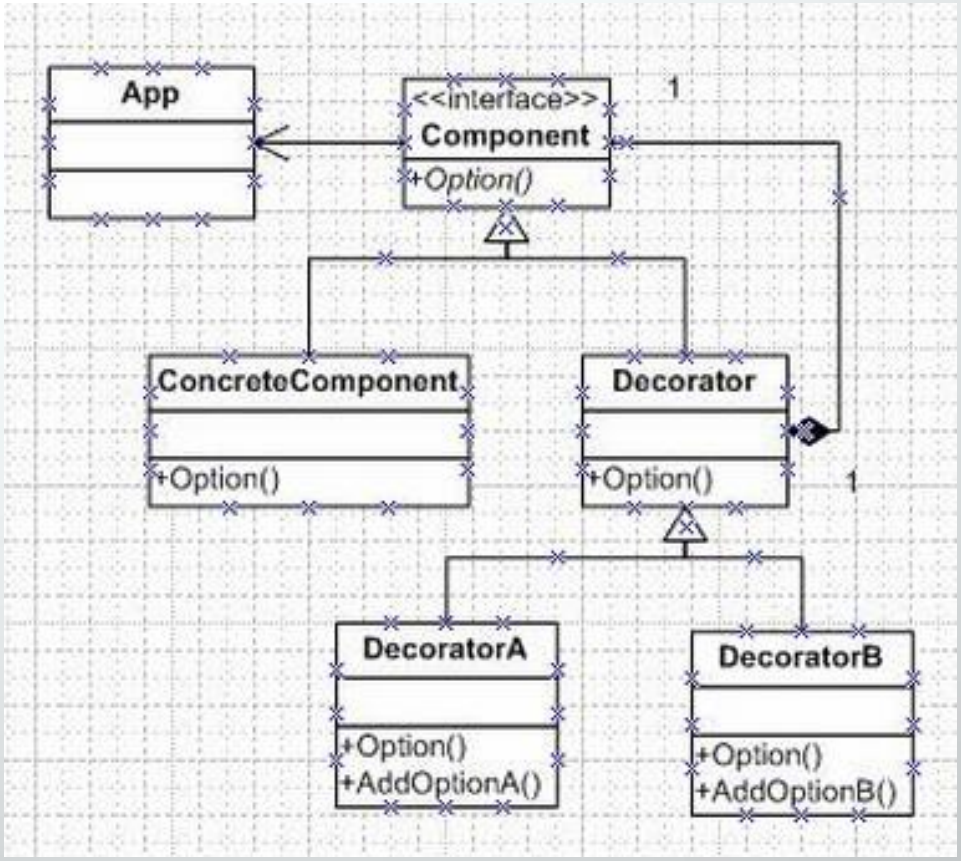
- 环境及问题
- 装饰模式详解
- 装饰模式实现
- 扩展练习

- **装饰模式（Decorator Pattern）。**
  - 装饰模式中的角色：
    - 油漆工(decorator)是用来刷油漆的
    - 被修饰者decoratee是被刷油漆的对象
  - 动态给一个对象添加一些额外的功能和职责，就象在墙上刷油漆。
  - 实现装饰模式有很多形式，最常见的一种就是“实现被装饰者类---定义被装饰者对象----使用被装饰者对象产生装饰者对象”。

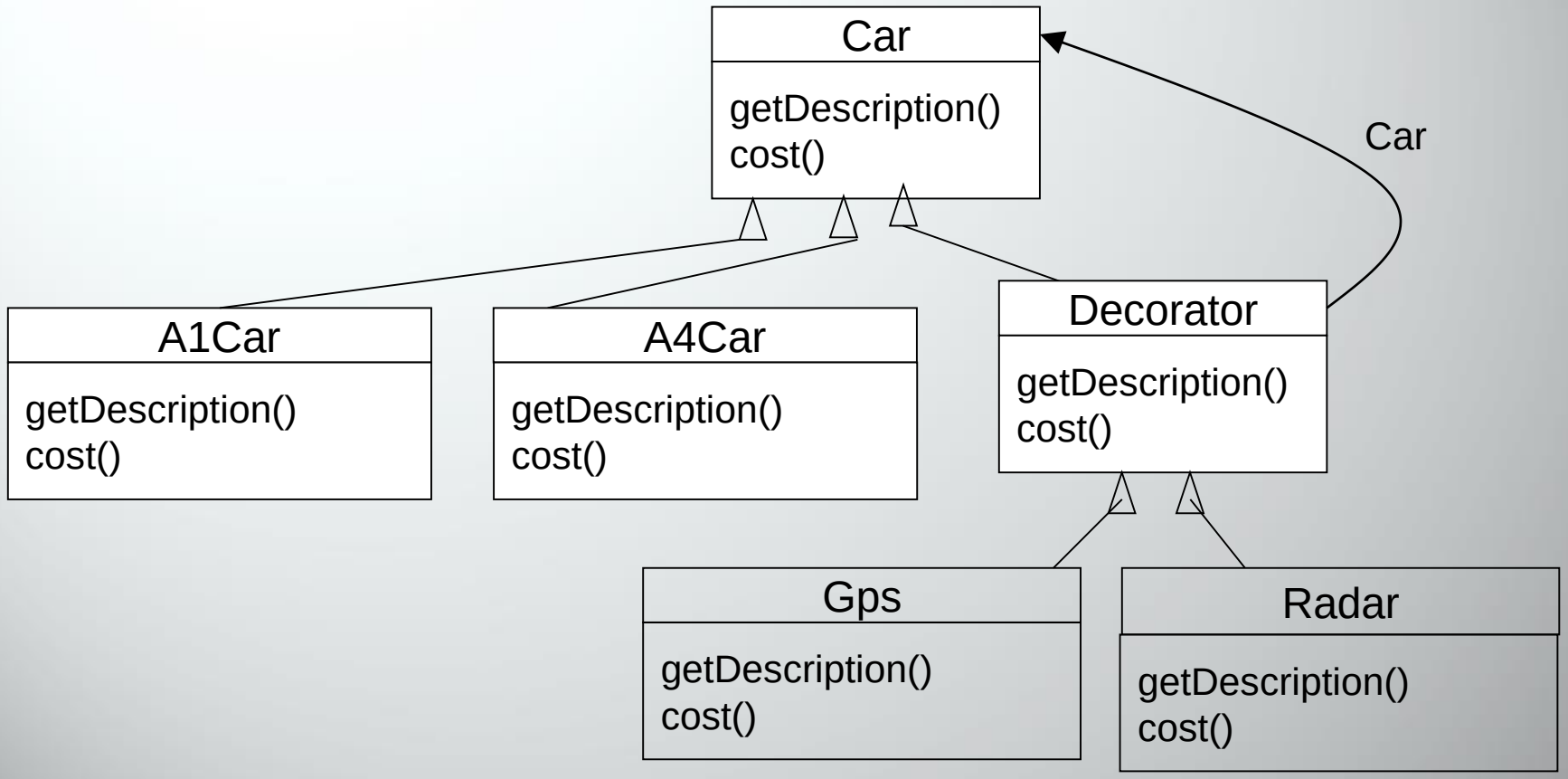
装饰模式的实现

- 首先定义被装饰者类
- 通过被装饰者对象产生装饰者对象

装饰者模式设计类图


App
<<interface>>
Component
+Option()
1
ConcreteComponent
+Option()
Decorator
+Option()
1
DecoratorA
+Option()
+AddOptionA()
DecoratorB
+Option()
+AddOptionB()

实现装饰者类图


Car
getDescription()
cost()
Car
A1Car
getDescription()
cost()
A4Car
getDescription()
cost()
Decorator
getDescription()
cost()
Gps
getDescription()
cost()
Radar
getDescription()
cost()

课程内容

- 环境及问题
- 装饰模式详解
- 装饰模式实现
- 扩展练习

装饰模式实现代码

```
abstract class Car
{
    protected int cost;
    protected string description;

    public abstract int getCost();
    public abstract string getDescription();
}
```

```
class A1Car :Car
{
    public A1Car()
    {
        cost = 100000;
        description = "Audi A1 Car";
    }

    public override int getCost()
    {
        return cost;
    }
    public override string getDescription()
    {
        return description;
    }
}
```

```
class A4Car :Car
{
    public A4Car()
    {
        cost = 120000;
        description = "Audi A4 Car";
    }

    public override int getCost()
    {
        return cost;
    }
    public override string getDescription()
    {
        return description;
    }
}
```

```
class A6Car :Car
{
    public A6Car()
    {
        cost = 160000;
        description = "Audi A6 Car";
    }

    public override int getCost()
    {
        return cost;
    }

    public override string getDescription()
    {
        return description;
    }
}
```

装饰模式实现代码

```
class Decorator :Car
{
    protected Car c;
    protected int decoratorcost;
    protected String decoratordescription;

    public void setCar(Car a)
    {
        c = a;
    }

    public override int getCost()
    {
        return decoratorcost + c.getCost();
    }

    public override string getDescription()
    {
        return c.getDescription() + decoratordescription;
    }
}
```

```
class Gps :Decorator
{
    public Gps()
    {
        decoratorcost = 500;
        decoratordescription = "with Gps";
    }
}
```

```
class Radar : Decorator
{
    public Radar()
    {
        decoratorcost = 1000;
        decoratordescription = "with Radar";
    }
}
```

装饰模式实现代码

```
class Program
{
    static void Main(string[] args)
    {
        A1Car a = new A1Car();
        System.Console.WriteLine(a.getDescription()+"价格："+a.getCost());

        A4Car b = new A4Car();
        System.Console.WriteLine(b.getDescription() + "价格：" + b.getCost());

        Decorator dec = new Gps();
        dec.setCar(b);
        System.Console.WriteLine(dec.getDescription() + "价格：" + dec.getCost());



        System.Console.Read();
    }
}
```

课程内容

- 环境及问题
- 装饰模式详解
- 装饰模式实现
- 扩展练习

扩展说明

- 装饰者与被装饰者具有相同的类型
- 可以用多个装饰者装饰一个对象
- 由于装饰者与被装饰者具有相同的类型，我们可以用装饰后的对象代替原来的对象。
- 装饰者在委派它装饰的对象作某种处理时，可以添加上自己的行为（功能扩展）（在委派之前或/和之后）。
- 对象可以在任何时候被装饰，因此我们能在运行时动态的装饰对象。

- **我有一家咖啡店，其中销售很多种类的咖啡，当然有些客人在点咖啡的时候也会点一些调料，如下：**
  - 咖啡种类
    - **Houseblend:家常混合咖啡**
    - **Decaf:无咖啡因咖啡**
    - **Darkroast:黑咖**
    - **Espresso:意大利浓咖啡**
    - **Latte：拿铁**
    - **Cappuccino：卡布奇洛**
  - 咖啡调料
    - **Mocha:摩卡（巧克力）**
    - **milk:牛奶**
    - **Soy:豆奶**
    - **Whip:起泡牛奶(经过搅打使奶油起泡)**

小结

- 装饰模式解决的问题是“如何动态的给一个对象添加功能”
- 装饰模式的解决方案是利用子对象，委派

Thank You，谢谢！