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

第四章 策略模式

任课教师：武永亮
wuyongliang@edu2act.org

上节回顾

- 将一个类的接口转换成客户希望的另外一个接口
- 使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。
- 尽可能保持已有的类不变的前提下，适应当前的系统。

课程内容

- 环境及问题
- 策略模式详解
- 策略模式实现
- 扩展练习

课程内容

- 环境及问题
- 策略模式详解
- 策略模式实现
- 扩展练习

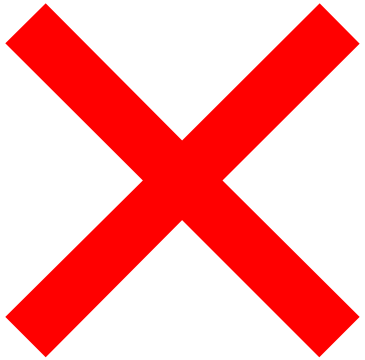
环境

- 武士可以随时更换武器！

请思考10分钟：如何对该需求进行设计？

环境

```
public class Context
{
    ......
    public void fighting(String type)
    {
        if(type == "knife")
        {
            //A战斗方案
        }
        else if(type == "bow")
        {
            //B战斗方案
        }
        else if(type == "cannon")
        {
            //C战斗方案
        }
        ......
    }
}
```

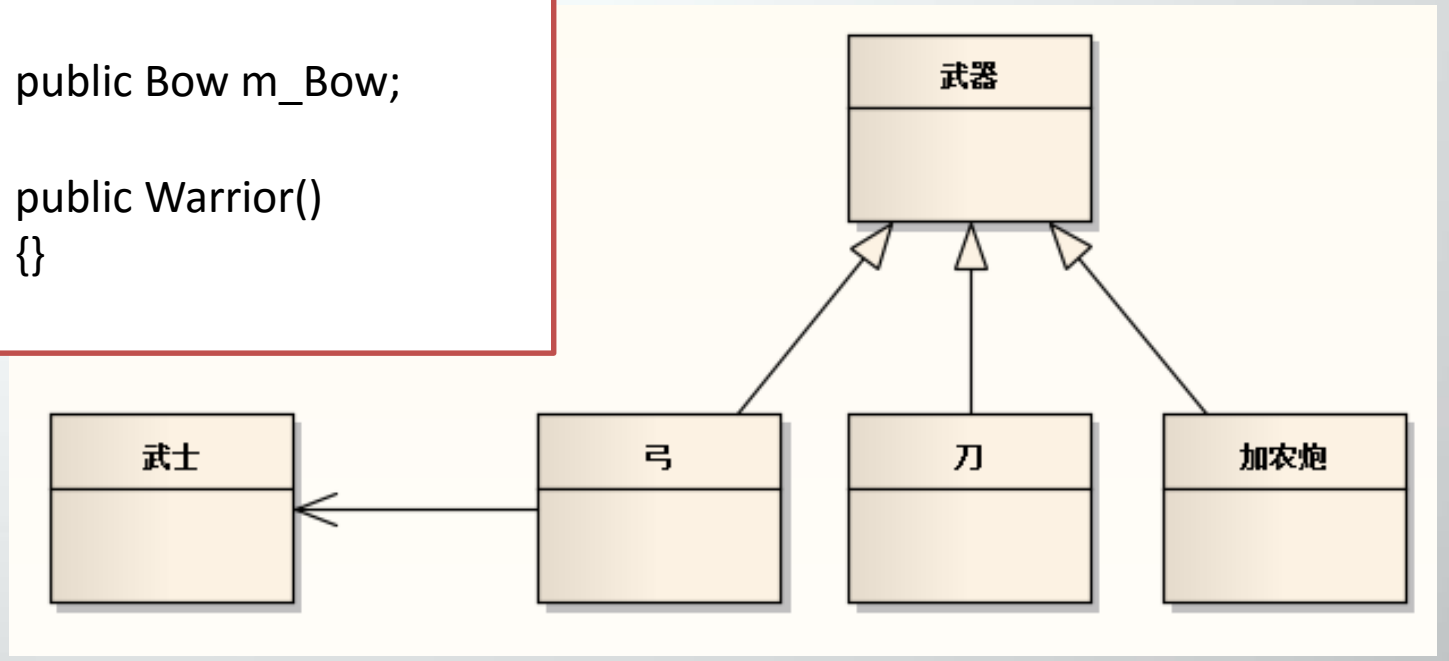


环境

环境

```
public class Warrior
 {
      public Bow m_Bow;

      public Warrior()
      {}
}
```

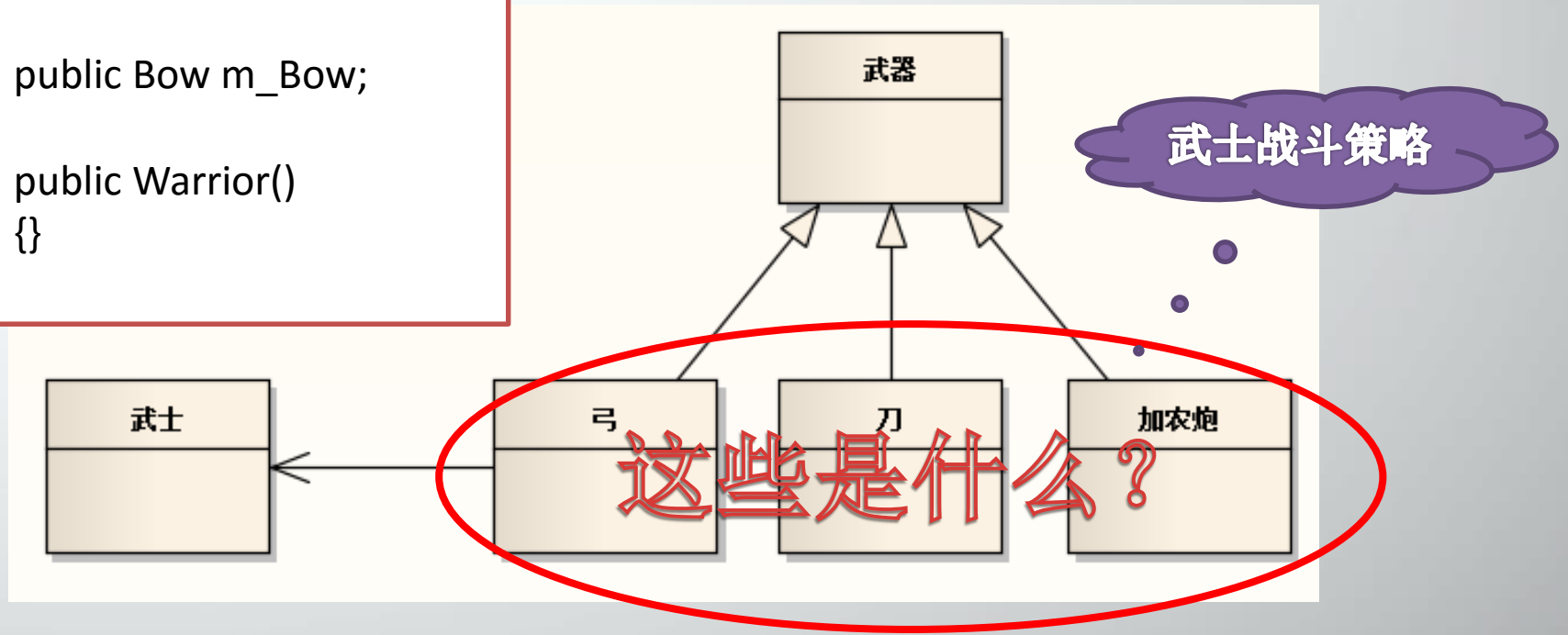


请思考：这样设计有什么样的问题？

环境

```
public class Warrior
 {
      public Bow m_Bow;

      public Warrior()
      {}
}
```



满足开闭原则吗？满足依赖倒置原则吗？

问题

- 将每一个一系列的算法封装起来。
- 而且使它们还可以相互替换。

策略模式（Strategy）

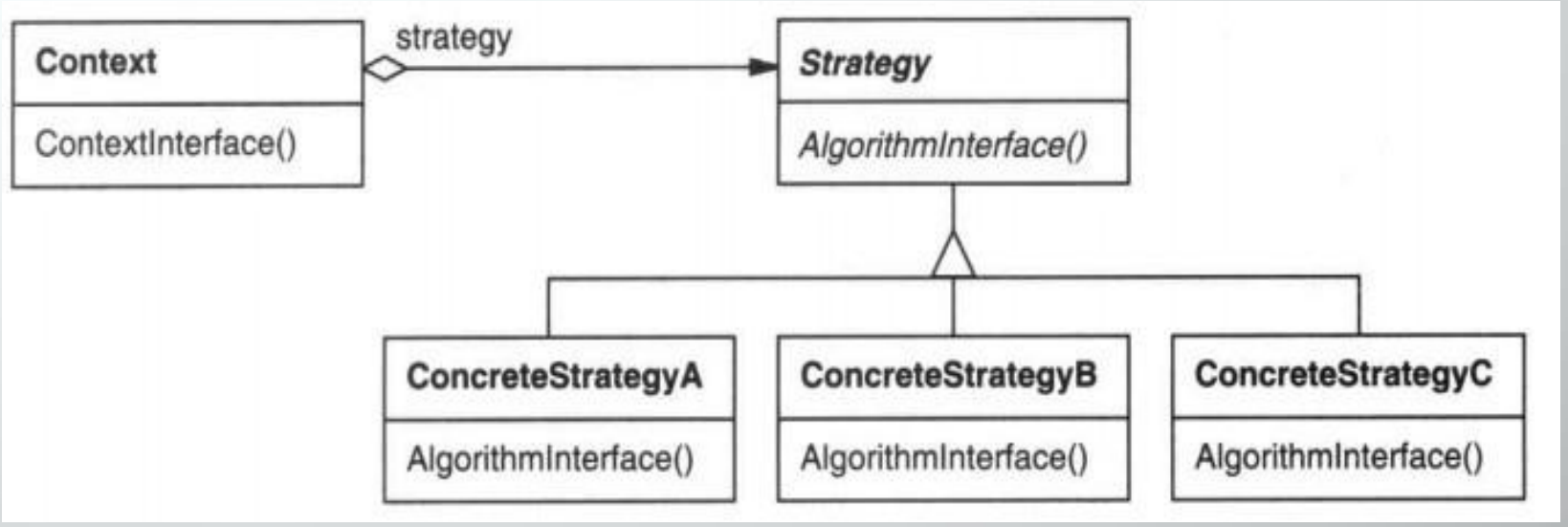
- 让算法独立于使用它的客户而独立变化。

课程内容

- 环境及问题
- 策略模式详解
- 策略模式实现
- 扩展练习

策略模式（Strategy Pattern）

- 策略模式定义了一系列的算法，并将每一个算法封装起来，而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化



策略模式（Strategy Pattern）

- 策略模式中的有以下的三种角色。
  - 抽象策略类(Strategy): 定义所有支持的算法的公共接口。
  - 具体策略类(ConcreteStrategy): 以Strategy接口实现某具体算法。
  - 环境类(Context) : 维护一个对Strategy对象的引用。可定义一个接口来让Strategy访问它的数据。

课程内容

- 环境及问题
- 策略模式详解
- 策略模式实现
- 扩展练习

策略模式实现步骤

- 定义抽象策略类
- 实现具体策略类
- 定义环境类

策略模式实现步骤一

- 定义抽象策略类。

```
interface IStrategy
{
    void fighting();
}
```

策略模式实现步骤二

- 实现具体策略类。

```
class Bow:IStrategy
{
    public void fighting()
    {
        Console.WriteLine("向敌人放冷箭中……");
    }
}
```

```
class Knife:IStrategy
{
    public void fighting()
    {
        Console.WriteLine("把敌人千刀万剐中……");
    }
}
```

```
class Cannon:IStrategy
{
    public void fighting()
    {
        Console.WriteLine("加农炮轰击敌人中……");
    }
}
```

策略模式实现步骤三

- 定义环境类。

```
class Context
{
    private IStrategy _strategy;

    public Context(IStrategy s)
    {
        this._strategy = s;
    }

    public void fighting()
    {
        this._strategy.fighting();
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Context context;

        context = new Context(new Knife());
        Console.WriteLine("选择武器为刀：");
        context.fighting();
        Console.WriteLine();

        context = new Context(new Bow());
        Console.WriteLine("选择武器为弓：");
        context.fighting();
        Console.WriteLine();

        context = new Context(new Cannon());
        Console.WriteLine("选择武器为加农炮：");
        context.fighting();
        Console.WriteLine();
    }
}
```

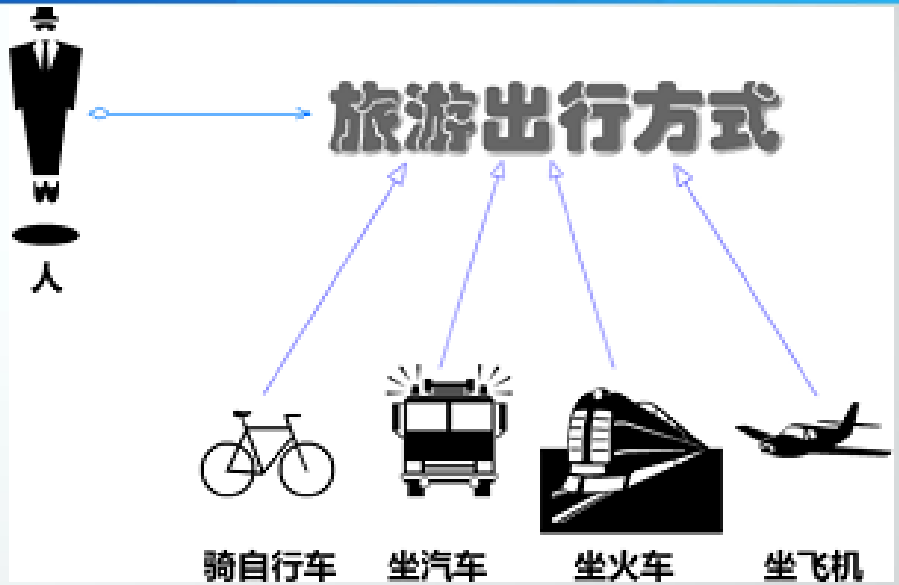
思考：适配器模式与策略模式的区别

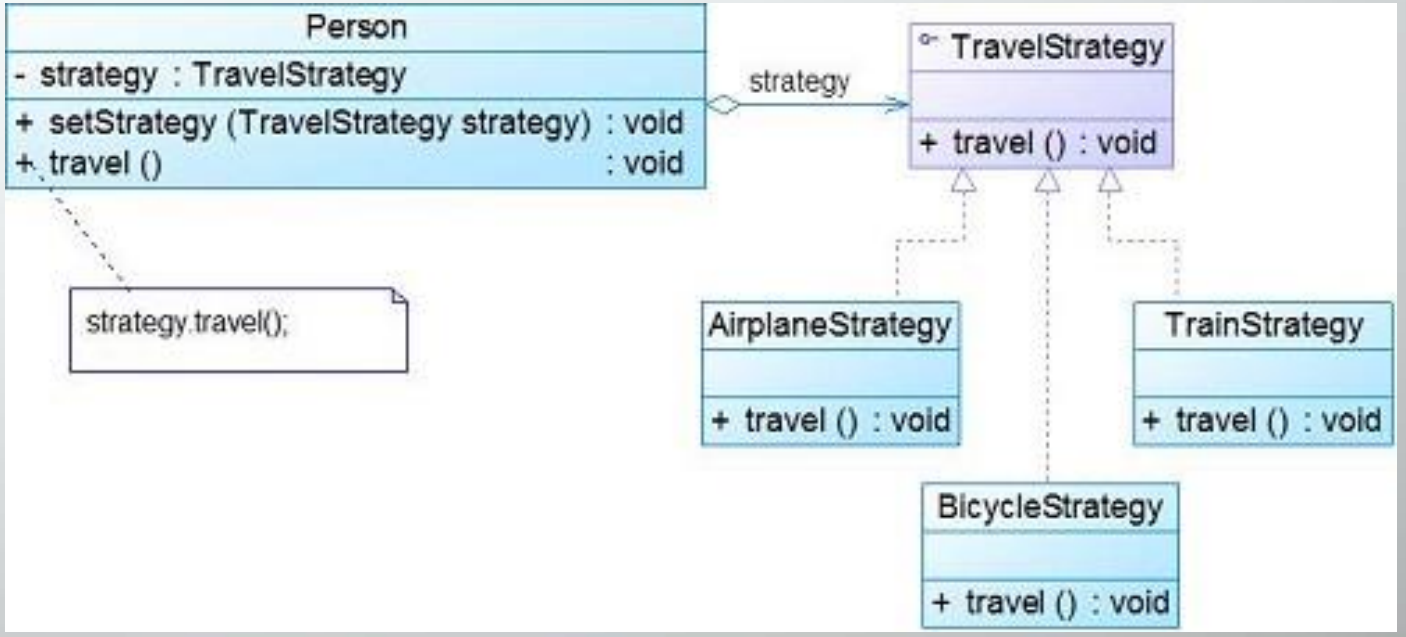
- 武器升级，其他接口实现与匹配。
- 武器变更，其他武器动态更换。

课程内容

- 环境及问题
- 策略模式详解
- 策略模式实现
- 扩展练习

案例练习


人
旅游出行方式
骑自行车
坐汽车
坐火车
坐飞机

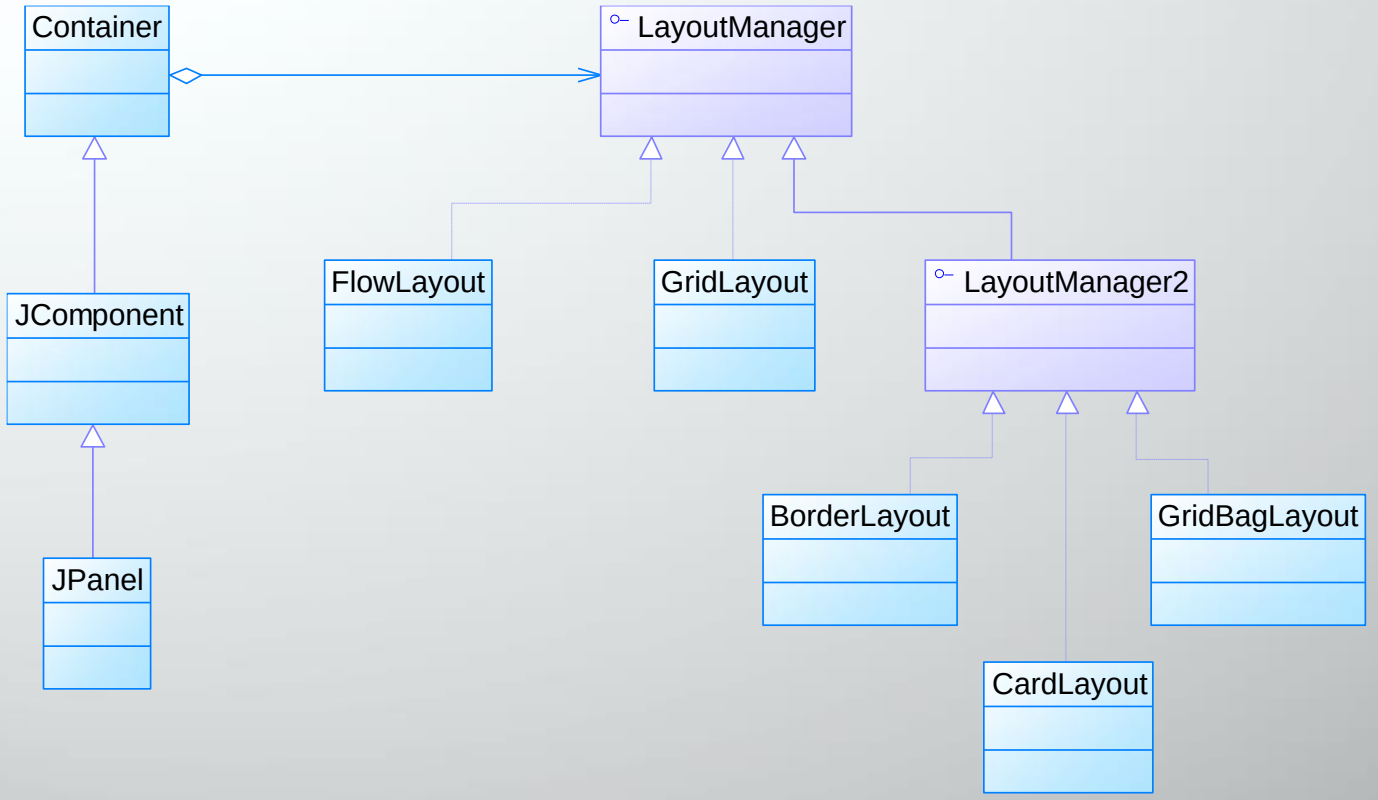

Person
- strategy : TravelStrategy
+ setStrategy (TravelStrategy strategy) : void
+ travel () : void
strategy
TravelStrategy
+ travel () : void
strategy.travel();
AirplaneStrategy
+ travel () : void
TrainStrategy
+ travel () : void
BicycleStrategy
+ travel () : void

案例练习


商场收银系统
单价：
90
确定
数量：
2
重置
计算方式：
满200返40
单价:90 数量：2. 实收：180 正常收费
单价:90 数量：2. 实收：144 打8折
单价:90 数量：2. 实收：180 满200返40
总计：
594

扩展说明

- Java SE的容器布局管理就是策略模式应用的一个经典实例



扩展说明

- 策略模式的优点
  - 策略模式提供了对“开闭原则”的完美支持，用户可以在不修改原有系统的基础上选择算法或行为，也可以灵活地增加新的算法或行为。
  - 策略模式提供了管理相关的算法族的办法。
  - 策略模式提供了可以替换继承关系的办法。
  - 使用策略模式可以避免使用多重条件转移语句。

扩展说明

- 策略模式的缺点
  - 客户端必须知道所有的策略类，并自行决定使用哪一个策略类。
  - 策略模式将造成产生很多策略类。

小结

- 策略模式适用环境
  - 如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。
  - 一个系统需要动态地在几种算法中选择一种。
  - 不希望客户端知道复杂的、与算法相关的数据结构，在具体策略类中封装算法和相关的数据结构，提高算法的保密性与安全性。

Thank You，谢谢！