

# Object oriented Analysis &Design

## 面向对象分析与设计

### Lecture \_16 Design Pattern-Adapter

主讲: 陈小红

日期:

# GoF设计模式的分类

(1) Creational (创建型) 5个

(2) Structural (结构型) 7个

(3) Behavioral (行为型) 11个

	创建型	结构型	行为型
类	Factory Method	Adapter_Class	Interpreter Template Method
对象	Abstract Factory Builder Prototype Singleton	Adapter_Object Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

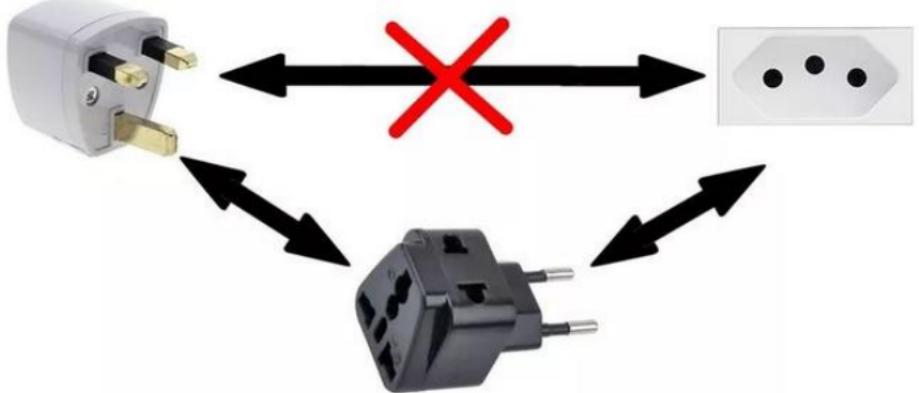
# 引子

- NBA 需要翻译，在CBA我不需要
- 姚明



# Adapter

- 转换插头

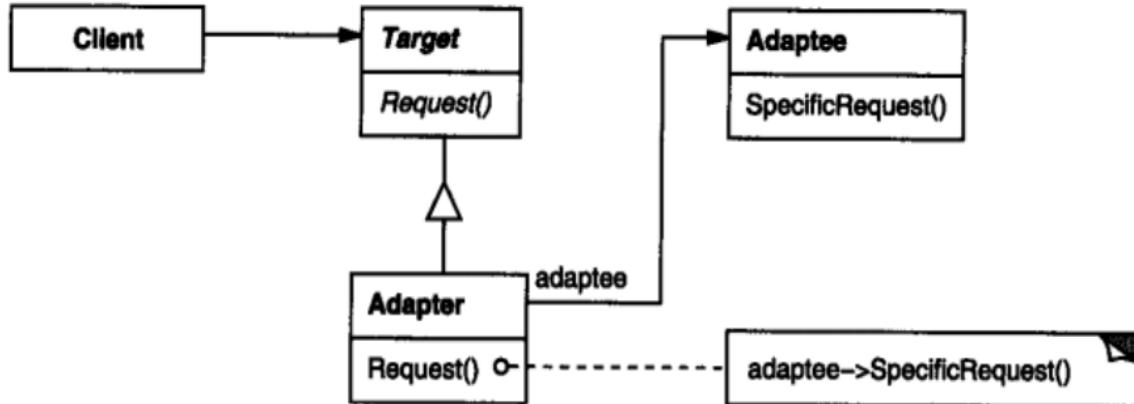


## 更多例子

- 笔记本的电源适配器
- USB读卡器



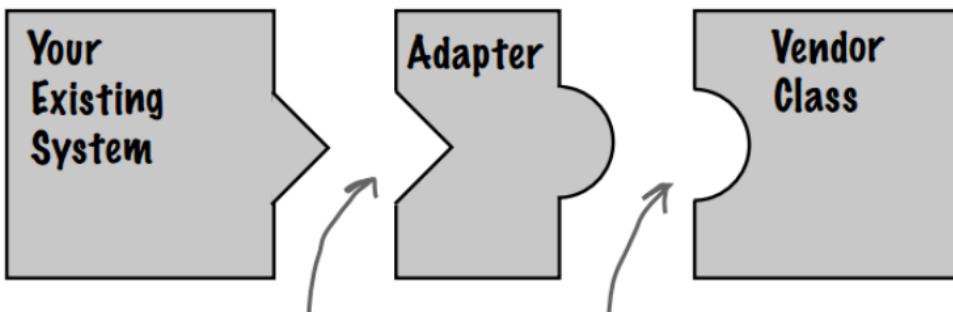
## 对象的适配器模式



- Target: 目标抽象类，定义客户要用的特定领域的接口。
- Adapter: 适配器，调用另一个接口，作为一个转换器。
- Adaptee: 源，定义一个接口，Adapter需要接入。
- Client: 客户调用类，协同对象符合Adapter适配器。

# 动机

- 在软件系统中，由于应用环境的变化，常常要将“一些现存的对象”放在新的环境中应用，但是新环境要求的接口是这些现存对象所不满足的。
- 在不改变原有实现的基础上，将原先不兼容的

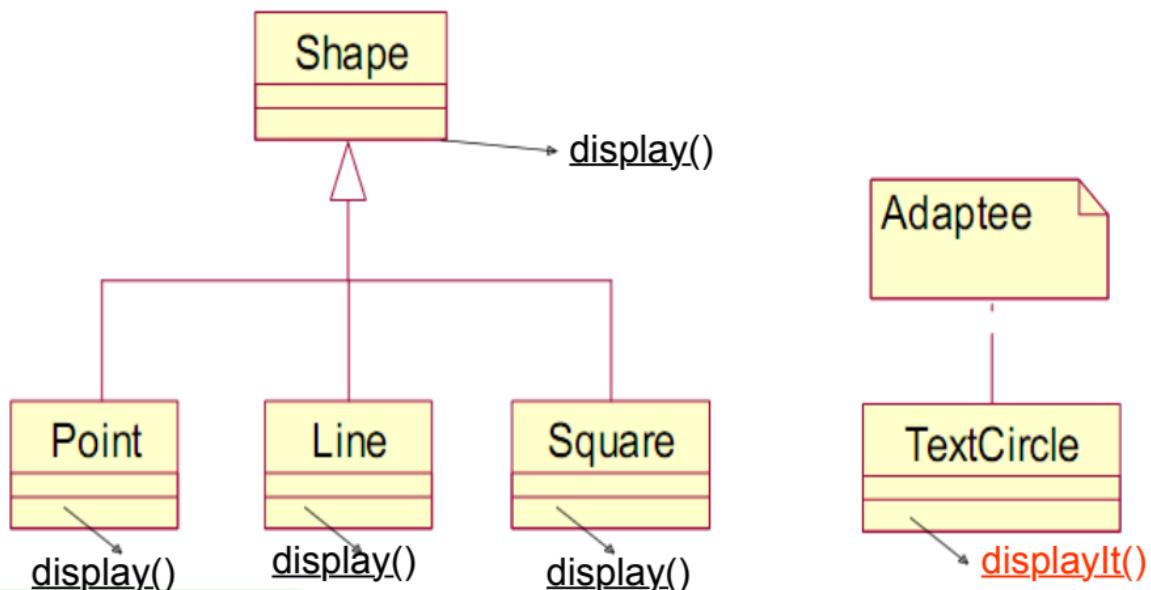


# 适配器定义

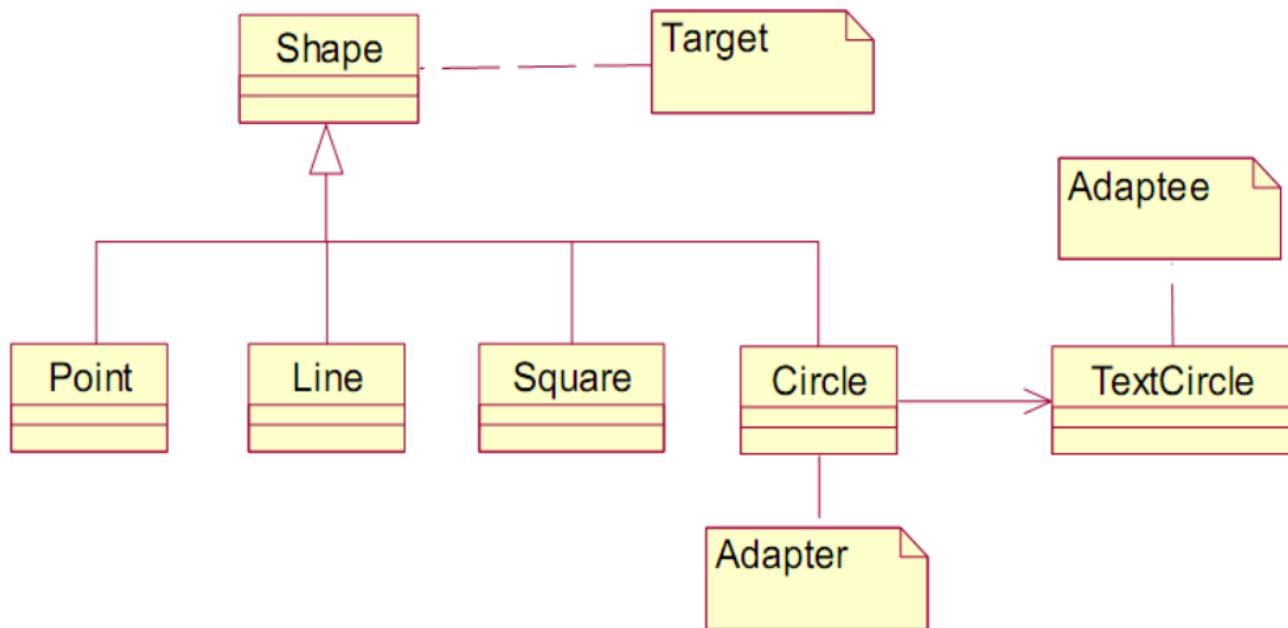
适配器模式使得一个接口与其它接口兼容，从而给出多个不同接口的统一抽象。

该模式是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。

# 添加绘制圆的需求前的类结构：



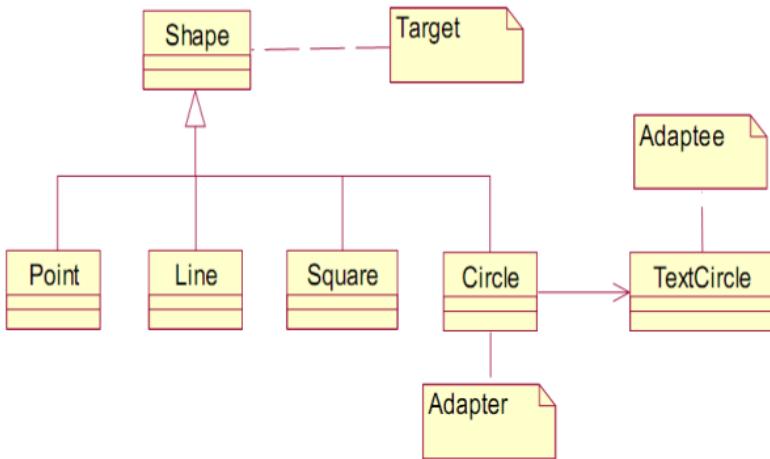
## 添加了圆的绘制后的类结构：





可以看出 Shape、Circle 和 TextCircle 三者的关系是和标准适配器模式中 Target、Apater、Apatee 三者的关系相对应的。我们只关心这个画图程序中是怎么来使用适配器模式的。看看 Circle 的实现代码

```
class Circle extends Shape  
{  
    //这里引用了 TextCircle  
    private TextCircle tc;  
    public Circle ()  
    {  
        tc= new TextCircle();  
    }  
    void public display()  
  
    {  
        tc.displayIt();  
    }  
}
```



//在规定的方法里面调用 TextCircle 原来的方法

## 现实中的例子

显示器是用来显示图形的，它是不能显示数据，它只能够接受来自图形发送设备的信号。可是我们手头上只有CPU这个产生各种描述图形的数据的数据发送器。我们需要将这些数据让显示器进行显示，可是这两个部件却是不兼容的。于是我们需要一个中间设备，它能够将CPU“适配”于显示器，这便是显卡——图形适配器

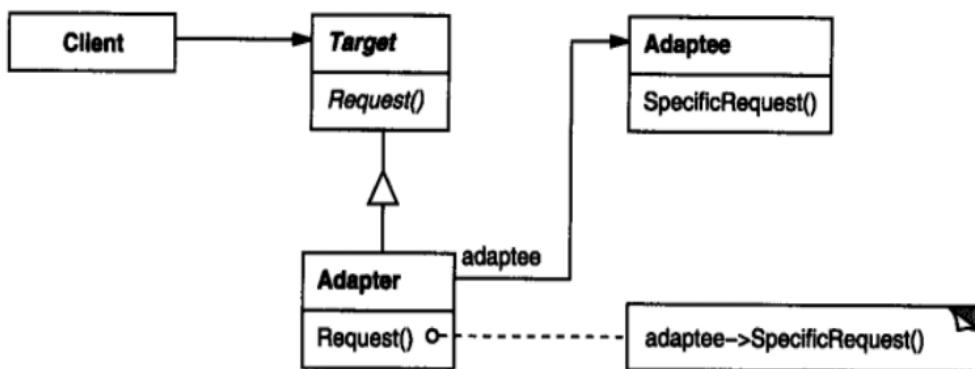


# 找出adapter模式中的角色

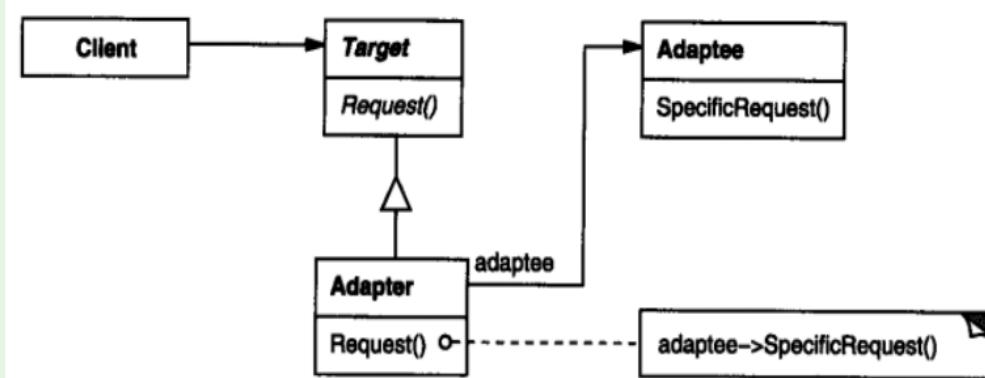


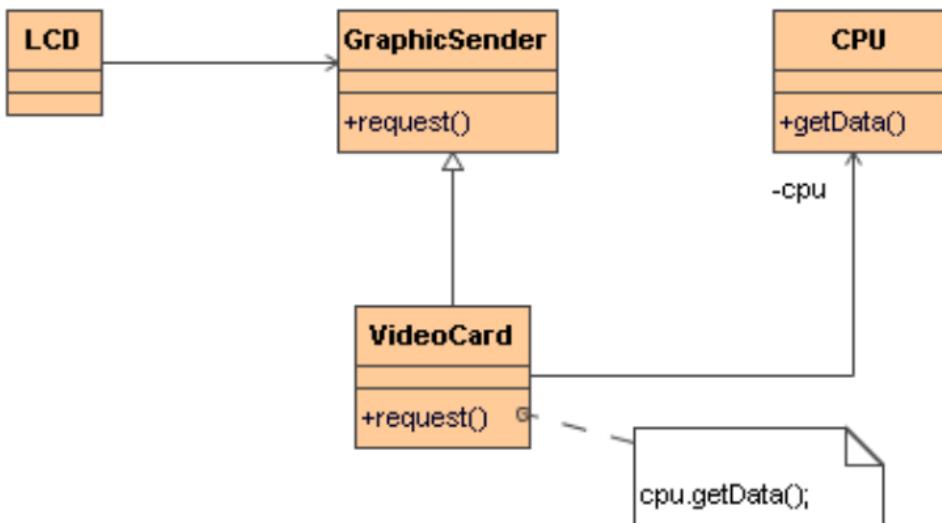
Client?  
Target?  
Adaptee?  
Adapter

显示器是用来显示图形的，它是不能显示数据，它只能够接受来自图形发送设备的信号。可是我们手头上只有CPU这个产生各种描述图形的数据的数据发送器。我们需要将这些数据让显示器进行显示，可是这两个部件却是不兼容的。于是我们需要一个中间设备，它能够将CPU“适配”于显示器，这便是显卡——图形适配器。



显示器(Client)是用来显示图形的，它是不能显示数据，它只能够接受来自图形发送设备(Target)的信号。可是我们手头上只有CPU(Adaptee)这个产生各种描述图形的数据的数据发送器。我们需要将这些数据让显示器进行显示，可是这两个部件却是不兼容的。于是我们需要一个中间设备，它能够将CPU“适配”于显示器，这便是显卡——图形适配器(Adapter)。





## Next: 类图->java代码

```
public class LCD {  
  
    public static void main(String[] args) {  
  
        // CPU经过显卡的适配后“变”成了图形发送装置  
        GraphicSender gs =  
            new VideoCard(new CPU0);  
        System.out.println(gs.request());  
    }  
  
}
```

```
// 图形发送设备 (Target)
public class GraphicSender {
    /**
     * 传送图形信号
     */
    public String request() {
        return "Graphic sender";
    }
}
```

我们的CPU(Adaptee)只能输出0/1数据，它是一个计算器，而不是图形发送设备(Target)。

```
// CPU
public class CPU {
    /**
     * CPU输出的数据
     */
    public String getData() {
        return "CPU data";
    }
}
```

这个时候我们的显卡(Adapter)的作用便体现出来了，它负责对CPU进行适配，通过将CPU传过来的数据转换成图形信号，从而将CPU伪装成一个图形发送设备。

// 显卡，即我们的适配器

```
public class VideoCard extends GraphicSender {
```

// 被代理的设备

```
private CPU cpu = null;
```

```
/**
```

\* 装入被代理的设备

```
*/
```

```
public VideoCard (CPU cpu) {
```

```
    this. cpu = cpu;
```

```
}
```

```
/**
```

\* 被代理的设备传过来的数据转换成为图形输出

```
*/
```

```
public String request() {
```

```
    return cpu.getData();
```

```
}
```

```
}
```

# 在什么情况下使用适配器模式

在以下各种情况下使用适配器模式：

- 1、系统需要使用现有的类，而此类的接口不符合系统的需要。
- 2、想要建立一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作。这些源类不一定有很复杂的接口。