Object Oriented Analysis & Design
面向对象分析与设计

Lecture_09 GOF 设计模式（一）

1）单实例　　2）适配器　　3）外观　　4）观察者

主讲: 姜宁康 博士

- ## 3、GOF设计模式二: 适配器模式 Adapter

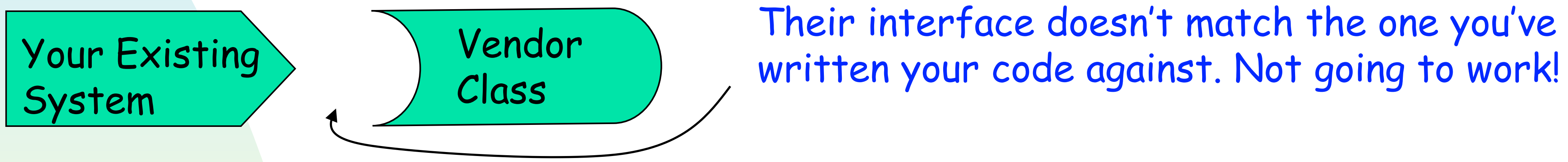
  - 为中国市场生产的电器，到了美国，需要有一个转接器才能使用墙上的插座，这个转接器的功能、原理?

课程之前：复习单实例模式

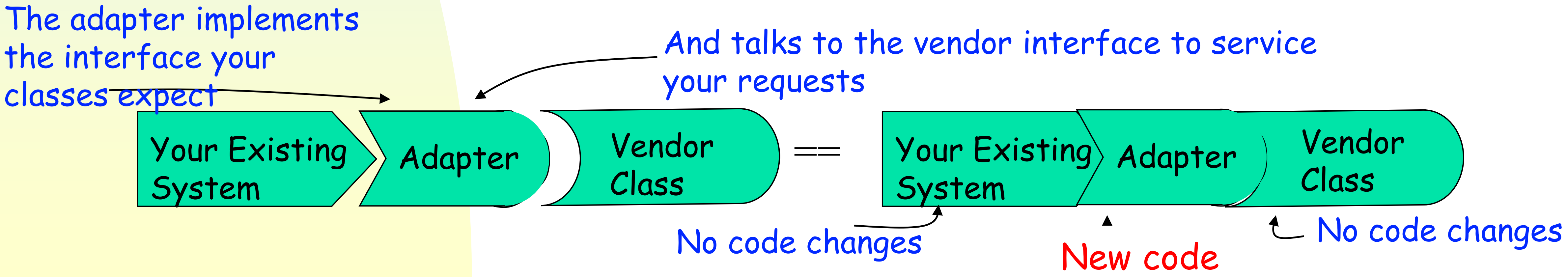
- SingleTon的三个关键点
  - 1）私有的：构造函数
  - 2）私有的：成员变量，记录这个单实例
  - 3）公有的getter函数：没有实例时创建它；已有实例则返回该实例

3.1 现实遇到的问题: 适配器 Adapters

- 现实生活中充满着需要适配的场合
  - 你知道吗?
- 面向对象中的适配器 Object oriented adapters
  - Scenario: you have an existing software system that you need to work a new vendor library into, but the new vendor designed their interfaces differently than the last vendor


Your Existing System
Vendor Class
Their interface doesn't match the one you've written your code against. Not going to work!

  - What to do? Write a class that adapts the new vendor interface into the one you're expecting.



3.2 例 现有一只火鸡，但需要的是一只鸭子 …..

- **If it walks like a duck and quacks like a duck, then it might be a duck-turkey wrapped with a duck adapter….**

```
public interface Duck {
      public void quack ();
      public void fly ();
}
public class MallardDuck
             implements Duck {
      public void quack () {
          System.out.println("Quack");
       }
       public void fly ( ) {
          System.out.println ("I am flying");
       }
 }
```

```
//遇到一只家禽，火鸡 Meet the fowl!
 public interface Turkey {
    public void gobble ();
    public void fly ( );
  }
 public class WildTurkey implements Turkey {
     public void gobble ( ) {
         System.out.println("Gobble Gobble");
     }
     public void fly ( ){
          System.out.println("I'm flying a short
distance");
     }
 }
```

Concrete implementations are similar -- just print out the actions

3.2 例 Now....

- 需要用火鸡来代替鸭子的时候，因为两者的接口不同，不能直接使用
  - 咋办?

```
public class TurkeyAdapter implements Duck {
     Turkey turkey;
    public TurkeyAdapter (Turkey turkey) {
      this.turkey = turkey;
    }
    public void quack ( ) {
      turkey.gobble ( );
    }
    public void fly ( ){
      for (int j = 0; j<5; j++)
          turkey.fly ( );
    }
   }
}
```

First, you need to implement the interface of the type you are adapting to. This is the interface your client expects. 希望有只鸭子

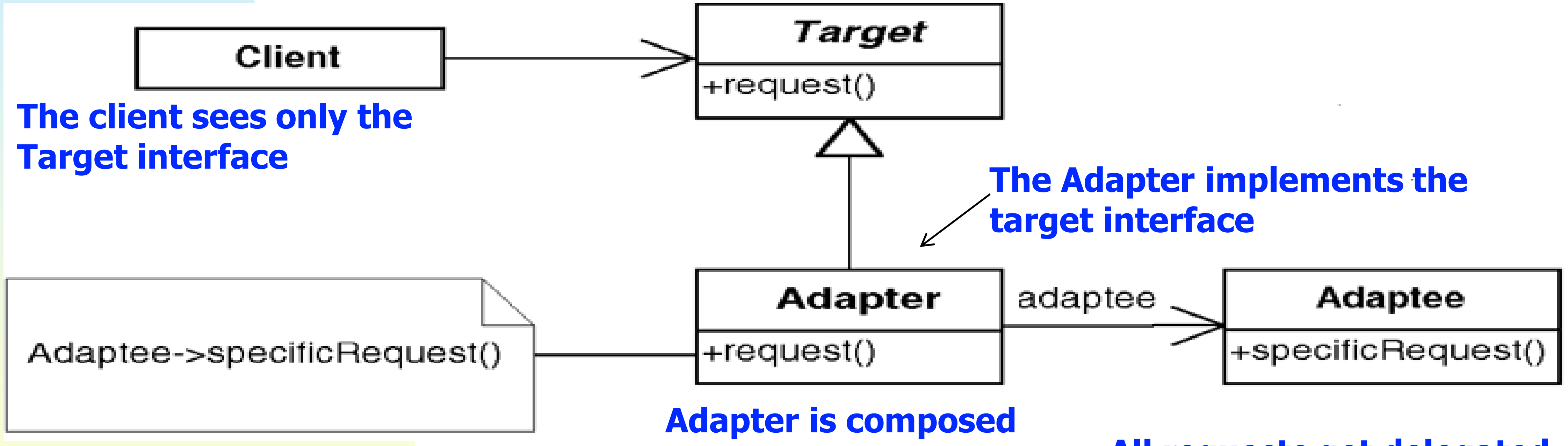
Next, we need to get a reference to the object that we are adapting; here we do that through the constructor. 传进来一只火鸡

Now we need to implement all the methods in the interface; the quack() translation between classes is easy; just call the gobble method. 将就着吧，需要鸭子叫的时候，就让让火鸡叫吧

Even though both interfaces have a fly ( ) method, Turkeys fly in short spurts -- they can't do long distance flying like ducks. To map between a Duck's fly ( ) method and a Turkey's we need to call the Turkey's fly ( ) method five times to make up for it. 需要鸭子飞的时候，让火鸡多飞几次，假装是鸭子飞。因为火鸡飞一次距离较短

3.3 适配器模式 The Adapter Pattern Defined

The Adapter Pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces 将一个类的接口转换成客户希望的另外一个接口，Adapter模式使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



The client sees only the Target interface

The Adapter implements the target interface

Adapter is composed with the Adaptee

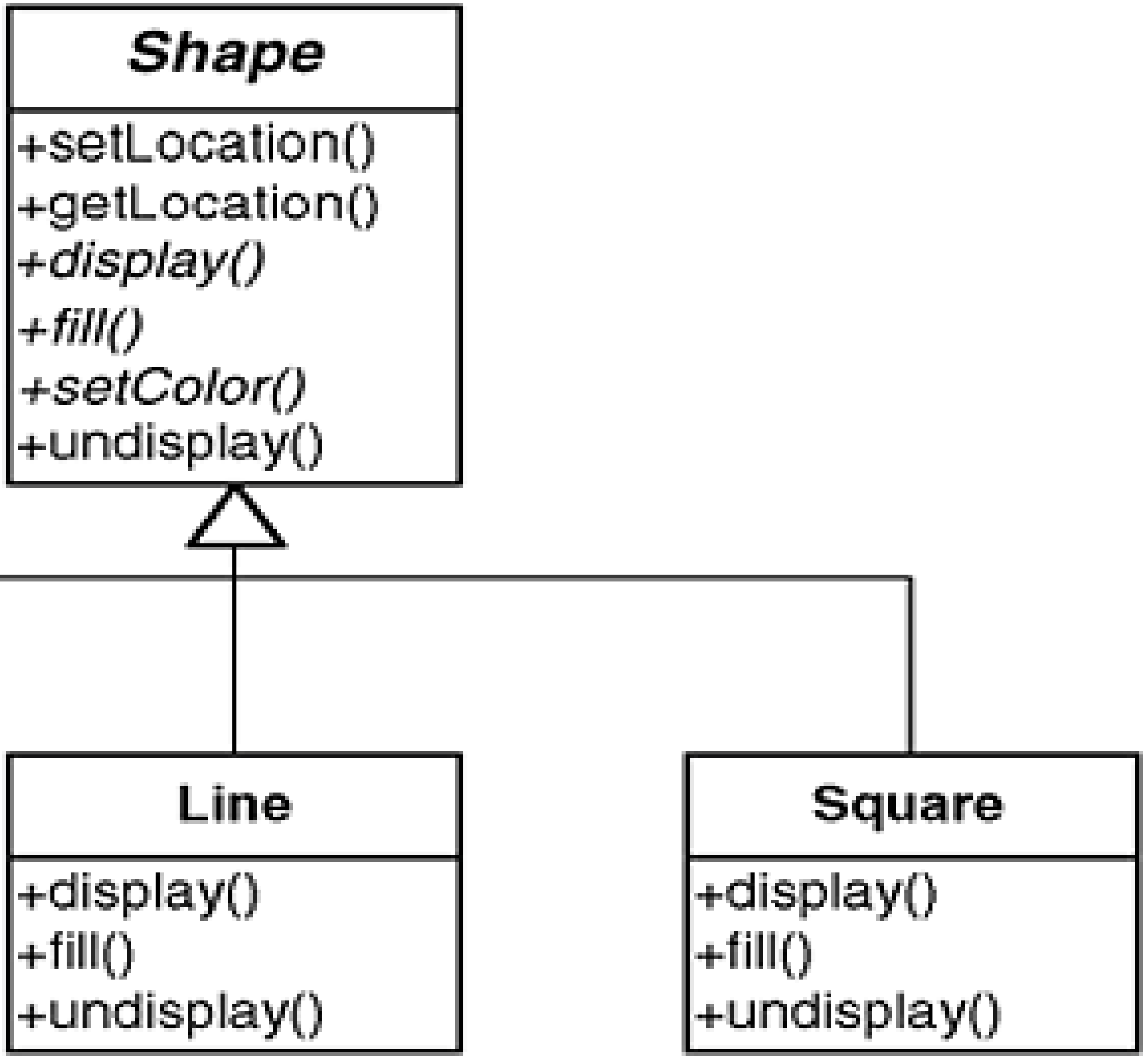
All requests get delegated to the Adaptee

Full of good OO design principles:多个好的OO设计原则
--Use of object composition 对象组合
--Pattern binds the client to an interface and not an implementation 客户面向接口而不是实现

3.4 适配器 Example

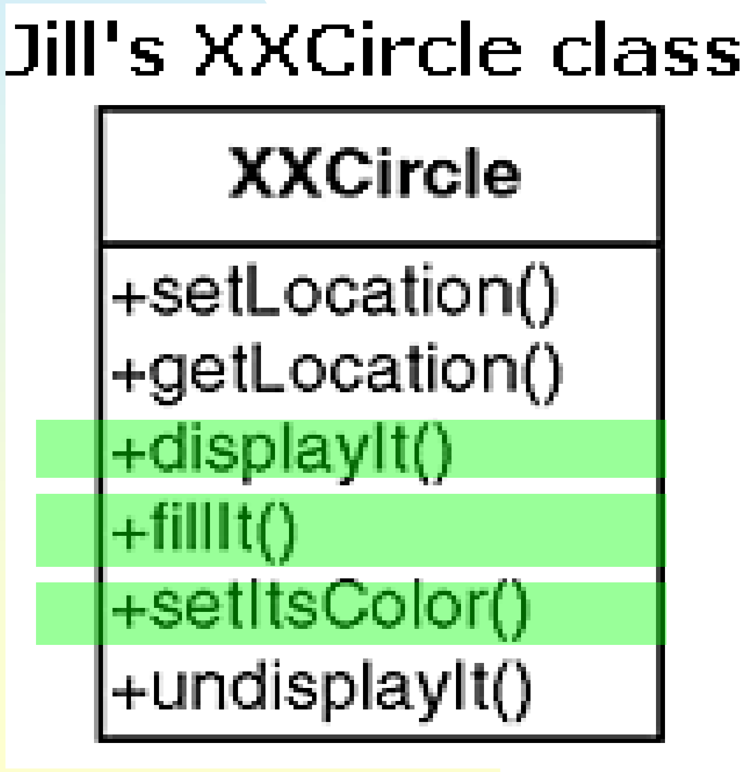
Define the interface and then implement in derived classes

Suppose now we are asked to implement a circle, a new kind of Shape ?



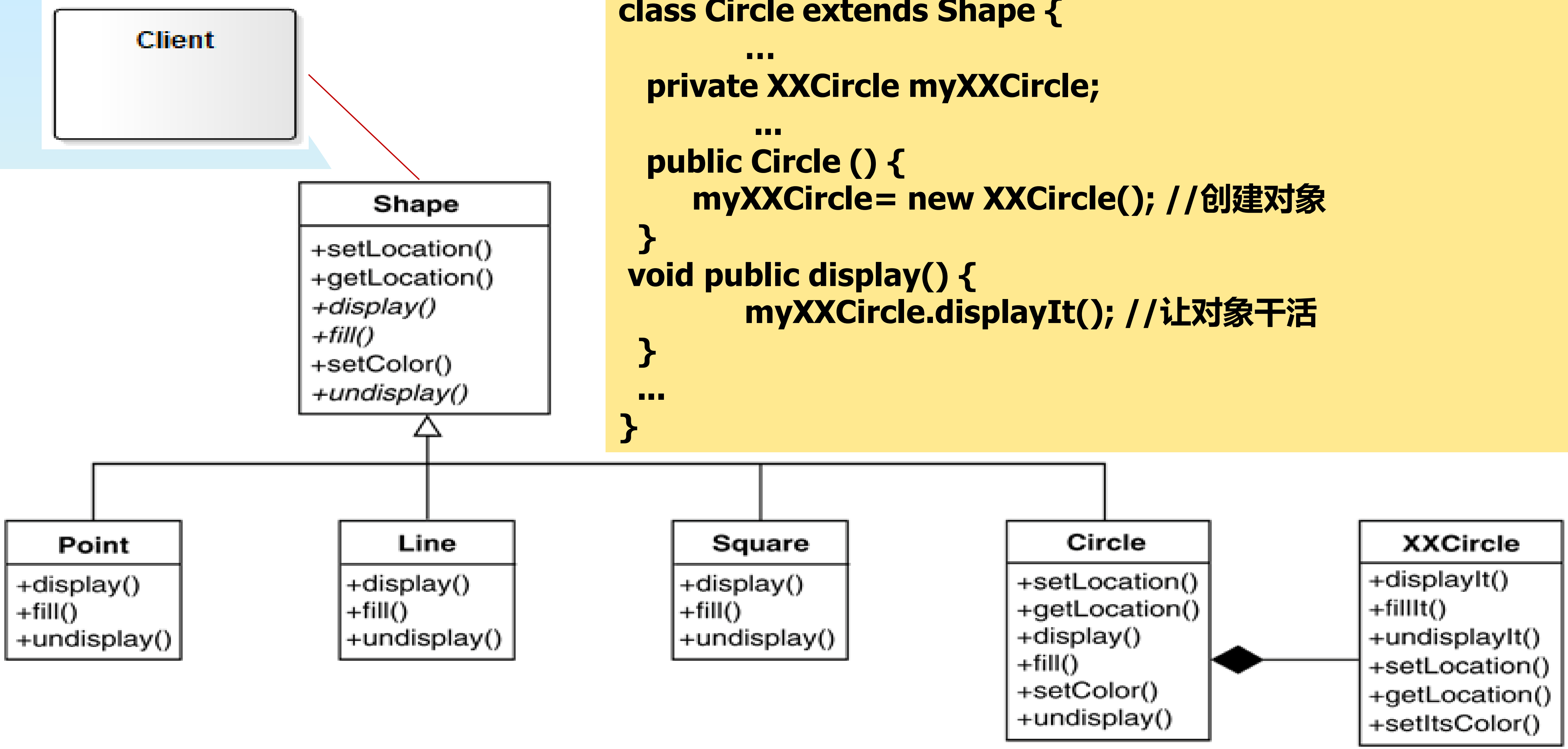
3.4 适配器 Example

我知道Jill已经实现画圆的功能，命名为XXCircle I discover that Jill has already written a class she called XXCircle that already handles circles



眼看需要的模块就在眼前，但是用不起来，自己又不想重新写一个，咋办?
I have what I want almost within reach, but I cannot use it and I do not want to rewrite it. What can I do?

3.4 适配器Example: Circle "wraps" XXCircle



Java Code Fragments: Implementing the Adapter Pattern

```
class Circle extends Shape {
            ...
   private XXCircle myXXCircle;
            ...
   public Circle () {
       myXXCircle= new XXCircle(); //创建对象
  }
 void public display() {
            myXXCircle.displayIt(); //让对象干活
  }
  ...
}
```

3.5 The Adapter Pattern: Key Features

Intent	Match an existing object beyond your control to a particular interface
Problem	A system has the right data and behavior but the wrong interface. Typically used when you have to make something a derivative of an abstract class
Solution	The Adapter provides a wrapper with the desired interface
Participants and collaborators	The Adapter adapts the interface of an Adaptee to match that of the Adapter's Target (the class it derives from). This allows the Client to use the Adaptee as if it were a type of Target
Consequences	The pattern allows for preexisting objects to fit into new class structures without being limited by their interfaces
Implementation	Contain the existing class in another class. Have the containing class match the required interface and call the methods of the contained class

3.6 实现细节 Check list

- 两个角色 Identify the players
  - Adaptee：The Adapter adapts the interface of an Adaptee to match that of the Adapter‘s Target (the class it derives from)
  - Client：use the Adaptee as if it were a type of Target
- 标识客户需要用到的接口 Identify the interface that the client requires
- 设计一个“包装器”使得 Adaptee 符合客户的需要
- adapter/wrapper 类里面必须要有一个 adaptee class的对象（负责工作)
- adapter/wrapper 负责“映射” 客户接口与 adaptee 的接口
- client uses (is coupled to) the new interface

3.7 适配器小结 Summary

- **当现有的系统需要使用另一个类Adaptee的功能，而那个类的接口又不符合现有的系统，就要使用适配器** When you need to use an existing class and its interface is not the one you need, use an adapter
- **适配器负责把Adaptee类的接口转换成客户类需要的格式，以完成客户的工作** An adapter changes an interface into one a client expects
- **实现一个适配器的工作量取决于目标接口的规模** Implementing an adapter may require little work or a great deal of work depending on the size and complexity of the target interface

- 本讲结束