## 一句话概括：

**一个类代表另一个类的功能。**

## 补充介绍：

代理模式（Proxy Pattern）中，我们创建现有对象的代理对象，向外界提供功能接口。

代理模式旨在为提供一个对象的代理对象，从而控制被代理对象的访问。

代理模式主要解决在直接访问对象时带来的问题，比如要访问的对象在远程，再或者要访问的对象需要安全控制。

从字面上看也很明显，就是代表某个对象去做某件事，比如你办房产证懒得跑路，怎么办呢，可以找个结构代理你本人跑一下咯，你只需要提供你的主要信息（身份证，购房合同，房款发票等），机构就能全权代表你去做这件事了，同样的对象也只需要提供它的接口和重要信息，代理类也能完成它的工作，并且还可以加入很多额外的动作，比如在访问对象前做什么事，访问后做什么事等。

## 参与角色:

1）被代理对象的基类或接口

2）被代理对象类

3）代理对象类

## 优点：

1、职责清晰。 2、高扩展性。 3、智能化。

## 缺点：

1、由于在客户端和真实主题之间增加了代理对象，因此有些类型的代理模式可能会造成请求的处理速度变慢。 2、实现代理模式需要额外的工作，有些代理模式的实现非常复杂。

## 使用案例或场景：

**使用场景**：

按职责来划分，通常有以下使用场景： 1、远程代理。 2、虚拟代理。 3、Copy-on-Write 代理。 4、保护（Protect or Access）代理。 5、Cache代理。 6、防火墙（Firewall）代理。 7、同步化（Synchronization）代理。 8、智能引用（Smart Reference）代理。

**案例：**

1、Windows 里面的快捷方式。 2、猪八戒去找高翠兰结果是孙悟空变的，可以这样理解：把高翠兰的外貌抽象出来，高翠兰本人和孙悟空都实现了这个接口，猪八戒访问高翠兰的时候看不出来这个是孙悟空，所以说孙悟空是高翠兰代理类。 3、买火车票不一定在火车站买，也可以去代售点。 4、一张支票或银行存单是账户中资金的代理。支票在市场交易中用来代替现金，并提供对签发人账号上资金的控制。 5、spring aop。

## 示例程序

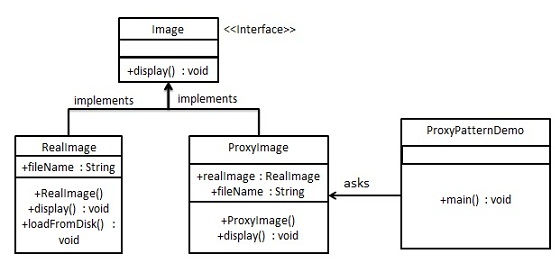
需要源码的朋友可以前往github下载：

<https://github.com/aharddreamer/chendong/tree/master/design-patterns/demo-code/design-patterns>

**程序简介**：

我们创建一个Image接口以及它的实现类，ProxyImage是一个代理类，减少RealImage对象加载的内存占用。

ProxyPatternTest 是测试代理模式的类。



**代码：**

**public interface** Image {  
 **void** display();  
}

**public class** RealImage **implements** Image {  
 **private** String **fileName**;  
  
 **public** RealImage(String fileName) {  
 **this**.**fileName** = fileName;  
 loadFromDisk(fileName);  
 }  
 @Override  
 **public void** display() {  
 System.***out***.println(**"Display "** + **fileName**);  
 }  
  
 **private void** loadFromDisk(String fileName){  
 System.***out***.println(**"Loading "** + fileName);  
 }  
}

**public class** ProxyImage **implements** Image {  
  
 **private** RealImage **realImage**;  
 **private** String **fileName**;  
  
 **public** ProxyImage(String fileName) {  
 **this**.**fileName** = fileName;  
 }  
  
 @Override  
 **public void** display() {  
 **if** (**realImage** == **null**) {  
 **realImage** = **new** RealImage(**fileName**);  
 }  
 System.***out***.println(**"======Proxy display before========="**);  
 **realImage**.display();  
 System.***out***.println(**"======Proxy display after========="**);  
 }  
}

**public class** ProxyPatternTest {  
 **public static void** main(String[] args) {  
 Image proxyImage = **new** ProxyImage(**"test.jpg"**);  
 System.***out***.println(**"代理对象第一次display（图像需要从磁盘加载）:"**);  
 proxyImage.display();  
 System.***out***.println(**"代理对象第二次display（图像不需要重新加载）:"**);  
 proxyImage.display();  
 }  
}

运行结果：

代理对象第一次display（图像需要从磁盘加载）:

Loading test.jpg

======Proxy display before=========

Display test.jpg

======Proxy display after=========

代理对象第二次display（图像不需要重新加载）:

======Proxy display before=========

Display test.jpg

======Proxy display after=========

参考：

《代理模式》菜鸟教程网站