里氏代换原则 法海捉拿白蛇新解

应用场景举例：

《白蛇传》是中国四大民间传说之一，妇孺皆知。

在大多数人的感觉和印象中，白蛇是一个善良痴情、知恩图报、温柔友善、美貌绝伦、冰雪聪明、明辨是非、救苦救难的活菩萨；而法海却是一个仗着自己的法力高强、打着降妖除魔的口号而恶意拆散许仙和白娘子这对恩爱夫妻负面形象。大多说人之所以觉得如此，主要是因为影视中的白蛇善良的无以复加。试想，如果传说和影视中的白蛇不是表现的善良，恐怕人们恨不得早些让法海去把白蛇收服呢。

但是，法海真的那么坏吗？法海真的一无是处吗？

法海不坏。法海是一个非常敬业的人！

法海多年来苦心钻研佛法、潜心修炼是为了降妖除魔、解救世人。法海的原则是凡是妖魔都要被降服。尽管你可能说，妖魔也有善良的啊。首先这个善良本身就已经加入了你主观意志。是你想象中的善良，是你强加的。另外，人和妖的世界时不同的世界，不同的世界做事和行为方式肯定是有极大的碰撞，对一个世界好的事情，对另外一个世界可能就是彻头彻尾的坏了。一句话：两个世界语言沟通有障碍。就按照你说的白蛇善良，你就在你的妖界善良就行了，干嘛非要跑到人间来。为了报恩？报恩非要嫁给许仙才算是报恩吗？报恩的方式有无数种，如果每一个收到许仙恩惠的女子都要嫁给许仙，你白蛇怎么办？人和妖界既然是两个不同的世界，就肯定有很大的不同，就肯定会有很大的摩擦。法海在发誓要收服白蛇前肯定也想过白蛇的善良，但更多的是：你在妖界善良也就罢了，为何来到人间？怎么保证你不危害世人？所以凡事出来的妖魔，法海就必然收服了。

其实法海说到底也只是非常敬业而已。逻辑步骤很简单：法海的任务是是降妖除魔、造福于众生；白蛇是妖；所以要除掉。如此敬业的人，怎能不赢的我们的掌声呢？

定义：

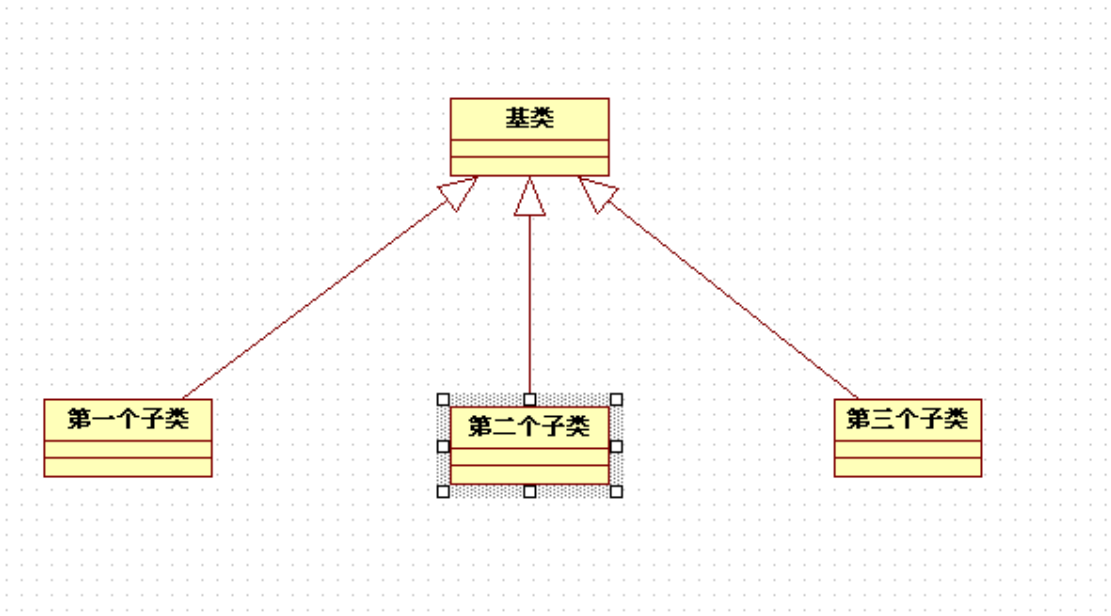
里氏代换原则（Liskov Substitution Principle）是指：一个软件实体如果使用的是基类的话，那么也一定适用于其子类，而且它根本觉察不错使用的是基类对象还是子类对象；反过来的代换这是不成立的，即：如果一个软件实体使用一个类的子类对象，那么它不能够适用于基类对象。

里氏代换原则是由麻省理工学院（MIT）计算机科学实验室的 Liskov 女士，在 1987 年的 OOPSLA 大会上发表的一篇文章《Data Abstraction and Hierarchy》里面提出来的，主要阐述了有关继承的一些原则，也就是什么时候应该使用继承，什么时候不应该使用继承，以及其中的蕴涵的原理。2002 年，软件工程大师 Robert C. Martin，出版了一本《Agile Software Development Principles Patterns and Practices》，在文中他把里氏代换原则最终简化为一句话："Subtypes must be substitutable for their base types"。也就是，子类必须能够替换成它们的基类。

里氏代换原则讲的是基类和子类的关系，只有这种关系存在的时候里氏代换原则才能够成立。

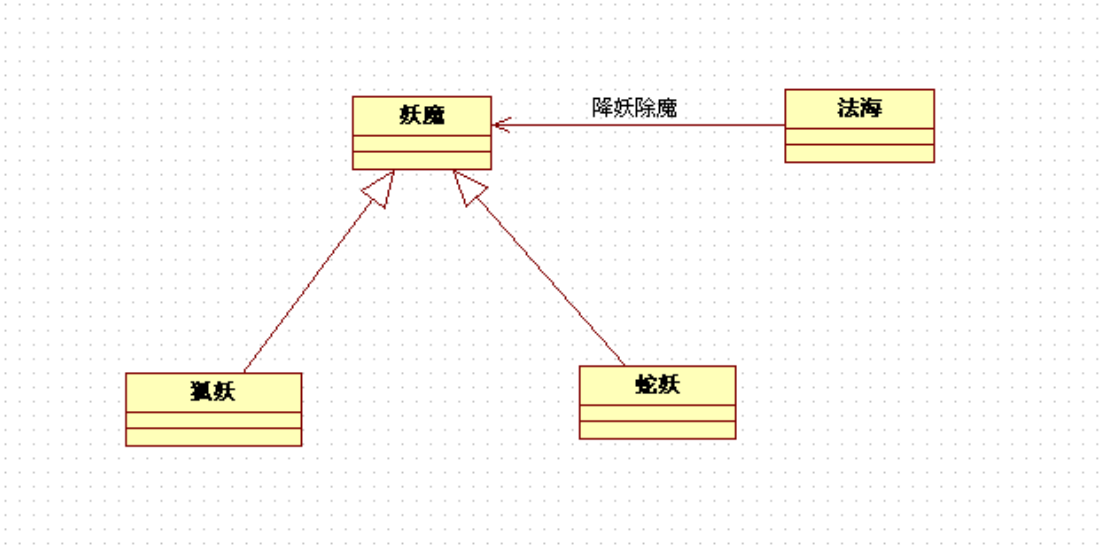
里氏代换原则是实现开放封闭原则的具体规范。这是因为：实现开放封闭原则的关键是进行抽象，而继承关系又是抽象的一种具体实现，这样 LSP 就可以确保基类和子类关系的正确性，进而为实现开放封闭原则服务。

如下图所示：

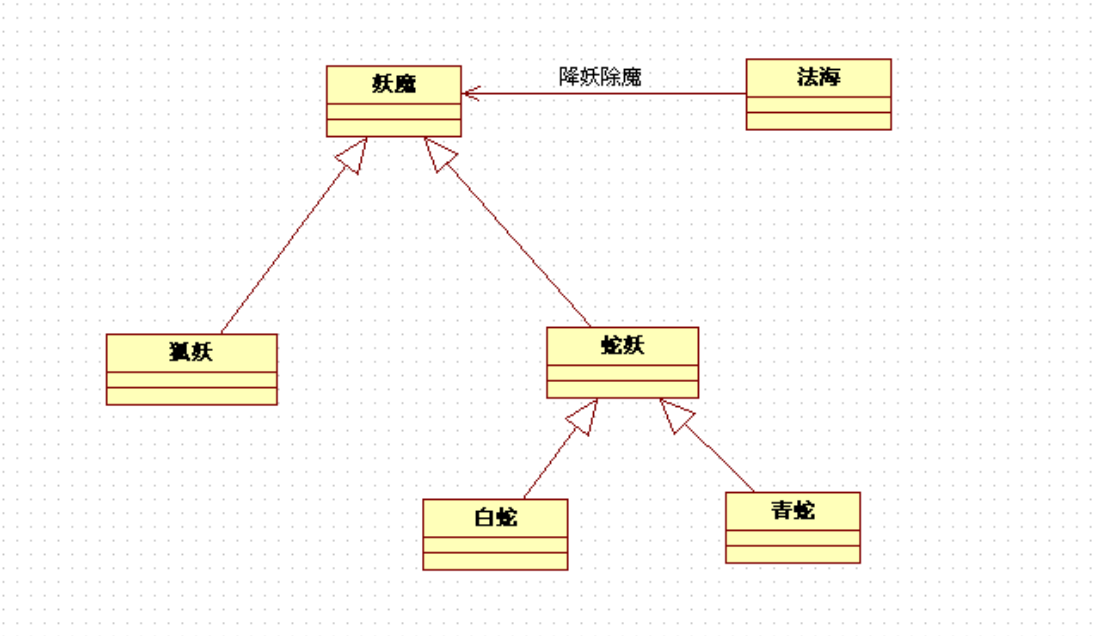


故事分析：

法海的任务是降妖除魔、造福众生。凡是妖魔现身，法海必定全力诛灭之。



而白蛇是修炼千年的蛇妖，蛇妖也是妖。



法海要做的就是不管你是什么要么，只要是要么我就要把你收服，以防再次出来危害人间。他并不用区分你是蛇妖还是狐妖，更不用管你蛇妖中的青蛇还是白色。

凡是对妖魔这个基类适用的 白蛇、青蛇就全部适用。

Java 代码实现：

妖魔的基类：

```
package com.diermeng.DesignPattern.LSP;

/*
 * 妖魔的基类
 */
```

```
public abstract class Spirit {
    /*
     * 妖魔的行为方法
     */
    public abstract void say();
}
```

蛇妖的基类

```
package com.diermeng.DesignPattern.LSP;
/*
 * 蛇妖的基类，继承妖魔，可以扩展方法属性
 */
public abstract class Snake extends Spirit {

    @Override
    public abstract void say();

}
```

白蛇类：

```
package com.diermeng.DesignPattern.LSP;
/*
 * 白蛇对蛇妖的继承实现
 */
public class WhiteSnake extends Snake {

    @Override
    public void say() {
        System.out.println("我是白蛇");
    }

}
```

建立一个测试类，代码如下：

```
package com.diermeng.DesignPattern.LSP.client;

import com.diermeng.DesignPattern.LSP.Spirit;
import com.diermeng.DesignPattern.LSP.WhiteSnake;
```

```
public class LSPTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Spirit spirit = new WhiteSnake();
        spirit.say();
    }

}
```

程序运行结果如下：

```
我是白蛇
```

已有应用简介：

这里主要分析一下 Java 编译器对里氏代换原则的支持机制。

在 Java 中如果子类覆盖了基类的方法，子类是该方法的访问权限必须是不能低于它在父类中的访问权限的。这样在才能保证在客户端程序调用父类相应方法而需要使用子类的相应的方法代替时（调用父类的方法时需要使用子类相应的方法是普遍的），不会因为子类的方法降低了访问权限而导致客户端不能在继续调用。

温馨提示：

里氏代换原则是很多其它设计模式的基础。

它和开放封闭原则的联系尤其紧密。违背了里氏代换原则就一定不符合开放封闭原则。