单一职责原则 乔峰 VS 慕容复

应用场景举例：

江湖盛传：北乔峰，南慕容。

少室山下，天下群雄云集。

“你们一起来吧，我萧峰何惧!”，一声豪情和怒吼，乔峰卷入了和慕容复、游坦之、丁春秋一决生死之战。乔峰果然不愧是天下第一豪侠，以一敌三，你来我往，打得不可开交。乔峰使出了降龙十八掌中的“亢龙有悔”，此时慕容复忙往后退，情急之下，使出了自己的绝技“游龙引凤”来化解乔峰如此强烈的进攻，此时慕容复双腿选在了亭子的柱子上，王语嫣心想：“表哥使用游龙引凤自然是不会败的了，可是面对天下众英雄的面，竟然和游坦之、丁春秋等这样的人联手对付大英雄乔峰，这也有些太不公平了，表哥，你在想些什么啊？”。此时被乔峰手下保护的段誉心急如焚，心想，这个不行，唯恐大哥有什么意外，毕竟总是乔峰神功盖世，眼前面对也是江湖上一流的高手，这样下去如何是好？突然，段誉挣脱众人的包括冲了上来，对慕容复使用激将法说：“你们两个打一个算什么英雄好汉，慕容复，有本事来和我单打啊!”。慕容复本来就对段誉一直以来对王语嫣的爱慕之意深表不满，又被他这么一激怒，怒声到：“找死!”，就像段誉扑来。此时段誉以从神仙姐姐那里学来的“凌波微步”的奇功和慕容复纠缠。此时，虚竹正在和收拾丁春秋。而乔峰正在迎战游坦之，游坦之虽然练成了很毒辣的武功，但也绝非是乔峰的对手，数招之内便被乔峰从塔顶扔了下来，结果弄到一个腿部骨折而惨败的下场！段誉白在慕容复的剑下，宁死不屈。慕容复正欲一剑了解了段誉，段正淳喝道：“休伤我儿!”挡住了慕容复的剑。慕容复非常恼怒，数招之内就把慕容复打伤在地。眼看慕容复就要杀死自己父亲，躺在地上的段誉“啊”的一声使出了大理国的绝学“六脉神剑”,几招之内，慕容复惨败，正要进一步进攻时，此时他的梦中情人王语嫣柔声叫到：“段公子手下留情啊”，这段誉，一听到王语嫣的声音，就神魂颠倒、不知所以了，慕容复乘此机会偷袭段誉，乔峰见状，一边喝道：“三弟小心”，一边瞬间出手，粉碎

了慕容复的武器，慕容复还没反应过来，就已经被乔峰高高举起在了半空中，乔峰喝道：“我萧峰大好男儿，岂能与你这等小人齐名”，随即把慕容复恨恨的抛了出去！慕容复在天下英雄面前惨败，自觉颜面无常，欲拔剑自刎，被灰衣人阻拦…

定义：

单一职责原则(Single Responsibility Principle):就一个类而言，应该仅有一个引起它变化的原因。换句话说，一个类的功能要单一，只做与它相关的事情。

如果一个完成额外的不太相关的功能或者完成其它类的功能，这就会使得一个引起一个类变化的因素太多，如果类的一处需要修改，其它和它相关连的代码都会受到影响，这就直接导致一旦系统出现了问题就难以调试困境，同时这样也非常不利于维护。

遵循单一职责原则也会给测试带来极大的方便。

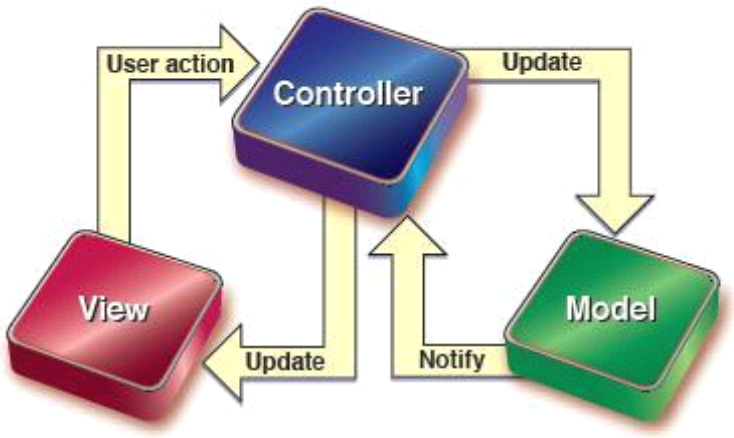
违背单一职责原则会降低类的内聚性、增强类的耦合性。

违背单一职责原则会导致错误呈现几何级数的增长，因为类之间的关联性太强，每一个类都会对其他类有影响，一个类出现错误极可能会导致其他相关联的类出现错误，而且关联类联合起来还有可能产生新的错误。

在软件开发中，人们越来越意识到单一职责原则的重要性，美工只需要负责美工界面，业务层的人员只需写好业务代码，而数据层的人员只需关注数据层的工作即可。这样每个人都以自己专程协同工作，工作效率就得到了大大的提高了。

现在软件开发的经典模式 MVC 模式，也非常好的体现了单一职责原则。MVC(Model-View-Control)就是模型、视图、控制器三层架构模式，其中 M 是指数据模型、V 是指用户界面、C 则是控制器。采用 MVC 模式使得数据和表现相分离，同一个数据层可以有不同的显示层。数据层和显示层的改变互不影响。这就非常有利于提高软件的可维护性和可复用性，同时也方便了软件的管理工作和提高软件开发效率。

如下图所示：



故事分析：

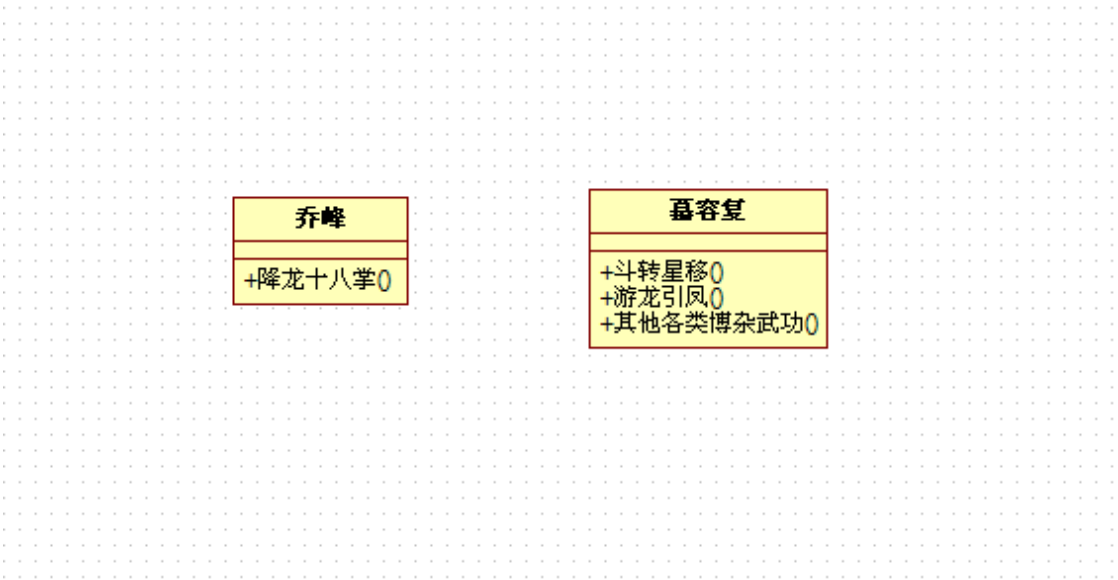
有王语嫣这个武学博士相助，同时集天下几乎所有武学与一身“以彼之道，还施彼身”的慕容复却输了，输给了懂得六脉神剑的段誉，也输给了用降龙十八掌打遍天下的乔峰。抛

开其它的原因不谈，慕容复之所以会输，是因为他虽然懂得很多武功，甚至懂得天下几乎百分之八十以上的武功，但是因为复国心切，确很少做到能够精通；从另外一个角度讲，也是修炼这么多武功害了慕容复。你想啊，一个年纪轻轻的人修炼这么多武功，怎么可能有时间去把每种武功都精通呢，而且还会累的要死。其实，慕容复就像一匹狼一样，面对着一大群样，想把每一个都吃掉！结果是可能基本咬死了每一只羊，却无暇去好好品尝和消化。而段誉和乔峰就不一样了。段誉会六脉神剑。这里我们主要分析一下乔峰。乔峰自幼被少林寺收养，刻苦勤奋，潜心修炼降龙十八掌，直至到达无招胜有招的地步。当然乔峰这样的真英雄、大豪杰自然是实战经验丰富的。就这样，当“以彼之道，还施彼身”的慕容复遇到了用降龙十八掌打遍天下无敌手的乔峰是，慕容复输了。

从设计原则的角度考虑，慕容复输是因为他违背了单一职责原则，他把太多的精力分散于各种武学之中，他迷失在了武学之中。当然，不可否认，慕容复确实是年轻有为的武学天才。但是临阵实战的时候讲究的是你真正的实力，而不是以懂得的武功的多少决定胜负的。修炼太多的武功种类必然让人太累，而且非常容易走火入魔。这就像吃饭一样，吃了太多未必是什么好事，而且几乎肯定不是好事。这是因为，一方面吃了太多，对自己的消化系统和身体都不好，容易发胖等；另外一方面讲，吃的多不一定消化的多，而且很多时候反而削弱消化能力，因为吃的太多带来了太多的负担，影响了消化系统的功能。而乔峰就不一样了，他在打好基础的情况，专注于降龙十八掌，直到能够随心所欲、运用自如。当然乔峰这样的盖世豪侠对武功本质的和武功精髓的理解也是高于慕容复的。而且，乔峰还有一个特定，就是遇强则强、越战越勇。

单一职责原则告诉我们，一个类应该只有一个引起该类变化的原因。这样有利提高类的可维护性和可复用性。如果把乔峰和慕容复比作两个类的话，引起乔峰变化的就是降龙十八掌，而且乔峰把降龙十八掌做到了极致，无论是敌是友，乔峰都可以用降龙十八掌化解。而慕容复呢，引起他变化的原因就太多了，虽然“以彼之道，还施彼身”几乎令所有江湖人士都敬畏三分，但是因为多而不精，遇到像乔峰这样的高手后，恐怕就无法“以彼之道，还施彼身”了。

这里，我们把乔峰的降龙十八掌看做是乔峰的方法，而慕容复懂的武功也看作慕容复拥有的方法，建模的图形如下：



Java 代码实现：

乔峰的类：

```
package com.diermeng.designPattern.SRP;
/*
 * 乔峰
 */
public class Qiaofeng {
    /*
     * 姓名
     */
    String name;

    /*
     * 无参构造方法
     */
    public Qiaofeng() {}
    /*
     * 带参数的构造方法
     */
    public Qiaofeng(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    /*
     * 乔峰的功夫绝技展现
     */
    public void gongfu()
    {
        if(this.getName()!=null){
            System.out.println("我是"+this.getName()+" 使用降龙十八掌，无论是敌是友我都可以应对自如,这都要感谢我遵循了单一职责原则！");
        }
        else{
            System.out.println("使用降龙十八掌，无论是敌是友我都可以应对自如，这都要感谢我遵循了单一职责原则！");
        }
    }
```

```
}
```

慕容复的类图：

```
package com.diermeng.designPattern.SRP;
/*
 * 慕容复
 */
public class MuRongfu {
    /*
     * 姓名
     */
    String name;

    /*
     * 无参构造方法
     */
    public MuRongfu() {}
    /*
     * 带参数的构造方法
     */
    public MuRongfu(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    /*
     * 慕容复的功夫展现
     */
    public void gongfu()
    {
        if(this.getName()!=null){
            System.out.println("我是"+this.getName()+" 我会很多武功，对绝大多数人来说我都可以做到以彼之道还施彼身，但是如果遇到乔峰这样的高手，我就Over了，这都是没有遵循单一职责原则惹的祸！");
```

```
        }
        else{
            System.out.println(" 我会很多武功，对绝大多数人来说我都可以做到以彼之道还施彼身，但是如果遇到乔峰这样的高手，我就Over了，这都是没有遵循单一职责原则惹的祸！");
        }
    }
}
```

建立一个测试类，代码如下：

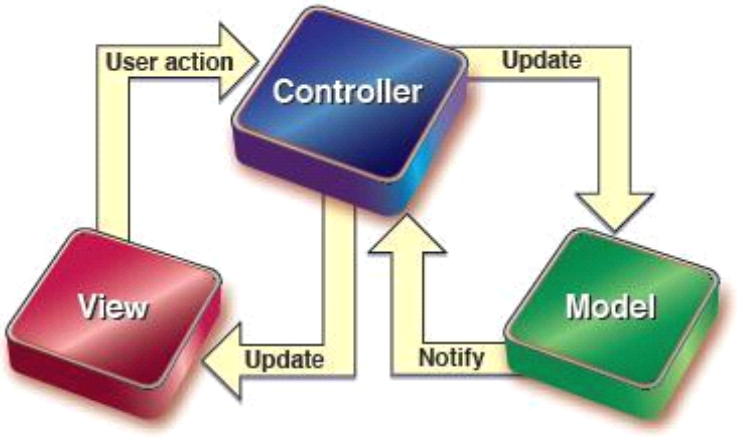
```
package com.diermeng.designPattern.SRP.client;

import com.diermeng.designPattern.SRP.MuRongfu;
import com.diermeng.designPattern.SRP.Qiaofeng;

public class GongfuTest {

    /**
     * @param args
     */
    public static void main(String[] args) {

        Qiaofeng qiaofeng = new Qiaofeng("乔峰");
        MuRongfu muRongfu = new MuRongfu("慕容复");

        qiaofeng.gongfu();
        muRongfu.gongfu();
    }

}
```

程序运行结果如下：

```
我是乔峰 使用降龙十八掌，无论是敌是友我都可以应对自如,这都要感谢我遵循了单一职责原则！
我是慕容复 我会很多武功，对绝大多数人来说我都可以做到以彼之道还施彼身，但是如果遇到乔峰这样的高手，我就Over了，这都是没有遵循单一职责原则惹的祸！
```

已有应用简介：

我们这里已经以 MVC 模式为例来分析单一职责原则的应用。



模型车、视图层、控制层各司其责、相互独立，一个模型可以有多个视图，一个视图可以有多个控制器，同样的一个控制器也可以由多个模型。MVC 基本的处理流程如下： 用户与视图交互，视图接受并反馈用户的动作；视图把用户的请求传给相应的控制器，有控制器决定调用哪个模型，然后由模型调用相应的业务逻辑对用户的请求进行加工处理，如果需要返回数据，模型会把相应的数据返回给控制，由控制器调用相应的视图，最终由视图格式化和渲染返回的数据，以一种对用户尽可能友好的方式展现给用户。

温馨提示：

专注于一件事情，专注于一切事情。

如果这个世界每个人都专注于做一件事情，并把这件事情做到极致，世界会是怎样的一种和谐美好呢？

遵循单一职责原则的乔峰使少林寺的扫地僧也不禁发出感叹：“降龙十八掌果然天下第一！”

树立一个高远的目标，然后拼尽一切力量的去实现这个目标。

愿单一职责原则保佑您！