合成聚合复用原则 刘邦 VS 韩信

应用场景举例：


VS


一次，刘邦闲着没事，又想起了故人韩信，于是打开 QQ 想和韩信聊天，正巧韩信也在，开始寒暄了几句，就在不自觉中进入严肃的问题，刘邦问：“韩兄觉得如果是我带兵，最多能够带多少呢？”，韩信立即回复说：“十万”，刘邦心想本王竟然只能带十万，那你韩信又能够带多少呢，于是暂时按捺了心中的郁闷，很客气的问道：“那请问韩兄最多能够带多少呢”，韩信又立即回复道：“我啊，那自然是越多越好啦”，刘邦看到此言顿时大怒，而且语气还极其的傲慢，还“啊，啦”的，刘邦心想：“虽然你韩信带兵打仗有道，天下皆知，但是这样对本王说话也太过分了吧”，刘邦正要发飙，随即停了一下，深谙世道的刘邦问了一句：“将军神勇盖世，带兵百万，却为何会在我领导下呢？”，刘邦想：“好你个韩信，叫了你几声韩兄你就不知道自己是谁了，如果回答不上来，或是回答不好，看我如何收拾你！”，等了大约三秒钟，QQ 闪了一下，只见上面赫然写道：“陛下虽不善统兵，却善御将”。刘邦大悦！

定义：

合成聚合复用原则（Composite Aggregate Reuse Principle,简称为 CARP）经常又被人们称为合成复用原则（Composite Reuse Principle,简称为 CRP）.合成聚合复用原则是指在一个新的对象中使用原来已经存在的一些对象，是这些原来已经存在的对象称为新对象的一部分，新的对象通过向这些原来已经具有的对象委派相应的动作或者命令达到复用已有功能的目的。

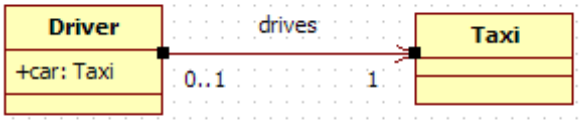
合成复用原则跟简洁的表述是：要尽量使用合成和聚合，尽量不要使用继承。

聚合（Aggregation）是关联关系的一种，用来表示一种整体和部分的拥有关系。整体持有对部分的引用，可以调用部分的能够被访问的方法和属性等，当然这种访问往往是对接口和抽象类的访问。作为部分可以可以同时被多个新的对象引用，同时为多个新的对象提供服务。

合成（Composition）也是关联关系的一种，但合成是一种比聚合强得多的一种关联关系。在合成关系里面，部分和整体的生命周期是一样的。作为整体的新对象完全拥有对作为部分的支配权，包括负责和支配部分的创建和销毁等，即要负责作为部分的内存的分配和内存释放等。从这里也可以看出来，一个合成关系中的成员对象是不能喝另外的一个合成关系共享的。

为何“要尽量使用合成和聚合，尽量不要使用继承”呢？这是因为：第一，继承复用破坏包装，它把超类的实现细节直接暴露给了子类，这违背了信息隐藏的原则；第二：如果超类发生了改变，那么子类也要发生相应的改变，这就直接导致了类与类之间的高耦合，不利于类的扩展、复用、维护等，也带来了系统僵硬和脆弱的设计。而是用合成和聚合的时候新对象和已有对象的交互往往是通过接口或者抽象类进行的，就可以很好的避免上面的不足，而且这也可以让每一个新的类专注于实现自己的任务，符合单一职责原则。

聚合关系的示意图如下所示：



聚合关系的示意图如下所示：

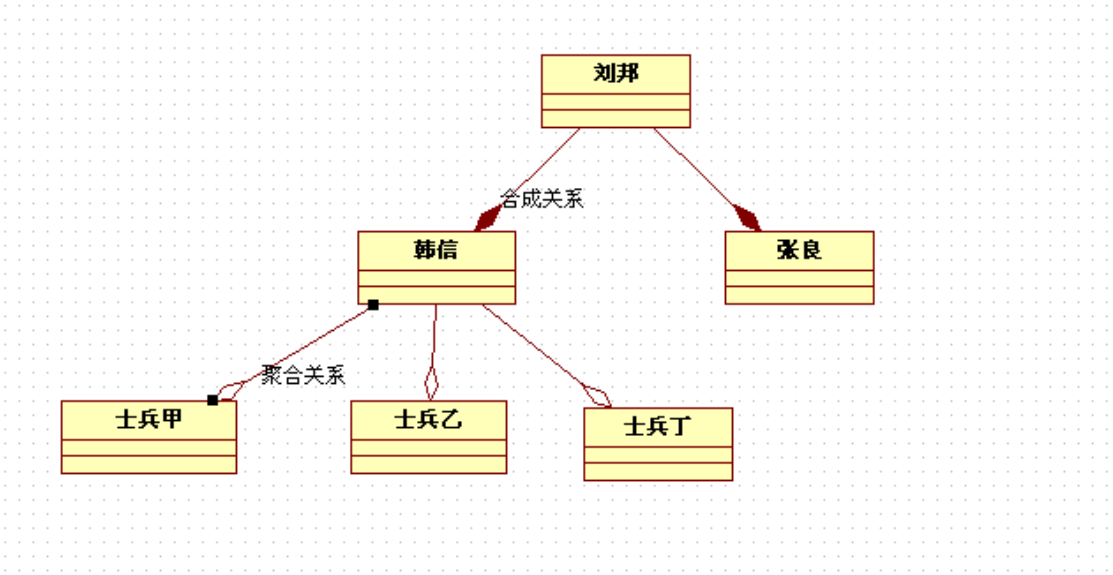
故事分析：

“韩信带兵，多多益善”，韩信之所以有很大的影响力，一方面归功他建立了很大的功勋；另外一方面是因为他手握重兵。建立很大的功勋甚至是功高盖主，这就直接导致韩信在军队中深得人心，毕竟，作为士兵，很少有不愿意追随一个在战场上无往不胜的领袖的；手握重兵，有合成聚合的原则来说就是对自己统领的士兵将士保持有引用，就是聚合。韩信手

中聚合有一大批能征善战的人，随时听从韩信的调遣，这是不能不让刘邦戒备的。而现在，又说了一个自己带兵多多益善的话，这怎能不让刘邦感觉生气呢？

而“普天之下，莫非王土；四海之内，莫非王臣”，韩信拥有士兵，刘邦却拥有天下，何况韩信也并非拥有全部士兵。“君叫臣死，臣不得不死”，刘邦拥有对天下苍生的生杀大权。韩信很清楚，“陛下虽不善统兵，却善御将”。刘邦是拥有一种比韩信更强的“拥有”关系，可以主宰生死，同时也可以主宰韩信的生死。这就相当于合成关系。更重要的是韩信也是合成关系的一个成员，是刘邦的将领。所以说刘邦比韩信更加强悍！

如下图所示：



Java 代码实现：

新建大臣的接口：

```
package com.diermeng.designPattern.CARP;
/*
 * 大臣的接口
 */
public interface Minister {
    /*
     * 大臣能够执行的动作
     */
    public void duty();
}
```

士兵的接口

```
package com.diermeng.designPattern.CARP;
/*
 * 士兵的接口
 */
public interface Soldier {
```

```
    /*
     * 士兵能够执行的动作
     */
    public void duty();
}
```

韩信对大臣接口的实现

```
package com.diermeng.designPattern.CARP.impl;

import com.diermeng.designPattern.CARP.Minister;
import com.diermeng.designPattern.CARP.Soldier;
/*
 * 韩信对大臣接口的实现
 */
public class Hanxin implements Minister {
    //对士兵的聚合关系
    Soldier[] soldiers;

    /*
     * 无参构造方法
     */
    public Hanxin() {}
    /*
     * 有士兵参数的构造方法
     */
    public Hanxin(Soldier[] soldiers) {
        super();
        this.soldiers = soldiers;
    }

    /*
     * 获取士兵的集合
     */
    public Soldier[] getSoldiers() {
        return soldiers;
    }

    /*
     * 设置士兵的集合
     */
```

```
    public void setSoldiers(Soldier[] soldiers) {
        this.soldiers = soldiers;
    }
    /*
     * 韩信的职能
     * @see com.diermeng.designPattern.CARP.Minister#duty()
     */
    public void duty() {
        System.out.println("我是刘邦的大臣，永远忠实于刘邦");

    }

}
```

士兵 A 对士兵接口的实现

```
package com.diermeng.designPattern.CARP.impl;

import com.diermeng.designPattern.CARP.Soldier;
/*
 * 士兵A
 */
public class SoldierA implements Soldier {
    /*
     * 士兵A的职责
     * @see com.diermeng.designPattern.CARP.Soldier#duty()
     */
    public void duty() {
        System.out.println("我是韩信的士兵A");

    }

}
```

士兵 B 对士兵接口的实现

```
package com.diermeng.designPattern.CARP.impl;

import com.diermeng.designPattern.CARP.Soldier;
/*
 * 士兵B
 */
public class SoldierB implements Soldier {
    /*
```

```
     * 士兵B的职责
     * @see com.diermeng.designPattern.CARP.Soldier#duty()
     */
    public void duty() {
        System.out.println("我是韩信的士兵B");

    }

}
```

刘邦类

```
package com.diermeng.designPattern.CARP.impl;

import com.diermeng.designPattern.CARP.Minister;

/*
 * 刘邦类
 */
public class Liubang {
    //拥有大臣
    Minister[] minister;

    /*
     * 无参构造方法
     */
    public Liubang() {}
    /*
     * 把大臣的数组作为参数传入构造方法
     */
    public Liubang(Minister[] minister) {
        super();
        this.minister = minister;
    }

    /*
     * 获取大臣的集合
     */
    public Minister[] getMinister() {
        return minister;
    }

    /*
     * 设置大臣的集合
```

```
     */
    public void setMinister(Minister[] minister) {
        this.minister = minister;
    }

    /*
     * 刘邦的职能
     */
    public void duty()
    {
        System.out.println("我是皇帝，普天之下，莫非王土；四海之内，莫非王臣");
    }
}
```

建立一个测试类，代码如下：

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

import com.diermeng.designPattern.CARP.Minister;
import com.diermeng.designPattern.CARP.Soldier;
import com.diermeng.designPattern.CARP.impl.Hanxin;
import com.diermeng.designPattern.CARP.impl.Liubang;
import com.diermeng.designPattern.CARP.impl.SoldierA;
import com.diermeng.designPattern.CARP.impl.SoldierB;

/*
 * 测试类的客户端
 */
public class CARPClient {
    public static void main(String[] args)
    {
        //声明并实例化士兵A
        Soldier soldierA = new SoldierA();
        //声明并实例化士兵B
        Soldier soldierB = new SoldierB();
        //构造士兵数组
        Soldier[] soldiers = {soldierA,soldierB};

        //声明并实例化韩信，同时传入士兵数组
        Minister hanxin = new Hanxin(soldiers);

        //构造大臣数组
```

```
        Minister[] minister = {hanxin};
        //声明并实例化刘邦，同时传入大臣数组
        Liubang liubang = new Liubang(minister);

        liubang.duty();

        //循环输出大臣
        for(Minister aminister:liubang.getMinister()){
            aminister.duty();
        }

    }
}
```

程序运行结果如下：

```
我是皇帝，普天之下，莫非王土；四海之内，莫非王臣
我是刘邦的大臣，永远忠实于刘邦
```

已有应用简介：

对于面向对象的软件系统而言，如何提高软件的可维护性和可复用性始终是一个核心问题。合成聚合复用原则的合理而充分的使用时非常有利于构建可维护、可复用、可扩展和灵活性好的软件系统。合成聚合复用原则作为一种构造优质系统的手段几乎可以应用到任何环境中去。对于已有的应用这里就不在赘述啦。

温馨提示：

合成聚合复用原则虽然几乎可以应用到任何环境中去，但是这个原则也有自己的缺点。因为此原则鼓励使用已有的类和对象来构建新的类的对象，这就导致了系统中会有很多的类和对象需要管理和维护，从而增加系统的复杂性。

同时，也不是说在任何环境下使用合成聚合复用原则就是最好的，如果两个类之间在符合分类学的前提下有明显的“IS-A”的关系，而且基类能够抽象出子类的共有的属性和方法，而此时子类有能通过增加父类的属性和方法来扩展基类，那么此时使用继承将是一种更好的选择。