

(一) 针对 PHP 中大量的 if 判断的重构

1) 案例一

一个 if 嵌套

```
<?php
$table = 'test_order';
if ($table === 'test_order' OR
    $table === 'test_client' OR
    $table === 'test_area'
){
    echo $table;
} else {
    die('err');
}
```

用数组解决解决

```
<?php
$table = 'test_order';
$arr = array(
    'test_order' => "",
    'test_area' => "",
    'test_client' => "",);
if (isset($arr[$table])) {
    echo $table;
} else {
    die('err');
}
```

很明显用数组代码更少，更重要的是，数据可以作为值来传递，而在 php 中，代码是不能作为数据结构来传递的。当然，有时候用数组会损失一点点效率，但不是系统瓶颈的话，是不值得去优化的——过早优化之所以不好，是因为很多时候优化 约等于 “写死”，也就失去了灵活性。

2) 案例二

大段的要死人的 if 语句

```
<?php
$int_pattern = '3';

if ($inkai_pattern == "1") {
    echo "";
} elseif ($inkai_pattern == "2") {
    echo "";
} elseif ($inkai_pattern == "3") {
    echo "";
}
```

```
} elseif ($inkai_pattern == "4") {  
    echo "";  
} elseif ($inkai_pattern == "5") {  
    echo "";  
} elseif ($inkai_pattern == "6") {  
    echo "";  
}  
}
```

用 switch 来解决问题

```
<?php  
$int_pattern = '3';  
switch ($_pattern)  
{  
    case 1:  
        echo "";  
        break ;  
    case 2:  
        echo "";  
        break ;  
    case 3:  
        echo "";  
        break;  
    case 4:  
        echo "";  
        break;  
    case 5:  
        echo "";  
        break;  
    case 6:  
        echo "";  
        break;  
    case 101:  
        echo "";  
        break;  
}
```

3) 案例三

状况：你有一个复杂的条件（if-else if-else）语句，那么从 **if**、**else if**、**else** 三个段落中分别提炼出函数。

```
1 <?php
2 class Order{
3     private $_winterRate = 0.1;
4     private $_winterServiceCharge = 10;
5     private $_summerRate = 0.2;
6
7     public function charge($quantity){
8         $charge = 0;
9         if(date('F') < 7 || date('F') > 9) // No Summer
10            $charge = $quantity*$this->_winterRate + $this->_winterServiceCharge;
11        else
12            $charge = $quantity * $this->_summerRate;
13        return $charge;
14    }
15 }
16
17 // Callback
18 $order = new Order();
19
20 var_dump($order->charge(20));
```

Decompose Conditional
Decompose Conditional (分解条件式)

如果发现嵌套的 (nested) 条件逻辑，我通常会先观察是否可以使用 *Conditional with Guard Clauses* (250)。如果不行，才开始分解其中

范例 (Example)
把每个分支的判断条件提炼到一个独立的函数中
假设我要计算购买某样商品的总价 (总价=数量*单价)，而这个商品的单价是不同的：



```
1 <?php
2 class Order{
3     private $_winterRate = 0.1;
4     private $_winterServiceCharge = 10;
5     private $_summerRate = 0.2;
6
7     public function charge($quantity){
8         $charge = 0;
9         if($this->_notSummer())
10            $charge = $this->_summerCharge($quantity);
11        else
12            $charge = $this->_winterCharge($quantity);
13        return $charge;
14    }
15
16    private function _notSummer(){
17        return date('F') < 7 || date('F') > 9;
18    }
19
20    private function _summerCharge($quantity){
21        return $quantity*$this->_summerRate;
22    }
23
24    private function _winterCharge($quantity){
25        return $quantity * $this->_winterRate + $this->_winterServiceCharge;
26    }
27 }
28
29 // Callback
30 $order = new Order();
31 var_dump($order->charge(20));
```

如果发现嵌套的 (nested) 条件逻辑，我通常会先观察是否可以使用 *Conditional with Guard Clauses* (250)。如果不行，才开始分解其中

看一段注释那样清晰明白。

范例 (Example)
假设我要计算购买某样商品的总价 (总价=数量*单价)，而这个商品的单价是不同的：

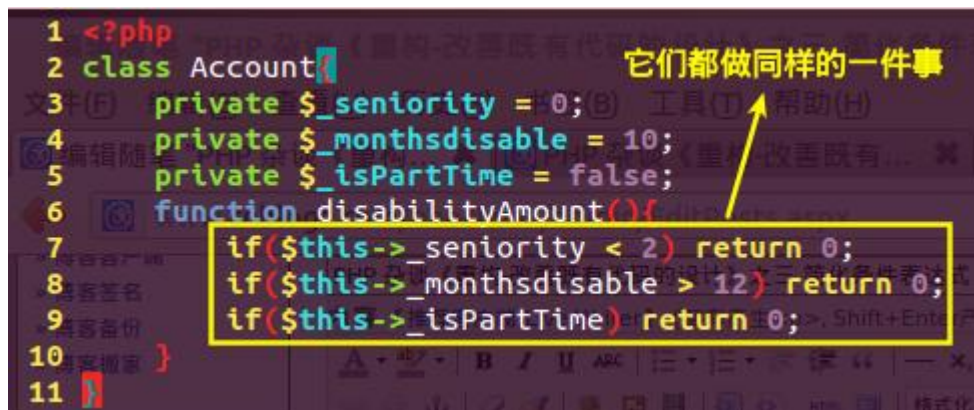
我把每个分支的判断条件都提炼到一个独立函数中，如下：

4) 案例三

状况：你有一些条件测试，都得到相同的结果，那么将这些测试合并为一个条件式，并将这个条件提炼称为一个独立的函数。

动机：1、合并后的条件代码会告诉你“实际上只有一次条件检查，只不过有数个并列条件需要检查而已”，——使检查的用意更清晰。

2、为 Extract Method 做好准备。——将检查条件提炼成一个独立函数，对于理清代码意义非常有用。它把描述“做什么”的语句换成了“为什么这样做”。



```
1 <?php
2 class Account{
3     private $_seniority = 0;
4     private $_monthsdisable = 10;
5     private $_isPartTime = false;
6     function disabilityAmount(){
7         if($this->_seniority < 2) return 0;
8         if($this->_monthsdisable > 12) return 0;
9         if($this->_isPartTime) return 0;
10    }
11 }
```

它们都做同样的一件事



```
1 <?php
2 class Account{
3     private $_seniority = 0;
4     private $_monthsdisable = 10;
5     private $_isPartTime = false;
6     function disabilityAmount(){
7         if($this->_seniority < 2 || $this->_monthsdisable > 12 || $this->_isPartTime) return 0;
8     }
9 }
10 }
```

logical-OR



```
1 <?php
2 class Account{
3     private $_seniority = 0;
4     private $_monthsdisable = 10;
5     private $_isPartTime = false;
6     function disabilityAmount(){
7         if($this->_isNotEligibleForDisability()) return 0;
8     }
9     private function _isNotEligibleForDisability(){
10         return $this->_seniority < 2 || $this->_monthsdisable > 12 || $this->_isPartTime;
11     }
12 }
13 }
```

以函数名称表达该语句所检查的条件。

条件语句的“合并理由”也同时指出了“不要合并”的理由：如果你认为你的这些检查的确彼此独立，的确不应该被视为同一次检查，那么就on不要使用本项重构。因为在这种情况下，你的代码已经清楚表达出自己的意义。

5) 案例五

状况：函数中的条件逻辑使人很难看清正常的执行路径，那么使用卫语句（Guard Clauses）表现所有特殊情况。

```
public function getPayAmount(){
    $result = 0;
    if($this->_isDead) $result = $this->_deadAmount();
    else{
        if($this->_isSeparated) $result = $this->_seperatedAmount();
        else{
            if($this->_isRetired) $result = $this->_retiredAmount();
            else $result = $this->_normalPayAmount();
        }
    }
    return $result;
}
```

9.5 Replace Nested Conditional with Guard Clauses



```
public function getPayAmount(){
    if($this->_isDead) return $this->_deadAmount();
    if($this->_isSeparated) return $this->_seperatedAmount();
    if($this->_isRetired) return $this->_retiredAmount();
    return $this->_normalPayAmount();
}
```

条件式的两种形式：

- 1、所有分支都属于正常行为：使用[if ... else..]
- 2、条件式极其罕见：应该单独检查该条件，并在该条件为真时，立刻从函数中返回。——这样的单独检查常常被称为“卫语句”

Replace Nested Conditional with Guard Clauses 精髓：给某一分支以特别重视。

6) 案例六

你手上有个表达式，它根据对象型别的不同而选择不同的行为，那么将这个条件式的每个分支放进一个 **subclass** 内的覆写函数中，然后将原始函数声明为抽象函数。

```
1 <?php
2 class Entry {
3     private $_url = array('tv' => 'www.test.com/tv', 'movie' => 'www.test.com/movie');
4     public function getRank($type){
5         $result = array();
6         switch($type){
7             case '电影':
8                 $url = $this->_url('movie');
9                 $areas = array('大陆', '香港', '台湾', '美国');
10                $titles = array('全部', '大陆剧', '港澳剧');
11                $links = array('http://.../index_91.html',
12                               'http://.../index_91.html', 'http://.../index_91.html'
13                );
14                foreach($areas as $k => $area){
15                    $ranking[$k]->title = $titles[$k];
16                    $ranking[$k]->link = $links[$k];
17                    $ranking[$k]->area = $areas[$k];
18                }
19                break;
20             case '电视剧':
21                 $url = $this->_url('tv');
22                 $areas = array('大陆', '美国');
23                 $titles = array('全部', '港澳剧');
24                 $links = array(
25                     'http://.../index_92.html', 'http://.../index_92.html'
26                 );
27                 foreach($areas as $k => $area){
28                     $ranking[$k]->title = $titles[$k];
29                     $ranking[$k]->link = $links[$k];
30                     $ranking[$k]->area = $areas[$k];
31                 }
32                break;
33             default:
34                break;
35        }
36        return $result;
37    }
38 }
```

此代码的坏味道：

- 1、它太长，当视频有新类型的时候，它会变得更长。
- 2、它明显做了不止一件事。
- 3、它违反了单一权责原则，因为它有好几个修改它的理由。
- 4、它违反了开放闭合原则，因为每当添加新类型时，必须修改它。不过最麻烦的可能是到处皆有类似结构(`_get 类型名 Rank()`)的函数。



```

1  <?php
2  class Entry{
3  system public function getRank($type){
4      $result = array();
5      switch($type){
6  客签名 case '电影':
7  客备份 $rank = new MovieRank();
8  客搬家 $result = $rank->getRank();
9      break;
10 分 类 case '电视剧':
11      $rank = new TelevisionRank();
12      $result = $rank->getRank();
13      break;
14      default:
15          break;
16      }
17      return $result;
18  }
19 }
20
21 abstract class Supper_Rank{
22     protected $_url = array('tv' => 'www.test.com/tv', 'movie' => 'www.test.com/movie');
23     abstract protected function getRank();
24 }
25
26 class MovieRank extends Supper_Rank{
27     public function getRank(){
28         $url = $this->_url('movie');
29         $areas = array('大陆', '香港, 台湾', '美国');
30         $titles = array('全部', '大陆剧', '港澳剧');
31         $links = array('http://.../index_91.html',
32             'http://.../index_91.html', 'http://.../index_91.html'
33         );
34         foreach($areas as $k => $area){
35             $ranking[$k]->title = $titles[$k];
36             $ranking[$k]->link = $links[$k];
37             $ranking[$k]->area = $areas[$k];
38         }
39         return $ranking;
40     }
41 }
42
43 class TelevisionRank extends Supper_Rank{
44     public function getRank(){
45         $url = $this->_url('tv');
46         $areas = array('大陆', '美国');
47         $titles = array('全部', '港澳剧');
48         $links = array(
49             'http://.../index_92.html', 'http://.../index_92.html'
50         );
51         foreach($areas as $k => $area){
52             $ranking[$k]->title = $titles[$k];
53             $ranking[$k]->link = $links[$k];
54             $ranking[$k]->area = $areas[$k];
55         }
56         return $ranking;
57     }
58 }

```

(二) 提取函数

状况：我看见一个过长的函数或者需要一段注释才能让人理解用途的代码，那么将这段代码放进一个独立函数中，并让函数名称解释改函数的用途。

```
1 <?php
2 class ClassName{
3     private $_order;
4     private $_name;
5
6     public function printOwing(){
7         $e = $_order.elements();
8         $outstanding = 0.0;
9
10        // Print Banner
11        echo "*****\n";
12        echo "*****Customer Owe*****\n";
13        echo "*****\n";
14
15        // Calculate $outstanding
16        while($e.hasMoreElements()){
17            $e = $e.nextElement();
18            $outstanding += $e.getAmount();
19        }
20
21        // Print Details
22        echo "name: {$_name}\n";
23        echo "amount: $outstanding\n";
24        echo "*****\n";
25    }
26
27 }
```

Extract Method

```
1 <?php
2 class ClassName{
3     private $_order;
4     private $_name;
5
6     public function printOwing(){
7         $this->_printBanner();
8         $outstanding = $this->_getOutStanding();
9         $this->_printDetails($outstanding);
10    }
11
12    private function _printBanner(){
13        echo "*****\n";
14        echo "*****Customer Owe*****\n";
15        echo "*****\n";
16    }
17
18    private function _getOutStanding(){
19        $e = $_order.elements();
20        $outstanding = 0.0;
21        while($e.hasMoreElements()){
22            $e = $e.nextElement();
23            $outstanding += $e.getAmount();
24        }
25        return $outstanding;
26    }
27
28    private function _printDetails($outstanding){
29        echo "name: {$_name}\n";
30        echo "amount: $outstanding\n";
31        echo "*****\n";
32    }
33 }
```

动机：

简短而有良好命名的函数：——finely grained

- 1、复用机会大。
- 2、函数读起来像读一系列 comments。
- 3、函数覆写容易。

重点：函数长度关键在于函数名称和函数本体之间的语义距离。如果提炼动作可以强化代码的清晰度，那么就去做。作法：

1、创建新函数，根据函数的意图命名——以它“做什么”命名，而不是以它“怎样做”命名。=》即使 Extract Function 非常简单，例如只是消息或函数调用，只要新 Function 能够以更好方式昭示代码意图，你也应该提炼它。但如果你想不出更有意义的名称，就别动它。

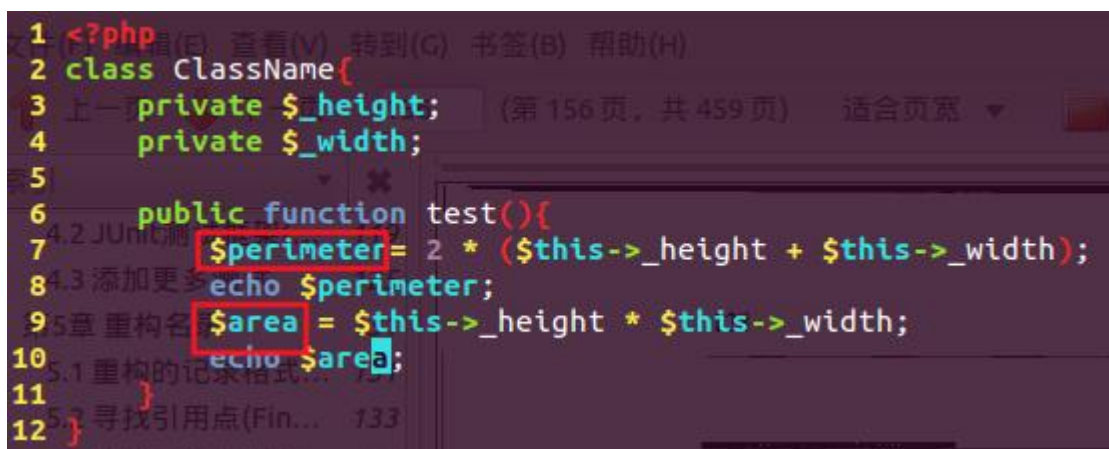
- 2、将 Extract 的代码从 Source Function 中 Move 到 New Function 中。

(三) 分离临时变量

某个临时变量被赋值超过一次，它既不是循环变量，也不是集合变量。那么针对每次赋值，创建一个独立的，对应的临时变量。



```
1 <?php
2 class ClassName{
3     private $_height;
4     private $_width;
5
6     public function test(){
7         $temp = 2 * ($this->_height + $this->_width);
8         echo $temp;
9         $temp = $this->_height * $this->_width;
10        echo $temp;
11    }
12 }
```



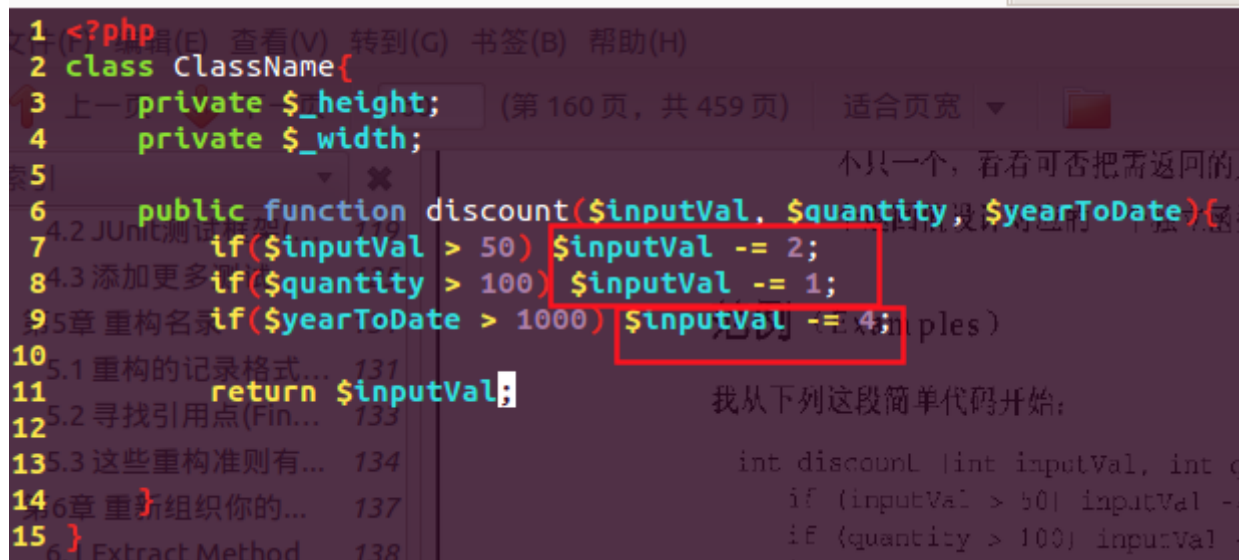
```
1 <?php
2 class ClassName{
3     private $_height;
4     private $_width;
5
6     public function test(){
7         $perimeter = 2 * ($this->_height + $this->_width);
8         echo $perimeter;
9         $area = $this->_height * $this->_width;
10        echo $area;
11    }
12 }
```

动机：

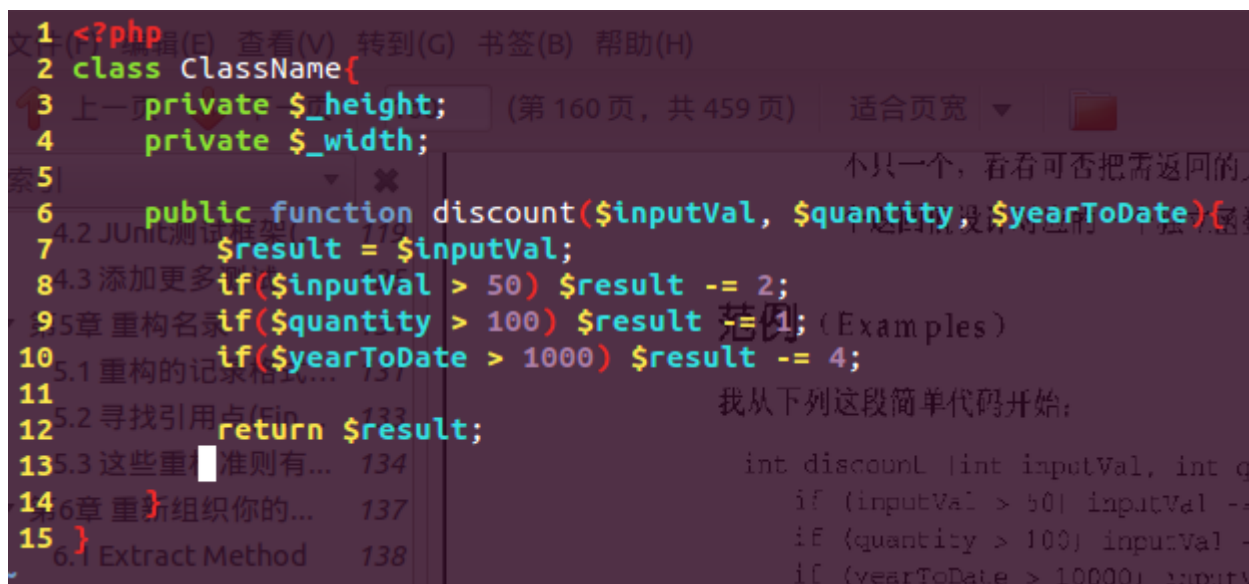
- 1、如果临时变量承担多个责任，它就应该被替换为多个临时变量。每个变量只承担一个责任。
- 2、同一个临时变量承担两件不同的事情，会令 review 变得糊涂。

(四) 移除参数变量

如果你的代码对参数进行赋值，那么以一个临时变量取代该参数的位置。



```
1 <?php
2 class ClassName{
3     private $_height;
4     private $_width;
5
6     public function discount($inputVal, $quantity, $yearToDate){
7         if($inputVal > 50) $inputVal -= 2;
8         if($quantity > 100) $inputVal -= 1;
9         if($yearToDate > 1000) $inputVal -= 4;
10
11     return $inputVal;
12 }
```



```
1 <?php
2 class ClassName{
3     private $_height;
4     private $_width;
5
6     public function discount($inputVal, $quantity, $yearToDate){
7         $result = $inputVal;
8         if($inputVal > 50) $result -= 2;
9         if($quantity > 100) $result -= 1;
10        if($yearToDate > 1000) $result -= 4;
11
12    return $result;
13 }
```

(五) 移动方法

如果一个类中的方法与另一个类有很多交流，那么我们就在另一个类中建立一个有类似功能的新函数，将旧函数变成一个单纯的 **Delegating Method**，或是将旧函数移除。

```

<?php 一页 下一页 173 (第 173 页, 共 459 页)
class Class1{
    private $_class2;
    public function __construct(){
        $this->_class = new Class2();
    }
    public function method(){
        $this->_class->t1();
        $this->_class->t2();
        $this->_class->t3();
    }
}

class Class2{
    public function t1(){
        echo "t1()\n";
    }
    public function t2(){
        echo "t2()\n";
    }
    public function t3(){
        echo "t3()\n";
    }
}

// 测试代码
$class = new Class1();
$class->method();

```



```

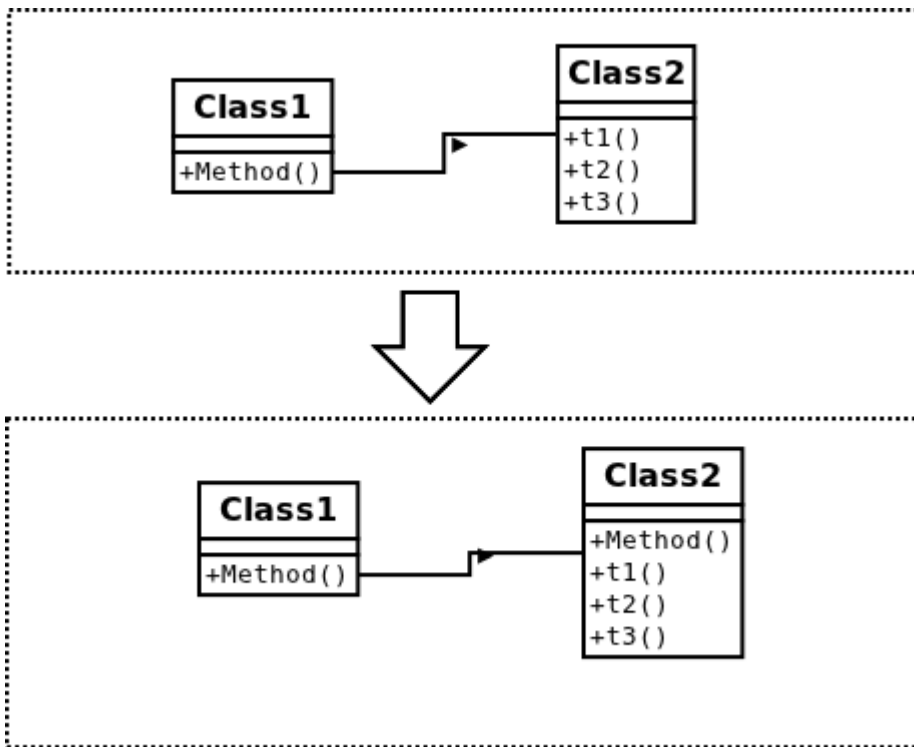
<?php 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助
class Class1{
    private $_class2;
    public function __construct(){
        $this->_class2 = new Class2();
    }
    public function method(){
        $this->_class2->method();
    }
}

class Class2{
    public function method(){
        $this->t1();
        $this->t2();
        $this->t3();
    }
    public function t1(){
        echo "t1()\n";
    }
    public function t2(){
        echo "t2()\n";
    }
    public function t3(){
        echo "t3()\n";
    }
}

// 测试代码
$class = new Class1();
$class->method();

```

类图:



动机:

- 1、如果一个类与另一个类有高度耦合，我就会 Move Method。——class 更简单，更干净利落的实现系统交付的任务。
- 2、移动一些值域，就要检查是否使用另一个类的次数必使用所驻对象的次数还多。