WORKS ✓

```
1   USE_RM = 1
2
3   if USE_RM:
4       from RM import RM1
5       RM = RM1()
6       RM.ER("xy", "onetoone", "directional")
7
8       class _Component(object):
9           def _setThing(self, thing):          RM.R(self, thing, "xy")
10          def _getThing(self):            return RM.P(self, "xy")
11          def _getBack(self):             return RM.B(self, "xy")
12          thing = property(_getThing, _setThing)
13          back = property(_getBack)
14  else:
15      class _Component(object):
16          def __init__(self):
17              self.thing = None
18              self.back = None
19
20  class Component(_Component):
21      def Do(self):                           # virtual
22          raise 'Do not implemented'
23
24  class Thing(Component):
25      def Do(self):
26          return 'hi'
27
28  class Decorator(Component):
29      pass
30
31  class Decorator1(Decorator):
32      def Do(self):
33          return '** ' + self.thing.Do() + ' **'
34
35  class Proxy(Component):
36      def __init__(self):
37          Component.__init__(self)
38          self.thing = Thing()
39
40      def Do(self):
41          assert self.thing, 'Oops, have lost the real thing object'
42          return self.thing.Do()
43
44      def AddDecorator(self, d):
45          self.InsertDecorator(d)  # TODO make this append to the list of decorators.
46
47      def InsertDecorator(self, d):
48          """
49          Inserts the decorator first in the list of decorators.
50          """
51          d.thing = self.thing     # make decorater point to the thing you were pointing to
52          self.thing = d           # point to the decorator
53
54          if not USE_RM:
55              d.back = self
56              d.thing.back = d
57
58      def RemoveDecorator(self, d):
59          """
60          Find the thing_A pointing to the dying decorator and
61          point it at the thing the dying decorator was pointing to.
62          """
63          assert d.back
64          d.back.thing = d.thing
65
66          if not USE_RM:
67              d.thing.back = d.back   # update back pointer!!
68
69
70
71  import unittest
72
73
74  class TestCase01(unittest.TestCase):
75
76      def setUp(self):
77          self.proxything = Proxy()
78          self.d1 = Decorator1()
79
80      def checkNoDecoration(self):
81          self.proxything = Proxy()
82          res = self.proxything.Do()
83          #print res
84          assert res == 'hi'
85
86      def checkOneDecoration(self):
87          self.proxything.AddDecorator(self.d1)
88          res = self.proxything.Do()
89          #print res
```

can switch this on or off

3 extra lines of complexity. :(

2 different ways of implementing these two pointers.

this is what RM saves you from having to maintain. :)