

```

1  """
2  Relationship manager revisited.
3  Version 1.2
4  June 2003.
5  (c) Andy Bulka
6  http://www.atug.com/andypatterns
7
8
9
10
11
12
13
14
15
16
17
18
19
20 """
21
22 class RM1:
23     def __init__(self):
24         from relationshipmanager import RelationshipManager
25         self.rm = RelationshipManager()
26         self.enforcer = {}
27
28     def ER(self, relId, cardinality, directionality="directional"):
29         # enforceRelationship(id, cardinality, directionality)
30         self.enforcer[relId] = (cardinality, directionality)
31
32     def _RemoveExistingRelationships(self, fromObj, toObj, relId):
33         def ExtinguishOldFrom():
34             oldFrom = self.B(toObj, relId)
35             self.NR(oldFrom, toObj, relId)
36         def ExtinguishOldTo():
37             oldTo = self.P(fromObj, relId)
38             self.NR(fromObj, oldTo, relId)
39         if relId in self.enforcer.keys():
40             cardinality, directionality = self.enforcer[relId]
41             if cardinality == "onetoone":
42                 ExtinguishOldFrom()
43                 ExtinguishOldTo()
44             elif cardinality == "onetomany": # and directionality == "directional":
45                 ExtinguishOldFrom()
46
47     def R(self, fromObj, toObj, relId):
48         # addRelationship(f, t, id)
49         self._RemoveExistingRelationships(fromObj, toObj, relId)
50         self.rm.AddRelationship(fromObj, toObj, relId)
51
52         if relId in self.enforcer.keys():
53             cardinality, directionality = self.enforcer[relId]
54             if directionality == "bidirectional":
55                 self.rm.AddRelationship(toObj, fromObj, relId)
56
57     def P(self, fromObj, relId):
58         # findObjectPointedToByMe(fromMe, id, cast)
59         return self.rm.FindObject(fromObj, None, relId)
60
61     def B(self, toObj, relId):
62         # findObjectPointingToMe(toMe, id cast)
63         return self.rm.FindObject(None, toObj, relId)
64
65     def PS(self, fromObj, relId):
66         # findObjectsPointedToByMe(fromMe, id, cast)

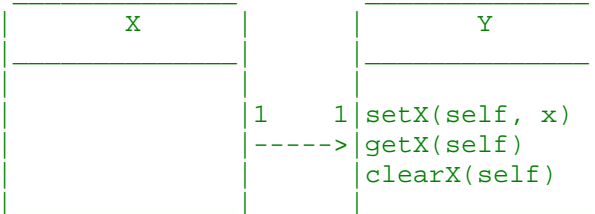
```

```

67         return self.rm.FindObjects(fromObj, None, relId)
68
69     def NR(self, fromObj, toObj, relId):
70         # removeRelationship(f, t, id)
71         self.rm.RemoveRelationships(fromObj, toObj, relId)
72
73         if relId in self.enforcer.keys():
74             cardinality, directionality = self.enforcer[relId]
75             if directionality == "bidirectional":
76                 self.rm.RemoveRelationships(toObj, fromObj, relId)
77
78
79 import unittest
80
81 RM = None
82
83 class TestCase01_OneToOne(unittest.TestCase):
84
85     def setUp(self):
86         global RM
87         RM = RM1()
88
89     def checkOneToOne_XSingularApi_YNoApi(self):
90         """
91
92         |-----|-----|
93         |         X         |         Y         |
94         |-----|-----|
95         | void setY(y) | 1 | 1 |
96         | Y   getY()   | ----> |
97         | void clearY() |
98         |-----|-----|
99
100        """
101        class X:
102            def __init__(self):          RM.ER("xtoy", "onetoone", "directional")
103            def setY(self, y):          RM.R(self, y, "xtoy")
104            def getY(self):            return RM.P(self, "xtoy")
105            def clearY(self):          RM.NR(self, self.getY(), "xtoy")
106
107        class Y:
108            pass
109
110        x1 = X()
111        x2 = X()
112        y1 = Y()
113        y2 = Y()
114        # Initial situation
115        assert x1.getY() == None
116        assert x2.getY() == None
117
118        # After clearing pointers
119        x1.clearY()
120        assert x1.getY() == None
121        assert x2.getY() == None
122
123        # After setting one pointer, x1 -> y1
124        x1.setY(y1)
125        assert x1.getY() == y1
126        assert x2.getY() == None
127
128        # After setting x2 -> y1, we cannot allow a situation where
129        # both x's to point to the same y, since this would be "many to one".
130        # The existing x1 -> y1 must be auto deleted by the relationship manager
131        # relationship enforcer.
132        assert x1.getY() == y1

```

```

133     x2.setY(y1)
134     assert x1.getY() == None # relationship should have been auto removed
135     assert x2.getY() == y1
136
137     # Clear one pointer
138     x1.clearY()
139     assert x1.getY() == None
140     assert x2.getY() == y1
141
142     # Clear other pointer
143     x2.clearY()
144     assert x1.getY() == None
145     assert x2.getY() == None
146
147     # Change from pointing to one thing then point to another
148     x1.setY(y1)
149     x1.setY(y2)
150     assert x1.getY() == y2
151
152     # Ensure repeat settings do not disturb things
153     x1.clearY()
154     x2.clearY()
155     # x1 -> y1, x2 -> None
156     x1.setY(y1)
157     assert x1.getY() == y1
158     assert x2.getY() == None
159     # repeat
160     x1.setY(y1)
161     assert x1.getY() == y1
162     assert x2.getY() == None
163
164     # x1 -> None, x2 -> y1
165     x2.setY(y1)
166     assert x1.getY() == None
167     assert x2.getY() == y1
168     # repeat
169     x2.setY(y1)
170     assert x1.getY() == None
171     assert x2.getY() == y1
172
173     # x1 -> y2, x2 -> y1
174     x1.setY(y2)
175     assert x1.getY() == y2
176     assert x2.getY() == y1
177     # repeat
178     x1.setY(y2)
179     assert x1.getY() == y2
180     assert x2.getY() == y1
181
182
183     def checkOneToOne_XNoApi_YSingularApi(self):
184         """
185
186         
187
188
189
190
191
192
193
194         """
195         class X:
196             pass
197
198         class Y:

```

```

199         def __init__(self):           RM.ER("xtoy", "onetoone", "directional")
200         def setX(self, x):             RM.R(x, self, "xtoy")
201         def getX(self):                return RM.B(self, "xtoy")
202         def clearX(self):              RM.NR(self.getX(), self, "xtoy")
203
204     x1 = X()
205     x2 = X()
206     y1 = Y()
207     y2 = Y()
208
209     # Initial situation
210     assert y1.getX() == None
211     assert y2.getX() == None
212
213     # After clearing pointers
214     y1.clearX()
215     assert y1.getX() == None
216     assert y2.getX() == None
217
218     # After setting one pointer, thus x1 -> y1
219     y1.setX(x1)
220     assert y1.getX() == x1
221     assert y2.getX() == None
222
223     # Want to show two x's pointing to same y
224     # Cannot do this since need access to an x api to do the 2nd link
225     # but this unit test assumes that the X has no API at all.
226     pass
227
228     # A y can be pointed to by many x's
229     # An x can only point at one y at a time
230     # So if x1 -> y1 and then x1 -> y2 then y1 is being pointed to by no-one.
231     # After setting other pointer, both x's pointing to same y, thus x1 & x2 -> y1
232     y2.setX(x1)
233     assert y1.getX() == None # should be auto cleared
234     assert y2.getX() == x1
235
236     # Clear one pointer
237     y1.clearX()
238     assert y1.getX() == None
239     assert y2.getX() == x1
240
241     # Clear other pointer
242     y2.clearX()
243     assert y1.getX() == None
244     assert y2.getX() == None
245
246     # Change from x1 -> y1 to x2 -> y1 (pointing to one thing then point to another)
247     y1.clearX()
248     y2.clearX()
249     y1.setX(x1)
250     y1.setX(x2)
251     assert y1.getX() == x2
252     assert y2.getX() == None
253
254     # Ensure repeat settings do not disturb things
255     y1.clearX()
256     y2.clearX()
257     y1.setX(x1)
258     assert y1.getX() == x1
259     assert y2.getX() == None
260     # repeat
261     y1.setX(x1)
262     assert y1.getX() == x1
263     assert y2.getX() == None
264

```

```

265
266     def onetoneasserts(self, x1, x2, y1, y2):
267
268         def assertallclear():
269             assert x1.getY() == None
270             assert x2.getY() == None
271             assert y1.getX() == None
272             assert y2.getX() == None
273
274         # Initial situation
275         assertallclear()
276
277         # After clearing pointers
278         x1.clearY()
279         x2.clearY()
280         y1.clearX()
281         y2.clearX()
282         assertallclear()
283
284         # After setting one pointer, x1 <-> y1
285         x1.setY(y1)
286         assert x1.getY() == y1
287         assert x2.getY() == None
288         assert y1.getX() == x1
289         assert y2.getX() == None
290
291         # After clearing that one pointer, x1 <-> y1
292         x1.clearY()
293         assertallclear()
294
295         # After setting one pointer, via y API, x1 <-> y1
296         y1.setX(x1)
297         assert x1.getY() == y1
298         assert x2.getY() == None
299         assert y1.getX() == x1
300         assert y2.getX() == None
301         y1.clearX()
302         assertallclear()
303
304         # After setting one pointer, via y API, x1 <-> y1
305         # then change it via x API, to           x1 <-> y2
306         # thus the old x1 <-> y1 must extinguish.
307         y1.setX(x1)
308         x1.setY(y2)
309         assert x1.getY() == y2
310         assert x2.getY() == None
311         assert y1.getX() == None
312         assert y2.getX() == x1
313         # repeat
314         y1.setX(x1)
315         x1.setY(y2)
316         assert x1.getY() == y2
317         assert x2.getY() == None
318         assert y1.getX() == None
319         assert y2.getX() == x1
320         # clear
321         x1.clearY()
322         assertallclear()
323
324         # Do same trick using opposite API's
325         x1.setY(y1) # instead of y1.setX(x1)
326         y2.setX(x1) # instead of x1.setY(y2)
327         # exactly the same assertions
328         assert x1.getY() == y2
329         assert x2.getY() == None
330         assert y1.getX() == None

```

```

331     assert y2.getX() == x1
332     # repeat
333     x1.setY(y1)
334     y2.setX(x1)
335     assert x1.getY() == y2
336     assert x2.getY() == None
337     assert y1.getX() == None
338     assert y2.getX() == x1
339
340     y2.clearX()
341     assertAllClear()
342
343
344     # Wire both x1-y1, x2-y2 using x API
345     x1.setY(y1)
346     x2.setY(y2)
347     assert x1.getY() == y1
348     assert x2.getY() == y2
349     assert y1.getX() == x1
350     assert y2.getX() == x2
351     # repeat wiring using opposite Y api, same asserts
352     y1.setX(x1)
353     y2.setX(x2)
354     assert x1.getY() == y1
355     assert x2.getY() == y2
356     assert y1.getX() == x1
357     assert y2.getX() == x2
358     # Now set x2-y1 using x API, should yield x1-None, x2-y1
359     x2.setY(y1)
360     assert x1.getY() == None
361     assert x2.getY() == y1
362     assert y1.getX() == x2
363     assert y2.getX() == None
364     # Repeat above set x2-y1 using y API, same asserts
365     y1.setX(x2)
366     assert x1.getY() == None
367     assert x2.getY() == y1
368     assert y1.getX() == x2
369     assert y2.getX() == None
370     # Now set x1-y2, using y API, should yield x1-y2, x2-y1
371     y2.setX(x1)
372     assert x1.getY() == y2
373     assert x2.getY() == y1
374     assert y1.getX() == x2
375     assert y2.getX() == x1
376     # Repeat above set x1-y2, using x API, same asserts
377     x1.setY(y2)
378     assert x1.getY() == y2
379     assert x2.getY() == y1
380     assert y1.getX() == x2
381     assert y2.getX() == x1
382
383     x1.clearY()
384     assert x1.getY() == None
385     assert x2.getY() == y1
386     assert y1.getX() == x2
387     assert y2.getX() == None
388     y1.clearX()
389     assertAllClear()
390
391     def checkOneToOne_XSingularApi_YSingularApi(self):
392         """
393         Since both sides have an API, then this is bidirectional
394
395         |-----X-----|         |-----Y-----|
396

```

```

397
398 |void  setY(y) |1    1|setX(self, x) |
399 |Y      getY() |<---->|getX(self)   |
400 |void  clearY()|      |clearX(self) |
401 |_____|      |_____|
402 """
403 class X:
404     def __init__(self):          RM.ER("xy", "onetoone", "bidirectional")
405     def setY(self, y):          RM.R(self, y, "xy")
406     def getY(self):            return RM.P(self, "xy")
407     def clearY(self):          RM.NR(self, self.getY(), "xy")
408
409 class Y:
410     def __init__(self):          RM.ER("xy", "onetoone", "bidirectional")
411     def setX(self, x):          RM.R(self, x, "xy")
412     def getX(self):            return RM.P(self, "xy")
413     def clearX(self):          RM.NR(self, self.getX(), "xy")
414
415 x1 = X()
416 x2 = X()
417 y1 = Y()
418 y2 = Y()
419 self.onetooneasserts(x1,x2,y1,y2)
420
421 def checkOneToOne_XSingularApi_YSingularApi_Alt(self):
422     """
423     Alternative implementation of same API.
424
425     |_____|      |_____|
426     |      X      |      |      Y      |
427     |_____|      |_____|
428
429 |void  setY(y) |1    1|setX(self, x) |
430 |Y      getY() |<---->|getX(self)   |
431 |void  clearY()|      |clearX(self) |
432 |_____|      |_____|
433
434 Proves that the bidirectional one to one - API using only P()
435 (whether implemented as two synchronised relationships,
436 or whether implemented as a smart single bi relationship)
437 can also be implemented as directional one to one - API using B() and P()
438 (whether implemented as a single relationship,
439 or whether implemented as two synchronised (more efficient) relationships)
440
441 Note that the above alternative API implementation is a pure
442 combination of the two directional APIs from unit tests
443     XSingularApi_YNoApi
444     XNoApi_YSingularApi
445 using the directional relationship "xtoy".
446 """
447 class X:
448     def __init__(self):          RM.ER("xtoy", "onetoone", "directional")
449     def setY(self, y):          RM.R(self, y, "xtoy")
450     def getY(self):            return RM.P(self, "xtoy")
451     def clearY(self):          RM.NR(self, self.getY(), "xtoy")
452 class Y:
453     def __init__(self):          RM.ER("xtoy", "onetoone", "directional")
454     def setX(self, x):          RM.R(x, self, "xtoy")
455     def getX(self):            return RM.B(self, "xtoy")
456     def clearX(self):          RM.NR(self.getX(), self, "xtoy")
457
458 x1 = X()
459 x2 = X()
460 y1 = Y()
461 y2 = Y()
462 self.onetooneasserts(x1,x2,y1,y2)

```

```

463
464
465
466 class TestCase02_OneToMany(unittest.TestCase):
467
468     def setUp(self):
469         global RM
470         RM = RM1()
471
472     def checkOneToMany_XPluralApi_YNoApi(self):
473         """
474         One to Many
475
476         |-----|           |-----|
477         |           X           |           |           Y           |
478         |-----|           |-----|
479
480         |addY(self, y)         | 1       * |
481         |getAllY(self)        |-----> |
482         |removeY(self, y)     |           |
483         |-----|           |-----|
484
485         X has the required plural API,
486         Y has no API.
487         """
488     class X:
489         def __init__(self):          RM.ER("xtoy", "onetomany", "directional")
490         def addY(self, y):           RM.R(self, y, "xtoy")
491         def getAllY(self):          return RM.PS(self, "xtoy")
492         def removeY(self, y):       RM.NR(self, y, "xtoy")
493
494     class Y:
495         pass
496
497     x1 = X()
498     x2 = X()
499     y1 = Y()
500     y2 = Y()
501     self.onetomanyasserts(x1,x2,y1,y2)
502
503     def checkOneToMany_XPluralApi_YSingularApi(self):
504         """
505         One to Many, BI
506
507         |-----|           |-----|
508         |           X           |           |           Y           |
509         |-----|           |-----|
510
511         |addY(self, y)         | 1       * |setX(self, x)
512         |getAllY(self)        |-----> |getX(self)
513         |removeY(self, y)     |           |clearX(self)
514         |-----|           |-----|
515
516         X has the required plural API,
517         Y has the reciprocal singular API
518
519         Since there are two API's, one on each class, this makes it a bidirectional :
520
521         However !! there still remains a strong sense of directionality since the on
522         is directional i.e. the one is the X and the many is the Y.
523
524         Thus RM.R on both API's must be always done from the X to the Y.
525
526         So in a sense, the relationships should be named "xtoy" even though it is bi
527
528

```



```

595 |_____| |_____|
596
597 """
598 class X:
599     def __init__(self):          RM.ER("xtoy", "onetomany", "directional")
600     def addY(self, y):          RM.R(self, y, "xtoy")
601     def getAllY(self):         return RM.PS(self, "xtoy")
602     def removeY(self, y):      RM.NR(self, y, "xtoy")
603
604 class Y:
605     def setX(self, x):          RM.R(x, self, "xtoy")
606     def getX(self):            return RM.B(self, "xtoy")
607     def clearX(self):          RM.NR(self.getX(), self, "xtoy")
608
609 x1 = X()
610 x2 = X()
611 y1 = Y()
612 y2 = Y()
613 self.onetomanyasserts(x1,x2,y1,y2,yapi=1)
614
615
616 def onetomanyasserts(self, x1, x2, y1, y2, yapi=0):
617
618     def assertallclear():
619         assert x1.getAllY() == []
620         assert x2.getAllY() == []
621         if yapi:
622             assert y1.getX() == None
623             assert y2.getX() == None
624     def assertSituation00():
625         assert x1.getAllY() == [y1]
626         if yapi:
627             assert y1.getX() == x1
628
629     # Initial situation
630     assertallclear()
631
632     # clearing pointers that do not exist, should be ok.
633     x1.removeY(y1)
634     assertallclear()
635     x1.removeY(y2)
636     assertallclear()
637     x2.removeY(y1)
638     assertallclear()
639     x2.removeY(y2)
640     assertallclear()
641     if yapi:
642         y1.clearX()
643         assertallclear()
644         y2.clearX()
645         assertallclear()
646
647
648     """
649     +-----+
650     |Add a single X to Y relationship|
651     +-----+
652     """
653     x1.addY(y1)
654     """
655     ,-----, ,-----,
656     (  x1  )---->(  y1  )
657     `-----' `-----'
658     """
659     assertSituation00()
660     # now remove it

```

```

661     x1.removeY(y1)
662     assertallclear()
663
664     # Add initial relationship, from the y side
665     if yapi:
666         y1.setX(x1)
667         assertSituation00()
668         # now remove it, from the y side
669         y1.clearX()
670         assertallclear()
671
672
673     """
674     +-----+
675     |Add two relationships coming from a single X|
676     |to multiple Y's.                            |
677     +-----+
678     """
679     def assertSituation01():
680         """
681         ( x1 )---->( y1 )
682         |
683         |
684         |
685         |
686         |----->( y2 )
687         |
688         |
689         """
690         assert x1.getAllY() == [y1, y2], "Actual situation %s" % x1.getAllY()
691         if yapi:
692             assert y1.getX() == x1
693             assert y2.getX() == x1
694     def assertSituation02():
695         """
696         ( x1 )    ( y1 )
697         |
698         |
699         |
700         |----->( y2 )
701         |
702         |
703         """
704         assert x1.getAllY() == [y2]
705         if yapi:
706             assert y1.getX() == None
707             assert y2.getX() == x1
708     # Add two relationships, from x API
709     x1.addY(y1)
710     x1.addY(y2)
711     assertSituation01()
712     # now remove y1
713     x1.removeY(y1)
714     assertSituation02()
715     # now remove y2
716     x1.removeY(y2)
717     assertallclear()
718
719     # Add two relationships, from the y api side.
720     if yapi:
721         assertallclear()
722         y1.setX(x1)
723         y2.setX(x1)
724         assertSituation01()
725         # now remove y1
726         y1.clearX()
727         assertSituation02()

```

```

727         # now remove y1
728         y2.clearX()
729         assertallclear()
730
731
732
733         """
734         +-----+
735         |Add same relationship twice|
736         +-----+
737         """
738         def assertSituation03():
739             assert x1.getAllY() == [y1]
740             if yapi:
741                 assert y1.getX() == x1
742         def assertSituation04():
743             assert x1.getAllY() == [y1, y2]
744             if yapi:
745                 assert y2.getX() == x1
746         def assertSituation05():
747             if yapi:
748                 assert y1.getX() == None
749                 assert x1.getAllY() == [y2]
750         x1.addY(y1)
751         x1.addY(y1)
752         """
753
754         ( x1 )---->( y1 )
755         \-----/
756
757
758         ( y2 )
759         \-----/
760
761         """
762         assertSituation03()
763
764         x1.addY(y2)
765         x1.addY(y2)
766         """
767         ( x1 )---->( y1 )
768         \-----/
769         \----->
770         \----->
771         ( y2 )
772         \-----/
773
774         """
775         assertSituation04()
776         # now remove y1 (again, twice, just to check robustness)
777         x1.removeY(y1)
778         x1.removeY(y1)
779         """
780         ( x1 )---->( y1 )
781         \-----/
782
783
784         ( y2 )
785         \-----/
786
787         """
788         assertSituation05()
789         # now remove y2 twice
790         x1.removeY(y2)
791         x1.removeY(y2)
792
793         assertallclear()

```

```

793
794
795     """
796     +-----+
797     |Add same relationship twice, from Y side|
798     +-----+
799     """
800     if yapi:
801         y1.setX(x1)
802         y1.setX(x1)
803         """
804         ( x1 )---->( y1 )
805         (-----)
806
807
808
809         ( y2 )
810         (-----)
811
812         """
812         assertSituation03()
813         y2.setX(x1)
814         y2.setX(x1)
815         """
816         ( x1 )---->( y1 )
817         (-----)
818         \
819         \
820         \
821         \>( y2 )
822         (-----)
823
824         """
824         assertSituation04()
825         # now remove y1, from Y side (again, twice, just to check robustness)
826         y1.clearX()
827         y1.clearX()
828         """
829         ( x1 )---->( y1 )
830         (-----)
831
832
833
834         ( y2 )
835         (-----)
836
837         """
837         assertSituation05()
838         # now remove y2 twice, from Y side
839         y2.clearX()
840         y2.clearX()
841
842         assertallclear()
843
844
845     """
846     +-----+
847     |Add two relationships, then add a third |
848     |relationship which effects an previous relationship.|
849     +-----+
850     """
851     # Make x1 -> y1,y2
852     assertallclear()
853     x1.addY(y1)
854     x1.addY(y2)
855     """
856     ( x1 )---->( y1 )
857     (-----)
858

```

```

859
860
861     ( x2 ) > ( y2 )
862     \-----/
863
864     """
865     # Now make x2 -> y1
866     x2.addY(y1)
867     """
868
869     ( x1 ) > ( y1 )
870     \-----/
871     /-----\
872     ( x2 ) > ( y2 )
873     \-----/
874
875     After much thought, I believe the addition of the x2 -> y1 relationship
876     should extinguish the existing x1 -> y1 since y's can only be pointed to by
877     If you want to keep the existing x1 -> y1 then you actually are describing t
878     many to many, directional, no y api, scenario.
879     """
880
881     assert x1.getAllY() == [y2]
882     assert x2.getAllY() == [y1]
883     if yapi:
884         assert y1.getX() == x2
885         assert y2.getX() == x1
886
887     x1.removeY(y2)
888     x2.removeY(y1)
889     assertallclear()
890
891     """
892     +-----+
893     |Two different X's point to the same Y.
894     |Again enforcement that y only pointed to by one X,
895     |and that the original relationship is extinguished.
896     +-----+
897     """
898
899     def assertSituation06():
900         assert x1.getAllY() == []
901         assert x2.getAllY() == [y1]
902         if yapi:
903             assert y1.getX() == x2
904
905     x1.addY(y1)
906     """
907
908     ( x1 ) ----> ( y1 )
909     \-----/
910
911     ( x2 )
912     \-----/
913
914     """
915     x2.addY(y1)
916     """
917
918     ( x1 ) .> ( y1 )
919     \-----/
920     |
921     ( x2 ) ---+
922     \-----/
923
924     """
925     assertSituation06()
926
927     x2.removeY(y1)
928     assertallclear()

```

```
925
926
927     # Same as above, except wired via Y's API
928     if yapi:
929         y1.setX(x1)
930         y1.setX(x2)
931         assertSituation06()
932         y1.clearX()
933         assertallclear()
934
935     assertallclear()
936
937
938
939
940 def suite():
941     suite1 = unittest.makeSuite(TestCase01_OneToOne, 'check')
942     suite2 = unittest.makeSuite(TestCase02_OneToMany, 'check')
943     alltests = unittest.TestSuite((suite1, suite2))
944     return alltests
945
946 def main():
947     """ Run all the suites.  To run via a gui, then
948         python unittestgui.py NestedDictionaryTest.suite
949         Note that I run with VERBOSITY on HIGH :- ) just like in the old days
950         with pyUnit for python 2.0
951         Simply call
952         runner = unittest.TextTestRunner(descriptions=0, verbosity=2)
953         The default arguments are descriptions=1, verbosity=1
954     """
955     runner = unittest.TextTestRunner(descriptions = 0, verbosity = 2) # default is d
956     #runner = unittest.TextTestRunner(descriptions=0, verbosity=1) # default is desc
957     runner.run(suite())
958
959 if __name__ == '__main__':
960     main()
961
962
```

```
1 class RelationshipManager:
2     def __init__(self): # Constructor
3         self.Relationships = []
4     def AddRelationship(self, From, To, RelId=1):
5         if not self.FindObjects(From, To, RelId):
6             self.Relationships.append( (From, To, RelId) ) # assoc obj
7     def RemoveRelationships(self, From, To, RelId=1):
8         if not From or not To:
9             return
10        lzt = self.FindObjects(From, To, RelId)
11        if lzt:
12            for association in lzt:
13                self.Relationships.remove(association)
14    def FindObjects(self, From=None, To=None, RelId=1):
15        resultlist = []
16        match = lambda obj,list,index : obj==list[index] or obj==None
17        for association in self.Relationships:
18            if match(From,association,0) and match(To,association,1) and RelId==associ.
19                if From==None:
20                    resultlist.append(association[0])
21                elif To==None:
22                    resultlist.append(association[1])
23                else:
24                    resultlist.append(association)
25        return resultlist
26    def FindObject(self, From=None, To=None, RelId=1):
27        lzt = self.FindObjects(From, To, RelId)
28        if lzt:
29            return lzt[0]
30        else:
31            return None
32    def Clear(self):
33        del self.Relationships[0:]
34
35    import unittest, random
36
37    class TestCase00(unittest.TestCase):
38        def setUp(self):
39            self.rm = RelationshipManager()
40        def checkBasic00(self):
41            self.rm.AddRelationship('a','b')
42            self.rm.AddRelationship('a','c')
43            assert self.rm.FindObjects('a',None) == ['b','c']
44            assert self.rm.FindObjects(None,'a') == []
45            assert self.rm.FindObjects(None,'b') == ['a']
46            assert self.rm.FindObjects(None,'c') == ['a']
47        def checkBasic01Singular(self):
48            self.rm.AddRelationship('a','b')
49            self.rm.AddRelationship('a','c')
50            assert self.rm.FindObject(None,'b') == 'a'
51            assert self.rm.FindObject(None,'c') == 'a'
52            assert self.rm.FindObject('a',None) == 'b' # could have been 'c' - arbitrary
53
54    def suite():
55        suite1 = unittest.makeSuite(TestCase00,'check')
56        alltests = unittest.TestSuite( (suite1,) )
57        return alltests
58
59    def main():
60        """ Run all the suites. To run via a gui, then
61            python unittestgui.py NestedDictionaryTest.suite
62            Note that I run with VERBOSITY on HIGH :- ) just like in the old days
63            with pyUnit for python 2.0
64            Simply call
65            runner = unittest.TextTestRunner(descriptions=0, verbosity=2)
66            The default arguments are descriptions=1, verbosity=1
```



```
67         """
68         runner = unittest.TextTestRunner(descriptions=0, verbosity=2) # default is descr
69         #runner = unittest.TextTestRunner(descriptions=0, verbosity=1) # default is desc
70         runner.run( suite() )
71
72     if __name__ == '__main__':
73         main()
```