```python
"""
Relationship manager revisited.
Version 1.2
June 2003.
(c) Andy Bulka
http://www.atug.com/andypatterns

  ____      _       _   _                 _     _
 |  _ \ ___| | __ _| |_(_) ___  _ __  ___| |__ (_)_ __
 | |_) / _ \ |/ _` | __| |/ _ \| '_ \/ __| '_ \| | '_ \
 |  _ <  __/ | (_| | |_| | (_) | | | \__ \ | | | | |_) |
 |_| \_\___|_|\__,_|\__|_|\___/|_| |_|___/_| |_|_| .__/
                                                  |_|
  __  __
 |  \/  | __ _ _ __   __ _  __ _  ___ _ __
 | |\/| |/ _` | '_ \ / _` |/ _` |/ _ \ '__|
 | |  | | (_| | | | | (_| | (_| |  __/ |
 |_|  |_|\__,_|_| |_|\__,_|\__, |\___|_|
                              |___/
"""

class RM1:
    def __init__(self):
        from relationshipmanager import RelationshipManager
        self.rm = RelationshipManager()
        self.enforcer = {}

    def ER(self, relId, cardinality, directionality="directional"):
        # enforceRelationship(id, cardinality, directionality)
        self.enforcer[relId] = (cardinality, directionality)

    def _RemoveExistingRelationships(self, fromObj, toObj, relId):
        def ExtinguishOldFrom():
            oldFrom = self.B(toObj, relId)
            self.NR(oldFrom, toObj, relId)
        def ExtinguishOldTo():
            oldTo = self.P(fromObj, relId)
            self.NR(fromObj, oldTo, relId)
        if relId in self.enforcer.keys():
            cardinality, directionality = self.enforcer[relId]
            if cardinality == "onetoone":
                ExtinguishOldFrom()
                ExtinguishOldTo()
            elif cardinality == "onetomany": # and directionality == "directional":
                ExtinguishOldFrom()

    def R(self, fromObj, toObj, relId):
        # addRelationship(f, t, id)
        self._RemoveExistingRelationships(fromObj, toObj, relId)
        self.rm.AddRelationship(fromObj, toObj, relId)

        if relId in self.enforcer.keys():
            cardinality, directionality = self.enforcer[relId]
            if directionality == "bidirectional":
                self.rm.AddRelationship(toObj, fromObj, relId)

    def P(self, fromObj, relId):
        # findObjectPointedToByMe(fromMe, id, cast)
        return self.rm.FindObject(fromObj, None, relId)

    def B(self, toObj, relId):
        # findObjectPointingToMe(toMe, id cast)
        return self.rm.FindObject(None, toObj, relId)

    def PS(self, fromObj, relId):
        # findObjectsPointedToByMe(fromMe, id, cast)
```

```
            return self.rm.FindObjects(fromObj, None, relId)

        def NR(self, fromObj, toObj, relId):
            # removeRelationship(f, t, id)
            self.rm.RemoveRelationships(fromObj, toObj, relId)

            if relId in self.enforcer.keys():
                cardinality, directionality = self.enforcer[relId]
                if directionality == "bidirectional":
                    self.rm.RemoveRelationships(toObj, fromObj, relId)


    import unittest

    RM = None

    class TestCase01_OneToOne(unittest.TestCase):

        def setUp(self):
            global RM
            RM = RM1()

        def checkOneToOne_XSingularApi_YNoApi(self):
            """
             _______________        _______________
            |       X       |      |       Y       |
            |_______________|      |_______________|
            |               |      |               |
            |void  setY(y) |1    1|               |
            |Y     getY()  |----->|               |
            |void  clearY()|      |               |
            |_______________|      |_______________|

            """
            class X:
                def __init__(self):        RM.ER("xtoy", "onetoone", "directional")
                def setY(self, y):         RM.R(self, y, "xtoy")
                def getY(self):     return RM.P(self, "xtoy")
                def clearY(self):          RM.NR(self, self.getY(), "xtoy")

            class Y:
                pass

            x1 = X()
            x2 = X()
            y1 = Y()
            y2 = Y()
            # Initial situation
            assert x1.getY() == None
            assert x2.getY() == None

            # After clearing pointers
            x1.clearY()
            assert x1.getY() == None
            assert x2.getY() == None

            # After setting one pointer, x1 -> y1
            x1.setY(y1)
            assert x1.getY() == y1
            assert x2.getY() == None

            # After setting x2 -> y1, we cannot allow a situation where
            # both x's to point to the same y, since this would be "many to one".
            # The existing x1 -> y1 must be auto deleted by the relationship manager
            # relationship enforcer.
            assert x1.getY() == y1
```

```
        x2.setY(y1)
        assert x1.getY() == None  # relationship should have been auto removed
        assert x2.getY() == y1

        # Clear one pointer
        x1.clearY()
        assert x1.getY() == None
        assert x2.getY() == y1

        # Clear other pointer
        x2.clearY()
        assert x1.getY() == None
        assert x2.getY() == None

        # Change from pointing to one thing then point to another
        x1.setY(y1)
        x1.setY(y2)
        assert x1.getY() == y2

        # Ensure repeat settings do not disturb things
        x1.clearY()
        x2.clearY()
        # x1 -> y1, x2 -> None
        x1.setY(y1)
        assert x1.getY() == y1
        assert x2.getY() == None
        # repeat
        x1.setY(y1)
        assert x1.getY() == y1
        assert x2.getY() == None

        # x1 -> None, x2 -> y1
        x2.setY(y1)
        assert x1.getY() == None
        assert x2.getY() == y1
        # repeat
        x2.setY(y1)
        assert x1.getY() == None
        assert x2.getY() == y1

        # x1 -> y2, x2 -> y1
        x1.setY(y2)
        assert x1.getY() == y2
        assert x2.getY() == y1
        # repeat
        x1.setY(y2)
        assert x1.getY() == y2
        assert x2.getY() == y1


    def checkOneToOne_XNoApi_YSingularApi(self):
        """
         _______________        _______________
        |       X       |      |       Y       |
        |_______________|      |_______________|
        |               |      |               |
        |               |1    1|setX(self, x)  |
        |               |----->|getX(self)     |
        |               |      |clearX(self)   |
        |_______________|      |_______________|

        """
        class X:
            pass

        class Y:
```

```
            def __init__(self):        RM.ER("xtoy", "onetoone", "directional")
            def setX(self, x):         RM.R(x, self, "xtoy")
            def getX(self):     return RM.B(self, "xtoy")
            def clearX(self):          RM.NR(self.getX(), self, "xtoy")

        x1 = X()
        x2 = X()
        y1 = Y()
        y2 = Y()

        # Initial situation
        assert y1.getX() == None
        assert y2.getX() == None

        # After clearing pointers
        y1.clearX()
        assert y1.getX() == None
        assert y2.getX() == None

        # After setting one pointer, thus x1 -> y1
        y1.setX(x1)
        assert y1.getX() == x1
        assert y2.getX() == None

        # Want to show two x's pointing to same y
        # Cannot do this since need access to an x api to do the 2nd link
        # but this unit test assumes that the X has no API at all.
        pass

        # A y can be pointed to by many x's
        # An x can only point at one y at a time
        # So if x1 -> y1 and then x1 -> y2 then y1 is being pointed to by no-one.
        # After setting other pointer, both x's pointing to same y, thus x1 & x2 ->
        y2.setX(x1)
        assert y1.getX() == None  # should be auto cleared
        assert y2.getX() == x1

        # Clear one pointer
        y1.clearX()
        assert y1.getX() == None
        assert y2.getX() == x1

        # Clear other pointer
        y2.clearX()
        assert y1.getX() == None
        assert y2.getX() == None

        # Change from x1 -> y1 to x2 -> y1 (pointing to one thing then point to anot
        y1.clearX()
        y2.clearX()
        y1.setX(x1)
        y1.setX(x2)
        assert y1.getX() == x2
        assert y2.getX() == None

        # Ensure repeat settings do not disturb things
        y1.clearX()
        y2.clearX()
        y1.setX(x1)
        assert y1.getX() == x1
        assert y2.getX() == None
        # repeat
        y1.setX(x1)
        assert y1.getX() == x1
        assert y2.getX() == None
```

```python

        def onetooneasserts(self, x1, x2, y1, y2):

            def assertallclear():
                assert x1.getY() == None
                assert x2.getY() == None
                assert y1.getX() == None
                assert y2.getX() == None

            # Initial situation
            assertallclear()

            # After clearing pointers
            x1.clearY()
            x2.clearY()
            y1.clearX()
            y2.clearX()
            assertallclear()

            # After setting one pointer, x1 <-> y1
            x1.setY(y1)
            assert x1.getY() == y1
            assert x2.getY() == None
            assert y1.getX() == x1
            assert y2.getX() == None

            # After clearing that one pointer, x1 <-> y1
            x1.clearY()
            assertallclear()

            # After setting one pointer, via y API, x1 <-> y1
            y1.setX(x1)
            assert x1.getY() == y1
            assert x2.getY() == None
            assert y1.getX() == x1
            assert y2.getX() == None
            y1.clearX()
            assertallclear()

            # After setting one pointer, via y API, x1 <-> y1
            # then change it via x API, to          x1 <-> y2
            # thus the old x1 <-> y1 must extinguish.
            y1.setX(x1)
            x1.setY(y2)
            assert x1.getY() == y2
            assert x2.getY() == None
            assert y1.getX() == None
            assert y2.getX() == x1
            # repeat
            y1.setX(x1)
            x1.setY(y2)
            assert x1.getY() == y2
            assert x2.getY() == None
            assert y1.getX() == None
            assert y2.getX() == x1
            # clear
            x1.clearY()
            assertallclear()

            # Do same trick using opposite API's
            x1.setY(y1)                             # instead of y1.setX(x1)
            y2.setX(x1)                             # instead of x1.setY(y2)
            # exactly the same assertions
            assert x1.getY() == y2
            assert x2.getY() == None
            assert y1.getX() == None
```

```
        assert y2.getX() == x1
        # repeat
        x1.setY(y1)
        y2.setX(x1)
        assert x1.getY() == y2
        assert x2.getY() == None
        assert y1.getX() == None
        assert y2.getX() == x1

        y2.clearX()
        assertallclear()


        # Wire both x1-y1, x2-y2 using x API
        x1.setY(y1)
        x2.setY(y2)
        assert x1.getY() == y1
        assert x2.getY() == y2
        assert y1.getX() == x1
        assert y2.getX() == x2
        # repeat wiring using opposite Y api, same asserts
        y1.setX(x1)
        y2.setX(x2)
        assert x1.getY() == y1
        assert x2.getY() == y2
        assert y1.getX() == x1
        assert y2.getX() == x2
        # Now set x2-y1 using x API, should yield x1-None, x2-y1
        x2.setY(y1)
        assert x1.getY() == None
        assert x2.getY() == y1
        assert y1.getX() == x2
        assert y2.getX() == None
        # Repeat above set x2-y1 using y API, same asserts
        y1.setX(x2)
        assert x1.getY() == None
        assert x2.getY() == y1
        assert y1.getX() == x2
        assert y2.getX() == None
        # Now set x1-y2, using y API, should yield x1-y2, x2-y1
        y2.setX(x1)
        assert x1.getY() == y2
        assert x2.getY() == y1
        assert y1.getX() == x2
        assert y2.getX() == x1
        # Repeat above set x1-y2, using x API, same asserts
        x1.setY(y2)
        assert x1.getY() == y2
        assert x2.getY() == y1
        assert y1.getX() == x2
        assert y2.getX() == x1

        x1.clearY()
        assert x1.getY() == None
        assert x2.getY() == y1
        assert y1.getX() == x2
        assert y2.getX() == None
        y1.clearX()
        assertallclear()

    def checkOneToOne_XSingularApi_YSingularApi(self):
        """
        Since both sides have an API, then this is bidirectional
         _______________        _______________
        |       X       |      |       Y       |
        |_______________|      |_______________|
```

```
            |              |      |              |
            |void  setY(y) |1    1|setX(self, x) |
            |Y     getY()  |<---->|getX(self)    |
            |void  clearY()|      |clearX(self)  |
            |______________|      |______________|
            """
            class X:
                def __init__(self):        RM.ER("xy", "onetoone", "bidirectional")
                def setY(self, y):         RM.R(self, y, "xy")
                def getY(self):     return RM.P(self, "xy")
                def clearY(self):          RM.NR(self, self.getY(), "xy")

            class Y:
                def __init__(self):        RM.ER("xy", "onetoone", "bidirectional")
                def setX(self, x):         RM.R(self, x, "xy")
                def getX(self):     return RM.P(self, "xy")
                def clearX(self):          RM.NR(self, self.getX(), "xy")

            x1 = X()
            x2 = X()
            y1 = Y()
            y2 = Y()
            self.onetooneasserts(x1,x2,y1,y2)

        def checkOneToOne_XSingularApi_YSingularApi_Alt(self):
            """
            Alternative implementation of same API.

             _____________        _____________
            |       X      |      |       Y      |
            |______________|      |______________|
            |              |      |              |
            |void  setY(y) |1    1|setX(self, x) |
            |Y     getY()  |<---->|getX(self)    |
            |void  clearY()|      |clearX(self)  |
            |______________|      |______________|

            Proves that the bidirectional one to one - API using only P()
             (whether implemented as two synchronised relationships,
              or whether implemented as a smart single bi relationship)
            can also be imlpemented as directional one to one - API using B() and P()
             (whether implemented as a single relationship,
              or whether implemented as two synchronised (more efficient) relationships)

            Note that the above alternative API implementation is a pure
            combination of the two directional APIs from unit tests
                XSingularApi_YNoApi
                XNoApi_YSingularApi
            using the directional relationship "xtoy".
            """
            class X:
                def __init__(self):        RM.ER("xtoy", "onetoone", "directional")
                def setY(self, y):         RM.R(self, y, "xtoy")
                def getY(self):     return RM.P(self, "xtoy")
                def clearY(self):          RM.NR(self, self.getY(), "xtoy")
            class Y:
                def __init__(self):        RM.ER("xtoy", "onetoone", "directional")
                def setX(self, x):         RM.R(x, self, "xtoy")
                def getX(self):     return RM.B(self, "xtoy")
                def clearX(self):          RM.NR(self.getX(), self, "xtoy")

            x1 = X()
            x2 = X()
            y1 = Y()
            y2 = Y()
            self.onetooneasserts(x1,x2,y1,y2)
```

```python



class TestCase02_OneToMany(unittest.TestCase):

    def setUp(self):
        global RM
        RM = RM1()

    def checkOneToMany_XPluralApi_YNoApi(self):
        """
        One to Many

         _________________        _______________
        |        X        |      |       Y       |
        |_________________|      |_______________|
        |                 |      |               |
        |addY(self, y)    |1    *|               |
        |getAllY(self)    |----->|               |
        |removeY(self, y) |      |               |
        |_________________|      |_______________|

        X has the required plural API,
        Y has no API.
        """
        class X:
            def __init__(self):        RM.ER("xtoy", "onetomany", "directional")
            def addY(self, y):         RM.R(self, y, "xtoy")
            def getAllY(self):  return RM.PS(self, "xtoy")
            def removeY(self, y):      RM.NR(self, y, "xtoy")

        class Y:
            pass

        x1 = X()
        x2 = X()
        y1 = Y()
        y2 = Y()
        self.onetomanyasserts(x1,x2,y1,y2)

    def checkOneToMany_XPluralApi_YSingularApi(self):
        """
        One to Many, BI

         _________________        _______________
        |        X        |      |       Y       |
        |_________________|      |_______________|
        |                 |      |               |
        |addY(self, y)    |1    *|setX(self, x)  |
        |getAllY(self)    |<---->|getX(self)     |
        |removeY(self, y) |      |clearX(self)   |
        |_________________|      |_______________|

        X has the required plural API,
        Y has the reciprocal singular API

        Since there are two API's, one on each class, this makes it a bidirectional

        However !! there still remains a strong sense of directionality since the on
        is directional i.e. the one is the X and the many is the Y.

        Thus RM.R on both API's must be always done from the X to the Y.

        So in a sense, the relationships should be named "xtoy" even though it is bi


```

```
                                  ___                 ___
            _________          ___d888b___         ___d888
         d888888888b______d88888888888b_____d8888888
        d88888888888888888888888888888888888888888888
        Y88P     Y8888888888P       Y888888888P

        ASIDE:
        Only in the many to many case would you consider using a name like "xy" for
        relationshipId.  ??

        But even then, you often have a many to many that is directional
        e.g. many brothers to many sisters - you must get the directionality right
        e.g. "brothertosister" is one relationship
             X (brother)                                Y (sister)
             addSister(s)  RM.R(this,s,'btos')          addBrother(b)  RM.R(b,this,'
             getSisters()                               getBrothers()

        Notice the directionality is always the one way.

        Will there ever be a many to many where directionality DOESN'T matter?
        Perhaps if there are two different objects pointing to each other, there
        are going to be the 'attachment points' - thus you ALWAYS need to know
        who is on what side of the relationship.

        END ASIDE.

        h88b     h8888888888b       h888888888b
        q88888888888888888888888888888888888888888888
         q888888888p""   "q88888888888p""q8888888
                            ""q888p""       ""q888

        This too, has two implementations, do it as a BI, with proper pointers
        from y to x (rather than relying on backpointers).
        Note that implementationally, both cases can be done with a single relations
        or both cases can be done with a pair of relationships.
        one relationship, a bi
        """
        class X:
            def __init__(self):        RM.ER("xtoy", "onetomany", "bidirectional")
            def addY(self, y):         RM.R(self, y, "xtoy")
            def getAllY(self):  return RM.PS(self, "xtoy")
            def removeY(self, y):      RM.NR(self, y, "xtoy")

        class Y:
##            def setX(self, x):         RM.R(self, x, "xy")
            def setX(self, x):         RM.R(x, self, "xtoy")  # though bi, there is
            def getX(self):     return RM.P(self, "xtoy")
            def clearX(self):          RM.NR(self, self.getX(), "xtoy")

        x1 = X()
        x2 = X()
        y1 = Y()
        y2 = Y()
        self.onetomanyasserts(x1,x2,y1,y2,yapi=1)


    def checkOneToMany_XPluralApi_YSingularApi_Alt(self):
        """
        Alternative implentation, using "directional" and B()
         _________________        _______________
        |        X        |      |       Y       |
        |_________________|      |_______________|
        |                 |      |               |
        |addY(self, y)    |1    *|setX(self, x)  |
        |getAllY(self)    |<---->|getX(self)     |
        |removeY(self, y) |      |clearX(self)   |
```

```
            |_________________|      |_______________|

            """
            class X:
                def __init__(self):        RM.ER("xtoy", "onetomany", "directional")
                def addY(self, y):         RM.R(self, y, "xtoy")
                def getAllY(self):  return RM.PS(self, "xtoy")
                def removeY(self, y):      RM.NR(self, y, "xtoy")

            class Y:
                def setX(self, x):         RM.R(x, self, "xtoy")
                def getX(self):     return RM.B(self, "xtoy")
                def clearX(self):          RM.NR(self.getX(), self, "xtoy")

            x1 = X()
            x2 = X()
            y1 = Y()
            y2 = Y()
            self.onetomanyasserts(x1,x2,y1,y2,yapi=1)


        def onetomanyasserts(self, x1, x2, y1, y2, yapi=0):

            def assertallclear():
                assert x1.getAllY() == []
                assert x2.getAllY() == []
                if yapi:
                    assert y1.getX() == None
                    assert y2.getX() == None
            def assertSituation00():
                assert x1.getAllY() == [y1]
                if yapi:
                    assert y1.getX() == x1

            # Initial situation
            assertallclear()

            # clearing pointers that do not exist, should be ok.
            x1.removeY(y1)
            assertallclear()
            x1.removeY(y2)
            assertallclear()
            x2.removeY(y1)
            assertallclear()
            x2.removeY(y2)
            assertallclear()
            if yapi:
                y1.clearX()
                assertallclear()
                y2.clearX()
                assertallclear()


            """
            +--------------------------------+
            |Add a single X to Y relationship|
            +--------------------------------+
            """
            x1.addY(y1)
            """
             ,-----.      ,-----.
            (  x1   )--->(  y1   )
             `-----'      `-----'
            """
            assertSituation00()
            # now remove it
```

```
            x1.removeY(y1)
            assertallclear()

            # Add initial relationship, from the y side
            if yapi:
                y1.setX(x1)
                assertSituation00()
                # now remove it, from the y side
                y1.clearX()
                assertallclear()


            """
            +-------------------------------------------+
            |Add two relationships coming from a single X|
            |to multiple Y's.                            |
            +-------------------------------------------+
            """
            def assertSituation01():
                """
                 ,-----.      ,-----.
                (  x1   )--->(  y1   )
                 `-----'.     `-----'
                        |
                         \      ,-----.
                          `--->(  y2   )
                                `-----'
                """
                assert x1.getAllY() == [y1, y2], "Actual situation %s" % x1.getAllY()
                if yapi:
                    assert y1.getX() == x1
                    assert y2.getX() == x1
            def assertSituation02():
                """
                 ,-----.      ,-----.
                (  x1   )    (  y1   )
                 `-----'.     `-----'
                        |
                         \      ,-----.
                          `--->(  y2   )
                                `-----'
                """
                assert x1.getAllY() == [y2]
                if yapi:
                    assert y1.getX() == None
                    assert y2.getX() == x1
            # Add two relationships, from x API
            x1.addY(y1)
            x1.addY(y2)
            assertSituation01()
            # now remove y1
            x1.removeY(y1)
            assertSituation02()
            # now remove y2
            x1.removeY(y2)
            assertallclear()

            # Add two relationships, from the y api side.
            if yapi:
                assertallclear()
                y1.setX(x1)
                y2.setX(x1)
                assertSituation01()
                # now remove y1
                y1.clearX()
                assertSituation02()
```

```
            # now remove y1
            y2.clearX()
            assertallclear()



        """
        +---------------------------+
        |Add same relationship twice|
        +---------------------------+
        """
        def assertSituation03():
            assert x1.getAllY() == [y1]
            if yapi:
                assert y1.getX() == x1
        def assertSituation04():
            assert x1.getAllY() == [y1, y2]
            if yapi:
                assert y2.getX() == x1
        def assertSituation05():
            if yapi:
                assert y1.getX() == None
            assert x1.getAllY() == [y2]
        x1.addY(y1)
        x1.addY(y1)
        """
         ,-----.      ,-----.
        (  x1   )--->(  y1   )
         `-----'      `-----'

                       ,-----.
                      (  y2   )
                       `-----'
        """
        assertSituation03()

        x1.addY(y2)
        x1.addY(y2)
        """
         ,-----.      ,-----.
        (  x1   )--->(  y1   )
         `-----' `.   `-----'
                   `-.
                      `>,-----.
                      (  y2   )
                       `-----'
        """
        assertSituation04()
        # now remove y1 (again, twice, just to check robustness)
        x1.removeY(y1)
        x1.removeY(y1)
        """
         ,-----.      ,-----.
        (  x1   )--->(  y1   )
         `-----'      `-----'

                       ,-----.
                      (  y2   )
                       `-----'
        """
        assertSituation05()
        # now remove y2 twice
        x1.removeY(y2)
        x1.removeY(y2)

        assertallclear()
```

```python


        """
        +----------------------------------------+
        |Add same relationship twice, from Y side|
        +----------------------------------------+
        """
        if yapi:
            y1.setX(x1)
            y1.setX(x1)
            """
             ,-----.      ,-----.
            (  x1   )--->(  y1   )
             `-----'      `-----'

                           ,-----.
                          (  y2   )
                           `-----'
            """
            assertSituation03()
            y2.setX(x1)
            y2.setX(x1)
            """
             ,-----.      ,-----.
            (  x1   )--->(  y1   )
             `-----' `.   `-----'
                       `-.
                          `>,-----.
                          (  y2   )
                           `-----'
            """
            assertSituation04()
            # now remove y1, from Y side (again, twice, just to check robustness)
            y1.clearX()
            y1.clearX()
            """
             ,-----.      ,-----.
            (  x1   )--->(  y1   )
             `-----'      `-----'

                           ,-----.
                          (  y2   )
                           `-----'
            """
            assertSituation05()
            # now remove y2 twice, from Y side
            y2.clearX()
            y2.clearX()

            assertallclear()


        """
        +-----------------------------------------------------+
        |Add two relationships, then add a third              |
        |relationship which effects an previous relationship.|
        +-----------------------------------------------------+
        """
        # Make x1 -> y1,y2
        assertallclear()
        x1.addY(y1)
        x1.addY(y2)
        """
          ,-----.      ,-----.
         (  x1   )--->(  y1   )
          `-----' `.   `-----'
```

```
                    `-.
            ,-----.     `>,-----.
           (  x2   )    (  y2   )
            `-----'      `-----'
        """
        # Now make x2 -> y1
        x2.addY(y1)
        """
         ,-----.        ,-----.
        (  x1   )     >(  y1   )
         `-----' `. ,'  `-----'
                   /-.
            ,-----,'    `>,-----.
           (  x2   )    (  y2   )
            `-----'      `-----'
        After much thought, I believe the addition of the x2 -> y1 relationship
         should extinguish the existing x1 -> y1 since y's can only be pointed to by
        If you want to keep the existing x1 -> y1 then you actually are describing t
         many to many, directional, no y api, scenario.
        """
        assert x1.getAllY() == [y2]
        assert x2.getAllY() == [y1]
        if yapi:
            assert y1.getX() == x2
            assert y2.getX() == x1

        x1.removeY(y2)
        x2.removeY(y1)
        assertallclear()


        """
        +-----------------------------------------------------+
        |Two different X's point to the same Y.               |
        |Again enforcement that y only pointed to by one X,  |
        |and that the original relationship is extinguished.|
        +-----------------------------------------------------+
        """
        def assertSituation06():
            assert x1.getAllY() == []
            assert x2.getAllY() == [y1]
            if yapi:
                assert y1.getX() == x2

        x1.addY(y1)
        """
         ,-----.       ,-----.
        (  x1   )--->(  y1   )
         `-----'       `-----'
           ,-----.
          (  x2   )
           `-----'
        """
        x2.addY(y1)
        """
         ,-----.       ,-----.
        (  x1   )   .>(  y1   )
         `-----'    |  `-----'
           ,-----.  |
          (  x2   --+
           `-----'
        """
        assertSituation06()

        x2.removeY(y1)
        assertallclear()
```

```python


            # Same as above, except wired via Y's API
            if yapi:
                y1.setX(x1)
                y1.setX(x2)
                assertSituation06()
                y1.clearX()
                assertallclear()

            assertallclear()




    def suite():
        suite1 = unittest.makeSuite(TestCase01_OneToOne, 'check')
        suite2 = unittest.makeSuite(TestCase02_OneToMany, 'check')
        alltests = unittest.TestSuite((suite1,suite2))
        return alltests

    def main():
        """ Run all the suites.  To run via a gui, then
                python unittestgui.py NestedDictionaryTest.suite
            Note that I run with VERBOSITY on HIGH  :-) just like in the old days
            with pyUnit for python 2.0
            Simply call
              runner = unittest.TextTestRunner(descriptions=0, verbosity=2)
            The default arguments are descriptions=1, verbosity=1
        """
        runner = unittest.TextTestRunner(descriptions = 0, verbosity = 2) # default is d
        #runner = unittest.TextTestRunner(descriptions=0, verbosity=1) # default is desc
        runner.run(suite())

    if __name__ == '__main__':
        main()


```

```python
class RelationshipManager:
  def __init__(self):       # Constructor
      self.Relationships = []
  def AddRelationship(self, From, To, RelId=1):
      if not self.FindObjects(From, To, RelId):
        self.Relationships.append( (From, To, RelId) ) # assoc obj
  def RemoveRelationships(self, From, To, RelId=1):
      if not From or not To:
          return
      lzt = self.FindObjects(From, To, RelId)
      if lzt:
          for association in lzt:
              self.Relationships.remove(association)
  def FindObjects(self, From=None, To=None, RelId=1):
      resultlist = []
      match = lambda obj,list,index : obj==list[index] or obj==None
      for association in self.Relationships:
          if match(From,association,0) and match(To,association,1) and RelId==associ
              if From==None:
                  resultlist.append(association[0])
              elif To==None:
                  resultlist.append(association[1])
              else:
                  resultlist.append(association)
      return resultlist
  def FindObject(self, From=None, To=None, RelId=1):
      lzt = self.FindObjects(From, To, RelId)
      if lzt:
        return lzt[0]
      else:
        return None
  def Clear(self):
      del self.Relationships[0:]

import unittest, random

class TestCase00(unittest.TestCase):
    def setUp(self):
        self.rm = RelationshipManager()
    def checkBasic00(self):
        self.rm.AddRelationship('a','b')
        self.rm.AddRelationship('a','c')
        assert self.rm.FindObjects('a',None) == ['b','c']
        assert self.rm.FindObjects(None,'a') == []
        assert self.rm.FindObjects(None,'b') == ['a']
        assert self.rm.FindObjects(None,'c') == ['a']
    def checkBasic01Singular(self):
        self.rm.AddRelationship('a','b')
        self.rm.AddRelationship('a','c')
        assert self.rm.FindObject(None,'b') == 'a'
        assert self.rm.FindObject(None,'c') == 'a'
        assert self.rm.FindObject('a',None) == 'b' # could have been 'c' - arbitrary

def suite():
    suite1 = unittest.makeSuite(TestCase00,'check')
    alltests = unittest.TestSuite( (suite1,) )
    return alltests

def main():
    """ Run all the suites.  To run via a gui, then
            python unittestgui.py NestedDictionaryTest.suite
        Note that I run with VERBOSITY on HIGH  :-) just like in the old days
        with pyUnit for python 2.0
        Simply call
          runner = unittest.TextTestRunner(descriptions=0, verbosity=2)
        The default arguments are descriptions=1, verbosity=1
```

```
        """
        runner = unittest.TextTestRunner(descriptions=0, verbosity=2) # default is descr
        #runner = unittest.TextTestRunner(descriptions=0, verbosity=1) # default is desc
        runner.run( suite() )

    if __name__ == '__main__':
        main()
```