Dynamic Inheritance - fun with IQuackFu

Boo's IQuackFu interface enables adding behavior to a class at runtime. This example shows how to add dynamic inheritance to boo, also called mixin programming or delegation.

The following Mixup class can be used to add new class behavior to an object at runtime. First, the code for Mixup:

```
// Bill Wood, Nov 2004
import System.Reflection

class Mixup (IQuackFu):

 static defaultBF = BindingFlags.Instance |
BindingFlags.Static |\
      BindingFlags.Public |
BindingFlags.FlattenHierarchy

 static invokeBF = defaultBF |
BindingFlags.InvokeMethod

 static getPropertyBF = defaultBF |
BindingFlags.GetProperty |
BindingFlags.GetField

 static setPropertyBF = defaultBF |
BindingFlags.SetProperty |
BindingFlags.SetField

 private methods = {}

 private properties = {}

 def QuackInvoke(name as string, args as
```

```
(object)) as object:
  o = methods[name]
  if o is null:
   raise
System.InvalidOperationException("Method
${name} not found in class
${self.GetType()}")
  return o.GetType().InvokeMember(name,
invokeBF, null, o, args)

 def QuackGet(name as string, params as
(object)) as object:
  o = properties[name]
  if o is null:
   raise
System.InvalidOperationException("Member
${name} not found in class
${self.GetType()}")
  return o.GetType().InvokeMember(name,
getPropertyBF, null, o, null)

 def QuackSet(name as string, params as
(object), value as object) as object:
  o = properties[name]
  if o is null:
   raise
System.InvalidOperationException("Member
${name} not found in class
${self.GetType()}")
  return o.GetType().InvokeMember(name,
setPropertyBF, null, o, (value,))
```

```
def mixup(o as object):
 t = o.GetType()
 for m in t.GetMethods(invokeBF):
  if methods[m.Name] is null:
   methods[m.Name] = o
 for m in t.GetProperties(getPropertyBF):
  if properties[m.Name] is null:
   properties[m.Name] = o
 for m in t.GetFields(getPropertyBF):
  if properties[m.Name] is null:
   properties[m.Name] = o
 return o

def unmixup(o as object):
 helper = def (hash as Hash):
  d = []
  for m in hash:
   if m.Value is o:
    d.Add(m.Key)
  for s in d:
   hash.Remove(s)
 helper(methods)
 helper(properties)
 return o
```

```
 def constructor():
  mixup(self)
```

Mixup can be instantiated directly:

```
class Test:
 def say(s):
  print(s)

z as duck = Mixup()  // z is a mixed up
class!
l = z.mixup(List())  // add a List() to z
z.mixup(Test())  // add a Test() to z
z.Push("hi from x")  // Push using List.Push
z.say("${z.Count}, ${z.Pop()}, ${z.Count}")
// Output: 1, hi from z, 0
print z isa List  // not REALLY a list, just
has List behavior
z.unmixup(l)  // remove List behavior
z.Push("hi") //
System.NullReferenceException: Method Push
not found in class Mixup
```

Or, it can be subclassed:

```
class X(Mixup):    // X subclasses Mixup
 private _list = []
 def constructor():
  mixup(_list)  // add list behavior to X
  for i in range(10):
   _list.Add(i) // initialize _list

x as duck = X()
x.Push("hi from x")
print x.Pop(), x.Pop(), x.Pop() // hi from x
9 8
```