```
import operator

def magic(C):
  def onDecorator(aClass):
    class onInstance:
      def __init__(self, *args, **kargs):
        self.wrapped = aClass(*args, **kargs)
        setattr(C, 'referents', dict())
      def __getattr__(self, attr):
        if attr == 'wrapped': return self.__dict__[attr]
        operator.setitem(getattr(C, 'referents'), attr, self.wrapped)
        setattr(C, attr, self.wrapped.__class__.__dict__[attr])
        return getattr(self.wrapped, attr)
      def __setattr__(self, attr, value):
        if attr == 'wrapped': self.__dict__[attr] = value
        else: setattr(self.wrapped, attr, value)
    return onInstance
  return onDecorator

def delegation(self, attribute):
  for x in self.referents.items():
    if attribute in x[1].__dict__: return x[1].__dict__[attribute]
  raise AttributeError("Attribute «{}» doesn't exist".format(attribute))

class SharedMem(type):
  def __new__(meta, classname, supers, classdict):
    classdict['__getattr__'] = delegation
    return type.__new__(meta, classname, supers, classdict)

Empty = SharedMem('Empty', (), dict(Empty.__dict__))
```