```
# TheCompanyClass.py
""" Module illustrates how a 2-dim array can be a
claaa
attribute. A cost-inventory=purchase order application
illustrates it
all.
"""
from numpy import *

class Company(object):
    """
Class  that can be used to process purchase orders
    using a cost and inventory array.


  Attributes:
        I : A numpy 2D float array. The Inventory array
        C : A numpy 2D
float array  The Cost array.
       TV : total value [float]

    Class Invariants. I
and C have the same row and column dimensions and
    TV is the value of the total inventory.


    The column dimension of I and C equals the number of products.
    The row dimension of
I and C equals the number of factories.

    A purchase order array is a 1-dim numpy array of
nonnegative floats whose
    dimension equals the numer of products.


"""
    def __init__(self,Inventory,Cost):
        """ Returns a
reference to a Company object.

        PreC: Inventory and Cost are 2-dim numpy arrays
of the same size.
        """
        self.I = Inventory
        self.C = Cost

      (m,n) = self.I.shape
        # Compute the total value.
        TV = 0
        for k in
range(m):
            # Add in the value of the inventory in factory k
            for j in
range(n):
                TV+=Inventory[k,j]*Cost[k,j]
        self.TV = TV

    def
show(self):
       """ Displays the Company object self.

"""
       print '\nThe Inventory Array:\n'
       print self.I
       print
'\nThe Cost Array:\n'
       print self.C
       print '\nTotalValue = %1d' % self.TV



    def Order(self,PO):
        """ Returns a 1-dim numpy array whose k-th
entry is the cost when the
```

```
        PO is processed by factory k.

        PreC: PO is a
valid purchase order
        """
        C = self.C
        (m,n) = C.shape

   theCosts = zeros(m)
        for k in range(m):
            # Compute the cost  to factory
k.
            for j in range(n):
                theCosts[k]+= C[k,j]*PO[j]
        return
theCosts

    def CanDo(self,PO):
        """ Returns a list of valid
factory indices that indicate
        which factories have sufficient inventory to fill PO. The
empty
        list is returned if no factory has sufficient inventory.

        PreC:
PO is a valid purchase order
        """
        I = self.I
        (m,n) =
I.shape
        Who = []
        for k in range(m):
            # Check if factory k has enough
inventory
            if all(I[k,:]>=PO):
                Who.append(k)
        return Who


    def theCheapest(self,PO):
        """ Returns the tuple (kMin,costMin)
where kMin is the index of
        the factory that can most cheaply fill PO and costMin is the
associated
        bill. Returns None if no factory has sufficient inventory.


PreC: PO is a valid purchase order
        """
        # Determine the costs for
each factory and who can fill the PO
        theCosts = self.Order(PO)
        Who =
self.CanDo(PO)
        if len(Who)==0:
            # No factory has sufficient inventory

     return None
        else:
            costMin = inf
            # Look for the minimum
cost among the factories that have
            # sufficient inventory.
            for k in
Who:
                if theCosts[k] < costMin:
                    cMin = theCosts[k]

             kMin = k
            return (kMin,costMin)


    def
Update(self,k,PO):
        """ Update self.I and self.TV assuming that factory
```

```
k
        processed purchase order PO

        PreC: k is a valid factory index and PO
is a valid purchase order
        """
        n = len(PO)
        I = self.I

    C = self.C
        TV = self.TV
        for j in range(n):
            # Deplete the
inventory of product j and reduce self.TV accordingly.
            I[k,j] = I[k,j] - PO[j]

       TV = TV - C[k,j]*PO[j]
        self.I = I
        self.TV = TV
```