CS 100: Section Assignment 12

(For the week of April 26)

Section assignments are discussed in section and are not submitted for grading. They relate to recent lecture topics and usually to the current Programming Assignment. Prelim questions are based on Section Assignments, Programming Assignments, and Lecture examples.

1. To the class `Retailer`, add an instance method `iValue(int i)` that returns the total value of *i*-th store's inventory. Add a fragment to `ShowRetailer` that prints the index of the store that has the greatest inventory in value.

2. To the class `Retailer` add a method

```
public void ShiftInventory(int i1, int i2, int j)
```

that depletes to zero store `i1`'s inventory of item `j` and increases store `i2`'s inventory of item `j` by the same amount.

3. To the class `MailOrder` add an integer-valued instance method `buyOut(int i,double z)` that returns the cost to a buyer at `z` who orders everything in the *i*-th store and has to pay for shipping.

4. To the class `MailOrder` add another integer-valued instance method `shipTo(int i)` that returns the cost of shipping all the inventory from all the stores to store *i*. Add a fragment to `ShowMailOrder` that prints the index of the best store to keep open given that all inventory is to be shipped to a single store. I.e., determine an index `i` (0 <= `i` < `nStores)` so that `shipTo(i)` is minimized.

```
public class Retailer
{
   protected int nStores;     // Number of stores.
   protected int nItems;      // Number of items.
   protected int[][] price;   // Price array: price[i][j] is the price (in dollars) of item j in store i.
   protected int[][] inv;     // Inventory array: inv[i][j] is the inventory of item j in store i.

   // The Constructor.
   public Retailer(int[][] P, int[][] I)
   // P is the price array and I is the inventory array. They are the same size. The number
   // of stores equals the number of rows and the number of retail items equals the number of columns.
   {
      nStores = P.length;
      nItems = P[0].length;
      price = copy(P);
      inv   = copy(I);
   }


   // i satisfies 0 <= i < nStores.
   // PO represents a purchase order. It has length nItems and P0[k] is the
   // quantity of item k that the customer wishes to buy.
   // Yields true if the ith store can handle the purchase order, i.e.,
   // PO[j] <= inv[i][j], j=0,...,nItems-1.
   public boolean iCanDo(int i, int[] PO)
   {
      int j=0;
      while(j<nItems && PO[j]<=inv[i][j])
         j++;
      return (j==nItems);
   }
```

```
// i satisfies 0 <= i < nStores.
// PO represents a purchase order. It has length nItems and P0[j] is the
// quantity of item j that the customer wishes to buy.
// Yields the cost of processing the purchase order at store i.
public int iCost(int i, int[] PO)
{
   int value = 0;
   for(int j=0;j<nItems;j++)
      value = value + price[i][j]*PO[j];
   return value;
}

// PO represents a purchase order. It has length nItems and P0[j] is the
// quantity of item j that the customer wishes to buy.
// If no store can process the purchase order then an appropriate message is
// printed and each store increases its inventory just enough so that it can
// process the PO "the next time".
// Otherwise, the index of the store that can most cheaply fill the PO is printed
// along with the minimum cost. The store that processes the order has its inventory
// depleted accordingly.
public void processPO(int[] PO)
{
   int i = 0;
   while (i<nStores && !iCanDo(i,PO))
      i++;
   if (i==nStores)
   {
      // No store has sufficient inventory.
      System.out.println("No single store can process this order at this time.");
      updateInv(PO);
   }
   else
   {
      // Find the store that can most cheaply fill the PO.
      int minCost = iCost(i,PO);
      int bestStore = i;
      for (int s=i+1;s<nStores;s++)
         if (iCanDo(s,PO) && iCost(s,PO)<minCost)
         {
            bestStore = s;
            minCost = iCost(s,PO);
         }
      System.out.println("Price at store " + bestStore + " is " + minCost);
      updateInv(bestStore,PO);
   }
 }

 // PO represents a purchase order. It has length nItems and P0[j] is the
 // quantity of item j that the customer wishes to buy. The inventory of every
 // store is increased (if necessary) so that it can just handle the PO.
 public void updateInv(int[] PO)
 {
    for(int i=0;i<nStores;i++)
       for(int j=0;j<nItems;j++)
           if (inv[i][j]<PO[j])
              inv[i][j] = PO[j];
 }

 // i satisfies 0 <= i < nStores
 // PO represents a purchase order. It has length nItems and P0[j] is the
 // quantity of item j that the customer wishes to buy.
 // The inventory for store i is adjusted to reflect the processing of the PO.
 public void updateInv(int i, int[] PO)
 {
    for(int j=0;j<nItems;j++)
       inv[i][j] = inv[i][j] - PO[j];
 }
```

```
    // Displays the inventory array.
    public void showInv()
    {
       System.out.println(" ");
       for(int i=0;i<nStores;i++)
       {
          for(int j=0;j<nItems;j++)
             Format.print(System.out," %3d ",inv[i][j]);
          System.out.println(" ");
       }
       System.out.println(" ");
    }

   // Yields a reference to an array that is an identical copy of A.
   public static int[][] copy(int[][] A)
   {
      int nRows = A.length;
      int nCols = A[0].length;
      int[][] B = new int[nRows][nCols];
      for(int i=0;i<nRows;i++)
         for(int j=0;j<nCols;j++)
            B[i][j] = A[i][j];
      return B;
   }
}


import java.io.*;
public class ShowRetailer {
       public static void main(String args[]){
          // A sample price array.
          int[][] eg_P = { { 32 , 20 , 22 , 12 , 19 , 17 , 31 , 12} ,
                            { 34 , 22 , 25 , 10 , 13 , 19 , 33 , 15} ,
                            { 34 , 21 , 24 , 13 , 17 , 17 , 34 , 12} };
          // A sample inventory array.
          int[][] eg_I = { { 10 , 20 , 60 , 10 , 40 , 70 , 20 , 10} ,
                           { 90 , 40 , 80 , 50 , 30 , 40 , 80 , 90} ,
                           { 30 , 60 , 40 , 30 , 90 , 70 , 30 , 80} };
          // Build a store and process three customers.
          Retailer TallMart = new Retailer(eg_P,eg_I);
          TallMart.showInv();
          int[] PO1 = {10,10,10,10,10,10,10,10}; TallMart.processPO(PO1); TallMart.showInv();
          int[] PO2 = {20,20,20,20,20,20,20,20}; TallMart.processPO(PO2); TallMart.showInv();
          int[] PO3 = {100,0,0,0,0,0,0,0};       TallMart.processPO(PO3); TallMart.showInv();
       }
}
/* Output:

  10   20   60   10   40   70   20   10
  90   40   80   50   30   40   80   90
  30   60   40   30   90   70   30   80

Price at store 0 is 1650

   0   10   50    0   30   60   10    0
  90   40   80   50   30   40   80   90
  30   60   40   30   90   70   30   80

Price at store 1 is 3420

   0   10   50    0   30   60   10    0
  70   20   60   30   10   20   60   70
  30   60   40   30   90   70   30   80

No single store can process this order at this time.

 100   10   50    0   30   60   10    0
 100   20   60   30   10   20   60   70
 100   60   40   30   90   70   30   80

*/
```

```
public class MailOrder extends Retailer
{
    /* Inherits variables
    protected int nStores;     // Number of stores.
    protected int nItems;      // Number of items.
    protected int[][] price;   // Price array: price[i][j] is the price (in dollars) of item j in store i.
    protected int[][] inv;     // Inventory array: inv[i][j] is the inventory of item j in store i.
    */
    protected double[] weight;   // Weight array.   weight[j] is the weight(kg) of item j,
                                 // j=0,...,nItems-1.
    protected double[] location; // Store locations. The stores are located on the x-axis and for
                                // i = 0,1,...,nStores-1, location[i] is its x-coordinate (in km).
    protected double   rate;     // Shipping rate. The cost to ship one kg of merchandise one km


    public MailOrder(int[][] P, int[][] I, double[] w, double[] x, double r) {
       // P is the price array and I is the inventory array. They are the same size. The number
      // of stores equals the number of rows and the number of retail items equals the number of columns.
      // w is the weight array and x is the location array.

          super(P,I);
          location = copy(x);
          weight   = copy(w);
          rate = r;
       }

    // Yields a reference to an array that is an exact copy of a.
    public static double[] copy(double[] a){
       int n = a.length;
       double[] b = new double[n];
       for(int i=0;i<n;i++)
          b[i] = a[i];
       return b;
    }

    // i satisfies 0 <= i < nStores.
    // PO represents a purchase order. It has length nItems and P0[j] is the
    // quantity of item j that the customer wishes to buy.
    // Yields the cost of processing the purchase order at store i.
    public int iCost(int i, int[] PO, double x){
       // Find the value of the merchandise.
       int value = iCost(i,PO);

       // Determine the total shipping cost.
       double shippingWeight = 0;
       for(int j=0;j<nItems;j++)
          shippingWeight = shippingWeight + weight[j]*PO[j];
       int shippingCost = (int)(rate*shippingWeight*Math.abs(x-location[i]));

       return value + shippingCost;
    }

   // PO represents a purchase order. It has length nItems and P0[j] is the
   // quantity of item j that the customer wishes to buy.
   // If no store can process the purchase order then an appropriate message is
   // printed and each store increases its inventory just enough so that it can
   // process the PO "the next time".
   // Otherwise, the index of the store that can most cheaply fill the PO is printed
   // along with the minimum cost. The store that processes the order has its inventory
   // depleted accordingly.
   public void processPO(int[] PO, double x)
   {
      int i = 0;
      while (i<nStores && !iCanDo(i,PO))
         i++;
      if (i==nStores){
         // No store has sufficient inventory.
         System.out.println("No single store can process this order at this time.");
         updateInv(PO);
      }
```

```
      else {
         // Find the store that can most cheaply fill the PO.
         // NOTE: iCost is overloaded. Java can tell by the signature to use the version
         // above and not the version in the parent class Retailer.
         int minCost = iCost(i,PO,x);
         int bestStore = i;
         for (int s=i+1;s<nStores;s++)
            if (iCanDo(s,PO) && iCost(s,PO,x)<minCost) {
               bestStore = s;
               minCost = iCost(s,PO,x);
            }
         System.out.println("Price at store " + bestStore + " is " + minCost);
         updateInv(bestStore,PO);
      }
    }
}

import java.io.*;
public class ShowMailOrder {
   public static void main(String args[]){
        // A sample price array.
        int[][] eg_P = { { 32 , 20 , 22 , 12 , 19 , 17 , 31 , 12} ,
                         { 34 , 22 , 25 , 10 , 13 , 19 , 33 , 15} ,
                         { 34 , 21 , 24 , 13 , 17 , 17 , 34 , 12} };

        // A sample inventory array.
        int[][] eg_I = { { 10 , 20 , 60 , 10 , 40 , 70 , 20 , 10} ,
                         { 90 , 40 , 80 , 50 , 30 , 40 , 80 , 90} ,
                         { 30 , 60 , 40 , 30 , 90 , 70 , 30 , 80} };

       // A sample weight array
        double[] eg_w = { .10 , .20 , .05 , .15 , .10 , .08 , .07 , .30};

       // A sample location array
       double[] eg_x = {30.2 , 7.6 , 48.9};

       double eg_r = 2.0;

        // Build a mail order business and process three customers located at x = 10, 30, and 50.
        MailOrder TallMart = new MailOrder(eg_P,eg_I,eg_w, eg_x, eg_r);
        TallMart.showInv();
        int[] PO1 = {10,10,10,10,10,10,10,10}; TallMart.processPO(PO1,10); TallMart.showInv();
        int[] PO2 = {20,20,20,20,20,20,20,20}; TallMart.processPO(PO2,30); TallMart.showInv();
         int[] PO3 = {100,0,0,0,0,0,0,0};       TallMart.processPO(PO3,50); TallMart.showInv();
   }
}
/* Output:

  10   20   60   10   40   70   20   10
  90   40   80   50   30   40   80   90
  30   60   40   30   90   70   30   80

Price at store 1 is 1760

  10   20   60   10   40   70   20   10
  80   30   70   40   20   30   70   80
  30   60   40   30   90   70   30   80

Price at store 2 is 4233

  10   20   60   10   40   70   20   10
  80   30   70   40   20   30   70   80
  10   40   20   10   70   50   10   60

No single store can process this order at this time.

 100   20   60   10   40   70   20   10
 100   30   70   40   20   30   70   80
 100   40   20   10   70   50   10   60

*/
```