**The assignment**

You are to write a program to perform an analysis of inventory for a small wholesale food products company.  Your program will do the following:

* Read data from a text file that describes food items, warehouses, and transactions over a several day period.
* Use the transaction information to adjust the inventories for each warehouse.
* Print out a report to the console that answers several questions about the data.

You are only required to write the program that processes the data (using the Inventory class) and prints the report.  During this process, however, you may elect to write additional classes or change prewritten classes that I give you.  You are allowed to do this, but you must hand in all files needed to run your solution.  Full credit will be given to programs that print correct reports for our large test cases.

Before starting, you should download these files:

Item.java   
Inventory.java   
FoodItem.java   
data1.txt - small test case  
data2.txt - a medium test case  
data3.txt - a large test case

To perform the required data processing, write an application (a Java program with a main method) that reads in the data file and prints out the report to the console.  Either read from data.txt or prompt the user to enter a filename (relative to the current directory).  Here is an example of a small data file:

|  |
| --- |
| FoodItem - UPC Code: 0353264991 Shelf life: 2 Name: chestnut puree with vanilla  FoodItem - UPC Code: 0984523912 Shelf life: 1 Name: the orange box  Warehouse - Columbus  Warehouse - Scottsdale  Warehouse - Tacoma  Start date: 05/01/2010  Receive: 0984523912 7 Tacoma  Receive: 0353264991 6 Tacoma  Receive: 0353264991 2 Scottsdale  Request: 0984523912 5 Tacoma  Next day:  Request: 0984523912 4 Columbus  Request: 0353264991 4 Columbus  Next day:  Receive: 0353264991 9 Columbus  Receive: 0984523912 2 Columbus  Request: 0353264991 3 Scottsdale  Next day:  Request: 0984523912 8 Scottsdale  Receive: 0984523912 8 Columbus  Request: 0984523912 3 Tacoma  Request: 0353264991 10 Tacoma  Request: 0984523912 5 Tacoma  End |

Each line of the data file provides one piece of information to your program.  The first word(s) of each line indicate what to do with the data on that line.  Your program should read the file one line at a time and process the lines like this:

* **Food Item** ‑ UPC Code: *number* Shelf life: *number* Name: *name goes here*

Each food item is specified on a line that starts with the words 'Food Item'.  A food item has a UPC code (a single string of numbers), a shelf life (in days), and a name (a string of words that ends at the end of the line).  This instruction tells your program that this food item will be used, so your program will need to store this food item information.  Note that the UPC code uniquely identifies each item.

* **Warehouse** ‑ *Warehouse name goes here*

Each warehouse inventory is specified on a line that starts with the word 'Warehouse'.  A warehouse has a name (a string of words that ends at the end of the line) that is the name of the city where the warehouse is located.  This instruction tells your program that this warehouse exists, so your program will need to store the warehouse name and keep track of an inventory for this warehouse.  Every warehouse name is unique.

* **Start Date**: *MM*/*DD*/*YYYY*

The start date indicates a month, day, and a year for which record keeping should begin.  (This instruction format will always have the same number of characters.)  The start date is used to keep track of when transactions happen.  The instructions on the following lines all happen on some effective date.  Your program will need to keep track of the effective date.

* **Receive**: *UPCnumber* *quantity* *warehouse name goes here*

The receive command indicates that a warehouse has received some food items on the current effective date.  Each receive instruction has a UPC code (indicating the item received), a positive quantity, and the name of the warehouse receiving the item.  Your program should record the item and quantity as received in the correct warehouse's inventory.  In addition, the inventory should keep track of the day on which this item will expire in the inventory.  Compute the expiration date by adding the item's shelf life to the current effective date.

* **Request**: *UPCnumber* *quantity* *warehouse name goes here*

The request command indicates that a food item has been requested for delivery from a warehouse.  Each request instruction has a UPC code (indicating the item requested), a positive quantity, and the name of the warehouse that should attempt to deliver the item.  Your program should remove that quantity of items from the correct warehouse's inventory.  It is possible that the warehouse may not have that item, or may run out.  If multiple groups of the item exist in that warehouse's inventory, remove matching items with the oldest expiration date first.

* **Next Day**:

The next day instruction advances the effective date by one day.  All following instructions should be treated as occurring on the following day.  Your program should advance the effective date.  In addition, your program should remove any expired inventory from the warehouses when this instruction is processed.  (When a food item's expiration day is reached, it expires at the *start* of that day.)

* **End**:

Indicates the end of the file.  Your program should stop reading the instructions when it encounters this one.

After your program processes all the data file instructions, your program should analyze the data to answer these questions by printing results to the console:

* **Which products do not exist in any warehouse?** Print out this title, "Unstocked products", followed by a sorted list of products, one per line.  (Use the FoodItem toString method.)  Do not print expiration dates, warehouse names, or quantities.  Just list the products (no duplicates) that are absent from every warehouse.
* **Which products still exist in every warehouse?** Print out this title, "Fully stocked products", followed by a sorted list of products, one per line.  (Use the FoodItem toString method.) Do not print expiration dates, warehouse names, or quantities.  Just list the products (no duplicates) that are in every warehouse.
* **What is the total quantity of items (of any kind) in all the warehouses?**  (This should only be the quantity of unexpired items, of course.)  Sum up the remaining item quantities and print out "Items remaining: " followed by the quantity.

All output should be to the console. Disable any debugging output you have prior to submitting your program - we need clean, easy-to-read answers to our questions in order to grade your programs.

After your program finishes outputting data, it should terminate.

Full credit will be given to original, well documented solutions that print correct reports for the large data files we will use.    Ouput to a GUI instead of the console will provide 10 points extra credit.

**My recommendations**

This assignment is designed to be straightforward if you use Java collections (sets and maps), and very hard if you do not.  Use Java collections to make grouping and looking up data easy.  At a minimum, I recommend at least:

* A Map<String, FoodItem> for mapping UPC codes to FoodItem objects to make it easy to find the food item that matches a UPC.  (You will populate this map.)
* A Map<String, Inventory<FoodItem>> that maps warehouse names to an inventory.  That way, you can easily find the inventory for a given warehouse name.  (You will populate this map.)
* Liberal use of Set<FoodItem> objects to help you compute the report.  (You will want to use the retainAll and removeAll methods on sets.)
* Use the keySet method on a Map to get the set of keys from the map.

When looping through the elements of a set or list, use the 'for each' loop in Java wherever possible.  (You cannot loop through the elements in a set using an index.)

I recommend adding three helper methods to the Inventory class:  Add a method to return a set of the FoodItem objects in the inventory.  Add a method to remove all expired items from an inventory.  Add a method to return the total quantity of items in an inventory.

To read the data, I recommend using the nextLine method in the Scanner class to read a line of text as a String, then .trim() to remove any trailing space.  (Remember, though, that Strings are immutable.)  After you have the line of text, either use a second Scanner on that string to break it into words or numbers, or break it into sections by searching through the string with indexOf to find the positions of marker text.

I also recommend *not* creating any additional classes for this assignment (except for the one that contains your main method, of course).  It is designed for you to be able to write entirely within a reasonably-sized main method.  The existing Inventory, Item, and FoodItem classes, in combination with the Java collections, are very sufficient for containing the data.