**The assignment**

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

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

You are 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 the large test cases.

Before starting, you should download these files:

Item.java   
Inventory.java   
FoodItem.java

TransactionReader.java   
data1.txt - small test case  
data2.txt - a medium test case  
data3.txt - a large test case

Note that there is also a DatedItem class that is an **inner class** of the Inventory class. The DatedItem is composed of a FoodItem, an expiration date, and a quantity.

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 (or to a file).  Either read from data{1,2,3}.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 transaction 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.

A food item arriving at a warehouse on a given date is inventoried and stored on the shelf the same day. Consequently it can be shipped to a retailer if it is requested that same day, otherwise it remains on the shelf until it reaches its expiration date **after which it can no longer be sold**. For example, a food item arriving on 5/10/2010 with a shelf life of one day will reach its expiration the following day (5/11/2010). It cannot be sold **after** 5/11/2010 because it is spoiled.

* **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 transactions 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 date. **Food Items cannot be sold the day after their expiration 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.  **Requested items are never back-ordered**. 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 transactions should be treated as occurring on the following day.  Your program should advance the effective date.  **In addition, your program should remove any inventory that is past its expiration day.**

* **End**:

Indicates the end of the file.  Your program should stop reading the transactions when it encounters this one. The report you are producing is for the current day.

For example, in the transaction file above, the transactions ended on 5/4/2010, so the report you generate will be as of 5/4/2010.

After your program processes all the data file transactions, your program should produce a statistical report that is **sorted by warehouse and by upc code**. All output should be to a file. Disable any debugging output you have prior to submitting your program - we need clean, easy-to-read reports.

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.

**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.  Also use collections to perform sorting. 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.)
* Use the keySet method on a Map to get the set of keys from the map.
* Create a separate class, FoodItemStatistics, that stores statistics for each unique food item such as total stock received, expired, etc.
* A Map<String, FoodItemStatistics> for mapping UPC codes to FoodItemStatistics objects to make it easy to find the food item stats that matches a UPC.  **Note that each warehouse will store its keep its own statistics for the food items in its inventory**.

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.) Or use an iterator.

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.

UPC Code: 0353264991 Shelf life: 2 Name: chestnut puree with vanilla

UPC Code: 0984523912 Shelf life: 1 Name: the orange box

Transactions and Expirations for Tacoma

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Chestnut Puree | | Orange Box | |
| 5/1 | **6** | **RECEIVED** | **7** | **RECEIVED** |
| 5/1 |  |  | **5** | **REQUESTED** |
| 5/2 |  |  | **2** | **EXPIRED** |
| 5/3 | **6** | **EXPIRED** |  |  |
| 5/4 | **10** | **REQUESTED** | **8** | **REQUESTED** |
|  |  |  |  |  |
| Stock Received | **6** |  | **7** |  |
| Stock Requested | **10** |  | **13** |  |
| Stock Shipped Out | **0** |  | **5** |  |
| Stock Expired | **6** |  | **2** |  |
| **Stock On Hand** | **0** |  | **0** |  |

Transactions and Expirations for Columbus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Chestnut Puree | | Orange Box | |
| 5/1 | **-** | **-** | **-** | **-** |
| 5/2 | **4** | **REQUESTED** | **4** | **REQUESTED** |
| 5/3 | **9** | **RECEIVED** | **2** | **RECEIVED** |
| 5/4 |  |  | **2** | **EXPIRED** |
| 5/4 |  |  | **8** | **RECEIVED** |
|  |  |  |  |  |
| Stock Received | **9** |  | **10** |  |
| Stock Requested | **4** | **REQUESTED** | **4** | **REQUESTED** |
| Stock Shipped Out | **0** |  | **0** |  |
| Stock Expired | **0** |  | **2** |  |
| **Stock On Hand** | **9** |  | **8** |  |

Transactions and Expirations for Scottsdale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Chestnut Puree | | Orange Box | |
| 5/1 | **2** | **RECEIVED** |  |  |
| 5/2 | **-** | **-** | **-** | **-** |
| 5/3 | **3** | **REQUESTED** |  |  |
| 5/4 | **-** |  | **8** | **REQUESTED** |
|  |  |  |  |  |
| Stock Received | **2** |  | **0** |  |
| Stock Requested | **3** |  | **8** |  |
| Stock Shipped Out | **2** |  | **0** |  |
| Stock Expired | **0** |  | **0** |  |
| **Stock On Hand** | **0** |  | **0** |  |

**Relevant Formulas**

Stock On Hand = Stock Received Stock – (Shipped Out + Stock Expired)

Requests Filled (%) = Total Stock Shipped Out / Total Stock Requested

Spoilage (%) = Total Stock Expired / Total Stock Received

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Report Dated 5/10/2010 through 5/14/2010**

**Statistics by Warehouse**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Columbus**

Chocolate Puree

Stock Received 9

Stock Requested 4 Requests Filled: 0%

Stock Shipped Out 0

Stock Expired 0 Spoilage: 0%

Stock On-Hand 9

Orange Box

Stock Received 10

Stock Requested 4 Requests Filled: 0%

Stock Shipped Out 0

Stock Expired 2 Spoilage: 20%

Stock On-Hand 8

**Scottsdale**

Chocolate Puree

Stock Received 2

Stock Requested 3 Requests Filled: 67%

Stock Shipped Out 2

Stock Expired 0 Spoilage: 0%

Stock On-Hand 0

Orange Box

Stock Received 0

Stock Requested 8 Requests Filled: 0%

Stock Shipped Out 0

Stock Expired 0 Spoilage: 0%

Stock On-Hand 0

**Tacoma**

Chocolate Puree

Stock Received 6

Stock Requested 10 Requests Filled: 0%

Stock Shipped Out 0

Stock Expired 6 Spoilage: 100%

Stock On-Hand 0

Orange Box

Stock Received 7

Stock Requested 13 Requests Filled: 38.5%

Stock Shipped Out 5

Stock Expired 2 Spoilage: 28.6%

Stock On-Hand 0