CSE214

HOMEWORK 6 - SPRING 2025

HOMEWORK 6 - due Friday, April 22nd no later than 6:00PM

REMINDERS:

- **Submit your files only via CodeGrade on the Content page in Brightspace. Access CodeGrade by clicking "HW6 - CODEGRADE SUBMISSION LINK" for your submission.**
- **Use of a package is optional. If you wish to use it, make sure to name it "hw6" (all in lowercase). Otherwise, you will lose points.**
- **If your program doesn't run on CodeGrade or if your score on CodeGrade is less than 50%, the maximum grade you can receive for this assignment is 85.**
- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **You may use any Java API class that you wish.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

In this assignment, you will be looking into the day to day processes that go on in a grocery store from the employee's perspective. You are tasked to write an inventory management system, where you will be using hash tables to store information about each item in the store's inventory.

The employees will use this software to track the inventory of items in their grocery store. They should be able to add items either by loading text files (which contain JSON information for each item to add to the inventory) or by manually inputting information. At the end of each business day, they will be able to process all sales information (stored in a file) and update the inventory levels of all items, generating orders for items that need replenishment.

ABOUT JSON

JSON (JavaScript Object Notation) is method of storing data in a human-readable format using key/value pairs. The data that is associated with each JSON object is enclosed in curly braces. For example, if we had an object that had `firstName` and `age` information associated with it, the general format of the JSON representation for that object would be:

```
{
    "firstName": "Bob",
    "age": "22"
}
```

It is also possible to denote arrays in JSON by enclosing a set of comma-separated JSON objects in square brackets. This is of the format:

```
[
    {
        "firstName": "Alice",
        "age" : "21"
    },
    {
        "firstName": "Thomas",
        "age": 19
    }
]
```

You will be using the JSON format to load items contained in files into your inventory management system. These items will have the fields:

```
{
    "itemCode": "",    // UPC/Item Code
    "itemName": "",    // Name of the item
    "avgSales": "",    // Average daily sales of the item
    "qtyInStore": "", // Quantity of item present in store
    "price": "",       // Price of item
    "amtOnOrder": ""   // Amount currently on order
}
```

The daily sales information is also stored in JSON formatted files. These items will have the fields:

```
{
    "itemCode": "",    // UPC/Item Code being sold
    "qtySold": ""      // Quantity sold during the current day
}
```

Since Java does not supply a JSON as part of the standard JDK, you will be using an external library called json-simple. You will need to add this library to your project.

- **Download the json-simple library here. (Do NOT submit this JAR file with your homework files. You will lose 7 points)**
- **For NetBeans:**
  - **Click on your project in the Projects column on the left, right click on the "Libraries" option, then click on the option "Add JAR/Folder...". This will then let you add the JAR files.**
- **For Eclipse:**

- Right click on the project in Project Explorer -> Properties -> Add External JARS -> find the JAR you wish to add and then add it, apply and close.

You can now use the json-simple library, such as:

```
import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

FileInputStream fis = new FileInputStream(filename);
InputStreamReader isr = new InputStreamReader(fis);
JSONParser parser = new JSONParser();
JSONArray objs = (JSONArray) parser.parse(isr); // objs is a JSONArray which contains all JSONObjects found in the InputStream
JSONObject obj = (JSONObject) objs.get(0);      // obj is the first JSONObject in the objs JSONArray
String val = (String) obj.get("name");          // This will extract the value for the 'name' field in obj
```

There is a tutorial for using json-simple available at Tutorials Point. For more information about the JSON format, refer to Wikipedia's JSON article.

ABOUT SERIALIZATION

Serialization is way of converting a state representation of an object into a byte stream. Deserialization is the reverse process, where a byte stream is used to recreate the original Java object in memory. Since we want to be able to save all of our inventory and sales records for the previous days when quitting the application, you will be implementing Serializable to enable serialization and deserialization of your classes. You should write your HashedGrocery item to a "grocery.obj" file when quitting the program, and attempting to load a HashedGrocery object upon starting the program if a "grocery.obj" file is found.

In order to support serialization, any class that you wish to serialize must implement Serializable. You do not have to write any custom methods while implementing this interface, as Java does provide a default implementation. Although it is not required, you may find it helpful to declare a `static final long serialVersionUID`. This variable is used to ensure that the current version of the class is compatible with the serialized version of the object. Certain changes to the class structure (such as removing, renaming, or adding member variables) may make it so that the readObject() method is unable to correctly process an old version. Note that you can omit this variable if you choose and Java will calculate a default value for it.

A generalized structure for a class which implements the Serializable interface would be as follows:

```
import java.io.Serializable;

public class MySerializableClass implements Serializable
{
    public static final long serialVersionUID = 1L; // Version 1
    // Class variables/methods/etc as usual
}
```

To serialize data:

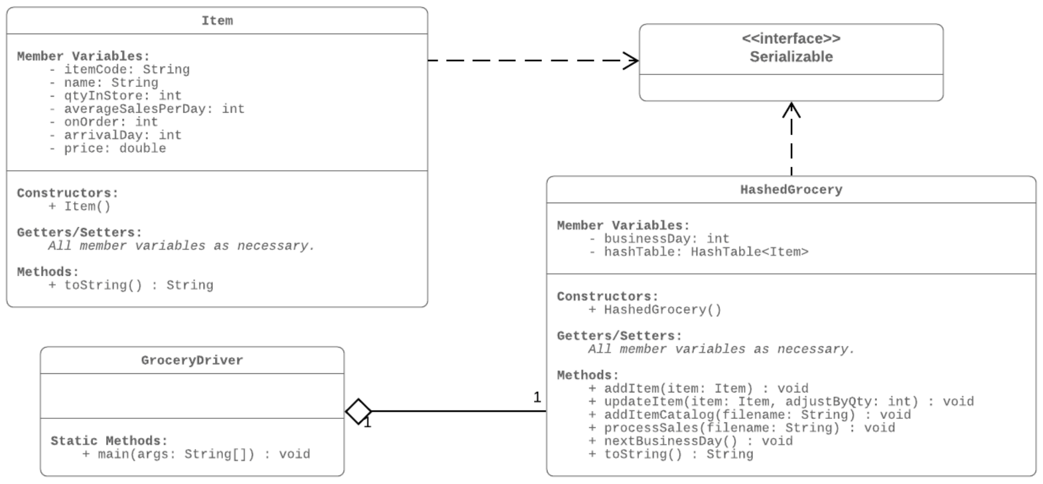
```
FileOutputStream file = new FileOutputStream("data.obj");
ObjectOutputStream fout = new ObjectOutputStream(file);

fout.writeObject(objToWrite); // Here "objToWrite" is the object to serialize
fout.close();
// Note that if data.obj does not exist, this will automatically create it and write into it.
```

To deserialize data:

```
try {
    FileInputStream file = new FileInputStream("data.obj");
    ObjectInputStream fin  = new ObjectInputStream(file);
    obj = (MyObjectType) fin.readObject();
    fin.close();
} catch(IOException e) {
    // This exception is thrown if data.obj does not exist.
    // Since there is nothing to load, here you should just create a new object, instead.
}
```

UML

The UML Diagram for all the classes as specified below is as follows:



SPECIFICATIONS

1. Item

Write a fully-documented class `Item` which will contain all the information regarding an item with the following data fields:

- **`String itemCode` - This must be unique, as it will serve as the key in your hash table.**
- **`String name`**
- **`int qtyInStore`**
- **`int averageSalesPerDay`**
- **`int onOrder` - How many have already been ordered for restocking.**
- **`int arrivalDay` - This variable will keep track of the business day at which your order for this item will arrive. Every order takes 2 whole days to ship and will arrive on the beginning of the third day, so you should assign a value to this variable depending on that and the business day you are currently on. For example, if you are on business day 1, then you should set this variable to 4. If there is nothing currently on order, then the value of this variable should be 0. See sample I/O for examples.**
- **`double price`**

Additionally, include the following methods:

- **`public Item()` - Constructor. You may include additional constructors that take parameters as necessary.**
- **Getters and setters for all data fields, as necessary.**
- **`public String toString()` - Returns a String representation of this Item.**

2. HashedGrocery

Write a fully-documented class `HashedGrocery` which will contain all the methods that will allow eployees to perform different functions on the inventory of the grocery store. Here, you will use the item code of every item as the key of a hash table. You can either use the HashTable or HashMap data structure to store your inventory. You should store the following information:

- **`int businessDay` - The current business day. Starts from 1.**
- **A hash table to store all item records.**
- **You can add any other data fields you find necessary.**

Also implement the following methods:

- **`public HashedGrocery()` - Constructor. You may include additional constructors that take parameters as necessary.**
- **`public void addItem(Item item)` - Adds `item` to the hash table. An exception should be thrown if an item with the same item code already exists in the hash table.**
- **`public void updateItem(Item item, int adjustByQty)` - Changes the `qtyInStore` amount of `item` by `adjustByQty`.**
- **`public void addItemCatalog(String filename)` - Adds all the items present in the text file. The supplied file should contain JSON information for one or more `Item` objects. Any duplicate items within the file should be ignored. An exception thrown if `filename` does not exist.**
- **`public void processSales(String filename)`**
  - **Processes `filename` to see which items have been sold that day. The supplied file should contain JSON information for one or more sales (see the discussion in the "About JSON" section).**
  - **While processing this information, if the item exists in your inventory, you should update the `qtyInStore` amount by subtracting the sold amount from it. You will then have to check if there is enough remaining in the store for the next three days. Since it takes 2 whole days to receive an order, if you don't have enough on hand for the next three days, then you need to place an order. To calculate the amount to order, you have to use the information given to you in the average sales per day variable of the item.**
  - **To calculate the amount to order for restocking: (number of days you want to order for [which is 2]) x (Average sales per day)**
  - **To calculate the minimum quantity to be in stock: (number of days you want stock for [which is 3]) x (Average sales per day)**
  - **If the quantity in store is less than the minimum that is required to be in stock, then you should order more.**
  - **If an order has already been placed for an item, then you don't need to order more.**
  - **If an order was placed, print a message saying how much of the item was ordered.**
  - **When processing a sale, if the quantity sold is greater than the quantity in the store, print a message and do not process the sale. See sample I/O for format.**

    - This method should throw an exception if the file does not exist.
- `public void nextBusinessDay()` - Increases `businessDay` by 1, and checks to see if any orders have arrived. If so, it updates the quantities of items in the store for those orders which have arrived, then sets the `onOrder` and the `arrivalDay` variables for those items to 0.
- `public String toString()` - Prints all items in a neatly formatted table. See sample I/O for format.

3. GroceryDriver

Write a fully-documented class `GroceryDriver`, which will act as the main driver for the inventory management system. When the program first starts, it should attempt to deserialize any previously serialized data. When the program terminates, it should attempt to serialize the hash table containing all item information. The class should supply a `public static void main(String[] args)` method which supports the following menu options:

- L (Load item catalog) - Load items from a text file containing JSON information into the grocery store
- A (Add item) - Allow the user to add individual items by hand
- B (Process Sales) - Adjust inventory and place orders based upon the sales information present in the supplied file
- C (Display all items) - User can display all items and their associated data fields
- N (Next Business Day) - Move to next business day
- Q (Quit) - Quit the application, saving all item information to "grocery.obj"

4. You will need classes to handle the exceptions thrown (see class specifications above for exception classes you need).

NOTE 1: You may include additional methods, variables, or classes as necessary or as you find convenient for the implementation.

NOTE 2: The lists generated by the C command must be sorted according to the item code to ensure consistency and match the expected output in CodeGrade. You can easily store the output in an array (or an ArrayList) and then use any of Java's sorting methods, such as Arrays.sort() or Collections.sort(), to sort the list.

INPUT FORMAT

- Each menu operation is entered on its own line and should be case insensitive (i.e. 'q' and 'Q' are the same).
- If, while trying to add items to the hash table, an entry already exists for a particular Item Code, do not update the entry.
- For the Load/Add commands, if the input information is valid, construct the object accordingly. Otherwise, print an error message and return to the main menu.

OUTPUT FORMAT

- Each command should output the result (as shown in the sample IO below) after each operation is performed.
- All menu operations must be accompanied by a message indicating what operation was performed and whether or not it was successful.

SAMPLE INPUT/OUTPUT

```
// Comments in green, input in red,, output in black.
// NOTE: The files used in this sample are available here(Do NOT submit these data files with your homework files. You will lose points) .

Grocery.obj does not exist. Creating new HashedGrocery object...

Business Day 1.

Menu :

(L) Load item catalog
(A) Add items
(B) Process Sales
(C) Display all items
(N) Move to next business day
(Q) Quit

Enter option: L

Enter file to load: itemCatalog_1.txt

690123401: BIC Black Pens added to inventory
661290409: Cheerios added to inventory
215672341: 5 Gum added to inventory
521378123: Kleenex Tissues added to inventory
432085123: Milk added to inventory
908125001: Coffee added to inventory

// Menu not displayed in sample i/o

Enter option: A

Enter item code: 387698724
Enter item name: Cheetos
Enter Quantity in store: 40
Enter Average sales per day: 10
Enter price: 2.99

387698724: Cheetos added to inventory

// Menu not displayed in sample i/o
```

```
Enter option: C

Item code   Name                Qty   AvgSales   Price    OnOrder    ArrOnBusDay
--------------------------------------------------------------------------------
215672341   5 Gum                80         30    2.01          0              0
387698724   Cheetos              40         10    2.99          0              0
432085123   Milk                 30         19    3.99          0              0
521378123   Kleenex Tissues      50         20    2.12          0              0
661290409   Cheerios             60         10    3.99          0              0
690123401   BIC Black Pens       40         10   10.98          0              0
908125001   Coffee               40         30   10.99          0              0


// Menu not displayed in sample i/o

Enter option : B

Enter filename : QtySales_1.txt

690123401: 9 units of BIC Black Pens are sold.
432085123: 17 units of Milk are sold. Order has been placed for 38 units
521891235: Cannot buy as it is not in the grocery store
215672341: 30 units of 5 Gum are sold. Order has been placed for 60 units

// Menu not displayed in sample i/o

Enter option : C

Item code   Name                Qty   AvgSales   Price    OnOrder    ArrOnBusDay
--------------------------------------------------------------------------------
215672341   5 Gum                50         30    2.01         60              4
387698724   Cheetos              40         10    2.99          0              0
432085123   Milk                 13         19    3.99         38              4
521378123   Kleenex Tissues      50         20    2.12          0              0
661290409   Cheerios             60         10    3.99          0              0
690123401   BIC Black Pens       31         10   10.98          0              0
908125001   Coffee               40         30   10.99          0              0

// Menu not displayed in sample i/o

Enter option: N

Advancing business day...
Business Day 2.

No orders have arrived.

// Menu not displayed in sample i/o

Enter option: L

Enter file to load: itemCatalog_3.txt

145910231: Frito Lays added to inventory
680124585: Gain Flings added to inventory
146012891: Lucky Charms added to inventory
765324890: Twix Chocolate added to inventory
324345764: Voss Artesian Water added to inventory
661290409: Cannot add item as item code already exists

// Menu not displayed in sample i/o

Enter option: C

Item code   Name                Qty   AvgSales   Price    OnOrder    ArrOnBusDay
---------------------------------------------------------------------------------
145910231   Frito Lays           20          7    3.99          0               0
146012891   Lucky Charms         30         12    3.99          0               0
215672341   5 Gum                50         30    2.01         60               4
324345764   Voss Artesian Water  40         20    4.99          0               0
387698724   Cheetos              40         10    2.99          0               0
432085123   Milk                 13         19    3.99         38               4
521378123   Kleenex Tissues      50         20    2.12          0               0
661290409   Cheerios             60         10    3.99          0               0
680124585   Gain Flings          20          9   12.99          0               0
690123401   BIC Black Pens       31         10   10.98          0               0
765324890   Twix Chocolate       30         10    7.99          0               0
908125001   Coffee               40         30   10.99          0               0

// Menu not displayed in sample i/o

Enter option: B

Enter filename: QtySales_2.txt

765324890: 9 units of Twix Chocolate are sold. Order has been placed for 20 units
908125001: 30 units of Coffee are sold. Order has been placed for 60 units
432085123: Not enough stock for sale. Not updated.
146012891: 10 units of Lucky Charms are sold. Order has been placed for 24 units
387698724: 10 units of Cheetos are sold.

// Menu not displayed in sample i/o

Enter option: C

Item code   Name                Qty   AvgSales   Price    OnOrder    ArrOnBusDay
```

```
--------------------------------------------------------------------------------
145910231   Frito Lays          20         7    3.99         0              0
146012891   Lucky Charms        20        12    3.99        24              5
215672341   5 Gum               50        30    2.01        60              4
324345764   Voss Artesian Water  40       20    4.99         0              0
387698724   Cheetos             30        10    2.99         0              0
432085123   Milk                13        19    3.99        38              4
521378123   Kleenex Tissues     50        20    2.12         0              0
661290409   Cheerios            60        10    3.99         0              0
680124585   Gain Flings         20         9   12.99         0              0
690123401   BIC Black Pens      31        10   10.98         0              0
765324890   Twix Chocolate      21        10    7.99        20              5
908125001   Coffee              10        30   10.99        60              5

// Menu not displayed in sample i/o

Enter option: N

Advancing business day...
Business Day 3.

No orders have arrived.

// Menu not displayed in sample i/o

Enter option: A

Enter item code: 680124585
Enter item name: Strawberries
Enter Quantity in store: 23
Enter Average sales per day : 9
Enter price : 4.99

680124585: Cannot add item as item code already exists

// Menu not displayed in sample i/o

Enter option: Q

HashedGrocery is saved in grocery.obj

Program terminating normally...

// Program is started again.

HashedGrocery is loaded from grocery.obj

Business Day 3.

// Note that since the program found and loaded a grocery.obj file,
// The inventory system already contains entries and knows that we are on
// business day 3.

// Menu not displayed in sample i/o

Enter option: N

Advancing business day...
Business Day 4.

Orders have arrived for:

215672341: 60 units of 5 Gum
432085123: 38 units of Milk

// Menu not displayed in sample i/o

Enter option: C

Item code   Name                Qty   AvgSales   Price    OnOrder    ArrOnBusDay
--------------------------------------------------------------------------------
145910231   Frito Lays          20         7    3.99         0              0
146012891   Lucky Charms        20        12    3.99        24              5
215672341   5 Gum              110        30    2.01         0              0
324345764   Voss Artesian Water  40       20    4.99         0              0
387698724   Cheetos             30        10    2.99         0              0
432085123   Milk                51        19    3.99         0              0
521378123   Kleenex Tissues     50        20    2.12         0              0
661290409   Cheerios            60        10    3.99         0              0
680124585   Gain Flings         20         9   12.99         0              0
690123401   BIC Black Pens      31        10   10.98         0              0
765324890   Twix Chocolate      21        10    7.99        20              5
908125001   Coffee              10        30   10.99        60              5

// Menu not displayed in sample i/o

Enter option: Q

HashedGrocery is saved in grocery.obj

Program terminating normally...
```