ASSIGNMENT REPORT

Information for use

Introduction:

The program, droneDirect, is designed to store data regarding different stores and their inventories. It then stores order data as well and displays queries related to that particular order.

Installation:

You need to unzip the folder and then compile all the files. Use "javac *.java" to compile every file in the folder quickly. Please do not delete and move files from the folder as some files are dependent on other files in order program to run successfully.

Dependencies:

Class	Dependent on
droneDirect	userInterface, report
userInterface	DSAHashTable, DSAGraph, Order, fileManager
report	DSAHashTable, DSAGraph, Order, fileManager
UnitTestDSAGraph	DSAGraph
UnitTestDSAHashTable	DSAHashTable, Order
UnitTestOrder	Order
UnitTestfileManager	fileManager, DSAHashTable, DSAGraph, Order

Description for each file:

- **droneDirect:** used to start the program and go into interactive or report mode.
- **userInterface:** helps the user to choose from 8 different options and produce results.
- **report:** creates a report according to command line arguments.
- **DSAGraph:** helps to store stores and displays them.
- **UnitTestDSAGraph:** helps in testing of DSAGraph functions.
- **DSAHashTable**: helps to store inventories and displays them.
- **UnitTestDSAHashTable:** helps in testing of DSAHashTable functions.
- **Order:** helps to store order.
- **UnitTestOrder:** helps in testing of Order functions.
- **fileManager**: helps to read and save adt (Normal & Serial).
- **UnitTestfileManager:** helps in testing of fileManager functions.

Terminology/Abbreviations:

- user = userInterface
- invt = Inventory
- rep = report
- prod = product
- qty = quantity

Walkthrough:

For Interactive mode, there are a total of 8 options.

1. The first option has a nested menu, it basically helps in loading files (location & order data). It also helps in the loading of serialized data.
2. The second option displays all the inventory of one particular store that the user input according to the order.
3. The third option displays the distance between two specific stores that the user inputs.
4. The fourth option displays the route that will be taken by the drone to deliver the order.
5. The fifth option lists information of all locations.
6. The sixth option lists information of all location inventories.
7. The seventh option helps to save the serialize location data.
8. And finally, the last option helps the user to exit the program.

For Report mode, the user has the enter a total of 3 files in order: output_file, inventory_file, order_file. If all the files are valid, a properly formatted output will be produced in the output file.

Functionalities:

I believe everything is working properly in the program and displays a specific error if anything is wrong. However, I have used one built-in adt of LinkedList for DSAGraph. I believe I could have made my own LinkedList but due to time constraints, I did not have a choice to not use it.

Future Work:

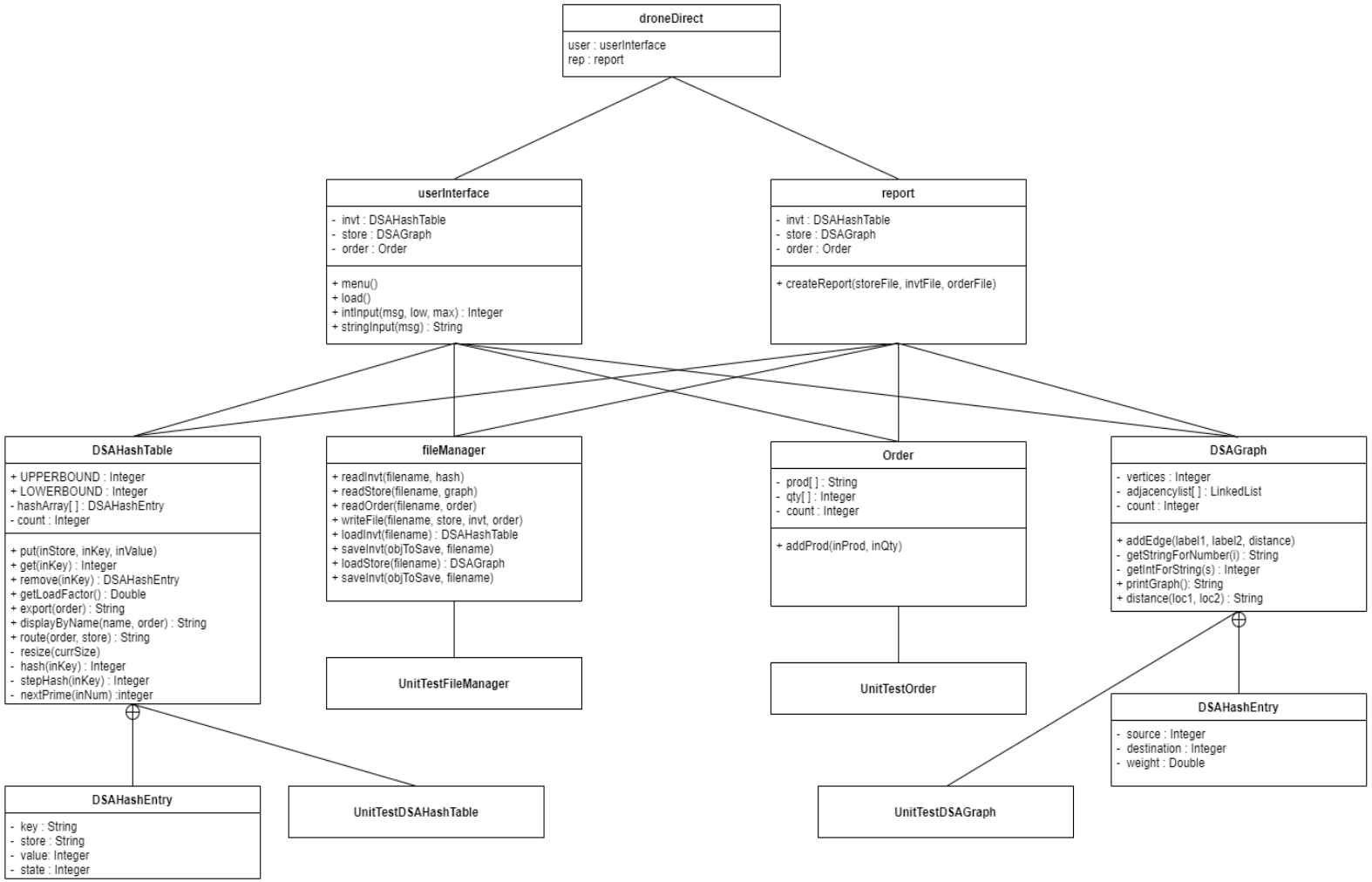
I would convert the LinkedList back to a self-written abstract data type if I have more time to spend on this assignment. I can also add more functionality by using other files such as adding product information. (I would have tried to do it in this one if the files made more sense and I had a little more time to do it)

Traceability Matrix

SNo.		REQUIREMENTS	Design/Code	Test
1.1	Drive/Menu & Modes	Program displays interactive mode with “-i” argument.	droneDirect -i	Run to check
1.2		Program displays report mode with “-r” argument.	droneDirect -r (filenames)	Run to check
1.3		In interactive mode, user enter command and program reponds	userInterface.menu()	Run to check
2.1	FileIO	Inventory is read and user is prompted for filename	fileManager.readInvt()	UnitTestfileManager
2.2		Store are read and user is prompted for filename	fileManager.readStore()	UnitTestfileManager
2.3		Order is read and user is prompted for filename	fileManager.readOrder()	UnitTestfileManager
2.4		File is written	fileManager.writeFile()	UnitTestfileManager
2.5		Serialized inventory is loaded	fileManager.loadInvt()	UnitTestfileManager
2.6		Serialized store is loaded	fileManager.loadStore()	UnitTestfileManager
2.7		Serialized inventory is saved	fileManager.saveInvt()	UnitTestfileManager
2.8		Serialized store is saved	fileManager.saveStore()	UnitTestfileManager
3.1	ADT insertion	Inventory is stored in hash table	hash.put()	UnitTestDSAHashTable
3.2		Stores are stored in graph	graph.addEdge()	UnitTestDSAGraph
3.3		Order is stored in order class	order.addOrder()	UnitTestOrder

SNo.		REQUIREMENTS	Design/Code	Test
4.1	Displays	Inventory is displayed	hash.export()	UnitTestDSAHashTable
4.2		Graph is displayed	graph.printGraph()	UnitTestDSAGraph
4.3		Inventory of only one store is displayed	hash.displayByName()	UnitTestDSAHashTable
4.4		Distances between two stores are displayed	graph.distance()	UnitTestDSAGraph
4.5		Route is displayed for order delivery	hash.route()	UnitTestDSAHashTable
5.1	Miscellaneous	Number of inventory items	hash.getCount()	UnitTestDSAHashTable
5.2		Number of stores	graph.getCount()	UnitTestDSAGraph
5.3		Number of products in order	order.getCount()	UnitTestOrder
5.4		Removing a product from inventory	hash.remove()	UnitTestDSAHashTable

UML DIAGRAM


droneDirect
user : userInterface
rep : report
userInterface
- invt : DSAHashTable
- store : DSAGraph
- order : Order
+ menu()
+ load()
+ intInput(msg, low, max) : Integer
+ stringInput(msg) : String
report
- invt : DSAHashTable
- store : DSAGraph
- order : Order
+ createReport(storeFile, invtFile, orderFile)
DSAHashTable
+ UPPERBOUND : Integer
+ LOWERBOUND : Integer
- hashArray[ ] : DSAHashEntry
- count : Integer
+ put(inStore, inKey, inValue)
+ get(inKey) : Integer
+ remove(inKey) : DSAHashEntry
+ getLoadFactor() : Double
+ export(order) : String
+ displayByName(name, order) : String
+ route(order, store) : String
- resize(currSize)
- hash(inKey) : Integer
- stepHash(inKey) : Integer
- nextPrime(inNum) :integer
fileManager
+ readInvt(filename, hash)
+ readStore(filename, graph)
+ readOrder(filename, order)
+ writeFile(filename, store, invt, order)
+ loadInvt(filename) : DSAHashTable
+ saveInvt(objToSave, filename)
+ loadStore(filename) : DSAGraph
+ saveInvt(objToSave, filename)
Order
- prod[ ] : String
- qty[ ] : Integer
- count : Integer
+ addProd(inProd, inQty)
DSAGraph
- vertices : Integer
- adjacencylist[ ] : LinkedList
- count : Integer
+ addEdge(label1, label2, distance)
- getStringForNumber(i) : String
- getIntForString(s) : Integer
+ printGraph(): String
+ distance(loc1, loc2) : String
UnitTestFileManager
UnitTestOrder
DSAHashEntry
- source : Integer
- destination : Integer
- weight : Double
DSAHashEntry
- key : String
- store : String
- value: Integer
- state : Integer
UnitTestDSAHashTable
UnitTestDSAGraph

Class Description

droneDirect.java: This class is the main class to start the program. The purpose of this class is to direct you to an interactive mode or report mode using the command line argument you entered. If an invalid command line argument is entered, it will show you how to use the program properly. This class is dependent on userInterface.java and report.java. I choose to create this class separately as the program will be more specific and a programmer can easily add functionality or edit something as both modes (interactive and report) are in separate classes.

userInterface.java: This class leads you to interactive mode. It has a total of 8 options to choose from. However, almost every option requires you to load complete data first (location & order) in order for other functions to work. If files are not loaded, a proper error message is displayed as to what is required to be loaded. The other options are pretty self-explanatory as to what you want to do with the program. This class is dependent on DSAGraph.java, DSAHashTable.java, order.java, and fileManager.java. I choose to create this class so that it can have an appropriate call to other classes which it is dependent on, in order for the program to work properly in interactive mode.

report.java: This class leads you to report mode. It takes in a total of 3 parameters which are entered at the start of the program as a command line argument. On the terminal, it will only display to you that a report has been successfully created with the file name. The user does not need to interact with this class at all except running the program at the start. This class is dependent on DSAGraph.java, DSAHashTable.java, order.java, and fileManager.java. This program deals with appropriate calls to other classes and creates a file as a result. I choose to create this class so that if there's anything that needs to be changed in report mode, it can be changed from here.

DSAGraph.java: This class helps to store the data of stores and display them as well. The user would not have direct contact with this class as it is much of a behind the scenes class for storing store.csv in appropriate adt (graph) and display them properly. It also has a private inner class that helps this adt to run properly which is known as “DSAGraphWeight”. I choose to create this class so that it can store and display location data properly as I think this is the best adt to store it.

UnitTestDSAGraph.java: This class helps in testing the functions of DSAGraph.java.

DSAHashTable.java: This class helps to store the data of inventories and display them as well. The user would not have direct contact with this class as it is much of a behind the scenes class for storing inventories.csv in appropriate adt (hash table) and display them properly. It also has a private inner class that helps this adt to run properly which is known as “DSAHashEntry.java”. This class kind of depends on order.java as it displays data with respect to the order file. I choose to create this class so that it can store and display inventory data properly as I think this is the best adt to store it.

UnitTestDSAHasTable.java: This class helps in testing the functions of DSAHashTable.java.

order.java: This class helps to store the data of order. The user would not have direct contact with this class as it is much of a behind the scenes class for storing order.csv (any version) in appropriate adt (array). This class does not really display anything by itself but helps to filter out data for other classes. I choose to create this class so that it can order data properly as I think this is the best adt to store it.

UnitTestOrder.java: This class helps in testing the functions of DSAOrder.java.

fileManager.java: This class helps to read different types of files, load and save serialized data and write to a CSV file when report mode is selected. The user will not have direct contact with this class. The names of the files are required to be entered from the user. I choose to create this class so that all the reading, writing, loading, and displaying (serial data) can be done with calls to this class.

UnitTestfileManager.java: This class helps in testing the functions of fileManager.java.

Justifications

Choosing Abstract Data Types (adt) for different data:

- **inventories.csv:** I decided to store the inventories in Hash Table as we can use the products as a key and other data can also be store. The main reason to store inventories in the hash table was that products will keep getting out of stock and re-stocked as well and because of its efficiency. The hash table will keep increasing and decreasing in size according to the situation. However, in this assignment, only one order is being dealt with every time the program is executed so this would not happen unless a really big order is received but still, we do not need to predict the amount of storage required for inventories. Hash Tables are faster and more efficient even with a big set of data when compared to other adts. So then when we have to display the data, the output will be generated quite quickly. Therefore, I think this is the best adt of all to store the inventories.csv file.
- **stores.csv:** I decided to store the location or stores in Graph as we can connect the locations using vertices and edges properties. The main reason to choose this adt over others is that the distance can be stored as weight so this will be a perfect match to store this data. We can also calculate the routes required to collect the order. This display of graphs is more appealing when we compare them with other adts which is exactly what we need for this type of scenario when displaying distances between two stores.
- **order.csv:** I decided to store orders in normal arrays. However, I made a class first so that accessing and creation of that array will be easier and properly implemented. I wanted to store this data in something like arrays of struct (like in C language) but unfortunately, Java does not support that so I instead decided to accomplish it in this way. I also thought of storing this data in some other adt but it did not really make sense to me and I thought it would be useless to store in them so I picked this idea instead. (Idea for this type of implementation was taken from previous work completed by myself for another unit)

- **Container Classes:** userInterface.java acts as a container class for interactive mode and is used to call different functions for the user to use. report.java acts as a container class for report mode and is used to create an output file with processed information.

********** Assumptions are on next page. **********

Assumptions

- For the first option, I assume that each file name will be provided by the user at appropriate prompts so I created different options for them. I created a nested menu when the user selects “Load Data” Inventories and stores are stored by calling one option but the user is asked to enter the names of both files separately. Order has its own option. As expected, serialized data will not be present before it is saved so it has another separate option in the nested menu.
- The display for inventories (one single store or all) is based on the order that has been loaded to avoid presenting useless products which are not required.
- The display for stores (distance between two or all) does not depend on the order as it seems more appropriate to show all stores and their distances or the just two stores regardless of order.
- If a store has been visited, the drone will try to pick up the maximum required discreet number of products from which might be available at other stores to avoid useless travel to that store. The exception, in this case, is that the store has less quantity of that product than the customer requires so it then visited another store to fulfill the quantity needs of the order.
- The route begins and ends at “Home”.
- Report mode requires three command line arguments which are output file, inventory file, and order file. It displays an error if files inputted are not formatted properly or incorrect names are provided.
- Report mode displays “Location Overview” & “Inventory Overview”. I also added the route that will be used for delivering the order as it will different every time a different order is loaded.

References

- **droneDirect:** I created this class for this assignment.
- **userInterface:** I created this class for this assignment.
- **report:** I created this class for this assignment.
- **DSAGraph:** This class is taken from Practical 6 submitted by me, however some additional functionalities have been added for assignment needs.
- **UnitTestDSAGraph:** I created this class for this assignment.
- **DSAHashTable**: This class is taken from Practical 7 submitted by me, however some additional functionalities have been added for assignment needs**.**
- **UnitTestDSAHashTable:** This class is taken from Practical 7 submitted by me, however some additional functionalities have been added for assignment needs**.**
- **Order:** I created this class for this assignment.
- **UnitTestOrder:** I created this class for this assignment.
- **fileManager**: I created this class for this assignment.
- **UnitTestfileManager:** I created this class for this assignment.