CAB302 Assignment 2 - Report

Ashley Husband – n9721657

Josh Lowe – n9745734

1 Project Architecture Description

This section of the report describes each class of the application at a high level, outlining interactions with other classes and detailing the methods implemented.

1.1 Item

The Item class primarily acts as an object for use in other classes. An Item object is simply a collection of data, consisting mainly of primitive type, except for the required temperature (temp). The required temperature is an Integer object because for items that don't need to be refrigerated, the temperature will be null. As seen in the UML Class diagram the Items are first initialised as "itemList", an array of items for the Store class, for use in other classes. This is so that there is a global consensus of item object instantiation, so comparisons between items in different Stock objects are possible. The Item object is the core object of the application, being at a very low level.

The Item object is constructed by taking these values as parameters and assigning them to each attribute. The extent of the item object however is simply just a lot methods to return these values, such as getCost, getPrice, getReorderPoint, etc.

1.2 Stock

The Stock class also primarily acts as an object, which holds a HashMap of <Item, Integer>. In the context of the application the Stock object is simply list of items and their quantities. It may seem that there is a one to many relationship between Item and Stock, however, there is not. Stock simply retrieves the itemList from the Store class, as seen in section 2, the UML diagram. Stock is used throughout the application, in trucks, store inventories, sales logs, etc. As further described in section 1.4 and section 1.3, there is zero to one relationship between Stock to Truck and Store, due to Truck and Store containing an inventory.

The class itself implements a variety of methods. For retrieving variables it has getQuantity, which returns the quantity of an item, getTotalPrice, which retrieves the total price of the stock, and getTotalCost, which retrieves the total cost of the stock. Other methods include addQuantity and removeQuantity, which is used for modifying the quantity of an item in the stock. Lastly, reorderRequired checks whether the quantity of an item is below the reorderPoint, to which it returns a boolean value illustrating whether the item needs to be reordered or not. All these methods are used throughout the application in almost every class.

1.3 Store

The Store class is very different from the other classes. As seen in the UML Class Diagram in section 2, it has a zero to one relationship as an instance of itself. This means that there can only be one instance of the Store object at one time. Therefore the Store class is of the Singleton Pattern. There is a practical use for this pattern, because in the context of the application, there can only be one store a time, and in this manner it acts as a global variable, making the application much more easy to develop.

The Store object itself holds a name, capital, and inventory. Also to note, upon initialisation, as described in section 1.1, it initialises a list of items, "itemList", for use throughout the application. The methods implemented in this class include getInstance, getCapital, modifyCapital, getItemList, and a static method, loadSalesLog. The getInstance class can be thought of as the constructor, in rough terms. As this class is part of the singleton pattern, when this is called for the first time, the Store object is initialised, however, when called afterwards, that instance is simply returned. The rest of the get methods simply return the attributes of the object. The modifyCapital method takes an amount and either adds or subtracts it from the capital. Lastly, the loadSalesLog method loads a file and receives a Stock object from the ReadCSV class, then reduces the store inventory, but also increases the capital.

1.4 Truck

The Truck class primarily acts as an abstract class, extending ColdTruck and RegTruck. The Truck object itself is simply a Stock object, but with a defined capacity. It initialises the “cargo” variable as a Stock object and takes “cargoCapacity” integer as a parameter upon construction. Through this, the subclasses of ColdTruck and RegTruck can implement their different capacities through a simple initialisation. As seen in the UML Diagram in section 2, there is a zero to one relationship between the Stock class and the Truck class, while there is a zero to many relationship between the Truck class and the Manifest class, which will be detailed in section 1.7.

While it does implement some concrete methods, it applies abstraction to implement two empty methods of addCargo and getCost. The addCargo method takes an Item object and an integer for quantity as a parameter. Due to the different allowable items to be added in each different truck, this method is implemented in each subclass. Furthermore, the getCost method for finding the cost of each truck is also an abstract method due to the different calculations for finding the cost. The getCargo method sums all the quantities inside the truck’s “cargo” Stock object and returns them, this is to be used in calculations for the cost. Lastly, the getStock method simply returns the “cargo” Stock object for calculations in the manifest and .csv reading.

1.5 ColdTruck

The ColdTruck class contains implementations of the abstract methods of the Truck class, detailed previously in section 1.4. The ColdTruck begins by initialising the superclass Truck object with a capacity of *800*, as per the specification. However, the ColdTruck constructor takes a temperature (int) parameter, which is assigned to the private attribute “temperature”. Before this however, the temperature is checked whether it is in the valid range of *-20* to *10* degrees. If it fails this test, a DeliveryException is thrown. Implementations of the abstract method addCargo contains a simple check whether the item quantity exceeds the capacity of the truck, to which it throws another exception. Lastly, getCost is an instant return of a given formula for finding the cost of the truck dependant on it’s temperature.

1.6 RegTruck

The RegTruck class contains similar implementations of the Truck abstract methods to ColdTruck, however, it slightly differs. Firstly, it initialises the superclass with a capacity of *1000*, and does not take temperature as a parameter in the construction. Furthermore, in the addCargo implementation, while roughly the same as the one detailed in section 1.5, it adds another check to see whether there are any items being added that require refrigeration. If this check fails, a delivery exception is thrown. Lastly, the getCost method differs simply in the equation for calculating the cost, scaling with the quantity rather than the temperature.

1.7 Manifest

While the manifest class has some static methods, it primarily acts as an object representing a one to many dependency between Truck objects and itself. This is because a manifest is simply a list of trucks. Therefore, this class is part of the observer pattern, because when one truck is modified, the manifest will be updated automatically. The context for this is that a manifest is the output of an “order” which requests a required amount of items for the store. However, a Truck object has a limited capacity, so for large orders, there needs to be more than one Truck, a list.

The non-static methods for the Manifest class include the constructor, addTruck, getManifest, and getCost. The addTruck method is the method that allows parts of the application to add trucks to the list, The getManifest retrieves the manifest object, getCost calculates the cost of each truck and all their items.

Lastly static methods include automateManifest, loadManifest. The automateManifest method is a lengthy method that, briefly, retrieves the given inventory and calculates which items needs to be topped up, after that, it creates the required trucks and fills them with cargo efficiently according to their costs. Lastly it returns the Manifest object created. The loadManifest calls the ReadCSV class to receives a manifest object, to which it increases the store inventory but reduces the capital.

1.8 ReadCSV

The ReadCSV class contains three static methods, intialiseItems, createItem, and readManifest. The initialiseItems method reads an item_properties formatted file and intiialises a set of items from the specification. The createItem method is used as a helper method for intialiseItems, it takes the data as a parameter and parses it into an item and returns the Item object. The readManifest method reads a manifest formatted file and returns a Manifest object after parsing through the data.

1.9 WriteCSV

The WriteCSV class has a single static method, writeManifest. It simple receives a manifest object and writes a new .csv file according to the fileName paramater chosen. The file is formatted properly and illustrates every truck in the manifest and the items they contain.

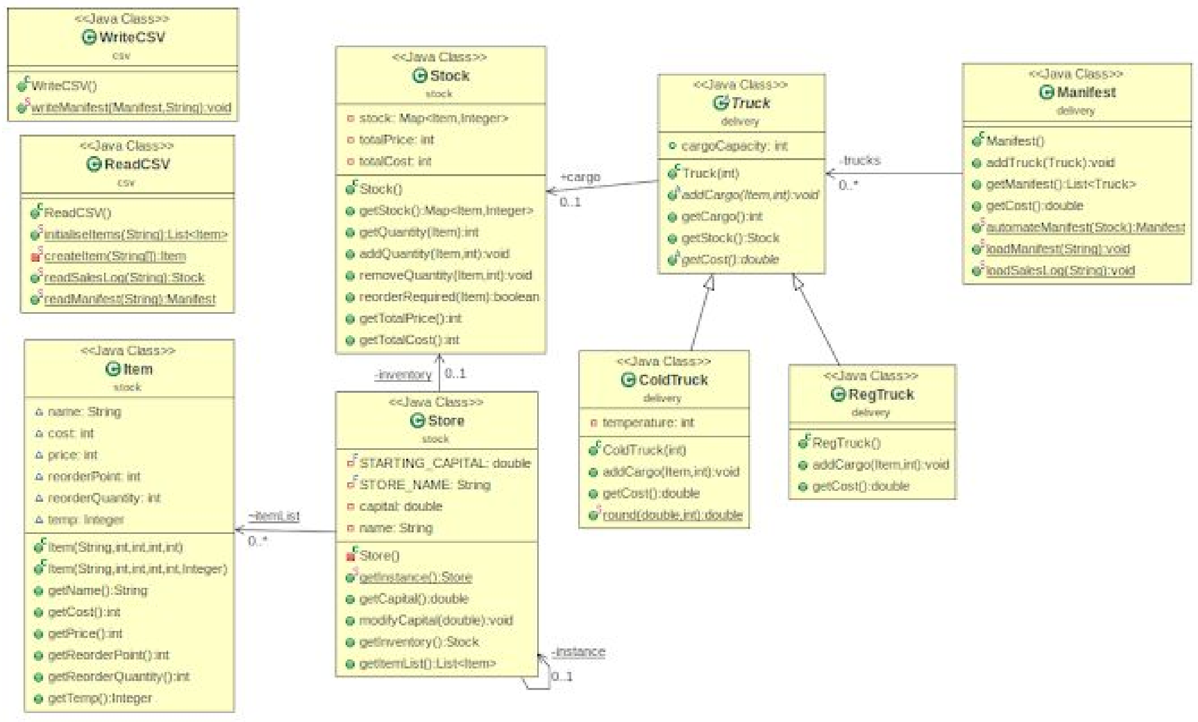
1.10 Exceptions

- CSVException: An exception that is thrown when the CSV Parser cannot parse a CSV file

- StockException: An exception that is thrown when an Item cannot be created or added to a Stock
- DeliveryException: An exception that is thrown when a Truck cannot be created or added to a Manifest

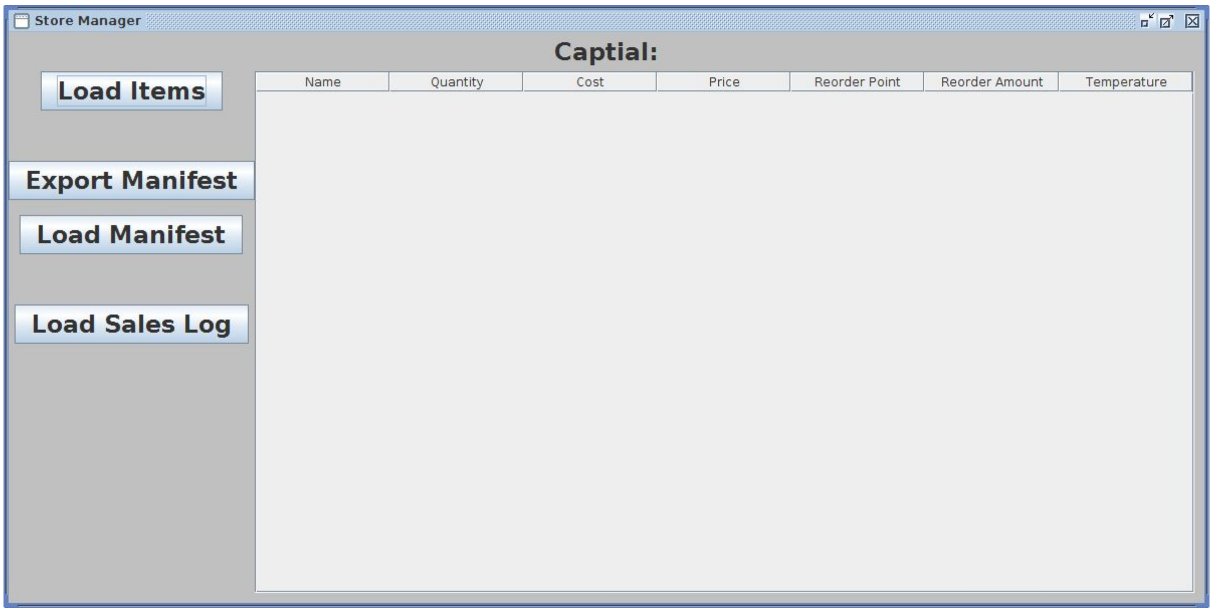
2 Class Diagram

Below is a UML Class Diagram of the application, detailing the relationships between each of the classes.



3 GUI Report

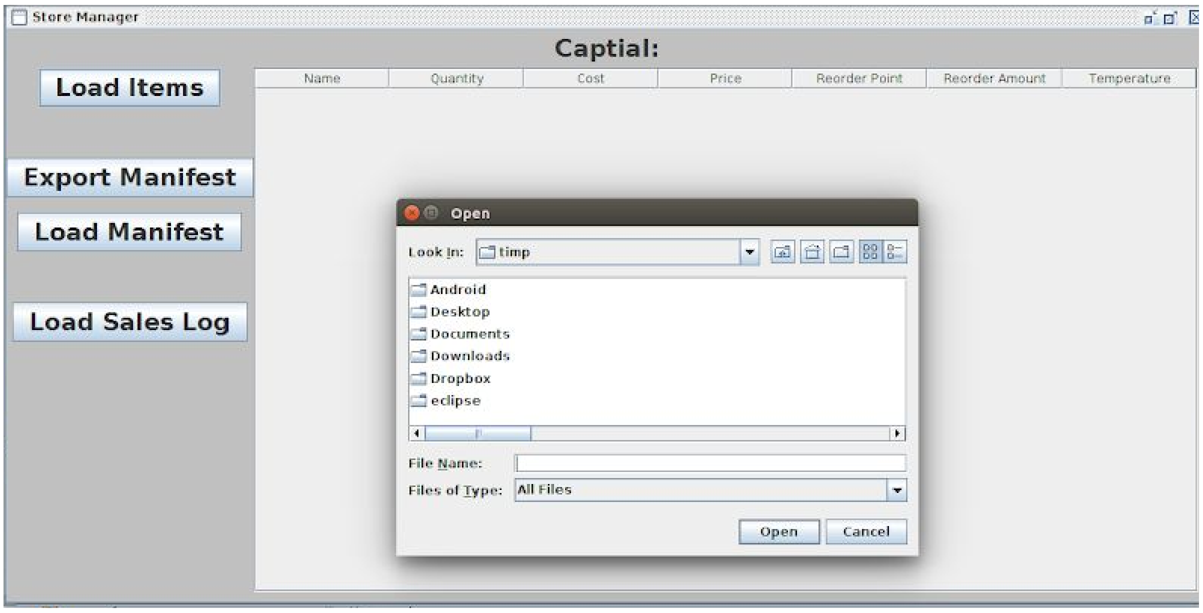
- This is a 2-5 page (or longer- make this as long as you need) report showing every item of functionality, using screenshots to assist
- Show both normal behaviour and abnormal behaviour- e.g.:
    - What happens if you load a file that isn't in the correct format?
    - What happens if the sales log is greater than your inventory?
- (Hint: on Windows use Alt+Printscrn to take screenshots of individual windows. If you need to get multiple windows in the one screenshot, take a fullscreen screenshot with Printscrn and crop it in a paint program like MsPaint)



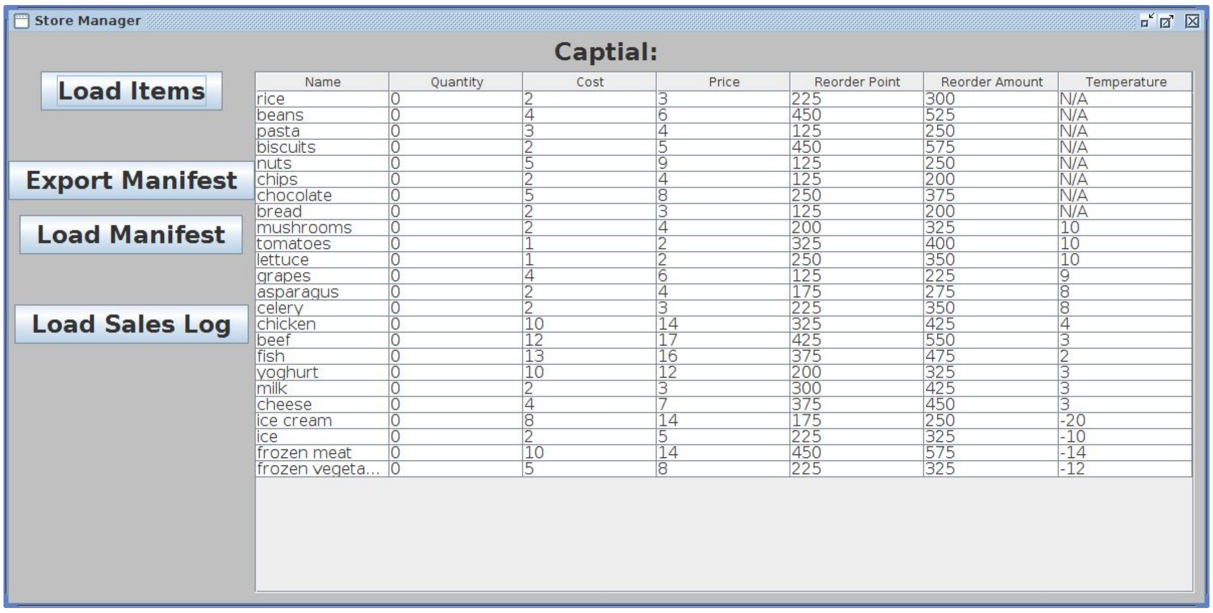
Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature

Upon running of the application the user is shown this window, which shows an empty table of data and buttons along the side.

Upon the pressing of the "Load Items" button, the user is presented with the following window.



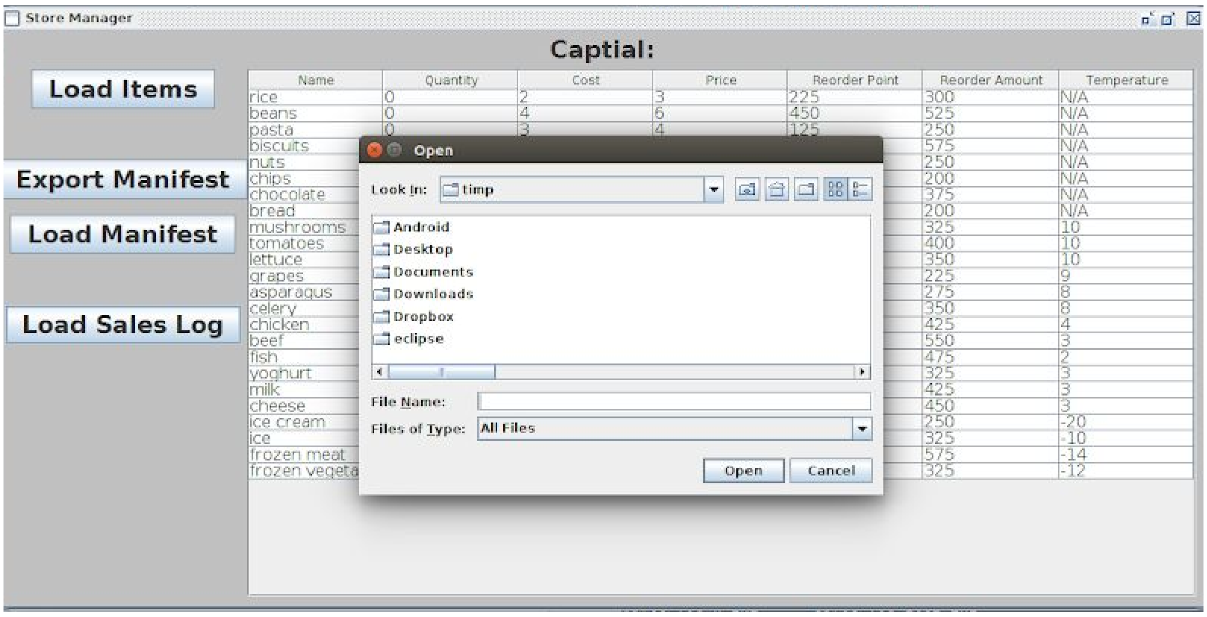
This allows the user to select an item properties file and load it to the GUI, shown below.



Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature
rice	0	2	3	225	300	N/A
beans	0	4	6	450	525	N/A
pasta	0	3	4	125	250	N/A
biscuits	0	2	5	450	575	N/A
nuts	0	5	9	125	250	N/A
chips	0	2	4	125	200	N/A
chocolate	0	5	8	250	375	N/A
bread	0	2	3	125	200	N/A
mushrooms	0	2	4	200	325	10
tomatoes	0	1	2	325	400	10
lettuce	0	1	2	250	350	10
grapes	0	4	6	125	225	9
asparagus	0	2	4	175	275	8
celery	0	2	3	225	350	8
chicken	0	10	14	325	425	4
beef	0	12	17	425	550	3
fish	0	13	16	375	475	2
yoghurt	0	10	12	200	325	3
milk	0	2	3	300	425	3
cheese	0	4	7	375	450	3
ice cream	0	8	14	175	250	-20
ice	0	2	5	225	325	-10
frozen meat	0	10	14	450	575	-14
frozen vegeta...	0	5	8	225	325	-12

This window shows that the table is filled with each item in the store, their properties and quantity.

Upon the pressing of “Load Manifest”, the user is also presented with the following window.


Store Manager

Captial:

Load Items

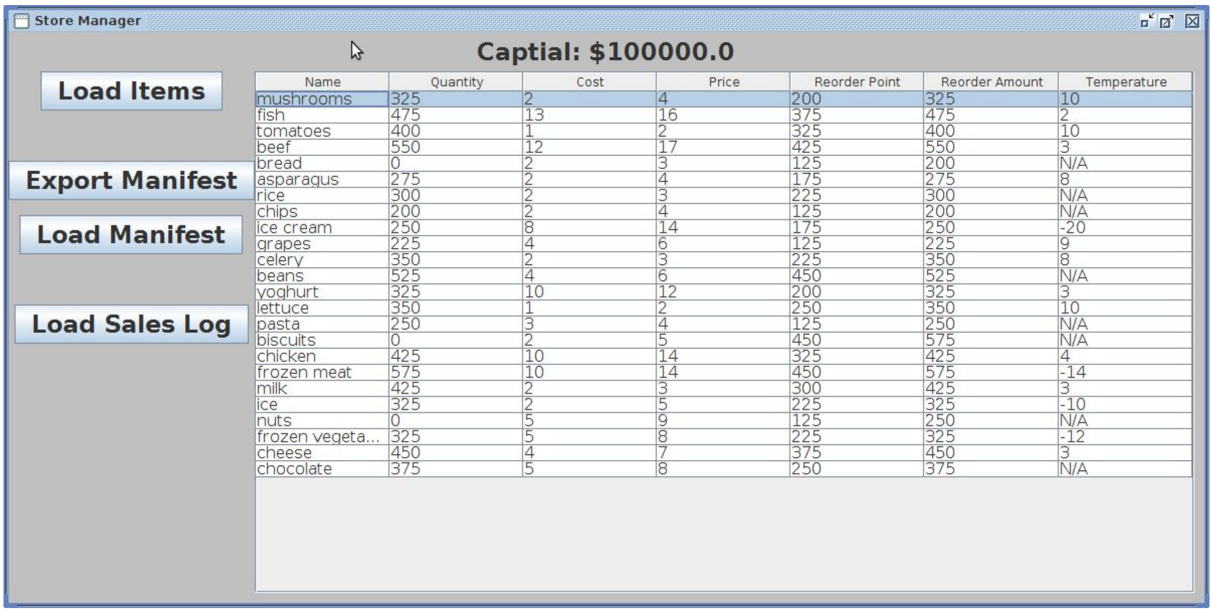
Export Manifest

Load Manifest

Load Sales Log

Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature
rice	0	2	3	225	300	N/A
beans	0	4	6	450	525	N/A
pasta	0	3	4	125	250	N/A
biscuits					575	N/A
nuts					250	N/A
chips					200	N/A
chocolate					375	N/A
bread					200	N/A
mushrooms					325	10
tomatoes					400	10
lettuce					350	10
grapes					225	9
asparagus					275	8
celery					350	8
chicken					425	4
beef					550	3
fish					475	2
yoghurt					325	3
milk					425	3
cheese					450	3
ice cream					250	-20
ice					325	-10
frozen meat					575	-14
frozen vegeta					325	-12

This window allows the user to select a file to load as a manifest.


Store Manager

Captial: $100000.0

Load Items

Export Manifest

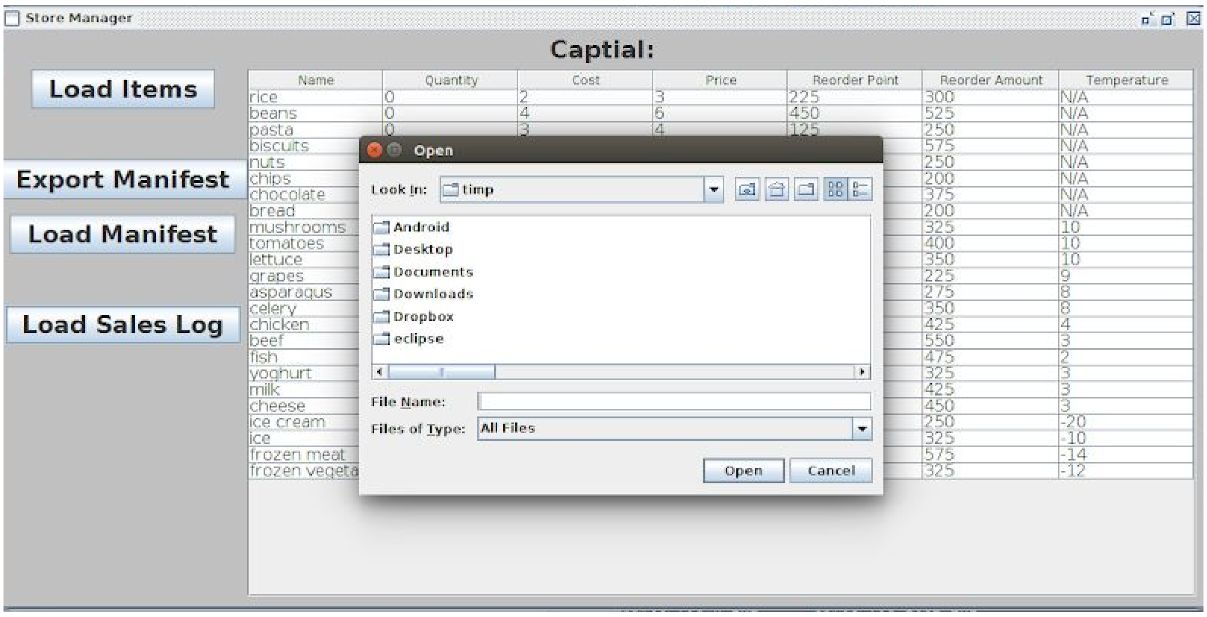
Load Manifest

Load Sales Log

Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature
mushrooms	325	2	4	200	325	10
fish	475	13	16	375	475	2
tomatoes	400	1	2	325	400	10
beef	550	12	17	425	550	3
bread	0	2	3	125	200	N/A
asparagus	275	2	4	175	275	8
rice	300	2	3	225	300	N/A
chips	200	2	4	125	200	N/A
ice cream	250	8	14	175	250	-20
grapes	225	4	6	125	225	9
celery	350	2	3	225	350	8
beans	525	4	6	450	525	N/A
yoghurt	325	10	12	200	325	3
lettuce	350	1	2	250	350	10
pasta	250	3	4	125	250	N/A
biscuits	0	2	5	450	575	N/A
chicken	425	10	14	325	425	4
frozen meat	575	10	14	450	575	-14
milk	425	2	3	300	425	3
ice	325	2	5	225	325	-10
nuts	0	5	9	125	250	N/A
frozen vegeta...	325	5	8	225	325	-12
cheese	450	4	7	375	450	3
chocolate	375	5	8	250	375	N/A

After loading the manifest, the item quantities are updated

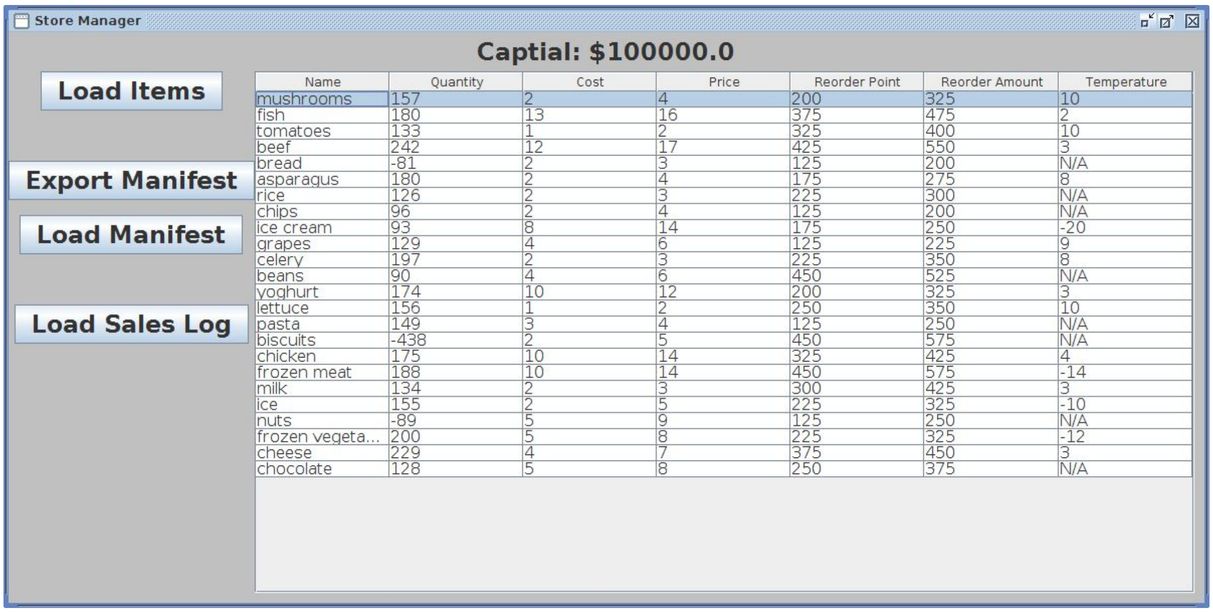
Upon the pressing of “Load Sales Log”, the user is presented with again with the following window.



Captial:

Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature
rice	0	2	3	225	300	N/A
beans	0	4	6	450	525	N/A
pasta	0	3	4	125	250	N/A
biscuits					575	N/A
nuts					250	N/A
chips					200	N/A
chocolate					375	N/A
bread					200	N/A
mushrooms					325	10
tomatoes					400	10
lettuce					350	10
grapes					225	9
asparagus					275	8
celery					350	8
chicken					425	4
beef					550	3
fish					475	2
yoghurt					325	3
milk					425	3
cheese					450	3
ice cream					250	-20
ice					325	-10
frozen meat					575	-14
frozen vegeta					325	-12

This window allows the user to select a file to load as a manifest.



Captial: $100000.0

Name	Quantity	Cost	Price	Reorder Point	Reorder Amount	Temperature
mushrooms	157	2	4	200	325	10
fish	180	13	16	375	475	2
tomatoes	133	1	2	325	400	10
beef	242	12	17	425	550	3
bread	-81	2	3	125	200	N/A
asparagus	180	2	4	175	275	8
rice	126	2	3	225	300	N/A
chips	96	2	4	125	200	N/A
ice cream	93	8	14	175	250	-20
grapes	129	4	6	125	225	9
celery	197	2	3	225	350	8
beans	90	4	6	450	525	N/A
yoghurt	174	10	12	200	325	3
lettuce	156	1	2	250	350	10
pasta	149	3	4	125	250	N/A
biscuits	-438	2	5	450	575	N/A
chicken	175	10	14	325	425	4
frozen meat	188	10	14	450	575	-14
milk	134	2	3	300	425	3
ice	155	2	5	225	325	-10
nuts	-89	5	9	125	250	N/A
frozen vegeta...	200	5	8	225	325	-12
cheese	229	4	7	375	450	3
chocolate	128	5	8	250	375	N/A