

ASSIGNMENT 2: INVENTORY MANAGEMENT APPLICATION

Group 102

Thomas Nash – n09761829

John Huynh – n09154566

CAB302: Software Development

Due 27/05/2018

Program Architecture

Overview

This project consists of six different packages (CSV, Delivery, Stock, Exception, Entry Point and GUI) in which all classes and testcases are held.

Throughout the design of this stock management system, object-oriented programming was always at the core of the development processes. This was the case largely due to a stock management system being essentially an *object* management system when looked at programmatically.

In section A1 of the assignment specification, several objects types are detailed which have all been implemented successfully.

A further look at these types and classes follows, with details regarding the interaction between them. A class diagram has also been created.

Classes and Types

Item.java

In a store using our stock management system, for every item located in the store, an equivalent Item object exists within the program. This item holds item variables from the item's name (itemName) to its temperature (temp).

An Item object, when constructed consists of itemName, manufactureCost, sellPoint, reorderPoint, reorderAmount and the item's temperature.

The Item class located in Item.java supplies getter methods for these variables.

Stock.java

Whenever a store using our stock management system views a list of items, it is viewing a Stock object.

A Stock object in our implementation consists of a `HashMap<Item,Integer>` which stores an Item object and its respective quantities.

The Stock type, created by the Stock class in Stock.java provides manipulation of these stock objects and their items through methods such as `addItem(Item item, int Quantity)`, `addQuantity(String itemName, int quantity)` and `remove(String itemName, int quantity)`.

It also supplies getter methods to receive a tableformat, number of items in the stock, the stock's HashMap, individual Item objects, and individual item's quantities.

StockException.java

When anything goes wrong with a *Stock* object, it must be handled by the program in some manner, at the very least by throwing a *StockException*.

The *StockException.java* class provides a throwable exception which returns a descriptive string to the user should an exception occur.

Store.java

Any store running the program is represented as an object from a programming standpoint. Upon the execution of the system, a *Store()* object is created to contain and monitor the variables and objects a physical store would hold (e.g. Capital, Inventory, Sales Logs, Manifests).

This implementation creates a *Store* object instance by using the Singleton Pattern to ensure only one instance of a *Store* object exists within the program space.

To hook the instance of the *Store* the method *getInstance()* is used to return the *Store* object.

Store.java has inventory manipulation methods called *setInventory(Stock)*, *addInventory(Stock)*, *removeInventory(HashMap<String,Integer>)*.

Other setter and getter methods include; *addCapital(Int)*, *resetCapital()*, *subtractCapital(Int)*, *setName(String)* and *getName()*.

CSVMachine.java

For this project, the storing of data has been done using CSV (Comma Separated Variables) files. This requires tailored read and write functionality to interact with the files effectively. To do this, *CSVMachine.java* has been created to supply the required CSV functionality.

To read the Item Properties, the method *readItemProperties(String filepath)* was created, which locates the CSV at the given filepath and returns a *Stock* object containing the items and their properties.

To read the Order Manifests, the method *readManifest(String filepath, Stock inventory)*. This will return a *Manifest* object which displays the ordered *Items*, and on which *Truck* they are located.

Sales logs representing items sold can also be loaded using functions supplied by *CSVMachine.java*. This is done using the *readSalesLog(String filepath)* which will return a *HashMap<String,Integer>* containing the data within the selected CSV file.

Writing the manifest is done using the *writeManifest(Manifest manifest, String filepath)* to create/overwrite a CSV file at the given *filepath* with the data stored in the *Manifest* object. This is formatted as per the specification.

CSVFormatException.java

When any error occurs in relation to a CSV action being performed (for example: a user selects a .xls file instead of a .csv file), an exception must be thrown and handled appropriately.

This is done using CSVFormatException.java which supplies CSVException(*String* message) to relay a detailed description of the issue.

Truck.java

In this inventory management system, when a store requires a reorder be actioned for a group of *Item* objects, it involves the creation of a certain number of *Truck* objects to transport said objects to the store.

Truck.java is an implementation of object-oriented programming's ability to have an object take on multiple "forms". This is a method of inheritance known as Polymorphism and is widely used to create objects which require different methods but have many common elements function using the same interface.

Truck.java acts as an abstract super-class to other *Truck* types by providing parent methods which must be implemented.

RefrigeratedTruck.java

The RefrigeratedTruck.java class inherits Truck.java and allows the creation of a refrigeratedTruck() object.

A *refrigeratedTruck* holds a *Stock* object named cargoStock. This can be set using the add(*Stock* object) method provided by RefrigeratedTruck.java.

A *refrigeratedTruck* object also has getter methods getTemp() to get the temperature of the truck, and getCost() to get the total cost of the truck, calculated as per specification.

OrdinaryTruck.java

The OrdinaryTruck.java class inherits Truck.java and allows the creation of a ordinaryTruck() object.

An *ordinaryTruck* holds a *Stock* object named cargoStock. This can be set using the add(*Stock* object) method provided by OrdinaryTruck.java.

A *refrigeratedTruck* object also has getter methods getTemp() to get the temperature of the truck, and getCost() to get the total cost of the truck, calculated as per specification.

Manifest.java

Manifest.java is used to provide ordering functionality to the system along with optimising trucks and their cargo to keep costs as low as possible.

This class provides methods to import *Stock* objects(`addItemStock(Stock)`), create *Truck* objects(`createTrucks()`) and add imported *Stock* objects to the created *Truck* objects(`loadCargoToTrucks()`).

To keep track of the *Truck* objects, they are held in an `ArrayList<Truck>`. There are various getter methods to receive data regarding the *Manifest* (e.g. `getOptimisedCargo()` which returns a *Stock* object representing the optimised cargo). Another important getter method is the `getManifestCost()` which returns the total calculated cost of a *Manifest* in its entirety.

DeliveryException.java

If ever there is an error regarding a delivery (any act of creating or loading a manifest), it must be dealt with effectively by catching and correcting the error or at the very least providing as much possible insight to the user.

This is done using `DeliveryException.java` and its method `DeliveryException(String message)`, which provides the user a message regarding likely causes and possible solutions to the issue.

GUIApplication.java

`GUIApplication.java` is the program's primary class as it is the most reactive to a user's inputs.

It utilises *SwingGUI's* Swing utilities to display an interactive user interface to manage the *Store* object's inventory.

Upon calling the `Main()` method in `GUIApplication.java`, a GUI application is initialized and the program waits for a user *ActionEvent*.

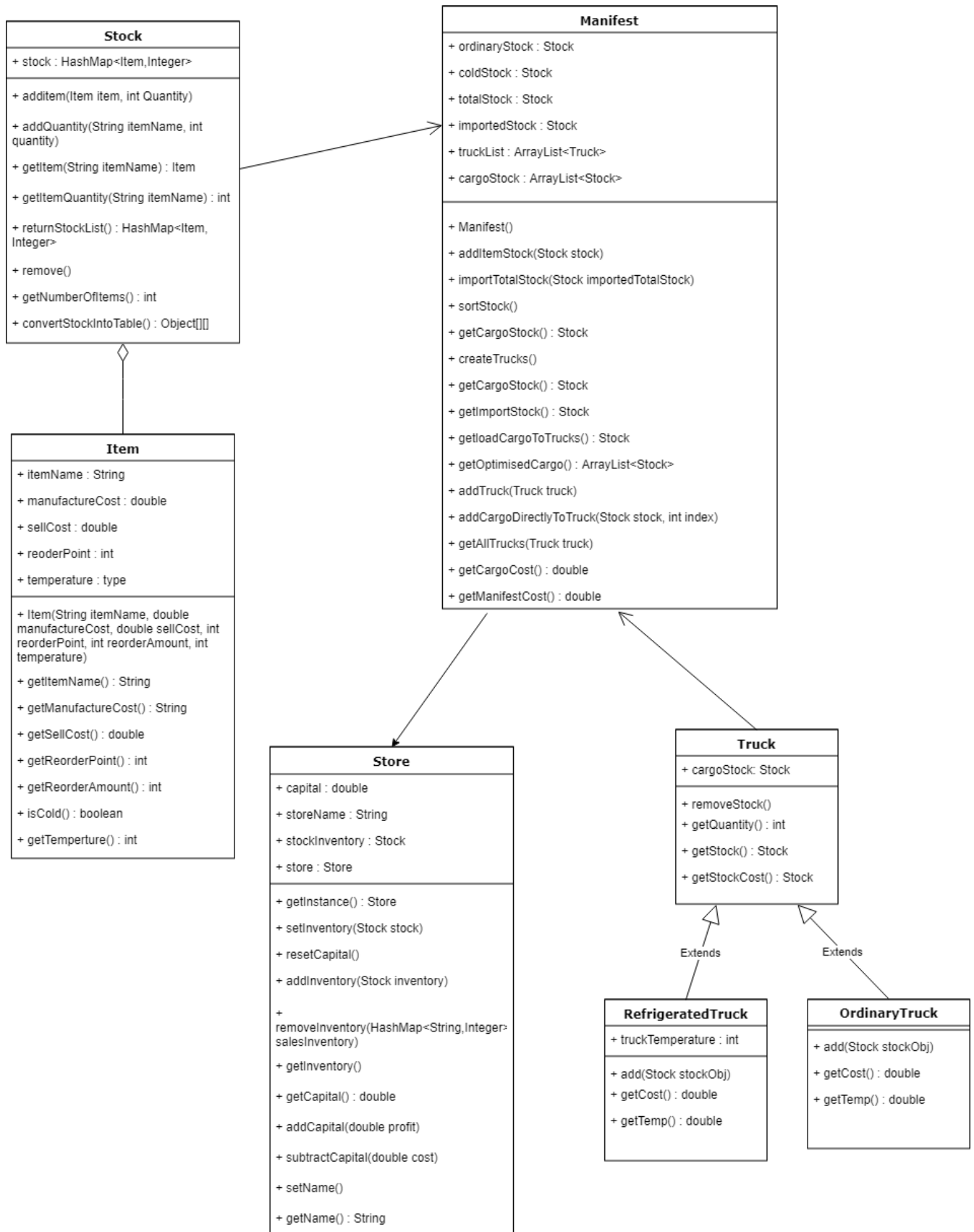
The method `actionPerformed(ActionEvent events)` prompts the execution of commands. For example, when a user clicks the "Extract Manifest" button, a manifest is created and populated with the *Store* object's inventory which needs reordering. This then calls the `writeManifest(String filepath)` method of `CSVMachine.java`, creating a manifest file ready to be loaded into the system by a manager.

`GUIApplication.java` interacts with and regularly calls from all classes to create a functional stock management interface.

ApplicationEntryPoint.java

`ApplicationEntryPoint.java` acts as the program's entry point and provides a *Main* loop which accesses `GUIApplication.java` to run its `Main()` method.

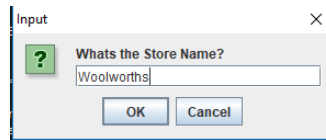
Class Diagram



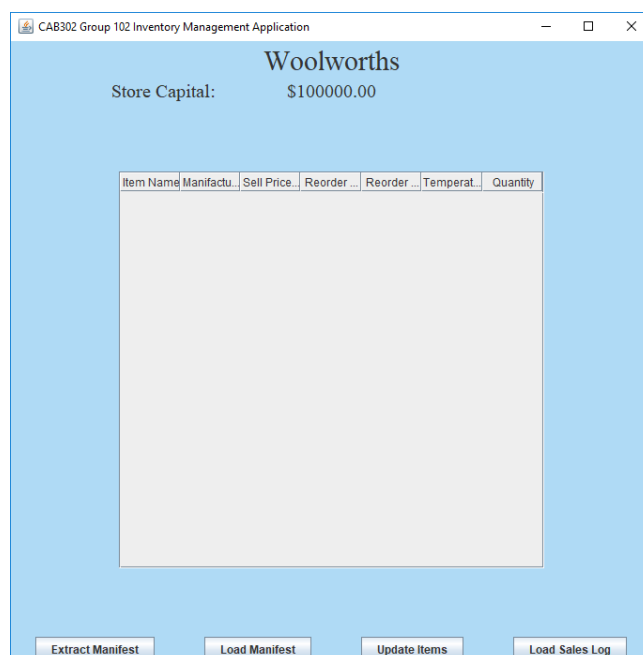
GUI Test Report

Initialization

Upon initialising the GUI we are greeted with an Input popup box prompting the user to enter the store name

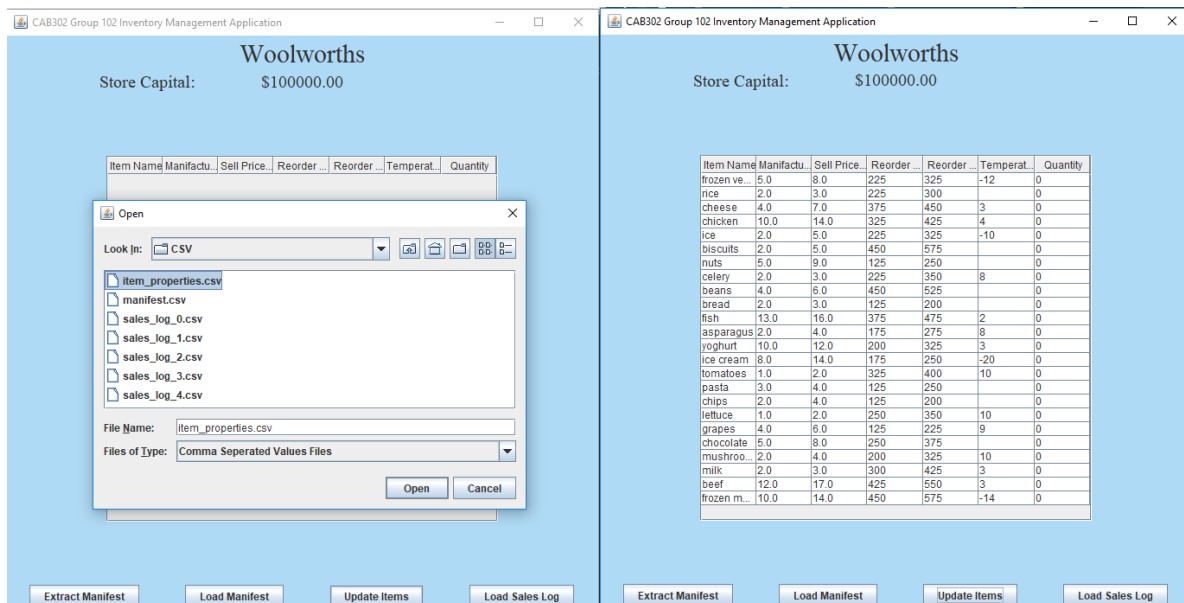


After the user enters a store name, the store name is displayed in the GUI along with its starting capital.



Loading Item Properties

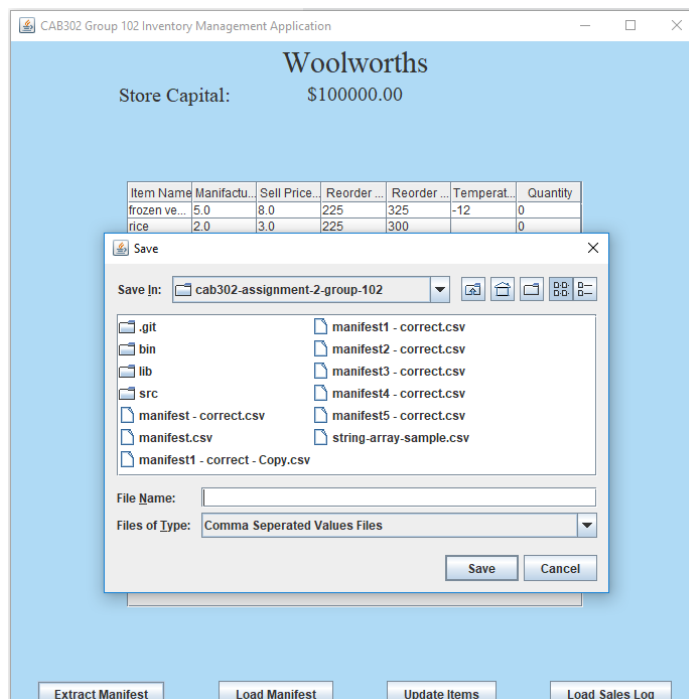
To load in the Item Properties, the Update Items button is clicked which will then display an open file dialog where the user can then select the Item Properties CSV file.



When the Item properties document is loaded the table is automatically populate the table with all the items with a starting quantity of 0.

Extract Manifest

To extract a manifest the button “Extract Manifest” needs to be clicked. Upon clicking a save file dialog will open where the user can save the manifest as whatever file name they want.

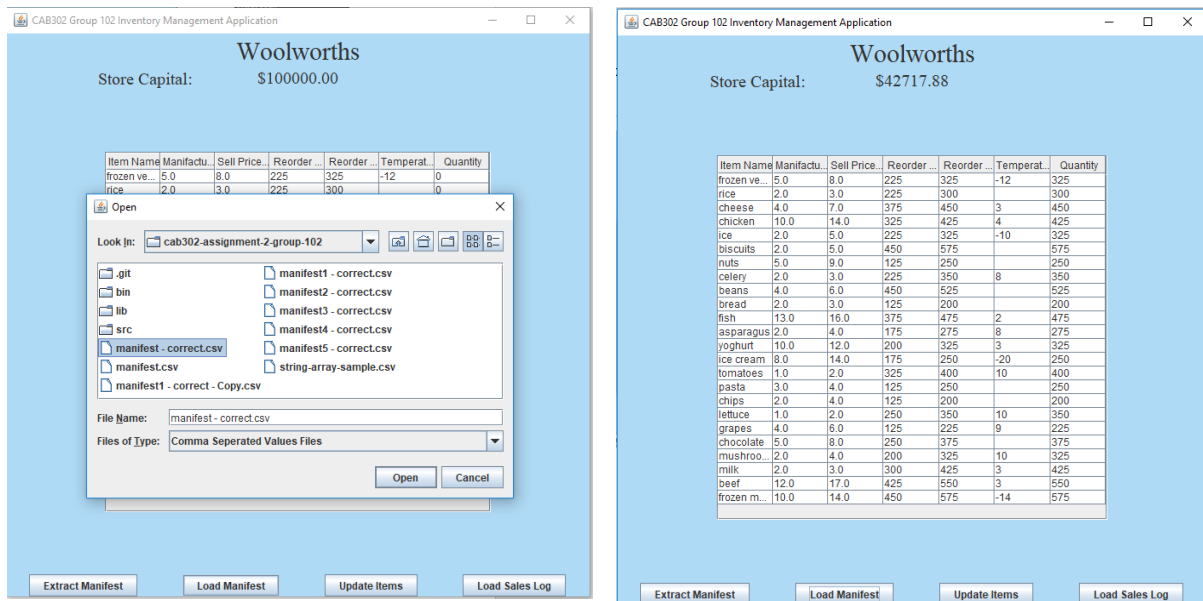


```
>Refrigerated  
ice cream,250  
frozen meat,550  
>Refrigerated  
frozen vegetable mix,325  
ice,325  
fish,125  
frozen meat,25  
>Refrigerated  
beef,450  
fish,350  
>Refrigerated  
beef,100  
cheese,450  
yoghurt,250  
>Refrigerated  
chicken,300  
yoghurt,75  
milk,425  
>Refrigerated  
asparagus,275  
celery,350  
grapes,50  
chicken,125  
>Refrigerated  
grapes,175  
lettuce,225  
tomatoes,400  
>Refrigerated  
pasta,250  
lettuce,125  
biscuits,100  
mushrooms,325  
>Ordinary  
biscuits,475  
bread,200  
rice,300  
nuts,25  
>Ordinary  
nuts,225  
chocolate,375  
beans,400  
>Ordinary  
beans,125  
chips,200
```

The picture on the right shows the output of the generated manifest in a desired CSV format setting.

Load Manifest

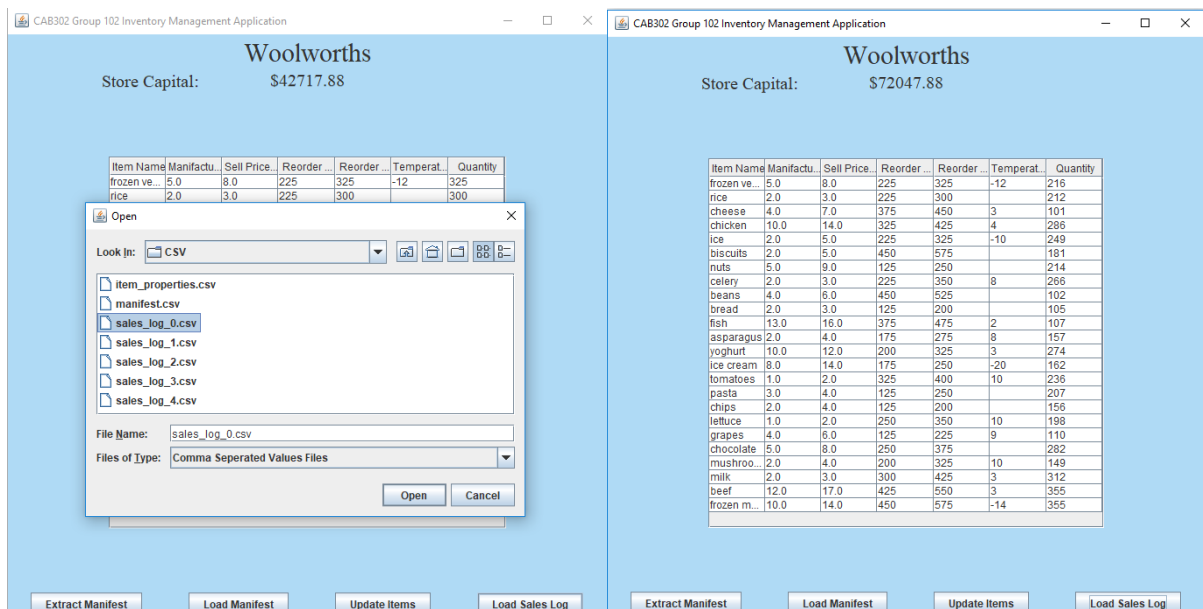
To load in manifest the “Load Manifest” button must be clicked. Upon clicking it will then prompt the user select the manifest file.



The inventory stock is then update when the manifest is loaded, and the store capital decreases as expected

Load Sales Log

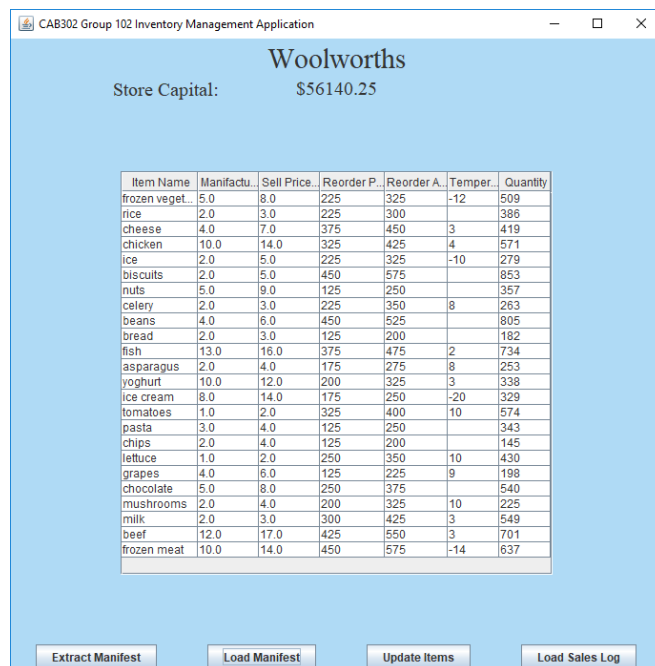
When the “Load Sales Log” button is clicked an open file dialog opens where the user can select the sales log they want to import.



When the sales log is imported the inventory is then updated and the store capital increases as expected.

Output of all Sales Log and Manifest imported

After all 4 sales logs were imported and all 5 manifests were imported, this was the result of the current store capital and the current store stock.

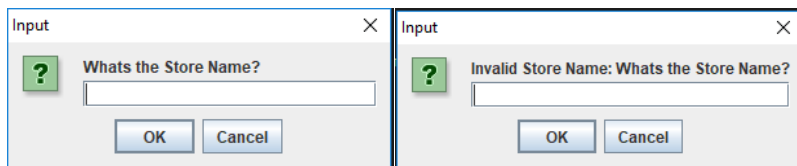


The screenshot shows the 'Woolworths' application window. At the top, it displays 'Store Capital: \$56140.25'. Below this is a table with 7 columns: Item Name, Manufactu., Sell Price., Reorder P., Reorder A., Temper., and Quantity. The table lists 25 items with their respective values. At the bottom of the window are four buttons: 'Extract Manifest', 'Load Manifest', 'Update Items', and 'Load Sales Log'.

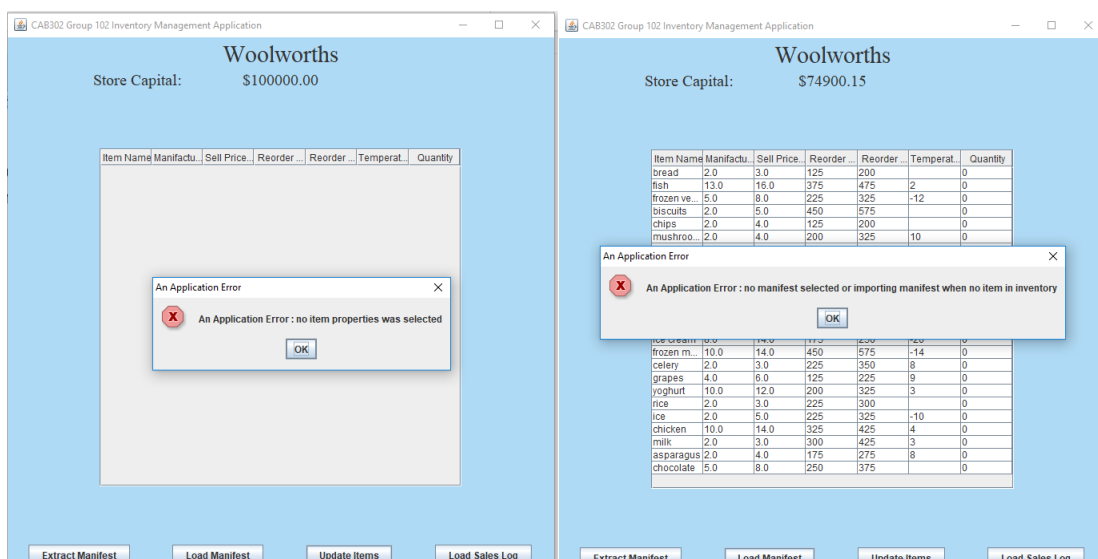
Item Name	Manufactu.	Sell Price.	Reorder P.	Reorder A.	Temper.	Quantity
frozen veget.	5.0	8.0	225	325	-12	509
rice	2.0	3.0	225	300		386
cheese	4.0	7.0	375	450	3	419
chicken	10.0	14.0	325	425	4	571
ice	2.0	5.0	225	325	-10	279
biscuits	2.0	5.0	450	575		853
nuts	5.0	9.0	125	250		357
celery	2.0	3.0	225	350	8	263
beans	4.0	6.0	450	525		805
bread	2.0	3.0	125	200		182
fish	13.0	16.0	375	475	2	734
asparagus	2.0	4.0	175	275	8	253
yoghurt	10.0	12.0	200	325	3	338
ice cream	8.0	14.0	175	250	-20	329
tomatoes	1.0	2.0	325	400	10	574
pasta	3.0	4.0	125	250		343
chips	2.0	4.0	125	200		145
lettuce	1.0	2.0	250	350	10	430
grapes	4.0	6.0	125	225	9	198
chocolate	5.0	8.0	250	375		540
mushrooms	2.0	4.0	200	325	10	225
milk	2.0	3.0	300	425	3	549
beef	12.0	17.0	425	550	3	701
frozen meat	10.0	14.0	450	575	-14	637

Exception Handling

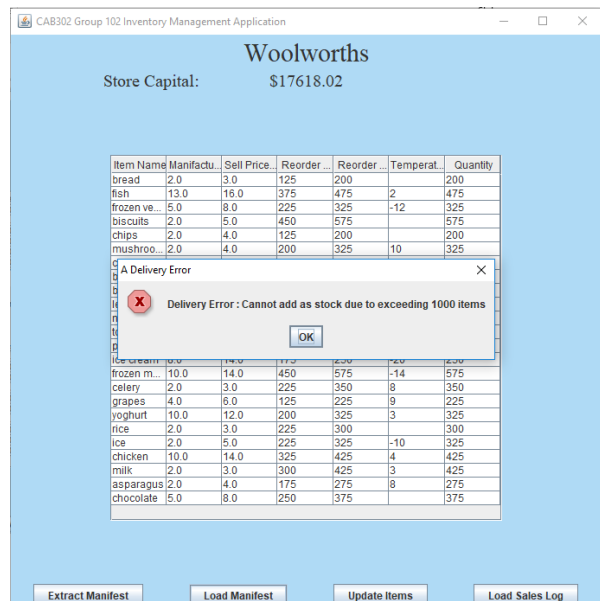
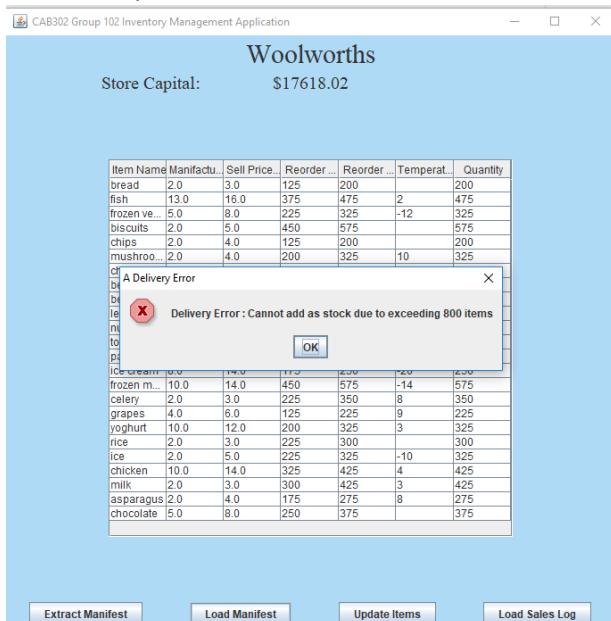
The first software error handling that occurs is when the user doesn't enter anything in the Input Popup box it will keep appearing until the user enters a valid store name.



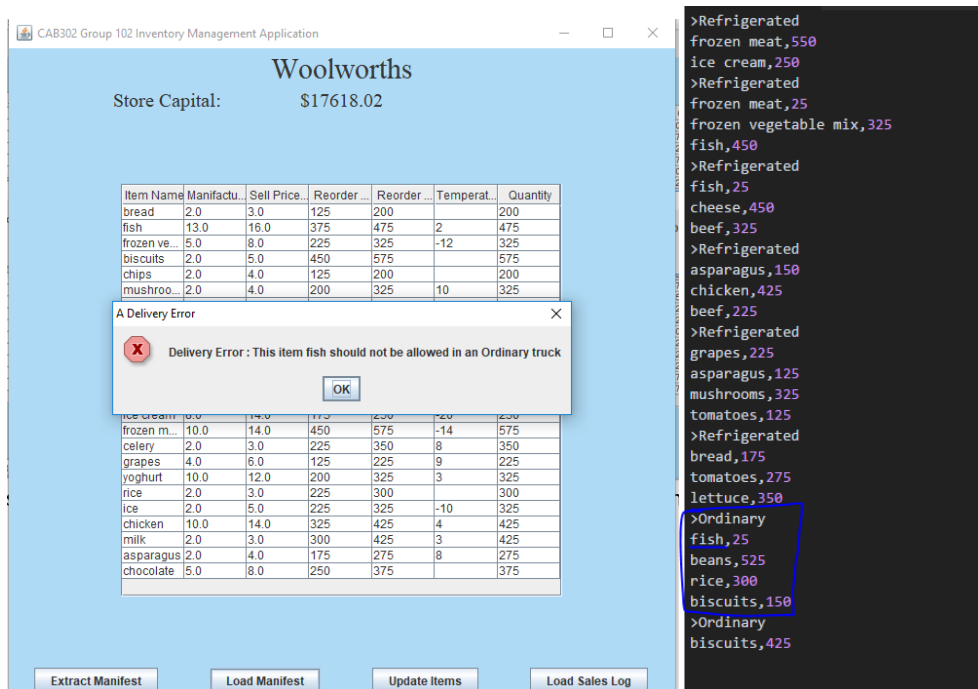
When the Update Items button is clicked and another CSV that isn't Item Properties is selected this error will appear this occurs for the same for the load manifest button and sales log button.



If the manager decides to modify the extracted manifest and modifies one of the truck cargo in this case a refrigerated truck to exceed 800 item then this error will pop up the same will occur for ordinary truck.

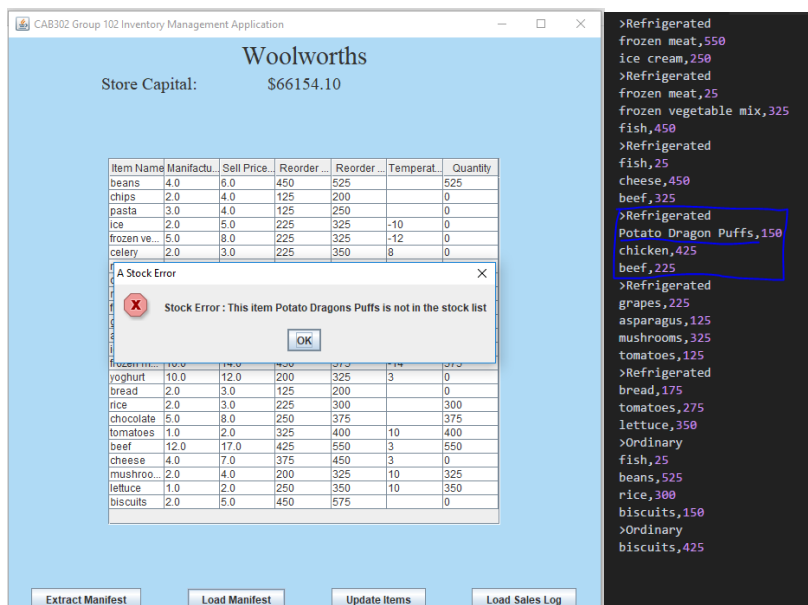


If the manager edits the manifest to put a refrigerated item in an ordinary truck the extracted, then this error will pop up.

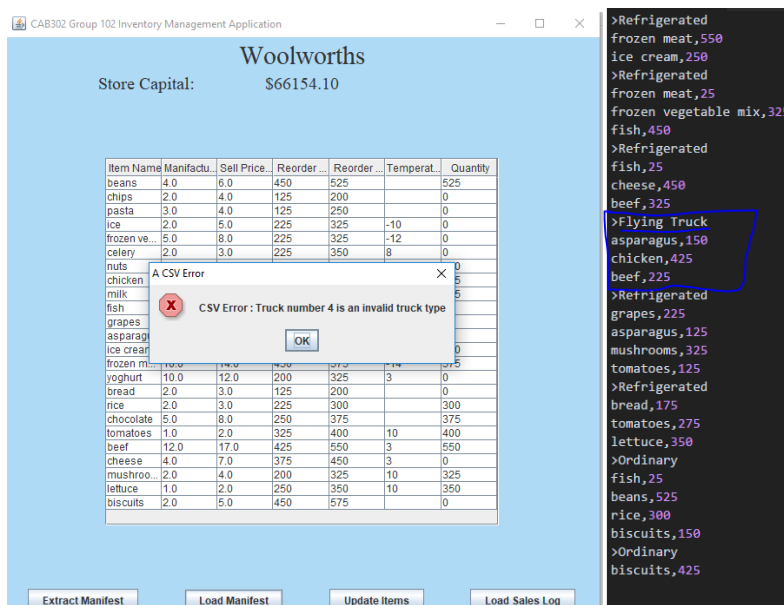


```
>Refrigerated
frozen meat,550
ice cream,250
>Refrigerated
frozen meat,25
frozen vegetable mix,325
fish,450
>Refrigerated
fish,25
cheese,450
beef,325
>Refrigerated
asparagus,150
chicken,425
beef,225
>Refrigerated
grapes,225
asparagus,125
mushrooms,325
tomatoes,125
>Refrigerated
bread,175
tomatoes,275
lettuce,350
>Ordinary
fish,25
beans,525
rice,300
biscuits,150
>Ordinary
biscuits,425
```

If the manager edits the manifest further and adds an item that is not currently in the inventory, then this error will appear.



If the manager edits the manifest CSV and modifies a truck type to something else that isn't ordinary or refrigerated truck then this error will appear



If the manager modifies the manifest and replaces a quantity number with a string then this error will occur.

The screenshot shows the 'Woolworths' inventory management application. The 'Store Capital' is \$72047.88. A table lists various items with their manufacturing dates, sell prices, reorder points, and quantities. A modal dialog box titled 'A CSV Error' is displayed, showing the message: 'CSV Error: The Value sadsadasdsad is not a number'. The dialog has an 'OK' button. To the right of the application window, a list of items is shown with their categories and quantities, including 'frozen meat, 550', 'ice cream, 250', 'Refrigerated', 'frozen meat, 25', 'frozen vegetable mix, sadsadasdsad', 'fish, 450', 'Refrigerated', 'fish, 25', 'cheese, 450', 'beef, 325', 'Refrigerated', 'asparagus, 150', 'chicken, 425', 'beef, 225', 'Refrigerated', 'grapes, 225', 'asparagus, 125', 'mushrooms, 325', 'tomatoes, 125', 'Refrigerated', 'bread, 175', 'tomatoes, 275', 'lettuce, 350', 'Ordinary', 'fish, 25', 'beans, 525', 'rice, 300', 'biscuits, 150', 'Ordinary', and 'biscuits, 425'.

If the manager modifies the manifest to removes all stock for a truck this error occurs demonstrating that the cost can't be calculated for an empty truck.

The screenshot shows the 'Woolworths' inventory management application. The 'Store Capital' is \$66154.10. A table lists various items with their manufacturing dates, sell prices, reorder points, and quantities. A modal dialog box titled 'A Delivery Error' is displayed, showing the message: 'Delivery Error : Cannot get Cargo Cost as there is no cargo in the truck'. The dialog has an 'OK' button. To the right of the application window, a list of items is shown with their categories and quantities, including 'Refrigerated', 'frozen meat, 550', 'ice cream, 250', 'Refrigerated', 'frozen meat, 25', 'frozen vegetable mix, 325', 'fish, 450', 'Refrigerated', 'fish, 25', 'cheese, 450', 'beef, 325', 'Refrigerated', 'Refrigerated', 'grapes, 225', 'asparagus, 125', 'mushrooms, 325', 'tomatoes, 125', 'Refrigerated', 'bread, 175', 'tomatoes, 275', 'lettuce, 350', 'Ordinary', 'bread, 25', 'beans, 525', 'rice, 300', 'biscuits, 150', 'Ordinary', and 'biscuits, 425'.

If the manager modifies the sales log to sell more what he has in the inventory, then this error will appear

The screenshot shows the 'Woolworths' Inventory Management Application window. The 'Store Capital' is \$42717.88. An error message box is displayed with the text: 'Stock Error : Cannot remove item biscuits due to remove quantity is greater then item quantity'. The background table lists various items with their respective prices and quantities. To the right of the application window, a list of items and their quantities is shown, including rice, beans, pasta, biscuits, nuts, chips, chocolate, bread, mushrooms, tomatoes, lettuce, grapes, asparagus, celery, chicken, beef, fish, yoghurt, milk, cheese, ice cream, ice, frozen meat, and frozen vegetable mix.

Item Name	Manufact...	Sell Price...	Reorder ...	Reorder ...	Temperat...	Quantity
chocolate	5.0	8.0	250	375		375
asparagus	2.0	4.0	175	275	8	275
lettuce	1.0	2.0	250	350	10	350
bread	2.0	3.0	125	200		200
cheese	4.0	7.0	375	450	3	450
ice	2.0	5.0	225	325	-10	325

rice,88
beans,423
pasta,43
biscuits,999999
nuts,36
chips,44
chocolate,93
bread,95
mushrooms,176
tomatoes,164
lettuce,152
grapes,115
asparagus,118
celery,84
chicken,139
beef,195
fish,368
yoghurt,51
milk,113
cheese,349
ice cream,88
ice,76
frozen meat,220
frozen vegetable mix,109

If the manager modifies the sales log to show an item being sold when it doesn't exist in the inventory then this error will show.

The screenshot shows the 'Woolworths' Inventory Management Application window. The 'Store Capital' is \$42717.88. An error message box is displayed with the text: 'Stock Error : Cannot get item Potato Pies quantity because item doesn't exist in stock'. The background table lists various items with their respective prices and quantities. To the right of the application window, a list of items and their quantities is shown, including rice, beans, pasta, Potato Pies, nuts, chips, chocolate, bread, mushrooms, tomatoes, lettuce, grapes, asparagus, celery, chicken, beef, fish, yoghurt, milk, cheese, ice cream, ice, frozen meat, and frozen vegetable mix.

Item Name	Manufact...	Sell Price...	Reorder ...	Reorder ...	Temperat...	Quantity
fish	13.0	16.0	375	475	2	475
beans	4.0	6.0	450	525		525
tomatoes	1.0	2.0	325	400	10	400
bread	2.0	3.0	125	200		200
beef	12.0	17.0	425	550	3	550
nuts	5.0	9.0	125	250		250
chicken	10.0	14.0	325	425	4	425

lettuce,152
yoghurt,51
chocolate,93
rice,88
mushrooms,176
ice,76
pasta,43
asparagus,118
ice cream,88
frozen m...,109