2018

CAB302 Project: Inventory Management Application

BY:

ALEXANDER ROZSA: N9992529

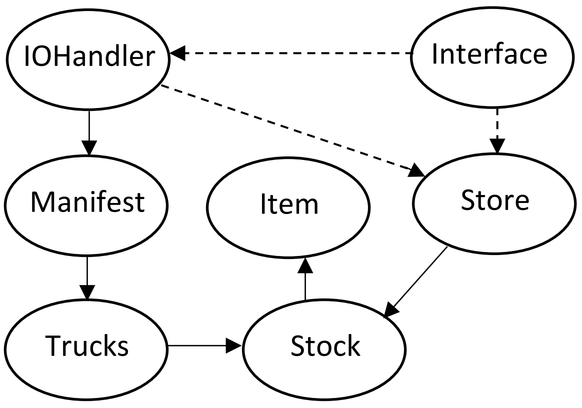
LIAM EDWARDS: N9729241

Technical Description

The aim of this project was to have a GUI-based application that takes defining back-end classes and displays outputs based on their values. The values needed to be displayed, of course, are the items contained within the "store" and their quantities as a means of managing the store's inventory. The below diagram is a representation of the program's classes and their dependencies ordered from the main class (Interface) down to the class which doesn't have any dependencies (Item). Interface doesn't have any direct dependencies; however, it does make calls to Store for info on the store and IOHandler to import and export data. IOHandler makes calls to Store for store info and relies on Manifest in order to export Manifests. Manifest relies on Trucks since it's a list of Trucks and that's the object type it's supposed to contain. Store and Trucks rely on Stock in order to store Item(s) and their quantities. Finally, Stock relies on Item to have many of them stored within it providing data to the program, whereas Item doesn't have any dependencies; it just supplies data.

A $\downarrow$ **B** = **A** has a direct dependency on **B** (constructs and contains an object of that class).

A $\dashdownarrow$ **B** = **A** has an indirect dependency on **B** (calls the public static functions).



Interface is both the GUI and the program entry point which applies polymorphism by extending JFrame and implementing ActionListener and ChangeListener. The program is begun by creating a new Interface with a whole bunch of GUI objects and their properties, along with the public static declaration of the Store. The extension of JFrame allows JFrame functions such as setVisible() and getContentPane() to be called without declaring a JFrame within the class and stating the name of that before every JFrame function call. Instead, the class IS the JFrame variable. Since Interface implements ActionListener, it has the additional function actionPerformed, which allows all the outputs of the buttons to be put into one function instead of declaring a new function every time a button needs an ActionListener. The same thing happens with ChangeListener, except stateChanged is the function that's added in, which loads in the store's inventory into a graph when the inventory tab is selected.

Store is the primary back-end class to the entire program. This singleton class represents the store and its values. It applies encapsulation by keeping its values private and has them get/set by using public functions. Since there will only be one store, it's better to have a static singleton store publicly accessible from any class without the need for local storage instead of declaring a new store every time in each class it's needed.

IOHandler is similar to the way Store is accessed, however it doesn't even have any type of constructor. It supplies the interface with file reading and writing functions in order to update the program.

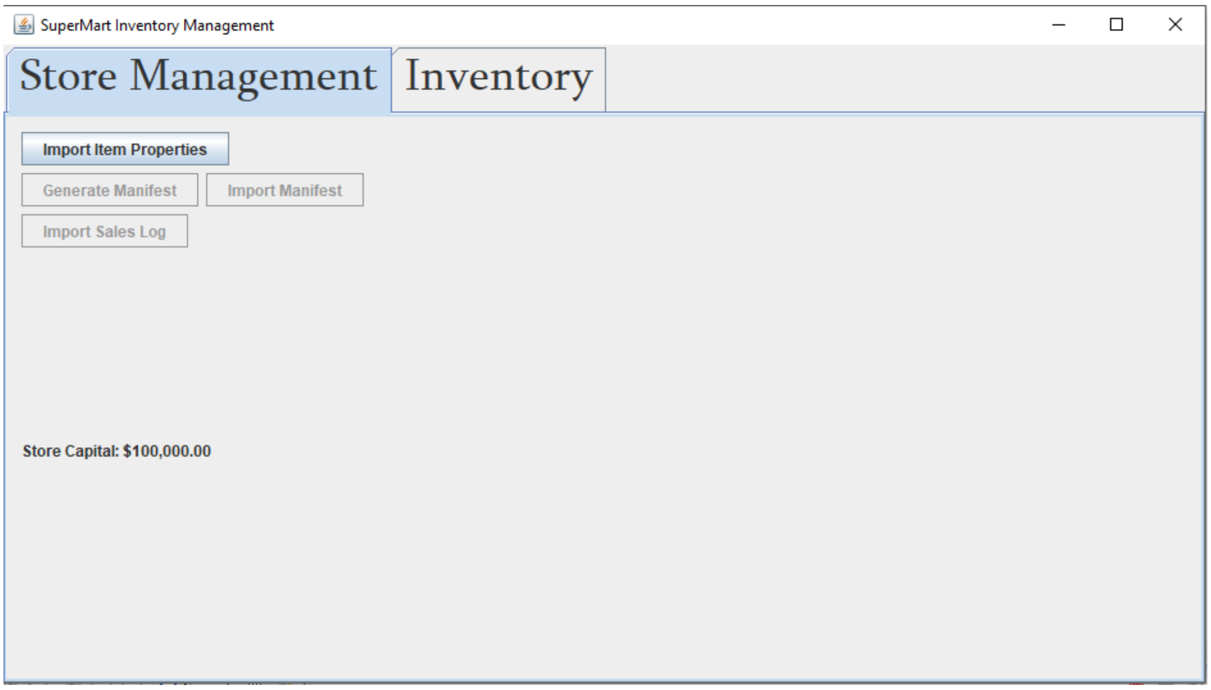
Manifest is a list of trucks. Like Store, it applies encapsulation by hiding a List of type Truck, and sets/gets its data with publicly available functions. It implements Iterable so that foreach loops can iterate off the class itself. It is used in IOHandler for the exportation of manifests.

The "Trucks" bubble in the diagram is not one class; it is a categorization of three classes: Truck, OrdinaryTruck and RefrigeratedTruck. Truck applies abstraction by being an abstract class that supplies basic truck functions to its inheritors: OrdinaryTruck and RefrigeratedTruck (which apply inheritance). Truck also applies encapsulation by having the maximum capacity and stock variables kept private, and even more so for RefrigeratedTruck since it additionally has temperature.

Simply put, Stock is the most important class in the program. It holds a HashMap that has integers assigned to Item keys which represent the quantity of each item. As shown in the diagram, Stock is used by Store, which allows the store to get items from its Stock variable and output it to the Interface, and also by Truck which allows costs to be calculated based on the number of items stored. It applies encapsulation for the HashMap and it also applies inheritance by implementing Iterable.

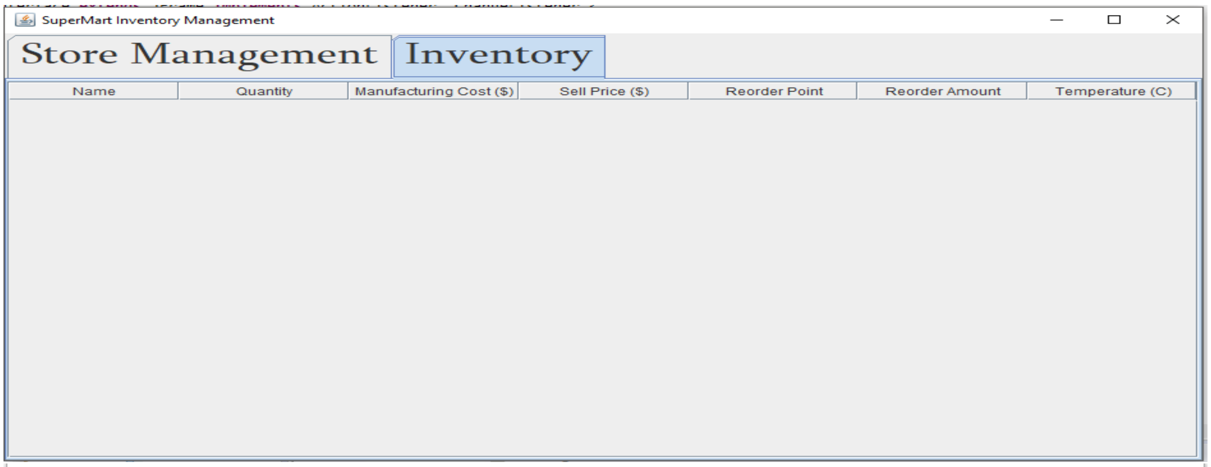
Lastly, Item is the class the program revolves around. Items encapsulate 6 pieces of data that define an item. These item properties determine whether it exists in a Stock object, how much to deduct or add to the Stores's capital, when to reorder and how much to reorder, and whether the item can be added into a specific type of Truck. It's a basic class that doesn't have any setter functions besides the constructors; only getters.

GUI Test Report

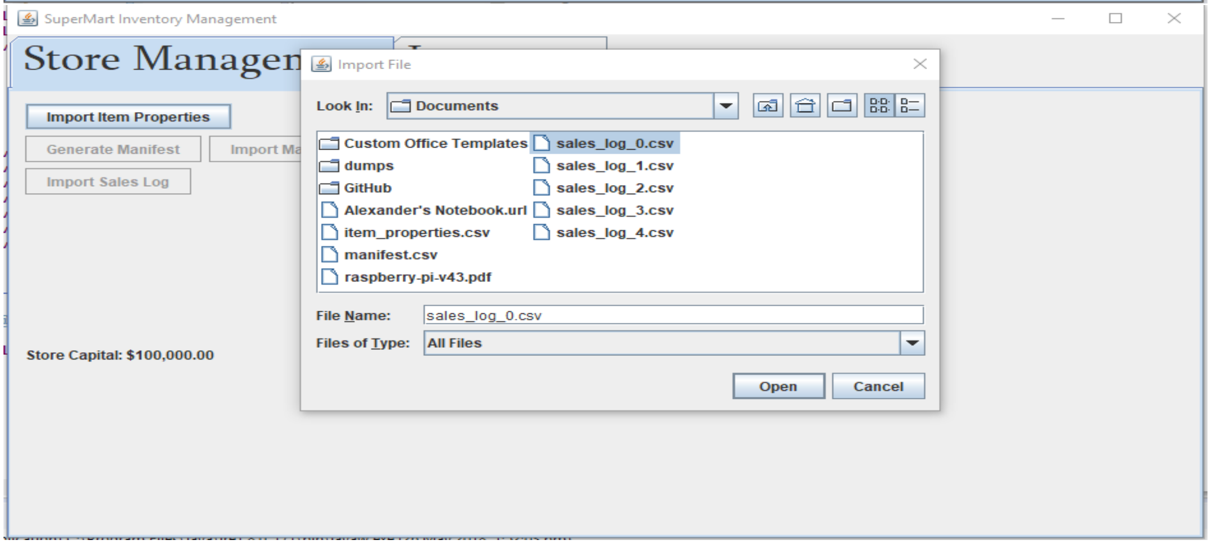


Store Capital: $100,000.00

This is the screen that meets users when the application is first opened. The only button enabled is the 'Import Item Properties' button, which only gives the user the option to import the item properties, as well as no items in Inventory (below).



Name	Quantity	Manufacturing Cost ($)	Sell Price ($)	Reorder Point	Reorder Amount	Temperature (C)

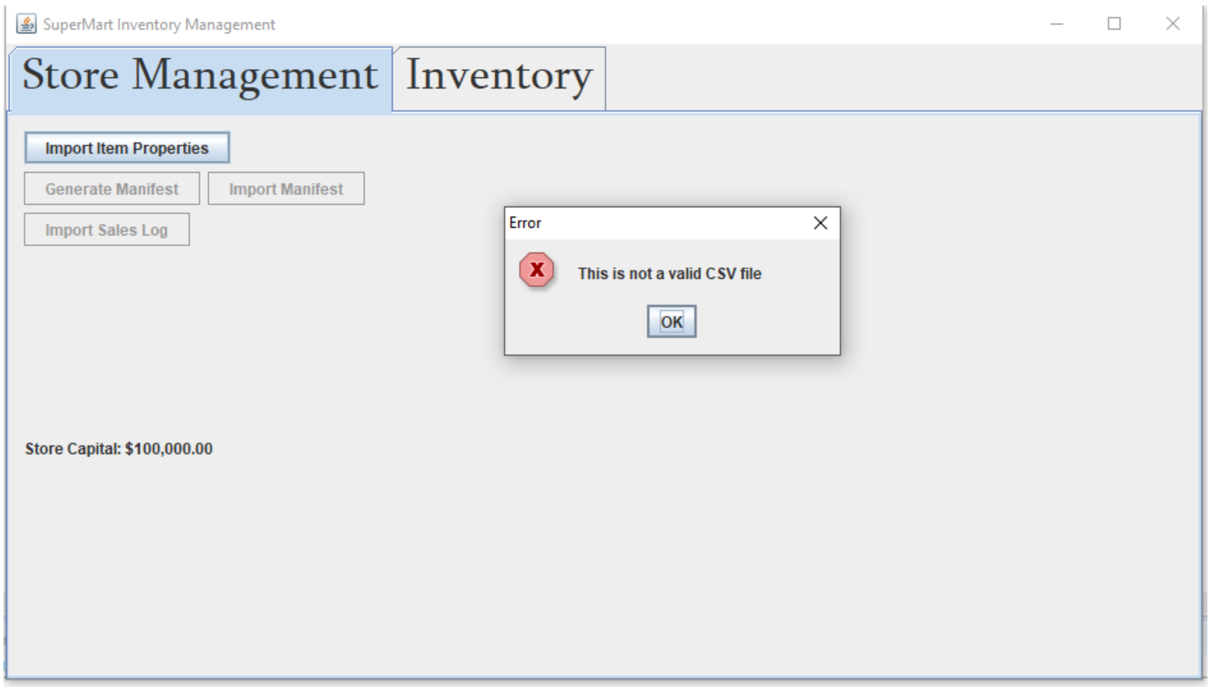


Store Capital: $100,000.00

File Name: sales_log_0.csv

Files of Type: All Files

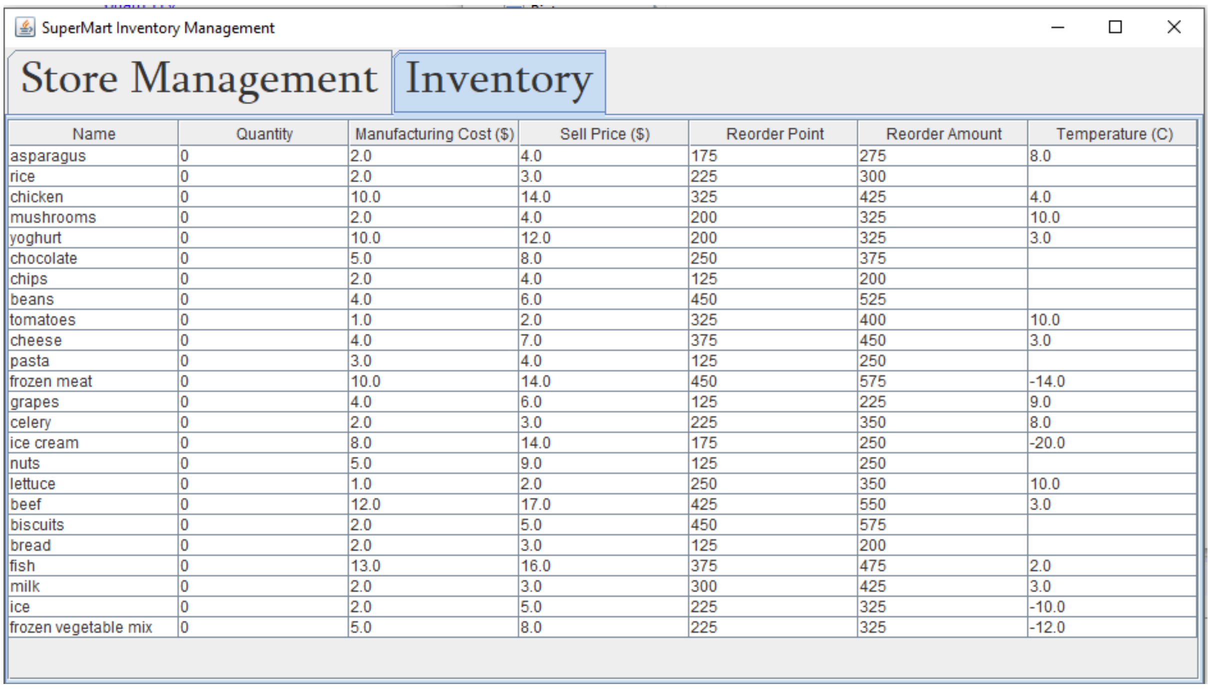
When the user goes to import an invalid item as shown above, they'll be greeted with an error message saying the file format was wrong.



This is not a valid CSV file

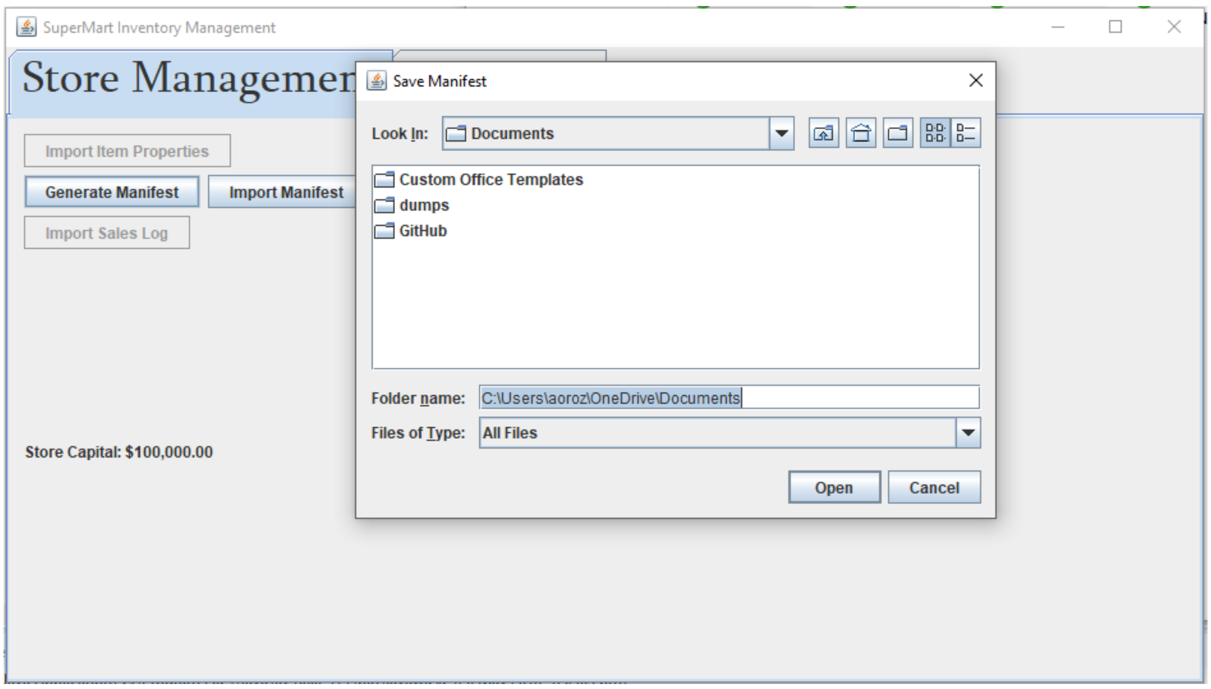
Store Capital: $100,000.00

However, once they go to import the correct file, the inventory tab will be updated with items.



Name	Quantity	Manufacturing Cost ($)	Sell Price ($)	Reorder Point	Reorder Amount	Temperature (C)
asparagus	0	2.0	4.0	175	275	8.0
rice	0	2.0	3.0	225	300	
chicken	0	10.0	14.0	325	425	4.0
mushrooms	0	2.0	4.0	200	325	10.0
yoghurt	0	10.0	12.0	200	325	3.0
chocolate	0	5.0	8.0	250	375	
chips	0	2.0	4.0	125	200	
beans	0	4.0	6.0	450	525	
tomatoes	0	1.0	2.0	325	400	10.0
cheese	0	4.0	7.0	375	450	3.0
pasta	0	3.0	4.0	125	250	
frozen meat	0	10.0	14.0	450	575	-14.0
grapes	0	4.0	6.0	125	225	9.0
celery	0	2.0	3.0	225	350	8.0
ice cream	0	8.0	14.0	175	250	-20.0
nuts	0	5.0	9.0	125	250	
lettuce	0	1.0	2.0	250	350	10.0
beef	0	12.0	17.0	425	550	3.0
biscuits	0	2.0	5.0	450	575	
bread	0	2.0	3.0	125	200	
fish	0	13.0	16.0	375	475	2.0
milk	0	2.0	3.0	300	425	3.0
ice	0	2.0	5.0	225	325	-10.0
frozen vegetable mix	0	5.0	8.0	225	325	-12.0

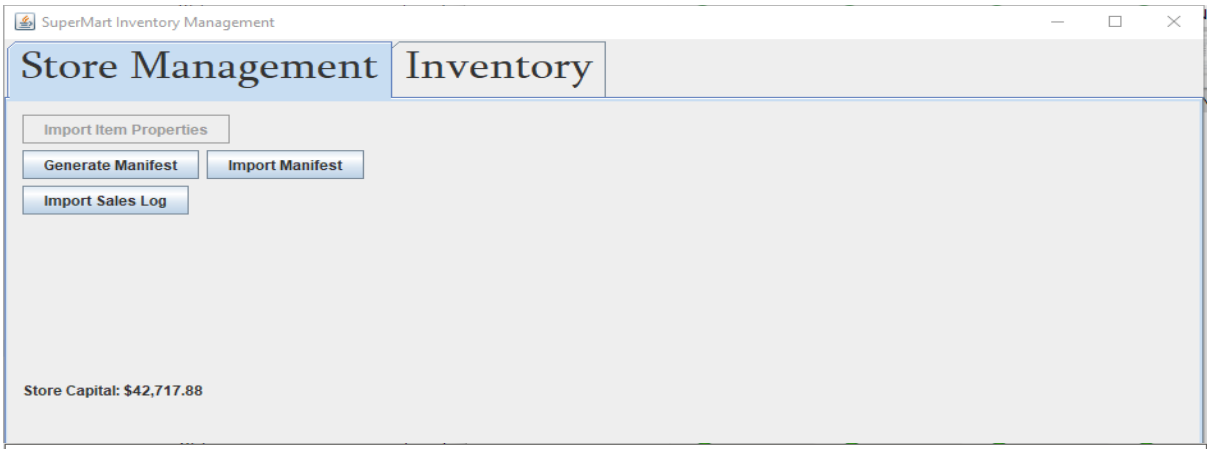
After that the 'Import Item Properties' button will be disabled, while the 'Generate Manifest' and 'Import Manifest' buttons will be enabled. When 'Generate Manifest' is pressed, a dialogue box will come up for the user to select the export path for the manifest. There is no possibility of choosing a wrong file here; the user can't fail.



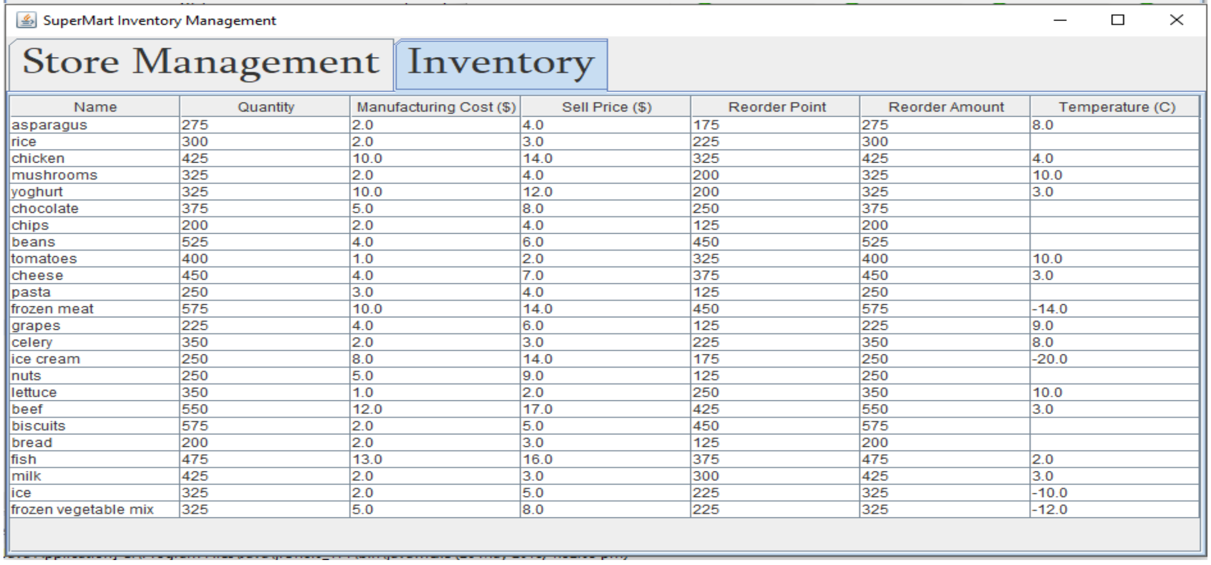
Folder name: C:\Users\aoroz\OneDrive\Documents

Store Capital: $100,000.00

Next, when the user clicks on 'Import Manifest' and they try to import something that isn't a manifest file, an error just like the error for choosing the incorrect file for importing item properties will appear. However once they import they correct file, the 'Import Sales Log' button will be enabled, and the store capital and inventory will be updated:

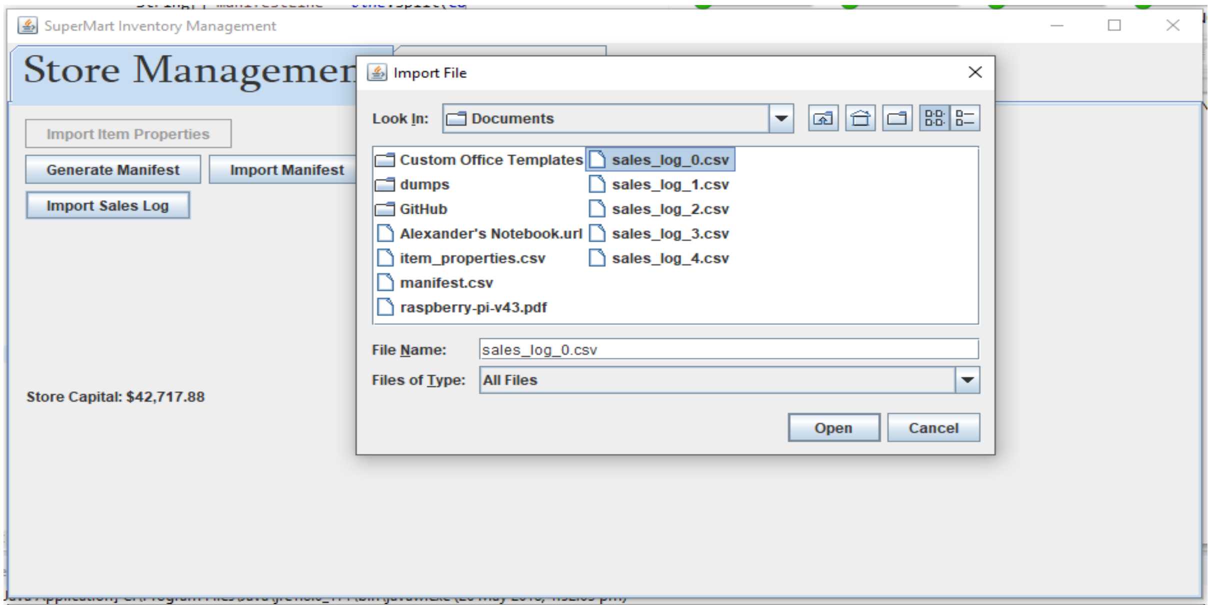


Store Capital: $42,717.88



Name	Quantity	Manufacturing Cost ($)	Sell Price ($)	Reorder Point	Reorder Amount	Temperature (C)
asparagus	275	2.0	4.0	175	275	8.0
rice	300	2.0	3.0	225	300	
chicken	425	10.0	14.0	325	425	4.0
mushrooms	325	2.0	4.0	200	325	10.0
yoghurt	325	10.0	12.0	200	325	3.0
chocolate	375	5.0	8.0	250	375	
chips	200	2.0	4.0	125	200	
beans	525	4.0	6.0	450	525	
tomatoes	400	1.0	2.0	325	400	10.0
cheese	450	4.0	7.0	375	450	3.0
pasta	250	3.0	4.0	125	250	
frozen meat	575	10.0	14.0	450	575	-14.0
grapes	225	4.0	6.0	125	225	9.0
celery	350	2.0	3.0	225	350	8.0
ice cream	250	8.0	14.0	175	250	-20.0
nuts	250	5.0	9.0	125	250	
lettuce	350	1.0	2.0	250	350	10.0
beef	550	12.0	17.0	425	550	3.0
biscuits	575	2.0	5.0	450	575	
bread	200	2.0	3.0	125	200	
fish	475	13.0	16.0	375	475	2.0
milk	425	2.0	3.0	300	425	3.0
ice	325	2.0	5.0	225	325	-10.0
frozen vegetable mix	325	5.0	8.0	225	325	-12.0

Finally, an import of a sales log will increase the capital and decrease the inventory:



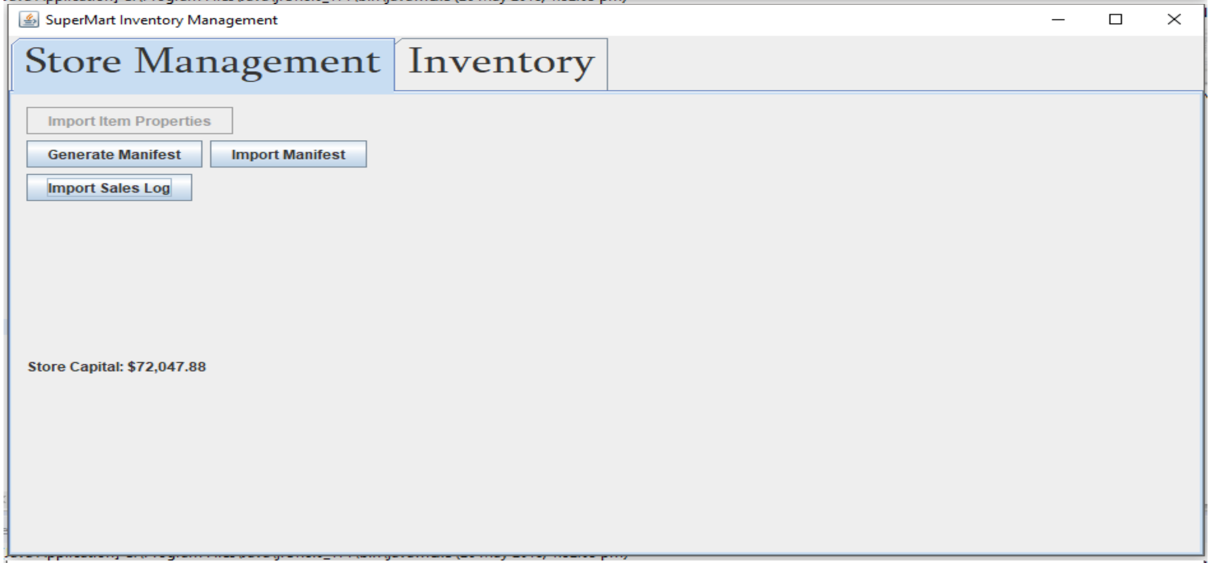
Look In: Documents

Custom Office Templates
dumps
GitHub
Alexander's Notebook.url
item_properties.csv
manifest.csv
raspberry-pi-v43.pdf
sales_log_0.csv
sales_log_1.csv
sales_log_2.csv
sales_log_3.csv
sales_log_4.csv

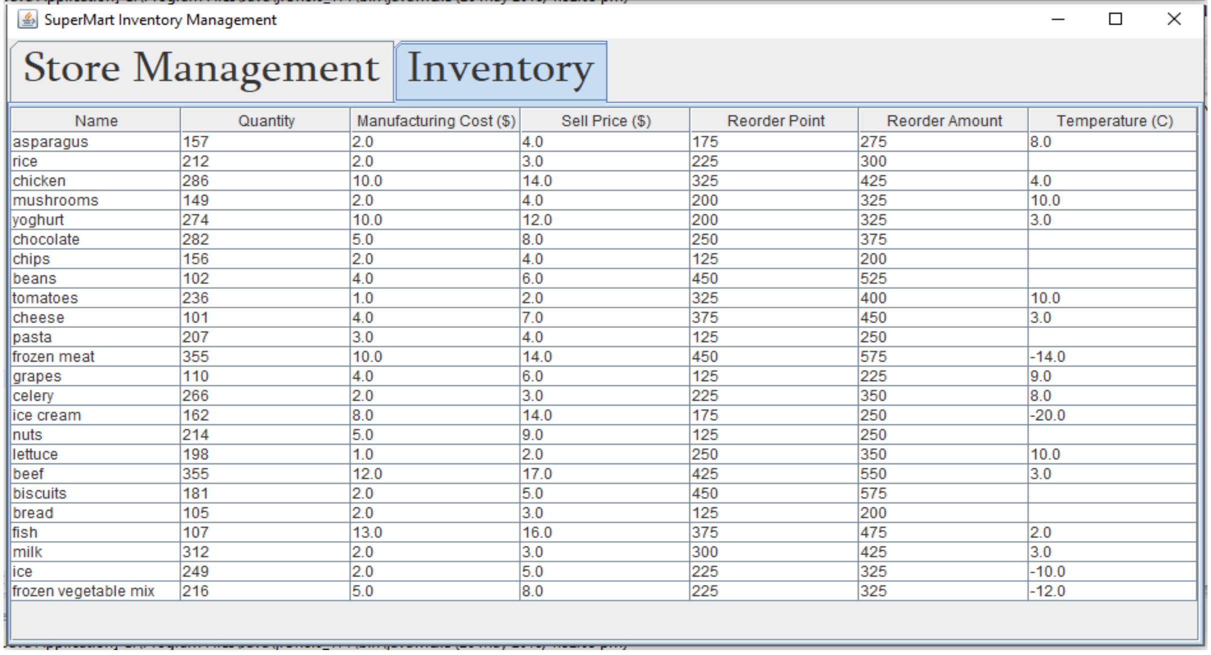
File Name: sales_log_0.csv

Files of Type: All Files

Store Capital: $42,717.88



Store Capital: $72,047.88



Name	Quantity	Manufacturing Cost ($)	Sell Price ($)	Reorder Point	Reorder Amount	Temperature (C)
asparagus	157	2.0	4.0	175	275	8.0
rice	212	2.0	3.0	225	300	
chicken	286	10.0	14.0	325	425	4.0
mushrooms	149	2.0	4.0	200	325	10.0
yoghurt	274	10.0	12.0	200	325	3.0
chocolate	282	5.0	8.0	250	375	
chips	156	2.0	4.0	125	200	
beans	102	4.0	6.0	450	525	
tomatoes	236	1.0	2.0	325	400	10.0
cheese	101	4.0	7.0	375	450	3.0
pasta	207	3.0	4.0	125	250	
frozen meat	355	10.0	14.0	450	575	-14.0
grapes	110	4.0	6.0	125	225	9.0
celery	266	2.0	3.0	225	350	8.0
ice cream	162	8.0	14.0	175	250	-20.0
nuts	214	5.0	9.0	125	250	
lettuce	198	1.0	2.0	250	350	10.0
beef	355	12.0	17.0	425	550	3.0
biscuits	181	2.0	5.0	450	575	
bread	105	2.0	3.0	125	200	
fish	107	13.0	16.0	375	475	2.0
milk	312	2.0	3.0	300	425	3.0
ice	249	2.0	5.0	225	325	-10.0
frozen vegetable mix	216	5.0	8.0	225	325	-12.0