Cab302 Assignment 2
SuperMart Stock System

Locke Nicholls

N9989269

Shahyan Hasan

N9587268

Program Architecture:
Main Entry Point: StoreGUI class.
Class Descriptions:
Item Class: Item is a generic way of representing an item across the application. It is used anywhere another class needs to keep track of items. For instance, when inside the store class, the items represent inventory. When used as a sales log, items represent the type and number of items sold. When used by a truck, they represent the trucks cargo.

Stock Class: Stock is a generic list of items, that can be used anywhere another class needs to keep track of multiple items. For instance, when used in the store class, stock is the list of items in inventory. In a truck, stock is the list of the trucks cargo. When processing sales logs, stock is a list of items sold.

Truck Class: Truck is an abstract class, that is extended by refrigerated and ordinary trucks. Holds a stock object that holds items representing items carried by a truck. Truck defines a number of methods common to all truck subclasses, which allows for polymorphism.

ordinaryTruck class. Ordinary trucks are a concrete implementation of Truck. They extend the class methods, and overwrite methods as necessary to calculate costs and validate their load

refrigeratedTruck class. Refrigerated trucks are a concrete implementation of Truck
They extend the class methods, and overwrite methods as necessary to calculate costs and validate their load

Manifest class: Manifest represents a list of trucks and the items each truck contains. Manifests are created by manifestGenerator, which calculates what the store needs to order and an optimal way to fulfil the order. They are also created by manifestParser, which is responsible for parsing and writing manifest csv files. Manifests are passed to the store, and logically, represent an order coming in

ManifestGenerator class: Manifest Generator calculates what the store needs to order to get all items above restock points. It calculates an optimal cost reducing method of loading trucks, and stores this data in a manifest.

manifestParser class: Manifest Parser handles parsing and writing manifest csv files. It takes manifest objects, and outputs them to a csv. It takes manifest csvs and turns them back into manifest objects. Enforces the standardised manifest format discussed in the specification.

itemParser class: Item Parser is called by the store and is used to initialise the store's stock. It takes a properties csv and turns it into a stock object with items with an initial quantity of zero.

SalesLogParser class: Sales Log Parser handles parsing sales logs csvs. It takes a sales csv, and turns it into a stock object with a list of items. The quantities of these items represent how many were sold.

Store class: The Store class is a singleton class. It has a stock object that contains item objects. These item objects contain important details about the item, and the their quantity field represents how many of that item the store has in inventory. The store takes stock objects as input from item parser and sales log parser. It uses the input from item parser to initialise the inventory, and the input from sales parser to increase capital and decrease item quantities. It accepts manifest objects as input from manifest parser, which decrease capital and increase item quantities.

StoreGUI class: Store GUI contains the code that implements the above backend classes and turns them into human readable and usable frontend GUI. It acts as a middleman between store, and the parser/generator classes. It calls the generators and parsers when a corresponding button is pressed, and passes the relevant data back to the store object. Once the store processes the input, the GUI outputs the changes to screen. The GUI displays the store capital as a label and store inventory as a table.

Exception classes:
Four exception classes outside of the java library have been defined and used by the application:

CSVFormatExemption: Called when there is an issue with CSVformatting, such as unparsable data or data that does not conform to expected standards

Stock Exception: Called when there is an issue with a stock object

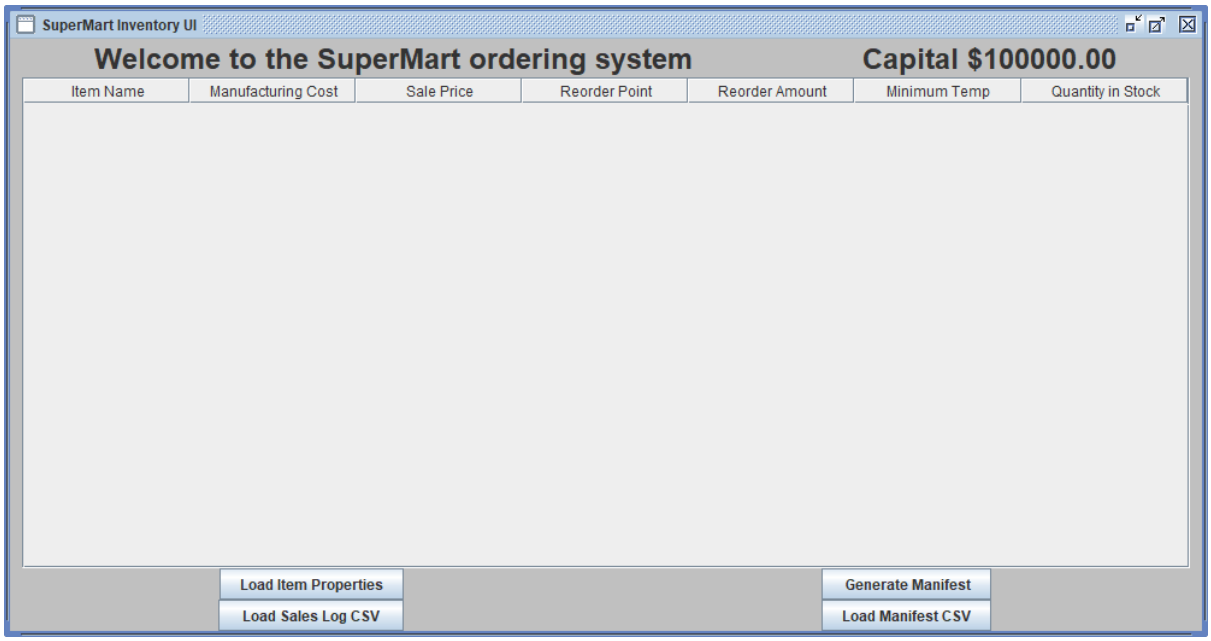
Delivery exception: Called when deliveries can not be accepted by the store, such as when there the delivery would bankrupt the store, or when the delivery contains items the store doesn't stock

Truck Exception: When there are issues with a truck object, such as loading too many items.

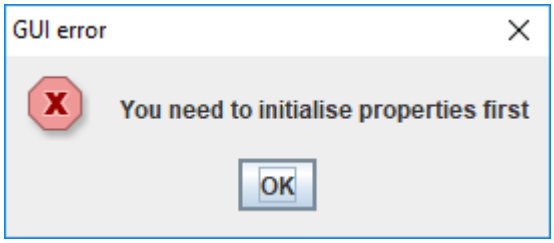
GUI Testing:

Expected Behaviour:

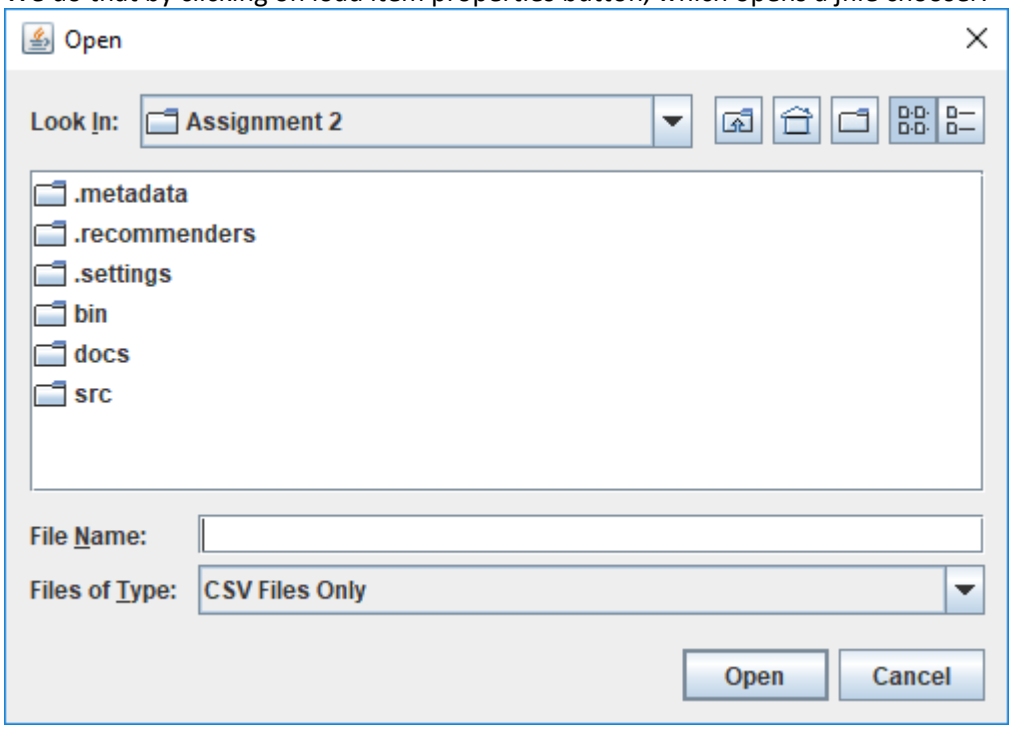
Gui on launch:



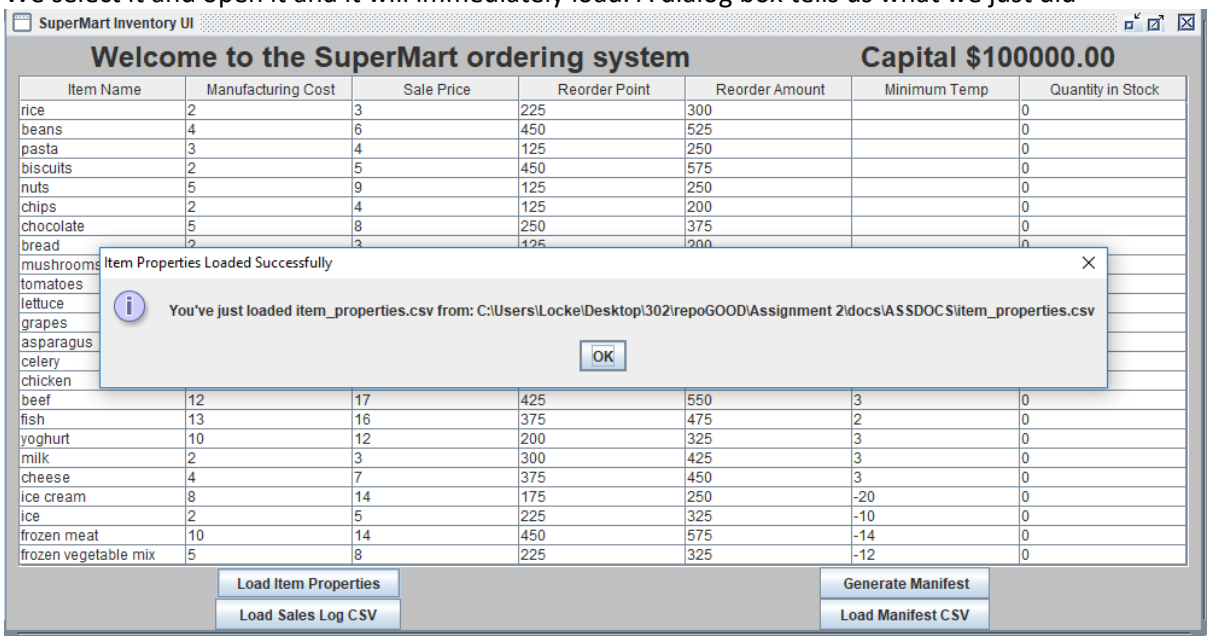
Clicking any button that isn't load item properties will tell the user they must load item properties first:



We do that by clicking on load item properties button, which opens a jfile chooser:

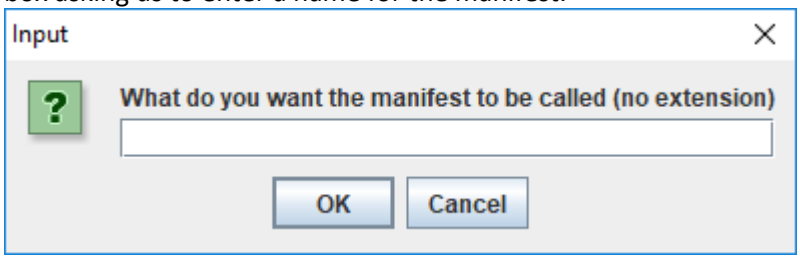


From here we navigate to our item properties. For us, it is in docs/ASSDOCS/item_properties.csv. We select it and open it and it will immediately load. A dialog box tells us what we just did

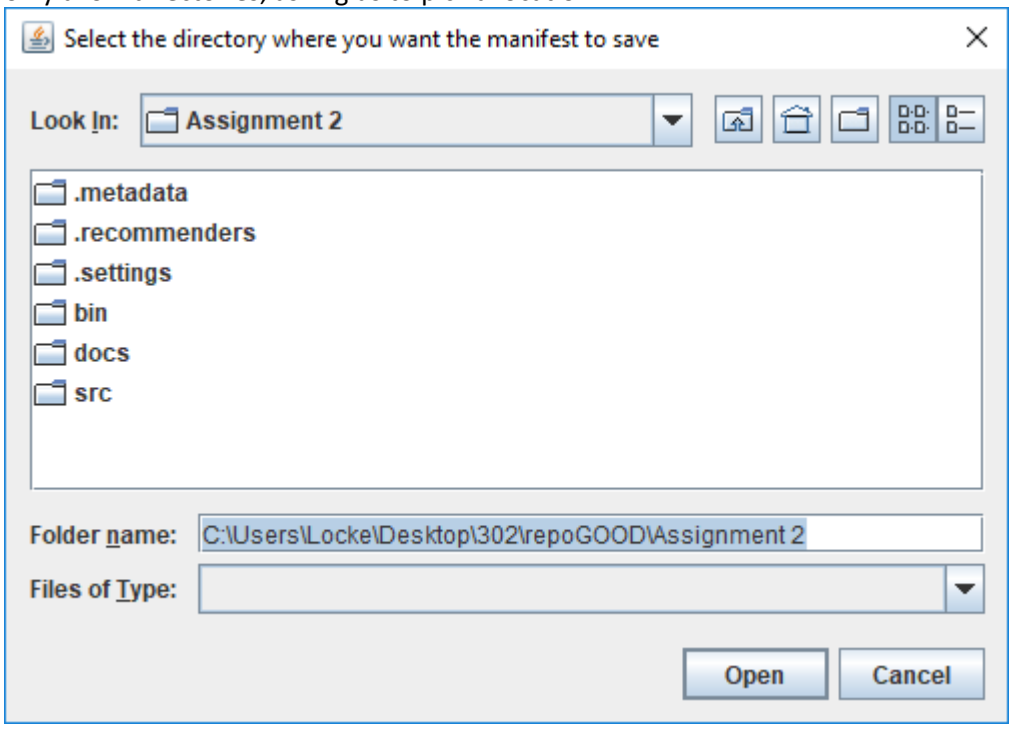


Item Name	Manufacturing Cost	Sale Price	Reorder Point	Reorder Amount	Minimum Temp	Quantity in Stock
rice	2	3	225	300		0
beans	4	6	450	525		0
pasta	3	4	125	250		0
biscuits	2	5	450	575		0
nuts	5	9	125	250		0
chips	2	4	125	200		0
chocolate	5	8	250	375		0
bread	2	3	125	200		0
mushrooms						
tomatoes						
lettuce						
grapes						
asparagus						
celery						
chicken						
beef	12	17	425	550	3	0
fish	13	16	375	475	2	0
yoghurt	10	12	200	325	3	0
milk	2	3	300	425	3	0
cheese	4	7	375	450	3	0
ice cream	8	14	175	250	-20	0
ice	2	5	225	325	-10	0
frozen meat	10	14	450	575	-14	0
frozen vegetable mix	5	8	225	325	-12	0

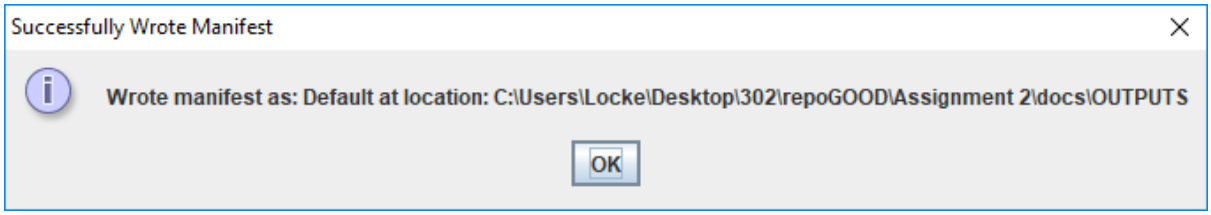
The next step is to generate a manifest. We do this with the generate manifest button. This opens a box asking us to enter a name for the manifest.



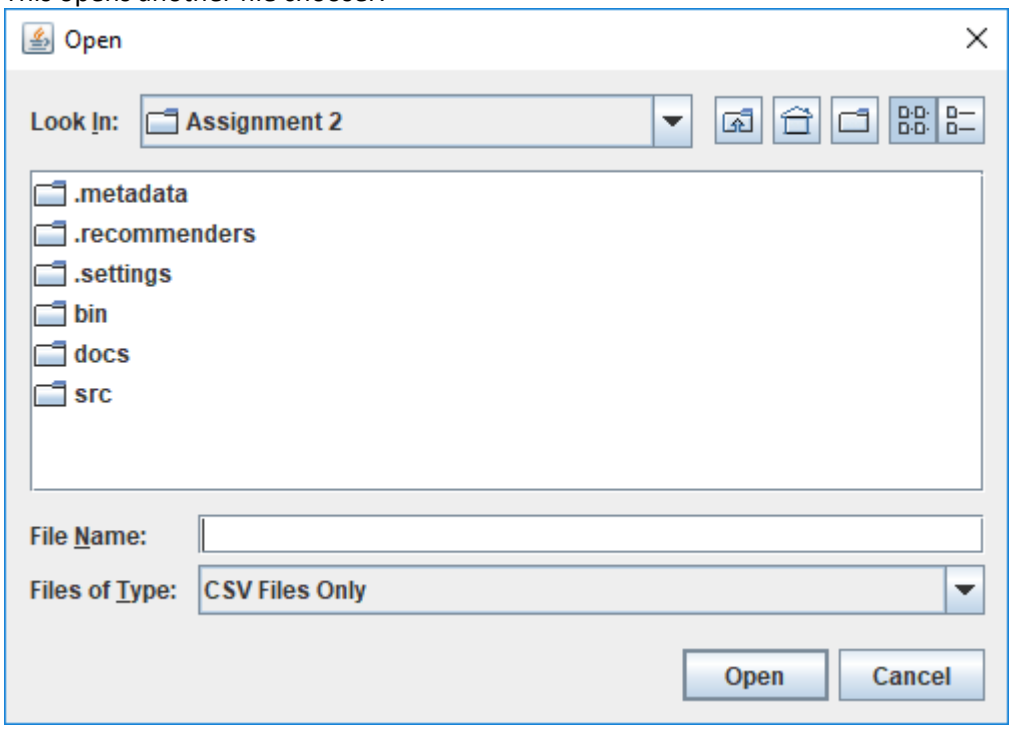
We enter a name for the manifest, and press ok. This immediately opens up a file chooser, set to only allow directories, asking us to pick a location:



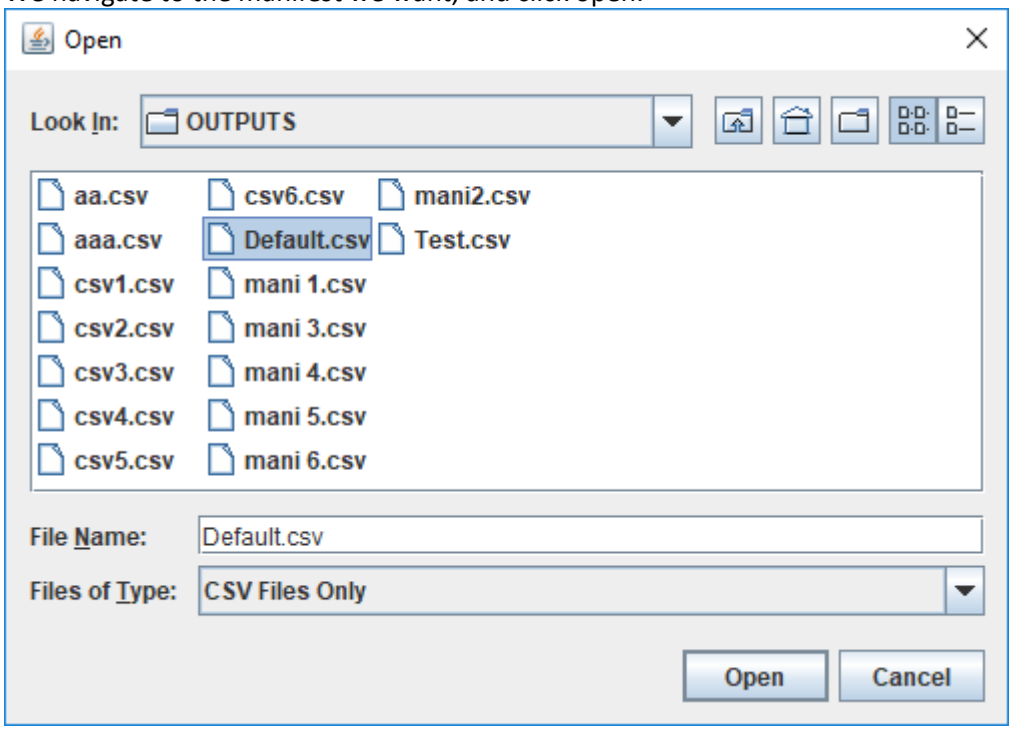
We choose a directory and click open.



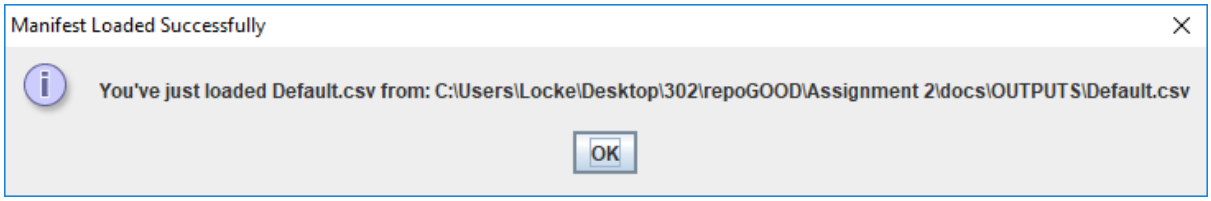
We get a dialog box telling us what we did
The next step is to load the manifest back in. We do this by clicking on the load Manifest CSV button. This opens another file chooser:



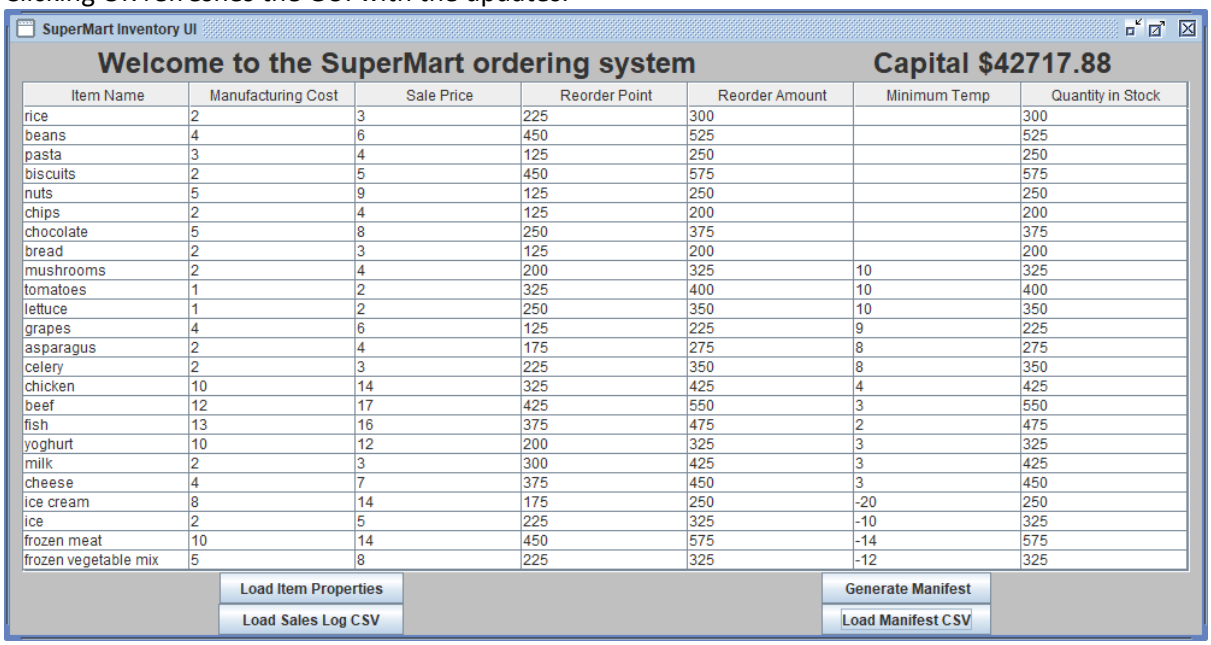
We navigate to the manifest we want, and click open.



Clicking open brings up a dialog box:

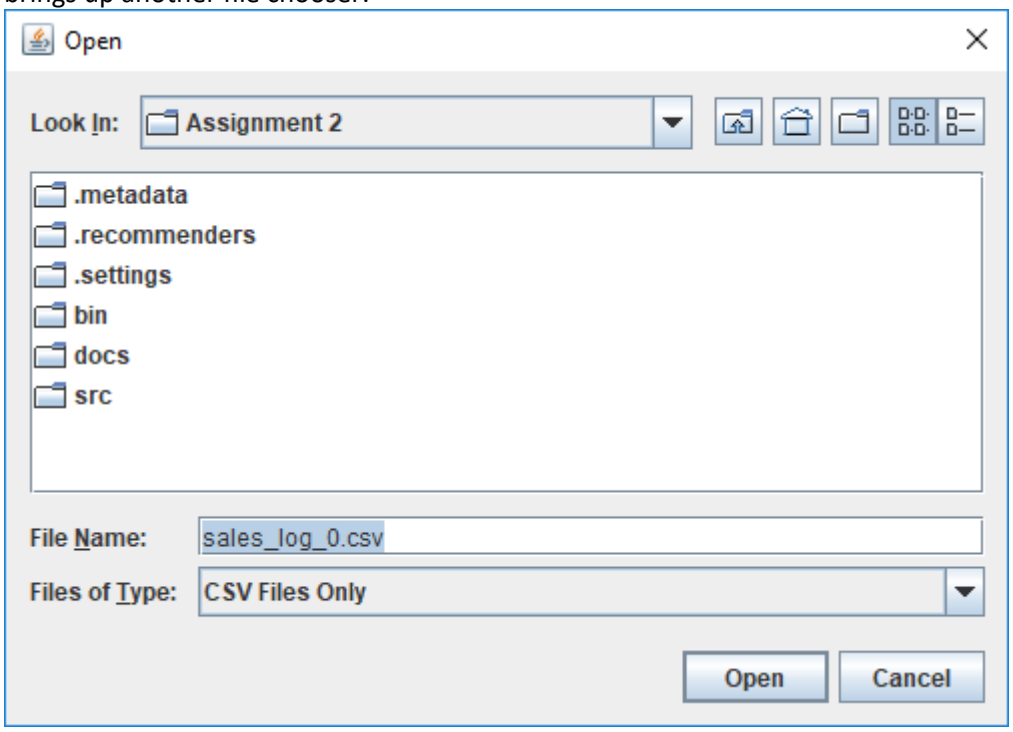


Clicking OK refreshes the GUI with the updates:

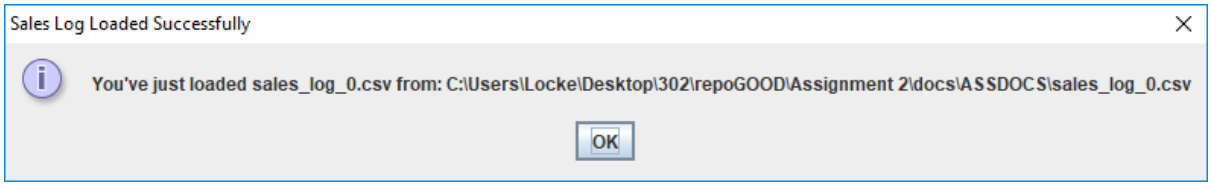


Item Name	Manufacturing Cost	Sale Price	Reorder Point	Reorder Amount	Minimum Temp	Quantity in Stock
rice	2	3	225	300		300
beans	4	6	450	525		525
pasta	3	4	125	250		250
biscuits	2	5	450	575		575
nuts	5	9	125	250		250
chips	2	4	125	200		200
chocolate	5	8	250	375		375
bread	2	3	125	200		200
mushrooms	2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

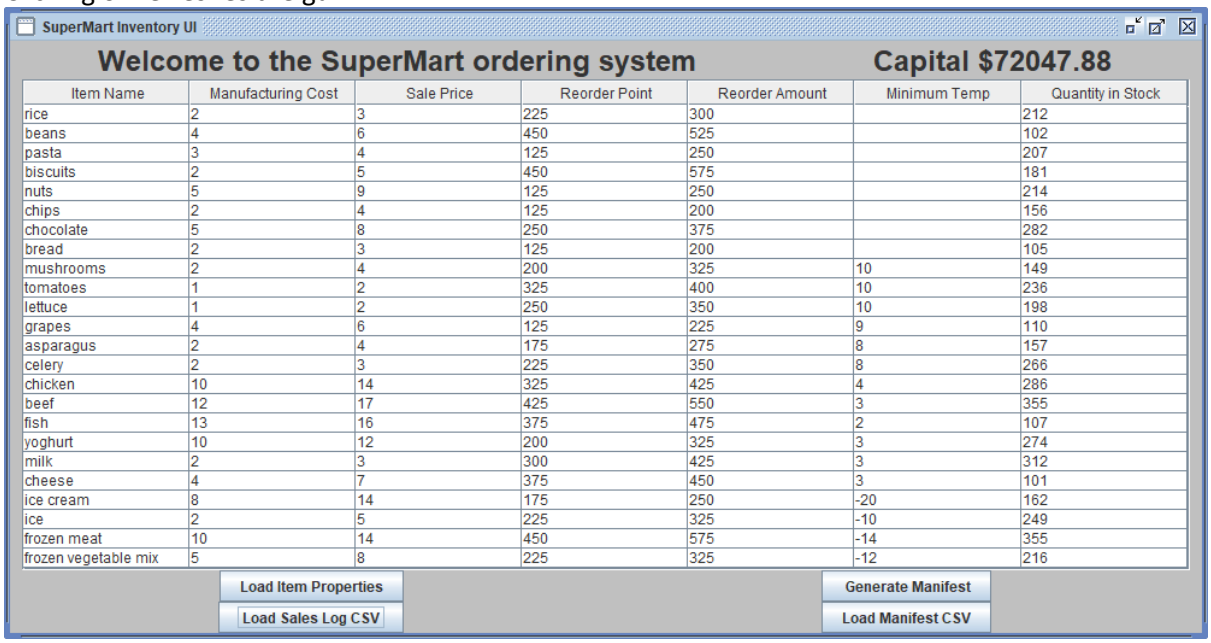
Note that all quantities match reorder amounts and the capital matches the expected result from the specification. From here, we load in some sales by clicking on the Load Sales Log CSV button. This brings up another file chooser:



Opening sales_log_0.csv (the included one from the specification), we get another message telling us what we did

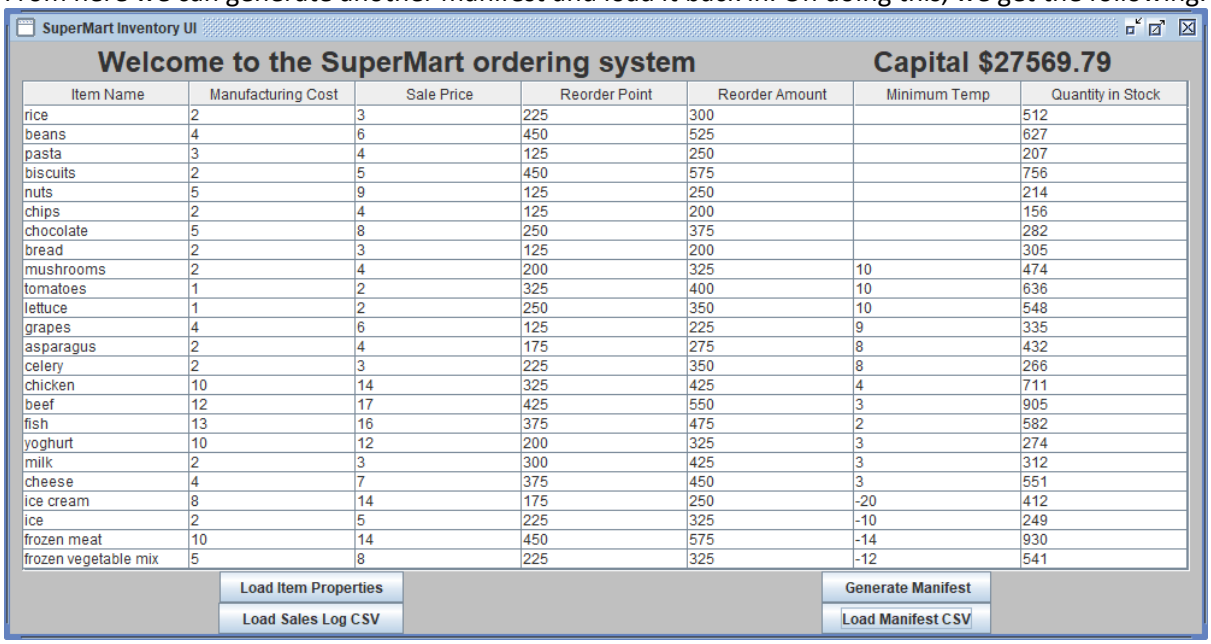


Clicking ok refreshes the gui:



Item Name	Manufacturing Cost	Sale Price	Reorder Point	Reorder Amount	Minimum Temp	Quantity in Stock
rice	2	3	225	300		212
beans	4	6	450	525		102
pasta	3	4	125	250		207
biscuits	2	5	450	575		181
nuts	5	9	125	250		214
chips	2	4	125	200		156
chocolate	5	8	250	375		282
bread	2	3	125	200		105
mushrooms	2	4	200	325	10	149
tomatoes	1	2	325	400	10	236
lettuce	1	2	250	350	10	198
grapes	4	6	125	225	9	110
asparagus	2	4	175	275	8	157
celery	2	3	225	350	8	266
chicken	10	14	325	425	4	286
beef	12	17	425	550	3	355
fish	13	16	375	475	2	107
yoghurt	10	12	200	325	3	274
milk	2	3	300	425	3	312
cheese	4	7	375	450	3	101
ice cream	8	14	175	250	-20	162
ice	2	5	225	325	-10	249
frozen meat	10	14	450	575	-14	355
frozen vegetable mix	5	8	225	325	-12	216

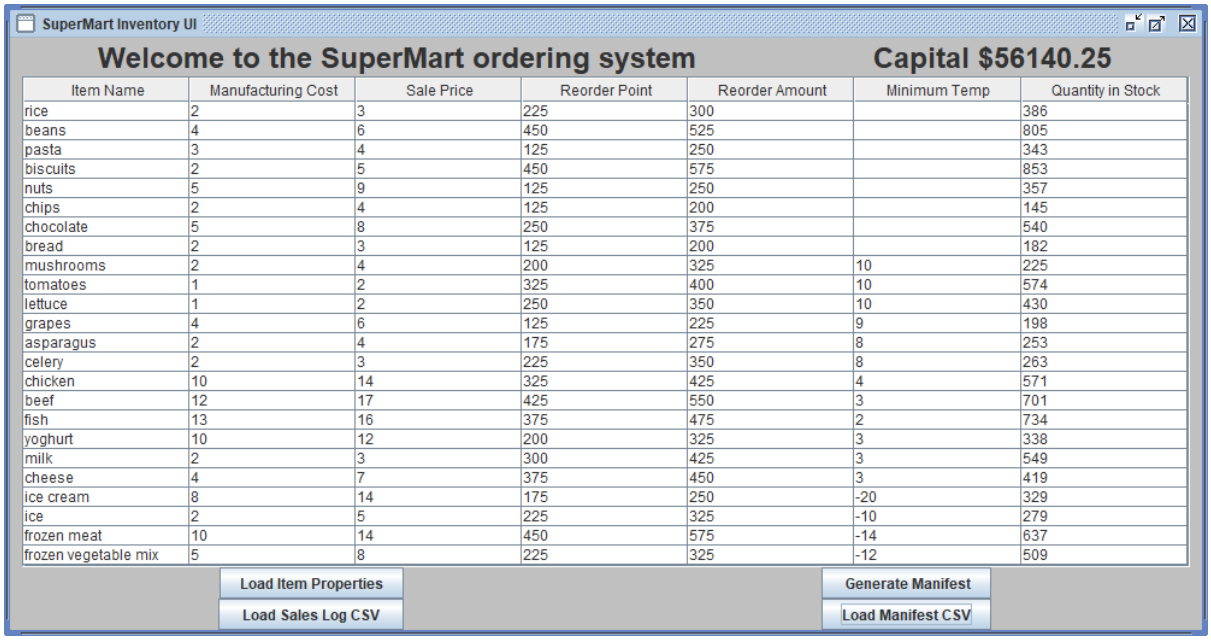
From here we can generate another manifest and load it back in. On doing this, we get the following:



Item Name	Manufacturing Cost	Sale Price	Reorder Point	Reorder Amount	Minimum Temp	Quantity in Stock
rice	2	3	225	300		512
beans	4	6	450	525		627
pasta	3	4	125	250		207
biscuits	2	5	450	575		756
nuts	5	9	125	250		214
chips	2	4	125	200		156
chocolate	5	8	250	375		282
bread	2	3	125	200		305
mushrooms	2	4	200	325	10	474
tomatoes	1	2	325	400	10	636
lettuce	1	2	250	350	10	548
grapes	4	6	125	225	9	335
asparagus	2	4	175	275	8	432
celery	2	3	225	350	8	266
chicken	10	14	325	425	4	711
beef	12	17	425	550	3	905
fish	13	16	375	475	2	582
yoghurt	10	12	200	325	3	274
milk	2	3	300	425	3	312
cheese	4	7	375	450	3	551
ice cream	8	14	175	250	-20	412
ice	2	5	225	325	-10	249
frozen meat	10	14	450	575	-14	930
frozen vegetable mix	5	8	225	325	-12	541

Note that capital matches the expected value after sales log one from the specification.

After loading in all the sales logs and generating manifests and loading them all back in, we get the final values:



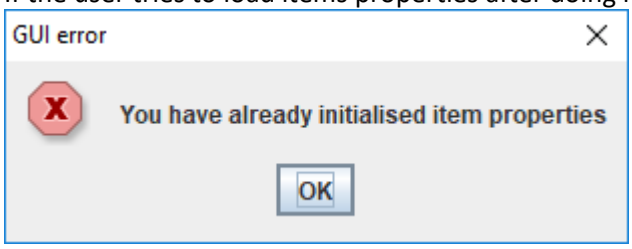
Item Name	Manufacturing Cost	Sale Price	Reorder Point	Reorder Amount	Minimum Temp	Quantity in Stock
rice	2	3	225	300		386
beans	4	6	450	525		805
pasta	3	4	125	250		343
biscuits	2	5	450	575		853
nuts	5	9	125	250		357
chips	2	4	125	200		145
chocolate	5	8	250	375		540
bread	2	3	125	200		182
mushrooms	2	4	200	325	10	225
tomatoes	1	2	325	400	10	574
lettuce	1	2	250	350	10	430
grapes	4	6	125	225	9	198
asparagus	2	4	175	275	8	253
celery	2	3	225	350	8	263
chicken	10	14	325	425	4	571
beef	12	17	425	550	3	701
fish	13	16	375	475	2	734
yoghurt	10	12	200	325	3	338
milk	2	3	300	425	3	549
cheese	4	7	375	450	3	419
ice cream	8	14	175	250	-20	329
ice	2	5	225	325	-10	279
frozen meat	10	14	450	575	-14	637
frozen vegetable mix	5	8	225	325	-12	509

Note that Capital matches the expected capital after sales log 4, and that all the item quantities match their expected outcomes. When behaving as expected, with the expected input, the GUI gets the expected results

Exceptional Behaviour:

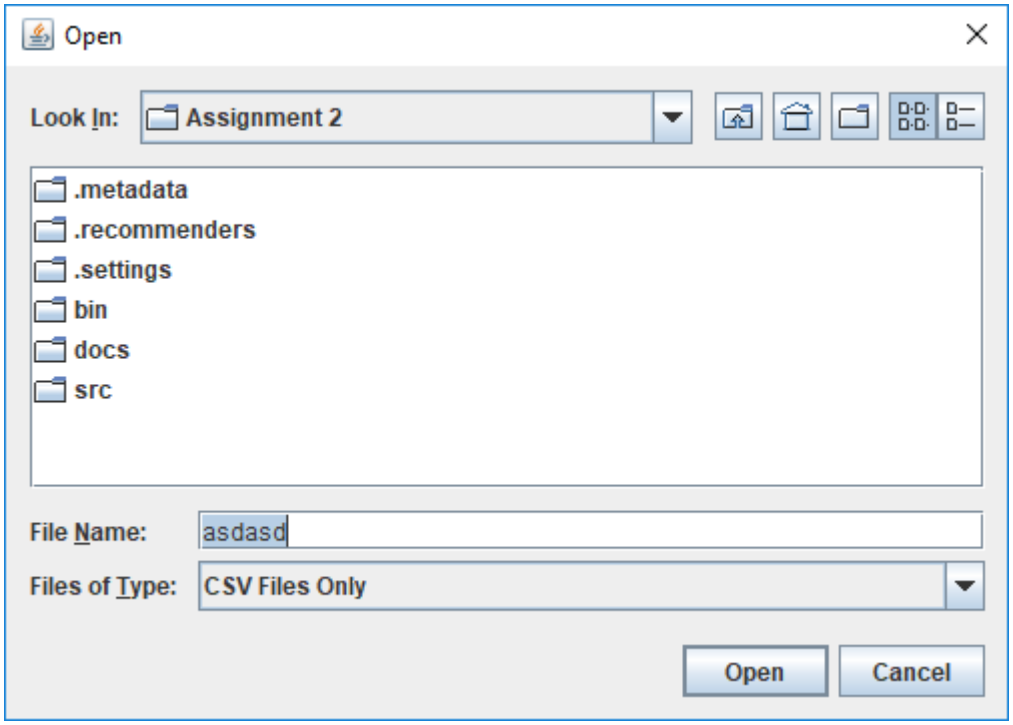
Loading Item Properties:

If the user tries to load items properties after doing it successfully earlier”

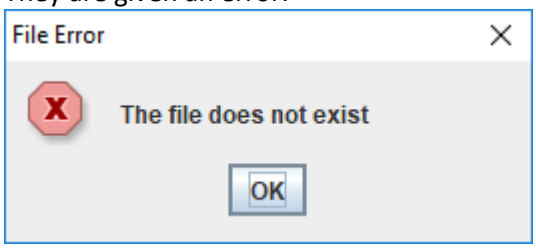


They get an error

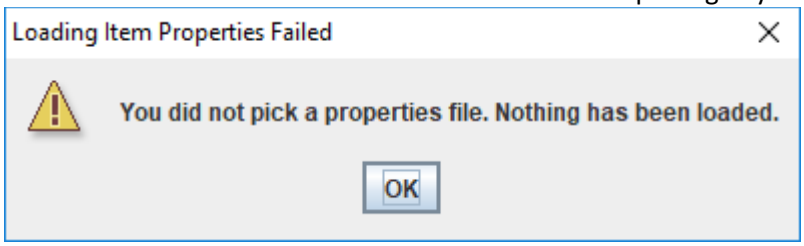
If the user selects a file that does not exist:



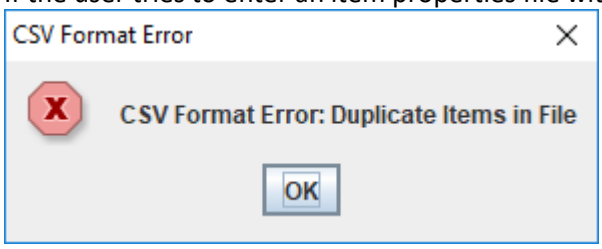
They are given an error:



If the user closes out of the selection screen without picking anything, they are given a warning.:



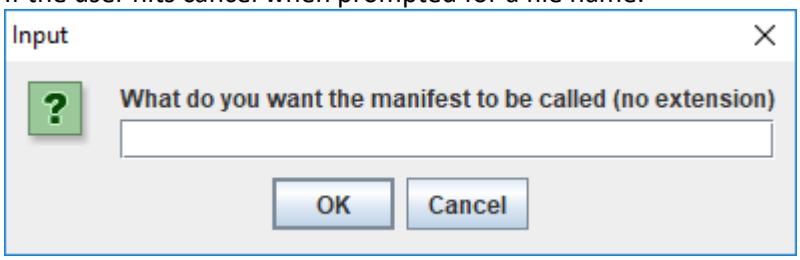
If the user tries to enter an item properties file with duplicates



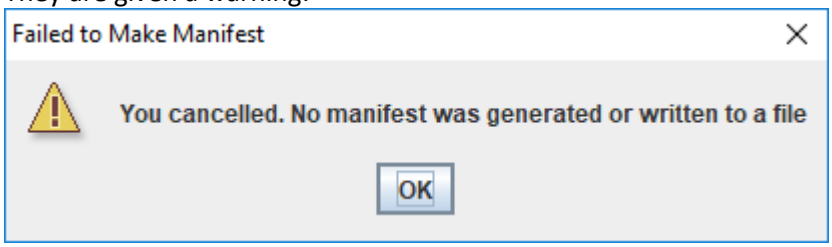
They are given an error

Generating Manifest:

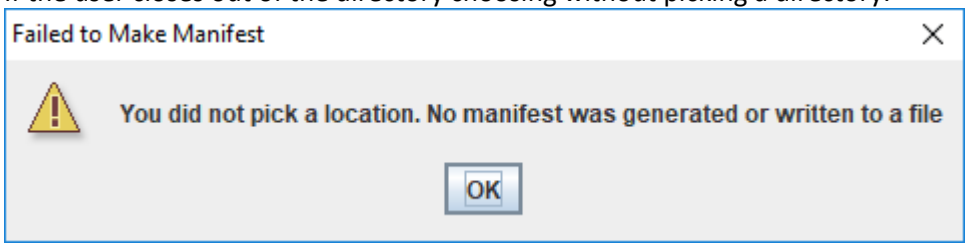
If the user hits cancel when prompted for a file name:



They are given a warning:

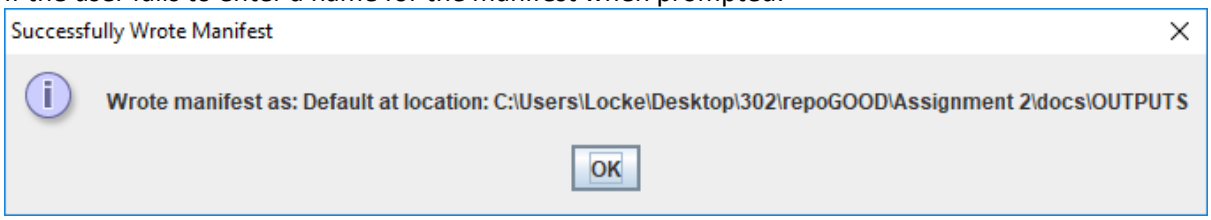


If the user closes out of the directory choosing without picking a directory:



They are given a warning

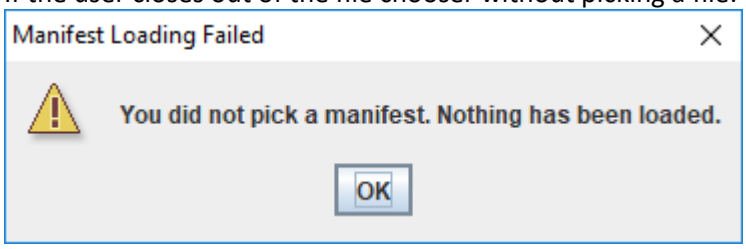
If the user fails to enter a name for the manifest when prompted:



It gives the manifest the name “default”

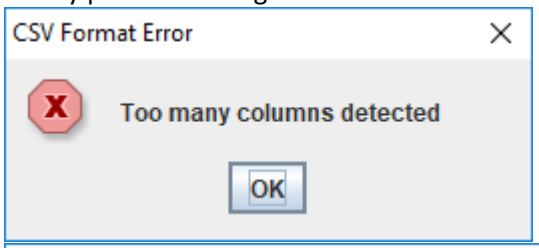
Loading a Manifest:

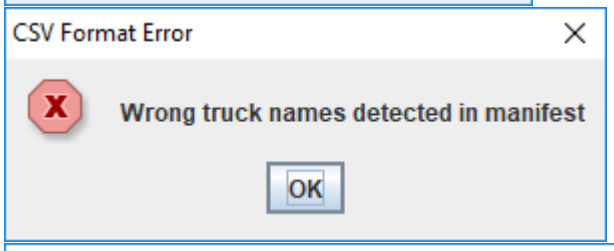
If the user closes out of the file chooser without picking a file:

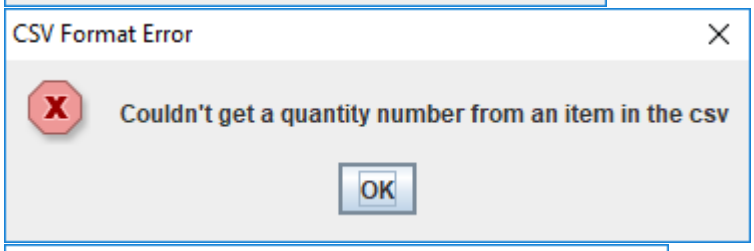


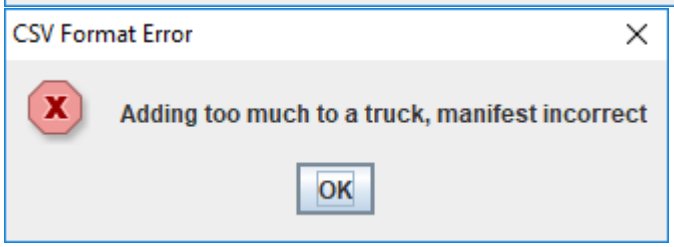
They get a warning

If they pick something that isn't a correct manifest:



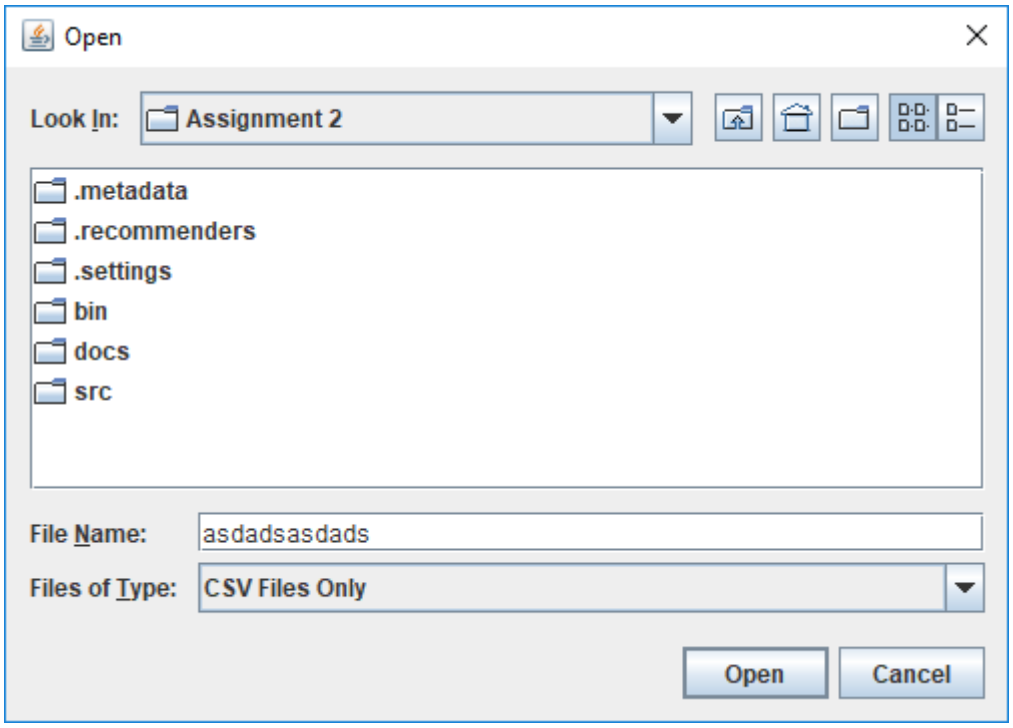




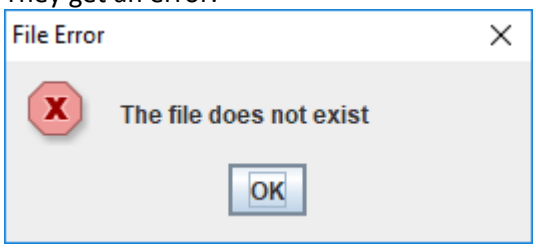


They get an error and get told what was wrong

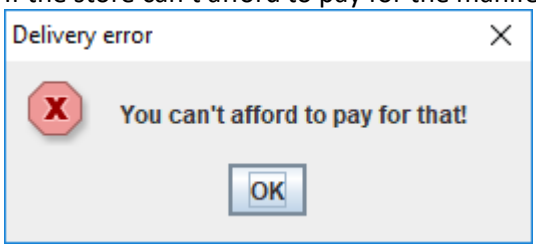
If the user tries to load a file that doesn't exist:



They get an error:

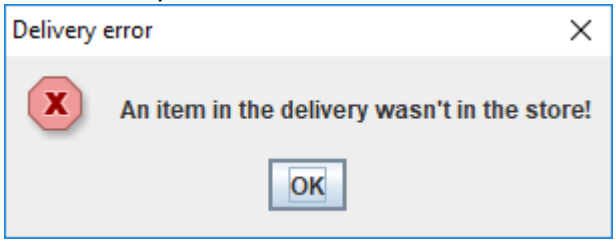


If the store can't afford to pay for the manifest:



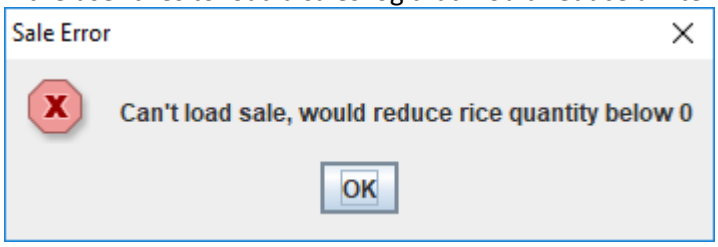
They get an error

If the delivery has items the store doesn't sell:



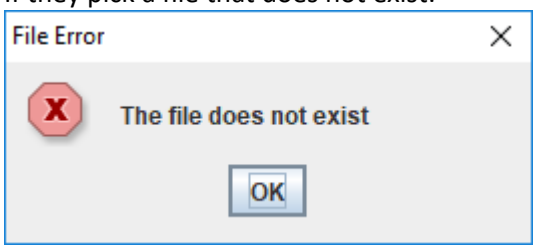
Sales Log Loading

If the user tries to load a sales log that would reduce an item quantity below 0:



They get an error

If they pick a file that does not exist:



They get an error

If they try to pick files that are invalid sales logs, they get the following errors: