CAB302 Assignment 2

Inventory Management Application

Student Name: Luke Reynolds, Johnathan Gonzalez

Student Number: n18023481, n9821112

Date Submitted: 27-05-2018

# Technical Description

# 1.1 Program Architecture

The program architecture was designed using a test-driven development approach. In general, tests were designed with a high-level scope and then refined to meet specific goals. UML and class diagrams have been implemented to illustrate a visual representation of the relationships among the packages, classes and objects to ultimately give the reader a bird’s eye view of the program architecture.

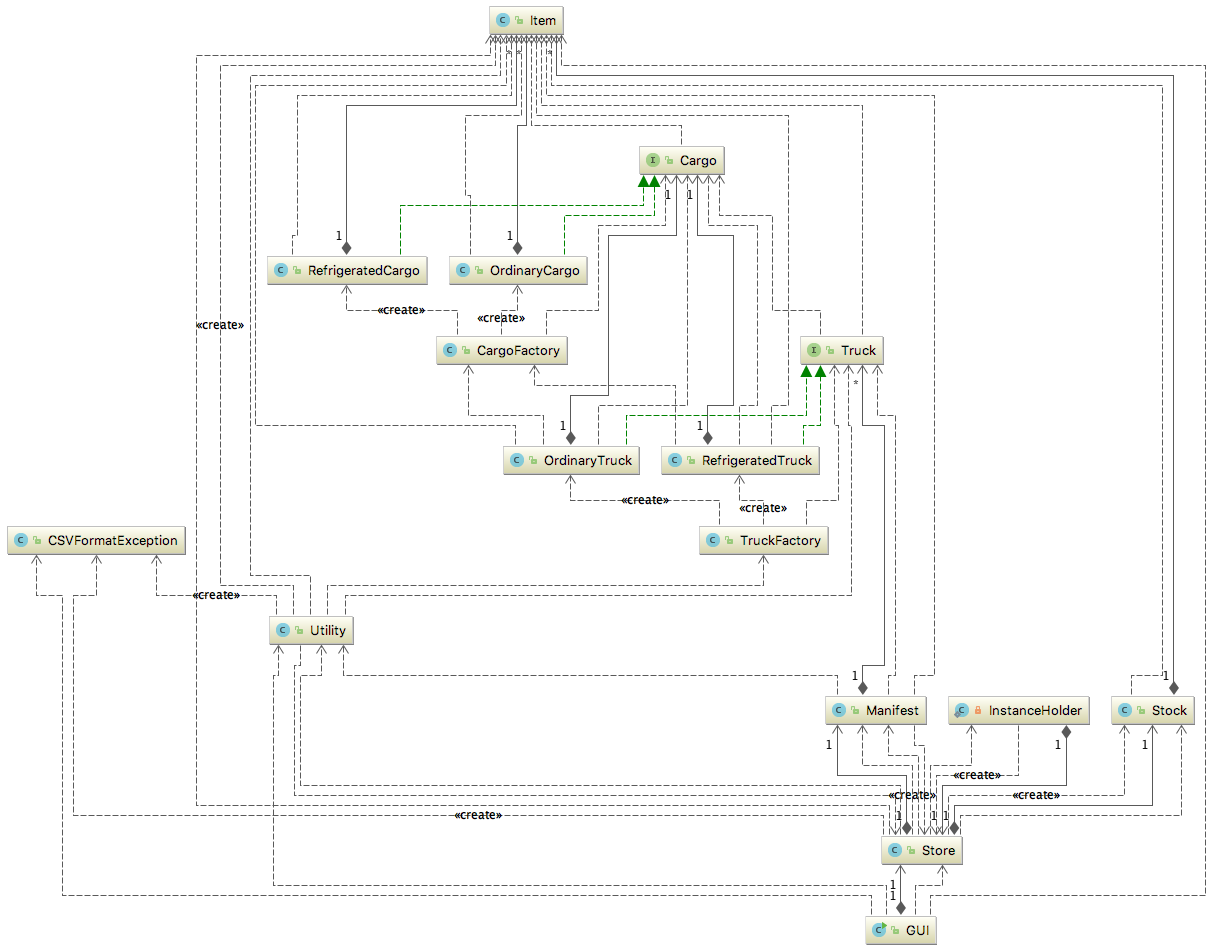


Figure - Program Architecture

# 1.2 Object Oriented Design

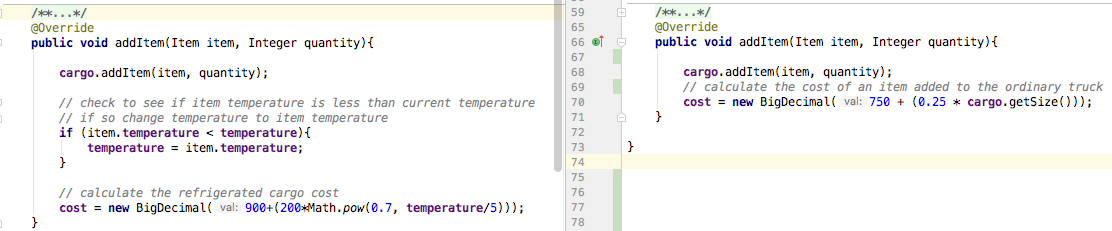
Throughout the program we applied many of the principles associated with object oriented programming such as abstraction, polymorphism, encapsulation, class inheritance and class composition. Combining these techniques allowed us to create flexible relationships between objects and classes.

Figure - Example of polymorphism

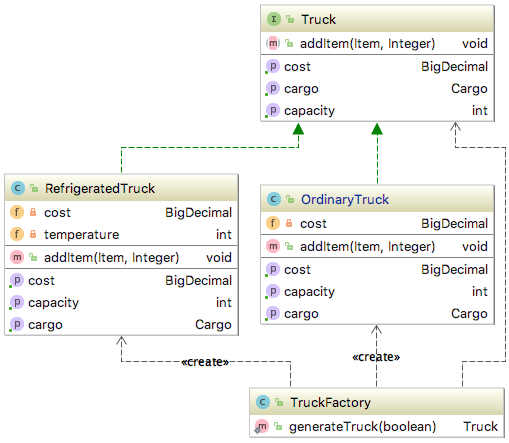
In this example of polymorphism we can see how two truck subclasses implement the addItem method differently to calculate a new cost value by overriding the default addItem method.

Figure - Example of Factory Pattern

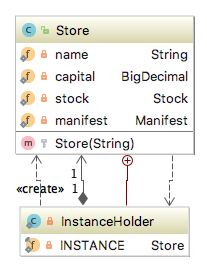
Class inheritance was also utilized to achieve reusability and continuity among similar class types. In the example to the right you can see how the Refrigerated Truck class and the Ordinary Truck class inherit the methods and properties of the Truck interface.

This also illustrates an instance where we implemented the factory pattern to create truck objects specific to the circumstances.

# 1.3 Design Patterns

***Factory Pattern***

We used two design patterns in our programs implementation; the factory pattern and the singleton. As mentioned previously we used the factory pattern to create truck objects based on an input. We used this pattern again for the cargo objects as we wanted to enforce a flexible and decoupled design.

***Singleton Pattern***

Apart from the factory pattern we used the singleton pattern for implementing the store object. This gave us an opportunity to expose composition instead of inheritance and to easily expose a single immutable state to the GUI.

# 3. Graphics User Interface (GUI) Test Report

Figure - Example of Singleton

# 3.1 Home Page

Figure - GUI outline

The main screen of the user interface is laid into four main sections.

1. Current capital display
2. View items list button
3. Item list table
4. Load files buttons

# Item List

Figure - Load Items List example 1

The load buttons section holds four buttons used to load the necessary files into the program.

1. Load sales log
2. Load Manifest
3. Export Manifest
4. Load Item List

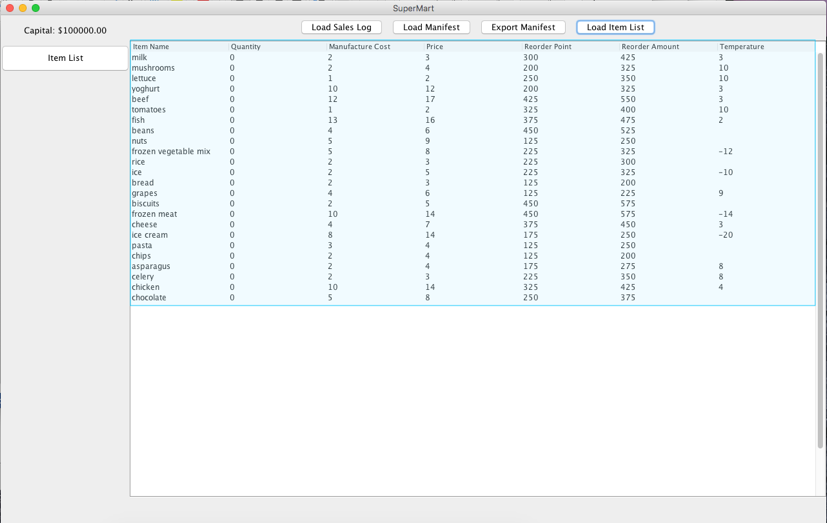
In this example, we illustrate how to load the items list into the program. In the first frame, after clicking the **Load Item List** button a window prompt appears that allows you to select the appropriate file for loading.

Figure – Items List View

After opening the item\_properties.csv file item list table is populated with the item list in a tabular format.

If you fail to load an appropriate item list file an exception will be thrown and the user will be shown this error message modal.

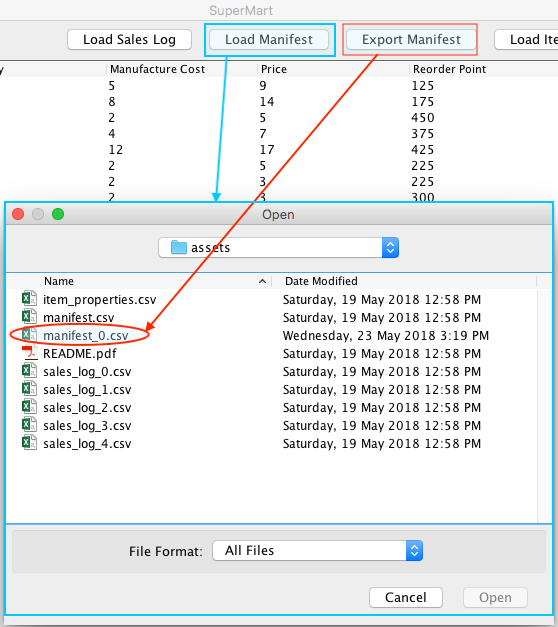
3.3 Generate & Load Cargo Manifest

Figure - Generate a Manifest

Generating a new manifest

To generate a new manifest the user would first click on the **Export Manifest** button. A manifest is then generated in the assets folder which can be loaded into the system.

Loading a new manifest

To load the new manifest the user should then click on the **Load Manifest** button. This will launch the file browser. In this example, the generated manifest is *manifest\_0.csv* which can then be opened and loaded.

Change in available capital

After the manifest has been loaded into the system the available **Capital** will be reduced to reflect the purchase and transportation of the goods. You will also notice that the quantity column has been updated to reflect the new stock levels.

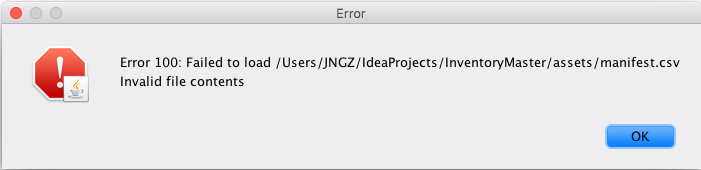
Manifest exception handling

Figure - Exception Handling example

Once again if the user fails to load the appropriate file type the system will throw an exception and display an error modal.

3.4 Load Sales Log

Loading a new sales log

To load a sales log, the user would click on the **Load Sales Log** button. This action would present the file browser, where the user would select the appropriate sales log file. Unlike the export manifest function, sales logs are not generated by the system.

As sales logs are loaded into the system the available capital value is increased to reflect the sales made. You will also notice that the quantity of items available in the item list will decrease with respect to the sales made in sales log.

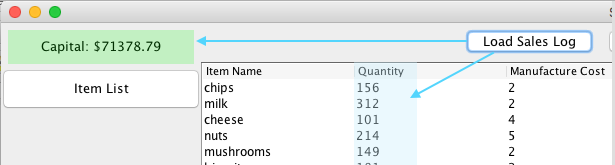
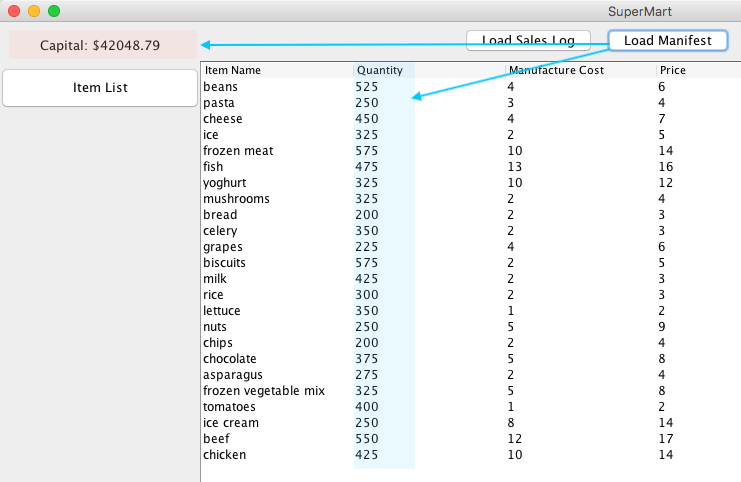
In this example, the available capital is increased to $71,378.79.

Figure 12 - Capital Decrease

Figure - Capital Increase

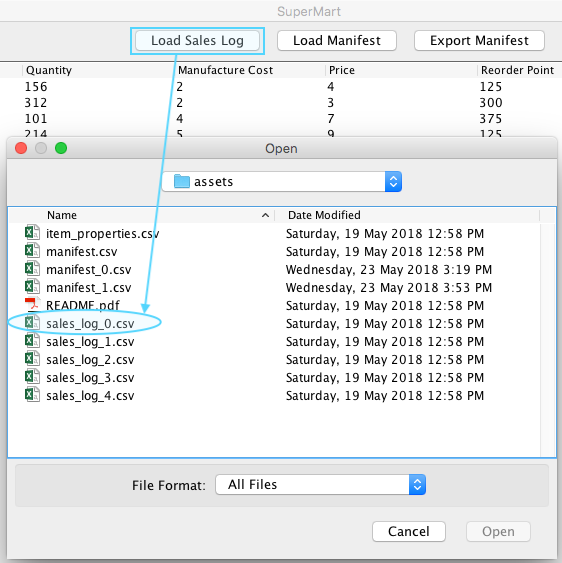


Figure - Load Sales Log

Sales log exception handling

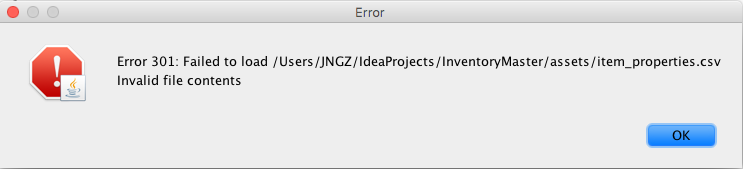
As with previous examples, when a user tries to load a file that is not a sales log an exception is thrown and an error modal is displayed. This exception is passed up through the hierarchy of classes and displayed on the GUI.

Figure 13 - Exception Handling example 2

# References

* <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html#simple>
* <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
* <https://stackoverflow.com/questions/20473325/gridlayout-java-center-alignment>