CAB302 Assignment 2:

Inventory Management Application Report

Names: John Layson, Nathan Thai

Student Numbers: n9823239, n9823361

# Program Architecture:

## Diagram:

**Polymorphism**

In Java’s Object-Oriented Programming, objects are written in separate classes therefore the main program needs a way to refer to those classes, polymorphism is commonly used in that area. When a parent/superclass is used to refer to a child/class object polymorphism is achieved. All of superclasses and subclasses use polymorphism as there are only one way to access a referenced declared variable. Once an item is declared the reference is created then the users can use the methods within that reference/item.

**Classes & Interaction**

*Interface Class*

Item Interface determines each item/object’s characteristics and behaviour, similar to creating a blueprint in terms of building architecture. It is the layout that the object takes. In this case, Item Interface gives Dry Item and Refrigerated Item classes name, cost, price, reorder point and reorder amount. This is because both Item classes must have those characteristics for the Store to function appropriately. Interfaces can contain constants which we are not currently using as each item have different characteristic values. Interfaces cannot be instantiated but they are implemented into other classes.

*Abstract Classes & Abstraction*

This type of class tends to be more general/less specific than concrete classes since it is further up on the hierarchy, hence that interfaces cannot be instantiated. *Abstraction* is implemented for hiding the implementation from the user, it can be seen what the object does but not how it does it.

Item Temp can give its subclasses the behaviour to have a temperature, while Trucks can also be inherited for the subclasses to have a temperature it includes a quantity to determine a refrigerated trucks quantity with that temperature.

The temperature in the Item Temp is used accordingly with the refrigerated truck. Since each Refrigerated Item has a required temperature, the refrigerated truck dispatched will be based on that temperature so that all the items within the cargo does not spoil.

*Concrete Classes*

Concrete classes are at the end of the hierarchy as they contain the constructors that initializes that specific item.

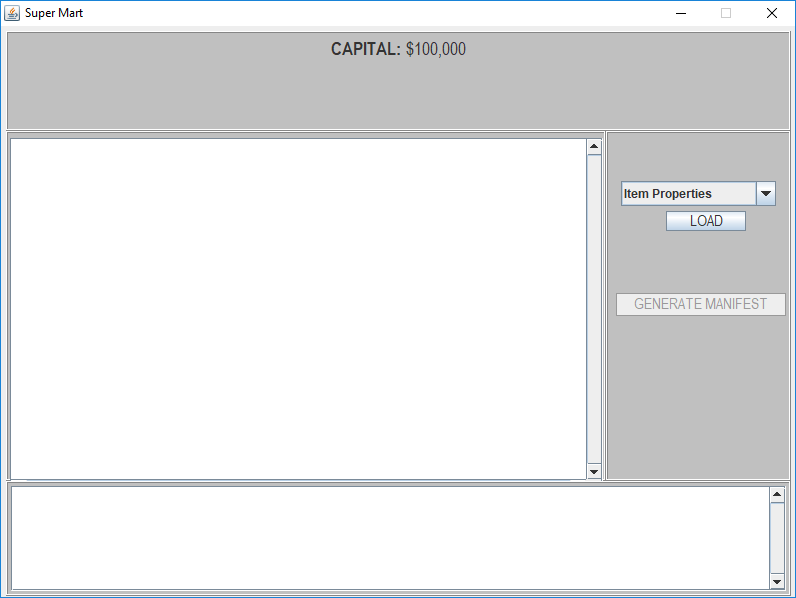
Our concrete classes:

* Dry Item
* Refrigerated Item
* Ordinary Truck
* Refrigerated Truck
* Stock
* Store
* Manifest

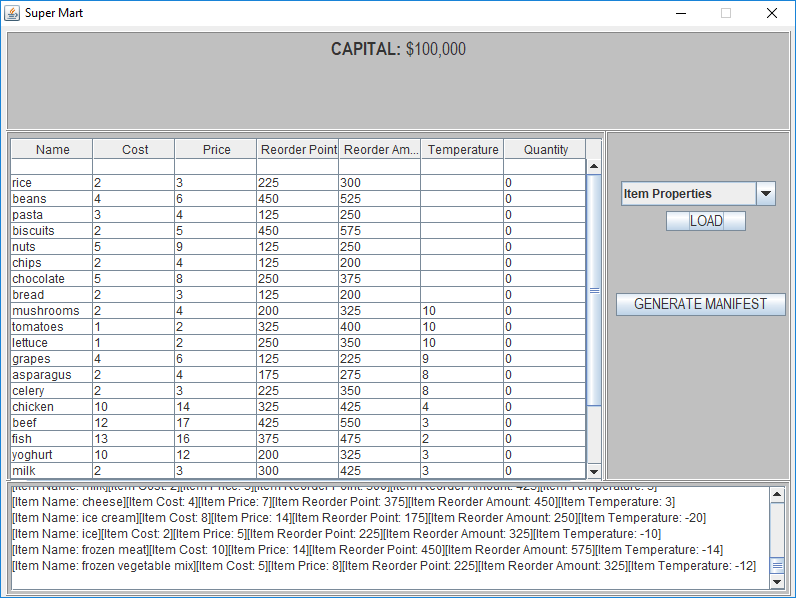
These classes have a constructor that creates an object that has the attributes passed within the parameter. They all interact with each other using the setters and getters within the classes. For example, an item “rice” in Stock have a quantity of “10” in Store and if we had to check the reorder point to restock, then we would look at the item “rice” individually by accessing it using the getter in Dry Item class, likewise for the refrigerated item. If “rice” is due for reordering, then a Manifest is created which determines the items needed to be reordered and which Trucks are needed to be dispatched. When the items are bought from the distribution centre, the Store then restocks those items and updates the Store’s capital and Stock accordingly. Each class is needed for the entire program to work hence that referencing/polymorphism plays a big role in the project as the objects in an OOP based program it is the only way to access an object’s attributes.

# GUI Test Report:

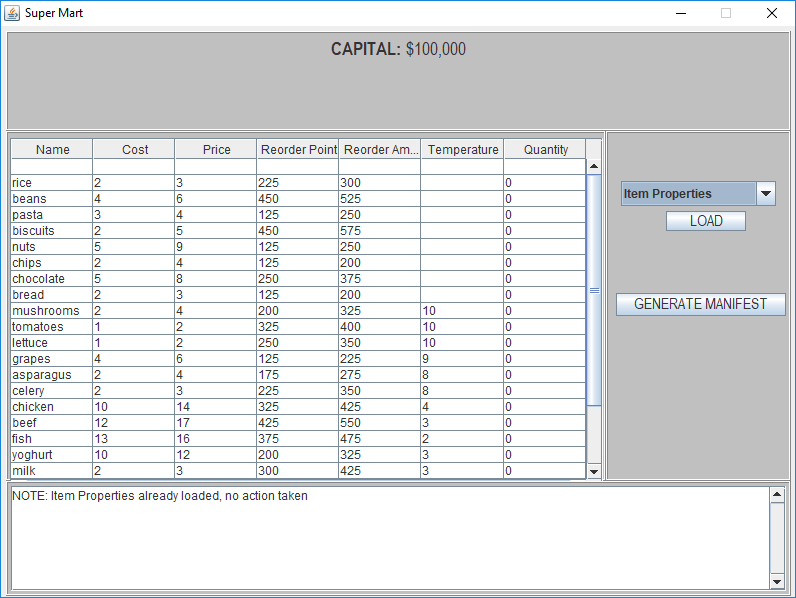
SCREEN 1 - Initial Page – GENERATE MANIFEST button is disabled before items are initiated



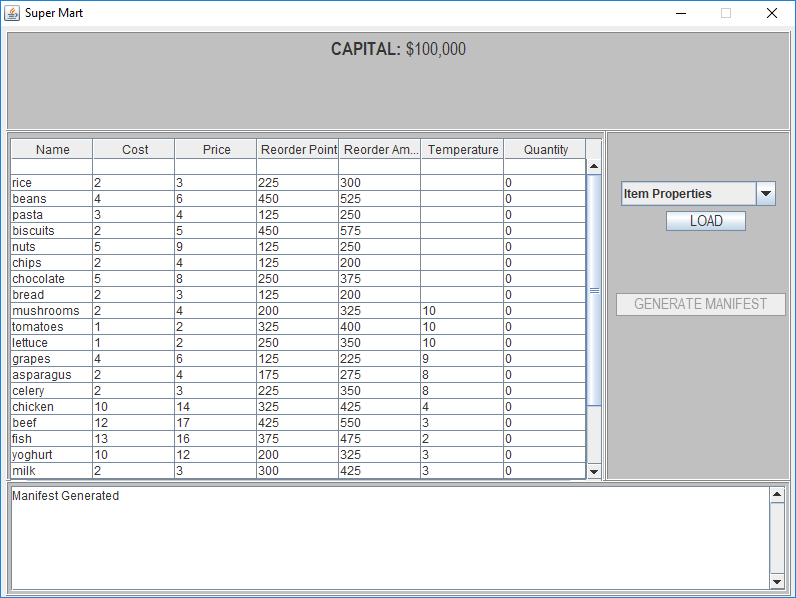
SCREEN 2 – After loading item properties – GENERATE MANIFEST enabled, inventory table is shown, and a console-like output showing the processed data.



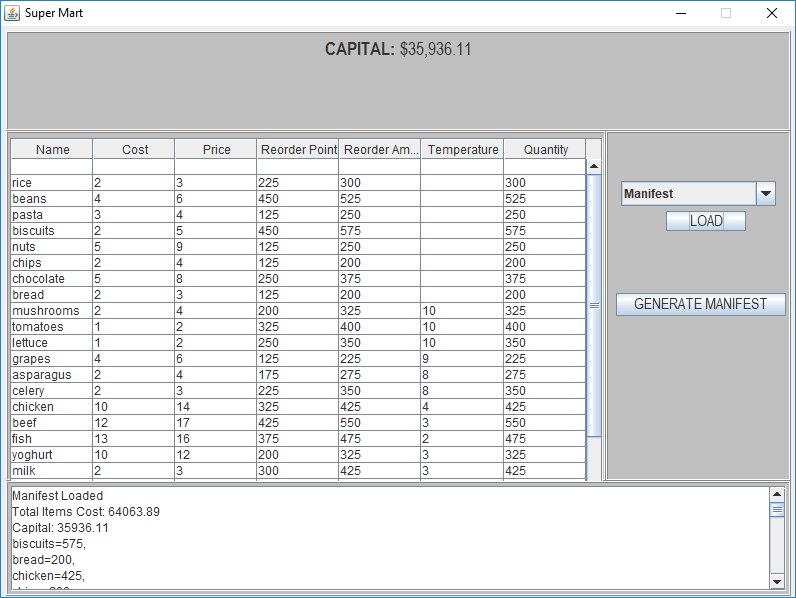
SCREEN 3 – Item properties already loaded error



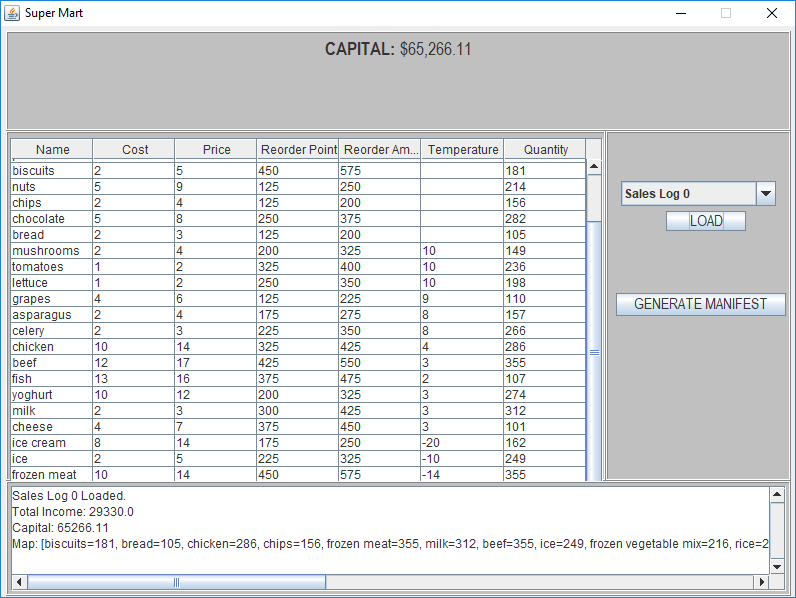
SCREEN 4 – GENERATE MANIFEST button clicked outputs a string in console “Manifest Generated”



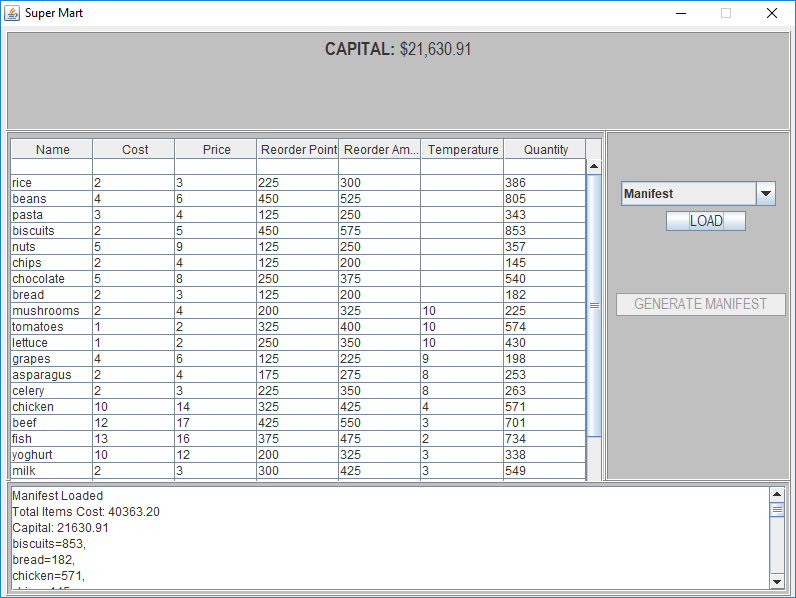
SCREEN 5 – Load Manifest – Updates each item quantity correctly, however the cost taken from capital has inaccuracies. Console outputs of the manifest results.



SCREEN 6 – Load Sales log 0 – shows the income and current capital in the console and also updates the inventory table.



SCREEN 7 – Final Inventory and Capital – After all the whole process of generating manifest, loading manifest and loading sales log. The final stock is correctly updated but the capital still has inaccuracies.



SCREEN 8 – Stock Exception – When loading a manifest and the store’s capital is insufficient, an error in the console appears to inform the client.

