CAB302 Assignment 2: Inventory Management Application

Lucas Wickham, Christopher Ayling

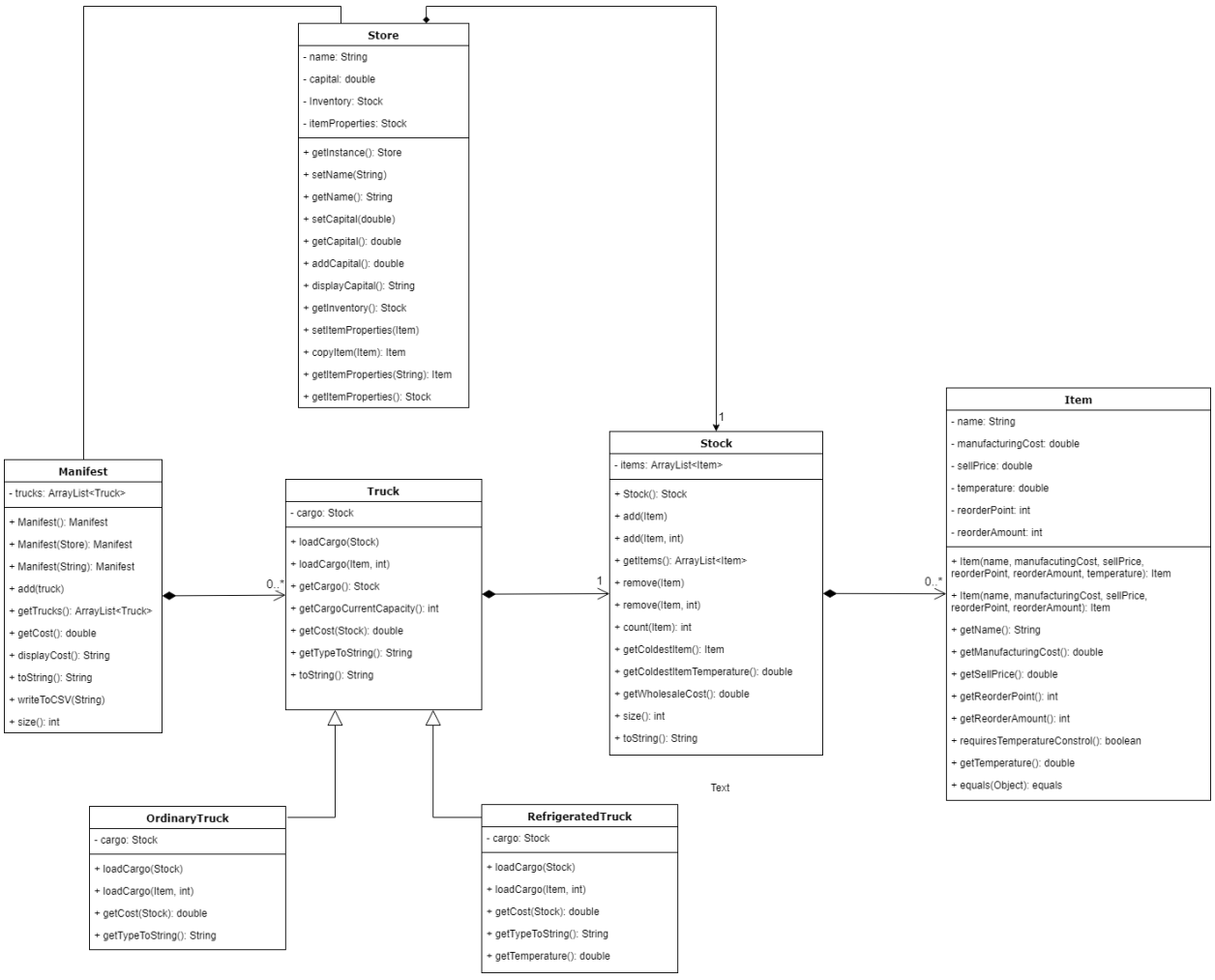
The application detailed in this report is an *inventory management application* developed for client SuperMart. The application is used to automate inventory management. The features of the application are as follows: Managers can upload sales logs and export and load shipping manifests. These manifests are guaranteed to be the best price and are submitted by the manager to the distribution centre manually.
The application is built using the Java programming language which makes use of the Swing library for *user interface* creation.
The process for building the application is as follows: a development team of two was formed and then specifications were reviewed. From these specifications, product mockups were created so both developers understood what they were creating. A remote git repository was then created for source control and development of *unit tests* began. Next the back and front ends were implemented. As the implementation began to take shape, some unit tests had to be added, removed or modified due to initial mistakes or lack of polish and new, better ideas coming to surface.
A log of the repository's history which demonstrates use of version control is attached.

Technical Description

Why Object Orientation?

SuperMart's inventory management application was written using the Object Orientated Programming (OOP) paradigm. This means that the code is structured very differently to classical procedurally written code. OOP is useful for modelling the interaction between objects such as those involved in this use case (store, items, trucks, inventory, manifest etc). OOP allows us to abstract and encapsulate the implementation details of the things these objects can do. OOP code is therefore more extensible and reusable (due to high modularity) and allows for easier debugging.

Architecture & OOP Design Concepts



Class Diagram (UML)

Item

Stock

Store

Truck

To aid in the calculation of cost and aclass was used to represent the trucks used for making shipments to the store. As trucks come in two types, refrigerated and ordinary but always perform the same functionality (holding items and having a price) it made sense to make use of *polymorphism* and therefore *inheritance*. *Method overloading* is used for defining the loadCargo methods to realise DRY principles.

OrdinaryTruck

The class OrdinaryTruck is used for representing trucks which carry items which do not require refrigeration. Ordinary Truck will throw a StockException if an item which required refrigeration is loaded into it.

RefrigeratedTruck

The class OrdinaryTruck is used for representing trucks which can carry items which require refrigeration. Refrigerated Truck will throw an exception if an item is added which requires a temperature outside of the safe range [-20, 10] degrees celsius.

Manifest

The Manifest class is used to represent shipping manifests which can be loaded into and exported from the application. Manifests fulfill two different roles but each requires the same functionality (holding trucks, having a cost) so instead of using *inheritance* or *polymorphism* to represent two different classes of Manifest e.g. ExportedManifest and LoadedManifest *method overloading* was used for the constructor.

The three constructors are detailed below:

- Manifest()
    - Creates an instance of Manifest with no trucks.
- Manifest(Store store)
    - Constructs an instance of Manifest which contains trucks that together contain the reorder amount of any items below their reorder point in the store's inventory. This manifest minimizes the cost of restocking by implementing the manifest optimization algorithm.

- Manifest(String path)
    - Constructs an instance of Manifest which matches the manifest specified by the CSV at the location specified by the path parameter.

The manifest optimization algorithm works by placing cold items together and is detailed below:

```
ALGORITHM optimizeManifest(Stock items):
      new manifest
      coldItems := items.getCold
      ordItems := items.getOrdinary
      coldItems := sortAscending(coldItems)
      truck := new coldTruck
      while coldItems not empty:
            get coldItems[0]
            load item into truck
            remove item from coldItems
            if truck is full:
                  add truck to manifest
                  truck := new coldTruck
      if truck not empty:
            while ordItems not null:
                  get ordItems[0]
                  load item into truck
                  remove item from colditems
                  if truck is full:
                        sdd truck to manifest
      while ordItems not empty:
            get ordItems[0]
            load item into truck
            remove item from coldItems
            if truck is full:
                  add truck to manifest
                  truck := new coldTruck
      return manifest
```

GUI Test Report [2-5 pages]

Capital

Some stuff

Inventory

Some stuff

Load Item Properties

Some stuff

Export Manifests

Some stuff

Load Manifests

Some stuff

Load Sales Log

Some stuff

Exception Handling

Some stuff